

Creació d'una aplicació des de zero

Projecte Integrat de Software

Entrega final



Víctor González Porras

Raul Asins Hidalgo

Víctor Sort Rubio

David Fernández Gómez

Soufiane Lyazidi Ahrillou Abraray

ÍNDEX

1. INTRODUCCIÓ.....	3
2. INTERFÍCIES D'USUARI REALITZADES.....	4
2.1. Vista d'accés.....	4
2.2. Vista d'inici de sessió.....	5
2.3. Vista de registre.....	6
2.4. Vista de validació de telèfon.....	7
2.5. Vista de cerca de grups.....	8
2.6. Vista de creació de grups.....	10
2.7. Vista de visualització de grup.....	11
2.8. Vista de creació de despesa.....	13
2.9. Vista de detalls de la despesa.....	14
2.11. Vista d'editar perfil.....	16
2.12. Vista canviar contrasenya.....	17
2.13. Vista històric de pagaments.....	18
2.14. Vista històric de despeses.....	19
2.15. Justificació dels layouts escollits.....	20
2.15.1. Constraint Layout.....	20
2.15.2. Linear Layout.....	20
2.15.3. Card View.....	20
2.15.4. Recycle View.....	21
2.15.5. Frame Layout.....	21
2.15.6. Relative Layout.....	22
3. DISSENY.....	23
3.1. Model overview.....	23
3.2. Viewmodel + view overviews.....	27
3.3. Base de dades.....	35
3.4. Diagrama de classes.....	37
4. TESTING PLAN.....	38
4.1. Unit testing.....	38
4.2. Integration testing.....	39
4.3. Performance Testing.....	39
4.4. Usability testing.....	40
5. PROBLEMES TROBATS.....	43
6. ANÀLISI DELS REQUERIMENTS INICIALS.....	47
7. FEINA PEL FUTUR.....	51
8. CONCLUSIONS FINALS I AGRAÏMENTS.....	52

1. INTRODUCCIÓ

L'aplicació Money Splitter és una solució desenvolupada per a la gestió de les despeses a nivell individual o col·lectiu, pensada especialment per gestionar els pagaments entre amics, companys de pis i de feina. Aquest informe representa la tercera entrega i l'última del projecte sent una progressió i una millora de la segona entrega.

En aquest informe, es detallen les interfícies d'usuari desenvolupades per a l'aplicació, proporcionant una justificació per a les eleccions dels diferents esquemes de disseny utilitzats. També s'examina l'arquitectura de disseny de l'aplicació, amb el corresponent diagrama de classes. S'inclou també l'estructura de la base de dades utilitzada per gestionar les dades dels usuaris i les despeses.

Per assegurar la qualitat i funcionalitat de l'aplicació, s'ha elaborat un pla de proves que inclou proves d'unitat, proves d'integració i proves d'usabilitat. Aquests aspectes seran explorats en aquest informe per tal de garantir una experiència d'usuari òptima i una aplicació confiable.

Finalment, es conclourà l'informe amb les conclusions finals i els agraïments a tots els membres del grup i a les persones que han contribuït a la realització d'aquest projecte.

Anotació important: el professor Alex Dickson amb els correus alexdickson53@gmail.com i alexdickson@ub.edu té accés a la consola del Firebase del projecte com a propietari, on pot revisar la base de dades (per veure els usuaris i les seves contrasenyes i poder iniciar sessió amb ells), el Storage (per veure les imatges) i l'Authentication.

2. INTERFÍCIES D'USUARI REALITZADES

Mentre anem explicant cada vista de forma no gaire extensa, mencionarem els layouts utilitzats i les raons per la seva elecció. Com a exemple per a les imatges ens basarem en un noi, el Víctor, que comparteix pis amb dues persones i a més té un grup personal per apuntar-se les seves despeses. Les dimensions dels elements han estat escollides per les dimensions del Pixel 6.

2.1. Vista d'accés

Aquesta és la vista principal que s'obre en iniciar l'aplicació. Conté el logo de l'aplicació i tres botons: un per descarregar els termes i condicions, un per registrar-se i un per accedir mitjançant l'inici de sessió. El botó de termes i condicions redirecciona a la pàgina web d'inici de sessió a la intranet del Món UB, sense motiu específic.

Hem utilitzat un Linear Layout a causa de la disposició vertical majoritària dels components, conjuntament amb un Constraint Layout per la col·locació adient dels botons “Registrarse” i “Acceder”. S'ha fet servir un ScrollView general per poder garantir que en canviar el dispositiu, es pugui accedir als botons inferiors.

