Universitat Oberta de Catalunya

uoc.edu

GameChat

Desarrollo de una aplicación Android de chats y juego

Chenghuan Ji
Máster en desarrollo de aplicaciones para dispositivos móviles
Trabajo final de máster

Consultor: Pau Dominkovics Coll
Profesor: Carles Garrigues Olivella

Fecha Entrega: 12/2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*GameChat: desarrollo de una aplicación Android de chats y juego*
Nombre del autor:	*Chenghuan Ji*
Nombre del consultor/a:	*Pau Dominkovics Coll*
Nombre del PRA:	*Carles Garrigues Olivella*
Fecha de entrega (mm/aaaa):	12/2020
Titulación:	*Máster de Desarrollo de Aplicaciones para Dispositivos Móviles*
Área del Trabajo Final:	*Trabajo fin del máster DADM*
Idioma del trabajo:	*Castellano*
Palabras clave	*Android, Chat, Juego-Multijugador*

Resumen del Trabajo.

Existen muchas aplicaciones de chats en el mercado, pero desafortunadamente la mayoría no son amenas.

En este trabajo se creará una aplicación de chats añadiéndole la funcionalidad de jugar un juego con su amigo. Por la sencillez, se ha escogido el juego Tic Tac Toe.

Se ha implementado empezando por la planificación del trabajo donde se establece fechas de los distintos hitos y las horas necesarias. Seguido del diseño donde además de la creación de los prototipos se ha realizado un estudio sobre el usuario y la arquitectura de la aplicación. Y para finalizar se ha añadido un apartado de conclusiones donde se incluyen las líneas del trabajo futuro.

La aplicación además permitirá la gestión de los amigos como la agregación de un nuevo amigo enviándole una solicitud de amistad, la gestión de las solicitudes de amistad, y la eliminación de los amigos.

La aplicación también permite la visualización del estado de conexión de sus amigos, así como la ocultación del propio estado.

La aplicación se adaptará a los móviles y tablets con versiones de Android igual o superiores al 5.0.

Abstract:

There are many chat apps on the market, but unfortunately most of them are not fun.

In this job a chat application will be created adding the functionality of playing a game with your friend. For simplicity, the Tic Tac Toe game has been chosen.

It has been implemented starting with the planning of the work where dates of the different milestones and the necessary hours are established. Followed by the design where, in addition to the creation of the prototypes, a study has been carried out on the user and the architecture of the application. And finally, a conclusions section has been added where the lines of future work are included.

The application will also allow the management of friends such as adding a new friend by sending him a friend request, managing friend requests, and removing friends.

The application also allows the visualization of the connection status of your friends, as well as the hiding of your own status.

The application will be adapted to mobiles and tablets with Android versions equal to or greater than 5.0.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Desde la aparición del Smartphone, las aplicaciones de tipo chat migraron de PC a móvil como el caso de Skype y otras aplicaciones nacieron directamente desde el móvil y éstas tuvieron incluso más éxito que las antiguas programas de pc. Entre ellas se destaca Whatsapp.

Whatsapp es la aplicación de mensajería más usada en Europa, en la que se envía mensajes mediante internet, así como otros documentos, videos, imágenes, grabación de audio, ubicaciones, etc de forma instantánea. Y permite la realización de llamadas y video llamadas ahorrando el coste que se supondría llamar por teléfono o enviar mensajes SMS tradicionales.

¿Por qué ha triunfado tanto Whatsapp?

1. Simple y sencillo de usar. A diferencia de los programas pc y de las páginas web como la página web Facebook, Whatsapp es una aplicación que mucho menos funcionalidades, y permite que usuarios nuevos aprenda a usarlos intuitivamente sin necesidad de tutoriales complejos.

2. Gratuito. Aunque nació pidiendo un dólar anualmente, pero esta primitiva desapareció en poco tiempo.

3. No hay publicidad obligatoria de ver. Como puede ser los casos del banner que son muy molestos para los usuarios.

¿Pero Whatsapp no tiene ninguna desventaja?

Whatsapp es una aplicación muy útil pero demasiado aburrido, sólo se permite realizar conversaciones y transferir archivos, pero no dispone de ninguna funcionalidad específica nada para entretener a los usuarios. Por lo que he decidido crear una aplicación similar, pero añadiendo amenidades como jugar juegos con su amigo para que la aplicación además de ser útil sea divertida.

Otro de los puntos que puede ser mencionado es, Whatsapp es muy abierto, quizás demasiado. Cualquier persona que tengo su número puede enviarles mensajes sin necesidad de ninguna petición de permiso. Esto puede provoca acoso en algunas ocasiones.

En conclusión, para este trabajo se ha decidido realizar una aplicación móvil para la plataforma de Android similar a Whatsapp pero añadiendo la funcionalidad de invitar a jugar.

Y otras funcionalidades que se piensa en implementar es que hay que agregar a la lista de amigos para poder chatear.

Es muy importante para la aplicación no quitar ese aprendizaje intuitivo del Whatsapp, por lo que se tendrá que diseñar de manera que los nuevos usuarios pueda usarlos sin necesidad de pedir ayudas a otros o buscar tutoriales.

Tampoco se pretende tampoco recrear con este trabajo 100% de las funcionalidades que lleva Whatsapp. Puesto que el trabajo sería demasiado extenso. Sólo se recreará la funcionalidad del chat. Y a partir de ello, añadir las funcionalidades ya mencionadas anteriormente.

Se trata de un trabajo en el que se requiere mayor esfuerzo a la hora de investigar y buscar la información por lo que el proceso de aprendizaje de las tecnologías formará la una gran parte del reto.

1.2 Objetivos del Trabajo

Objetivos globales del trabajo.

- Realizar un aprendizaje sobre la implementación de una aplicación de tipo Chat para Android.
- Realizar un aprendizaje sobre las tecnologías de Firebase.
- Realizar un aprendizaje sobre la implementación de un minijuego multijugador dentro de una aplicación Android.
- Poner en práctica los conocimientos adquiridos a lo largo de todo el máster.
- Adquirir experiencia en afrontar los retos que supone sacar adelante un proyecto completo.

Requisitos básicos funcionales de la aplicación:

- Funcionalidad de registro, inicio y cierre de sesión.
- Funcionalidad de chat.
- Funcionalidad de gestionar el listado de amigos.
- Funcionalidad de jugar un juego con su amigo.
- Se puede usar sin necesidad de registrar. Pudiendo acceder a la aplicación directamente con la cuenta de Google.

Requisitos no funcionales de la aplicación:

- La aplicación debe ser sencilla, intuitiva y fácil de usar.
- Los datos de los chats serán almacenados localmente.

1.3 Enfoque y método seguido

El trabajo se realizará de forma individual aprovechando la máxima flexibilidad en la organización y la planificación.

Para la implementación del trabajo, al tratarse de una aplicación, se puede dividir el trabajo completo en la implementación de trabajos individuales. Y se planificará acorde a la implementación de cada uno de estos trabajos. Puesto que una aplicación está compuesta de muchas funcionalidades, primero se elige

una funcionalidad a implementar, por ejemplo, la funcionalidad de registrar un usuario. Posteriormente se codificará dicha funcionalidad y finalmente se probará en un emulador.

Si todo sale bien se pasará a la siguiente funcionalidad, y en caso contrario se revisará el código y se corregirá los errores.

Esta es una manera segura para obtener un resultado que realmente funcione, sin errores. Quizás no sea la mejor manera de hacer, pero es más preferible un método más seguro, despacio y comprobando paso a paso que uno rápido y arriesgado.

Otro punto a mencionar es al tratarse de un trabajo en el que se desconoce la forma de realizar. Es esencial que haya una parte dedicada exclusivamente a la investigación, buscando información en internet. Una vez que tenga suficiente confianza. Se empezará con el diseño y la implementación del trabajo. Esto ayudará el diseño y la implementación posterior.

1.4 Planificación del Trabajo

Recursos necesarios para la realización del trabajo:

Hardware:

- un ordenador Windows o Mac para el desarrollo del software que tenga conexión a internet.
- un dispositivo móvil Android para el testeo del software.

Software:

- Android Studio: la herramienta principal de desarrollo de aplicaciones Android.
- Axure RP: para el desarrollo de los prototipos.
- Word: para la documentación del memorial.
- Excel: para la realización de las gráficas.
- GIMP: para la creación de las imágenes.

Otros:

- La página de firebase: donde se va a gestionar los bases de datos en la nube e implementación de autentificación y mensajería.
- La página de Lucidchart: para la realización de diagramas flujo y diagramas UML.
- Git: para el control de versiones.

Planificación temporal:
Para la realización de este trabajo no se tendrá en cuenta si se trata de día laboral o día festival.
Se estima una dedicación aproximada de 4 horas diarias.

La realización de este trabajo se divide en 6 etapas:

1. Planificación.
2. Investigación.
3. Diseño.
4. Implementación.
5. Entrega.
6. Defensa virtual.

Planificación:
Es la etapa inicial del trabajo donde se escoge el tema de la aplicación a implementar y se realiza un plan a seguir en todo el trabajo.

Horas previstas	Fecha de inicio	Fecha final
15	16-09-2020	07-10-2020

Investigación:
Esta etapa es necesaria debido al desconocimiento de las tecnologías a usar para la implementación del código y la falta de experiencia en la realización de aplicaciones similares.

Se trata de investigar la realización de una aplicación de tipo un chat y la realización de un juego multijugador utilizando las tecnologías firebase.

Tarea	Horas previstas	Fecha de inicio	Fecha final
Investigación del chat	40	18-09-2020	10-10-2020
Investigación del juego	15	11-10-2020	20-10-2020

Diseño:
En esta etapa se realizará una investigación de los posibles usuarios a dirigir y definir los posibles escenarios, también se diseñará la forma de la aplicación realizando prototipos necesarios diseñando las interfaces graficas de todas las pantallas.

Y finalmente se definirá la arquitectura que se utilizará para implementar el código, así como definir las clases y métodos principales mediante la realización de un diagrama UML.

Tarea	Horas previstas	Fecha de inicio	Fecha final
Investigación sobre el usuario y escenarios	10	08-10-2020	12-10-2020
Diseño del prototipado	20	13-10-2020	21-10-2020
Definición de la arquitectura	15	22-10-2020	28-10-2020

Implementación:
Es la etapa donde se implementará la aplicación diseñada en la etapa anterior. También se realizará las pruebas necesarias para garantizar un correcto funcionamiento de la aplicación.

Como mencionado anteriormente, se dividirá la implementación de toda la aplicación en la implementación de cada una de las funcionalidades.

Tarea		Horas previstas	Fecha de inicio	Fecha final
Funcionalidad de autenticación	Implementación	15	29-10-2020	02-11-2020
	Prueba y resolución de errores.	3	03-11-2020	03-11-2020
Funcionalidad de amigos	Implementación	30	04-11-2020	09-11-2020
	Prueba y resolución de errores.	3	10-11-2020	10-11-2020
Funcionalidad de chat	Implementación	65	11-11-2020	18-11-2020
	Prueba y resolución de errores.	15	19-11-2020	22-11-2020
Funcionalidad de notificación	Implementación	20	23-11-2020	26-11-2020
	Prueba y resolución de errores.	3	27-11-2020	27-11-2020
Juego	Implementación	15	28-11-2020	30-11-2020
	Prueba y resolución de errores.	3	01-12-2020	01-12-2020
Testeo final y resolución de errores		15	02-12-2020	04-12-2020
Memoria e instrucciones de compilación		40	04-12-2020	09-12-2020

Entrega:
En esta etapa se realiza la preparación de la entrega del trabajo. Se realizará las correcciones necesarias y se hará la presentación del trabajo.
También se realizará un manual de usuario y un manual de instrucciones de compilación en el emulador o dispositivo.

Tarea	Horas previstas	Fecha de inicio	Fecha final
Corrección de los posibles errores.	15	10-12-2020	15-12-2020
Preparación de la entrega: aplicación, memoria, manual…	20	16-12-2020	25-12-2020
Presentación	10	26-12-2020	30-12-2020

Defensa virtual.
Se trata ya no del desarrollo de la aplicación, pero forma parte del trabajo fin de máster donde se responderán a las preguntas formuladas por el tribunal de evaluación.

Horas previstas	Fecha de inicio	Fecha final
5	11-01-2021	15-01-2021

Resumen de las etapas:

Etapa	Horas previstas	Fecha de inicio	Fecha final
Planificación	15	16-09-2020	07-10-2020
Investigación	55	18-09-2020	20-10-2020
Diseño	45	08-10-2020	28-10-2020
Implementación	227	29-10-2020	09-12-2020
Entrega	45	10-12-2020	30-12-2020
Defensa	5	11-01-2021	15-01-2021
	392	Horas totales	

Diagrama de Gantt de las etapas:

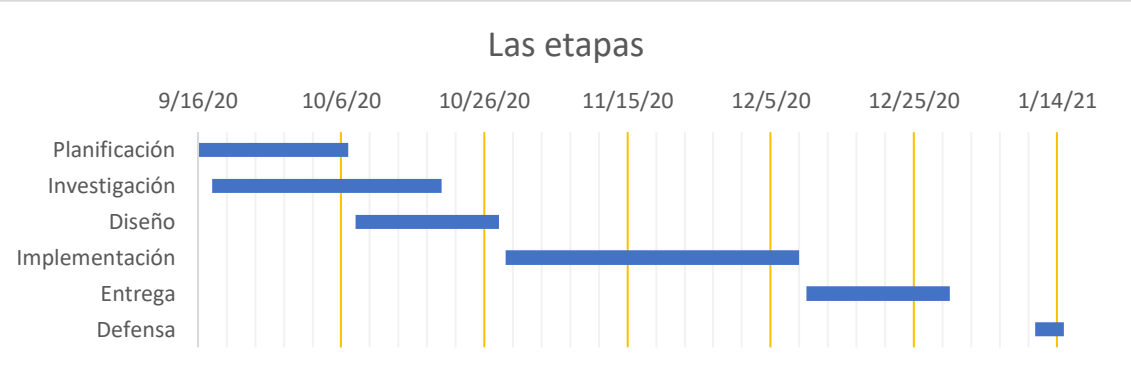

ilustración 1, Las etapas.

Diagrama de Gantt de las tareas:

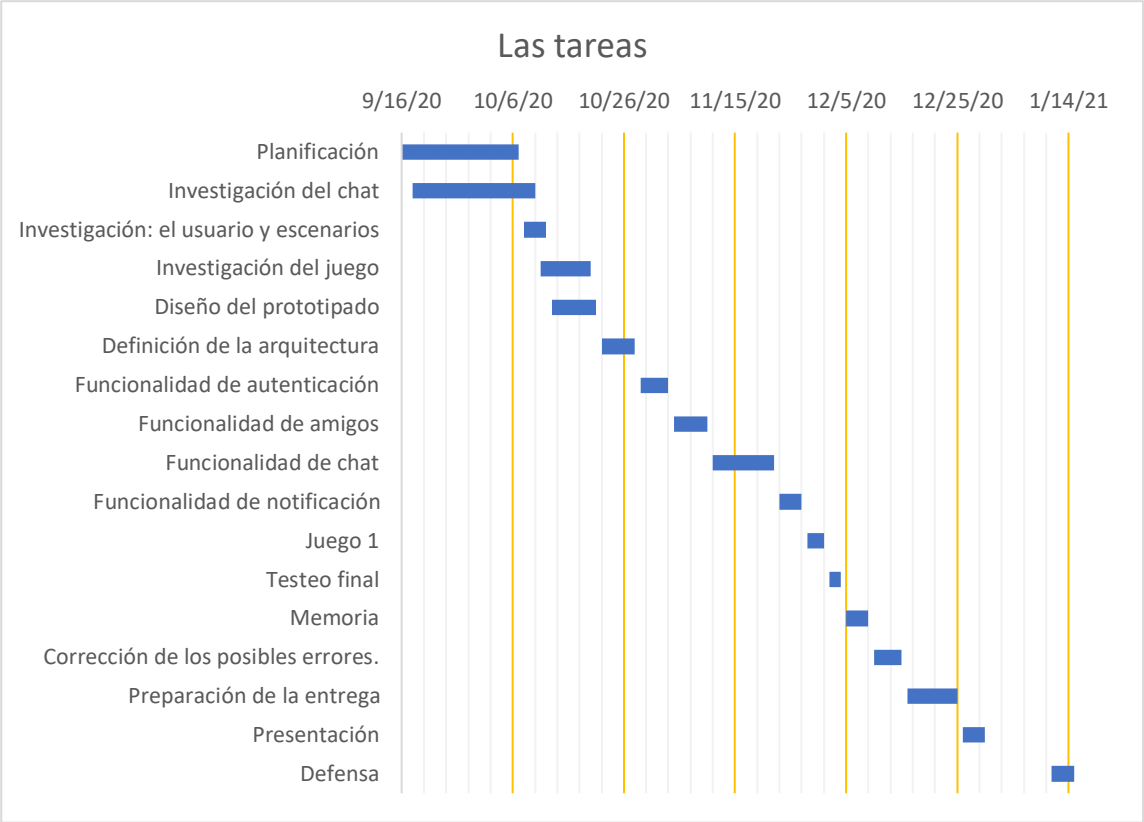

ilustración 2, Las tareas.

1.5 Breve sumario de productos obtenidos

Se debe entregar:

- La aplicación Game Chat. (Código fuente y apk)
- El documento de la memoria del trabajo.
- Las diapositivas de la presentación del trabajo.
- Manual del usuario.
- Instrucciones de compilación en el emulador o dispositivo.
- El vídeo de la presentación del trabajo.

1.6 Breve descripción de los otros capítulos de la memoria

Se comienza inicialmente con el estudio y análisis del diseño y la arquitectura que tendrá la aplicación. Se realizará también un estudio sobre los posibles usuarios, sus necesidades, objetivos, así como el contexto de uso. Esto nos ayuda a definir las funcionalidades necesarias a implementar en nuestra aplicación.

Adicionalmente se realizará los escenarios de uso describiendo desde el punto de vista del usuario de cómo utilizar el producto en casos concretos. Esto nos ayuda a definir el flujo necesario para implementar cada funcionalidad.

Posteriormente se pasa a diseñar los prototipos obteniendo una representación visual de la aplicación cumpliendo todas las funcionalidades definidas anteriormente, y evaluarlo antes de desarrollar el producto final, de tal manera que la aplicación resultante transmita una buena experiencia al usuario.

A continuación, se implementará la aplicación en sí, es cuando se desarrolla el código de la aplicación. Se realizará las pruebas manuales a medida que avance el trabajo asegurando el correcto funcionamiento de cada funcionalidad. En caso de haber errores, se corregirá antes de seguir.

Finalmente, una vez acabada la implementación de la aplicación se realizará el conjunto de documentos necesarios para la entrega incluyendo las instrucciones de compilación de compilación y el manual de usuario.

2. Diseño

2.1 Usuarios y contexto de uso

La aplicación está dirigida principalmente a la población joven, aun que puede ser utilizada por personas de cualquier edad. Puesto que es una aplicación principalmente de chat, no tiene restricciones de edad.

Se pretende atraer el máximo numero de usuarios posibles que supondrá mayor beneficio económico.

Contexto de uso:

- La parte de chat puede ser utilizada en cualquier momento que tenga unos segundos libres. Mientras que la parte de juego necesita más tiempo y será un juego multijugador por lo que necesita también que el otro jugador esté en línea y aceptar la petición.

- Puede utilizarse tanto en el entorno público por ejemplo en el autobús como en el entorno privado por ejemplo en la casa.

- Debe tener la conexión a internet si desea realizar charlar o jugar con su amigo en tiempo real.

- La aplicación debe ser instalada en un dispositivo con el sistema operativo de Android.

- No se recomienda usar la aplicación mientras esté en movimiento, por ejemplo, mientras camina por la calle, para evitar accidentes.

Para la realización de este apartado se ha realizado entrevistas en línea a los amigos.

Un posible usuario puede ser la siguiente:

ilustración 3, Usuario.

Nombre: Luis

Edad: 25

Nivel de estudio: grado

Profesión: dependiente de una oficina inmobiliaria.

Descripción:
Luis es un joven dependiente que trabaja en una oficina inmobiliaria. Vive solo en Madrid en el centro cerca de la estación de metro Sol. Su lugar de trabajo está situado a 30 minutos de su casa de 9 a 15, y suele coger el metro para ir al trabajo. Lleva 2 años trabajando en el puesto. Suele utilizar el móvil para las noticias y la comunicación de sus contactos en la red social. Dispone de bastante tiempo libre al día y tiene muchos amigos. Suele usar la aplicación Whatsapp para conversar con sus amigos y su familia.

2.2 Diseño conceptual

Los escenarios:
Los escenarios de uso describen la interacción de un usuario o varios usuarios con la aplicación con cierta motivación u objetivo. Se describe una breve historia incluyendo la información de cómo el usuario o los usuarios alcanzan su objetivo.

Descripción del escenario 1-Introducción y nuevo amigo:
Érase un día normal como cualquier otra. Quizás no tanto, en ese día Luis recibió una recomendación por parte de José que dice haber encontrado una nueva aplicación bastante amena que permitía tanto conversar como jugar.
Como José era el mejor amigo de Luis, entonces Luis decidió probarla.
Instaló la aplicación bajándola del Google Play.
Al entrar en la app, vio que podía entrar directamente con su cuenta google sin necesidad de Registrar. A eso le encantó Luis. Al entrar, se encontró con varias pestañas, a la que le daba más atención era la pestaña de amigos. Luis hizo un clic allí y no había ningún amigo. Pero vio un gran botón flotante con símbolo de "+". Seguramente que es para añadir un nuevo Amigó, pensó Luis.
Hizo un clic en el "+" y aparición una nueva pantalla que pedía introducir o bien el nombre de usuario de su amigo o bien el correo. Luis sabía el correo de José, así que lo introdujo y allí está su amigo José. Le envió una invitación.
Un rato después José le aceptó la invitación y ya tiene un amigo en la lista de amigos. Hizo clic en el elemento José de la lista de amigos y apareció la pantalla de Chat, desde aquí puede charlar con su amigo José.

Descripción del escenario 2-Jugar:
Es un día llovioso, no hay cliente en la oficina. Luis decide tomar una café y hacer un mini descanso. Saca el móvil, abre la aplicación de GameChat para ver si está alguno conectado.
Afortunadamente, José estaba conectado en aquel momento, le envía un mensaje preguntándole si quiere echar una partida en el "Juego1".
José acepta la petición y empieza la partida. No hay suerte esta vez para Luis, perdió. La partida duró 5 minutos. Tras la partida, Luis decide acabar el descanso y seguir con el trabajo.

Descripción del escenario 3-Notificación:
Érase una mañana en el camino al trabajo.
Luis estaba sentado en el metro mirando el periódico comprado esta mañana al salir de casa.

De repente suena el móvil. Era una notificación de GameChat. Se trata de un mensaje enviada por su novia Ana, pidiéndole una cena junta esta noche. Luis dejó al lado el periódico, sacó el móvil, hizo clic en la notificación, se abrió automáticamente la aplicación GameChat el chat con Ana, y Luis respondió su mensaje aceptando la petición.

Descripción del escenario 4-Borrado de la historia:
El móvil que llevaba José era un smartphone que le había costado sólo 100 euros. Sólo tenía 8GB de memoria interna, de las cuales sólo le quedaba 0.1GB de espacio disponible.
Quería ver si podía borrar algo para liberar más espacio.
Tras haber eliminado varias aplicaciones su movía ya tenía más memoria disponible pero todavía no era suficiente. En aquel momento vio GameChat. Pero no quería borrarlo entonces entro en la aplicación para ver si podía borrar la historia de los chats, seguramente pesan mucho. Hizo un clic largo en uno de los chats y aparición un menú contextual donde apareció la opción de borrar el chat. Entonces borró el chat y con eso borró también toda la historial del chat.

Descripción del escenario 5-Borrar Amigo:
“Es un tío pesado”, esa es la opinión de Luis hacia su nuevo amigo Eduardo. Quizás ya no podía decir que eran amigos porque esta misma tarde tuvieron una riña. Por la noche, nada más de llegar a casa, Luis ya estaba harto de ese Eduardo, y quiere borrarlo de la lista de amigos de GameChat.
Entonces, abre la aplicación, va a la pestaña de Amigos, hace un long clic sobre Eduardo, y apareció la opción de borrar amigo. Sin duda alguna Luis le borró.

2.3 Prototipado

En este apartado se enfoca lo que es el diseño de GUI de la aplicación.

