2021-2022

Carrusel

MEMORIA DEL TRABAJO DE FIN DE CICLO
DESARROLLO DE APLICACIONES WEB

LORENZO CANO SÁNCHEZ

Contenido

1. Introducción 2
   1.1. Orígen y contextualización del proyecto 2
   1.2. Objetivos del proyecto 2
2. Desarrollo 3
   2.1. Tecnologías empleadas 3
   2.2. Partición del desarrollo 3
   2.3. Front-end 4
      2.3.1. Diseño de la aplicación 4
      2.3.2. Pantallas y componentes 5
   2.4. Back-end 10
      2.4.1. Colecciones de la BBDD 10
      2.4.2. Seguridad 11
      2.4.2.1. Reglas de Firestore Database 11
      2.4.2.2. Reglas de Firestore Storage 12
3. Funcionalidades y uso de la aplicación 13
4. Posibles mejoras 20
   4.1. Notificaciones 20
   4.2. Envío de vídeos u otros elementos de la galería del dispositivo 20
   4.3. Autenticación mediante número de teléfono 20
   4.4. Mensajería cifrada 20
5. Bibliografía 21

1. Introducción

El objetivo de este proyecto es construir una aplicación móvil con tecnologías propias de desarrollo web, que funcionará como una aplicación de mensajería completa donde los usuarios podrán darse de alta e intercambiar mensajes de texto y e imágenes de forma sencilla, intuitiva y que resulte familiar. Para ello me he basado en características propias de Telegram y de WhatsApp, dos de las mayores aplicaciones de mensajería móvil.

1.1. Orígen y contextualización del proyecto

La idea del proyecto surge de WhatsApp, la red social más popular de los últimos años. La enorme popularidad de la plataforma nace de su propia sencillez, permitiéndote enviar mensajes a tus contactos de forma gratuita, al contrario que los SMS.

¿Cuál es su mayor incinveniente? La opacidad de la propia compañía con respecto a la recopilación y el tratamiento de los datos de sus usuarios. WhatsApp fue comprada por Facebook, compañía conocida por sus redes sociales y por la ingente cantidad de datos que recopila de sus usuarios, que muchos consideran innecesarios, para venderlos al mejor postor (normalmente empresas publicitarias). Es más, se les acusa de colaborar en labores de espionaje en favor de los intereses de la NSA, la CIA y demás organizaciones estadounidenses.

De ahí nació la idea de desarrollar esta aplicación, para presentar una alternativa a las aplicaciones de mensajería convencionales en caso de que la crispación hacia ciertas compañías y sus aplicaciones creciese, o se llegase incluso al extremo de prohibirse en algunos países.

1.2. Objetivos del proyecto

El objetivo del proyecto es desarrollar una aplicación móvil capaz de replicar las características de WhatsApp y Telegram, entre otras, con un tratamiento de los datos de usuario mucho más transparente. Además, esta aplicación deberá ser una herramienta moderna y escalable, por ello, se construyó la parte front-end con React Native (concretamente Expo) y la parte back-end se desplegará toda en Google Firebase, dotando a la aplicación de velocidad, rendimiento y escalabilidad.

Para lograrlo, se han de poder realizar las siguientes tareas:

- Autenticación del usuario en el sistema
- Alta, modificación y baja del usuario, así como manejo de su sesión en la app
- Alta, modificación y baja de salas de chat
- Listado de salas de chat abiertas, si las hay
- Listado de Contactos del teléfono. Para menores complicaciones, los contactos se identificarán por su email, no por su número de teléfono
- Creación de mensajes en cada sala de chat, así como envío y manejo de imagenes
- Envío de fotografías desde la pantalla de chat, imitando a la forma en la que se puede hacer desde WhatsApp

2. Desarrollo

En este punto se tratarán las tecnologías empleadas, así como los pasos que se han llevado a cabo para el desarrollo de la aplicación.

2.1. Tecnologías empleadas

- React Native y Expo (front-end): React Native es un framework JavaScript que permite crear aplicaciones móviles nativas para iOS y Android, basado en el framework del mismo nombre React. Expo es un conjunto de herramientas y servicios creados explícitamente para React Native, cuyo objetivo es agilizar y acelerar el desarrollo de aplicaciones en React Native. Además, Expo nos permite ejecutar y depurar nuestra aplicación en tiempo real de forma remota. Basta con tener un dispositivo Android con la aplicación oficial de Expo instalada, ejecutar el comando "npm run start" en la raiz del código fuente, y escanear el codigo QR que nos indique. Además, para dibujar los chats correctamente utilizaremos GiftedChat.

- Google Firebase (back-end): Firebase es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. Se basa en la idea de BaaS (Back-end as a Service), y nos permite tener tanto bases de datos en tiempo real como almacenamiento en la nube para la subida de ficheros. De este modo, no necesitamos contratar un servicio de alojamiento ni registrar un dominio para nuestra aplicación, sólo con las API Keys que nos proporcione Google desde Firebase podremos desplegar nuestra aplicación sin más complicaciones. Además, dota de una gran escalabilidad a nuestra app y, mientras que no excedamos los límites que nos imponen, es totalmente gratuito. Es perfecto para proyectos personales o no muy grandes, como éste.

- Git (control de versiones): Git nos permite controlar las versiones que vayamos realizando de nuestro proyecto, de modo que cada cambio realizado sobre el código quede registrado y, si lo usamos como corresponde, bien documentado.

2.2. Partición del desarrollo

El desarrollo de la aplicación puede dividirse en distintas secciones:

1. **Estructuración y planificación:** Esta parte ya se realizó en el anteproyecto. Consiste en analizar las distintas tecnologías a usar y planificar las distintas etapas del desarrollo de nuestra aplicación.
2. **Configuración del entorno de desarrollo:** Preparar e instalar el software necesario para trabajar y practicar sobre las nuevas tecnologías a aprender.
3. **Aprendizaje:** El más importante paso previo al desarrollo. Se realizarán distintos cursos online para aprender sobre React Native con Expo y sobre cómo aplicar correctamente las reglas de seguridad de Firestore Database y Firestore Storage.
4. **Puesta en práctica de lo aprendido:** Tras haber realizado todo lo anterior, toca ponerse con el desarrollo de la aplicación.