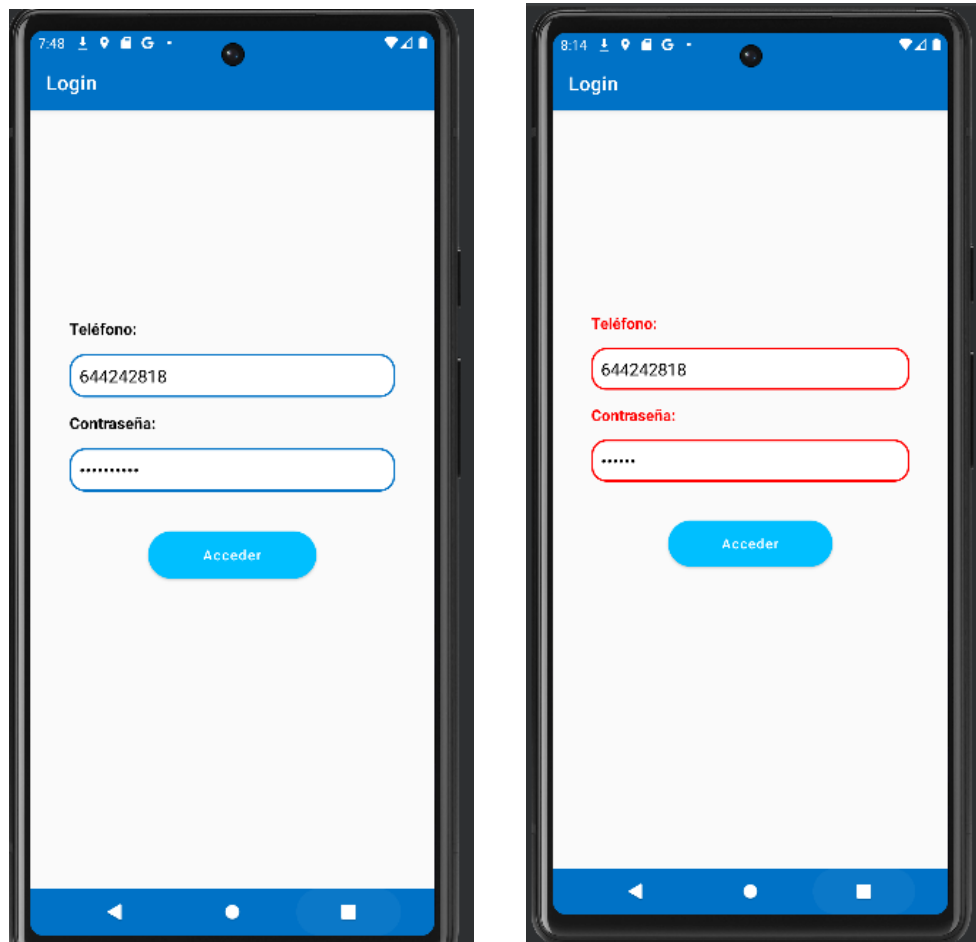


Figura 1: Vista d'accés

2.2. Vista d'inici de sessió

Aquesta vista permet als usuaris iniciar sessió utilitzant el seu número de telèfon i contrasenya. Conté dos EditText i un botó per confirmar l'accés. Hem afegit un contorn als EditText i omplert el seu fons de blanc pur, a diferència de com es pot veure als mockups, ja que creiem que això és més intuïtiu i estètic, proporcionant una millor experiència d'usuari. En el cas que l'usuari o la contrasenya siguin incorrectes, es mostrarà per pantalla i els texts es tornaran vermells.

S'ha seleccionat un Constraint Layout, per la simplicitat de la vista.

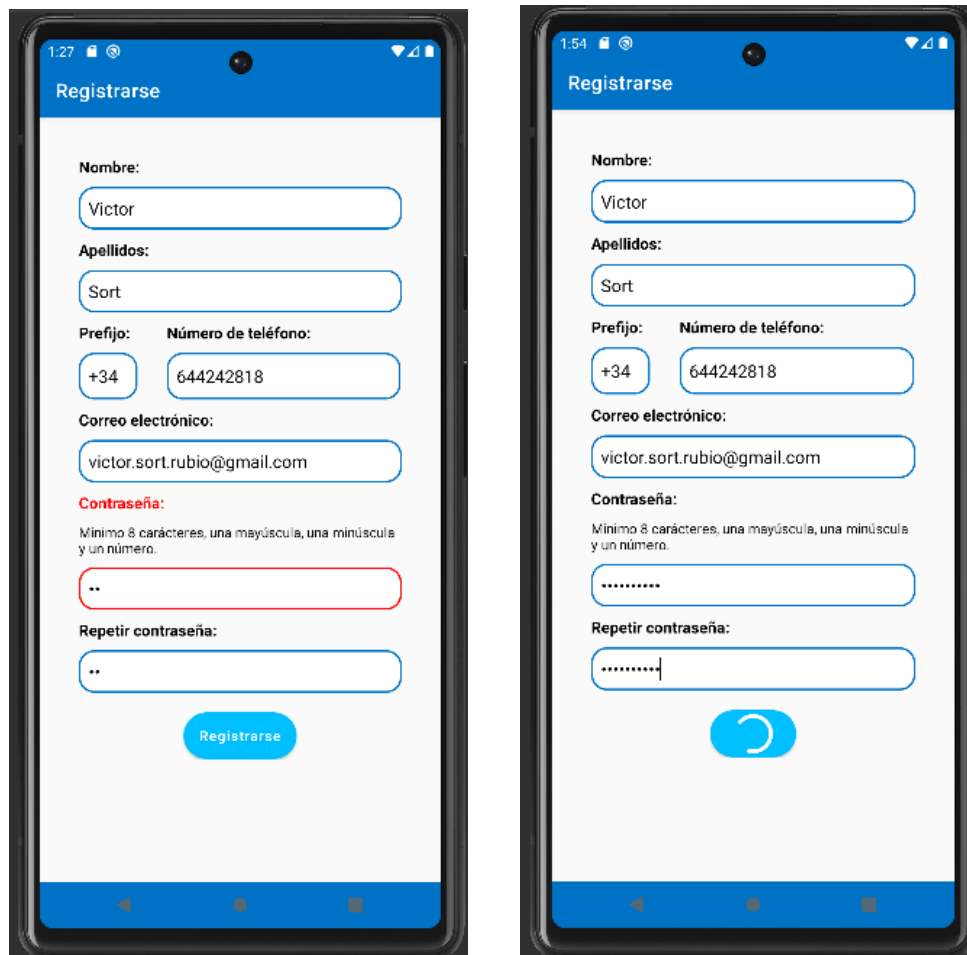


Figures 2 i 3: Vistes d'inici de sessió

2.3. Vista de registre

Vista per al registre d'un nou usuari. S'inclouen diversos EditText per a omplir la fitxa de l'usuari amb la següent informació: nom, cognom, prefix, telèfon, correu electrònic, contrasenya i confirmació de contrasenya. Tal com s'ha mencionat a la secció 2.2., s'ha decidit canviar la vista dels EditText per una millor UX. Si la contrasenya no coincideix o no compleix els requisits, no deixa fer el registre i es torna el text vermell.

Hem optat en aquest cas per un Linear Layout ja que els continguts s'alineen en una única direcció vertical, amb l'ús d'un Constraint Layout intern per la disposició horitzontal del prefix i el número de telèfon. Conté un ScrollView per la comoditat i adaptabilitat a nous dispositius.



The image displays two side-by-side screenshots of a mobile application's registration screen, titled "Registrarse". Both screens show the same form fields: "Nombre:" (Victor), "Apellidos:" (Sort), "Prefijo:" (+34), "Número de teléfono:" (644242818), "Correo electrónico:" (victor.sort.rubio@gmail.com), "Contraseña:" (with a red border and asterisks), and "Repetir contraseña:" (with asterisks). The left screenshot (Figure 4) shows a blue "Registrarse" button. The right screenshot (Figure 5) shows a blue button with a white circular loading spinner. The status bars at the top show the time as 1:27 on the left and 1:54 on the right.

Figures 4 i 5: Vista de registre

2.4. Vista de validació de telèfon

Vista per a confirmar el telèfon. Després que l'usuari hagi proporcionat les seves dades personals, l'usuari rep un missatge SMS al seu telèfon i en un EditText ha d'escriure el codi rebut i confirmar amb el botó.

S'ha seleccionat un Constraint Layout, per la simplicitat de la vista.



Figura 6: Vista de validació de telèfon

2.5. Vista de cerca de grupos

És la vista principal de l'aplicació. A dalt de tot es mostra un text de benvinguda i, a la dreta, el CardView amb la imatge de perfil porta al perfil de l'usuari. L'apartat de novetats es pot replegar per veure el RecyclerView de CardViews de Grups més gran. El botó flotant amb el símbol de "+" obre la vista per crear un nou grup. La lupa serveix

per cercar entre tots els grups. S'ha aconseguit que a mesura que es va escrivint el nom del grup desitjat, independentment de majúscules i minúscules, es vagin filtrant els grups sense necessitat d'haver d'acabar d'escriure el nom i sense haver de prémer el símbol de la lupa.

