

# **Creació d'una aplicació des de zero**

Projecte Integrat de Software

Segona entrega: Treball realitzat fins al moment



Raul Asins Hidalgo

Víctor González Porras

Víctor Sort Rubio

David Fernández Gómez

Soufiane Lyazidi Ahrillou Abraray

# ÍNDEX

<b>1. INTRODUCCIÓ.....</b>	<b>3</b>
<b>2. FEINA REALITZADA.....</b>	<b>4</b>
<b>3. INTERFÍCIES D'USUARI REALITZADES.....</b>	<b>5</b>
3.1. Vista d'accés.....	5
3.2. Vista d'inici de sessió.....	6
3.3. Vista de registre.....	6
3.4. Vista de validació de telèfon.....	7
Figura 4: Vista de validació de telèfon.....	8
3.5. Vista de cerca de grups.....	8
3.6. Vista de creació de grups.....	9
3.7. Vista de visualització de grup.....	10
3.8. Vista de detalls de la despesa.....	12
3.9. Vista de perfil.....	13
3.10. Vista d'editar perfil.....	14
3.11. Vista canviar contrasenya.....	15
3.12. Justificació dels layouts escollits.....	15
3.12.1 Constraint Layout.....	15
3.12.2 Linear Layout.....	15
3.12.3 Card View.....	15
3.12.4 Recycle View.....	15
3.12.5 Frame Layout.....	15
3.12.6 Relative Layout.....	15
<b>4. DISSENY.....</b>	<b>18</b>
4.1. Model overview.....	18
4.2. Viewmodel + view overviews.....	22
4.3. Base de dades.....	29
<b>5. FEINA PENDENT A FER.....</b>	<b>32</b>
<b>6. DIAGRAMA DE CLASSES.....</b>	<b>34</b>

## 1. INTRODUCCIÓ

L'objectiu d'aquesta segona tramesa és presentar tot el que hem avançat en el desenvolupament de l'APP fins al moment. En aquest informe mostrarem tot allò en què hem estat treballant durant les últimes setmanes, explicant les vistes fins ara creades, les funcionalitats implementades, les classes que integren fins ara l'APP i com es relacionen i la funció que compleix cada una d'elles, així com els canvis que hi ha hagut respecte a la primera entrega.

Al final de l'informe es troba un resum de tot el que ens queda per arribar al nostre objectiu final, satisfent al 100% els nostres objectius inicials. D'aquesta manera és fàcil visualitzar a on estem des de l'inici i on volem arribar, servint-nos també a nosaltres mateixos com a guia.

Mencionar que, per iniciar sessió sense crear-se un compte propi, es poden fer servir les credencials de l'usuari amb número de telèfon 612 34 56 78 i contrasenya 131313.

## 2. FEINA REALITZADA

És difícil d'aproximar, però si haguéssim de dir un percentatge del que tenim fet ja de l'aplicació, diríem un 70%, major que el 50% exigint per aquesta entrega. Com es veurà més endavant tenim acabades casi totes les vistes, i, a part de polir detalls, només queda acabar d'implementar la lògica interna.

En el moment d'entrega d'aquest informe, tenim realitzades les següents vistes i funcionalitats:

- La vista per defecte que permet escollir entre registrar-se o iniciar sessió.
- La vista per iniciar sessió amb la lògica interna de comprovar que l'usuari i contrasenya són correctes.
- La vista per registrar-se, a falta de comprovar que els camps obligatoris no estiguin buits i la contrasenya compleix els estàndards requerits.
- La vista amb el codi necessari per verificar el número de telèfon introduït, per tant, també està implementada la vista per introduir el codi de verificació.
- S'ha implementat la vista que mostra la llista dels grups de l'usuari i els botons de notificacions, tot i que no ha estat implementada la seva funcionalitat.
- S'ha implementat la vista per afegir un nou grup a falta d'habilitar l'opció d'afegir imatge d'un grup.
- S'ha implementat la vista per veure les despeses i detalls d'un grup concret, encara sense poder saldar els deutes, amb el desplegable i vista per veure els detalls d'una despesa concreta, encara sense poder-se editar les dades d'un grup.
- S'ha implementat part de la vista del perfil, mostrant el correu, el número de telèfon, el botó d'editar perfil i el botó d'ajudar el nostre projecte.
- S'ha implementat part de la vista d'editar perfil, on es pot tancar la sessió i canviar la contrasenya, obrint la vista de canviar contrasenya que també ha estat implementada.

A l'apartat 5 explicarem en detall el què ens falta per implementar.

### 3.INTERFÍCIES D'USUARI REALITZADES

Mentre anem explicant per sobre cada vista mencionarem els layouts utilitzats i les raons del seu ús.

#### 3.1. Vista d'accés

Vista principal que s'obre en iniciar l'aplicació. Trobem el logo de l'app i tres botons: un per descarregar els terminis i condicions, un per registrar-se i un per accedir-hi mitjançant un log in.

Fem servir un Constraint Layout, ja que el posicionament dels elements no afavoreix en gran manera, cap dels layouts vists.

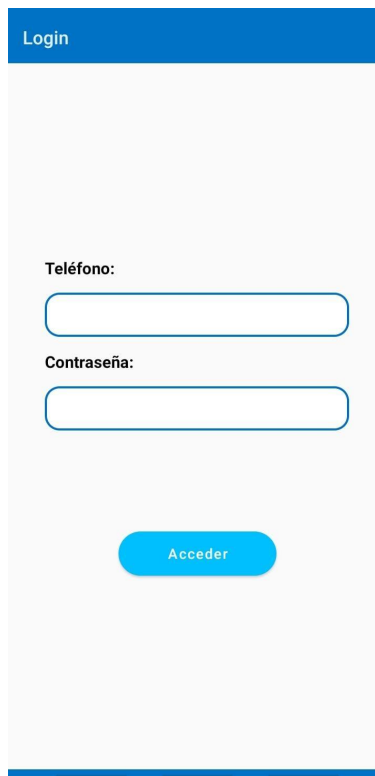


Figura 1: Vista d'accés

### 3.2. Vista d'inici de sessió

Vista per iniciar sessió mitjançant el número de telèfon i una contrasenya. Trobem dos EditText i un botó per a confirmar l'accés. Als EditText se'ls ha afegit una vora i el seu fons ha estat emplenat de blanc pur, a diferència de com es pot veure als mockups, ja que hem cregut que seria més intuïtiu i estètic i, per tant, resultaria en una millor experiència d'usuari.

Està clar que el Linear Layout seria el més adient per aquesta vista, ara bé, per la seva simplicitat hem optat simplement per un Constraint Layout, però per la següent entrega es preveu millorar-ho.



The image shows a mobile app login screen. At the top is a blue header bar with the word "Login" in white. Below the header, the background is light gray. There are two input fields: the first is labeled "Teléfono:" and the second is labeled "Contraseña:". Both fields are white with a thin blue border. Below the input fields is a blue button with the word "Acceder" in white. At the very bottom of the screen is a solid blue bar.