2.3. Front-end

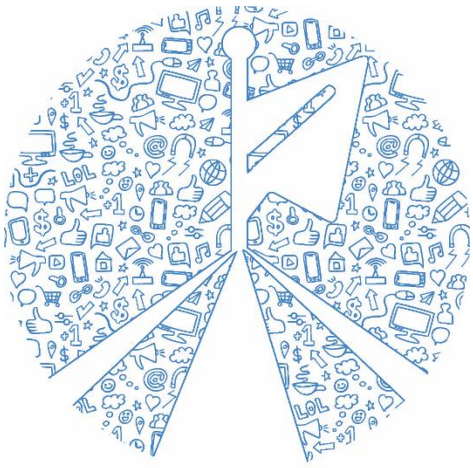
2.3.1. Diseño de la aplicación

Las hojas de estilos de React Native giran en torno a flexbox, lo cual por un lado es muy cómodo, ya que facilita el desarrollo de una interfaz de usuario responsive, y por otro, una auténtica pesadilla a la hora de querer utilizar layouts distintos según la vista o "view" en la que estemos. El diseño de la aplicación se centrará en imitar a la interfaz de usuario de Telegram.

Para ello, he definido una constante "theme" que contendrá tres cosas: una paleta de colores predefinida, llamada "colors", y dos layouts para los principales tipos de vistas, llamadas "container" y "perfil". Este tema irá dentro del fichero "utils.js" para su uso posterior en nuestro contexto "context/Context.js" y su correspondiente wrapper o empaquetador "context/ContextWrapper.js". Cabe destacar que el fondo de pantalla de los chats irá aparte, en la carpeta "assets", y no tiene que ver con el tema de la aplicación.

La paleta de colores será universal, no se usarán otros colores en la aplicación. Pero para algunas vistas, se requerirán pequeños ajustes visuales para su correcta disposición y no podremos usar ninguno de los estilos predefinidos. Además, el logotipo que servirá de icono de la aplicación también sigue el mismo esquema de colores. Aquí se muestra el contenido de "theme" a la izquerda, el icono de la aplicación arriba a la derecha, y la imagen que se muestra en la pantalla inicial de la aplicación abajo a la derecha:

```
export const theme = {
  colors: {
    background: "#cbdde8",
    foreground: "#25598a",
    primary: "#4484c6",
    tertiary: "#a6c6fa",
    secondary: "#2566d7",
    white: "white",
    text: "#3C3C3C",
    secondaryText: "#707070",
    danger: "#d42434",
    lightGray: "#b3b3b3",
    iconGray: "#626262",
  },
  container: {
      flex: 1,
      justifyContent: "center",
      alignItems: "center",
      backgroundColor: "white",
  },
  perfil: {
      flex: 1,
      justifyContent: "center",
      alignItems: "center",
      backgroundColor: "white",
      padding: 40
  },
};
```



2.3.2. Pantallas y componentes

Aquí se muestra un listado de los distintos componentes y pantallas que se han utilizado en la aplicación.

Fichero	Funcionalidad
Componente Avatar.js	- Recibe un usuario y un número por parámetros. - Dibuja un objeto <Image /> con el avatar del usuario especificado y del tamaño que se le mande por parámetros.
Componente CabeceraChat.js	- Lee el usuario y la sala por parámetros de ruta de React Native, y dibuja el avatar y el nombre de contacto del usuario con el que estas chateando. En caso de que no tengas al usuario en contactos, se mostrará su nombre de usuario dentro de la aplicación. Al pulsarlo, nos llevará a InfoUsuario.js, con los datos de dicho usuario. - También dibuja un icono de una papelera. Tocando en él, borrará la sala de chat actual y te mandará de vuelta a la lista de chats.
Componente ElementoLista.js	- El principal elemento en todas las listas de usuarios de la aplicación. Recibe numerosos parámetros que después manda a la pantalla Chat.js - Cuando se muestra en la pantalla de Contactos.js, muestra el nombre del contacto y su avatar. Manda el usuario del contacto a la pantalla Chat.js por parámetro. - Cuando se muestra en la pantalla de Chats.js, muestra el avatar, nombre de contacto o, si este no existe, nombre de usuario, una descripción con el último mensaje recibido o enviado (llamado lastMessage en la BBDD) y la fecha en que se recibió/envió éste. Manda el usuario y la sala que le corresponde a la pantalla Chat.js por parámetro. - Cuando se muestra en la pantalla de Contactos.js tras haber recibido el parámetro “image” de la pantalla Foto.js, muestra el nombre del contacto y su avatar. Manda el usuario del contacto, la sala que le corresponde y la foto tomada a la pantalla Chat.js por parámetro.

Componente IconoFlotanteContactos.js	- Muestra un botón flotante circular con el color secundario de fondo. Tiene un icono blanco de un bocadillo de texto. - Se muestra abajo a la derecha en la pantalla de Chats.js y, al pulsarlo, nos lleva a la pantalla Contactos.js
Componente IconoPerfil.js	- Muestra en un circulo el avatar del usuario actual. - Se muestra a la derecha de la barra superior de la pantalla “feed” dentro de App.js, y al pulsarlo nos lleva al componente Perfil.js
Context.js	- Establece el contexto global de la aplicación (GlobalContext). Utilizado luego por ContextWrapper.js para llevar distintos valores al resto de la aplicación. - Su propósito es llevar el tema de la aplicación y los distintos tipos de salas al resto de componentes. - Hay dos tipos de salas: “rooms”, filtradas por el último mensaje escrito, y “unfilteredRooms”, sin ningún tipo de filtro.
Hook useHooks.js	- Su propósito es obtener los contactos del dispositivo, asignarlos a un usuario de la BBDD, y devolverlos en una lista.
Pantalla Acceder.js	- Permite al usuario registrarse o identificarse para ingresar en el sistema. - Para el registro se le requerirá al usuario escribir su nueva contraseña dos veces, ya que no se podrá alterar.
Pantalla CrearPerfil.js	- Se creó para mostrarse específicamente tras registrarnos en el sistema. - Permite al usuario tomar una foto con al aplicación de la cámara nativa del dispositivo para hacerla su foto de perfil, y establecer su nombre de usuario.

Pantalla Chats.js	- Muestra las salas de chat abiertas en las que se encuentra el usuario actual, además de un icono flotante que te lleva a la lista de contactos del teléfono. - En caso de no estar en ninguna, mostrará un mensaje con un icono haciendoselo saber al usuario.
Pantalla Foto.js	- Al enfocarnos en ella, abrirá la aplicación de la cámara nativa del dispositivo para tomar una foto. - Si tomamos una foto, nos mostrará los contactos del teléfono para elegir a quién mandarsela.
Pantalla Contactos.js	- Dibuja una lista de los contactos de nuestro dispositivo, que obtenemos del hook useContactos() del fichero useHooks.js. - Al seleccionar un contacto, creará una sala de chat con el mismo. - En caso de que esta sala ya exista, simplemente nos llevará a ella.
Pantalla Chat.js	- Recibe por parámetro de ruta de React Native: usuario al que queremos escribir, sala (si existe) e imagen (si venimos de Foto.js) - Si no existe la sala, la crea, vuelves a la pantalla de Contactos.js, y te mostrará un cuadro de diálogo notidicando al usuario de que acaba de ser creada. - Si existe la sala, dibuja arriba del todo una barra superior con el componente CabeceraChat.js antes mencionado, abajo del todo una barra con un cuadro de texto, un icono de cámara y un icono con un avión de papel, y en el centro, si hay, los mensajes que se encuentren en esa sala dentro de unos globos, divididos por fecha de envío. - Los globos que se encontrarán a la derecha serán del color terciario de la aplicación, y mostrarán los mensaje del usuario actual.