2.3.1 Lista de las pantallas (activities y fragmets) básicas:

1.Pantalla de bienvenida: una pantalla con el icono de GameChat que simplemente es una transición antes de ir a la aplicación en sí.
2.Pantalla de registrar: para registrar en caso de no tener cuenta.
3.Pantalla de iniciar de sesión: para iniciar la sesión.
4.Pantalla de me he olvidado la contraseña: para recuperar la contraseña
5.Pantalla principal dividida en pestañas(viewpager).
    5.1-Pestaña de chats: donde se muestra una lista de chats guardados.
    5.2-Pestaña de amigos: donde se muestra la lista de amigos.
    5.3-Pestaña de solicitudes: donde se muestran tanto tus solicitudes enviadas como las solicitudes dirigidas a ti.
    5.4-Pestaña de perfil: donde se muestran los datos del usuario.
6.Pantalla del chat: donde se realiza la conversación y la petición a jugar.
7.Pantalla de añadir amigo: donde puedes buscar un amigo y realizar solicitud de amistad.
8.Pantalla de juego: donde puedes jugar con su amigo. Puede haber tantas pantallas de juego como los juegos disponibles.

2.3.2 Diagrama de flujo de navegación:

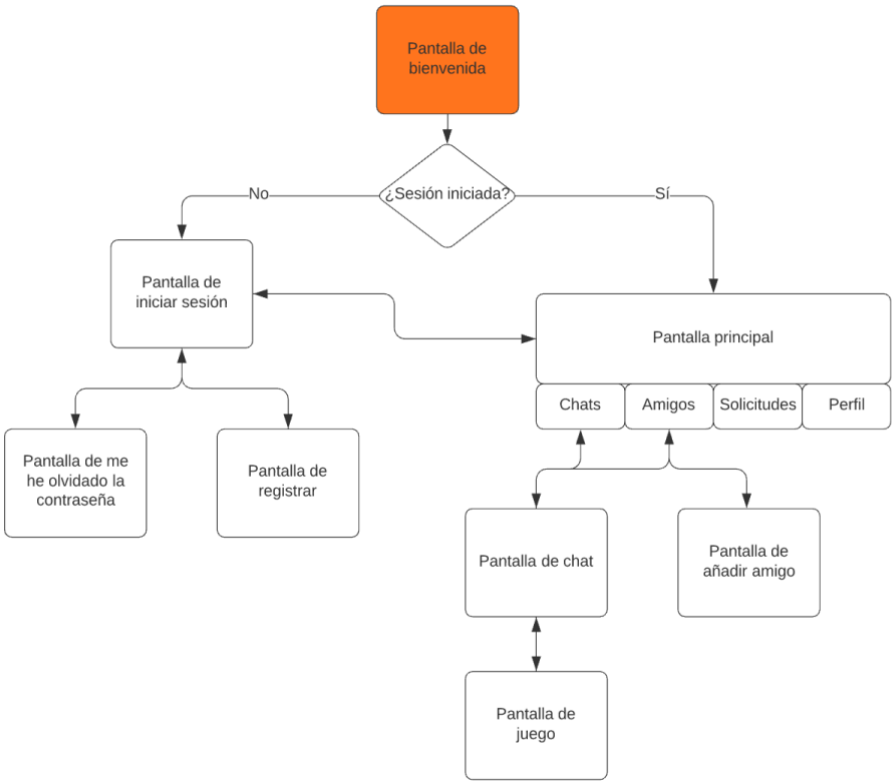

ilustración 4, Navegación de pantallas.

La imagen anterior representa el diagrama de navegación de las pantallas. Comenzando con la pantalla de bienvenida, donde se mostrará el icono de la aplicación y automáticamente nos dirigirá a la pantalla siguiente. En caso de tener sesión iniciada, nos iremos a la pantalla principal, y en caso contrario, nos iremos a la pantalla de registrar.

En la pantalla de registrar el usuario tendrá la opción de que ir a la pantalla de iniciar sesión si ya tiene cuenta.

En la pantalla de iniciar sesión, puede iniciarse la sesión que nos llevará a la pantalla principal. En el caso de que se le ha olvidado la contraseña, puede dirigir a la pantalla de me he olvidado la contraseña para recuperarla.

En la pantalla principal, por defecto entramos en la pestaña de chats. En los chats, al hacer clic uno de los chats nos iremos a la pantalla de chat correspondiente donde podemos realizar conversación o pedir a nuestro amigo a jugar, en ese caso nos iremos a la pantalla de juego.
En la pestaña de amigos, al hacer clic sobre un amigo tiene el mismo efecto que en la pestaña anterior. En esta pestaña tiene además una opción para añadir amigos donde nos dirigimos a la pantalla de añadir amigo.

2.3.3 Diseño de las pantallas

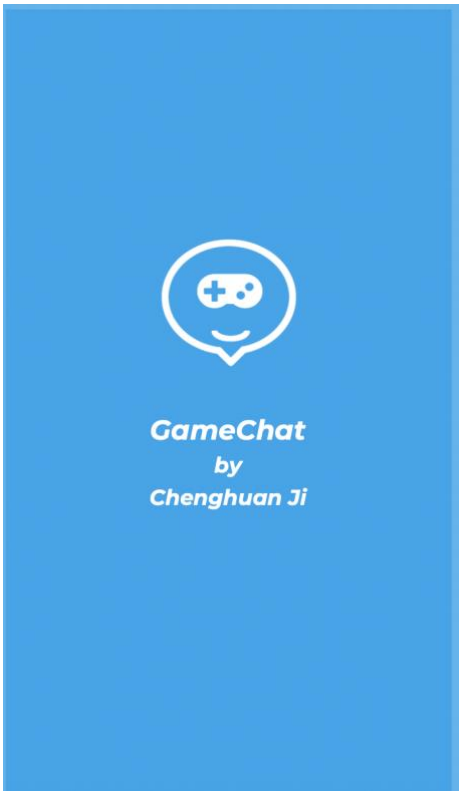

ilustración 5, Pantalla de bienvenida.

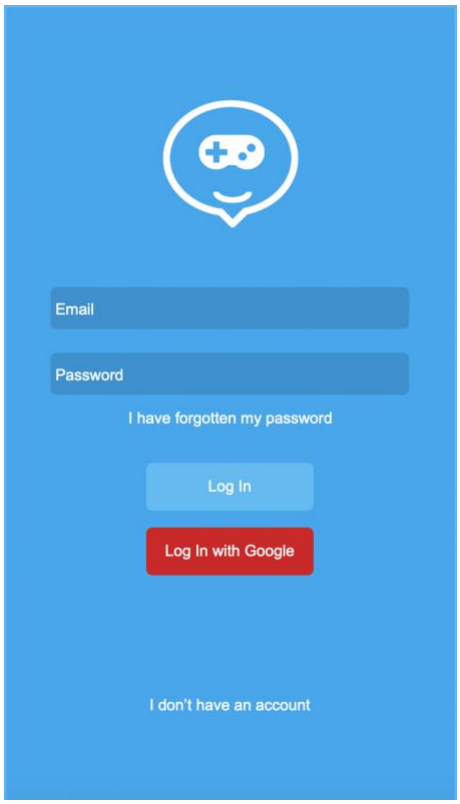

ilustración 6, Pantalla de iniciar la sesión.

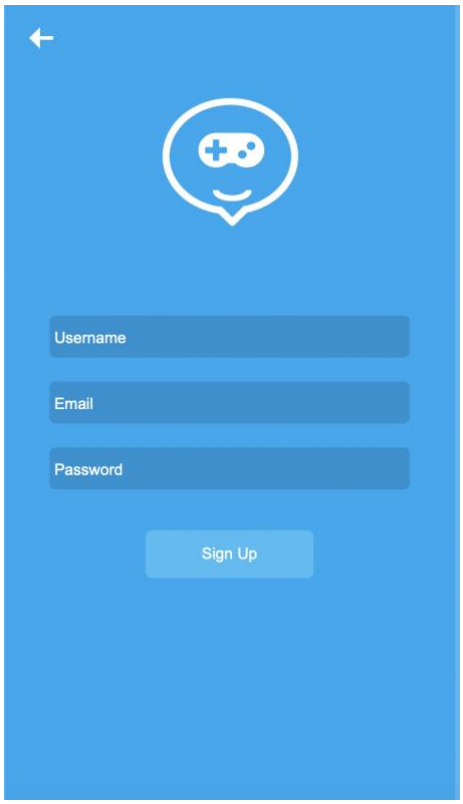

ilustración 7, Pantalla de registrar.

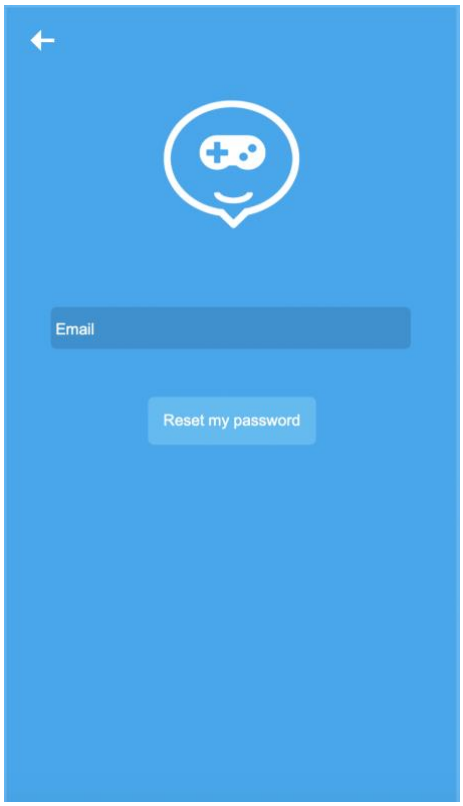

ilustración 8, Pantalla de restablecer la contraseña.

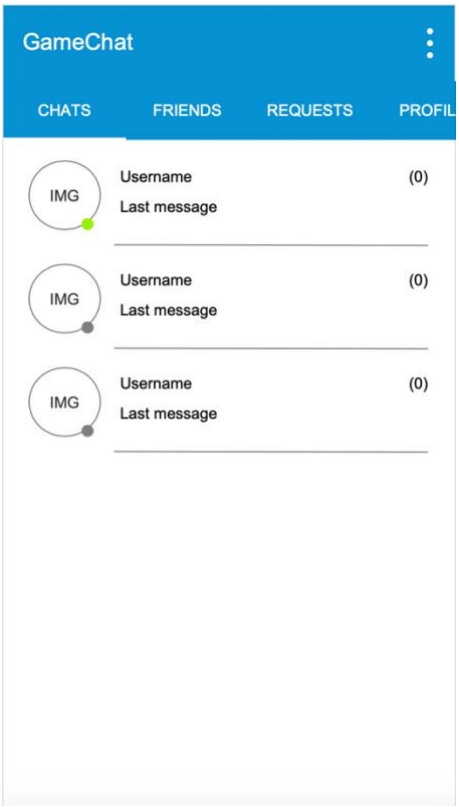

ilustración 9, Pantalla principal de chats.

ilustración 10, Pantalla principal de amigos.

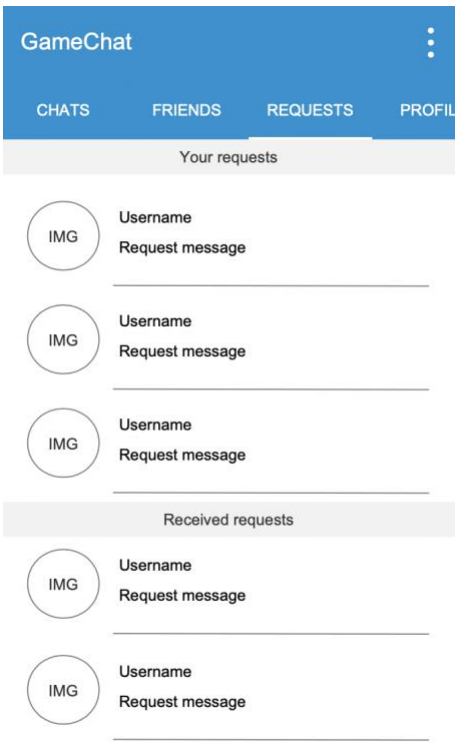

ilustración 11, Pantalla principal de solicitudes.

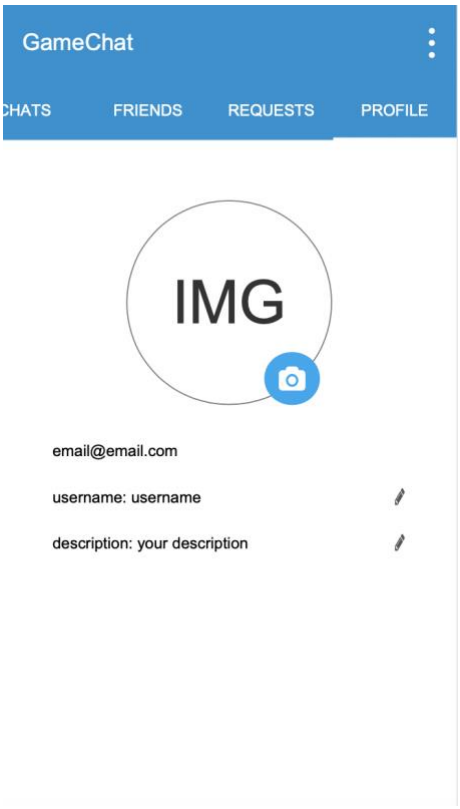

ilustración 12, Pantalla principal de perfil.

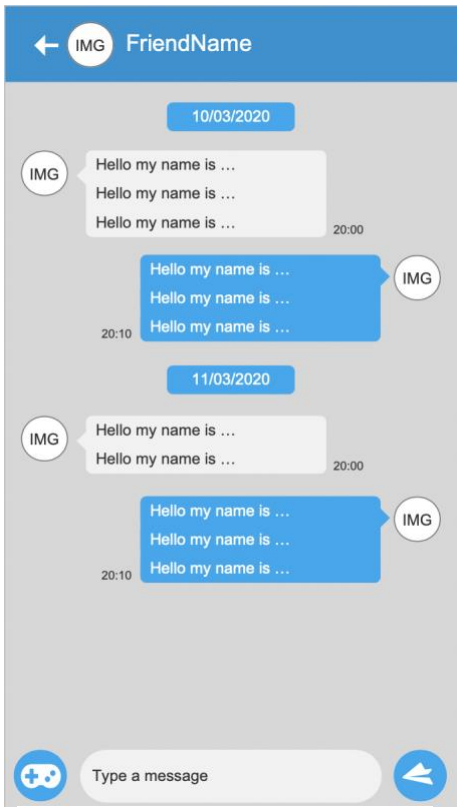

ilustración 13, Pantalla de chat.

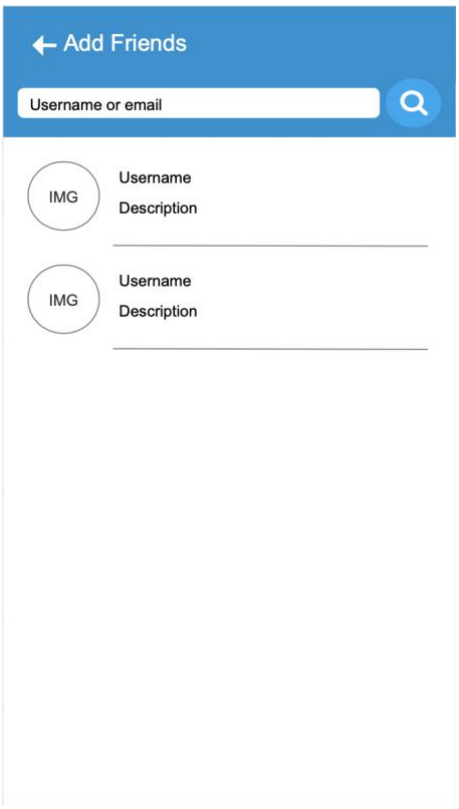

ilustración 14, Pantalla de añadir amigos.

ilustración 15, Pantalla de juego.

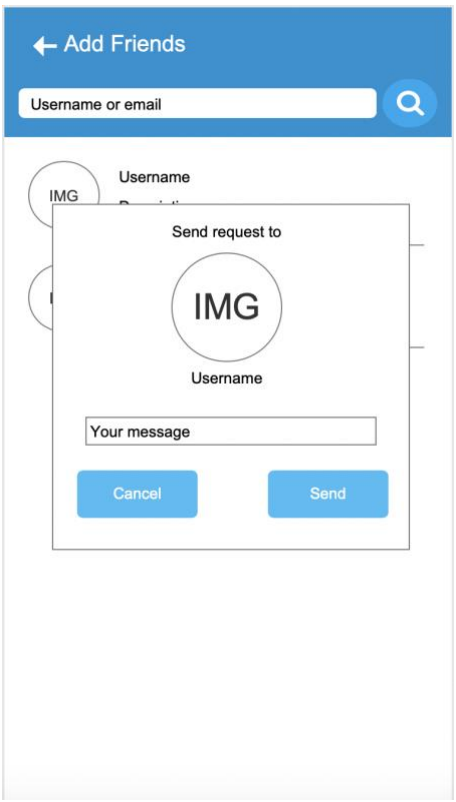

ilustración 16, Dialogo de añadir amigo.

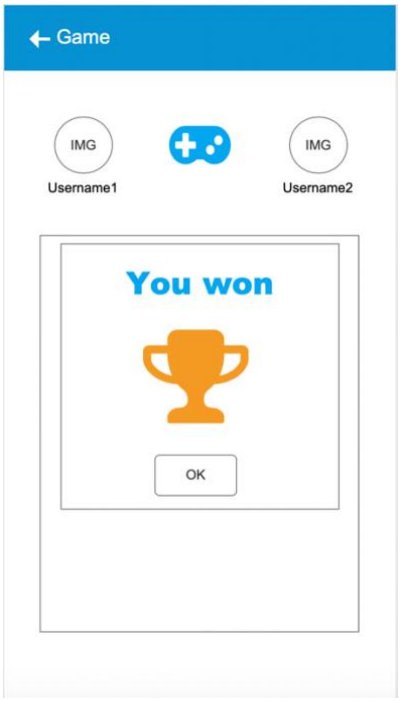

ilustración 17, Dialogo de juego ganado.

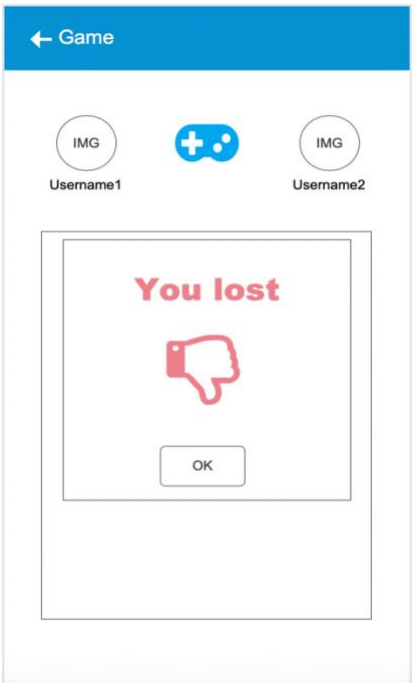

ilustración 18, Dialogo de juego perdido.

Para el registro del nuevo usuario se pide un nombre, el email y la contraseña puesto que el email y la contraseña son necesarios según el API de autenticación de Firebase y el nombre lo necesitamos para mostrarlo en la pantalla principal, y en el juego.

En la pantalla principal de chats se añadirá el número de mensajes sin leer entre paréntesis así el usuario puede saber si tiene algún mensaje sin leer de forma rápida.

En la pantalla principal se utiliza un TabLayout para poder navegar de forma rápida y cómoda para el usuario entre los distintos fragmentos.

En la pantalla principal de perfil no se permitirá la modificación del email puesto que no lo permite la API de autenticación, el email es único para cada cuenta.

En la pantalla principal de solicitudes se han dividido las solicitudes en mis solicitudes y solicitudes recibidas puesto que son tipos diferentes y que podemos tomar diferentes acciones según el tipo. Es decir, para las solicitudes recibidas podemos aceptarlas o rechazarlas, mientras que para mis solicitudes sólo podemos cancelarlas.

En la pantalla de chats se añade también el elemento central que indica la fecha de los mensajes puesto que resulta muy útil para el usuario saber en qué día se el mensaje es enviado.

2.4 Evaluación:

2.4.1 Definición de los casos de uso:

¿Qué es lo que puede hacer el usuario o los usuarios?

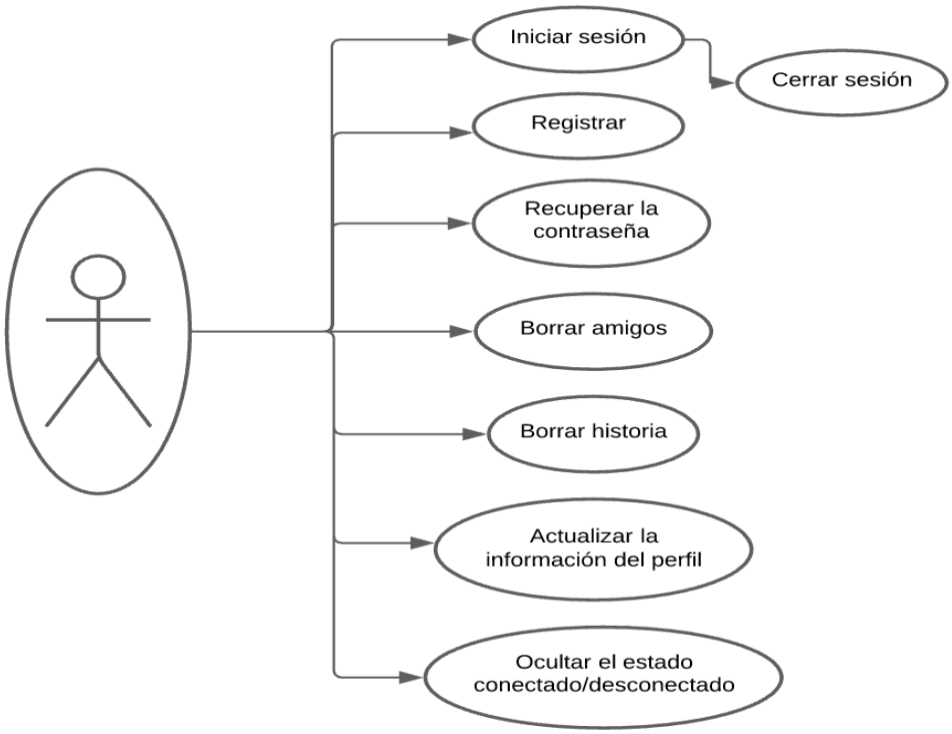

ilustración 19, Uso: un usuario.

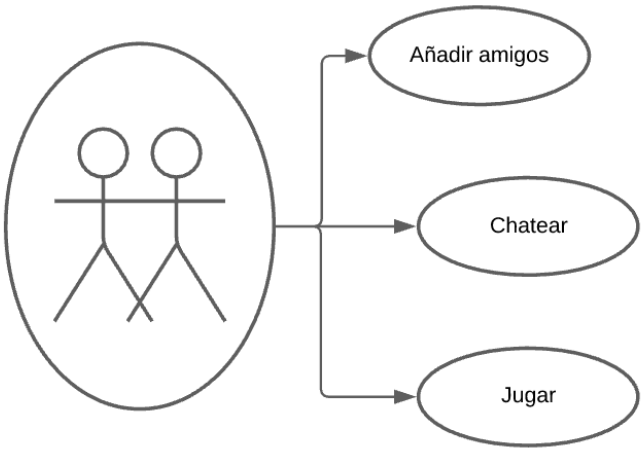

ilustración 20, Uso: dos usuarios.

Lista de los usos en detalle:
Se trata de detallar los usos identificando los actores, precondiciones, flujo y postcondiciones.