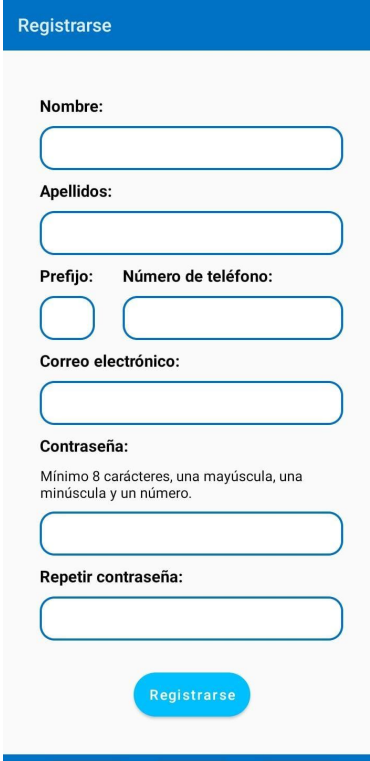
Figura 2: Vista d'inici de sessió

### 3.3. Vista de registre

Layout per a fer el registre d'un nou usuari. Trobem diversos EditText per a poder omplir la fitxa de l'usuari amb la informació següent: nom, cognom, prefix, telèfon, correu

electrònic, contrasenya i confirmació de contrasenya. Com hem mencionat al 3.2., hem canviat la vista dels EditText per una millor UX.

Està temporalment implementat amb un Constraint Layout, però per la versió final de l'APP serà implementat amb un Linear Layout, ja que l'estructura de llista vertical clarament, és totalment adient per aquesta vista. En el millor cas es farà amb diversos fragments, un per cada dada sol·licitada.



The image shows a registration form titled "Registrarse" in a blue header. The form is set against a light gray background and contains the following elements:

- Nombre:** A single-line text input field.
- Apellidos:** A single-line text input field.
- Prefijo:** A small, rounded square input field.
- Número de teléfono:** A single-line text input field.
- Correo electrónico:** A single-line text input field.
- Contraseña:** A single-line text input field. Below it, a note reads: "Mínimo 8 caracteres, una mayúscula, una minúscula y un número."
- Repetir contraseña:** A single-line text input field.
- Registrarse:** A blue, rounded rectangular button at the bottom center.

Figura 3: Vista registre

### 3.4. Vista de validació de telèfon

Vista per a confirmar el telèfon. L'usuari rep un missatge SMS i en un EditText ha d'escriure el codi rebut i confirmar amb el botó.

El més adient hauria estat un Linear Layout, però, s'ha seleccionat un Constraint Layout, per la simplicitat de la vista.

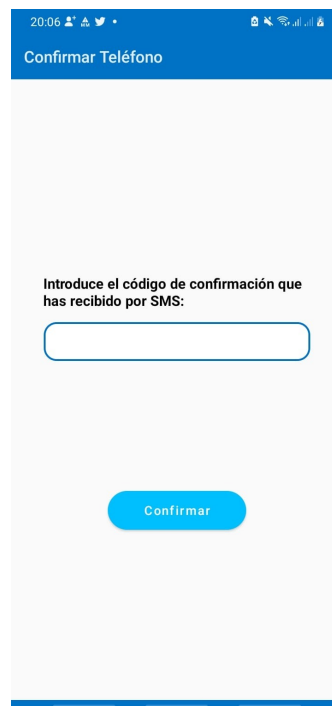


Figura 4: Vista de validació de telèfon

### 3.5. Vista de cerca de grups

Vista principal de l'aplicació, la icona de perfil porta al perfil de l'usuari, a la seva esquerra surt Hola, “nom de l'usuari”. L'apartat de novetats es pot replegar per veure el RecyclerView de CardViews de Grups. El botó flotant amb el símbol de + obre la vista per crear un nou grup.

Respecte als mockups, els botons de les notifikacions tenen una vora, ja que hem cregut que seria més intuïtiu i estètic, i, per tant, resulta en una millor experiència d'usuari. També hem eliminat el Fragment per transicionar entre els grups actuals i crear un nou grup, ja que no vam tenir en compte que aquest tipus d'estructures solen ser per vistes de prioritat/importància similar, cosa que no passava amb veure grups i crear un nou grup.

S'ha seleccionat un Linear Layout vertical, pel fet que la vista no deixa de ser una llista d'elements, per l'EditText s'ha fet servir un Relative Layout per poder posicionar la llupa per sobre de l'EditText. També s'ha fet servir un RelativeLayout pel RecyclerView i el



botó flotant, ja que el volíem per sobre dels elements del RecyclerView. A cada CardView s'ha fet servir un Linear Layout horitzontal i a dins dos Linear Layouts Verticals pels tres TextViews en llista, això, perquè el Table Layout (que seria el més adient) no permet que un element ocupi més d'una fila.



Figura 5: Vista de cerca de grups

### 3.6. Vista de creació de grups

Vista per crear un nou grup, compta amb tres EditText pel nom del grup, descripció i per afegir membres per número de telèfon, amb un botó a sobre (+) per afegir-los. Un cop polsat, es mostren com a TextViews a sota. Tres botons per afegir la imatge del grup, afegir membres des de contactes i crear el grup.

En aquest cas tret el tema ja mencionat de les vores, respecta bastant el disseny original dels MockUps.

S'ha fet servir un Linear Layout vertical, ja que la vista no deixa de ser una llista d'elements, un Relative Layout per l'EditText d'afegir membres pel número de telèfon, per així poder posicionar el botó de + per sobre de l'EditText.




Figura 6: Vista de creació de grups

### 3.7. Vista de visualització de grup

Vista que mostra la informació d'un grup concret. Trobem el nom del grup, una petita descripció, una imatge, tres botons i un FragmentContainerView. Els botons serveixen per saldar deutes, el primer, i els altres per transicionar entre el Fragment que conté el RecyclerView de les despeses del grup, el botó flotant per crear una nova despesa i el Fragment que conté el RecyclerView dels balanços. Si polsem sobre una despesa, es desplega la CardView amb els detalls de la despesa.

Respecte als MockUps, per la mateixa raó d'abans, hem eliminat la doble pestanya entre Despeses i crear una nova despesa. En aquest cas la doble pestanya feta amb fragments s'ha fet per les Despeses i el Balanç dels usuaris del grup. Canviar la foto de grup ja no serà un botó, ja que no té sentit que tingui tanta importància com el botó de saldar deutes, que és el botó principal, sinó que serà una tasca que es podrà fer en l'apartat d'editar grup, en cas que afegeixi la funcionalitat.

Altres cops s'ha fet servir un Linear Layout per les mateixes raons que pels casos anteriors, els Linear Layouts anuats pel nom, imatge, etc. del grup s'ha fet de manera anàloga que pels CardViews dels grups.