	- Los globos que se encontrarán a la izquierda serán de color blanco y mostrarán los mensaje del usuario con el que estemos chateando. - Todos los globos mostrarán la hora a la que se ha enviado el mensaje. - Si pulsamos sobre el icono del avión de papel, se enviará el mensaje. - Si pulsamos en el icono de la cámara, se abrirá la aplicación de cámara nativa del dispositivo. Si tomamos una foto, se enviará al otro usuario. - Las imágenes se mostrarán dentro de los bocadillos con forma rectangular. Si tocamos en ellas, se abrirá un modal <ImageView /> que nos permitirá ver la imagen más grande y hacerle zoom pellizcando la pantalla.
Pantalla InfoUsuario.js	- Muestra a la izquierda el avatar del usuario, y a la derecha: en grande y en negrita el nombre de contacto (si existe) o su nombre de usuario (displayName), debajo en gris y pequeño su nombre de usuario (displayName), y debajo el email del usuario junto con un icono. - Si se pulsa sobre el avatar, se abrirá un moda <ImageView /> que permitirá ampliar y hacer zoom al avatar del usuario. - Si se pulsa sobre el email del usuario, se copiará al portapapeles y mostrará un mensaje haciendoselo saber al usuario.
Pantalla Perfil.js	- Arriba, muestra la información del usuario logueado, permitiendo cambiar su nombre de usuario y su avatar. - Abajo , muestra dos botones: uno para cerrar sesión, y otro para borrar la cuenta de usuario. - Al pulsar el botón de borrar cuenta, nos llevará a la pantalla BorrarCuenta.js

Pantalla BorrarCuenta.js	- Muestra dos campos de texto y un botón rojo. - En los campos de texto, el usuario deberá escribir su contraseña dos veces. - Si el proceso es exitoso, se le sacará al usuario de la aplicación y se volverá a la pantalla Acceder.js
firebase.js	- Aquí es donde se colocan las API Keys que nos proporcione Google. Contiene todo lo necesario para desplegar nuestra aplicación y conectar con Firebase para manejar la autenticación y el alta de usuarios.
utils.js	- Aquí se encuentran diversas funciones y constantes que son de utilidad en toda la aplicación. Quitando la constante “theme”, el resto son funciones sacadas de la documentación oficial de Expo, necesarias para el correcto funcionamiento de la aplicación.
App.js	- El componente principal de la aplicación. En él se encuentra el objeto <NavigationContainer />, es decir, es donde se dibujan el resto de componentes y el que maneja toda la navegación dentro de la app. - Si el usuario no ha sido autenticado, dibujará la pantalla Acceder.js para que se autentique. - Si el usuario está autenticado, pero no tiene un nombre de usuario o “displayName”, te llevará a la pantalla CrearPerfil.js para completar tu perfil de usuario - Si todo está en orden, te llevará a la pantalla “feed”, que contiene un <Tab.Navigator /> con acceso a las pantallas Chats.js y Foto.js

2.4. Back-end

2.4.1. Colecciones de la BBDD

Google Firebase nos ofrece dos opciones a la hora de elegir crear una base de datos: Firestore Database, que nos permite crear BBDD no relacionales, y Realtime Database, que es igual que la anterior opción pero funciona en tiempo real. Para este proyecto he elegido usar la primera opción.

En Firestore Database. a las tablas se les llama colecciones, a los registros documentos, y a los campos, pues campos. Al ser una BBDD no relacional, estas colecciones pueden contener a su vez otras colecciones. A continuación se muestra una lista de los documentos que habrá en la BBDD, con sus respectivos campos. Aquellos campos marcados con un "?" indican que son opcionales, es decir, no son necesarios para el funcionamiento de la app.

- usuarios: colección
    - displayName: texto
    - email: texto
    - ?photoURL: texto
    - uid: texto
- salas: colección
    - mensajes: colección
        - _id: texto
        - createdAt: fecha
        - ?image: texto
        - text: texto
        - user: mapa
            - _id: texto
            - avatar: texto
            - name: texto
    - lastMessage: mapa
        - _id: texto
        - createdAt: fecha
        - ?image: texto
        - text: texto
        - user: mapa
            - _id: texto
            - avatar: texto
            - name: texto
    - listaParticipantes: matriz (contiene emails de los participantes)
    - participantes: matriz (contiene datos de los participantes, salvo el uid)
    - rid: texto

2.4.2. Seguridad

Uno de los aspectos más importantes de una aplicación es su seguridad. Para garantizar que ningún usuario o tercero hace algo que no debe en nuestra BBDD o en nuestro almacenamiento en la nube, Google nos permite escribir reglas de seguridad para proteger estos datos y asegurar un uso adecuado de los mismos por parte de los usuarios.

Todas estas reglas las podemos encontrar en el directorio “firebase”, dentro del código fuente del proyecto.

2.4.2.1. Reglas de Firestore Database

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /usuarios/{userId} {
      allow read: if isAuthenticated();
      allow write: if uidMatch(userId);
    }

    match /salas/{roomId} {
      allow create: if isAuthenticated();
      allow read,update,delete: if isChatMember(resource.data.listaParticipantes);

      match /mensajes/{messageId} {
        allow create,update: if isChatMember(getSala(roomId).data.listaParticipantes);
        allow read: if !existeSala(roomId) || isChatMember(getSala(roomId).data.listaParticipantes);
        allow delete: if false;
      }
    }

    function getSala(roomId) {
      return get(/databases/$(database)/documents/salas/$(roomId))
    }
    function existeSala(roomId) {
      return exists(/databases/$(database)/documents/salas/$(roomId))
    }

    function isAuthenticated() {
      return request.auth != null;
    }
    function uidMatch(uid) {
      return isAuthenticated() && uid == request.auth.uid
    }
    function isChatMember(lista) {
      return isAuthenticated() && request.auth.token.email in lista;
    }