Iniciar sesión	
Actores	Usuario1
Precondiciones	La sesión cerrada.
Flujo	Abrir la aplicación -> Introducir los datos de email y contraseña -> Hacer clic en iniciar sesión. O bien Abrir la aplicación -> Hacer clic en iniciar sesión con Google -> Seleccionar su cuenta de Google
Postcondiciones	No

Cerrar sesión	
Actores	Usuario1
Precondiciones	La sesión iniciada.
Flujo	Abrir la aplicación -> Hacer clic en el menú de opciones -> Seleccionar la opción de cerrar sesión
Postcondiciones	No

Registrar	
Actores	Usuario1
Precondiciones	La sesión cerrada.
Flujo	Abrir la aplicación -> Navegar hasta la pantalla de registrar -> Introducir los datos -> Hacer clic en registrar
Postcondiciones	Le enviará un correo de confirmación a la que tiene que confirmar

Recuperar contraseña	
Actores	Usuario1
Precondiciones	La sesión cerrada.
Flujo	Abrir la aplicación -> Navegar hasta la pantalla de recuperar contraseña -> Introducir los datos -> Hacer clic en resetear mi contraseña.
Postcondiciones	Le enviará un correo de recuperación de la contraseña donde puede recuperar su contraseña

Borrar amigo	
Actores	Usuario1
Precondiciones	La sesión iniciada.
Flujo	Abrir la aplicación -> Navegar hasta la pestaña de amigos -> Hacer un clic largo sobre el amigo a borrar -> Selecciona borrar amigo en el menú contextual -> clic en sí que quieres borrar.
Postcondiciones	No

Borrar historia de un chat	
Actores	Usuario1
Precondiciones	La sesión iniciada.
Flujo	Abrir la aplicación -> Navegar hasta la pestaña de chats -> Hacer un clic largo sobre el chat a borrar -> Selecciona borrar chat en el menú contextual -> clic en sí que quieres borrar.
Postcondiciones	No

Actualizar la información del perfil	
Actores	Usuario1
Precondiciones	La sesión iniciada.
Flujo	Abrir la aplicación -> Navegar hasta la pestaña de perfil -> Si quiere cambiar la foto: hacer clic en el icono de la foto que le navegará a la galería para seleccionar la foto. Selecciona la foto y le das a aceptar. Si quiere cambiar la descripción o nombre de usuario: hacer clic en el icono de lápiz -> introducir nueva descripción-> aceptar
Postcondiciones	No

Ocultar el estado conectado/desconectado	
Actores	Usuario1
Precondiciones	La sesión iniciada.
Flujo	Abrir la aplicación -> Hacer clic en el menú de opciones -> Seleccionar la opción de ocultar estado
Postcondiciones	No

Añadir amigo	
Actores	Usuario1, Usuario2
Precondiciones	Ambos tienen que tener la sesión iniciada.
Flujo	Usuario1: Abrir la aplicación -> Navegar a la pestaña de amigos -> clic en “+” -> Buscar su amigo->Escribir un mensaje de solicitud -> Enviar la solicitud. Usuario2: Abrir la aplicación -> Navegar a la pestaña de solicitudes -> clic en la solicitud del usuario1 -> Aceptar
Postcondiciones	No

Chatear	
Actores	Usuario1, Usuario2
Precondiciones	Ambos tienen que tener la sesión iniciada.
Flujo	Usuario1: Abrir la aplicación -> Navegar a la pestaña de amigos o chats -> clic en el amigo con quien quiere chatear-> chatear Usuario2: Recibirá una notificación si no tiene el chat abierto. Clic en la notificación -> Chatear
Postcondiciones	No

Jugar	
Actores	Usuario1, Usuario2
Precondiciones	Ambos tienen que tener la sesión iniciada.
Flujo	Usuario1: Abrir la aplicación -> Navegar a la pestaña de amigos o chats -> clic en el amigo con quien quiere jugar-> chatea un poco y luego enviarle la solicitud de jugar Usuario2: Recibirá una notificación si no tiene el chat abierto. Clic en la notificación -> Chatear y luego aceptar la solicitud de jugar
Postcondiciones	El usuario2 tiene que aceptar la petición de jugar dentro de los 30 segundos tras la solicitud.

2.4.2 Definición de la arquitectura:

A- Diagrama ULM correspondiente al diseño de las clases de datos:

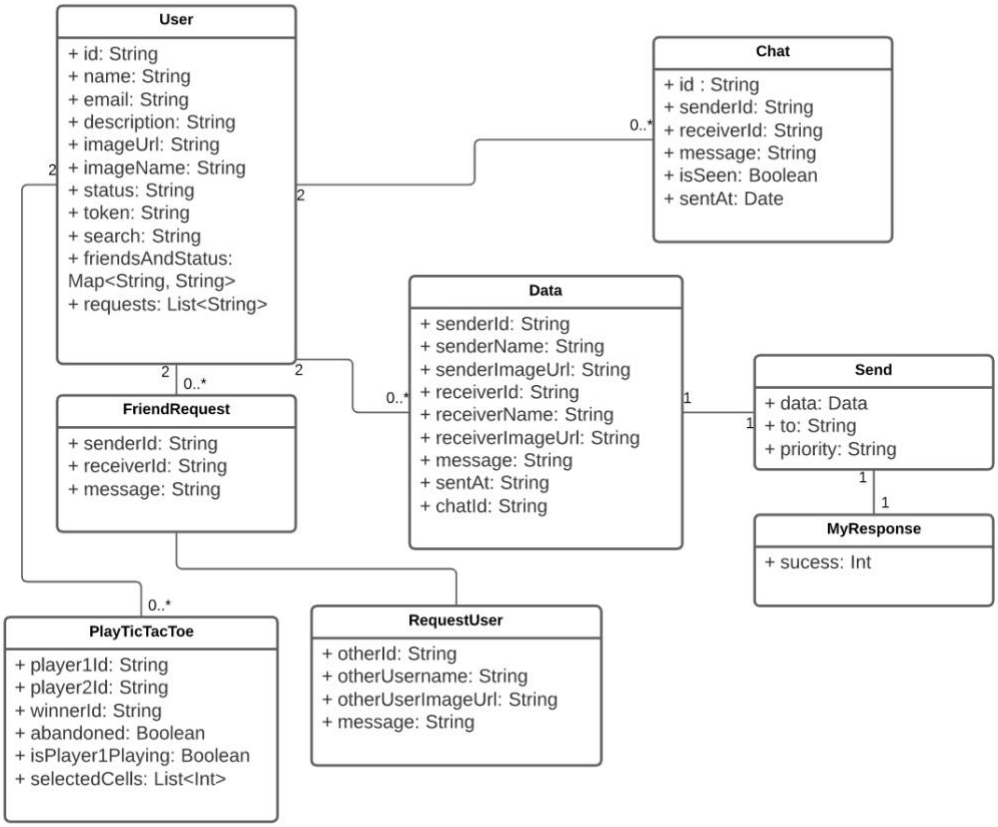

ilustración 21, UML de los data class.

En la ilustración anterior podemos observar el diagrama UML de las data class.

Chat:

La clase User: es una clase para almacenar la información del usuario. La información del propio usuario en la base de datos local puesto que queremos recuperar el usuario de forma rápida y también se guardarán la información de todos los usuarios en la nube.

La clase Chat: es una clase que guarda datos sobre cada mensaje que envía desde un usuario a otro. Serán guardadas de forma permanente (pero puede borrarlo si desea el usuario) en la base de datos local los chats que le corresponden.

Y también serán guardados todos los chats de forma temporal en la nube, o bien hasta que son vistas o bien se las eliminan el administrador de la aplicación

Solicitud de Amistad:

La clase FriendRquest: es una clase que sirve para guardar datos sobre las peticiones de amistad que se realizan entre dos usuarios. Van a ser guardadas

de forma temporal en la nube, hasta que acepten o rechacen la solicitud, pero no van a ser guardadas en la base de datos local.

La clase RequestUser: es una clase utilizada para mostrar las solicitudes en la pantalla principal de solicitudes.

Juego:
La clase Play que es una clase para almacenar la información de la partida de un Juego. Serán guardados de forma temporal en la nube, pero no van a ser guardadas en la base de datos local.

Puede que cuando implemente, necesitaremos más clases similares como, Play2 para manejar el juego2.

Notificaciones.
Las clases Data, Send y MyResponse son clases de datos necesarios para manejar la notificación, pero no van a ser guardados en base de datos. Donde en la clase Data se almacena información de la notificación.
La clase Send por su parte contiene la clase Data más una propiedad “to” indicando el token destinatario.
Y la clase MyResponse sirve para manejar la respuesta que recibimos del servidor.

Para enviar una notificación se utilizará el API de Retrofit que se creará un síngleton y interfaz especial para realizar las peticiones.

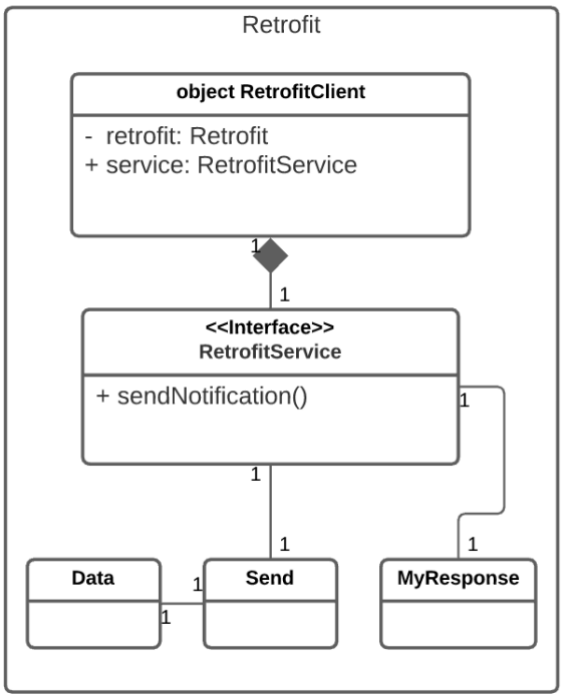

ilustración 22, Estructura de Retrofit.

Retrofit es una librería muy popular en la programación en Android. Permite realizar peticiones http de una forma muy sencilla y permite gestionar diferentes tipos de parámetros y pasar automáticamente la respuesta a un “POJO”(plain old Java object) gracias a los converters de Gson.
La interfaz de RetrofitService contiene todas las funciones que se utilizará para las peticiones http en este caso sólo tenemos una de tipo POST que nos servirá para enviar el mensaje a un servidor de Firebase Cloud Messaging.

El objeto(singleton) sirve para poder manejar de forma más fácil el API de retrofit desde cualquier parte de la aplicación.

Manejo de BBDD:
Para manejar los datos en la nube se creará un Singleton aparte de la siguiente forma:

ilustración 23, El síngleton FirebaseManager.

De esta manera nos permitirá utilizar estas funciones fácilmente en toda la aplicación.

Y para manejar el base de datos local se utilizará la herramienta Room creando siguientes clases e interfaz adicionales.

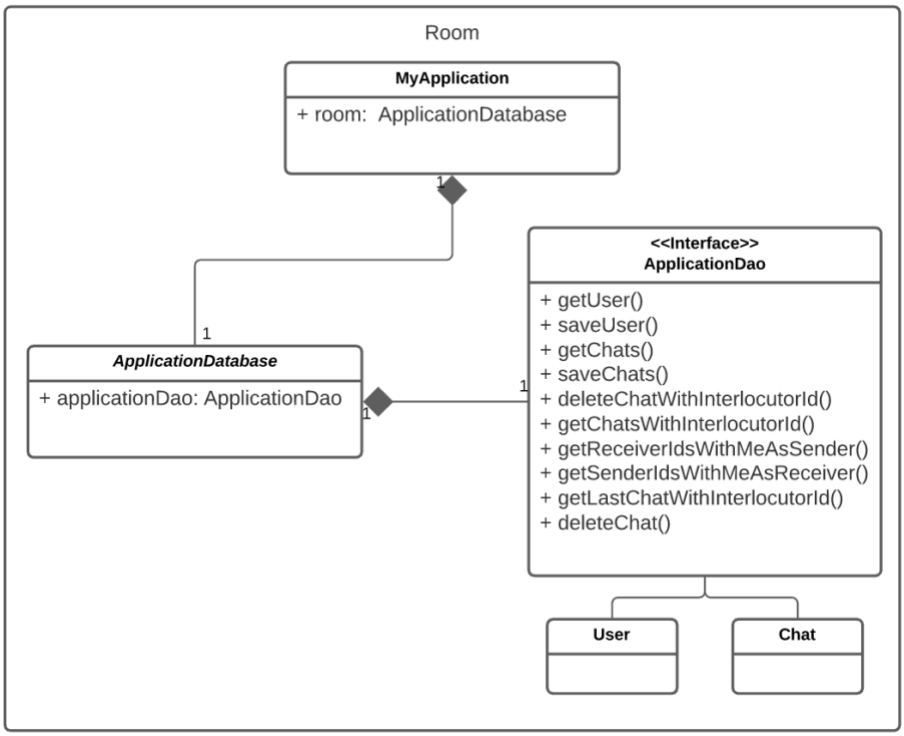

ilustración 24, Estructura del room.

Donde la clase MyApplication será la clase hija de Application, que es la clase de la aplicación, donde nos instanciamos un objeto público room con el cual nos manejaremos el base de datos.

ApplicationDatabase es una clase abstracta donde se indica las tablas mediante un decolador @Database y se asocia con la interfaz ApplicationDao.

ApplicationDao es la interfaz donde se albergan los métodos de petición al base de datos.

SharedPreferences:
Recordemos que se trata de un documento XML que sirve para guardar datos simples.
Adicionalmente se creará una clase MyPreferences para simplificar el uso de las sharedPreferences.

B- El diagrama UML correspondiente al diseño de las entidades y clases.

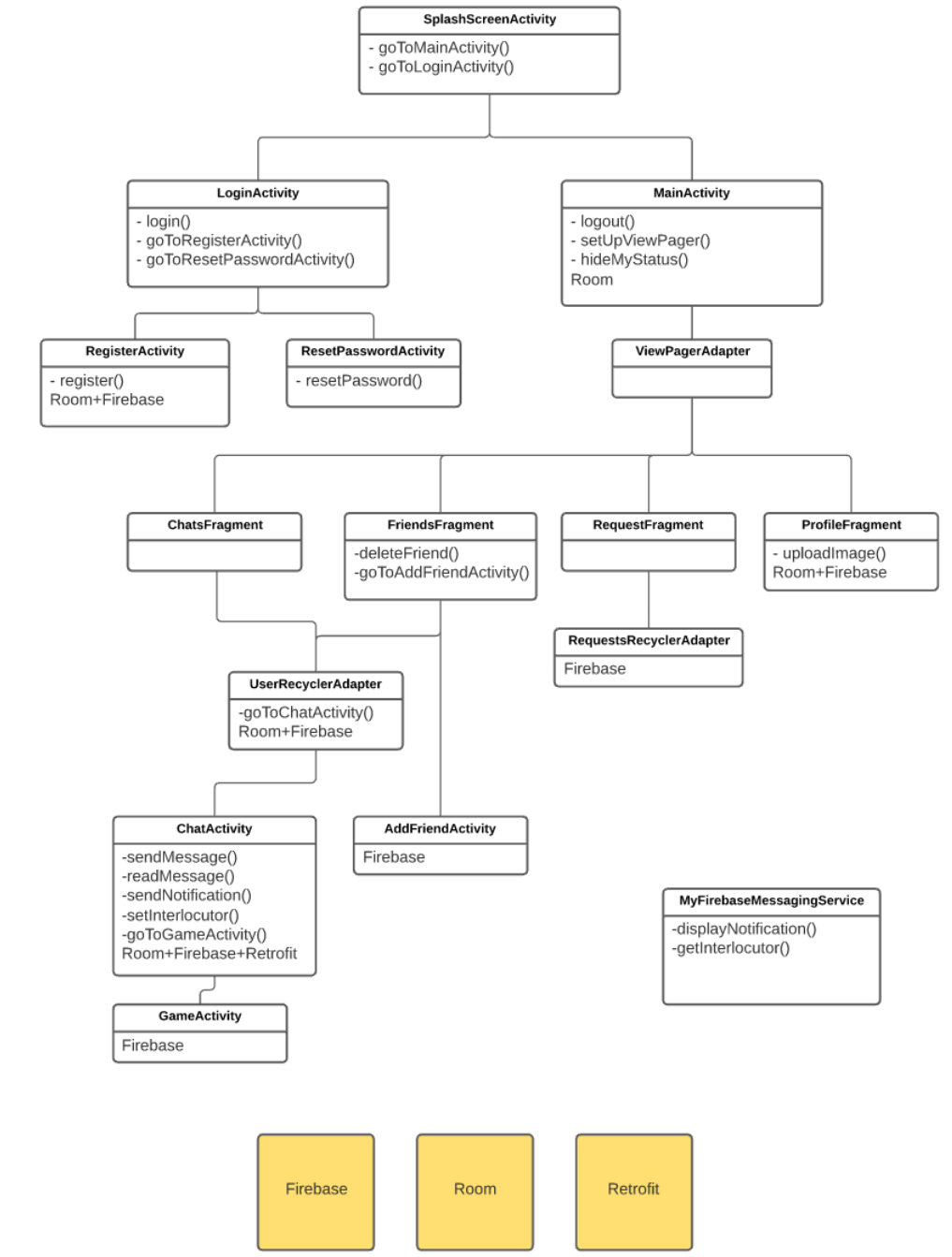

ilustración 25, Estructura de la aplicación.

En el diagrama anterior se muestra el UML de toda la aplicación donde sólo se ha incluido los métodos más importantes.

Nota: si en una clase aparece Room, Firebase o Retrofit, significa que utiliza el bloque de Room, Firebase o Retrofit mencionado en el apartado anterior.

C-Un diagrama explicativo de la arquitectura del sistema

Se aplicará el patrón de diseño de MVC (Modelo-Vista-Controlador).
Se trata de un patrón que permite crear aplicaciones fácilmente mantenibles y es comúnmente usado para desarrollar las interfaces del usuario. MVC separa la aplicación en tres partes: el modelo, la vista y el controlador.

- Modelo: es la parte donde se recoge toda la información de datos desde o bien una base de datos local o bien desde un servidor. Es la parte más reutilizable, podemos portar fácilmente el modelo de una aplicación a otra. En nuestro caso es la parte que se ha definido en el apartado A.

- Vista: reúne toda la información gráfica de la aplicación, es lo que el usuario puede ver en la pantalla cuanto se ejecuta la aplicación. En Android, la vista suele estar definida por archivos en lenguaje XML donde se definen fácilmente los layouts y otros elementos gráficos, aunque también se puede definir directamente los elementos visuales en las clases derivadas de Activity pero no es muy común.

- Controlador: funciona como la unión entre vista y modelo. Es donde se implementan todas las funcionalidades de la aplicación. En Android los controladores suelen estar definido por las clases derivadas de Activity, Fragment, Adapter, Service y otros.

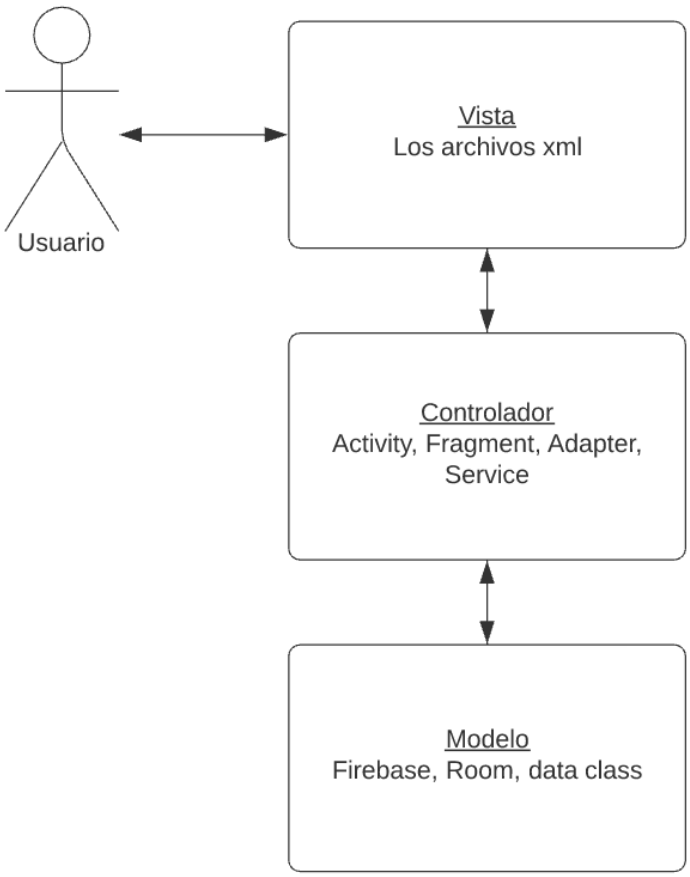

ilustración 26, Arquitectura del sistema.

3. Implementación

3.1 Introducción y bienvenida

Para la ordenación de este trabajo se ha ordenado los archivos en una serie de carpetas dependiendo del tipo archivo y clase correspondiente.

Al tratarse de una aplicación de chat, donde interviene varios dispositivos, no ha sido posible las pruebas automatizadas. Por lo que todas las pruebas han tenido que realizarlas manualmente.

3.1.1 Los archivos de kotlin

Los archivos de kotlin se ordenan en:

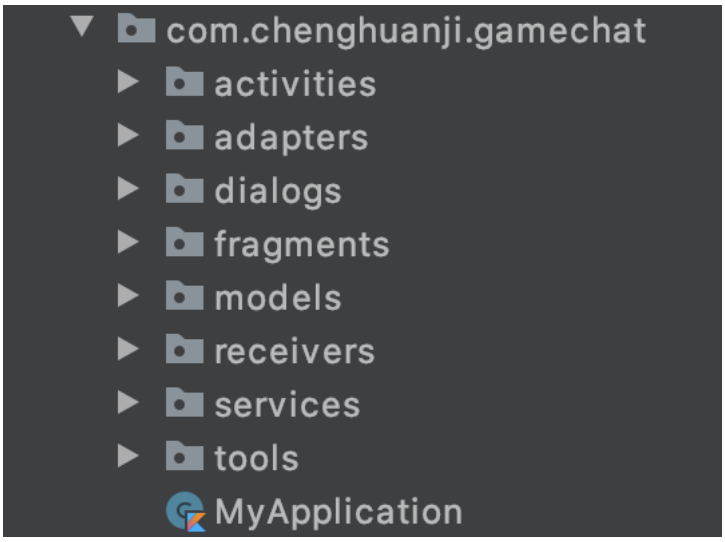

ilustración 27, distribución de carpetas de los archivos kotlin.

La clase MyApplication es una clase asociada a la aplicación en sí.

En la carpeta de activities de sitúan las clases correspondiente a las actividades de Android.
Recordando que las actividades en Android de refieren a cada una de las pantallas.

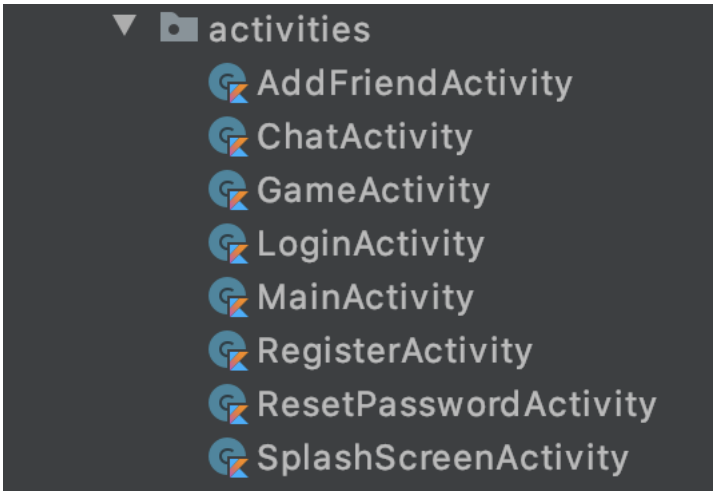

ilustración 28, lista de actividades.

En la carpeta de los adapters se sitúan los adaptadores que son clases necesarias para el manejo de unos views especiales de Android como el recyclerview y el viewpager.

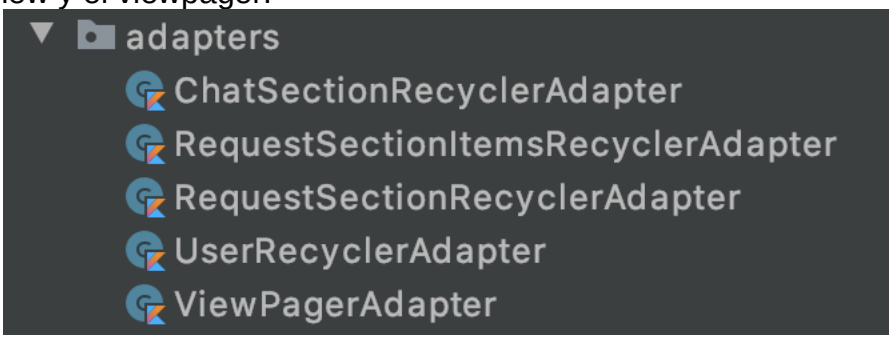

ilustración 29, lista de adaptadores.

En la carpeta de dialogs se sitúan las clases correspondientes a los distintos diálogos. Recordamos que los diálogos en Android son las “ventanas” flotantes.

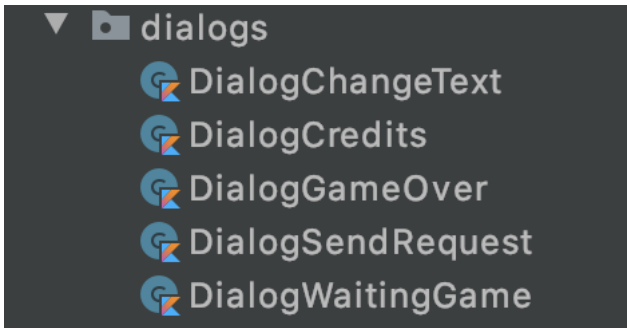

ilustración 30, lista de diálogos.

En la carpeta de fragments se sitúan las clases correspondientes a los fragmentos. Recordamos que los fragmentos en Android una porción de una pantalla.
En nuestro caso los fragmentos son necesarios para el funcionamiento del viewpager.