Pel Fragment de Despeses s'ha fet servir un Frame Layout, ja que és el que s'assigna per defecte als fragments, però a dins hi hem fet servir un RelativeLayout per tenir el botó d'afegir despesa per sobre del recycler de despeses del grup. Pel CardView de les despeses s'ha fet servir un Linear Layout vertical per poder tenir una part oculta de la CardView i només mostrar el concepte i el preu. Per la part detallada s'ha fet servir un Linear Layout i a la part de TextView i Button. S'ha fet servir un Linear Layout vertical, per les mateixes mencionades a apartats previs això és perquè, el Table Layout no permet que els elements ocupin més d'una fila.

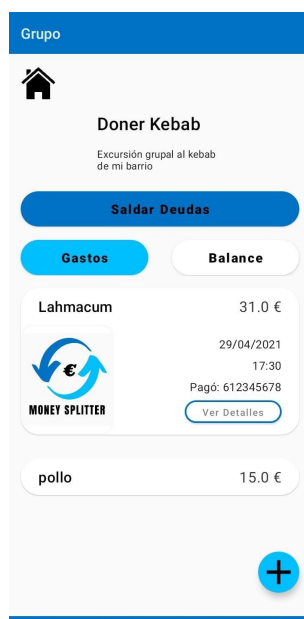


Figura 7: Vista de visualització d'un grup

### 3.8. Vista de detalls de la despesa

Vista per veure els detalls d'una despesa concreta. Mostra l'ImageView de la foto de la despesa TextView pel Títol, el cost total, la data, l'hora i qui ha pagat. També conté un RecyclerView on cada CardView és la part de cada usuari en aquesta despesa.

Aquesta vista no estava en els MockUps, ja que la part dels usuaris en la despesa es mostrava en el desplegable de la CardView de la vista anterior. Però ens hem adonat que això era inviable, pel fet que no podem controlar el nombre d'usuaris per despesa i fer-ho amb un RecyclerView seria una pèssima idea des del punt de vista d'experiència d'usuari, ja que és poc intuïtu (ningú s'espera un RV allà).

S'ha fet servir un Linear Layout, en ser una estructura de llista d'elements, pel que fa a la part de la imatge de la despesa i els detalls, s'ha fet de manera idèntica a la CardView de despeses.

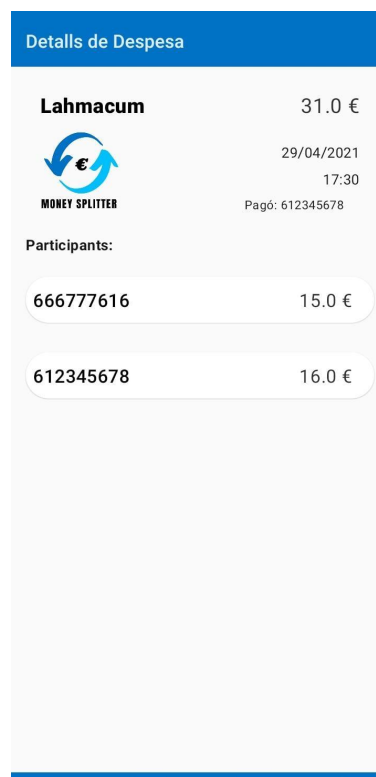


Figura 8: Vista de detalls de la despesa

### 3.9. Vista de perfil

Vista per veure el perfil de l'usuari. Conté l'ImageButton per tornar a la vista principal de cerca de grups i un CardView que conté l'ImageView de la imatge de perfil. També conté TextViews per indicar el nom i cognom de l'usuari, així com el seu ID i conté un Button per anar a la vista d'editar perfil.

Més avall, hi ha empleats diferents TextViews per fer més entenedor el contingut de la vista, juntament amb TextView semblants als EditText emprats a l'aplicació, amb la vora blava i fons blanc. En aquests EditText es mostren el correu i el número de telèfon de l'usuari. A més a més, hi ha els Buttons que permeten anar a la vista de l'històric de despeses, de l'històric de pagaments i a la pàgina web per ajudar-nos econòmicament.

S'ha fet servir un Linear Layout vertical, on dins hi ha un ScrollView, per poder desplaçar tot el contingut de la vista, perquè no cap sencer a la pantalla, i dins un Linear Layout on hi ha tots els elements esmentats anteriorment, escollit a causa de la clara vertical dels elements.



Figura 9: Vista del perfil (1)



Figura 10: Vista del perfil (2)

### 3.10. Vista d'editar perfil

Vista per a editar dades del perfil d'usuari. Trobem diferents botons que permeten canviar la imatge del perfil, canviar la contrasenya, desactivar les notificacions, tancar la sessió i eliminar el perfil d'usuari. També tenim l'ImageButton per tornar enrere a la vista del perfil i el CardView amb el ImageView de la imatge del perfil.

S'ha fet servir també un Linear Layout vertical i s'han fet els botons més allargats respecte els Mock Ups, per estètica.



Figura 11: Vista d'editar perfil

### 3.11. Vista canviar contraseña

La vista de canviar contrasenya conté tres EditText on s'han d'introduir la contrasenya actual de l'usuari i la nova contrasenya dos cops, així com TextViews explicatius. Per últim hi ha un botó per finalitzar l'activitat.

S'ha fet servir un LinearLayout vertical, on s'ha afegit tots els components esmentats.



Cambiar Contraseña

Contraseña actual:

Contraseña:

Mínimo 8 caracteres, una mayúscula, una minúscula y un número.

Repetir contraseña:

Cambiar

## **3.12. Justificació dels layouts escollits**

### **3.12.1. Constraint Layout**

El Constraint Layout és el Layout d'Android més flexible de tots que permet crear dissenys grans i complexos sense niar altres ViewGroups. Semblant al Relative Layout, les Views es relacionen entre si dins del mateix ViewGroup i el disseny principal, i és de fàcil manipulació, sense haver de manipular el codi XML.

És present a gran número de ViewGroups del nostre projecte: activity\_confirm\_mobile.xml, activity\_log\_in.xml, activity\_main.xml, activity\_profile.xml, activity\_sign\_up.xml, etc.

### **3.12.2. Linear Layout**

El Linear Layout és una distribució dels continguts de manera lineal, ja sigui vertical com horitzontal. Els continguts s'alineen en una única direcció, en el nostre cas hem fet servir una alineació vertical per a tots els ViewGroups.

Algunes de les activitats que incorporen el LinearLayout són: activity\_home\_groups.xml, activity\_home\_new\_group.xml.

### **3.12.3. Card View**

Les Cards View són layouts que permeten mostrar una informació en un format reduït i que conté aspectes més visuals, com imatges o informació molt precisa. En el nostre cas també seran interactives, ja que en pressionar una card ens donarà una informació.