  }
}
```

Para faclitar la obtención de datos, se han creado 5 funciones:

- **getSala():** recibe el id de una sala por parametro y devuelve el documento que le corresponde.
- **existeSala():** recibe el id de una sala y comprueba si le coresponde algún documento de la colección “salas”. Devuelve un booleano.
- **isAuthenticated():** comprueba si el usuario ha sido autenticado / está logueado. Devuelve un booleano.
- **uidMatch():** recibe un id de usuario por parametro y comprueba si está autenticado y si ese id le corresponde al usuario autenticado / logueado.
- **isChatMember():** recibe una lista por parametro y comprueba si el usuario ha sido autenticado / logueado y si su email se encuentra en la lista de participantes (“listaParticipantes”) de una sala de chat.

Para garantizar la seguridad en la BBDD, se ha asegurado lo siguiente:

- Un documento de la colección “usuarios” se puede:
    - Leer si el usuario está autenticado
    - Escribir si el id del usuario coincide con el del usuario logueado
- Un documento de la colección “salas” se puede:
    - Crear si el usuario ha sido autenticado
    - Leer, modificar y borrar si el usuario se encuentra en la lista de participantes (“listaParticipantes”) de esa sala.
- Un documento de la colección “mensajes” se puede:
    - Crear y modificar si es un participante de la sala que le corresponde a la colección de mensajes.
    - Leer si: o no existe la sala, o es un participante de dicha sala.
    - Eliminar: nunca.

2.4.2.2. Reglas de Firestore Storage

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /images/{userId}/pfp.jpeg {
      allow read: if request.auth != null;
      allow write: if request.auth.uid == userId;
    }
    match /images/salas/{roomId}/{imageId} {
      allow write, read: if request.auth.token.email in roomId.split(':')
    }
  }
}
```

Los ficheros que subirá nuestra aplicación serán imágenes en formato JPEG, y los podrá almacenar de dos formas:

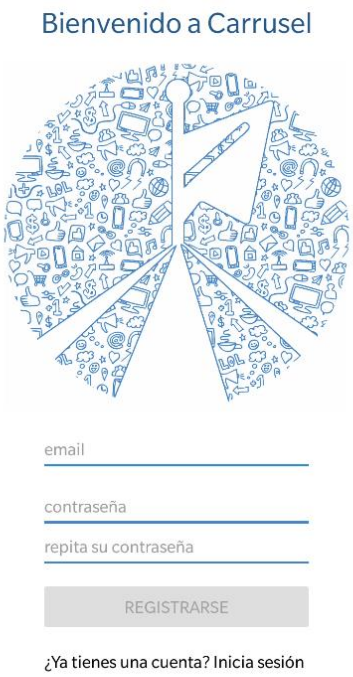
- Si es una foto de perfil, dentro de una carpeta con el id del usuario al que le corresponde, y con el nombre “pfp.jpeg”.
- Si es una foto que se ha mandado en una sala de chat, en la carpeta “salas” se creará otra carpeta (si no existe ya) con los correos de los participantes de la sala separados por dos puntos (<email1>:<email2>)

Para garantizar la seguridad en el almacenamiento de ficheros, se ha asegurado lo siguiente:

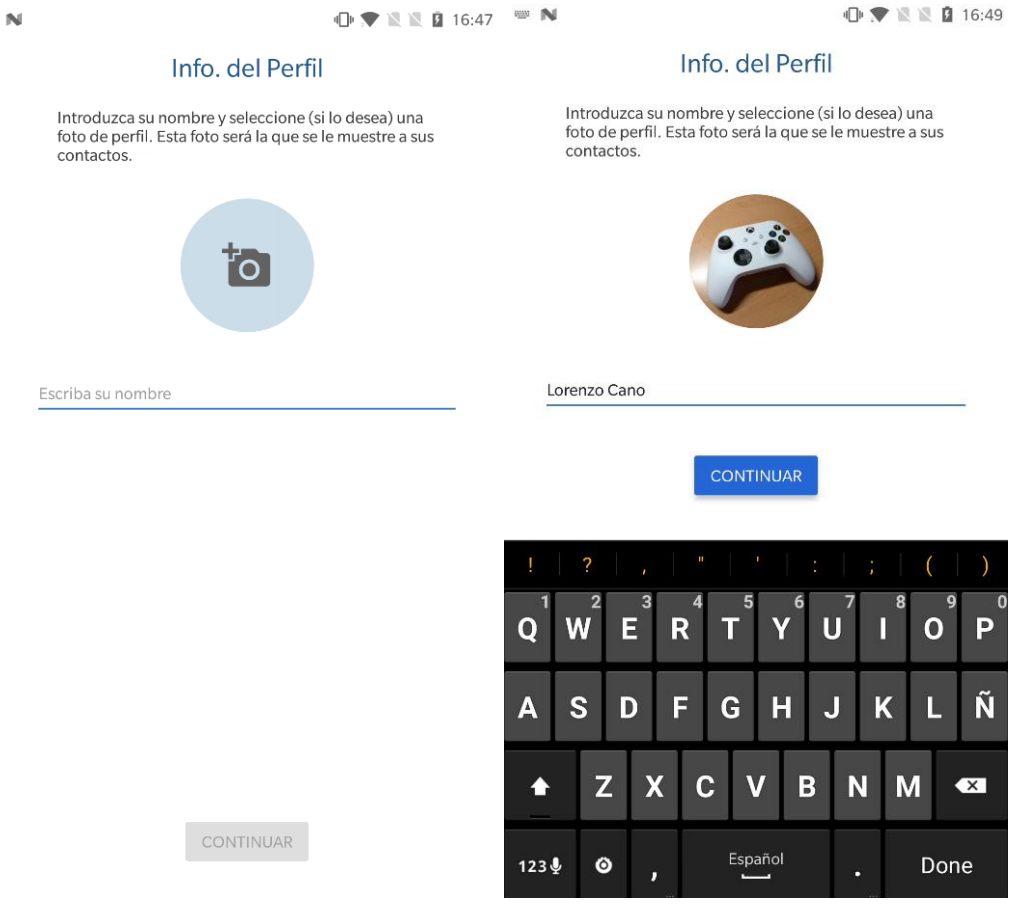
- Las fotos de perfil se podrán:
    - Leer si el usuario está autenticado.
    - Escribir si el id del usuario al que le corresponde esa foto (nombre de carpeta) es igual que el del usuario autenticado.
- Las fotos de las salas se podrán:
    - Leer y escribir si el email del usuario autenticado se encuentra en el nombre de la carpeta, que contiene dos emails (los de los participantes de la sala) separados por dos puntos.