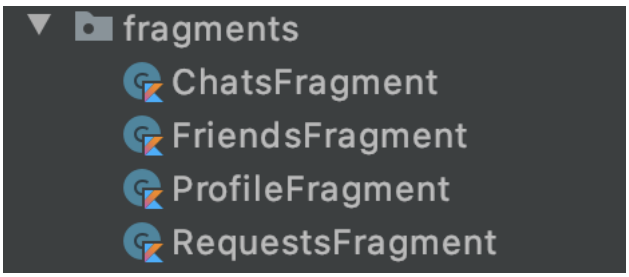

ilustración 31, lista de fragmentos.

En la carpeta models se sitúa todo el bloque modelo definido en el apartado de la arquitectura. Que son concretamente las clases de datos y otras clases necesarias para el funcionamiento de Firebase, Room, SharedPreferences y notificaciones.

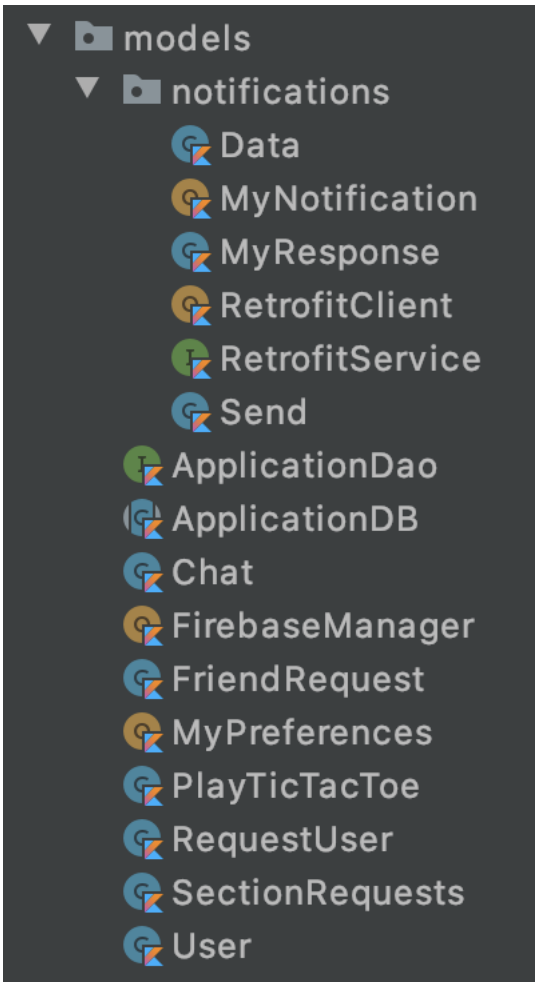

ilustración 32, lista de modelo.

En la carpeta de receivers se sitúa el receptor de transmisión (broadcast receiver) que es necesario para poder recibir las acciones de marcar como leído en las notificaciones.

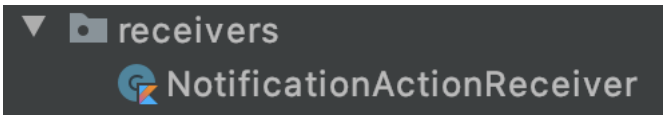

ilustración 33, lista de receptores.

En la carpeta de services se sitúan las clases correspondientes a los servicios. Recordamos que los servicios son componentes que realizan tareas en segundo plano a largo periodo.

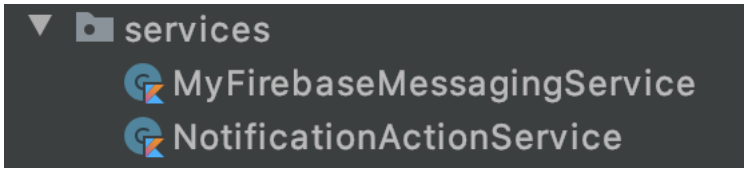

ilustración 34, lista de servicios.

Y en la carpeta de tools se sitúan las herramientas necesarias para el adecuado funcionamiento de otras clases o para facilitar la programación.

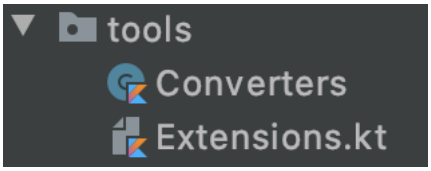

ilustración 35, lista de tools.

Concretamente, la clase converters es una clase donde se lista una serie de convertidores de clases que son necesarias para el adecuado funcionamiento de Room y el archivo Extensions.kt se agrupan los extension functions de Kotlin que facilitan la programación.

3.1.2 Los archivos XML y otros

Los archivos XML están ordenados en:

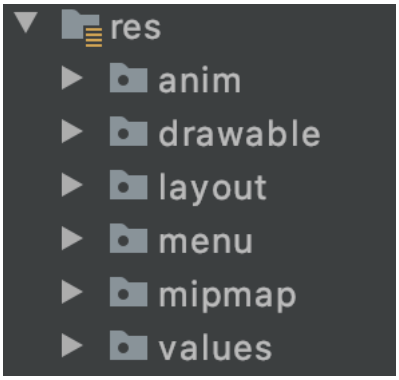

ilustración 36, lista de archivos XML.

En la carpeta anim se sitúan los archivos XML correspondientes a las animaciones.
En la carpeta drawable se sitúan los archivos XML y otros archivos correspondientes a las imágenes utilizadas en la aplicación, los diseños de cuadros para escribir y los ripples, que son las ondas que aparecen al dar clic sobre un elemento clicable.
En la carpeta layout se sitúan los archivos XML correspondientes al diseño de las actividades, los fragmentos, los diálogos, etc.
En la carpeta menú se sitúan los archivos XML correspondientes a los menús utilizados.
En la carpeta mipmap se sitúan los archivos utilizados para el icono de la aplicación.
Y en la carpeta values se sitúan los archivos de colors.xml, dimens.xml, strings.xml y themes.xml que sirven para definir color, dimensión, texto y temas usados.

También cabe mencionar otros archivos importantes como el AndroidManifest.xml donde se declara el conjunto de actividades, servicios y receptores y sus propiedades, también es el lugar donde se piden los permisos.
Y los archivos Gradle, donde podemos añadir las dependencias, variaciones en la compilación, etc.

3.1.3 La pantalla de bienvenida

Es una pantalla donde se muestra el icono, el nombre de la aplicación y el desarrollador.

Es la pantalla que se muestra al iniciar la aplicación. Se espera medio segundo y se redirigirá automáticamente a la siguiente pantalla.

En el caso de no tener la sesión iniciada se redirigirá a la pantalla de iniciar la sesión y en el caso de tener la sesión iniciada se redirigirá a la pantalla principal.

La funcionalidad está escrita en la actividad SplashScreenActivity y los elementos gráficos en el archivo activity_splash.xml.

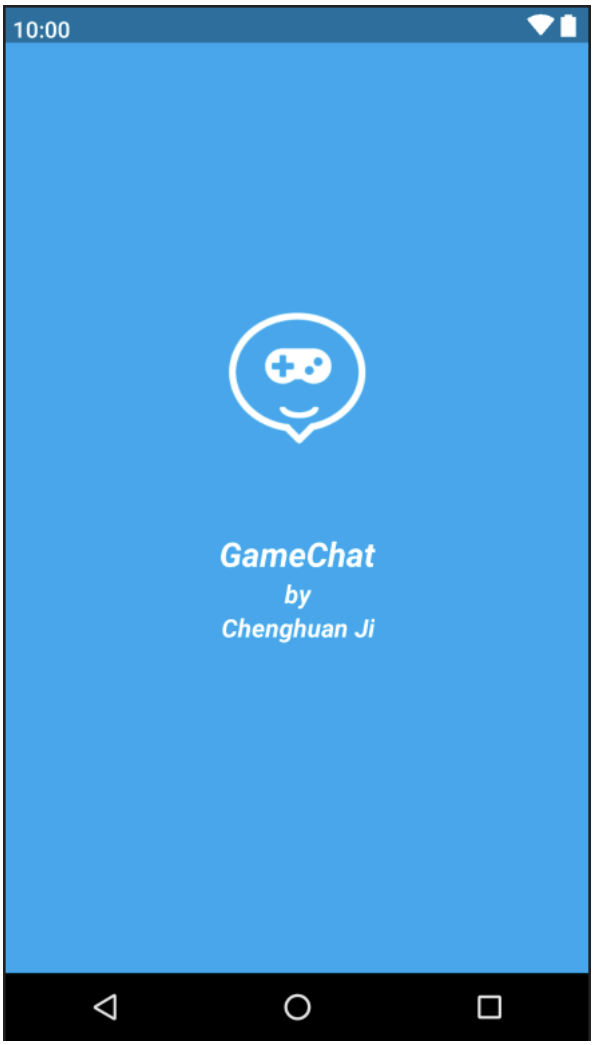

ilustración 37, pantalla de bienvenida implementada.

3.2 Funcionalidad de autenticación

La funcionalidad de autenticación se puede dividirse en “subfuncionalidades” de registrar, iniciar sesión y recuperar la contraseña que corresponden a las pantallas de registrar, iniciar sesión y restablecer la contraseña respectivamente.

Para implementar estas funcionalidades se ha utilizado la tecnología de **Authentication** de Firebase que proporciona servicios de backend, SDK fáciles de usar y bibliotecas de IU ya elaboradas para autenticar a los usuarios en tu app. Admite la autenticación mediante contraseñas, números de teléfono, proveedores de identidad federada populares, como Google, Facebook y Twitter, y mucho más.

3.2.1 Funcionalidad de registrar

La funcionalidad está escrita en la actividad RegisterActivity y los elementos gráficos en el archivo activity_register.xml.

Para el registro de un nuevo usuario, se debe introducir un nombre de usuario, un correo y una contraseña.

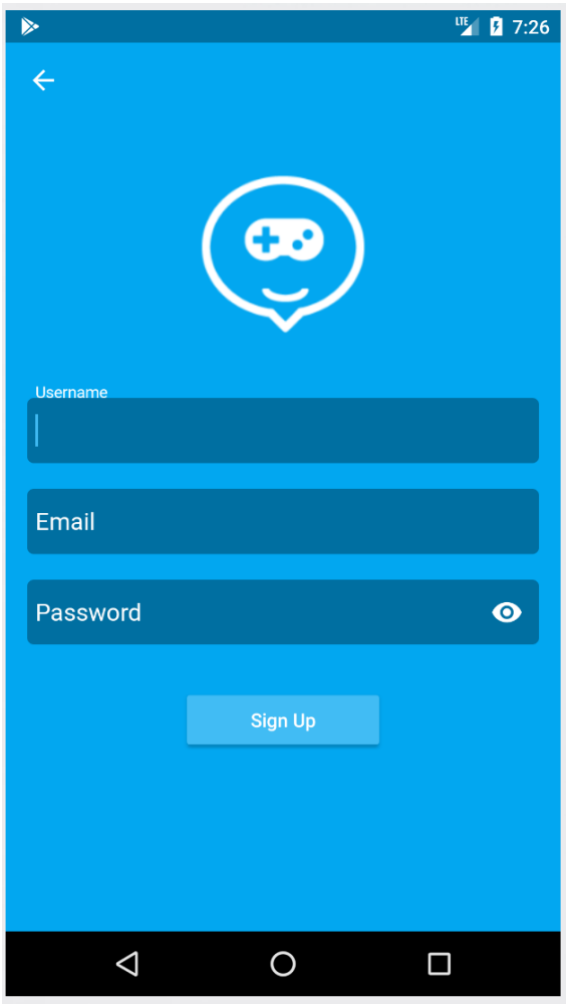

ilustración 38, pantalla de registro implementada.

Para el campo de contraseña se ha añadido un “elemento ojo” para poder ocultar/mostrar la contraseña, de esta manera el usuario puede comprobar si se ha introducido correctamente.

ilustración 39, campos de contraseña.

Se validará los campos antes de realizar la petición realizando las siguientes comprobaciones:
El nombre de usuario debe contener al menos 6 caracteres.
El email debe ser válido. Android tiene un patrón especializado para email.
La contraseña debe contener al menos 6 caracteres y al menos una letra y un número.

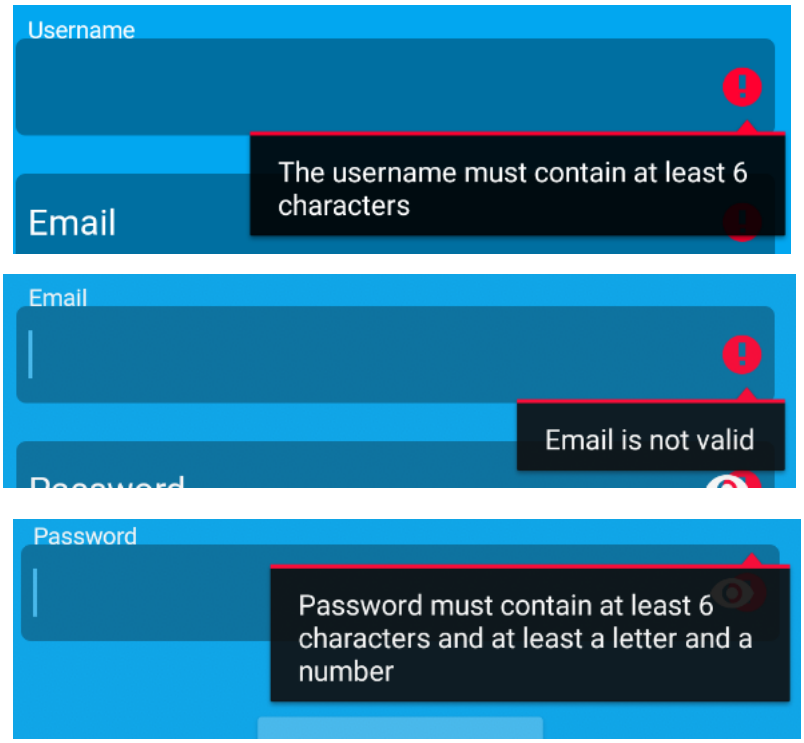

ilustración 40, validaciones.

Al registrar también se guardará temporalmente en sharedPreferences los datos de username y email para el posterior almacenamiento en bases de datos de firebase y bases de datos Room.
Este proceso ha tenido que realizarlo debido a que no tenemos todavía verificado el email y no vamos a escribir en bases de datos de firebase si no tiene email verificado.

Al registrar nos volvemos automáticamente a la pantalla de iniciar sesión.

Tras el registro se le enviará un correo donde tiene que verificarlo antes de iniciar la sesión.
Nota: el correo puede estar en “Correo no deseado o spam”.

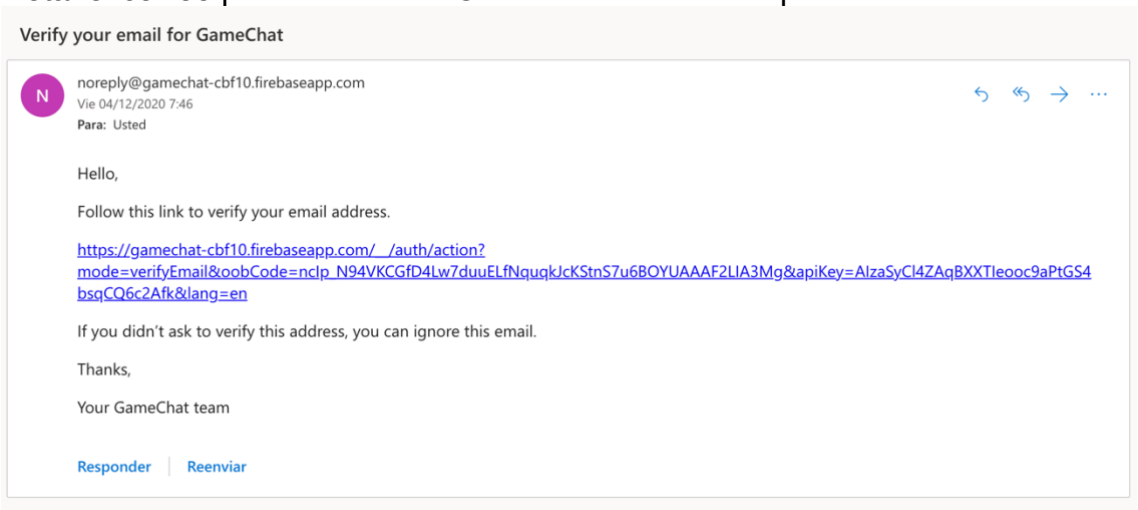

ilustración 41, email de confirmación.

<u>Prueba de que se ha registrado correctamente.</u>
Como mencionado anteriormente, las pruebas se han realizado manualmente. Se ha comprobado que sí aparece efectivamente en la página de firebase.

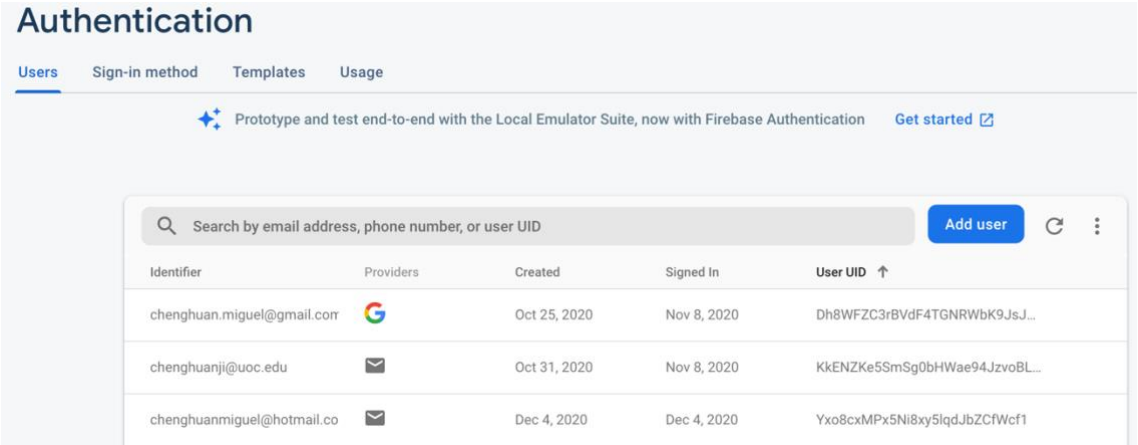

ilustración 42, prueba de registro.

3.2.2 Funcionalidad de iniciar sesión

La funcionalidad está escrita en la actividad LoginActivity y los elementos gráficos en el archivo activity_login.xml.

Se trata de iniciar la sesión si ya tiene cuenta o iniciar la sesión con la cuenta de Google.

También aprovecharemos aquí para guardar la información de usuario en base de datos de firebase y base de datos locales con Room.

¿Por qué no habíamos guardado estos datos en BBDD cuando registramos?
Por un lado, como mencionado anteriormente no teníamos verificado el email. Y por otro lado si se entra directamente con la cuenta de Google, no tendrá que registrar, sino que tiene que sacar la información del usuario cuando inicia la sesión y aprovecharla para guardar en BBDD.

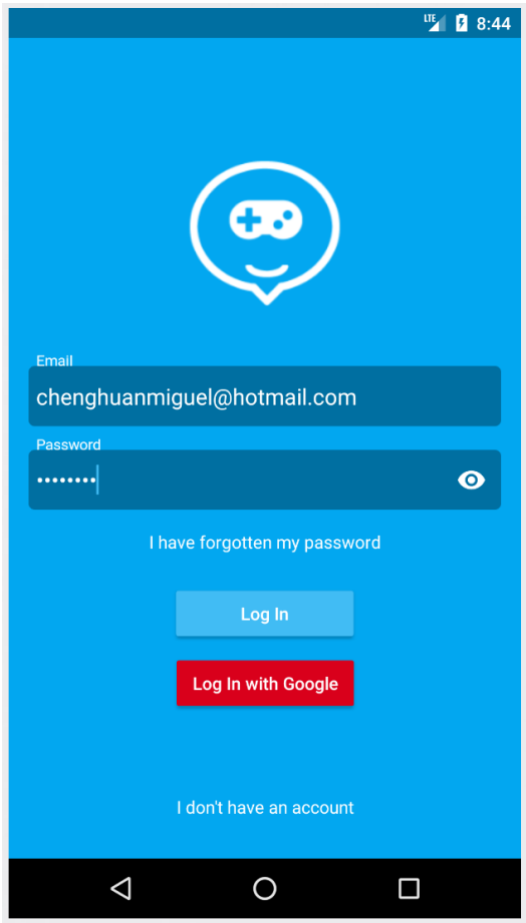

ilustración 43, pantalla de inicio implementada.

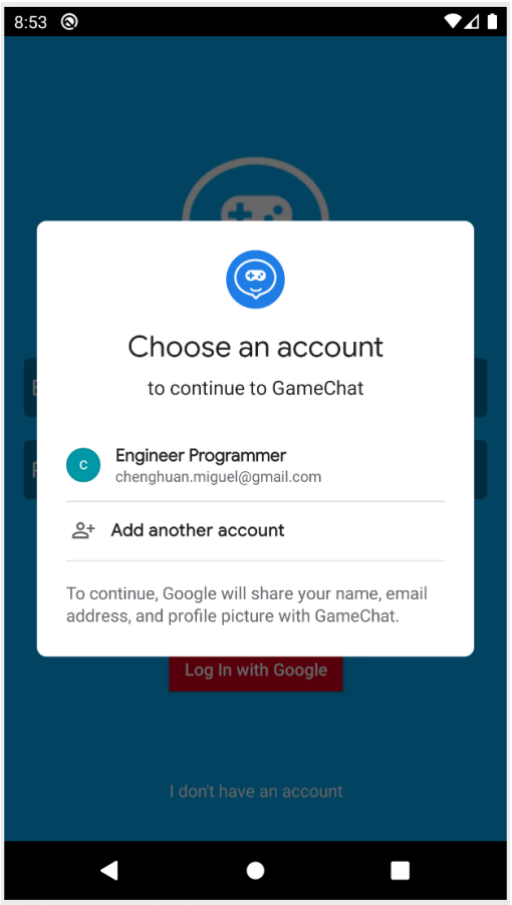

ilustración 44, pantalla de inicio con Google implementada.

Como puede observar, para iniciar la sesión se necesita introducción email y contraseña o bien la cuenta de Google.
También se validará los campos antes de realizar la petición de forma similar a la de la funcionalidad de registrar.

Firebase nos proporciona 2 tipos de bases de datos:
Realtime Database y Cloud Firestore son bases de datos en la nube que se sincronizan a tiempo real pudiendo recrear el Chat. Ambos son muy similares en cuanto al funcionamiento y la implementación en código.

¿Cuál escoger?

Cloud Firestore		
Datos almacenados	1 GiB en total	USD 0.18 por GiB
Salida de red	10 GiB por mes	Google Cloud pricing
Operaciones de escritura de documentos	20,000/día	USD 0.18 por cada 100,000
Operaciones de lectura de documentos	50,000/día	USD 0.06 por cada 100,000
Operaciones de eliminación de documentos	20,000/día	USD 0.02 por cada 100,000

ilustración 45, Precio de Cloud Firestore.

Realtime Database		
Conexiones simultáneas	100	200.000 por base de datos
GB almacenados	1 GB	USD 5 por GB
GB descargados	10 GB/mes	USD 1/GB
Varias bases de dato por proyecto	✕	✓

ilustración 46, Precio de Realtime Database.

Los anteriores precios corresponden al plan Spark(gratis) en la parte izquierda y el plan Blaze(pago) en la parte derecha. Podemos observar que Cloud Firestore se cobra principalmente por las peticiones que se realizan mientras que Realtime Database se cobra principalmente por el almacenamiento, no tiene limitaciones en las peticiones, pero sí en las conexiones simultáneas, pero en el plan Blaze permite conectar 200.000 dispositivos simultáneos, por lo que es más que suficiente.
Mientras que, el número de peticiones en un chat crece de forma exponencial al incrementar el número de usuarios.

En conclusión, para pocas peticiones, pero gran almacenamiento Cloud Firestore es la mejor opción. Mientras que, para muchas peticiones y poco almacenamiento, es preferible la opción Realtime Database.

En el caso de un chat, no vamos a almacenar gran cantidad de información en la nube, sin embargo, las peticiones van a ser masivas por lo que se escogerá el Realtime Database.

Prueba de que se ha escrito correctamente en los bases de datos.
Se ha comprobado que efectivamente ha escrito correctamente tanto en BBDD local como firebase.

