UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

Aplicación de control de finanzas personales y grupales.

Martín Navarro Hernández

Ingeniería en Ciencias de la Computacion

Trabajo de fin de carrera presentado

como requisito para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 19 de mayo de 2024

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Aplicación de control de finanzas personales y grupales.

Martín Navarro Hernández

Nombre del profesor, Título académico Fausto Pasmay, M.Sc.

Quito, 19 de mayo de 2024

© DERECHOS DE AUTOR

Nombres y apellidos: Martín Navarro Hernández

Código: 00212858

Cédula de identidad: 1719188904

Lugar y fecha: Quito, 19 de mayo de 2024

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

El presente trabajo describe el desarrollo de una aplicación móvil enfocada en dispositivos de la marca Apple. Dicha aplicación permite a los usuarios gestionar sus finanzas de una manera interactiva y fácil. La aplicación permite ingresar gastos personales así como también gastos compartidos con otros usuarios de la aplicación los cuales son gestionados mediante mensajería interna donde se puede comunicar las finanzas del grupo. Para el desarrollo de este trabajo se utilizó el lenguaje de programación nativo de iOS llamado Swift y su marco de diseño denominado SwiftUI. En cuanto a la parte de backend y base de datos se utilizó el servicio de Google de Firebase.

Palabras clave: iOS, Swift, SwiftUI, Firebase, Firestore, Finanzas

ABSTRACT

This work describes the development of a mobile application focused on Apple brand devices. This application allows you to manage your finances in an interactive and easy way. The application allows you to enter personal expenses as well as expenses shared with other users of the application, which are managed through internal messaging where the group's finances can be communicated. For the purpose of this work, you can use the native iOS program language called Swift and the design name is SwiftUI. This part of the backend and database is used by the Google Firebase service.

Keywords: iOS, Swift, SwiftUI, Firebase, Firestore, Finance

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Estado del arte

El área de estudio se centra en el desarrollo de aplicaciones móviles utilizando tecnologías clave como Swift, SwiftUI y Firebase.

Swift

Swift es un lenguaje relativamente moderno que se introdujo por la empresa Apple en el año 2014 y desde entonces se ha convertido en la programación oficial para productos de Apple como iOS, MacOS, Ipad OS, TvOs y watchOS. Este lenguaje fue desarrollado como una alternativa y reemplazo de Objective-C el cual era comúnmente utilizado para desarrollar aplicación enfocados en los productos de la marca. Una de las razones por la cual se creó Swift fue por la complejidad de usar y aprender Objective-C, Swift ofrece una versión más moderna y más fácil de aprender conservando la potencia y flexibilidad que se podía encontrar en Objective-C. Swift se caracteriza por ser un lenguaje de programación orientada a objetos que está diseñado para ser seguro y evitar errores comunes. Utiliza una sintaxis fácil de manejar y aprender y con los años este lenguaje ha ganado mucha popularidad y poco a poco la comunidad ha crecido.

SwiftUI

SwiftUI es un marco de interfaz gráfica que trabaja conjuntamente con el lenguaje Swift. Este marco fue presentado por Apple en el 2019 y al igual que Swift se convirtió en el marco de interfaz gráfica principal para los productos de Apple. Similar a Swift, SwiftUI fue creado para reemplazar otro marco, en este caso UIKit el cual tenía una complejidad para aprender. SwiftUI llegó para ser más fácil de aprender conservando esa potencia que caracterizaba a UIKit. Hay que considerar que SwiftUI es un marco de interfaz de usuario declarativo, lo que significa que las interfaces de usuario se definen como declaraciones. Las declaraciones describen cómo se debe ver y comportarse una interfaz de usuario, pero no especifican cómo se debe implementar.

Firebase

Firebase es una plataforma de backend de Google que ofrece una variedad de servicios para el desarrollo de aplicaciones móviles y web. Fue presentado en 2011 y en 2014 la compañía Google la adquirió. Esta plataforma ofrece algunos servicios, uno de ellos es una base de datos en tiempo real que permite a los desarrolladores acceder y almacenar datos de manera sincrónica. Estos datos se almacenan dentro de una nube y cabe recalcar que la base de datos es NoSQL por lo tanto guarda los datos en formatos JSON. También ofrece un servicio de autenticación que permite autenticar a los usuarios de la aplicación de forma sencilla y con diferentes maneras de hacerlo como acceder por cuentas de Facebook o gmail.

En Firebase como tal no existe un función que explícitamente permite crear un chat entre usuarios pero gracias a la base de tiempo real es posible crear algo similar. En el caso de Swift es importante añadir el archivo de GoogleServices-Info.plist e instalar el SDK de Firebase en la aplicación de Xcode donde se encuentre el proyecto. Una vez hecho se procede a crear la colección en la base de datos en tiempo real, es importante tener una buena estructura ya que se necesitará indicar la identificación o usuario del remitente, el mensaje y la hora y fecha en la que dicho mensaje se envió. Con este se permitirá que los mensajes se actualicen en tiempo real utilizando un escuchando que detectara algún cambio en la base de datos. Firebase también permite subir fotos en forma de link a la base de datos. Esto da la opción de implementar una forma de compartir fotos entre los usuarios en la que se descarga la imagen utilizando el link relacionado a la imagen que está en la base de datos de Firebase. Es importante resaltar que Firebase cuenta con una versión gratuita en la cual incluye la mayoría de servicios como Autenticación, Real Time Database y Firestore. Para el caso de la base de datos es importante mencionar que en esta versión gratuita solo se ofrece hasta 1 Gb de almacenamiento la cual está bien para la aplicación que se va a desarrollar.

Aplicaciones similares

Dentro del App Store de Apple existen algunas aplicaciones que comparten características con la que se desarrolló para este documento. Una de ellas es *Daily Expense*, la cual tiene 4.7 de rating y 70 reviews. Esta aplicación se caracteriza por ser una aplicación enfocada únicamente en el control de finanzas personales. Permite controlar los gastos por mes y filtrar los gastos por categorías. Una de las funcionalidades que llaman más la atención de esta aplicación es la ayuda visual que ofrece, en donde se puede observar una barra circular de progreso donde se ven los resumen mensuales por categoría del usuario.

Otra aplicación que tiene aspecto similares se llama *Splitwise.* Esta a diferencia de la aplicación anterior tiene otro concepto u objetivo. La funcionalidad de esta aplicación se basa en facilitar los gastos entre varias personas. La aplicación calcula los gastos de cada participante para que sean pagados equitativamente, sin embargo no existe mucha capacidad de administrar los grupos, así como editar los gastos de un participante en específico o enviar archivos multimedia como parte de un respaldo del pago.