Hem utilitzat una card view per a crear el group\_card\_layout, que ens permet veure una imatge del grup, el nom, la data i un petit resum del grup. Per la mateixa raó ho hem fet servir per bill\_card\_layout.xml, bill\_member\_card\_layout.xml



#### **3.12.4. Recycle View**

El Recycle View és un component de la biblioteca de suport d'Android que permet mostrar molta informació de manera eficient, contenint els múltiples Card Views sense haver de carregar-los tots si no es mostren en pantalla.

En el nostre cas, hem fet servir un Recycle View per a `activity_home_groups.xml`, on es mostren un llistat dels grups als quals pertany, però només mostrant els que ocupen l'espai indicat, per això s'ha fet ús d'un `ScrollView`. Analogament s'ha fet servir a `bills_fragment.xml`, `bills_details_activity.xml`

#### **3.12.5. Frame Layout**

El Frame Layout és bastant simple, alinea tots els elements amb la cantonada superior esquerra, fent que els components se sobreposin a no ser que els fem invisibles. Útil per a col·locar imatges o botons únics a totes les pantalles, però sense gaires més objectes dins del mateix layout

En el nostre cas fem servir el layout en les següents aplicacions: `fragment_bills.xml`. Ja que per defecte es la eina que encapsula els fragments, però internament segons la necessitat s'han fet servir altres layouts, segons s'ha considerat necessari. (RecyclerView en el nostre cas)

#### **3.12.6. Relative Layout**

Permet posicionar els elements respecte als altres.

En el nostre cas s'ha fet servir Pels botons flotants d'afegir grup i despesa, ja que els volíem per sobre del Recycler dels grups/despeses. Així com pel EditText d'afegir usuaris, ja que volem el botó de + per sobre del EditText

## 4. DISSENY

### 4.1. Model overview

La part del model és la part del programa que ha de contenir les dades amb les quals el sistema treballa. El que es comunica amb la base de dades i prové a la View (a través del ViewModel) les dades requerides per aquest últim.

En el nostre cas hem implementat la classe Bill per encapsular la informació essencial de les despeses que hi haurà. Aquesta informació és un identificador String únic, el nom de la despesa que li haurà donat l'usuari que l'ha creada i el cost total que ha tingut aquesta despesa, així com l'ID de l'usuari que l'ha pagada, l'hora i la data.

La classe Group ha estat implementada per encapsular la informació més necessària d'un grup concret. Conté un identificador, en format String, únic per cada grup, el nom d'aquest grup donat pels usuaris al crear-lo, la descripció que li han donat i la data en la qual ha estat creat, així com un String que conté una URL que identifica la foto associada a aquest grup. Cal mencionar que això és així de forma temporal, ja que actualment les fotos dels grups només poden ser a partir de URLs externes, però per l'entrega final no es guardarà la URL i es tindrà una URI o altres opcions per integrar les funcionalitats de seleccionar com a foto de grup una de la galeria de l'usuari o realitzada amb la càmera.

La classe User s'ha implementat per permetre'ns tenir encapsulada la informació d'un usuari concret, que conté un identificador únic per cada usuari, que consisteix en el seu número de telèfon +"@moneysplitter.com". Això s'ha dut a terme així, ja que, així facilita la gestió dels usuaris, en el sentit que ens podem despreocupar de la gestió dels intents d'usuaris que intenten fer ús del mateix id. Com verifiquem el número de telèfon, en ser part de l'id això ens assegura que cada ID d'usuari serà únic, encara que no obliguem a l'usuari a verificar el seu correu. No s'han implementat els registres de diverses vies, ja que Firebase no permet associar al mateix usuari un correu, telèfon,

etc. Però això ho tractarem en més detall a l'apartat de bases de dades. Seguint amb la classe User també inclou el nom de l'usuari, els seus cognoms, el seu número de telèfon, amb el seu prefix, la seva contrasenya i un String amb l'URL de la seva foto de perfil, que com hem mencionat de moment les fotos només són a partir d'URL, però en la següent entrega ho modificarem. A la classe User també hem fet que es pugui canviar la contrasenya, accedint tant a la base de dades com a l'eina d'autenticació.

Les classes anteriors implementen la interfície Serializable, perquè així ens permet a la Vista passar entre Activitats a través d'intents els objectes sencers, hem fet servir aquesta opció en comptes de passar l'ID i buscar el Group/User/Bill a la base de dades per evitar el temps extra que triguen les dades a carregar a la Vista en realitzar l'accés a la base de dades. Preveiem, en futur poder fer servir la classe Parcelable, que és una eina creada explícitament per transmetre objectes entre activitats.

Respecte a la classe UserBill, engloba la informació del que ha gastat un usuari concret en una despesa (Bill) concreta, per això conté l'ID del Bill on s'ha donat la despesa, el cost d'aquesta per l'usuari i l'ID de l'usuari que ha tingut aquest cost en la despesa.

Ara, les classes acabades amb Repository s'encarreguen de connectar amb la base de dades, per subministrar la informació. Ara tractarem en línies generals, com aquestes classes realitzen aquesta comunicació (que en bona part és similar).

Pel que fa a BillRepository, hi apliquem el patró de disseny Singleton, per així tenir una única instància, ja que per la seva essència no té sentit arribar a tenir més d'una instància del BillRepository. Com a atributs tenim un array de Strings que conté els IDs d'un grup concret i un FirebaseFirestore per poder accedir a la base de dades, obtenint d'aquest últim, la seva única instància al constructor. Tenim altres atributs que anirem mencionant més endavant. Aquest grup servirà per funcions que consisteixen a permetre filtrar les despeses d'un grup concret, en comptes d'importar totes les despeses, que no ens interessa.

Aquesta classe té principalment dues funcionalitats: carregar a l'array de despeses d'un grup els ID de totes les despeses que hi té associades i omplir un array passat per

paràmetre amb un objecte `Bill` per cada despesa del grup. Com l'accés a la base de dades es realitza en segon pla, moltes accions requereixen cridar listeners un cop acabades, ja que si un `ViewModel` sol·licita al model una llista de dades de la base de dades no pot realitzar accions directament amb aquestes dades i, per tant, ha d'esperar que aquestes dades hagin estat carregades del tot. Per això, fem servir listeners de manera que la tasca a portar a terme un cop carregades les dades l'afegim en el mètode `handler` del listener. Per això hem definit dues interfícies a la nostra `BillRepository` que faran de listeners. Per poder cridar aquests listeners en determinats mètodes per cada interfície tenim un atribut que és un `ArrayList` de tots els listeners que han implementat aquella interfície, així com un mètode per anar afegint els listeners creats a aquest array (previsiblement des del `ViewModel`).

El primer listener és `OnLoadGroupBillsIDListener` del qual es criden totes les seves implementacions un cop s'ha carregat a l'array d'IDs de despeses d'un grup concret (al mètode `setGroupBillsbyID`), que tenim com a atribut, tots els IDs de les despeses. El segon és `OnLoadGroupBillsListener`, per aquest es criden totes les seves implementacions al mètode `loadGroupBills` al listener de l'`onComplete` de l'activitat de carregar les despeses un cop ja hem filtrat per les despeses que no ens fan falta (les que són del grup sol·licitat, és a dir, les que el seu ID està a l'array d'IDs del grup sol·licitat).