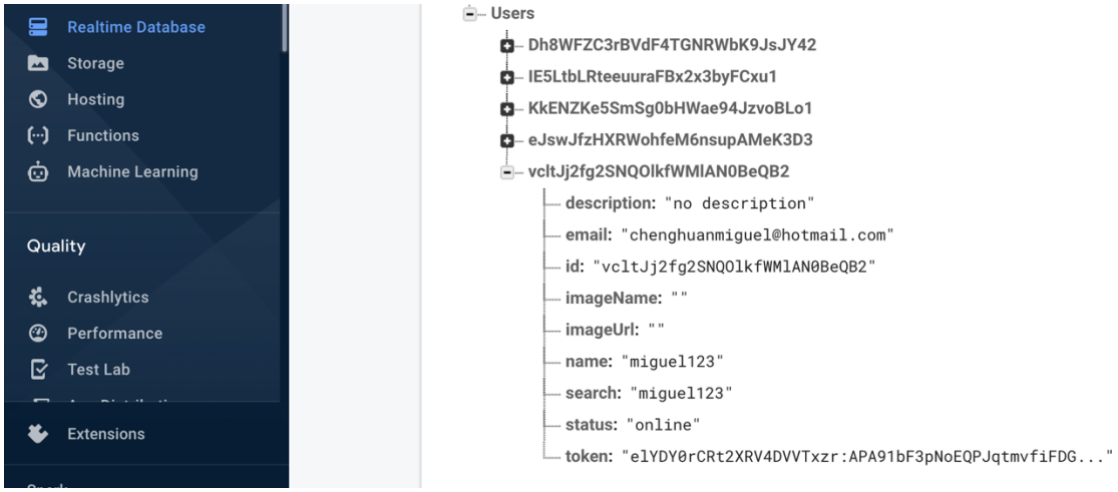

ilustración 47, prueba se correcta escritura de usuario en firebase.

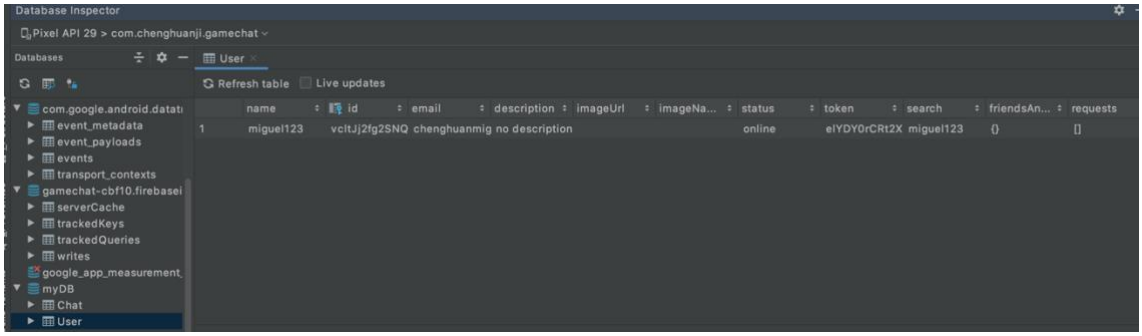

ilustración 48, prueba se correcta escritura de usuario en BBDD local.

3.2.3 Funcionalidad de restablecer contraseña

La funcionalidad está escrita en la actividad ResetPasswordActivity y los elementos gráficos en el archivo activity_reset_password.xml.

Se trata de poder restablecer la contraseña si el usuario lo ha olvidado su contraseña sabiendo su email.

En este caso, se le enviará un email para restablecer la contraseña.
Nota: el email puede estar en “Correo no deseado o spam”.

Se realizará la validación del email antes de realizar la petición.

Al realizar la petición, se nos redirigirá automáticamente a la pantalla de iniciar sesión.

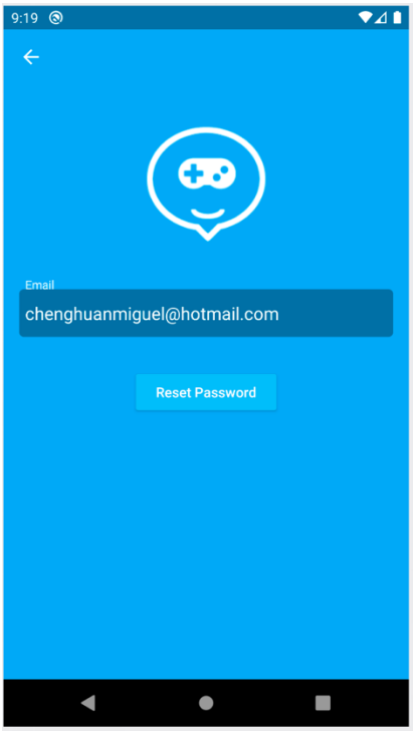

ilustración 49, pantalla de restablecer contraseña implementada.

Email de restablecimiento de contraseña recibido:

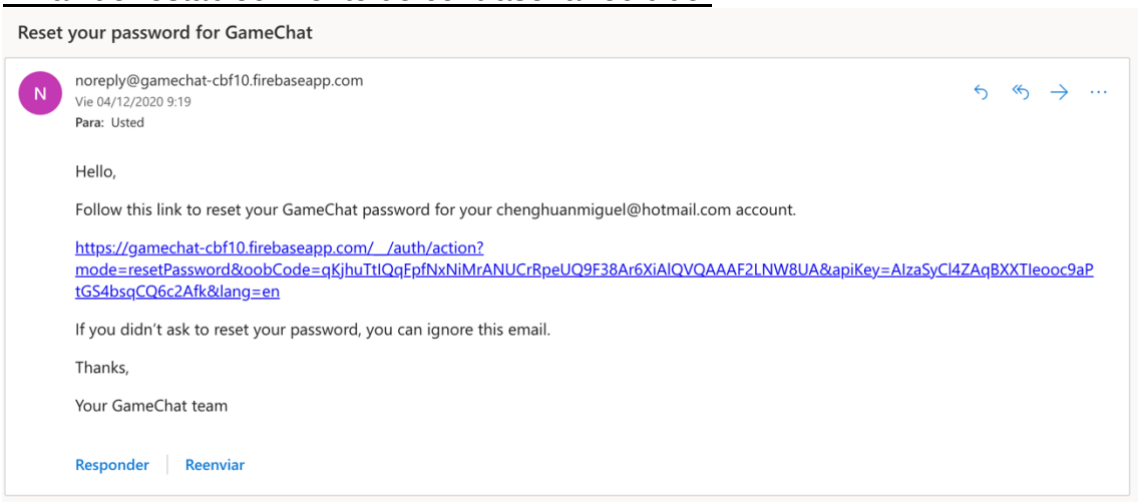

ilustración 50, email de restablecer contraseña.

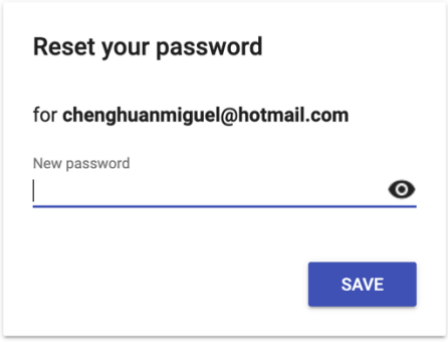

ilustración 51, restablecer contraseña.

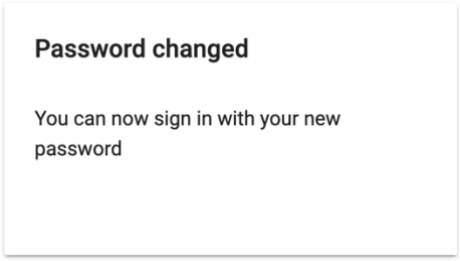

ilustración 52, contraseña cambiada.

Además de las funcionalidades anteriores se ha implementado la funcionalidad de cerrar la sesión que simplemente se cierra la sesión y se redirige a la pantalla de iniciar la sesión. Está implementada en MainActivity a través de un elemento del menú de opciones.

3.3 Funcionalidad de perfil

La funcionalidad está escrita en la actividad ProfileFragment y los elementos gráficos en el archivo fragment_profle.xml. Y el fragmento de profile forma parte de la actividad principal, MainActivity.

Se trata de poder modificar la información del perfil: la imagen, nombre de usuario y la descripción.

En principio, al registrar con email y contraseña, tiene la imagen por defecto un icono de persona y descripción: no description. Y si ha entrado con su cuenta de google sí que muestra su imagen, pero sigue sin tener descripción.

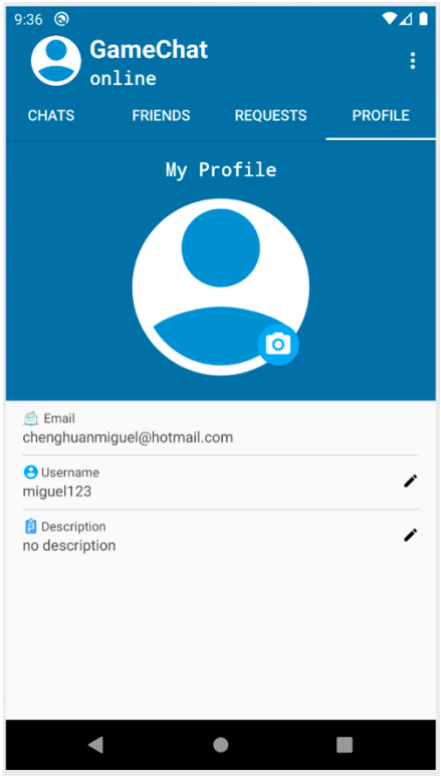

ilustración 53, pantalla de perfil implementada.

3.3.1 Estrategia para cambiar la imagen.

Recuerda que se trata de una imagen donde se va a visualizar no sólo por sí mismo, sino que también por los demás usuarios, por lo que tiene que subir la imagen a la nube.

Para ello se ha utilizado la tecnología Storage de Firebase que es un servicio de almacenamiento de objetos potente, simple y rentable construido para la escala de Google. Sirve para almacenar y entregar contenido generado por usuarios, como fotos o videos. Se utilizará en nuestra app para guardar la foto de los usuarios.

La estrategia consiste en:

1. El usuario da clic en el icono de cámara, donde se abrirá la aplicación para recoger la imagen. Por defecto se va a la carpeta Recient.

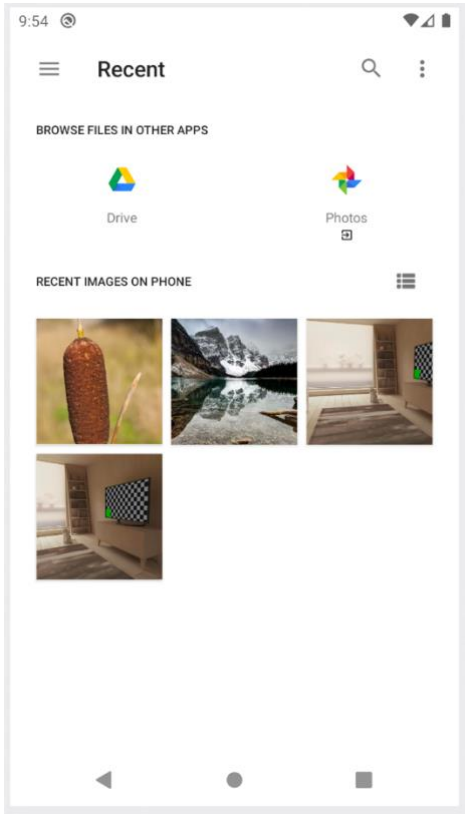

ilustración 54, recients.

2. El usuario escoge una imagen. Se vuelve automáticamente a la aplicación.
3. Con la imagen escogida, hacemos lo siguiente:
   - 3.1 sacamos el bitmap
   - 3.2 ponemos el nombre para subir como nombreDeUsuario_tiempo.ext por ejemplo miguel123_1607073829364.jpg
   - 3.3 reducimos proporcionalmente el tamaño del bitmap a como máximo 200px por lado.
4. Subimos el bitmap modificado. Y con esto obtendremos un URL de la imagen.
5. Guardamos el URL de la imagen en Firebase.
6. Borramos la imagen anterior si existe, en este caso sabes el nombre de la imagen anterior. Esto es para que cada persona sólo tenga una imagen almacenado en Storage de Firebase.
7. Actualizamos el nombre de la imagen en Firebase.

Conociendo el URL de la imagen es fácil mostrarla. Para ello se ha utilizado la herramienta Picasso.

Es obvio que la reducción de la imagen y lo de sólo permitir una imagen guardado por persona sirve para ahorrar espacio de almacenamiento, puesto que sólo disponemos hasta 5GB en total de forma gratuita en Storage de Firebase.

A continuación, se muestra el resultado de cómo pasa una imagen que pesaba más de 10MB a unos 25KB.

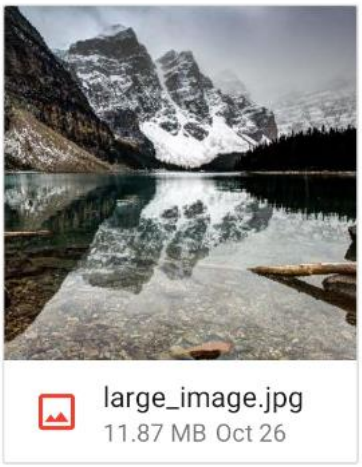

ilustración 55, imagen inicial.

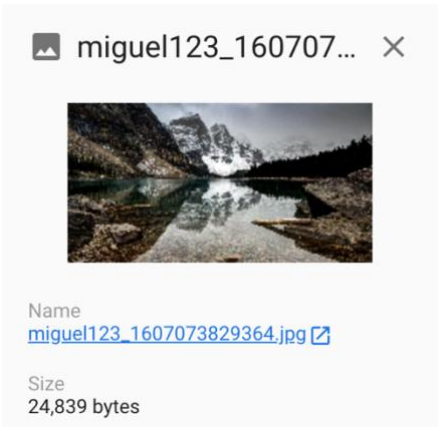

ilustración 56, imagen final subida.

3.3.2 Estrategia para cambiar el nombre de usuario o la descripción.

Para cambiar el nombre o la descripción de realiza de la manera muy similar. En estos casos simplemente de trata de modificar el nombre o la descripción tanto en BBDD local como en Firebase.

Para ello se ha creado un díalogo con un cuadro de texto para introducir el nuevo nombre o descripción.

Antes de efectuar los cambios se validará comprobando si contiene más de 6 caracteres. Y se ha añadido también un contador de caracteres limitando a 100 caracteres.

La diferencia de cambiar el nombre y la descripción es que al cambiar el nombre además de cambiar el campo nombre tiene que cambiar también el campo search que es el nombre, pero todo en minúsculas que nos servirá en el buscador.

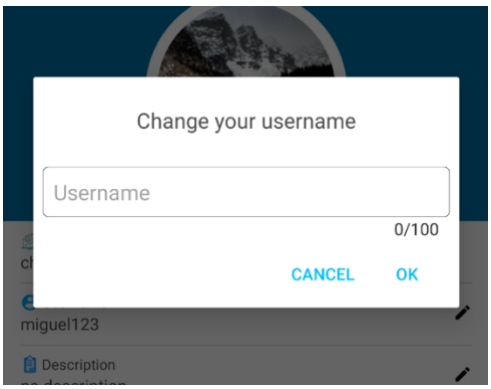

ilustración 57, dialogo cambiar nombre.

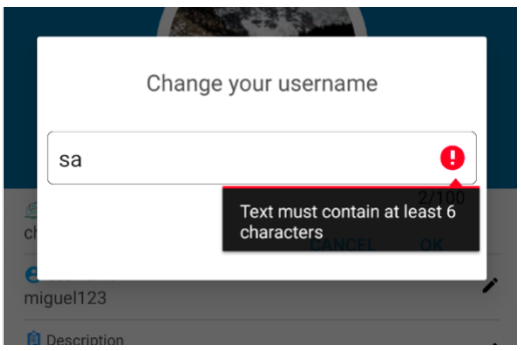

ilustración 58, dialogo cambiar nombre validación.

Y como resultado tendremos:
Como puede observar que ha realizado correctamente los cambios tanto locales como en la nube.

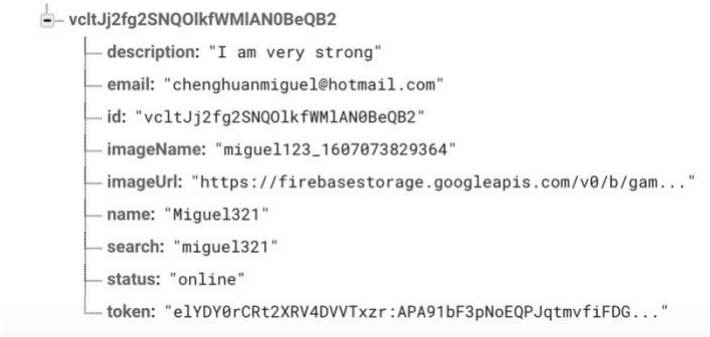

ilustración 59, firebase al cambiar el perfil.

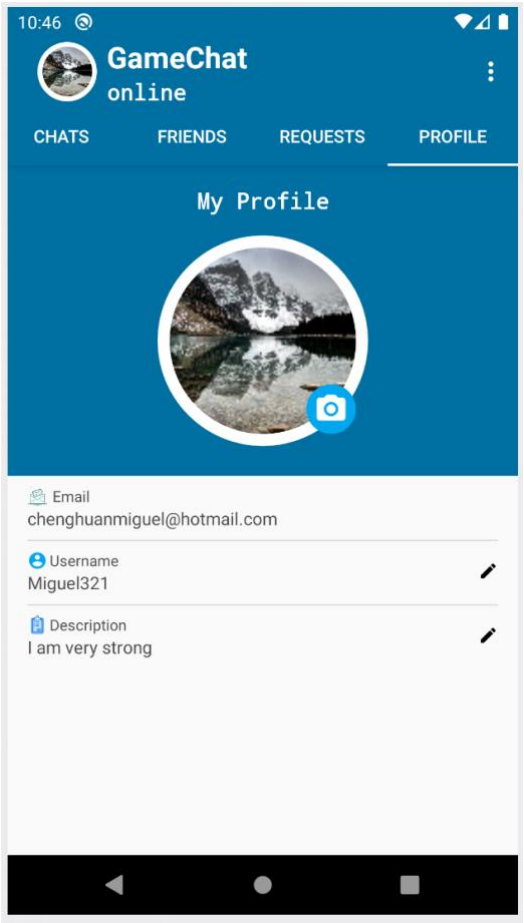

ilustración 60, pantalla de perfil implementada final.

3.4 Funcionalidad de Gestionar Amigos

Esta funcionalidad está compuesta por:

1. Buscar amigos.
2. Enviar la solicitud de amistad.
3. Mostrar solicitudes.
4. Aceptar o rechazar o cancelar la solicitud de amistad.
5. Mostrar amigos.
6. Eliminar amigos.

3.4.1 Funcionalidad de buscar amigos

Esta funcionalidad está implementada en la actividad AddFriendActivity y los elementos gráficos en el archivo activity_add_friend.xml.

Previamente se debe añadir un botón flotante en el apartado de amigos, implementado con FriendsFragment, para navegar a AddFriendActivity.

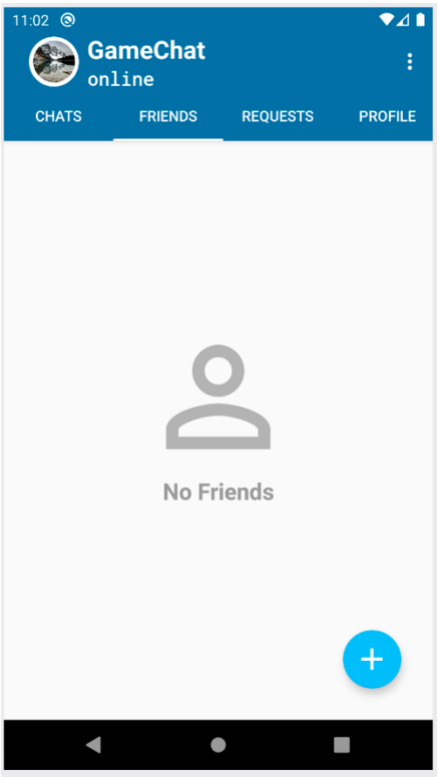

ilustración 61, pantalla principal de amigos implementada.

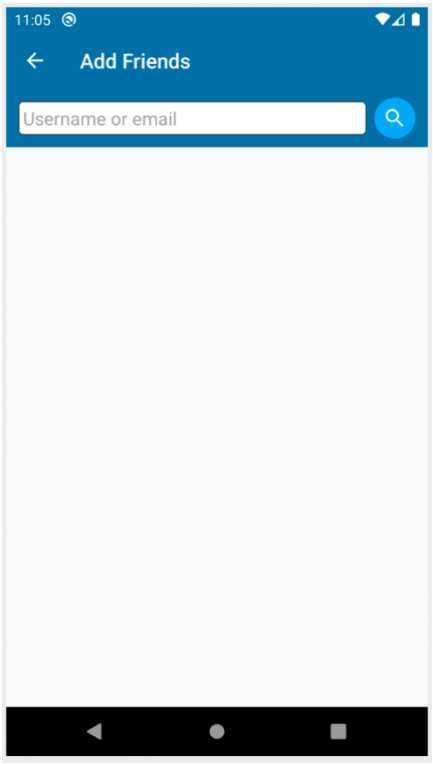

ilustración 62, pantalla de añadir amigos implementada.

Para buscar un amigo el usuario simplemente tiene que introducir el nombre de usuario o el correo a buscar. Y dar clic en el botón de lupa.

Simplemente se trata de realizar una consulta en los usuarios de firebase donde el campo search esté comprendido entre el texto introducido pasado a minúsculas y el texto introducido pasado a minúsculas + "\uf8ff" (es una forma de buscar en firebase para indicar que el texto a buscar empiece por el texto introducido). En este caso puede obtener varios resultados, lo limitamos a 50.

Se detectará automáticamente si el texto se trata de un email y se buscará por email. En este caso se obtendrá como máximo un resultado.

Se excluye del resultado el usuario mismo y sus amigos.

Resultados de búsquedas:

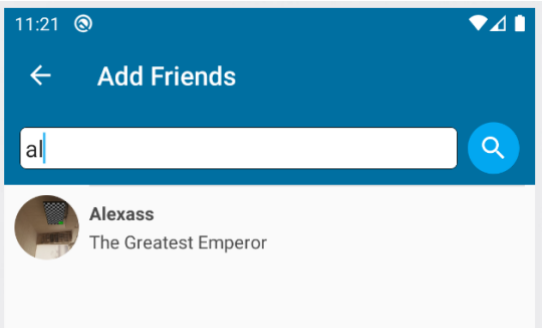

ilustración 63, buscar por nombre.

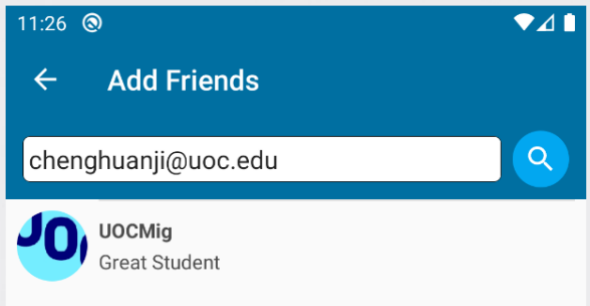

ilustración 64, buscar por email.

3.4.2 Funcionalidad de enviar la solicitud de amistad

Se trata de poder enviar las solicitudes de amistad a los usuarios encontrados. Está implementado en UserRecyclerAdapter utilizado en AddFriendActivity.

El procedimiento de solicitar amistado consiste simplemente en:
1. Al dar clic sobre un usuario encontrado, nos muestra un diálogo donde podemos escribir un mensaje. El mensaje es opcional, puede dejarlo en vacío.
2. Le damos a aceptar.
Lo podemos observar en la siguiente ilustración.

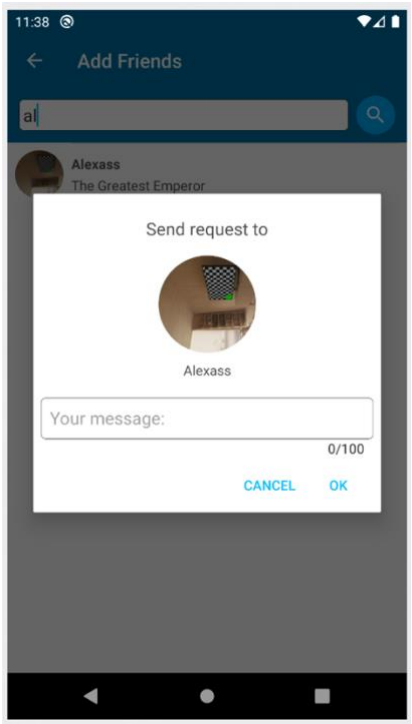

ilustración 65, dialogo de solicitud.

Para enviar la solicitud:

1. Instanciamos un objeto FriendRequest con los datos necesarios.
2. Guardamos ese FriendRequest en Firebase bajo el nombre de idDelEmisor_idDelReceptor. Lo hacemos así para poder encontrarlo rápido.
3. Añadimos en la lista de solicitudes de tanto el usuario emisor como del receptor.