La aplicación presentada en este documento comparte algunas de las funcionalidades presentadas en *Splitwise* y *Daily Expense,* como el crear grupos de gastos compartidos y la capacidad de almacenar gastos personales con ayuda visual como gráficos. Lo que hace a esta aplicación diferente de estas es que combina ambas características principales dentro de una aplicación basada completamente en el lenguaje nativo de Apple, Swift. Esta app permite tener un mejor manejo de las finanzas personales y ayuda al usuario a concientizar sus gastos al tener objetivos/límites mensuales. Por parte de los gastos compartidos, la app se caracteriza por tener un proceso de acepto/rechazo en cada mensaje correspondiente a un pago, además de tener la habilidad de mandar imágenes como respaldo.

Descripción del Problema

La combinación de una aplicación para gestionar gastos personales y compartidos aborda de manera integral los desafíos financieros tanto a nivel individual como en entornos grupales. En la vida cotidiana, llevar un control preciso de los gastos personales es esencial, pero la gestión de gastos compartidos, ya sea con amigos, familiares o compañeros de cuarto, añade una capa adicional de complejidad. La relevancia de esta aplicación radica en proporcionar una solución completa que no solo permite a los usuarios rastrear y categorizar sus gastos personales, sino también gestionar y dividir eficientemente los gastos compartidos. La justificación se encuentra en su capacidad para simplificar la contabilidad personal, promover la transparencia en los gastos compartidos y mejorar la comunicación financiera en grupos, brindando así una herramienta integral para una gestión financiera más efectiva tanto a nivel individual como colectivo. Esto contribuye a una toma de decisiones más informada y a relaciones interpersonales más saludables desde el punto de vista financiero.

Objetivo General

Mejorar la gestión financiera y promover el bienestar económico para individuos y grupos mediante el desarrollo e implementación de una aplicación integral para Gastos Personales y Compartidos.

Objetivo Específico

- Optimizar el seguimiento de gastos individuales mediante una interfaz intuitiva que permita a los usuarios registrar fácilmente sus gastos personales.
- Desarrollar funciones que permitan a los usuarios registrar fácilmente gastos compartidos, calcular automáticamente la parte de cada usuario y facilitar el proceso de liquidación dentro de los grupos.
- Incorporar herramientas de análisis visual para proporcionar a los usuarios información detallada sobre sus patrones de gasto, fomentando una mayor conciencia financiera y toma de decisiones informada.
- Garantizar actualizaciones en tiempo real sobre gastos compartidos, permitiendo a los usuarios mantenerse informados acerca de quién ha pagado, saldos pendientes y la dinámica financiera general del grupo.
- Implementar funciones de presupuestación que permitan a los usuarios establecer límites de gasto personales y grupales.
- Incluir características de comunicación, como mensajería dentro de la aplicación, para facilitar discusiones transparentes dentro de los grupos sobre gastos compartidos, asegurando que todos estén al tanto de la situación financiera.

DESARROLLO DEL TEMA

Diseño de interfaz gráfica

Profile View/Vista de perfil

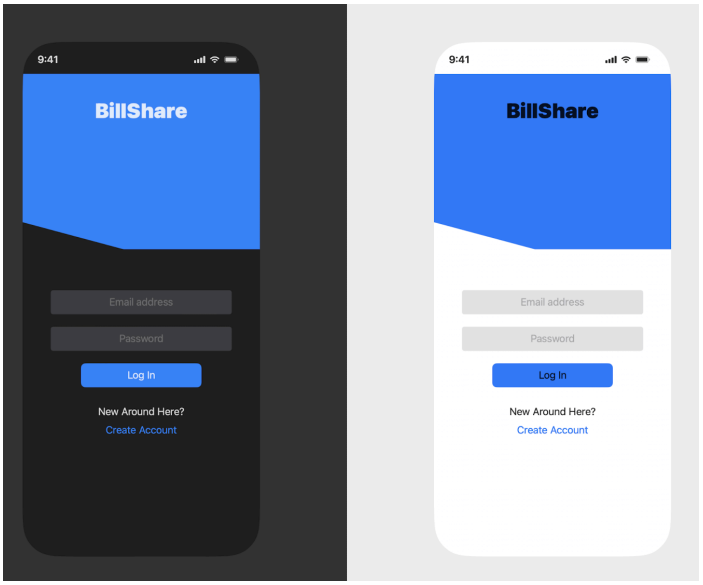

Figura #1: Diseño Vista de Login

Esta es la primera vista que va a ver el usuario una vez que abra la aplicación. Se compone de un form en donde hay dos campos. El primer campo corresponde al correo electrónico y el segundo campo corresponde a la contraseña. Es importante notar que para que estos campos se validen correctamente el usuario deberá tener una cuenta registrada en caso contrario deberá aplastar “Create Account” el cual lo llevará a la vista de registro.

Register View/ Vista de Registro

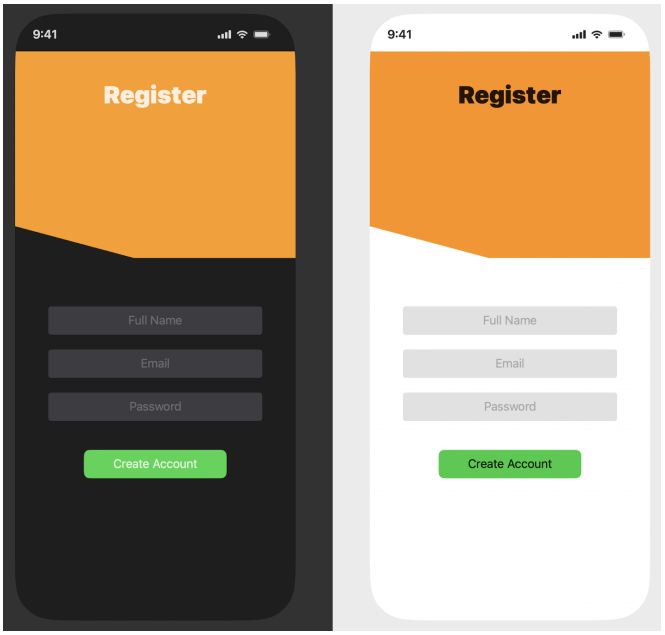

Figura #2: Diseño Vista de Registro

Si el usuario no ha creado una cuenta deberá ingresar a la vista de registro. Aquí el usuario debe llenar los campos del form para poder crear la cuenta y almacenarla en la base de datos de Firebase. Los campos del form se componen de tres campos, Nombre completo, correo electrónico y una contraseña. Una vez que se llenen estos campos el usuario debe pulsar el botón de crear cuenta. Si todo está correcto se crea la cuenta y se redirecciona al usuario a la vista de login/iniciar sesión.