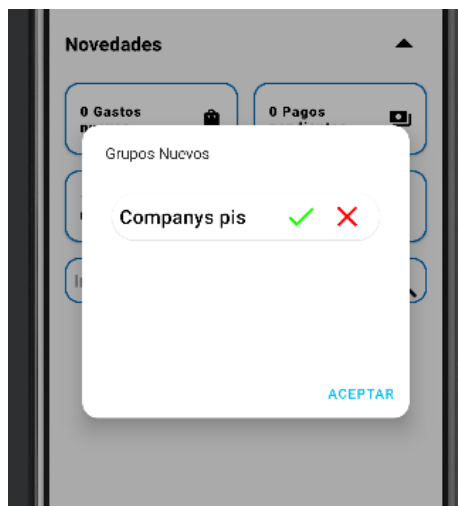
Les novetats són els avisos que s'envien quan es vol afegir un usuari a un nou grup, se li apunta una nova despesa o té un pagament pendent a fer o rebre. Quan es tingui un pagament pendent a fer, s'eliminarà quan la persona que l'ha de rebre seleccioni a l'aplicació, dins de la CardView de la notificació, que ja l'ha fet.

Respecte als mockups, els botons de les notificacions tenen un contorn, ja que hem cregut que seria més intuïtiu i estètic. 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 o importància similar, cosa que no passava amb les funcionalitats de 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 lupa 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. El motiu és que el Table Layout, que seria a priori el més adient, no permet que un element ocupi més d'una fila.



Figura 7: Vista de cerca de grup



Figures 8 i 9: Vista de cerca de grup - notificaciones

2.6. Vista de creació de grups

La vista per crear un nou grup inclou tres EditText per al nom del grup, la descripció i per afegir membres mitjançant el número de telèfon, amb un botó "+" per sobre per afegir-los. Quan es prem aquest botó, els membres s'afegeixen com a TextViews a la part inferior de la vista. També hi ha tres botons per afegir la imatge del grup, afegir membres des de la llista de contactes i crear el grup. El botó d'afegir una imatge pel grup obre un diàleg que fa escollir entre obrir la càmera o els arxius i en l'EditText es mostra l'adreça interna en el telèfon on hi és la imatge. Realment aquesta adreça no s'utilitzarà internament, però serveix com a UX perquè l'usuari sàpiga que s'ha seleccionat bé la imatge.

En aquest cas, s'ha tingut en compte el disseny original dels MockUps, amb l'excepció de les vores dels botons com s'ha mencionat anteriorment. S'ha utilitzat un Linear Layout vertical perquè la vista consisteix en una llista d'elements. Per a l'EditText per afegir membres mitjançant el número de telèfon, s'ha utilitzat un Relative Layout per poder posicionar el botó "+" per sobre de l'EditText.



Figura 10: Vista de creació de grups

2.7. 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 i el botó flotant per crear una nova despesa i el `Fragment` que conté el `RecyclerView` dels balanços del grup. Si polsem sobre una despesa, es desplega la `CardView` amb els detalls de la despesa.

Hem eliminat la doble pestanya entre visualitzar les despeses i crear-ne una de nova, gràcies a l'ús del botó flotant. En canvi, sí que alternarem la vista entre aquest fragment i el dels balanços. El càlcul dels balanços de cada usuari segueix un algorisme explicat a la primera entrega.

Per a l'estructura de la vista, s'ha utilitzat un `Linear Layout` vertical, ja que la vista consisteix en una llista d'elements. Els `Linear Layouts` aniuats per al nom, la imatge i altres elements del grup es van crear de manera anàloga a les `CardViews` dels grups del 2.6. Pel `Fragment` de Despeses s'ha utilitzat un `Frame Layout`, ja que és el que s'assigna per defecte als fragments, però a l'interior s'ha utilitzat un `RelativeLayout` per col·locar el botó d'afegir despesa per sobre del `RecyclerView` de despeses del grup. Pel `CardView` de les despeses s'ha utilitzat un `Linear Layout` vertical per permetre tenir una part oculta de la `CardView` i només mostrar el concepte i el preu. Per a la part detallada, s'ha utilitzat un `Linear Layout` amb `TextViews` i `Buttons`. S'ha optat per un `Linear Layout` vertical per raons similars a les mencionades anteriorment, ja que el `Table Layout` no permet que els elements ocupin més d'una fila.