3. Funcionalidades y uso de la aplicación

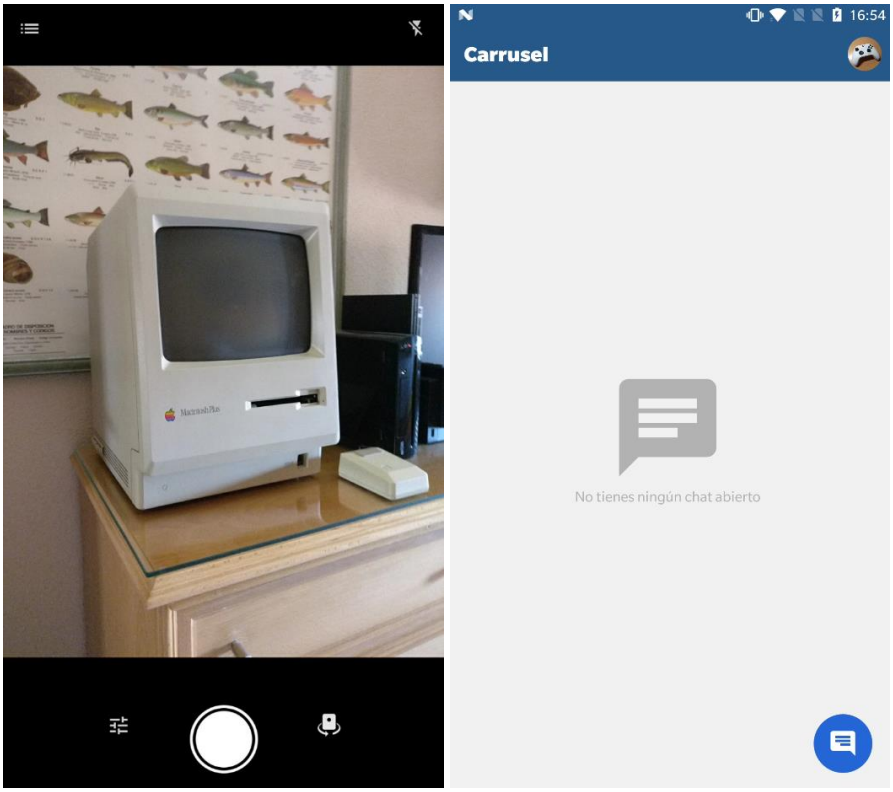
A continuación se van a mostrar las distintas funcionalidades de la app de una forma más visual, mediante capturas de pantalla y una breve descripción de lo que podemos hacer en cada pantalla.



Nada mas abrir la aplicación, si no tenemos ninguna sesión iniciada, nos recibirá con esta pantalla que, según si tocamos en el texto de debajo de todo, nos permitirá o registrarnos a autenticarnos con nuestro email y contraseña (por supuesto, para registrarnos deberemos escribir un email válido y las contraseñas han de coincidir). Si nos autenticamos, iremos a la pantalla principal. Si no, mostrará esta pantalla de abajo:

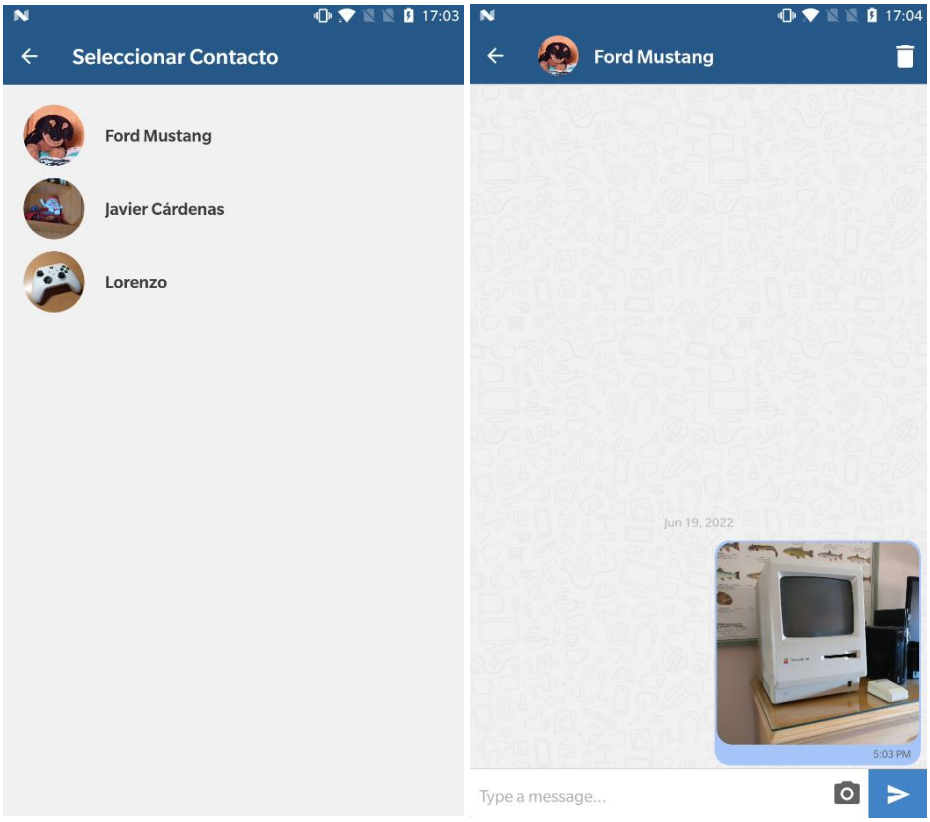


Aquí podremos establecer nuestro nombre de usuario y nuestra foto de perfil, siendo esta última opcional. Al pulsar en continuar, nuestro usuario ya estará creado y pasaremos a la pantalla principal:

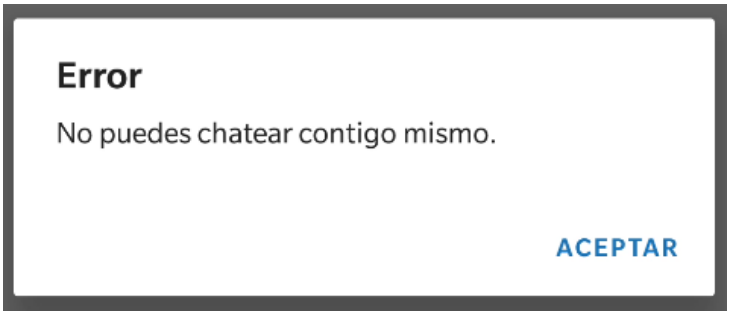


A la derecha se muestra la pantalla de Chats, donde aparecerán todas las salas de chat en las que nos encontremos. Además, podremos crear nuevas conversaciones pulsando sobre el icono flotante azul de abajo, y ver y editar nuestro perfil pulsando sobre nuestro avatar arriba a la derecha.