Dashboard

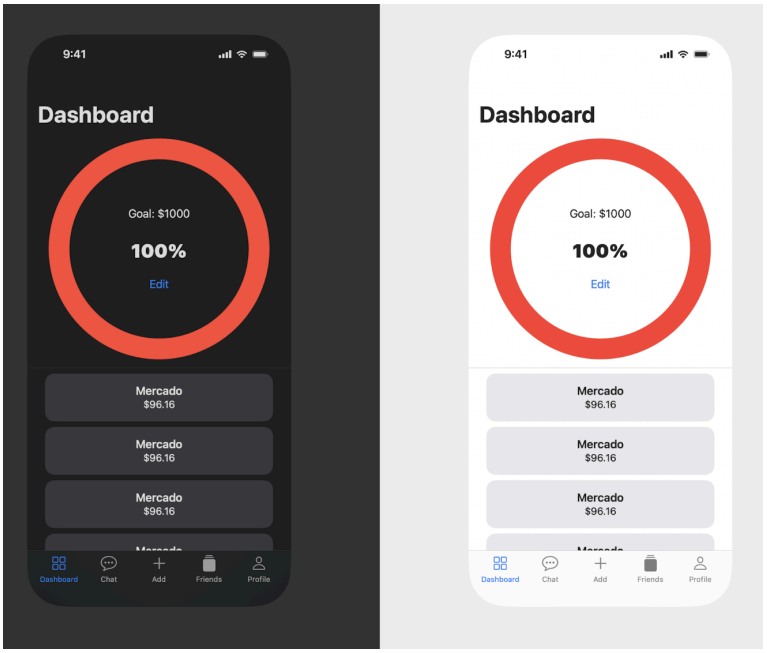

Figura #3: Diseño Vista de Dashboard

Si todo es correcto y el usuario inicia sesión correctamente la primera vista que verá de la aplicación es la vista del dashboard. Esta vista se compone de una Circular Progress bar donde el usuario ingresa un objetivo/límite de gasto mensual. Dentro del Circular Progress bar el usuario puede visualizar el valor que puso como objetivo/límite y el porcentaje de lo que ha gastado con respecto a ese límite. Abajo de este elemento se encuentra una lista de items. Estos ítems pertenecen a los gastos o ingresos que el usuario añada en la misma aplicación. El valor acumulado de esos gastos se verá reflejado en el Circular Progress bar. Aquí también tiene la opción de editar dicho límite. Esta vista se compone de un ScrollView que permite visualizar todos los ítems perteneciente a los gastos/ingresos. Todas las vistas a partir de esta se encuentra dentro de un Navigation View que permite tener el menú en la parte inferior de la pantalla y permite al usuario navegar fácilmente por las vistas de la aplicación.

Chat/Mensajería

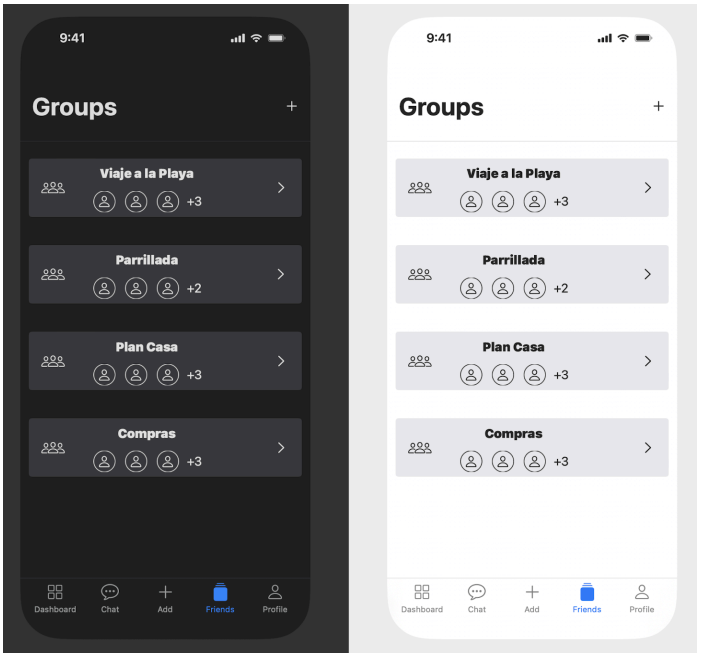

Figura #4: Diseño Vista de Chat

Esta vista corresponde a la función de mensajería que incluye la aplicación. Esta se compone de un scrollview donde el usuario puede ver todos los chat a los que él pertenece. En cada ítem/chat se puede visualizar el nombre del chat al igual que algunas fotos de perfil de los usuarios que pertenecen a dicho chat. Si el usuario aplasta en uno de esos ítems se le llevará a la vista perteneciente a ese chat. Además en esta vista se pueden crear chat grupales donde el usuario escoge el nombre y los usuario que formarán parte, para hacer esto deberá aplastar en el "+" que se encuentra en la esquina superior derecha de la aplicación.

Vista de Chat Grupal

Figura #5: Diseño Vista de Registro

Esta vista es la que el usuario va a ver si aplasta en uno de los ítems perteneciente a la vista de los chats. El usuario puede ver el título del chat así como también el valor total que se gastó en ese chat. Abajo de la descripción del chat se encuentran los mensajes del chat. Los mensajes enviados por otros usuarios se verán pegados al lado izquierdo de la pantalla. Estos se mostrarán con un cuadrado donde se verá su foto de perfil, nombre y el valor pagado al respectivo usuario. Los mensajes enviados por el propio usuario se verán pegados al lado derecho de la pantalla. Al inferior de la pantalla se encuentra el contenedor donde el usuario manda el mensaje. Aquí el usuario escoge el usuario al que va a pagar. Una vez escoja el usuario procede a ingresar el valor(en números) y tendrá la opción de adjuntar una imagen, preferiblemente un comprobante de pago, y por último se encuentro el boton de enviar, que una vez se aplaste se envía el mensaje con toda la información descrita por el usuario y se actualiza la base de datos en tiempo real.

Social

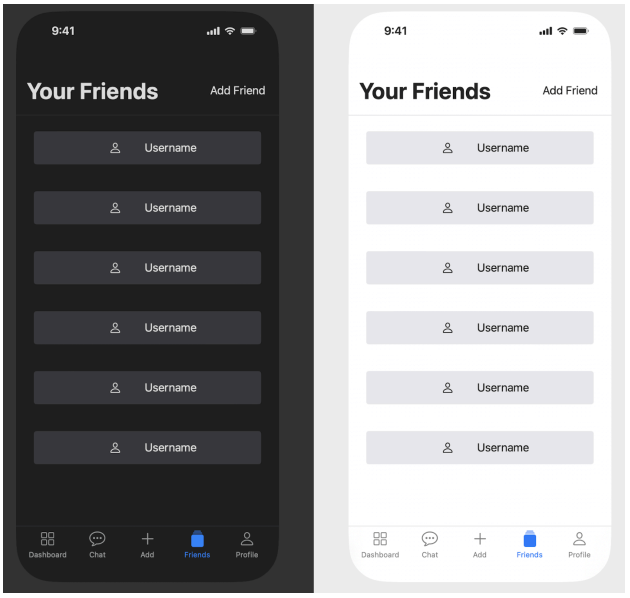

Figura #6: Diseño Vista de Amigos

Esta vista está compuesta de un Scrollview donde el usuario puede ver todos los usuario con los que tiene una relación de amistad. Cada ítem de esta vista es un contenedor donde se muestra la foto de perfil y el nombre del usuario. El usuario tiene la opción de aplastar los ítems de la lista de amigos. Una vez que aplaste se abrirá la vista sobre la información bancaria de dicho usuario. El usuario también tiene la opción de añadir una nueva amistad al aplastar "Add friend"