Por ejemplo. Una solicitud de amistad enviado desde Miguel321 a Alexass resultaría:

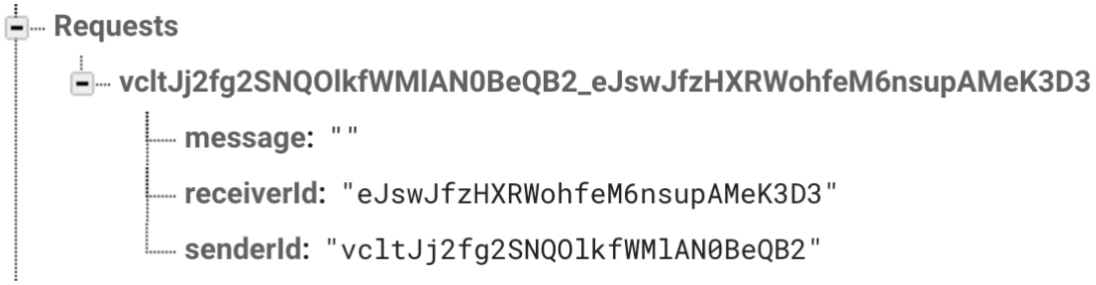

ilustración 66, la solicitud de amistad en firebase.

ilustración 67, Miguel321 como emisor.

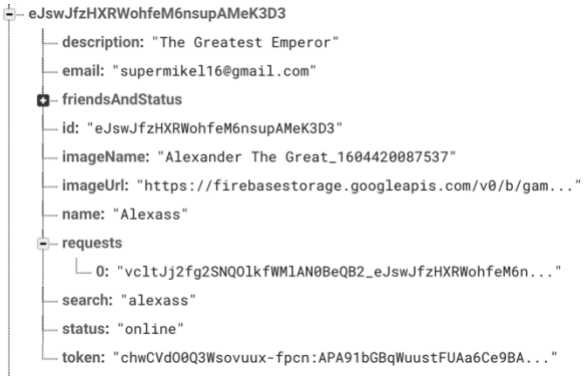

ilustración 68, Alexass como receptor.

Vemos que se ha añadido en la lista de requests de tanto Miguel321 como Alexass el nombre de la solicitud.

3.4.3 Funcionalidad de mostrar solicitudes

La nueva solicitud se muestra en la pestaña de solicitudes cuya funcionalidad se implementa en RequestsFragment que a su vez forma parte del MainActivity y las interfaces gráficas en fragment_requests.xml.

Las solicitudes se muestran de forma distinta según si es enviado o recibido.
Para el muestreo de las solicitudes se ha empleado un recyclerview dentro de otro recyclerview. Aunque también se podría haber creado dos listas independientes. Se ha implementado de esa manera por razones académicos.

¿Cómo se entera de que ha habido una solicitud nueva o se ha eliminado la solicitud?

0. En la clase User tiene una lista de requests que agrupan los nombres de las solicitudes.
1. Se ha creado usuario de tipo MutableLiveData<User> en MyApplication.
2. Tiene un “listener” que escucha los cambios producidos del propio usuario en Firebase en la clase MainActivity y modifica al valor de ese MutableLiveData.
3. Y en la pestaña de solicitudes tiene un observer que escucha los cambios de ese MutableLiveData. Así al haber un cambio en el usuario, se refrescará lista de solicitudes.
4. No hay que olvidar de parar el observer si de destruyen las vistas.

Se ha programado de esta manera porque así sólo necesitamos un listener escuchando al usuario en Firebase para todas las pestañas en vez de un listener para cada pestaña.
De esta manera haremos menos peticiones a Firebase con lo cual disminuirá el coste y mejorará el rendimiento.

Para mostrar la solicitud:

1. Se busca si tiene algún elemento en su lista de solicitudes.
2. En caso positivo, se buscará la información de cada petición en firebase recogiendo el mensaje.
3. Para cada petición se buscará la información del otro usuario en firebase: nombre e imagen.
4. Sabiendo el id, el nombre, la imagen y el mensaje del otro usuario se instancia los RequestUsers para cada petición. Y con estos RequestUsers se construye las dos listas. Según si se trata de emisor o receptor se meterá a una lista u otra.

También se ha implementado el (Número De Solicitudes) en el título de la pestaña si el número es mayor que 0. Esto se realiza en MainActivity.

Resultado:

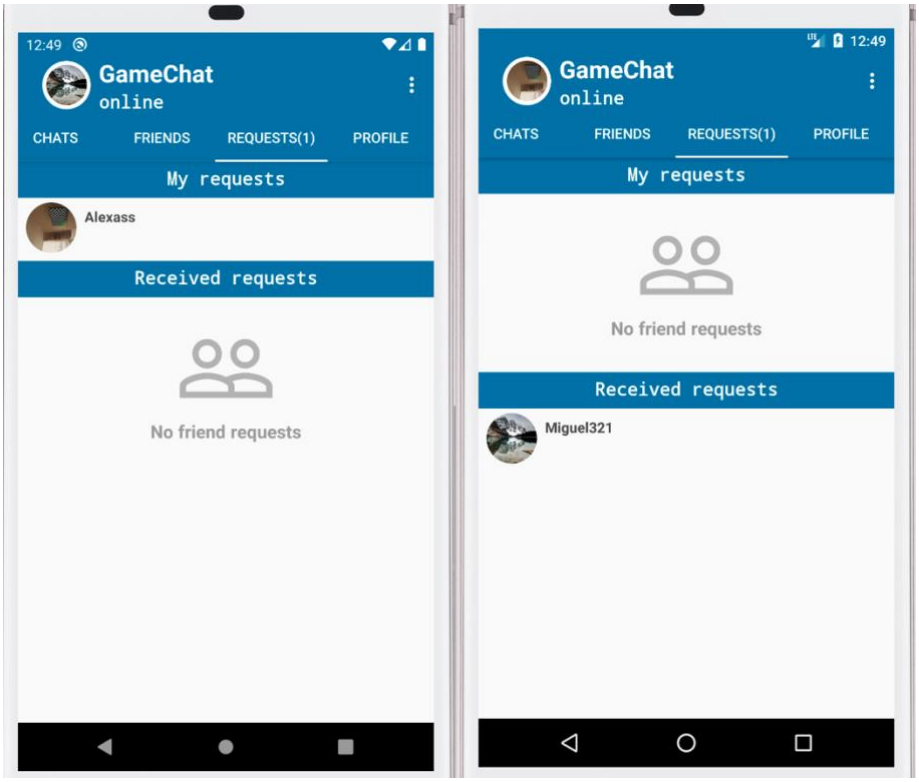

ilustración 69, demostración de las solicitudes.

3.4.4 Funcionalidades de aceptar, rechazar y cancelar la solicitud de amistad

Son implementadas en el RequestSectionItemsRecyclerAdapter que está implementada en RequestSectionRecyclerAdapter y ésta a su vez está implementada en el RequestsFragment.

Se podrá aceptar o rechazar la solicitud si el usuario es el receptor. Y se podrá cancelar la solicitud si el usuario es el emisor.

Se mostrará un diálogo u otro dependiendo si es emisor o receptor.

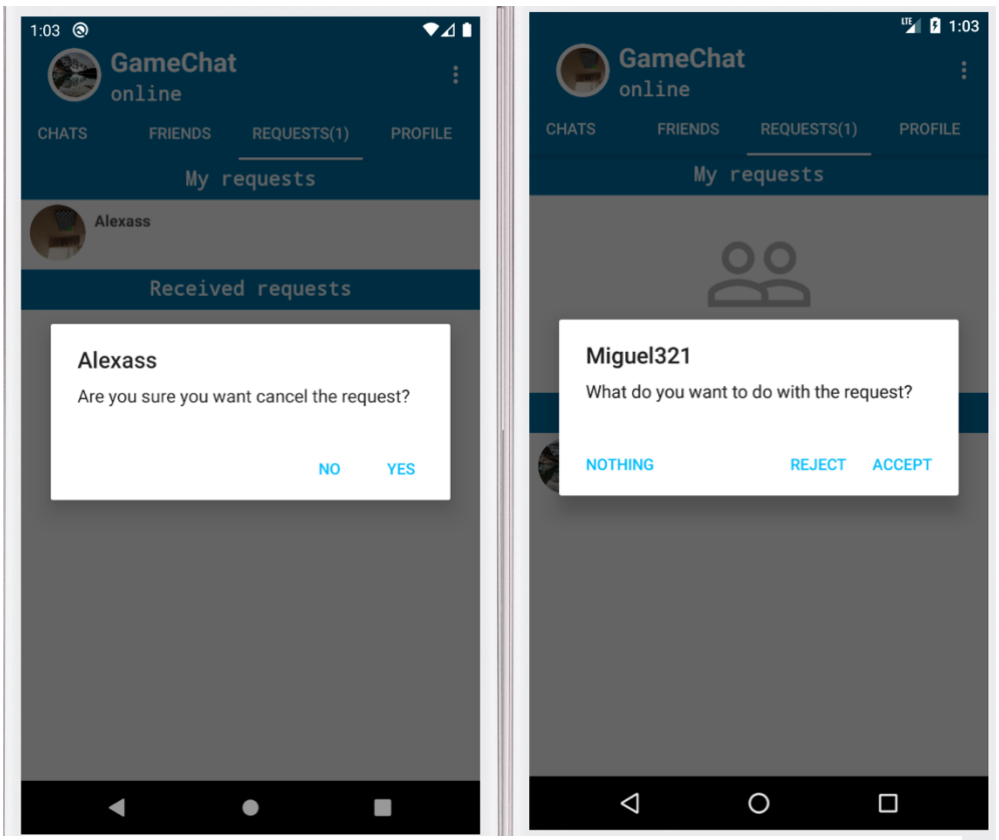

ilustración 70, los diálogos para manejar las solicitudes.

Para cancelar la solicitud:
Es la más sencilla de los tres, simplemente tiene que borrar:
1. La solicitud en Firebase.
2. El nombre de solicitud en mi lista de solicitudes en Firebase y local.
3. El nombre de solicitud en la lista de solicitudes del otro usuario en Firebase.
Previamente tienes se sacar la información del otro usuario en Firebase.

Para rechazar la solicitud:
Es casi lo mismo que el anterior, pero recordando que ahora el usuario es el receptor y el otro usuario es el emisor.

Para aceptar la solicitud:
Es como el anterior, pero añadiendo el otro usuario al mapa de friendsAndStatus. FriedsAndStatus es un diccionario con claves igual a los ids de los amigos y los valores igual a los estados conectado y desconectado. Se utilizará para saber si un amigo está conectado o no de forma más eficiente.

3.4.5 Funcionalidades de mostrar amigos

Esta funcionalidad está implementada en el FriendsFragment que a su vez forma parte del MainActivity y las interfaces gráficas en fragment_friends.xml.

Se muestran la lista de amigos con un único recyclerview con los caracteres del nombre de usuario, la descripción, la imagen y el estado de conexión.

Para mostrar amigos:

1. Previamente, recordamos que teníamos creado un MutableLiveData<User> en MyApplication. Se escucha en FriendsFragment a través de un observer los cambios de ese MutableLiveData.
2. Conociendo los datos de nuestro usuario, sabemos la información de FriedsAndStatus. Para cada amigo buscamos su información en Firebase.
3. Una vez recogida las informaciones de los amigos, se crean la lista de Users necesaria para mostrar en el recyclerview.
4. Ordenamos la lista primero según el estado de conexión y luego según el orden alfabético del nombre de usuario.
5. Mostrar la lista en el recyclerview a través del adaptador UserRecyclerAdapter.
6. No hay que olvidar de parar el observer si se destruyen las vistas.

Como resultado tendremos:

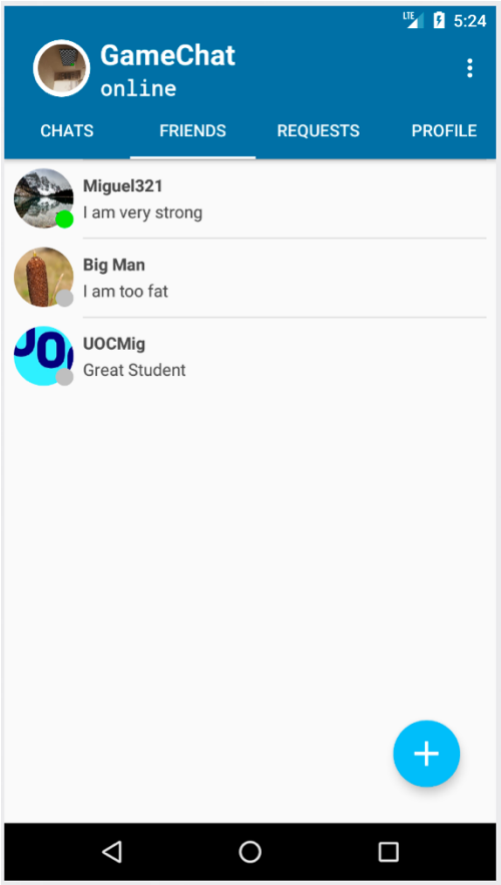

ilustración 71, pantalla principal de amigos implementada.

3.4.6 Funcionalidades de eliminar amigos

Esta funcionalidad está implementada en el FriendsFragment que a su vez forma parte del MainActivity y las interfaces gráficas en fragment_friends.xml.

Para eliminar un amigo, el usuario tiene que hacer:

1. Hacer un clic largo sobre un amigo. Nos aparecerá un menú contextual y el amigo a eliminar sombreada.
2. Hacer un clic sobre la opción de Delete Friend. Nos aparecerá un diálogo preguntándonos si realmente queremos eliminarlo.
3. Le da que sí.

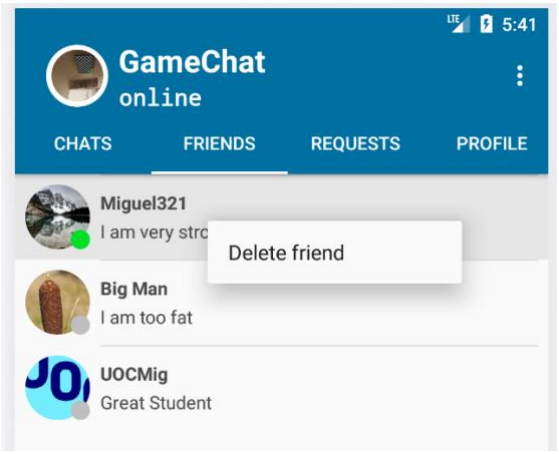

ilustración 72, menú contextual para eliminar amigo.

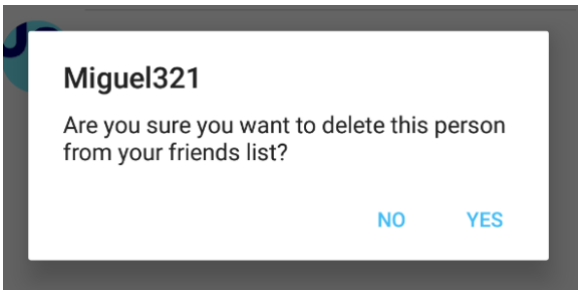

ilustración 73, diálogo para eliminar amigo.

Para sombrear el amigo simplemente se ha cambiado el color de fondo del elemento. Y si con el menú contextual abierto se da a otro lugar de la pantalla se cerrará el menú contextual y se quitará el sombreado.

Estrategia para eliminar un amigo:

Simplemente hay que eliminar el amigo de la lista de friendsAndStatus del usuario tanto local como en la nube. Y también eliminar el usuario de la lista de friendsAndStatus del amigo en la nube.

Por ejemplo, si Alexass elimina Miguel321 como amigo. Se eliminará Miguel321 de la lista de friendsAndStatus de Alexass y se eliminará Alexass de la lista de friendsAndStatus de Miguel321.

Se ha decidido hacer de esta manera ya que supuestamente que el amigo es recíproco, si Alexass no es amigo de Miguel321, entonces Miguel321 no puede ser amigo de Alexass.

3.5 Funcionalidad de Chat

Esta funcionalidad está compuesta por:
1. Tratamiento de mensajes.
2. Mostrar chats.
3. Eliminar chats.

3.5.1 Funcionalidad de tratamiento de mensajes

Esta funcionalidad está implementada en ChatActivity y las interfaces gráficas en activity_chat.xml.

Para acceder al ChatActivity puede hacerlo o bien dando clic sobre un amigo de la pestaña de amigos o bien clic sobre un chat de la pestaña de chats.

Es importante entender el esquema de cómo se trata un mensaje antes de pasar a definir las funcionalidades compuestas. De hecho, esta parte es la parte fundamental de la aplicación.

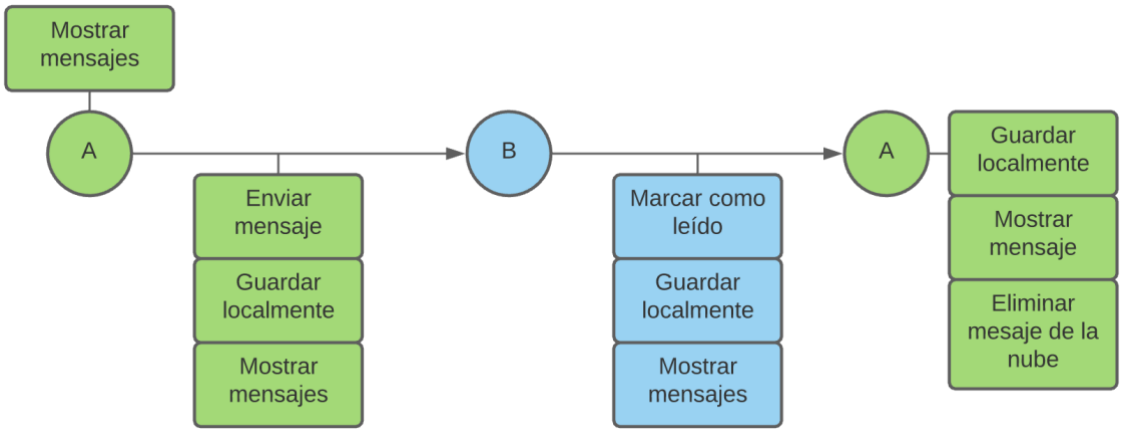

ilustración 74, tratamiento de mensajes.

Nada más de entrar en ChatActivity, se cargará los mensajes guardados localmente y se mostrarán.

Si el usuario A envía un mensaje a B. El A realizará:
1. Un envío de mensaje
2. El guardado local del dicho mensaje.
3. Presentación del dicho mensaje en la pantalla.

Si usuario B recibe un mensaje de A. El B realizará:
1. Marcado del dicho mensaje como leído
2. El guardado local del dicho mensaje.
3. Presentación del dicho mensaje en la pantalla.

Para poder realizar estas acciones, se necesita añadir un listener que escucha a los mensajes recibidos (que son enviados por su amigo).

Si el usuario A recibe el mensaje de enviado por él mismo y previamente leído por B. El A realizará:
1. El guardado local del dicho mensaje.

2. Presentación del dicho mensaje en la pantalla.
3. Eliminación del dicho mensaje en la nube.
Para poder realizar estas acciones, se necesita añadir un listener que escucha a los mensajes enviados por sí mismo.

Por lo que podemos descomponer esta funcionalidad en:
1. Enviar mensajes.
2. Marcar el mensaje como leído.
3. Eliminar el mensaje leído de la nube.
4. Mostrar mensajes.

3.5.1.1 Funcionalidad de enviar mensajes

Una vez estando en la pantalla de ChatActivity, para enviar un mensaje simplemente se escribe un mensaje sobre el cuadro de abajo y se da al icono de enviar.

Estrategia de enviar un mensaje (y guardarlo localmente):
1. Se instancia un objeto Chat con los datos necesarios. Donde el id del chat es idEmisor_idReceptor_Tiempo (Date().time) que será también el nombre del chat con lo que vamos a guardar en Firebase. Y poniendo en principio sentAt Date().time que será modificado posteriormente.

2. Se guarda el dicho Chat el BBDD local con Room. El sentAt será modificado posteriormente cuando tengamos el tiempo firebase.

3. Se guarda el Chat en Firebase con una referencia: Chats/idReceptor/idEmisor de esta manera vamos a poder recuperarla luego de forma más eficiente. Aquí el sentAt será ServerValue.TIMESTAMP.

¿Por qué ServerValue.TIMESTAMP?
Recordemos que para instanciar un chat se necesita: id, senderId, receiverId, message, isSeen y sentAt.

Si usamos el sentAt el Date().time que es el tiempo del sistema en Long, luego vamos a tener un problema a la hora de ordenar los chats puesto que los usuarios pueden estar en diferentes zonas horarias o tener tiempos diferentes.

Para solventar este problema, se utilizará el tiempo que el firebase recibe. Este tiempo será común para todos, y lo pondrá firebase. Para ello se pone sentAt = ServerValue.TIMESTAMP.

Como resultado:

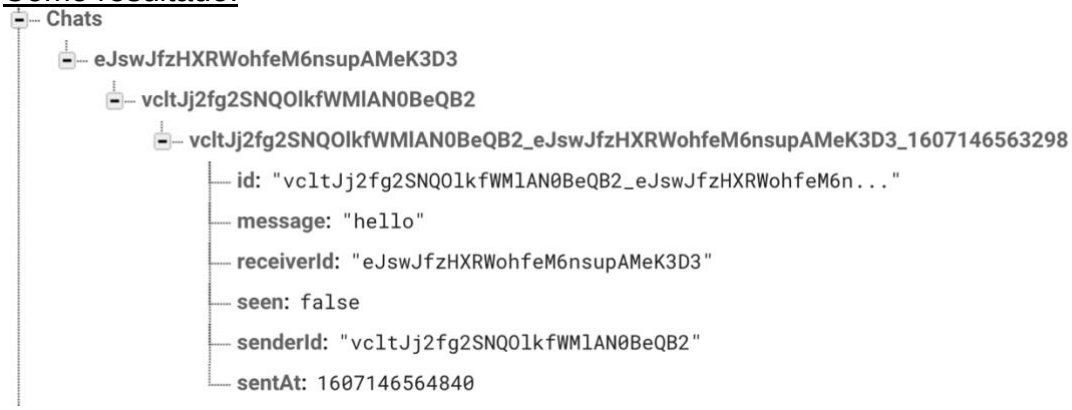

ilustración 75, chat en firebase.

3.5.1.2 Funcionalidad de marcar el mensaje como leído

Imaginase que somos el usuario B. Para marcar el mensaje enviado por A como leído se necesita añadir un listener en ChatActivity para escuchar los mensajes enviados de A a B.

Una vez que recogemos estos mensajes, simplemente cambiamos el isSeen = true, en firebase lo muestra como “seen”. Y lo volvemos a guardar en el mismo sitio en Firebase.

También aprovechamos para guardarlo localmente con Room este mensaje.

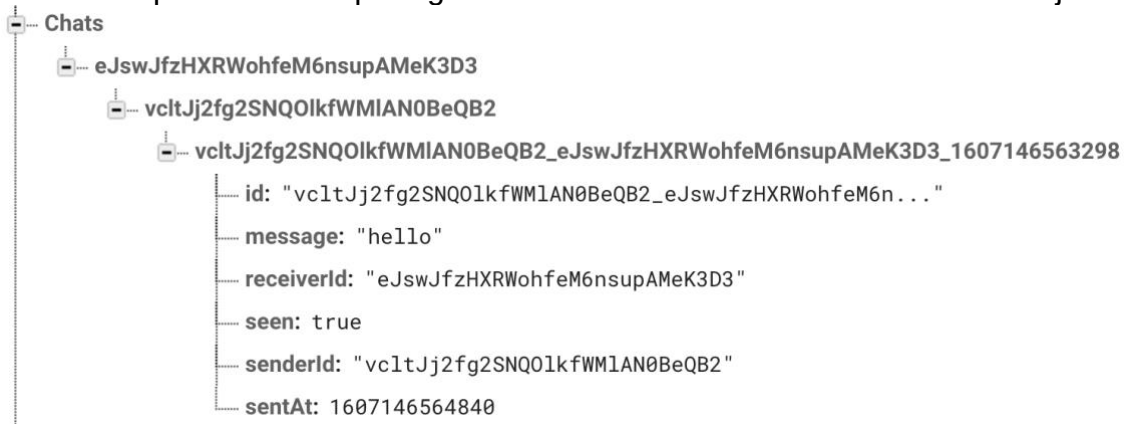

ilustración 76, chat leído en firebase.

3.5.1.3 Funcionalidad de eliminar el mensaje leído de la nube

¿Por qué eliminar el mensaje leído?
Como el firebase nos puede cobrar por la cantidad de datos descargados y almacenados, necesitamos eliminar todo lo que no es necesario para el funcionamiento de la aplicación. Así se descargaría menos datos al recoger estos datos de firebase.

Como mencionado anteriormente, para esta funcionalidad necesitamos añadir un listener en ChatActivity que escucha los mensajes enviados por sí mismo.

Previamente nos aprovechamos para guardar localmente estos mensajes y mostrarlas en la pantalla.

La estrategia de eliminar es sencilla. Se recoge estos mensajes, se comprueban sí están efectivamente leídos, y en caso positivo los eliminan de Firebase.

3.5.1.4 Funcionalidad de mostrar mensajes

Si no tienen los mensajes tendrá que cargar las previamente desde BBDD local.

Para mostrar los mensajes se ha utilizado un recyclerview.
Se ha implementado también elemento central de la fecha de envío.