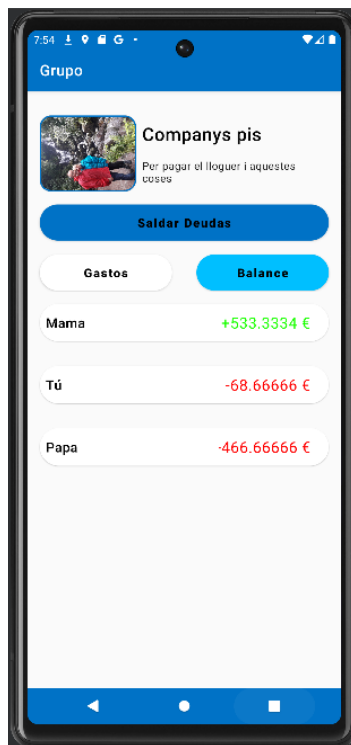


Figura 11: Vista de visualització de grup - Despeses

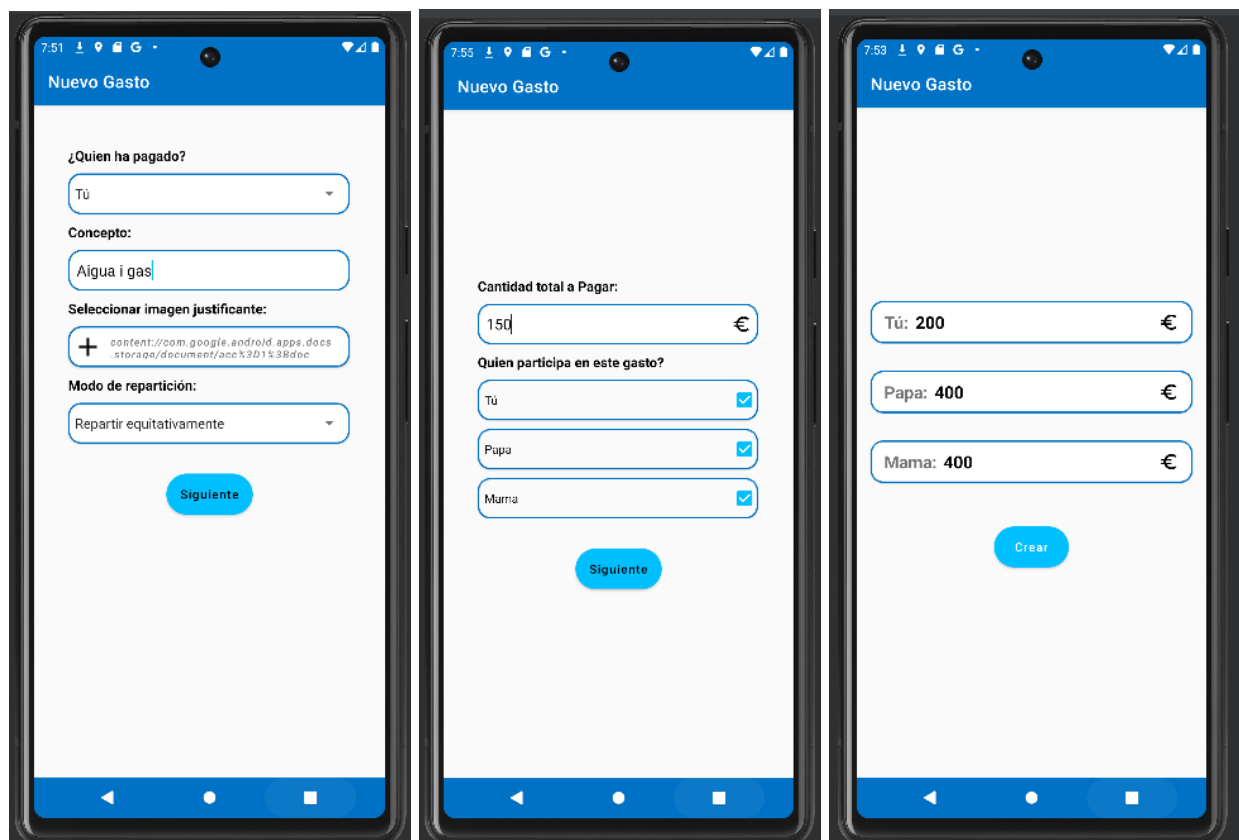


Figura 12: Vista de visualització de grup - Balanços

2.8. Vista de creació de despesa

La vista per crear una nova despesa inclou quatre EditText pel concepte de la despesa, qui l'ha pagat (surten les opcions amb un desplegable), afegir una imatge opcional i seleccionar el mode de repartició. Després de confirmar la despesa amb el botó, depenent del mode de repartició seleccionat es reconduïx a dues altres vistes on s'ha d'omplir informació, tal i com es veia en els MockUps.

En aquest cas, s'ha tingut en compte el disseny original dels MockUps, amb l'excepció de les vores dels botons com s'ha mencionat anteriorment. S'ha utilitzat un Linear Layout vertical perquè la vista consisteix en una llista d'elements.



Figures 13, 14 i 15: Vista de creació de despesa

2.9. Vista de detalls de la despesa

Vista per veure els detalls d'una despesa concreta. Mostra l'ImageView de la foto de la despesa així com el context, 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 podíem controlar el nombre d'usuaris per despesa i fer-ho amb un RecyclerView seria una pèssima idea des del punt de vista de l'UX, ja que és molt poc intuïtiu.

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.

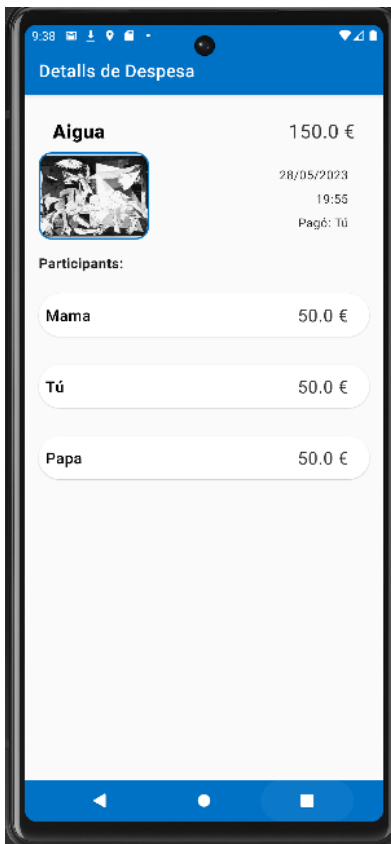


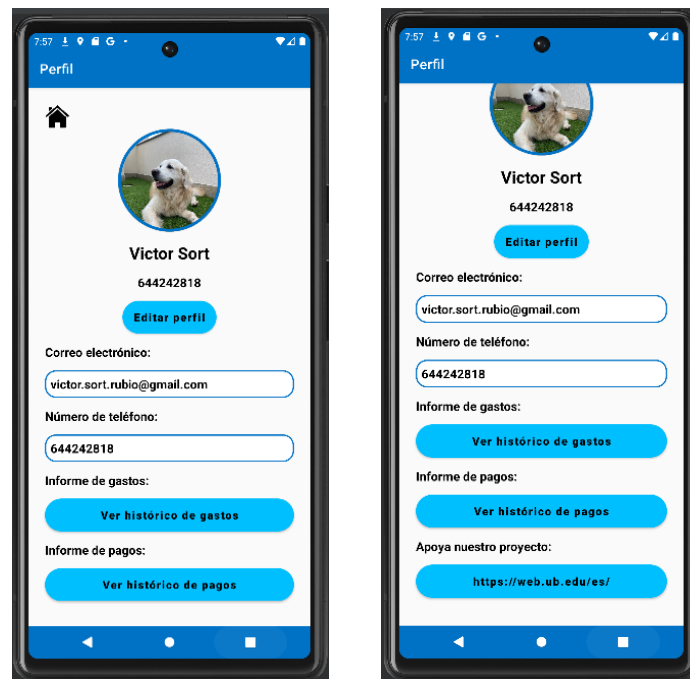
Figura 16: Vista de detalls de la despesa

2.10. 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 i de pagaments i a la pàgina web per ajudar-nos econòmicament. Aquesta última segueix sent la pàgina de la UB.

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.



Figures 17 i 18 Vista del perfil

2.11. 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 19: Vista d'editar perfil

2.12. Vista canviar contrasenya

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 els TextViews explicatius. Per últim hi ha un botó per finalitzar l'activitat, que mostra en vermell els camps que continguin dades incorrectes en cas que n'hi hagi.