Información de Contacto

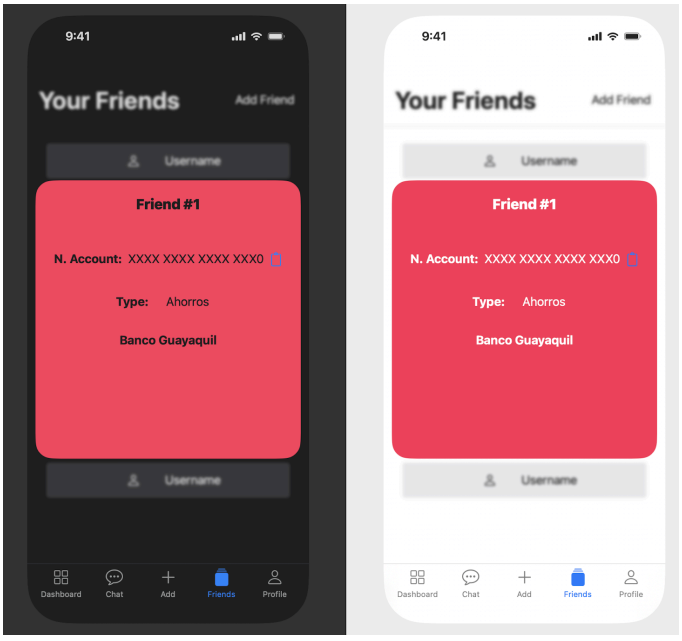

Figura #7: Diseño Vista de Información de Contacto

Esta vista corresponde a la información bancaria de una amistad. Esta vista aparece una vez que el usuario aplaste el correspondiente ítem en la vista de Social. Aqui se vera un carta de contacto donde se puede ver el número de cuenta, el tipo de cuenta y el nombre de su banco

Perfil

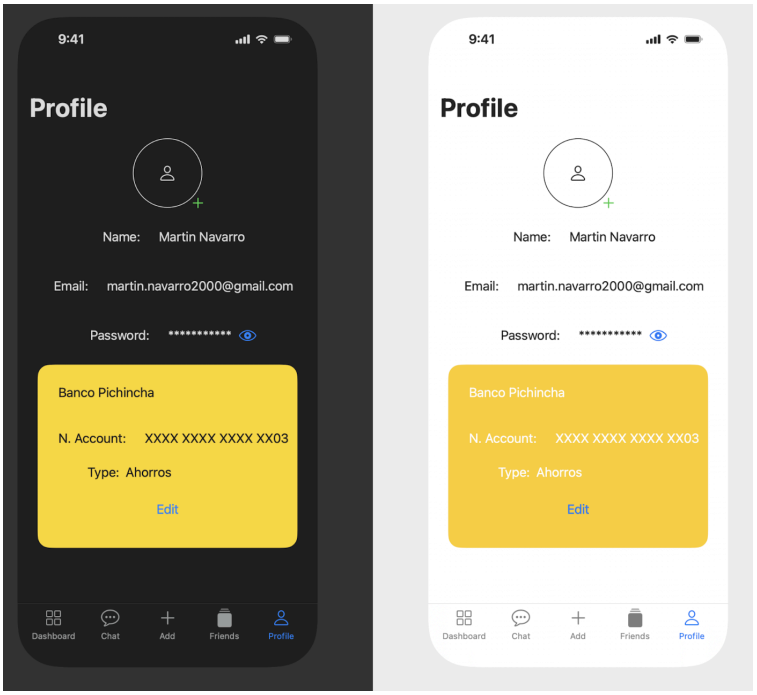

Figura #8: Diseño Vista de Perfil

Esta vista muestra la información del usuario. Aquí el usuario tiene la opción de cambiar su foto de perfil al aplastar el "+". También se muestra información importante como nombre, correo electrónico y contraseña la cual se puede visualizar si se aplasta en el icono del ojo. En esta vista se muestra el componente de información bancaria. Si el usuario aplastar el botón de editar puede cambiar la información bancaria y dependiendo el nombre del banco cambiará el color de la tarjeta de información.

Implementación del diseño utilizando SwiftUI

Basándose en el punto anterior sobre el diseño gráfico se procede a iniciar el desarrollo de la aplicación utilizando el marco de diseño SwiftUI. Se crea un perfil para cada vista declarada en el diseño gráfico y se procedió a poner los diferentes componentes, en este caso con datos predeterminados. Para estos se utilizó el repositorio de Apple para encontrar y entender los componentes que se utilizaron y qué variables acepta cada uno.