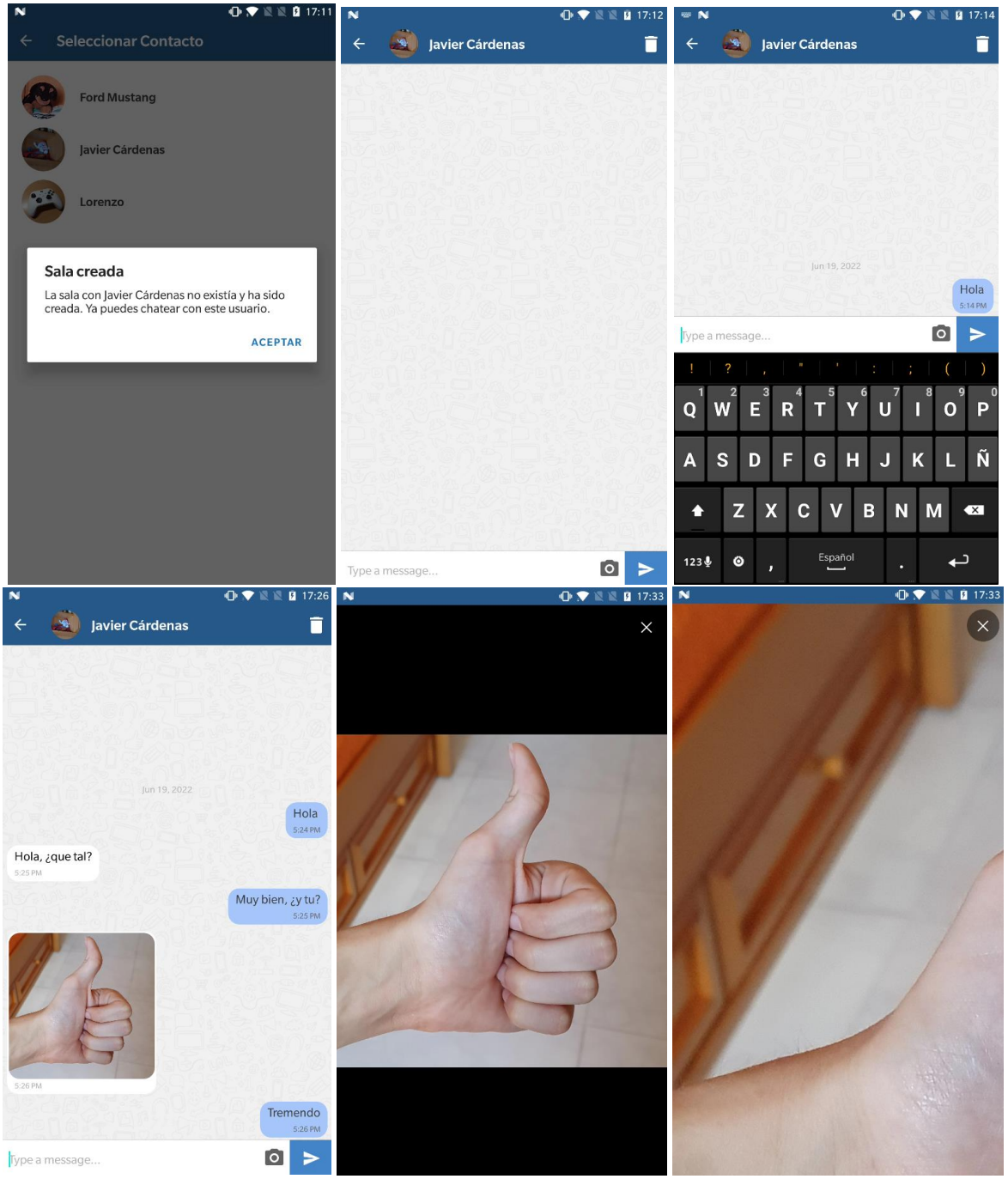
A la izquerda, se muestra la aplicación nativa de la cámara, se abre al deslizar hacia la izquerda sobre esta pantalla. Igual que en WhatsApp. Si tomamos una foto, nos mostrará la lista de contactos de nuestro dispositivo y podremos mandarsela. Si la sala con ese usuario no existe, se creará una y nos lo hará saber mediante un modal. Abajo se muestra como se la mandamos a uno de los contactos del teléfono (ya registrados en la app).



Visto esto, vamos a ver como mandamos un mensaje a un usuario desde la pantalla de Chats. Para ello, pulsamos en el botón flotante antes mencionado y seelccionamos un contacto de la lista. Si no existe una sala con ese usuario, se creará una y nos lo harán saber. Si existe, nos llevará a la sala. Y si el email del contacto coincide con el de tu usuario, no se te permitirá crear una sala.



Vamos a crear una sala con uno de nuestros contactos.

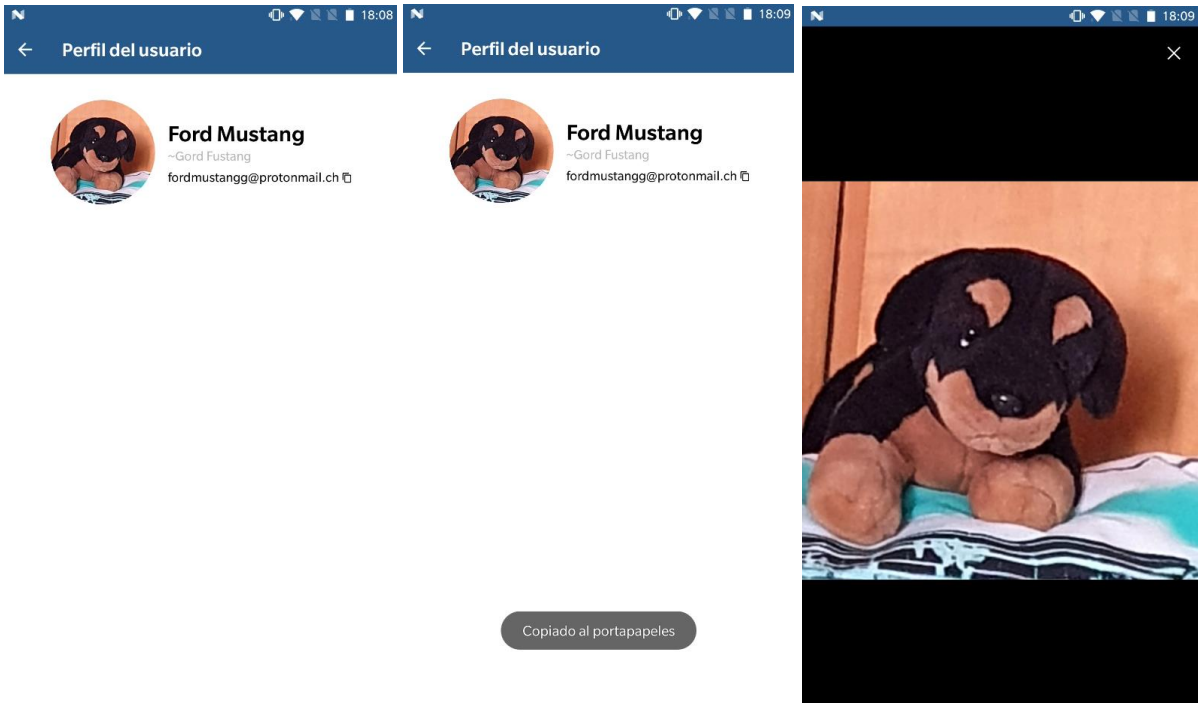


Como podemos ver, al ingresar en una sala vacía se nos presentará con muchos elementos: una cabecera, con el avatar del usuario, su nombre de contacto o de usuario y una papelera, un fondo de pantalla con un mosaico, y abajo, una barra de texto con dos iconos: uno de una cámara y otro de un avión de papel.