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



Figura 20: Vista de canviar contrasenya

2.13. Vista històric de pagaments

En aquesta vista, a més del Button per tornar al perfil i un TextView que defineix la vista, només conté una llista de tots els pagaments que ha fet o rebut l'usuari, ficats dins d'un ScrollView. La direcció de les fletxes i els colors dels pagaments indiquen clarament si són pagaments fets o rebuts, on sempre vermell és sinònim de reduir i verd de guanyar o rebre.

S'ha fet servir un LinearLayout vertical, on s'ha afegit tots els components esmentats. Dintre de cada pagament hi ha tres TextView que indiquen la persona a la que ha enviat o de la que ha rebut els diners, la data i la quantitat.

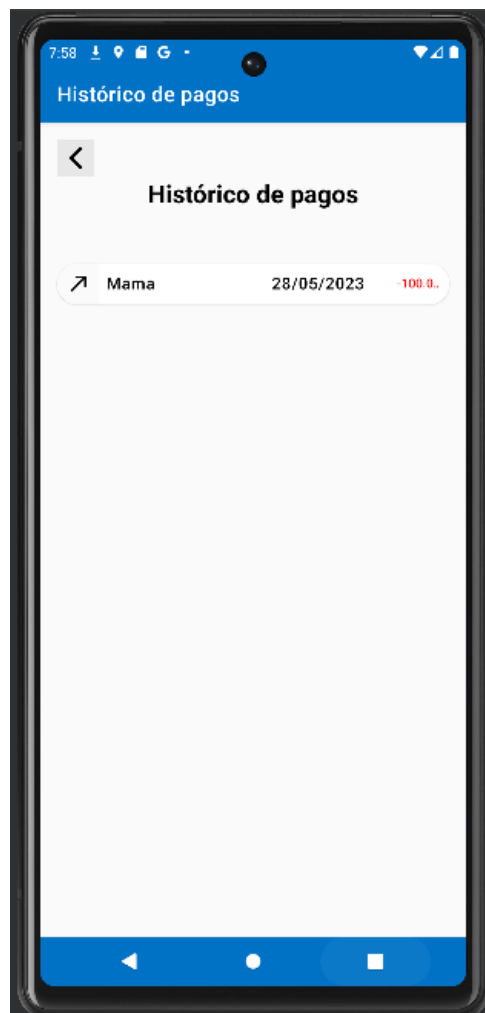
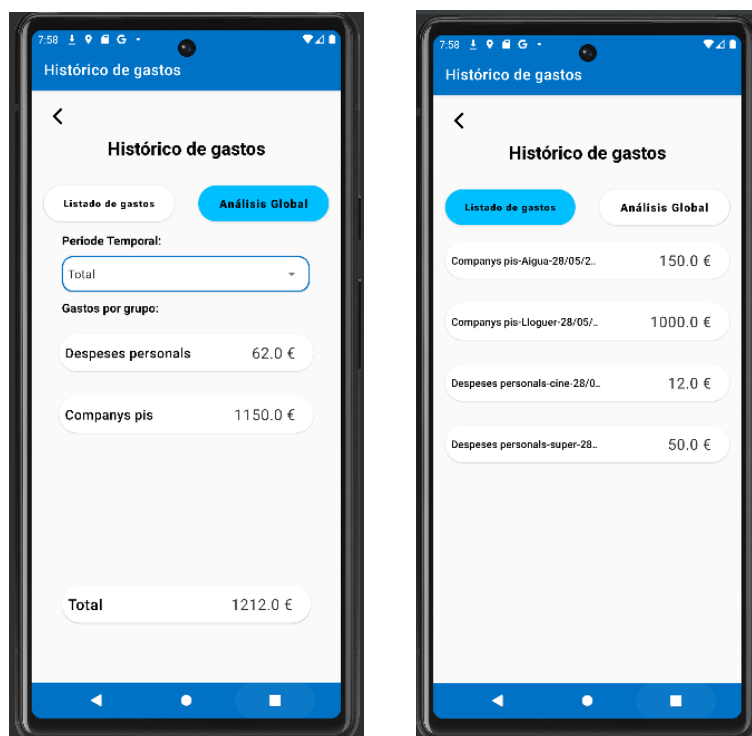


Figura 21: Vista d'històric de pagaments

2.14. Vista històric de despeses

Aquesta vista conté el Button per tornar al perfil i un TextView que defineix la vista, com a l'anterior vista. A més, té dos Buttons més per canviar entre els Fragments de veure totes les despeses, on només hi ha una llista de les despeses ficades dins d'un ScrollView, i el de veure les estadístiques o l'anàlisi global. El fragment de veure totes les despeses inclou totes les despeses en les quals ha participat l'usuari (encara que la despesa personal de l'usuari sigui menor). El fragment d'estadístiques mostra la suma de totes les despeses en les quals ha participat l'usuari dividides per grups i també mostra el total, tot en el període de temps indicat en el desplegable.

S'ha fet servir un LinearLayout vertical, on s'han afegit tots els components esmentats. Dintre de cada despesa hi ha TextViews amb la informació necessària per definir la despesa. A l'anàlisi global també s'han respectat bastant els mockups, podent escollir el període que vulguis en un Button i mostrant les estadístiques únicament d'aquest període de temps.



Figures 22 i 23: Vista d'històric de despeses

2.15. Justificació dels layouts escollits

2.15.1. Constraint Layout

El Constraint Layout és el layout més flexible d'Android, ja que permet crear dissenys grans i complexos sense necessitat d'utilitzar altres ViewGroups. És bastant semblant al Relative Layout, ja que les vistes es relacionen entre si dins del mateix ViewGroup i el disseny principal. A més a més, és fàcil de manipular sense haver de modificar el codi XML.

El Constraint Layout és present a un gran nombre de ViewGroups del nostre projecte, com ara `activity_confirm_mobile.xml`, `activity_log_in.xml`, `activity_main.xml`, `activity_profile.xml`, `activity_sign_up.xml`, entre d'altres.

2.15.2. 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 utilitzat una alineació vertical per a tots els ViewGroups.

Algunes de les activitats que incorporen el LinearLayout són `activity_home_groups.xml` i `activity_home_new_group.xml`.

2.15.3. Card View

Les Card Views són layouts que permeten mostrar informació en un format reduït i visualment més atractiu, com ara imatges o informació concisa. També poden ser interactives, ja que al prémer una card ens pot proporcionar més informació.

Hem utilitzat una Card View per crear el `group_card_layout`, que mostra una imatge del grup, el nom, la data i un petit resum del grup. També l'hem utilitzat en

bill_card_layout.xml i bill_member_card_layout.xml d'una manera similar i per mostrar la imatge de perfil de forma compacta.

2.15.4. Recycle View

El Recycler View és un component de la biblioteca de suport d'Android que permet mostrar gran quantitat d'informació de manera eficient. Conté múltiples Card Views i només carrega les que es mostren a la pantalla, optimitzant d'aquesta manera el rendiment.