Pel que fa a `GroupRepository` tenim una única instància, tornant a aplicar el patró `Singleton` amb `eager initialization`, per raons anàlogues a les mencionades a l'explicació de `BillRepository`. En referència als atributs de `GroupRepository`, la situació és similar a `BillRepository`, ja que tenim un atribut de tipus `Firestore` que ens permetrà accedir a la base de dades. També tenim un atribut que conté un `ArrayList` amb els IDs dels grups d'un usuari concret, aquest es podrà carregar a través del mètode `setUserGroupsbyID`, que se li passa l'ID de l'usuari que es volen carregar els IDs dels grups. Posteriorment mencionarem la resta d'atributs.

Aquesta classe té dues funcionalitats principalment, bastant relacionades entre elles, i aquestes són carregar a l'array de grups d'un usuari els ID de tots els grups que hi té

associats i omplir un array passat per paràmetre, afegint només els grups que el seu ID està a la llista, per carregar tots els grups existents sense filtre per usuari hi ha una funció idèntica, però que no realitza la comprovació de si el grup és de l'usuari. També tenim la funció d'obtenir una instància de Group que representa el grup amb ID passat per paràmetre, accedint a la base de dades per obtenir les dades del grup.

Per les mateixes raons mencionades prèviament fem servir listeners, que cridem en acabar les tasques de càrrega des de base de dades en dur-se a terme en segon pla. En aquest cas tenim definides 3 interfícies que faran de listeners.

El primer listener és `OnLoadUserGroupsIDListener` del qual es criden totes les seves implementacions un cop s'ha carregat a l'array d'IDs de grups d'un usuari concret (al mètode `setUserGroupsbyID`), que tenim com a atribut, tots els IDs de les despeses. El segon és `OnLoadUserGroupsListener`, per aquest es criden totes les seves implementacions al mètode `loadUserGroups` al listener de l'onComplete de l'activitat de carregar els grups un cop ja hem filtrat pels grups que no ens fan falta (els que són de l'usuari sol·licitat, és a dir, els que el seu ID està a l'array d'IDs del grup sol·licitat). També tenim `OnLoadGroupsListener`, és crida pel mètode `loadGroups`, que realitza el mateix que `loadUserGroups` però sense filtrar per grups. I un quart que es crida pel mètode `getGroup`.

Referent a la classe `UserRepository` altre cop comparteix moltes similituds amb `BillRepository` i `GroupRepository`, apliquem el patró singleton, per les mateixes raons que a `BillRepository`. Entre els atributs tenim un objecte `Firestore` que serveix per connectar amb la base de dades i una llista per cada definició de listener i tenir-hi totes les implementacions.

Les funcions principals d'aquesta classe són carregar en un array passat per paràmetre per un cert ID les dades de l'usuari amb l'ID passat per paràmetre i, la segona, afegir a la base de dades un nou usuari amb la informació passada per paràmetre.

Per obtenir un usuari concret necessitem una interfície que faci de listener, per les raons ja mencionades en el nostre cas és `OnLoadUsersListener`, i es crida el

mètode 'handler' un cop estem en l'onComplete del listener de la crida al mètode `getUser(userData, userID)` que retorna de la base de dades la informació de l'usuari amb l'id passat per paràmetre, així alertant als mètodes que esperen que s'hagin assolit les dades de l'usuari.

## 4.2. Viewmodel + view overviews

La part de view s'encarrega de la informació que prové el model a través de la base de dades. Aquesta comunicació Model-Vista no es realitza de forma directa, sinó que hi actua el ViewModel com a intermediari. Sent més concrets, el ViewModel s'encarrega de modificar el model a partir de les accions de l'usuari trameses a través de la Vista i informar la vista dels canvis del model, en el nostre cas els canvis en el model van lligats als canvis a la base de dades.

La nostra View està formada per les següents classes:

**MainActivity**, gestiona la interacció de l'usuari amb la vista primera que se li mostra en obrir l'aplicació.

**SignUpActivity** gestiona la interacció (a través de l>UserViewModel) de l'usuari amb el layout d'introduir les dades per crear un nou compte, un cop es prem el botó de registrar-se, un listener definit prèviament permet cridar a un mètode que s'encarrega de cridar al mètode que gestionarà les comprovacions prèvies a crear el nou usuari. Aquestes són; cridar el mètode que realitzar una comprovació del número de telèfon de manera que si es pot verificar automàticament el número de telèfon (cas OnVerification del listener definit a ser cridat per la base de dades), es procedeix a afegir l'usuari amb el seu ID (TLF+correu), en cas que no es pugui verificar automàticament (cas OnCodeSent del listener definit a ser cridat per la base de dades) llavors a través d'un intent obrim la vista de ConfirmMobileActivity, aquest intent ens ha de retornar el codi introduït per l'usuari i amb ell podem verificar el número de telèfon. Un cop verificat el número de telèfon cridem a la funció que registra l'usuari amb correu `TLF+"@moneysplitter.com"`, (hem explicat prèviament les raons d'aquest identificador,

fer servir el signUp amb correu de Firebase és només el mitjà per fer-ho, a l'apartat de Base de Dades s'explica amb més detall), que crida al mètode de Firebase per crear l'usuari amb `createUserWithEmailAndPassword`, del servei d'autenticació de Firebase, i en cas d'èxit (el listener sobre la tasca ens porta a l'OnSuccesful), cridem al mètode d'UserRepository que s'encarrega de crear un apartat pel nou usuari a la col·lecció d'users de la base de dades.

**ConfirmMobileActivity**, gestiona la interacció de l'usuari amb el layout d'introduir el codi de verificació del número de telèfon, al ser l'intent de SignUpActivity que obre la vista de confirmar el número de telèfon un intent que espera resposta ho fem servir per retornar el codi introduït.

**LoginActivity**, gestiona la interacció de l'usuari amb la vista que permet iniciar sessió a l'aplicació, de manera que un cop s'ha pulsat el botó del login un listener definit prèviament permet cridar a un mètode de l'UserViewModel que s'encarrega de cridar el mètode de la classe UserRepository, vist prèviament, per logejar l'usuari a través de la base de dades, de manera que si les dades són correctes, es crida l'OnSuccesful o l'OnFailed, això ja que hem implementat un listener que es crida un cop s'ha acabat l'activitat d'inici de sessió a la base de dades.

**HomeActivity**, gestiona la interacció de l'usuari amb la vista principal de l'aplicació com hem vist al punt 1. La primera tasca que realitza és recuperar l'objecte User de l'usuari que hi ha iniciat sessió des de la LoginActivity, per així carregar el nom de l'usuari a la part superior de la vista i, en general, poder personalitzar la vista. Recordem que aquesta transmissió de l'User no seria possible si no implementés la interfície Serializable.