Una vez tenga los mensajes (las instancias de Chat) dispuestas para mostrar se realiza:
0. Tener la lista de Chats ordenada según sentAt.
1. Se pasa la lista de Chats a la lista de Chats con header donde el header es la fecha de envío, un String.
2. Se muestra utilizando el ChatSectionRecyclerAdapter donde nos diferenciamos los elementos de 3 tipos.
    1. Los elementos de izquierda: son los mensajes enviados por el amigo.
    Podemos reconocerlo puesto que el senderId del Chat no es el Id del usuario.
    2. Los elementos de derecha: son los mensajes enviados por el usuario.
    Podemos reconocerlo puesto que el senderId del Chat es el Id del usuario.
    3. Los elementos centrales: son las fechas de envío.
    Podemos reconocerlo puesto que las fechas de envío es un String.

Resultados:

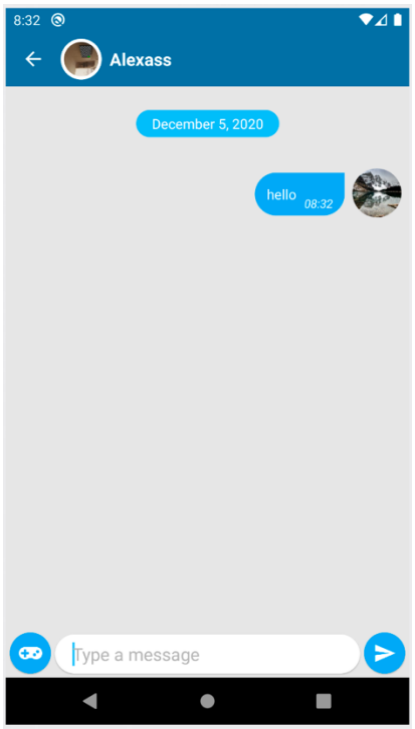

ilustración 77, mensaje no leído.

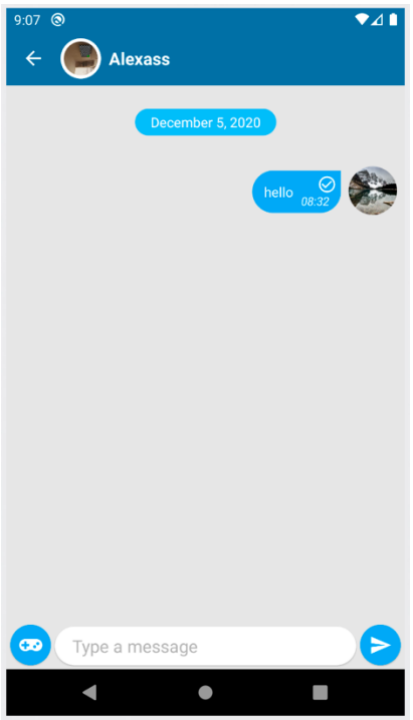

ilustración 78, mensaje leído.

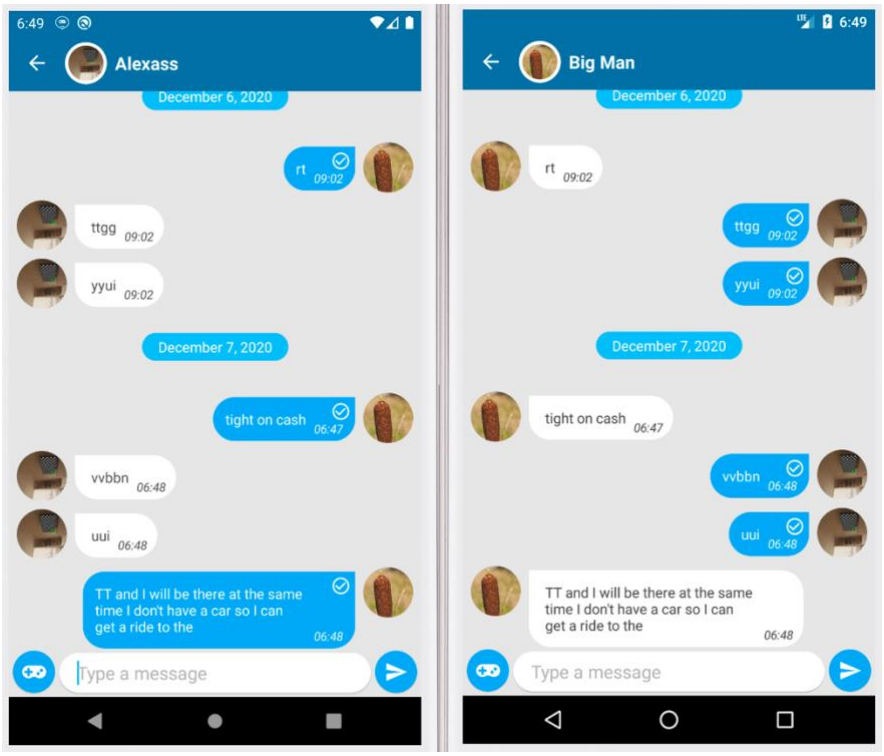

ilustración 79, Pantalla de chats implementada.

Se ha implementado también el “scroll indefinido”.
Como saben, si tenemos el gran número de mensajes, al cargar todos a la vez tardaría mucho tiempo, esto no es preferible desde el punto de vista de la experiencia del usuario.

Para solventar este problema, sólo se por defecto los últimos 50 mensajes. Y al hacer el scroll para arriba se cargará más mensajes.

3.5.2 Funcionalidad de mostrar chats con distintos amigos

Esta funcionalidad está implementada en ChatsFragment que a su vez pertenece al MainActivity y las interfaces gráficas en fragment_chats.xml.

Para sacar los distintos chats se debe sacar tanto de los locales como los de la nube.
Para los locales no es necesario un listener, pero para los de la nube se añade un listener que escucha a los chats recibidos.

Por defecto si un amigo no nos ha enviado ningún mensaje, en vez de mostrar el último mensaje se mostrará la descripción.

Se muestra también la cantidad de mensajes no leídos de cada chat en la parte derecha superior del elemento y en total en el título, si el número es mayor a 0, esta última característica se implementó en MainActivity.

Se muestran los chats a través de una lista con recyclerview y el UserRecyclerAdapter de forma muy similar al apartado de amigos.

Se muestra el estado de conexión de los amigos, si el otro usuario todavía es su amigo. Es decir, si tienes el chat, pero le has eliminado de tus amigos entonces siempre va a mostrar como desconectado.

Resultado:

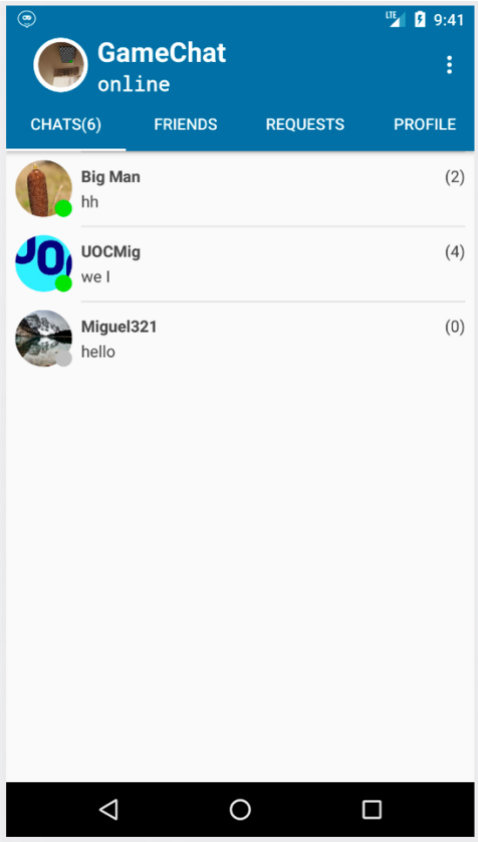

ilustración 80, Pantalla principal de chats implementada.

3.5.3 Funcionalidad de eliminar los chats

Esta funcionalidad está implementada en ChatsFragment que a su vez pertenece al MainActivity y las interfaces gráficas en fragment_chats.xml.

Para eliminar un chat, el usuario tiene que hacer:

1. Hacer clic largo sobre un amigo. Nos aparecerá un menú contextual y el chat a eliminar sombreada.
2. Hacer un clic sobre la opción de Delete Chat. Nos aparecerá un diálogo preguntándonos si realmente queremos eliminarlo
3. Le da que sí.

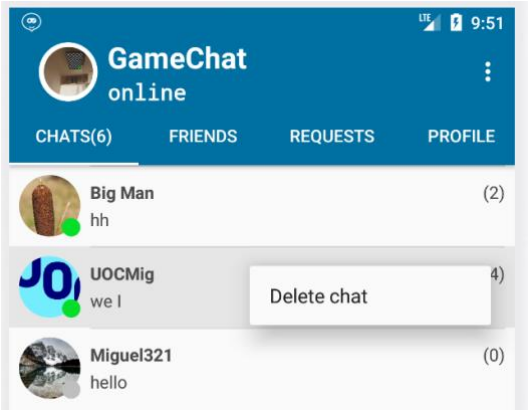

ilustración 81, menú contextual de eliminar chat.

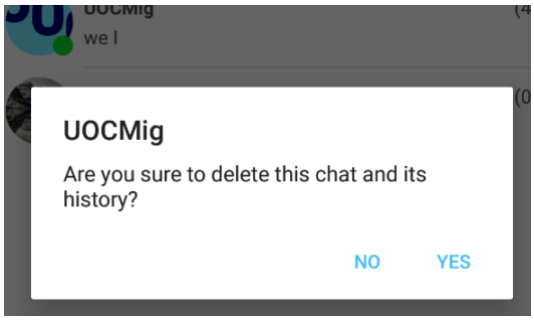

ilustración 82, diálogo de eliminar chat.

Como vemos que es muy similar a los pasos de eliminar un amigo.

La estrategia para eliminar un chat:

1. Se elimina todos los mensajes de ese amigo en Firebase si existen.
2. Se elimina todos los mensajes de ese amigo localmento con Room.
3. Se repinta la lista de recyclerview.

Hay que mencionar que, al hacer de esta manera, se eliminará también los mensajes no leídos de ese amigo si hay. Por lo que estos mensajes no van a ser leídos nunca.

3.6 Funcionalidad del estado de conexión

El cambio de estado está implementado principalmente en MyAppliation aunque también hay parte implementada en MainActivity y ChatActivity.

En resumen, el usuario esta online si cumple estas 4 condiciones:
1. Aplicación abierta corriendo en primer plano.
2. Sesión iniciada.
3. Con conexión a internet.
4. Con “show my status” marcada.

Entonces cada vez que hay algún cambio de las condiciones anteriores se debe cambiar el estado de conexión.

Para saber si tiene conexión a internet se ha comprobado de 2 maneras: uno comprobando si tiene alguna capacidad de red como wifi o internet móvil. Otro comprobando utilizando el api de Firebase.

¿Por qué lo comprobamos con 2 maneras?
Porque con sólo uno no es suficiente. La primera manera sólo se comprueba si tiene capacidad, pero no comprueba si en ese wifi tiene realmente internet, es posible conectarse a un wifi, pero ese wifi no tiene acceso a internet.
Y la segunda manera tiene una alta latencia.
Y también saber el si tiene acceso a internet nos sirve para bloquear algunas acciones no permitidas si no tiene internet. Por ejemplo, si no tiene internet no se permitirá el cambio del perfil, eliminar amigos…

Para detectar si la aplicación se va al background o foreground se ha implementado unos métodos en MyApplication que detectan estos eventos.

El show my status es una opción que se le da al usuario en el menú de opciones del MainActivity. Se guarda la opción elegida en sharedPreferences. Sirve para dejar que el usuario decidir si quiere mostrar o no su estado.

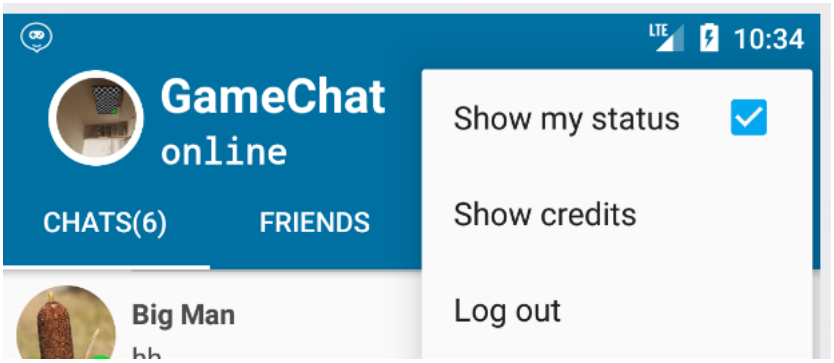

ilustración 83, menú de opciones de MainActivity.

3.7 Funcionalidad de las notificaciones

Para la implementación de las notificaciones se ha utilizado la tecnología de Cloud Messaging de Firebase que es una solución de mensajería multiplataforma que te permite enviar mensajes de forma segura y gratuita.

Se enviará una notificación cada vez de se envía un mensaje, pero sólo se mostrará la notificación si no estamos en ChatActivity chateando con la persona que nos envía el mensaje.

Para enviar una notificación se utiliza la herramienta Retrofit para hacer las peticiones REST, en este caso con el método Post, enviando objeto Send a la dirección: https://fcm.googleapis.com/fcm/send

3.7.1 Funcionalidad de mostrar notificaciones

Está implementada en MyFirebaseMessagingService utilizando el bloque de notifications de models.

Se ha implementado la agrupación de las notificaciones si tiene varias notificaciones mostradas utilizando el GroupSummary.

Se ha implementado la autocancelación del GroupSummary si no tiene hijos.

Se ha implementado la redirección al ChatActivity si hace un clic sobre una notificación.

<u>Resultado:</u>

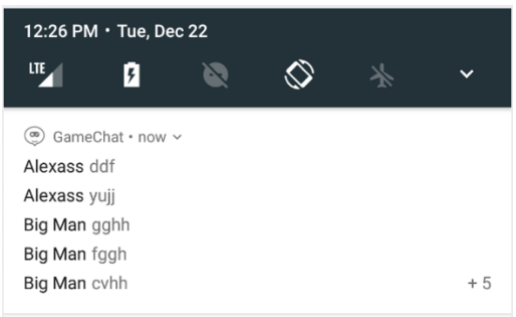

ilustración 84,
notificaciones agrupadas.

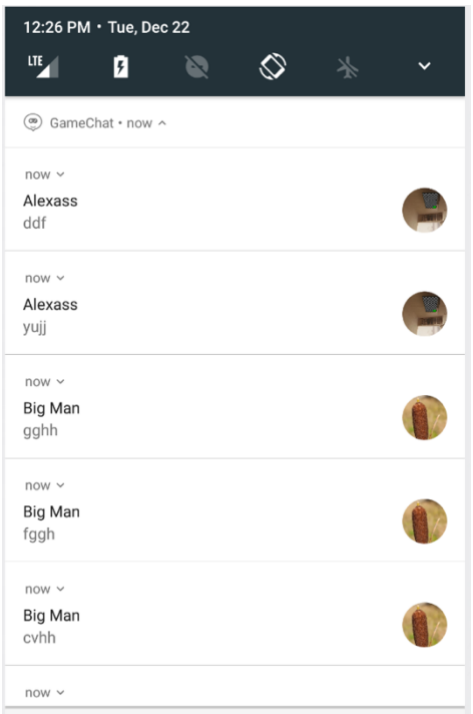

ilustración 85,
notificaciones con grupo
expandido.

3.7.2 Funcionalidad de marcar como leído en las notificaciones

Se ha implementado también un botón para marcar como leído un mensaje. Esta funcionalidad se implementa utilizando el NotificationActionReceiver que es un broadcast receiver dedicado a recibir los eventos de las notificaciones.

Cuando el NotificationActionReceiver recibe el evento, manda al NotifiationActionService que realice el trabajo de marcar el mensaje como leído.

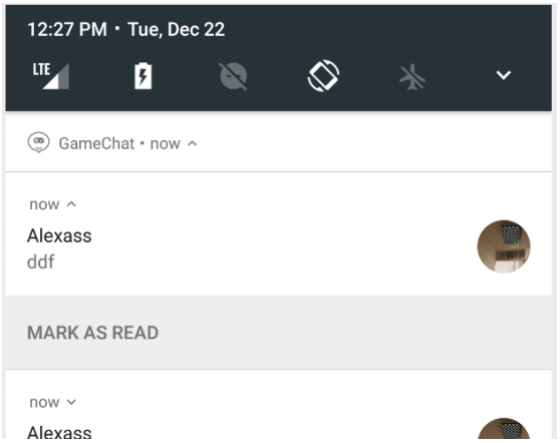

ilustración 86, notificación mark as read.

Se ha comprobado que el botón funciona correctamente tanto en emuladores como dispositivos reales.

3.8 Tic Tac Toe

Se ha escogido este juego para el trabajo debido a su sencillez y su gran reconocimiento por todo el mundo.
Se puede descomponer en:
1. Invitar a jugar.
2. Recibir la invitación.
3. Cancelar y aceptar o rechazar la invitación.
4. El juego.

Una de las limitaciones de esta aplicación es que, para poder recibir la invitación, el usuario debe situarse en la pantalla de Chat con la persona con quien desea jugar.

Por lo que antes de invitar a jugar a un amigo, se recomienda charlar con él previamente.

Los apartados 1 a 3 están implementados en ChatActivity y las interfaces gráficas en activity_chat.xml. Y el apartado de Juego está implementado en GameActivity con las interfaces gráficas en activity_game.xml

3.8.1 Funcionalidad de invitar a jugar

Estando en la pantalla de Chat para invitar a jugar a su amigo el usuario se debe hacer:
1. Hacer clic sobre el icono de consola que está situado por la esquina inferior izquierda. Aparecerá un menú flotante (pop up menu) con la opción de Tic Tac Toe.
2. Hacer clic en la opción de Tic Tac Toe. Aparecerá el diálogo preguntando si realimente quiere invitar a jugar.
3. Le da que sí. Automáticamente aparecerá un diálogo de estar esperando, con la opción de cancelar la invitación.

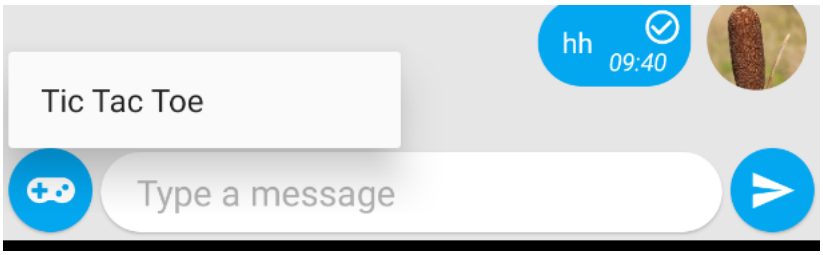

ilustración 87, menú tic tac toe.

Se ha añadido en forma de menú por si en el futuro quiera añadir otro juego.

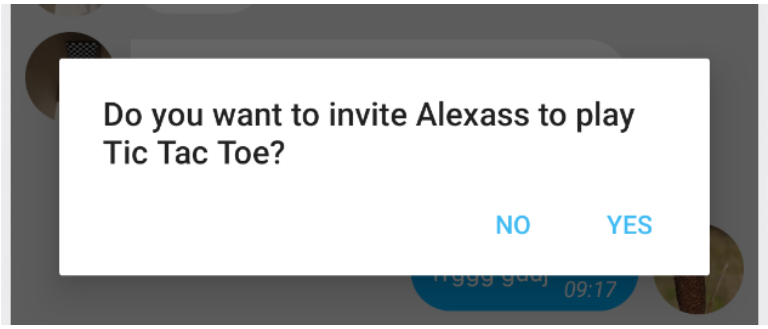

ilustración 88, diálogo de invitar a jugar.

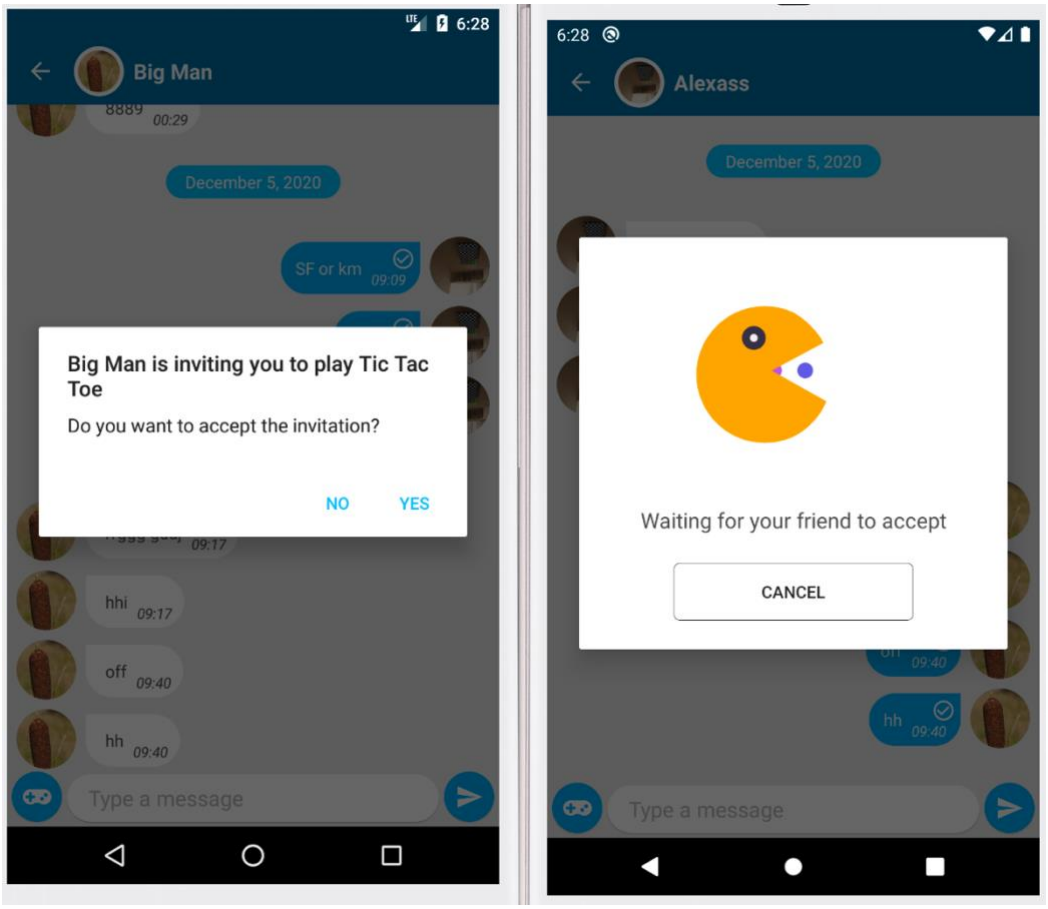

ilustración 89, diálogo de acciones con la invitación.

Estrategia para enviar la invitación:

1. Se instancia un objeto de tipo PlayTicTacToe.

2. Se sube a Firebase dicho objeto instanciado con la referencia: TicTacToe/idReceptor/idEmisor donde idEmisor será lo mismo que el id del usuario que realiza la invitación y también será lo mismo que el id del jugador1. Y el idReceptor será el id del amigo.

3. Se añadirá un listener para escuchar a la invitación enviado. Esto nos servirá para poder enterarse de la respuesta de invitación. Recordar de remover el dicho listener cuando no lo necesite, cuando ha cancelado, aceptado o rechazado la invitación.

ilustración 90, firebase, invitación nueva.

3.8.2 Funcionalidad de recibir la invitación

Como mencionado anteriormente, para poder recibir la invitación el usuario debe estar en el ChatActivity con la persona con quien desea jugar.

Para poder recibir la invitación, se añade un listener de Firebase en ChatActivity que escucha todas las invitaciones dirigidas al usuario. Es decir, con referencia TicTacToe/idUsuario/idAmigo.

Cuando haya una invitación nueva hacer que automáticamente aparezca el diálogo para aceptar o rechazar la invitación.

3.8.3. Funcionalidad de cancelar, aceptar y rechazar la invitación.

Se podrá cancelar la invitación si es el usuario emisor, y se podrá aceptar o rechazar la invitación si es el usuario receptor.

Si se cancela el diálogo el efecto es lo mismo que cancelar la invitación o rechazar la invitación dependiendo si se trata de emisor o receptor.

Estrategia para cancelar la invitación:

1. Eliminar el objeto PlayTicTacToe del Firebase.
2. Cerrar el diálogo de espera.

Estrategia para rechazar la invitación:

0. Previamente se debe comprobar que la invitación no ha sido cancelada para continuar. En caso de que la invitación ya ha sido cancelada simplemente se cierra el diálogo de aceptar o rechazar la invitación.

1. El usuario receptor modificará el objeto PlayTicTacToe poniendo la partida está abandonada.
2. El usuario receptor se cerrará el diálogo de aceptar o rechazar la invitación.
3. El usuario emisor se enterará de que la partida esta abandonada. Recordemos que tiene un listener que escucha las invitaciones enviadas. Entonces se eliminará la invitación del Firebase.
4. El usuario emisor se cerrará el diálogo de espera. Y se mostrará un mensaje con una tostada informándole que lo ha rechazado la invitación.