Todos estos elementos nos resultan muy familiares a aquellos que usamos aplicaciones de mensajería de forma cotidiana. Si pulsamos en el icono de la cámara, se abrirá la aplicación de la cámara y, si tomamos una foto, la mandará al otro usuario incrustada en un mensaje. Esta foto se podrá ampliar y pellizcar para hacer más zoom sobre ella.

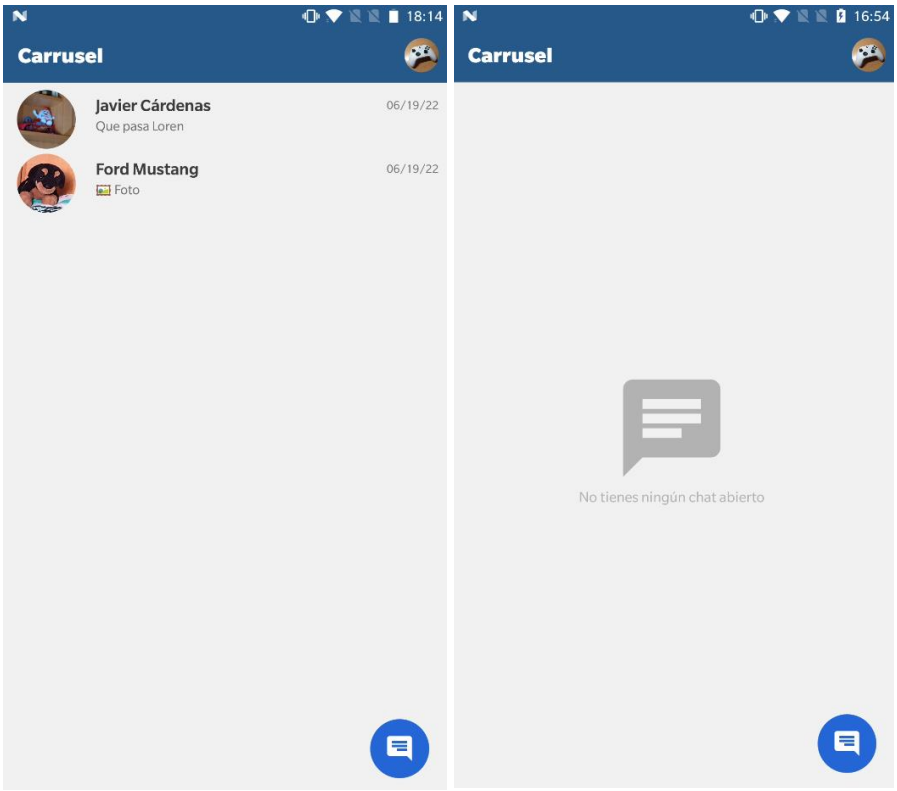
Si escribimos algo en el cuadro de texto y pulsamos sobre el icono del avión de papel, enviará el mensaje y aparecerá a la izquierda de nuestro chat en un globo color azul claro. Y, por el contrario, si recibimos un mensaje o foto del otro usuario, aparecerá a la derecha del chat en un globo de color blanco. Todo bien organizado, indicando fechas y horas de envío y recepción de mensajes.

Si pulsamos sobre el nombre de contacto o de usuario, nos llevará al perfil del usuario, donde se mostrarán sus datos: nombre de contacto (si existe), nombre de usuario, email y su avatar, el cual, si existe, podremos pulsar para verlo de forma ampliada. Si pulsamos sobre el email, se copiará al portapapeles y saltará un pequeño mensaje indicandonoslo.

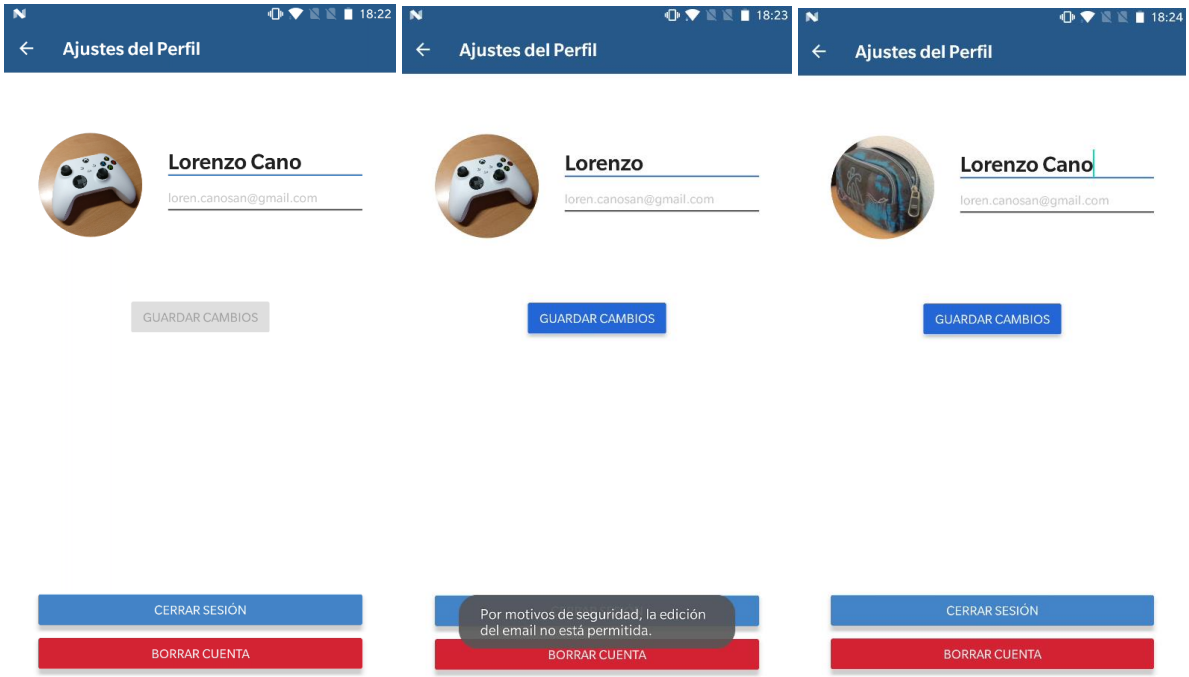


Si pulsamos sobre el icono de la papelera, nos preguntará si estamos seguros de que queremos borrar la sala. En caso de aceptar, finalmente se borrará la sala y volveremos a la pantalla principal.