En el nostre cas, hem utilitzat un Recycler View en activity_home_groups.xml per mostrar una llista dels grups als quals pertany. Només es mostren els grups que ocupen l'espai indicat, per això s'ha fet ús d'un ScrollView. També l'hem utilitzat en bills_fragment.xml i bills_details_activity.xml de manera similar.

2.15.5. Frame Layout

El Frame Layout és un layout bastant senzill que alinea tots els elements a la cantonada superior esquerra, fent que els components es sobreposin llevat que es facin invisibles. És útil per col·locar imatges o botons únics a totes les pantalles, sense molts altres objectes dins del mateix layout.

En el nostre cas, utilitzem aquest layout en l'aplicació fragment_bills.xml, ja que és el layout predeterminat per encapsular els fragments. Internament s'han utilitzat altres layouts segons la necessitat, com el RecyclerView en el nostre cas.

2.15.6. Relative Layout

El Relative Layout és un layout que permet posicionar els elements en relació amb altres elements, de manera similar al Constraint Layout.

En el nostre cas, hem utilitzat el Relative Layout pels botons flotants d'afegir grup i despesa, ja que els volíem situar per sobre dels Recyclers Views dels grups i despeses. També s'ha utilitzat per exemple per l'EditText d'afegir usuaris, ja que volíem el botó de “+” sobre l'EditText.

3. DISSENY

3.1. 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 proveeix a la View (a través del ViewModel) amb les dades requerides per aquest últim.

A la classe Bill, respecte l'entrega passada hem afegit un booleà per saber si la despesa ha estat saldada, així com un String per guardar la URL de descàrrega de la imatge que és al Firebase Storage.

A la classe Group hem afegit un String per guardar l'enllaç de descàrrega de la imatge del Firebase Storage i un element de la classe GroupMembers, que conté els membres d'un grup. Això ho fem així i no buscant-los cada cop que els necessitem de la base de dades per accelerar el funcionament de l'app, ja que en cas contrari es farien molts accesos a la base de dades.

La classe User no ha estat gaire modificada respecte l'entrega prèvia, només s'ha afegit un atribut per guardar la URL de descàrrega de la imatge del perfil del Firebase Storage.

Hem afegit la classe SaveSharedPreferences que s'encarrega de la informació de Login del dispositiu on està l'app. D'aquesta manera l'usuari no haurà de iniciar sessió cada cop que entra a l'APP. Fa servir la classe `SharedPreferences`, té mètodes per guardar les dades de l'usuari loggejat (`setLoggedIn`), per comprovar si s'han guardat les dades d'un usuari registrat en aquest dispositiu (`getLoggedStatus`) i per obtenir les dades guardades de l'usuari loggejat, les quals són el seu `userID` i la seva contrasenya, per fer-les servir en el `LogInActivity` i loggejar-lo automaticament.

També hem inclòs una classe per guardar les dades d'un deute concret, que és `MemberDebt`. Aquesta inclou com a atribus l'ID de l'usuari que l'ha de pagar, l'ID de a qui l'ha de pagar, la quantitat a pagar, la data on s'ha donat a saldar deutes que ha fet que sorgeixi aquest deute, així com el ID del deute. Cada deute té un ID únic que

consisteix en el nom del grup on s'ha donat a saldar deutes més el moment on ha ocorregut aquest OnClick en mil·lèsimes de segon.

També hem creat una classe anomenada UserContacts per emmagatzemar les relacions entre els usuaris de l'APP i, permetent mostrar-li missatges a l'usuari sobre altres usuaris amb el nom amb els quals els té afegits a contactes. Compta amb mètodes per carregar parells de dades usuari-contacte, i dos Maps un de userID-contacte i viceversa per facilitar la cerca. Els contactes es carreguen al moment inicial en obrir l'APP.

Com hem mencionat a l'entrega anterior, les classes acabades amb Repository (junt amb SettleGroupBills) s'encarreguen de connectar amb la base de dades, per subministrar la informació. Ara tractarem les classes noves que s'han creat i els canvis que han patit les que ja estaven.

Pel que fa a BillRepository, hem afegit diversos listeners per poder transmetre les dades un cop les obtenim de Firebase, que com ja hem menciona en la entrega prèvia, es donen en segon plà i per això són tan necessaris.

Entre les funcionalitats noves es troba: crear una nova despesa, creant un Map amb les dades de la despesa, que són les de la classe Bill més els membres, el nom i ID del grup on s'ha creat. Si l'usuari havia escollit una imatge, la pugem al Firebase Storage, si és un èxit cridem a la funció que obté la URL de descàrrega de la imatge i si aquesta operació ha estat exitosa cridem a la funció que finalment crea la despesa. En cas que no s'hagués escollit una imatge, es crida directament a la funció de creació de la despesa. Aquesta crea el camp de la despesa a la col·lecció Bills, l'afegeix a GroupBills com despesa del grup on s'ha creat, així com la posa com a notificació a cada membre de la despesa afegint-se-la amb el nom i ID de la despesa i del grup corresponent. Cridem al listener que adverteix l'usuari que s'ha acabat l'operació amb un enter que indica l'estat de la operació, per mostrar el missatge corresponent o acabar el procés.

Tenim un mètode per escoltar els canvis en bills, si s'ha creat o eliminat algun d'un grup concret, que crida un listener sobre Firebase i si hi ha hagut canvis crida al listener passat per paràmetre.

Un mètode per carregar les despeses d'un usuari concret des de la base de dades que es fa servir per veure l'històric de despeses que hi ha al perfil. Aquest recorre totes les despeses en Firebase i retorna a través d'un listener passat per paràmetre en un ArrayList aquelles que té l'usuari com a membre (és a dir, només si l'usuari ha participat en la despesa).

Un mètode per eliminar les notificacions de noves despeses, que es fa servir quan un usuari accepta la notificació. Aquesta elimina de firebase la notificació del Map de bills al document de l'usuari a la col·lecció billsNotifications. Ho fa primer trobant tots els bills de l'usuari i després crida un mètode que actualitza el Map sense el bill (updateNewBillNoti).

Un mètode, addNewBillNoti, per crear notificacions de despeses que és fa com sempre, per crear elements a Firebase.

loadNewBillsNoti, per carregar les notificacions d'un usuari i després es passa l'ArrayList amb el Map que representa cada una a través d'un listener passat per paràmetre.

Un seguit de mètodes listenFirestoreChanges, listenFirestoreChangesAux, listenFirestoreBillChanges, que fent servir implementacions similars a les anteriors, escolten les despeses d'un grup a GroupBills. Si s'ha afegit alguna, s'escolta a Bills per així poder saber en l'APP quan s'ha saldat un deute en temps real.