Estrategia para aceptar la invitación:

0. Previamente se debe comprobar que la invitación no ha sido cancelada para continuar. En caso de que la invitación ya ha sido cancelada se cierrá el diálogo de aceptar o rechazar la invitación y le mostrará el mensaje diciendo que el otro jugador puede haber cancelado la invitación mediante una tostada.

1. El usuario receptor modificará el objeto PlayTicTacToe poniendo la partida está el usuario receptor sea el jugador 2.
2. El usuario receptor se cerrará el diálogo de aceptar o rechazar la invitación. Y se navegará automáticamente a la pantalla del Juego como jugador2.
3. El usuario emisor se enterará que el jugador 2 ha aceptado la invitación. Se cerrará el diálogo de espera. Y automáticamente se navegará a la pantalla del Juego como jugador1.

3.8.4 La mecánica del Juego

Para que ambos jugadores detecten los cambios a tiempo real, se necesita implementar un listener escuchando al objeto PlayTicTacToe creada anteriormente en Firebase.

En el objeto PlayTicTacToe tiene una variable selectedCells para gestionar la mecánica del juego.

SelectedCells se trata de una lista de 9 enteros que pueden ser 0, 1 o 2.
0: si no ha sido seleccionado.
1: si ha sido seleccionado por el jugador 1.
2: si ha sido seleccionado por el jugador 2.

Cada número le corresponde a una celda del juego.

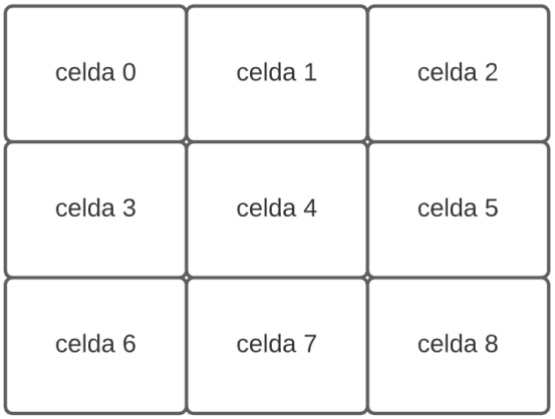

ilustración 91, la distribución de celdas.

Como podéis intuir, al comenzar la partida selectedCells vale
[0, 0, 0, 0, 0, 0, 0, 0, 0].
Si ahora el jugador 1 selecciona la casilla 4 entonces selectedCells valdrá:
[0, 0, 0, 0, 1, 0, 0, 0, 0].
Si ahora el jugador 2 selecciona la casilla 2 entonces selectedCells valdrá:
[0, 0, 2, 0, 1, 0, 0, 0, 0].
Así sucesivamente hasta que un jugador gane o hasta que se rellene todas las celdas que en este último caso tendremos un empate.

La gestión de turnos se realiza con una variable de PlayTicTacToe llamada isPlayer1Playing que será true si toca jugar el jugador 1 y false en caso contrario. Para diferenciarlo visualmente en el juego, simplemente se marcará en negrita el nombre del jugador que tenga el turno.

En cada turno se comprobará si hay ha finalizado la partida, es decir, si hay algún ganador o se ha empatado.

Para comprobar si hay algún ganador:
1. Creamos una lista de posiciones ganadoras. Que son si estas posiciones tienen todos 1 o todos 2 entonces habría ganado el jugador 1 o jugador 2 respectivamente.

En nuestro caso es:

```
arrayOf(
        intArrayOf(0, 1, 2),
        intArrayOf(3, 4, 5),
        intArrayOf(6, 7, 8),
        intArrayOf(0, 3, 6),
        intArrayOf(1, 4, 7),
        intArrayOf(2, 5, 8),
        intArrayOf(0, 4, 8),
        intArrayOf(2, 4, 6)
    )
```

2. Realizamos un bucle por esa lista comprobando si en esas posiciones en selectedCells tienen todos 1 o todos 2.

Y para comprobar si ha empatado es sencillo, simplemente comprobar que no hay ganador y que en el selectedCells no haya ningún 0.

Y dependiendo de que, si ha ganado, perdido o empatado, nos mostrará un diálogo u otro.

Para reducir el coste de Firebase, se eliminará la partida si se sale de la pantalla del Juego.

<u>Resultados:</u>

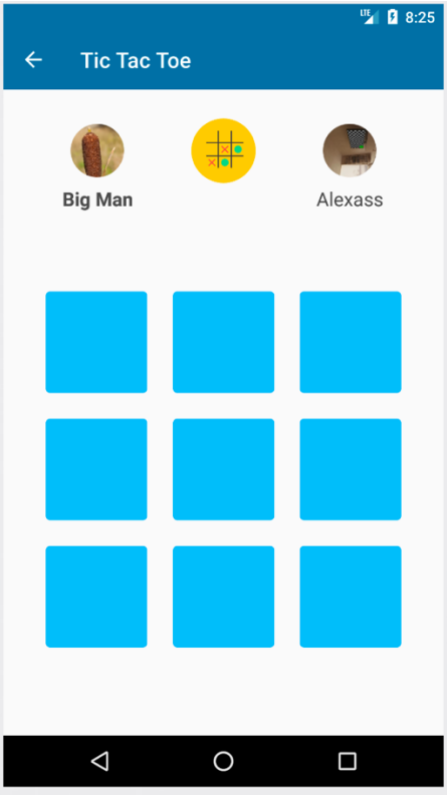

ilustración 92, pantalla de juego inicial.

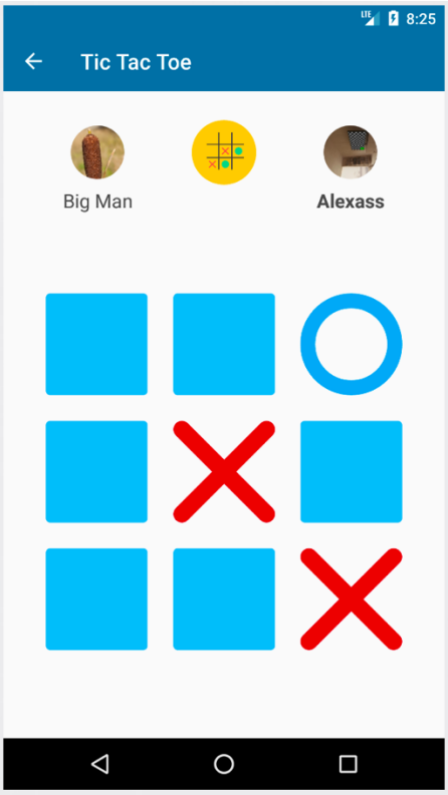

ilustración 93, pantalla de juego jugando.

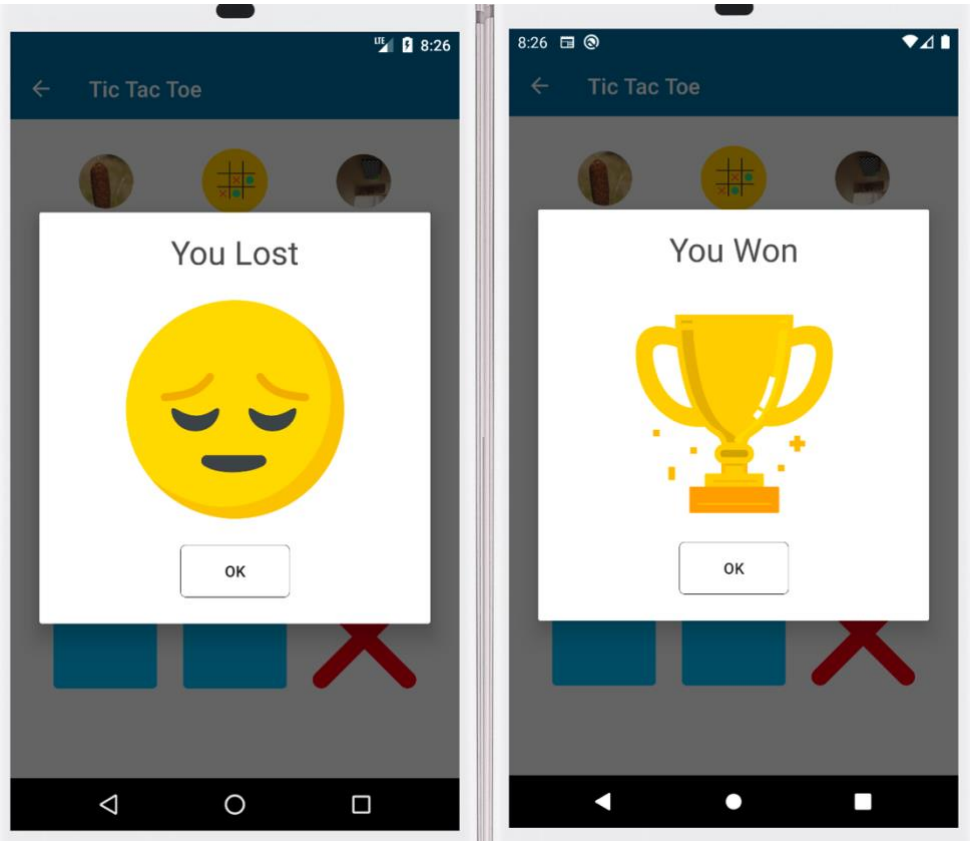

ilustración 94, pantalla de juego terminado.

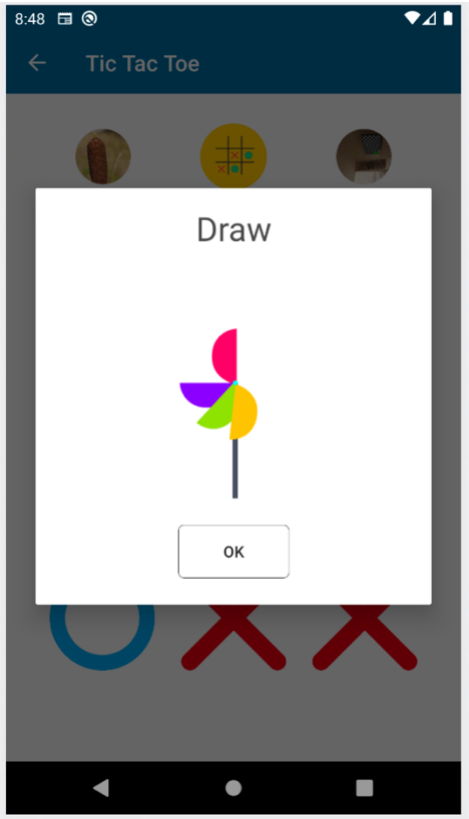

ilustración 95, pantalla de juego empatado.

3.9 Otras funcionalidades

Admob

Es una herramienta que ofrece Google de monetizar la aplicación a través de anuncios integrados de forma híbrida en la aplicación. Estos anuncios pueden ser de tipo banner, que es básicamente una barra; intersticial, que es un anuncio de una pantalla completa; o de videos que también es un anuncio de pantalla completa con reproducción de un vídeo.

En nuestro caso se ha implementado el intersticial puesto que es cancelable en cualquier momento. Los banners no son cancelables y nos videos se deben reproducir enteros para cobrar lo que empeorarían la experiencia del usuario.

Se reproduce un anuncio de tipo intersticial cada vez que se termina el juego puesto que allí es el momento menos molesto para el usuario.

Crashytics de Firebase

Es una herramienta que ofrece Google que permite informar fallas en tiempo real, hacer un seguimiento de los problemas de estabilidad que afectan la calidad de tu app, priorizarlos y corregirlos.

Multilenguaje

Por defecto el idioma de la aplicación está en inglés. Se traduce automáticamente a español si su Android está configurada en español.

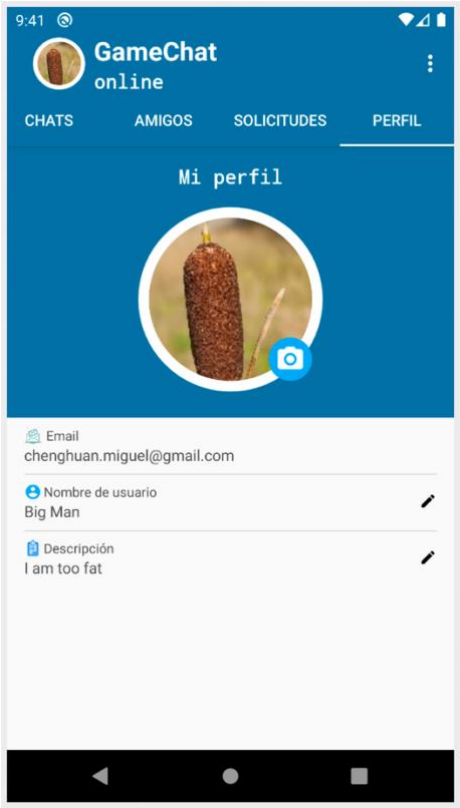

ilustración 96, pantalla principal en español.

Créditos

Se ha añadido un diálogo de créditos haciendo las atribuciones necesarias. Este diálogo se muestra en el menú de opciones de la pantalla principal.

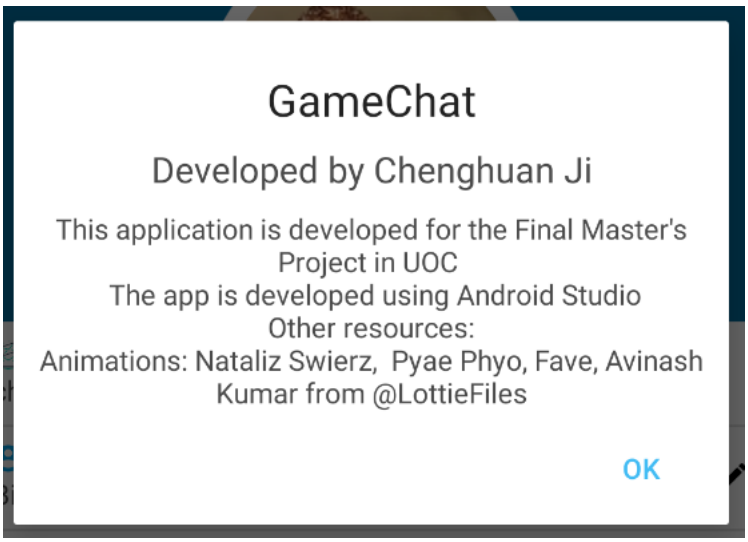

ilustración 97, créditos.

Esto es necesario ya que las animaciones son descargadas de la página LottieFiles, y la atribución del autor es necesaria para poder usar.

Optimización y reducción del código:

Se ha aplicado las herramientas de la reducción, la ofuscación y la optimización del código añadiendo configurando el buildTypes en Gradle.
Esto permite que la APK resultante pese menos, con lo que ahorraría espacio para los usuarios al descargar e instalar la aplicación.

Publicación en fase beta:

Se ha publicado la aplicación en fase beta en Google Play de forma abierta. Cualquier persona puede descargarla y probarla en su móvil.

Puede descargar y probar la aplicación desde su móvil en:
https://play.google.com/store/apps/details?id=com.chenghuanji.gamechat

4. Conclusiones

En este apartado se va a realizar un análisis de todo el trabajo realizado para la realización de la aplicación GameChat para Android. Se va a comentar las lecciones aprendidas, si se ha logrado todos los objetivos, si ha sido necesario cambiar la planificación, líneas de trabajo futuras, etc.

4.1 Lecciones aprendidas

Se ha aprendido a diseñar, estructurar e implementar una aplicación mas o menos extensa empezando por la planificación, luego pasando por el prototipado y la estructuración de las clases y finalmente su implementación.

Se ha aprendido a implementar una aplicación de tipo Chat utilizando las tecnologías de Firebase. Concretamente usando la base de datos, realtime database, que refresca a tiempo real permitiendo la simulación de los envíos y recepciones de los mensajes.

Se ha aprendido a introducir un juego multijugador dentro de en una aplicación. También usado realtime database.

Se ha aprendido a incluir las notificaciones remotas usando el Firebase Cloud Messaging que incluso permite recibir notificaciones con la aplicación cerrada. Así como el tratamiento de los eventos en la notificación con un broadcast Receiver y un servicio.

Se ha aprendido el manejo de otras tecnologías de Firebase como la autenticación para suscribir a los usuarios y el storage para almacenar y recuperar los archivos grandes en la nube.

4.2 Logro de los objetivos

En principio había pensado en implementar 2 juegos en vez de uno sólo. Por la falta de tiempo y porque con sólo el Chat + un Juego ya es suficientemente extenso no se ha implementado el segundo juego.

Los restos objetivos han sido logrados. Se ha comprobado que funciona correctamente tanto desde diferentes emuladores como desde un dispositivo real.

4.3 Seguimiento de la planificación y metodologías

Se ha seguido de forma satisfactoria la planificación en general, aunque como mencionado anteriormente no ha sido posible la implementación de un segundo juego, puesto que las demás funcionalidades han necesitado más tiempo de lo previsto. Por lo que posteriormente se ha realizado una pequeña modificación en la planificación para quedarse coherente el trabajo.

La metodología usada ha sido adecuada. Puesto que ha permitido la implementación y la corrección de los errores a tiempo y terminar el trabajo respetando la fecha de entrega.

4.4 Líneas de trabajo futuro

Se puede ir añadiendo más juegos en el futuro para que la aplicación sea más entretenida, permitiéndose más opciones a elegir para los usuarios.

Se realizará la publicación definitiva de la aplicación si no se encuentra errores en la fase beta.

Se tendrá que habilitar quizás el plan Blaze de Firebase si va a haber muchos usuarios. O cambiar a otro servidor si se considera inadecuado este plan.

5. Glosario

- Actividad: componente Android que proporciona una pantalla de la aplicación con su interfaz.

- Broadcast receiver: es un componente en Android dedicado a recibir y responder eventos.

- Diálogo: es una “ventana” flotante en Android.

- Fragmento: componente Android que proporciona una porción de una pantalla de la aplicación con su interfaz.

- Menú contextual: es un menú flotante que se muestra al dar un clic largo sobre un elemento.
- Menú de opciones: es el menú que está en la esquina superior derecho, que se muestra al dar al icono de 3 puntos.

- Recyclerview: es un elemento visual de Android que está formado por una lista de subelementos permitiéndose mostrar estos subelementos en forma de lista.

- SharedPreferences: es un documento XML que sirve para guardar datos simples.

- Servicio: es un componente Android para realizar tareas en segundo plano a larga duración.

- Viewpager: es un elemento visual de Android que se caracteriza por permiter desplazarnos por distintos layouts o “páginas” dentro de una misma Activity simplemente deslizando las páginas a la derecha o izquierda.

- Tostada o toast: es un mensaje puramente informativo de Android que se muestra en la pantalla durante unos pocos segundos.

6. Bibliografía

1. Wikipedia de Whasapp. Visitada en 09/2020
https://en.wikipedia.org/wiki/WhatsApp

2. Ventajas y desventajas de Whasapp. Visitada en 09/2020
https://www.lasexta.com/tecnologia-tecnoxplora/apps/cinco-ventajas-cinco-desventajas-nuevo-whatsapp-web_2015012357f78f7c0cf2a2e945b3d0be.html

3. Video tutorial de Diagrama de Gantt. Visitada en 09/2020
https://www.youtube.com/watch?v=chR6kx4btDQ

4. Vídeo tutorial de UML. Visitada en 10/2020
https://www.youtube.com/watch?v=Z0yLerU0g-Q

5. Vídeo tutorial de una aplicación Chat. Visitada en 10/2020
https://www.youtube.com/watch?v=fJWFeW09qeE&list=PLzLFqCABnRQftQQETzoVMuteXzNiXmnj8

6. Más video tutoriales en Udemy. Visitada en 10/2020
https://www.udemy.com

7. Lucidchart. Visitada en 10/2020
https://lucid.app

8. Documentación general de Firebase. Visitada en 10/2020
https://firebase.google.com/docs/android/setup?hl=es-419

9. Precios de Firebase. Visitada en 10/2020
https://firebase.google.com/pricing

10. Consola de Firebase. Visitada en 11/2020
https://console.firebase.google.com

11. Documentación de Autenticación de Firebase. Visitada en 11/2020
https://firebase.google.com/docs/auth

12. Documentación de Realtime Database de Firebase. Visitada en 11/2020
https://firebase.google.com/docs/database

13. Documentación de Storage de Firebase. Visitada en 11/2020
https://firebase.google.com/docs/storage

14. Documentación de Capacidades Offline de Firebase. Visitada en 11/2020
https://firebase.google.com/docs/database/android/offline-capabilities#section-disk-persistence

15. Documentación de Cloud Messaging de Firebase. Visitada en 11/2020
https://firebase.google.com/docs/cloud-messaging

16. Documentación de Crashlytics de Firebase. Visitada en 11/2020
https://firebase.google.com/docs/crashlytics

17. Documentación de Room. Visitada en 11/2020
https://developer.android.com/training/data-storage/room

18. Pagina oficial de Lottie. Visitada en 11/2020
https://airbnb.io/lottie/#/android

19. Animaciones gratuitas de Lottie. Visitada en 11/2020
https://lottiefiles.com

20. Página oficial de Picasso. Visitada en 10/2020
https://square.github.io/picasso

21. Github de CircleImageView. Visitada en 10/2020
https://github.com/hdodenhof/CircleImageView

22. Pagina oficial de retrofit Visitada en 11/2020
https://square.github.io/retrofit

23. Pagina de inicio de Admob. Visitada en 11/2020
https://admob.google.com/home/

24. Documentación de anuncio Intersticial. Visitada en 11/2020
https://developers.google.com/admob/android/interstitial

25. Cómo compilar y ejecutar una aplicación. Visitada en 12/2020
https://developer.android.com/studio/run?hl=es-419

26. Optimización del código. Visitada en 12/2020
https://developer.android.com/studio/build/shrink-code

27. Documentación de Android. Visitada durante todo el trabajo.
https://developer.android.com/docs

28. Página para consultar dudas de Stackoverflow. Visitada durante todo el trabajo.
https://stackoverflow.com

7. Anexos

7.1 Manual de instrucciones de compilación.

Este manual tiene la funcionalidad de mostrar cómo se compila y se ejecuta la aplicación en un emulador a partir del código fuente proporcionado.

Previamente debe tener instalado el Android Studio y un emulador.

Para compilar y ejecutar la app, sigua los siguientes pasos:

1. En la barra de herramientas, selecciona la app en el menú desplegable de configuraciones de ejecución. Por defecto esto ya está seleccionado.
2. En el menú desplegable del dispositivo de destino, selecciona el dispositivo deseado en el que se va a ejecutar la aplicación. Si no lo tiene, puede crear uno seleccionando la opción de AVD Manager.
3. Haz clic en Run para comenzar a compilar y ejecutar la aplicación.

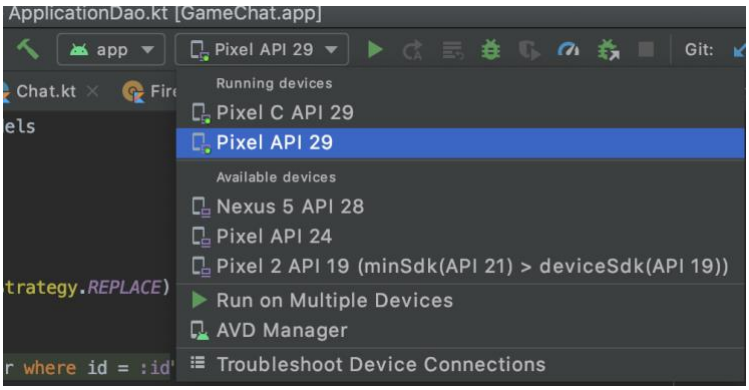

ilustración 98, instrucciones de compilación y ejecución.

Y desea compilar y generar el APK para compartir con los demás:
Seleccione en la barra de menús la opción Build -> Build Bundle(s) / APK(s) -> Build APK(s)

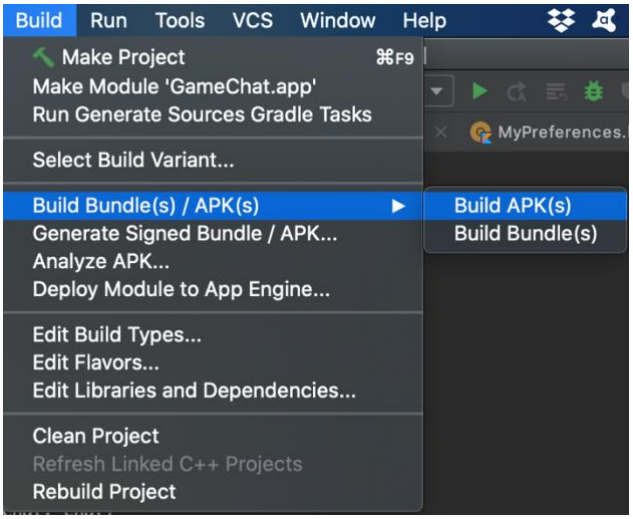

ilustración 99, instrucciones de compilación APK.