Arquitectura de la Aplicación

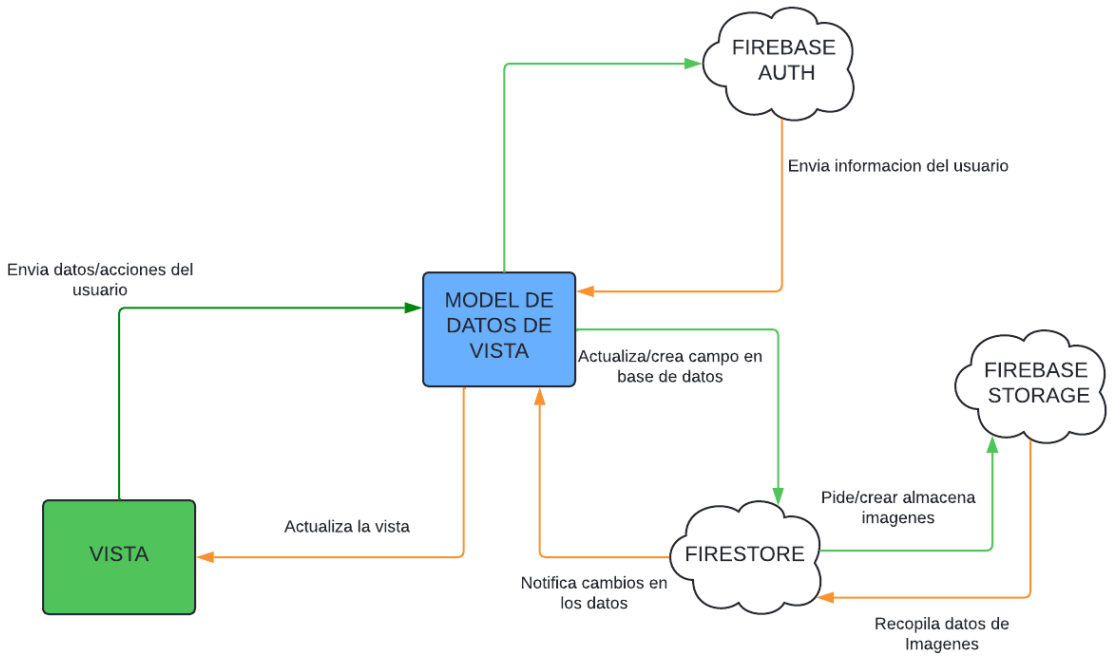

Figura #9: Diagrama Arquitectura de Aplicación

La arquitectura de la aplicación se divide en tres componentes. El primer componente corresponde a las vistas. En este componente los usuarios ingresan información en los campos y envían los datos a los modelos/ controladores de las vistas. Es importante mencionar que para cada vista existe un modelo de datos correspondiente ya que cada vista tiene diferentes funcionalidades y manejan de forma diferente la información recopilada.

El segundo componente de la arquitectura son los modelos de datos de la vista. Este componente se encarga de enlazar la información proveniente de la vista con la base de datos almacenada en Firebase. En caso de que el usuario se encuentre en la vista de registro, el modelo de datos se conecta primero con el servicio de autenticación de Firebase donde crea el usuario con los datos de los campos correspondientes a correo electrónico y contraseña. El modelo de datos crea o actualiza la información desde la aplicación hacia Firestore(base de datos). Para realizar esto Swift tiene un protocolo denominado *Codable* que permite transformar las instancias/clases hacia formatos externos como JSON, siendo esto importante ya que Firebase se caracteriza por ser una base de datos NoSQL. Este componente también se encarga de actualizar las información presentada en las vistas correspondientes, esto se logra ya que los modelo de datos son de tipo *Observable Object* por lo que detectan el cambio en los datos y actualizan las variables en la vista.

El tercer y último componente importante de la arquitectura de la aplicación es la base de datos, el cual está dentro del servicio de Firebase. Esta tiene varios servicios, la autenticación, Storage, y Firestore. A excepción de dos vistas, login y register, todos los modelos de datos de las vistas se conectan directamente con el servicio de Firestore, el cual se explicará más

adelante. En este servicio se encuentra toda la información correspondiente a los usuarios y grupos de mensajería. Este componente se encarga de recibir las información enviada por parte de los modelos de datos, en caso de que dicha información ya esté creada la base de datos se encarga de actualizar la información, caso contrario se crea un nuevo campo en la colección que corresponde dicha información. En caso de que el usuario suba una foto de perfil o mande un mensaje con un archivo multimedia el modelo de datos se conecta con Firestore y con Storage, este último almacena la foto mientras que en Firestore solamente se guarda la Url(de Storage). Este componente se encarga de notificar los cambios a los modelos de datos.

Arquitectura Base de Datos Firestore

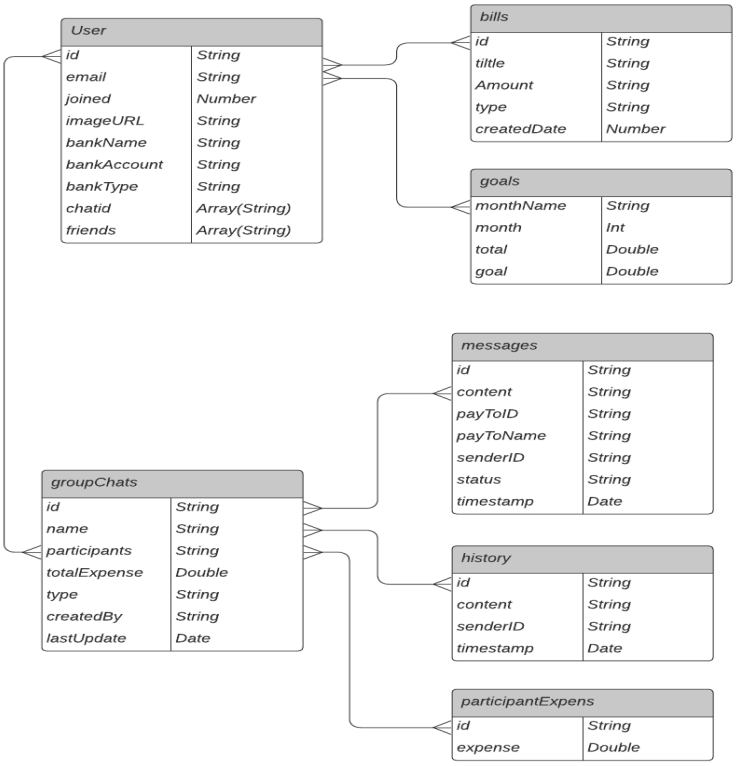

Figura #10: Diagrama Entidad-Relación Base de datos

Para la estructura de la base de datos de la aplicación se utilizó el servicio de Firebase llamado Firestore. Esta es una base de datos NoSQL por lo que los datos son almacenados como objetos tipo JSON. La base de datos se compone de dos colecciones, User y groupChats, como se puede observar en la figura #9 donde cada una de ellas tiene sus propios atributos. La colección User tiene dos subcoleccion llamada goals y bills las cuales corresponden a los gastos y objetivos personales del usuario. Estas dos subcolecciones comparten una relación muchos a muchos con la colección principal de User ya que un usuario puede tener muchos ítems por partes de las subcolecciones. Por el lado de la colección groupChat pasa una situación similar, esta tiene tres subcolecciones que

corresponden a los mensajes, historial y gasto por participante donde igualmente comparten una relación muchos a muchos con la colección principal. Es importante notar que entre las dos colecciones principales, User y groupChats, también existe una relación muchos a muchos donde por ejemplo un usuario puede tener varios chats grupales y un chat grupal puede tener varios usuarios.