Pel que fa respecte a GroupRepository, hi ha un mètode per eliminar una despesa que en línies generals va eliminant la despesa de tots els camps on hi té informació, això és: GroupBills, Bills, billNotifications, que un cop eliminat el Bill es crida a un listener passat per paràmetre. També elimina la imatge relacionada del Firebase Storage.

També tenim un mètode per eliminar grups similar al d'eliminar Bills. Un mètode per crear nous grups (addGroup), que és similar al que tenim per crear noves despeses, fent servir cadenes de mètodes auxiliars que es criden al listener d'OnSuccess cada cop que finalitza una tasca. També tenim mètodes per escoltar quan l'usuari es troba en un nou grup com amb les despeses (listenFirestoreChanges).

Cal mencionar que compta amb mètodes per carregar, afegir, eliminar i actualitzar les notificacions de grups nous, i un mètode que és per unir un grup amb un usuari pel cas on l'usuari accepta entrar al grup on se l'ha volgut afegir. Afegeix el groupId al document de l'usuari de UserGroups i al camp del grup membres afegeix l'userID.

S'ha implementat un mètode nou anomenat GroupExists, que comprova si ja existeix un grup amb el mateix ID que el que es vol crear. Si es així, mitjançant un listener passat per paràmetre notificarà l'error, no permetent així que se sobreescriui el grup ja existent.

En referència a la classe UserRepository, l'única modificació substancial respecte la versió anterior és que a la hora d'afegir un usuari afegim els documents que li corresponen a notifications, billsNotifications i settledNotifications amb Map respectius de cada un com a camp.

La nova classe que es comunica amb la base de dades és SettleGroupBills s'encarrega de la informació relacionada amb saldar deutes i les notificacions associades.

Conté com a funcionalitats, calcular el deute d'un grup i marcar-ho com a saldat, també, notificar a través d'un listener passat per paràmetre els canvis en el nombre de pagaments a rebre/realitzar d'un usuari concret, que la implementació és similar al altres mètodes, ja vists. La particularitat més destacable es que un cop eliminat un deute passa payments per fer-ne ús per a l'històric de pagaments. Un mètode que recopila tots els pagaments, ja realitzats amb els que està involucrat un usuari, com a qui ha de pagar o ser pagat. Ho fa recorrent tots els documents de payments i a través d'un listener passat per paràmetre retorna la llista de Maps amb aquests pagaments. També té mètodes per eliminar i carregar les notificacions de settledNotifications, tant

per agafar les dels pagaments a realitzar (que són les d'un document) i les dels pagament a rebres (que s'agafen passant per cada document els deutes que tenen l'usuari actual com a destinatari). També tenim un mètode per obtenir el balanç de cada membre d'un grup que recorre les despeses de Bills i per les que són d'aquest grup, al Map del balanç de cada usuari sumem el que han gastat en cada despesa i restem el que han pagat.

Ara la funció de calcular el deute (getGroupDebts) consisteix a calcular el balanç del grup en una altra funció que crida al listener que se li ha passat per paràmetre, que hem definit prèviament a la funció de saldar deutes getGroupDebts. En el OnGetDebts d'aquet listener cridem un mètode que retorna qui ha de pagar a qui amb un ArrayList de MemberDebt, seguint l'algorisme mostrat a la primera entrega. Afegim l'ID de cada MemberDebt amb la data actual fins als milisegons. Per així que hi puguin haver dues accionaments al saldar deutes seguits. Finalment, afegim cada MemberDebt a la base de dades a settledNotifications.

3.2. Viewmodel + view overviews

La part de view s'encarrega de mostrar la informació que proveeix el model a través de la base de dades i interactuar amb l'usuari. 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.

Només tractarem les classes noves o amb modificacions destacables respecte a la entrega anterior, aquestes són les següents classes:

MainActivity: s'ha afegit un codi que sol·licita els permisos a l'usuari per accedir als contactes i si accepta carreguem els seus contactes a UserContactsLocal associant telèfon (coincideix amb l'ID del usuari) amb nombre. També comprovem amb el

SaveSharedPreferences que si hi ha dades d'un Login previ les recuperem i iniciem sessió amb elles a l'Auth de Firebase.

SignUpActivity: s'ha afegit el ProgressBar que es mostra quan es prem el botó de registrar-se, es comproba que la contrasenya introduïda compleixi les normes i es posen els texts i camps en vermell si no s'han introduït bé les dades.

LoginActivity: s'ha afegit a la funció de login que si no s'han guardat les dades del Login a SharedPreferences és guardin, també hem afegit que es posi els camps de telèfon i contrasenya en vermell en cas que l'Auth de Firebase ens doni un login fallit.

Cal mencionar que ara es posa la ProgressBar i s'oculta el text del botó de loggejar per mostrar que s'està processant la petició.

HomeActivity: hem afegit la gestió dels botons de notificacions de manera que hem afegit un listener OnClick sobre cadascun. S'obre un DialogFragment que hem personalitzat per depeses noves, grups nous, pagaments pendents o a rebre. També cridem a mètodes de BillsViewModel i GroupViewModel que criden a mètodes de Repository per escoltar a Firebase canvis en el nombre de notificacions d'un usuari i ho avisa a un listener que hem definit al HomeActivity per així saber si hi ha hagut un canvi en el nombre i actualitzar el número en el text del botó que indica el nombre de notificacions. També hem afegit un OnTextChangedListener al buscador així filtrem cada cop que s'escriu un caràcter. Es defineix la gestió d'un listener nou del GroupCardAdapter (per eliminar grups) per cada ítem, el qual es crida després d'un OnLongClick i mostra un dialog per confirmar l'eliminació del grup per cridar el mètode del ViewModel que crida al mètode GroupRepository per eliminar grups.

GroupCardAdapter: s'ha afegit un listener que al bind del ViewHolder es crida el seu mètode quant s'ha fet un OnLongClick al card. També es recupera la URL de la imatge del grup. Si aquesta és nul·la, a l'ImageView es pinta el logo, si no, la imatge corresponent.

GroupActivity: s'ha afegit la gestió del Click al botó de saldar deutes, que quan es polsa creem un listener que es cridarà un cop ja es té qui ha de pagar quant a qui, on mostrem amb un dialog a l'usuari. Aquest mostra a qui ha de pagar i la quantitat, o si no ha de pagar res o no hi ha deutes a saldar. Segons els retorn com a parametre del handler del listener (que son a qui ha de pagar que és null sino ha de pagar a ningú i si els deutes ja havien estat saldats).

A la transició entre fragments a cada transició buidem la pila de fragments, per així al tornar enrere no tenir varies transicions entre fragments. També cridem de BillViewModel el mètode per escoltar el grup actual i dins del listener que es crida si hi ha canvis, actualitzem el objecte Group actual que tenim guardat, per així actualitzar instantàniament si un usuari accepta entrar en el grup.