La pantalla principal, según si tienes chats abiertos o no, se verá así:

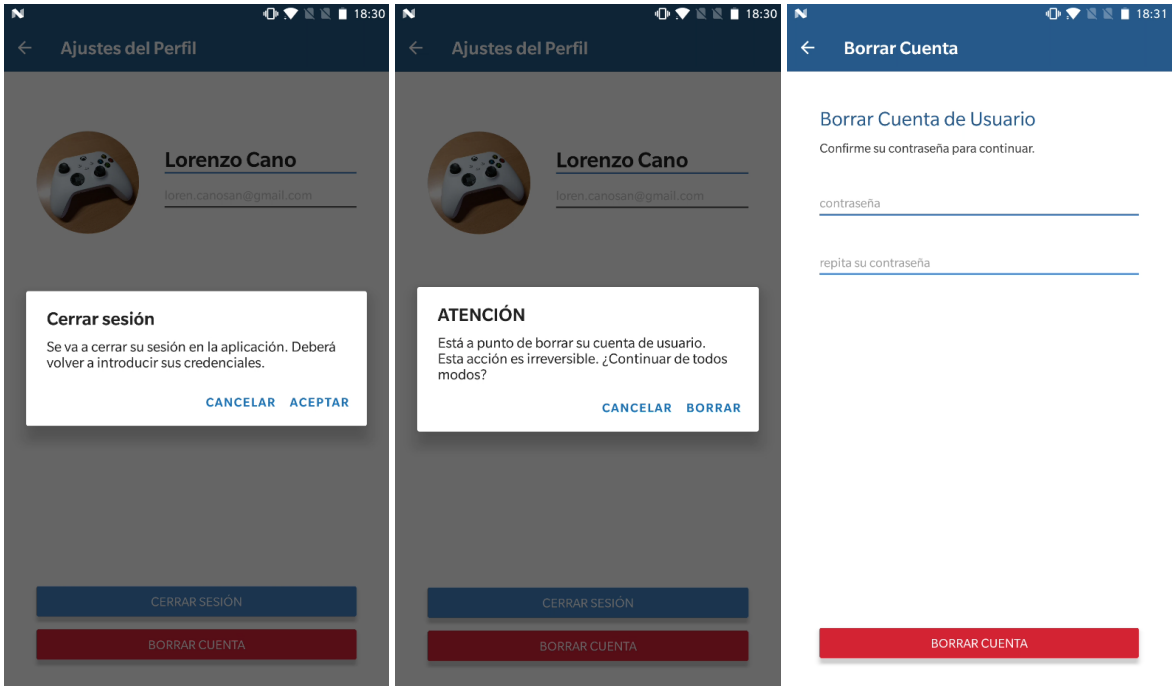


Finalmente, si accedemos a nuestro perfil, nos encontraremos con esto:



Se nos presentan distintos elementos en esta pantalla. Desde aquí podemos editar todos los detalles de nuestro usuario, salvo el email, por motivos de seguridad y para evitar posibles conflictos entre cuentas de usuario o suplantaciones de identidad. Sólo se podrá actualizar el perfil cuando se actualicen alguno de los datos, como muestran las capturas.

Además, con los dos botones de abajo podemos elegir entre cerrar la sesión y volver a la pantalla de acceso, o borrar nuestra cuenta de usuario. Para esto último, se nos pedirá escribir nuestra contraseña dos veces en otra pantalla y pulsar nuevamente sobre el botón. Una vez validados los credenciales del usuario, se borrará la cuenta y ya no podremos acceder de nuevo a la aplicación a menos que nos volvamos a registrar de nuevo.



4. Posibles mejoras

4.1. Notificaciones

Una característica esencial en aplicaciones de mensajería instantánea es el uso de las notificaciones para informar al usuario de los mensajes que ha recibido sin necesidad de abrir la aplicación. Lamentablemente, es una de las características que le faltan a este proyecto. Si bien es cierto que Firebase nos proporciona un servicio denominado Firebase Cloud Messaging (FCM) que nos podría ser de ayuda para implementarlas, también se podrían utilizar otros servicios alternativos como Expo Push Notifications, Amazon Simple Notification Service (SNS) o Azure Notification Hubs.

4.2. Envío de vídeos u otros elementos de la galería del dispositivo

El tamaño del proyecto, la dificultad del de desarrollo del mismo y la falta de tiempo han sido responsables de haberme impedido continuar con el desarrollo de nuevas características. Sin duda, sería una de las primeras características a implementar en caso de retomar y continuar este proyecto por mi cuenta.

4.3. Autenticación mediante número de teléfono

Google Firebase permite la autenticación y alta de usuarios a través de su número de teléfono. Para acceder a la app, el usuario tendría que escribir su número, y en otro campo de texto distinto un código que le llegaría por SMS.

Realmente no es complicado de implementar, pero, supuesto que para este proyecto necesitaba crear varias cuentas de usuario para hacer pruebas, el haber usado este método de autenticación me habría supuesto una gran limitación, ya que sólo dispongo de un número de teléfono.

4.4. Mensajería cifrada

La mensajería en la nube es muy conveniente, nos permite acceder a nuestras conversaciones desde cualquier dispositivo, ya que los chats se encuentran almacenados en el servidor de la aplicación. Sin duda, es la opción preferida de muchos usuarios, aunque también es la menos preferida de otros, entre los que me incluyo.

La mensajería cifrada permite que sólo los usuarios que se encuentran en esa sala puedan intercambiar mensajes, ya que estos se envían cifrados de tal forma que sólo el otro usuario pueda descifrarlos. Por ello, las conversaciones se almacenan de forma local en cada dispositivo en lugar de un servidor.

Una buena idea sería implementar el conocido Procolo Signal, que se basa en una versión del conocido protocolo criptográfico Diffie-Hellman, concretamente X3DH (Extended Triple Diffie-Hellman), y es completamente open-source.

5. Bibliografía

- React Native: https://reactnative.dev/
- Expo: https://expo.dev/
- Babel: https://babeljs.io/
- Npm: https://www.npmjs.com/
- MaterialCommunityIcons: https://github.com/oblador/react-native-vector-icons
- GiftedChat: https://github.com/FaridSafi/react-native-gifted-chat
- StackOverflow: https://stackoverflow.com/
- Photoshop: https://www.adobe.com/es/products/photoshop.html
- Visual Studio code: https://code.visualstudio.com/

Código fuente e instrucciones para la compilación y despliegue de la aplicación:
https://github.com/canosan/carrusel-app