Caso de Uso

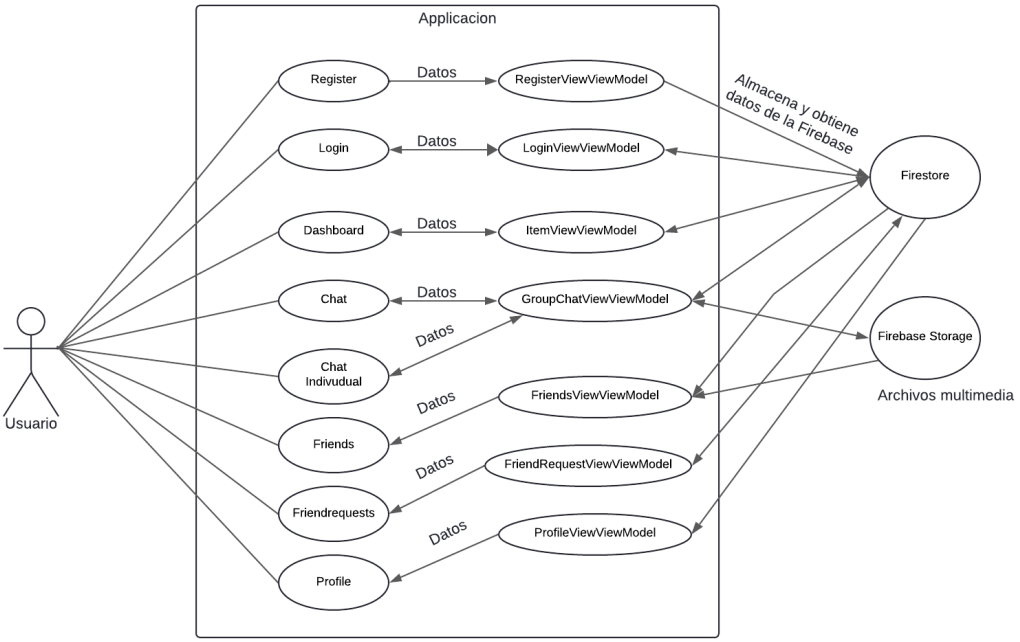

Figura #11: Diagrama de Caso de uso

Al ingresar a la aplicación el usuario se encontrará con la vista de login, si en esta caso el usuario no tiene una cuenta entonces deberá dirigirse a la vista de registro. Esta vista recopila los datos ingresados por el usuario y envía al modelo de datos de la vista que a su vez se encarga de codificar los datos y enviarlos a la base de datos de Firestore. Una vez tenga una cuenta creada el usuario podrá interactuar con las demas vista que se pueden observar en la figura #10. Cada vista tiene su propio modelo de datos correspondiente el cual se encarga de actualizar y recopilar los datos desde la aplicación hacia firestore y vice-versa. En algunos casos como la vista del perfil o la vista de los chats grupales los modelos de la vista también se encarga de conectarse con el servicio de Firebase Storage en donde se almacenan archivos multimedia, en esta caso imágenes de formato .jpg. El modelo se encarga de cargar la imagen desde este servicio y actualizar la vista con la imagen cargada.

Diagramas UML

Estructura Usuario y MonthlyGoalModel

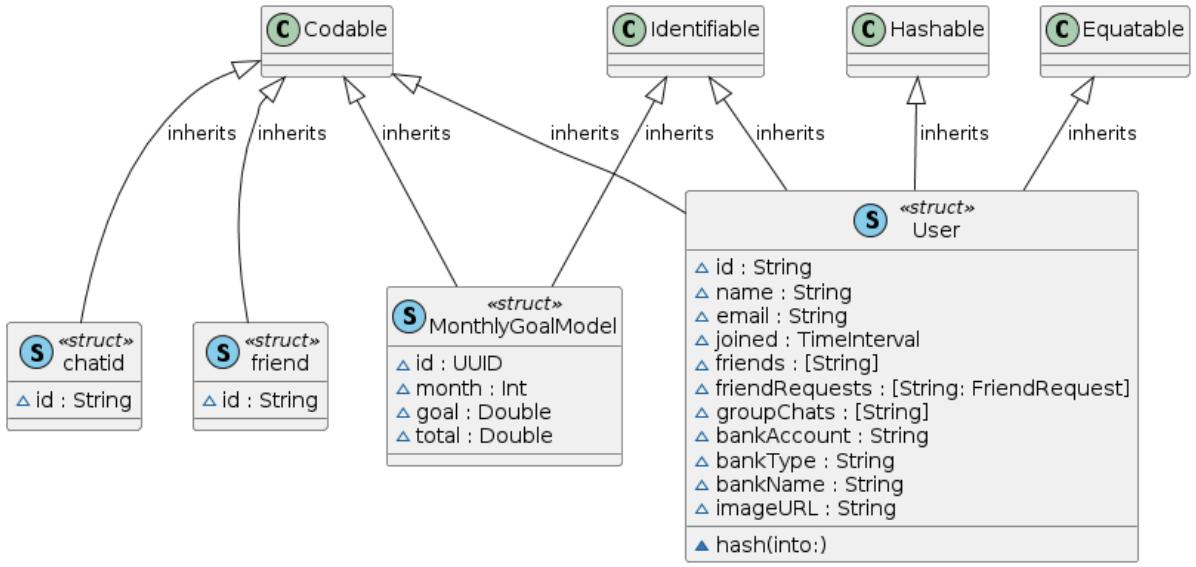

Figura #12: Diagrama UML Model Usuario, MonthlyGoalModel

La estructura de ‘User’ es el modelo de datos principal de la aplicación. En la aplicación se utiliza esta estructura para crear un usuario y poder almacenarlo en la base de datos de Firestore. Las variables de id, name, email y joined son de tipo ‘Let’, las cuales significan que se son obligatorios al momento de crear un usuario utilizando esta estructura. Las demás variables no son necesarias al momento que se crea un usuario sin embargo pueden variar dependiendo de las acciones que tome el usuario utilizando la aplicación. En el caso de la variable friendRequest se acepta un objeto de tipo ‘FriendRequest’, el cual es un diccionario que almacena las solicitudes de amistad. Cada clave es un ID de solicitud único.

En el caso de MonthlyGoalModel también se lo define como una estructura de datos en donde el usuario pone las variables acerca de su información bancaria. Esta estructura se inicializa con valores default al momento de crear un usuario, sin embargo se puede cambiar los valores y actualizarlos en la base de datos.

Es importante notar que las estructuras ‘User’ y ‘MonthlyGoalModel’ heredan ciertos protocolos provenientes de Swift. En el caso de ‘User’ hereda los protocolos de Codable, Identifiable, Hashable y Equitable, mientras que ‘MonthlyGoalModel’ hereda Codable e Identiable. Codable es un protocolo fundamental que provee Swift, el cual se compone de dos protocolos más pequeños denominados ‘Encodable’ y ‘Decodable’. Para el caso de ‘Encodable’ permite que los datos sean convertidos(codificados) a un formato externo como JSON, mientras que ‘Decodable’ permite decodificar desde un formato como JSON a una

instancia o clase. Esto es importante ya que Firebase Firestore es una base de datos NoSQL por lo que los datos se guardan en un formato JSON. Los protocolos de ‘Hashable’ y ‘Identifiable’ permiten que los datos sean utilizados como diccionarios y un identificador único que permite que los datos sean utilizados por las vistas en componentes como listas.