Les principals interaccions són accedir al perfil, que simplement és un listener implementat que cada cop que es prem l'ImageView de l'usuari es crea un Intent que obre la Vista que mostra el perfil d'usuari (ProfileActivity). També gestionar les cerques que es realitzen sobre els grups, mostrant al recycler només els grups que coincideixen amb el criteri de cerca. Aquest garbell es realitza al ViewModel, que reajusta el MutableLiveData. També s'encarrega de crear el listener que escolta si s'ha pulsat el

text de novetats per comprimir els 4 botons de notificacions, canviant la seva visibilitat amb una transició de `TransitionManager`, així com d'obrir amb un `Intent` l'activity de `GroupActivity`. I la que segurament és la seva tasca central, gestionar part de les inicialitzacions i crides perquè es pugui mostrar el `RecyclerView` desplaçable.

A continuació mencionarem per sobre què realitza. Primer s'obté una nova instància del `ViewModel GroupViewModel` a través de `new ViewModelProvider`, cridant, des de l'onCreate a una funció de la mateixa classe. Busquem el `RecyclerView` a través de `findViewById` i li assignem un layout manager, en el nostre cas un `LinearLayoutManager`. Inicialitzem el nostre `RecyclerView Adapter`, en el nostre cas `GroupCardAdapter`, i li assignem el nostre `RecyclerView` i passant-li la referència a la llista de grups que té el `HomeViewModel` pel seu `LiveMutableData` (no directament), posteriorment explicarem més en detall el funcionament del `RecyclerView Adapter`. Creem un listener que escoltarà quan interactuem amb un ítem del `RecyclerView` (en el nostre cas sempre que es polsi l'ítem en general, s'obrirà l'activitat `GroupActivity`). Creem un observer i ho associem a tots els futurs i actuals elements de la llista de grups de `HomeViewModel`, així els canvis de la llista de grups del `HomeViewModel` seran notificats al `RecyclerViewAdapter`. Cridem al mètode del `HomeViewModel` que gestiona les crides per carregar les dades a la llista de grups des de la base de dades.

**GroupCardAdapter**, a grans trets, fa de nexa entre les dades que ha de mostrar el `RecyclerView` de la llista de grups i la vista final d'aquestes. Té com a atributs una referència a la llista de grups que s'ha de mostrar. Té una classe `ViewHolder` dins, cosa que es podria fer fora realment, aquesta fa de placeholder de la vista dels ítems dels `RecyclerView` (`group_card_layout.xml`), és a dir, per la vista de cada ítem hi carrega les dades del grup corresponent.

Més enllà dels listeners per respondre a comportaments sobre els ítems del `RecyclerView` els mètodes principals de la classe són els que hereta de `RecyclerViewAdapter`<>, aquest són `onCreateViewHolder`, que crea un `View` genèrica amb layout que se li passa, en el nostre cas li passem `group_card_layout.xml` i li assignem una nova instància del `ViewHolder` que farà de pont entre l'objecte `Group` i el `View` de l'ítem.



L'altre mètode essencial és `onBindViewHolder()`, que crida el mètode del `ViewHolder` que s'encarrega de llegir els atributs del grup i assignar-los degudament a la vista de l'ítem del `recycler`. Aquest mètode a partir de la posició del grup a carregar obté l'objecte amb la referència a la llista de grups que haurà estat carregada prèviament.

**GroupActivity**, gestiona la interacció de l'usuari amb la vista 3.7, la primera tasca que realitza, com `HomeActivity`, és recuperar l'objecte `Group` del group que ha estat polsat, que s'ha enviat en l'intent que l'ha iniciada a l'activitat `HomeActivity`.

Les seves principals interaccions són el botó de saldar els deutes, editar informació del grup, totes funcionalitats no implementades encara, tornar al home, que consisteix en un listener al botó del Home, que un cop polsat acaba l'activitat, tornant al `HomeActivity`. Dos botons que permeten una transició entre el fragment que conté el `RecyclerView` de les despeses i el que conté el balanç de cada usuari. Aquesta transició es du a terme quan el listener escolta que s'ha polsat un dels botons, amb el `FragmentManager`, començem una transacció i reemplaçem al `FragmentManager` el fragment actual pel nou (en funció del botó polsat, si és Despeses ficarem el fragment de Balanç i viceversa), finalment fem un `commit` perquè els canvis es mostrin. Mencionar que en canviar de fragment es canvia el botó que és blau dels dos, que indica al fragment on estem, cosa que també es fa a la vista.

La gestió del `RecyclerView` es realitza al `BillsFragment`, ja que és al fragment on realment `RecyclerView` es mostra, encara que tot es vegi en la mateixa activitat.

**BillsFragment**, principalment, la tasca d'aquesta Vista és realitzar les inicialitzacions per poder mostrar el `RecyclerView` de les despeses. En línies generals, el procés és anàleg al fet amb el `RecyclerView` de grups, només substituint grups per despeses, per tant, no hi entrarem. Això sí, hi ha un parell de particularitats relacionades amb què les inicialitzacions es duguin a terme a un Fragment, treballem amb el `ViewModel` l'activitat `GroupActivity` que l'obtenim un cop tenim la instància de l'activitat que conté el Fragment. A la hora de donar el Context de `LayoutManager`, cridem `getContext()`, al no poder passar un Fragment com a context, també es podria fer passant l'activity en la que estem (en essència totes dues solucions passen el mateix, ja que el context actual

seria el de l'activitat que conté el fragment), i per assignar l'observer als elements de la llista de despeses el LifecycleOwner l'assignem cridant getViewLifecycleOwner(), altre cop també es podria haver fet amb l'activitat.

**BillCardAdapter**, no cal entrar en detalls, el funcionament és anàleg al GroupCardAdapter.

**NewGroupActivity**, s'encarrega de la vista 3.6. i principalment envia les dades del nou grup a crear al ViewModel GroupViewModel. Per cada membre afegit amb el seu número de telèfon des de l'EditText, s'afegeix un nou ítem a la vista en el LinearLayout amb id newMembersLayout, instanciant un layout concret per aquesta tasca (new\_member\_preview\_layout) a la seva corresponent vista. Quan es polsa el botó de crear un nou grup, es crida la funció del GroupViewModel addGroup amb les dades dels EditTexts. Encara falta implementar l'opció de poder afegir una imatge des de galeria o prendre-la des de la càmera.