Estructura GroupChat y Message

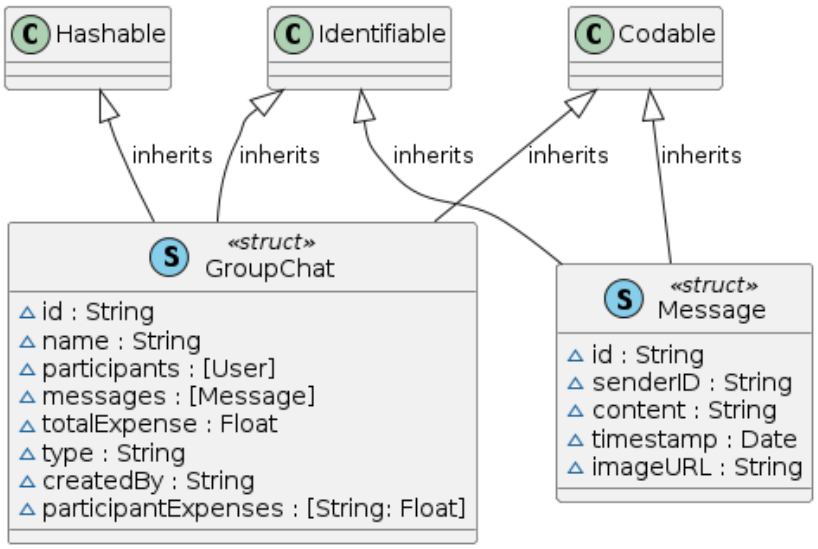

Figura #13: Diagrama UML GroupChat y Message

La estructura GroupChat y Message representan el modelo de datos que se utilizan para estructurar el servicio de Chat en tiempo real. Cada GroupChat tiene un id que se almacena en la base de datos. Tiene un lista de tipo Message el cual contiene el id del usuario que envió el mensaje el cual se obtiene leyendo de un objeto tipo ‘User’. Las demás variables como nombre, participantes, etc.. pueden ser variadas por los usuarios con valores de su preferencia. Algunos datos como Total Expense se inicializa con un valor de 0 al momento de crear sin embargo el usuario tiene la opción de cambiar dicho valor.

Clase RegisterViewViewModel

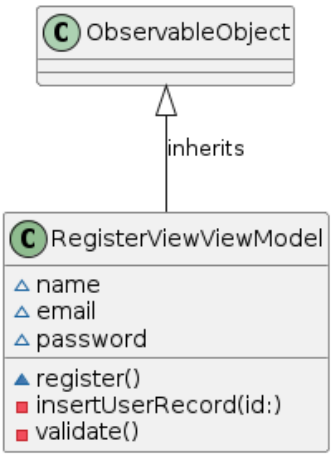

Figura #14: Diagrama UML RegisterViewViewModel

Este modelo de vista se encarga de controlar los datos de la vista de Registro. Tiene una función pública de register() la cual es la función que se encarga de insertar el usuario, específicamente, el correo electrónico y contraseña en el servicio de autenticación de Firebase. Dentro de esta se función se llaman a otras dos funciones del modelo de vista, las cuales son privadas. La función insertUserRecord() se encarga de insertar el usuario en la base de datos de Firestore. La función de validate() se encarga de verificar que el valor insertado en el campo de correo electrónico sea válido y que la contraseña inserta sea mayor que 8 caracteres.

Clase LoginViewViewModel

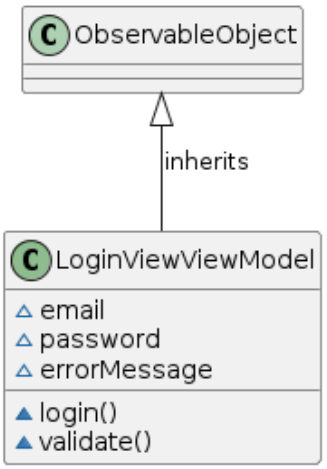

Figura #15: Diagrama UML LoginViewViewModel

El modelo de datos de la vista LoginViewViewModel se encarga de controlar los datos de la vista de Login. Esta clase tiene dos funciones, la primera función login() utiliza el método de la librería de FirebaseAuth denomina signIn para iniciar sesión en la aplicación y pasar a las

siguientes vistas. Dentro de la función de login() se llama a validate() la cual controla que los campos de la vista de correo electrónico y contraseña sean válidos y no estén vacíos.

Clase ProfileViewViewModel

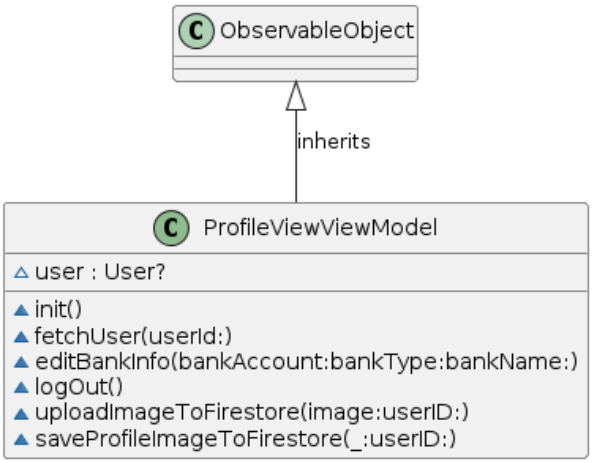

Figura #16: Diagrama UML ProfileViewViewModel

La clase ProfileViewViewModel se encarga de controlar los datos que se presentan en la vista del perfil del usuario. Esta clase tiene algunas funciones que permite obtener y editar los datos del usuario. La primera función de fetchUser() se encarga de obtener y decodificar la información del usuario desde la base de datos de Firestore. Una vez que se obtiene la información se asigna dicha información a una variable de tipo 'User', esto permite que dentro de la vista correspondiente se puede obtener la información correspondiente a dicho usuario. La función de editBankInfo() permite al usuario editar sus datos sobre la información bancaria. En la vista de perfil se llama a esta función y se asignan las variables que el usuario ingrese en los campos de su cuenta bancaria. Las funciones de uploadImageToFirestore() y saveProfileImageToFirestore() permite al usuario subir una imagen ya sea desde la librería o desde la cámara del dispositivo a servicio de Firebase llamado Storage. Este servicio permite guardar imágenes y otro tipo de formatos de multimedia, una vez que la imagen se almacene en este servicio se genera un URL correspondiente al usuario y se actualiza el campo de imageURL del usuario en Firestore con el URL generado. Por último se tiene la función de logOut() la cual utiliza el método de .signOut() de la librería de Firebase Auth para cerrar la sesión del usuario.

Clase GroupChatViewViewModel

Figura #17: Diagrama UML GroupChatViewViewModel

Dentro de la clase GroupChatViewViewModel se encuentran varias funciones que permiten el funcionamiento correspondiente con el chat de tiempo real y su creación.