**ProfileActivity**, gestiona la interacció de l'usuari amb la vista 3.9., veient així la informació de l'usuari, com el seu número de telèfon i el seu correu electrònic. Encara no s'ha implementat mostrar la foto de perfil, el nom i el ID. Per poder accedir a les dades de l'usuari, es recupera l'usuari passat a l'hora de fer l'intent. S'implementa un listener per tal que si es prem l'ImageButton amb forma de casa, es faci un intent per tornar al HomeActivity, tancant aquesta activitat, passant l'usuari. D'igual forma, es crea un listener del botó d'editar perfil, per obrir l'EditProfileActivity, però sense finalitzar la pròpia activitat. Els listeners dels botons de mostrar l'històric de despeses i pagaments estan implementats, però de moment no fan res. Per últim, el listener del botó de recolzar el nostre projecte, porta a la pagina web que és escrita al botó, sent aquesta la pàgina principal de la UB.

**EditProfileActivity**, gestiona la interacció de l'usuari amb la vista 3.10. Primer de tot, es recupera el User. En aquest cas, l'ImageButton amb forma de fletxa, té un listener que porta a ProfileActivity, fent un finish d'aquesta activitat i passant el User en l'intent. S'ha implementat un listener al botó de desactivar/activar notifikacions perquè el text variï entre les dues possibilitats, però encara no hi és implementada cap funcionalitat

respectiva a les notificacions. El listener del botó de tancar sessió, simplement finalitza l'activitat i fa un intent de mostrar la MainActivity. Per últim, el listener de canviar contrasenya, inicia un intent de mostrar l'activitat ChangePassword, passant-li també el User.

**ChangePassword**, gestiona la interacció de l'usuari amb la vista 3.11. Primer de tot, es recupera el User. Després, l'únic que es fa és implementar el listener del botó de canviar contrasenya. Aquest listener, el que fa és comprovar que la contrasenya introduïda sigui la mateixa que conté com a atribut el User passat, que, per tant, coincideix a la de la base de dades i l'eina d'autenticació; després comprova que la nova contrasenya sigui la mateixa els dos cops que ha de ser escrita, recuperant el text introduït als EditTexts, i, si tot és correcte, crida al mètode SetPassword amb la nova contrasenya, tanca l'activity i torna a l'activitat EditProfileActivity.

Pel que fa als ViewModels, son les classes que s'encarreguen de realitzar la comunicació Vista-Model, en el nostre cas, de moment, en tenim quatre: GroupViewModel, BillViewModel, BillMembersViewModel i UserViewModel.

**GroupViewModel**, compta amb els següents atributs, dos MutableLiveData, un per la llista actual de grups que es mostra en el RecyclerView de la vista 3.5, i un altre MutableLiveData per guardar la llista del total de grups que es podrien mostrar per l'usuari actualment. Això ho hem realitzat així per poder implementar el buscador. De manera que anem filtrant els elements del segon MutableLiveData en el primer, en funció del filtre de cerca. Fem servir MutableLiveData per guardar la col·lecció dels elements que es mostren pel RecyclerView, ja que és la millor eina per actualitzar les dades en temps real del RecyclerView, en permetre'ns que pugui ser observada pel HomeActivity, que en el nostre cas ho notifica al RecyclerView.

El ViewModel GroupViewModel també conté la instància del GroupRepository, que s'utilitza per cridar als mètodes de carregar dades sol·licitades per la Vista des de la base de dades utilitzant el model, en aquest cas GroupRepository.

Les funcionalitats d'aquest ViewModel són totes les relacionades amb els Objectes Group que requereixen informació de la base de dades per la Vista.

Tenim filterGroupsBy, que fa un simple garbell de tots els Groups mantenint només els que el seu nom conté la cadena del filtre. getGroup, per obtenir un grup concret de la base de dades, crear una instància del Listener de GroupRepository; GroupRepository.OnLoadGroupListener, que es crida un cop s'ha acabat a GroupRepository la cerca del grup, com hem explicat prèviament, i en el mètode per gestionar la crida a aquest listener cridem a tots els Listeners onGetGroupListener (interfície definida en aquesta classe perquè buscar un Group en la BBDD es dona en segon pla) que tenim definits i els passem el grup trobat. Així el HomeActivity pot aconseguir un grup de la base de dades. La funció loadGroupsFromRepositoryFiltered, defineix un Listener que en cridar-se, crida el mètode per buscar grups filtrant per un usuari, aquest Listener només es crida un cop s'ha escrit a GroupRepository l'array de grups d'un usuari concret, així ens assegurem que no filtrem amb l'array per un usuari quan l'array dels IDs de grups d'un usuari estigui buit. Prèviament al constructor, hem definit un listener perquè es crida quan es modifica es busquen els grups de la Base de dades amb el GroupRepository, a la funció gestora actualitzem el valor del MutableLiveData, això no es pot fer amb returns al ser tasques en segon pla.

Pels següents ViewModels passarem per sobre al ser les funcions similars, canviant Groups per Bills o altres ítems.

**UserViewModel**, conté una interfície OnGetUserListener i un mètode getUser que es comporta de forma anàloga a OnGetGroupListener i getGroup. Té la instància de UserRepository, pels mètodes d'addUser i getUser, que requereixen accés a la base de dades. Pel que fa a addUser simplement crida al mètode addUser d'UserRepository amb les dades de l'usuari passades per paràmetre, mètode tractat prèviament.

**BillViewModel**, per gestionar la interacció de la Vista amb els Bills que requereixen de càlculs o accessos a la base de dades (en general accessos al model), aquí a diferència de GroupViewModel, no definim un listener per cada cop que cridem loadBillsFromRepository, ja que llavors les diverses transicions entre vistes provoquen

que es creïn diversos listeners cosa que fa que hi hagi diversos listeners i en cridar al mètode de `GroupBillRepository` es cridi per totes les instàncies prèvies i, per tant, el `MutableLiveData` s'ompli d'elements duplicats. Per això només creem el `Listener OnLoadGroupBillsIDListener`, un cop al constructor, inicialitzant-lo com a atribut.

**BillMembersViewModel**, `ViewModel` actualment, exclusiu pel `MutableLiveData` que guarda els usuaris participants d'un Bill, que es mostra en el `RecyclerView` dels Detalls d'una despesa. Aquí `loadBillMembersFromRepository`, no ha de filtrar per cap ID, ja que només s'ha de buscar el camp de membre d'una despesa mitjançant `GroupBillsRepository`, per tant, no tenim cap `Listener` extra com l'`OnLoadGroupBillsIDListener`.

### 4.3. Base de dades

Les bases de dades són una part essencial d'una aplicació, per guardar grans volums de dades que no té sentit tenir carregats tota l'estona i, principalment, per tenir una persistència de dades.

Nosaltres, seguint amb la recomanació donada, hem fet servir una base de dades de Firebase, a causa de la seva senzillesa i adaptabilitat.