- **createGroupChat(name,participants, type, createdBy):** Esta función permite al usuario crear un chat grupal. El usuario debe dar obligatoriamente un nombre al grupo y escoger el tipo de chat que desea crear. Existen dos tipos: Pagos individuales, donde el administrador(creador del grupo) escoge cuánto debe pagar cada participantes, y pago igualitario, donde el valor a pagar es dividido entre el gasto total y el número de participantes. Esta función se encarga de ingresar y actualizar la información en la base de datos Firestore bajo la colección de groupChats.
- **Update Participants(chatId, participant, expense) y fecthParticipantExpense(group):** Estas funciones se encarga de controlar los gastos de cada usuario, una vez que el usuario realice un pago esta función se encarga de actualizar el valor en la vista y en la base de datos.
- **FetchGroup() y FetchFriends:** La función fetchGroup se encarga de cargar toda la información desde Firestore de los grupos a los que el usuario pertenece, mientras que la función de fetchFriends() se encarga de cargar la información de los usuarios/ participantes que pertenece a dicho grupo.
- **SendMessage(chatID, senderID, content):** Esta función toma como parámetros el id del chat, el id del usuario que envía el mensaje y el contenido del mensaje. Con estos parámetros la funciones crear un documento con dicha información en Firestore dentro de el documento del grupo actual.

- **ReceiveMessages(chatID):** Esta función es la que se encarga de recopilar los mensajes y sus consentidos desde la base de datos y almacenar en tiempo real utilizando un listener para que se pueda presentar dicha información en la vista de SwiftUI. Cada vez que la función encuentre un cambio en la base de datos, esta se actualiza y al mismo tiempo actualiza la vista con la información nueva.
- **AddToExpense() y AddToUserExpense():** Estas funciones se encargan de actualizar la información de los gastos personales de usuarios los cuales son correspondientes a la colección de 'bills'. Una vez que un usuario manda el mensaje, o pague se va a actualizar la información de gastos personales tanto del usuario que mandó el mensaje y del usuario al que se le pagó o mando el mensaje.
- **uploadImageToFirestore() y saveProfileImageToFirestore():** Estas funciones se asemejan a las funciones declaradas en el ProfileViewViewModel. En este caso las imágenes se suben a una dirección diferente dentro de Firebase Storage. El imageURL se actualiza en la colección de mensajes dentro de Firestore.

Conclusión

Los objetivos planteados se cumplieron adecuadamente. Esta aplicación satisface la necesidad de gestionar los gastos/ingresos personales de los usuarios. Además permite crear grupos para poder compartir gastos de una manera justa y equitativa. La aplicación fue creada en Swift, el cual se caracteriza por mantener la potencia de Objective C reduciendo su complejidad de aprender y utilizar. Utilizando librerías se pudo conectar la aplicación con una base de datos almacenada en Firebase. Se crearon modelos de datos que permite controlar los datos que ingresan y salen de las vista desde y hacia la base de datos, esto permitió que se puede crear relación de amistad entre los diferentes usuarios que tenga un cuenta en la aplicación. A partir de esto se pudo crear el chat de tiempo real donde los usuarios pueden agregar participantes así como también editar diferentes componentes del grupo como el gasto total o el gasto individual de cada participante.

En un plano personal y profesional esta aplicación me permitió involucrarme más a un nuevo lenguaje de programación como lo es Swift. El aprendizaje durante el proyecto me permitió tener mejor conocimiento en cuanto al desarrollo de aplicaciones y la conectividad de datos almacenados en la nube el cual corresponde a un campo importante en Ingeniería en Ciencias de la computación y en las herramientas que utilizamos diariamente.

Limitaciones

La aplicación al estar escrita 100% en el lenguaje Swift no puede ser descargada en dispositivos Android o de otros sistemas operativos que no pertenezcan a iOS. Es importante notar que esta aplicación solamente es una simulación de pagos, lo que significa que los mensajes correspondientes a pagos no están relacionados directamente con una tarjeta bancaria, transacción o cuenta bancaria.

Mejoras Futuras

Para mejoras futuras de la aplicación se debe considerar algunos cambios. Uno de las mejoras principales es el incremento de seguridad al momento de crear una cuenta, esto quiere decir que se incorpore servicio de identificación de correo electrónico para poder comprobar que el correo electrónico ingresado por el usuario sea verdadero, además de incorporar otros métodos para iniciar sesión como por ejemplo utilizando el inicio de sesión vía Email o via Facebook.

Otra mejora a considerar corresponde a la funcionalidad de los chats, específicamente en la parte de asignar los gastos de los participantes. La mejora sugiere que se añada mejor control de detección en caso de que la suma de la asignación de los pagos sea menor que el gasto total del grupo. Si es así entonces la aplicación deberá dar un aviso o notificación.

A un futuro también se debe considerar implementar servicios de pagos reales, como por ejemplo consumir el API de pagos de compañías como PayPal y/o Payphone. Por último también se puede considerar modificar la idea de los chats, donde los usuarios puedan mandar mensajes que no necesariamente contengan pagos.

REFERENCIAS BIBLIOGRÁFICAS

[1] Inc,. *Swift*. Apple Developer. https://developer.apple.com/swift/

[2] Hudson, P. (2024, April 11). *Introducing MVVM into your Swiftui Project*. Hacking with Swift. https://www.hackingwithswift.com/books/ios-swiftui/introducing-mvvm-into-your-swiftui-project

[3] Google. *Agrega firebase a Tu Proyecto de Apple | firebase for Apple Platforms*. Google. https://firebase.google.com/docs/ios/setup?hl=es-419

[4] Google. (n.d.-b). *Download files with cloud storage on web | cloud storage for firebase*. Google. https://firebase.google.com/docs/storage/web/download-files

[5] Google. (n.d.-b). *Upload files with cloud storage on web | cloud storage for firebase*. Google. https://firebase.google.com/docs/storage/web/upload-files

[6] Power, N. (2018, June 21). *Getting started with Firebase on IOS: Part 1*. Medium. https://medium.com/@niamhpower/getting-started-with-firebase-on-ios-part-1-612af4bcabd6

[7] *Swiftui*. Apple Developer Documentation. https://developer.apple.com/documentation/swiftui/

[8] Ytor. *Firebase tutorial: Real-time chat*. kodeco.com. https://www.kodeco.com/22067733-firebase-tutorial-real-time-chat

Anexo A: Código Fuente

El código fuente se encuentra en el siguiente enlace: https://github.com/martinnavarro00/proyectoIntegrador. Al ser un código escrito 100% en Swift se recomienda abrir el código en un IDE que soporte el desarrollo de este lenguaje, preferiblemente en XCode. Es importante notar que el repositorio contiene una archivo .plist(GoogleService-Info.plist) el cual mantiene la conexión entre la aplicación y la base de datos de Firebase. Los archivos correspondientes al layout se encuentra bajo la carpeta de de *Views*, mientras que los archivos correspondientes a los modelos de la vista se encuentra bajo de carpeta de *ViewModel.*