Pel que respecta a la creació i inici de sessió d'un nou usuari, fem servir el sistema Auth de Firebase, recordem que en la primera entrega el plantejament era poder iniciar sessió amb el telèfon, el correu o l'ID, com finalment tenint el telèfon i el correu hem vist redundant l'ús d'un ID, hem descartat aquesta via. Pel que fa al tema del telèfon Firebase permet verificar que un número és correcte, però no existeix una eina perquè l'usuari amb un número telefònic associat pugui iniciar sessió amb una contrasenya. A sobre, encara que féssim que per iniciar sessió amb telèfon no s'ignori la contrasenya i només es demani a l'usuari el codi de verificació, Firebase no permet associar a un mateix usuari dues vies d'inici de sessió, en aquest cas telèfon i correu.

Com no volíem que cada cop que l'usuari iniciï sessió hagi d'introduir el codi rebut, cosa que empitjoraria l'experiència d'usuari. I tampoc volíem renunciar a associar el compte

d'un usuari al número de telèfon, ja que seria molt més segur que el correu. Hem arribat a la següent solució, només es verificarà el telèfon en registrar-se i internament a l'usuari se li crearà un compte amb el correu <telèfon>@moneysplitter.com, en un futur aquest domini intern existirà i cada usuari tindrà el seu correu. Així per iniciar sessió l'usuari ha d'introduir el seu número de telèfon i la seva contrasenya i en el codi en verificar amb Firebase Auth se li afegirà al número introduït @moneysplitter.com.

La nostra base de dades, està formada per una col·lecció d'usuaris on cada usuari ve identificat per únic ID la unicitat d'aquest ID ve del fet que l'ID és el número de telèfon de l'usuari seguit del seu correu, com verifiquem els números de telèfon tenim assegurada la unicitat. Per cada usuari guardem el nom, cognoms, el telèfon, la contrasenya, el correu i una URL de la foto de perfil.

Tenim una col·lecció amb tots els grups existents, l'ID de cada grup consisteix en el nom donat a aquest per part de l'usuari, generant un enter aleatori + el nom, prèviament farem una comprovació de què aquesta combinació coincideixi amb la d'un grup ja creat, per tenir una unicitat global de l'ID de cada grup, això però, no s'ha implementat perquè no s'ha acabat la implementació de crear nous grups. Sobre cada grup guardem la següent informació el nom, la data, la descripció així com una URL a la foto d'aquest.

Pel que fa a les despeses tenim una col·lecció anomenada Bills on l'id es genera de forma similar, i cada despesa conté el nom i el cost total, pressuposem de moment que tot està donat en euros. El cost està en String, ja que el format numèric que permet Firebase és Long i podia problemes a l'hora de fer el cast al codi.

Ara falta tractar les relacions entre grups-usuaris, grup-despeses, és a dir, guardar a la base de dades quins són els membres d'un grup, quines són les despeses d'aquest. Per fer això, l'ideal seria un sistema de taules de relacions binàries entre usuaris-grups i despeses-grups, com es sol fer a SQL, però com que no és possible al nostre cas hem pres al següent estratègia, per les relacions usuari-grup creem una col·lecció anomenada UserGroups, de manera que cada element és l'ID d'un usuari i té un únic camp que és un array amb els grups als quals hi pertany de manera que per obtenir els

grups d'un usuari és tan senzill com obtenir el document d'ID de l'usuari del qual es volen saber els grups, l'únic punt negatiu d'aquesta manera és que per saber els usuaris d'un grup hem de recórrer tots els documents de la col·lecció per saber en quins el grup amb el qual filtre es troba, però l'altra solució seria crear una col·lecció que els documents siguin els IDs dels grups, però això crearia una duplicació d'informació innecessària, per això hem optat per la primera opció.

Pel que fa a les relacions grup-despeses s'ha procedit de forma anàloga.

## 5. FEINA PENDENT A FER

Com hem mencionat al primer punt, hem estimat que portem un 70% de l'aplicació desenvolupat i que en aquest apartat explicarem tot el que ens falta.

- Primer de tot hem de mencionar que mirant el codi es pot veure clarament que no és el millor disseny de software possible, és a dir, tenim un alt acoblament (les classes han de conèixer molta informació d'altres per utilitzar-les) i una no molt alta cohesió de les classes (algunes classes tenen varies responsabilitats, de fet, vulnerant el principi S dels SOLID). És cert que en línies generals apliquem el patró MVVM, però, no és suficient per això, per l'entrega final falta solucionar aquesta problemàtica aplicant a partir dels principis S.O.L.I.D. i els patrons GRASP, els patrons de disseny adients en cada moment.
- A moltes Vistes falta comprovar que certs camps obligatoris no es deixin buits, i alertar l'usuari en cas que vulgui realitzar l'acció corresponent amb el camp buit. Això és dona en la vista de registrar-se o crear un nou grup. La idea, en cas que intenti dur a terme l'acció corresponent deixant el camp buit, seria posar el TextView corresponent en vermell per indicar que no es pot deixar el camp buit.
- A moltes vistes es mostra el telèfon de l'usuari en comptes del seu nom i cognoms, per exemple, per mostrar qui ha pagat una despesa es posa el número de telèfon, això haurem de posar el nom i cognoms de l'usuari i d'alguna forma que es pugui veure el telèfon en cas que diversos usuaris comparteixi nom i cognoms.
- Falta assegurar la coordinació entre la creació d'un grup/despesa per un usuari i que se li mostri quasi al moment a la resta d'usuaris.
- Resta habilitar els botons que mostren diferents tipus de notificacions de l'aplicació a la pantalla de HomeActivity.



- També haurem de coordinar les transicions entre les diferents activitats eliminant les instàncies d'algunes quan sigui necessari, per exemple, no s'ha de poder tornar al menú del login un cop estem en HomeActivity, un cop se'n vagi a altres, per això farem servir un component de navegació: NavGraph.
- Falta implementar la funcionalitat del botó de "Saldar Deudas".
- Falta implementar el Fragment per veure el balanç dels membres del grup, així com poder modificar la informació del grup.
- Falta enviar les notificacions necessàries des de l'APP, per exemple, un avís que s'ha estat afegit en un nou grup, s'ha sol·licitat saldar els deutes, etc. Han estat tractades en detalls durant la primera entrega.
- De cara a millorar la vista del SignUp es preveu fer que sigui un seguit de Fragment que van mostrant-se un rere l'altre.
- Falta implementar l'opció de posar a l'APP imatges de la galeria de l'usuari o preses amb la càmera al moment. Tant pel perfil, grups i despeses.
- Falta permetre que es creïn noves despeses.
- Falta implementar la majoria de les funcionalitats del perfil.

Encara que semblin moltes tasques la majoria són curtes de realitzar, o similar a altres ja realitzades i, per tant, part del codi podrà ser reutilitzat. Les principals són habilitar l'ús d'imatges pròpies, revisar acoblament i cohesió del codi, saldar deutes i fer saltar les alertes corresponents.

## 6. DIAGRAMA DE CLASSES

S'ha adjuntat amb l'entrega un PNG del diagrama de classes. No es posa en aquest informe ja que seria il·legible aquí.

En ell es pot veure com es relacionen totes les classes, dividides en el model, la view i el viewmodel.