També s'ha inclòs mostrar en vermell el nom del grup en cas que es vulgui deixar buit i també es posa la ProgressBar i s'oculta el text del botó de crear grup per mostrar que s'està processant la petició.

BillsFragment: no hem tocat gaire el codi d'aquesta classe. Principalment cridem el mètode de BillViewModel que defineix un listener sobre els bills d'un grup de manera que si s'afegeix un es tornen a reomplir el MutableLiveData de despeses i del balanç.

BillCardAdapter: no cal entrar en detalls, el funcionament és anàleg al GroupCardAdapter i només s'ha afegit l'OnLongClick.

BalanceFragment: nova classe implementada de forma molt similar a BillsFragment. Realitzem el procés estàndard per omplir el seu RecyclerView amb BalanceCardAdapter com a adapter.

NewGroupActivity: en aquesta classe s'ha implementat l'opció d'afegir una imatge de grup. Quan es prem l'editText per afegir la imatge, s'obre un Dialog, que permet escollir entre els arxius del telèfon i la càmera. Quan es recupera la imatge dels arxius s'introdueix la seva adreça interna a l'editText només per donar retroacció a l'usuari de la compleció del procés. Si la imatge ha estat obtinguda a partir de la càmera, primer de

tot es comprova que tingui l'orientació adient, modificant la imatge si escau. Per últim es comprimeix i es posa l'adreça a l'editText també. En el moment de prémer el botó de crear grup, s'indica en vermell si no s'ha afegit un nom de grup i, en cas contrari, es crida el mètode del GroupViewModel per afegir el grup. Es recuperen tots els casos erronis mitjançant listeners i es mostren missatges per pantalla explicant els motius, com també es demanen els permisos adients si no es tenen.

NewBillActivity: conté dos mètodes per crear despeses que criden a mètodes del BillViewModel per crear despeses, un pel repartiment equitatiu i un altre pel personalitzat. Té un mètode per transicionar del fragment NewBillConfigFragment a NewBillPersonalizedFragment i NewBillUsersFragment. Aquests són cridats de NewBillConfigFragment en funció de si es vol un repartiment equitatiu o personalitzat. També guarda les dades que introdueix l'usuari a NewBillConfigFragment per enviar-les al crear la nova despesa.

NewBillConfigFragment: omple l'spinner de qui paga amb els membres del grup actual, que els obté de l'objecte Group de **NewBillActivity** així com l'spinner de la forma de pagament, un cop es polsa següent agafa les dades les guarda a la activity pare (concepte, pagador, uri de la imatge) i en funció del mode de pagament crida la funció per transicionar a un fragment o a un altre.

Quan no s'introdueix el camp del concepte de la despesa es posa en vermell i permet a l'usuari seguir en la creació del grup. Aquí es tracta la selecció de la imatge exactament igual que en NewGroupActivity.

NewBillPersonalizedFragment: inflem la vista *fragment_new_bill_personalized* i l'afegim a LinearLayout del Fragment per cada membre del grup, i hi posem un OnTextChanged Listener de manera que en un ArrayList guardem el valor actual per cada membre del grup (el valor que posem a l'inici per defecte és de 0). Quan es polsa crear cridem el mètode per crear la despesa de NewBillActivity de forma personalitzada.

NewBillUsersFragment: inflem la vista *fragment_new_bill_users* i l'afegim a LinearLayout del Fragment per cada membre del grup i hi posem un OnClicked al

checkbox que canvia l'estat true o false del Map dels usuaris que participen en la despesa, que inicialment tots estan a false. Quan es polsa crear cridem el mètode per crear la despesa de `NewBillActivity` de forma equitativa amb els membres participants.

ProfileActivity: s'ha afegit el `OnClick` per obrir les activitats `PaymentsActivity` i `ExpensesActivity` i l'obtenció de la imatge de perfil actualitzada a partir de l'URL que guarda l'usuari.

PaymentsActivity: realitzem les accions habituals per omplir el `RecyclerView` de la vista associada i hi fiquem com a adapter `PaymentsCardAdapter`. Carreguem tant els pagaments en els quals ha estat emissor com receptor, com ja s'ha explicat.

PaymentsCardAdapter: Similar a tants Adapters implementats, per aquesta raó no hi entrarem en detalls, només mencionar que no hi tenim cap listener.

ExpensesActivity: amb una estructura similar a `GroupActivity`, conté un parell de botons per transicionar entre fragments, `HistBillsListFragment` i `GlobalAnalysis`, on es mostrarà per defecte el primer. Compta amb un listener a cada un dels dos botons per mostrar cada Fragment i atributs com l'ID de l'usuari actual que es fan servir per omplir les dades del recycler de cada fragment amb les de l'usuari actual. Es mostren les despeses en les quals l'usuari ha estat partícep, com ja ha estat esmentat.

HistBillsListFragment: crida a la cadena d'instruccions usuals per inicialitzar el `RecyclerView` que té la seva vista `fragment_hist_bills_list` (hi defineix un layout manager, ho associa a l'adapter, definen els mètodes de gestió dels listeners, en aquest cas cap, i els observers sobre la referència de l'`ArrayList` que conté les dades) i l'omple amb les despeses on l'usuari ha estat partícep. Per fer-ho crida `loadBillsHistFromRepositor` del ViewModel `HistorialViewModel` que ha obtingut de la activitat pare. Fa servir d'adapter `BillHistCardAdapter`.

BillHistCardAdapter: un simple adapter anàleg a molts altres que hem vist durant l'assignatura, de fet és ben similar al `BalanceCardAdapter`. No compta amb cap listener,

ja que no necessitem escoltar cap interacció amb l'usuari amb les cards. Infla les cards *member_card_layout_bills_hist*.

GlobalAnalysisFragment: omple el spinner amb les 4 opcions de filtrat del total de diners gastats en el període marcat per l'opció seleccionada. Aquests són Total, Semanal, Mensual i Anual. Després omplim el RecyclerView de la vista que infla el OnCreatedView (que és *fragment_global_analysis*) i posem un listener de manera que si canvia l'element seleccionat del spinner, tornem a carregar les dades del recycler amb el nou filtre. També implementem un listener que ens ofereix HistorialViewModel per obtenir el total de despeses participades de tots els grups que es mostra a sota cada cop que es carreguen de nou el total de les despeses de cada grup, així mantenim el total actualitzat a la vista.

EditProfileActivity: en aquesta classe s'ha afegit el mateix funcionament d'obtenció d'una imatge nova pel perfil que en el cas de grups i bills. Això sí, s'emmagatzema en aquesta pròpia classe la imatge al Firebase Storage, permetent així la imeditesa de la modificació de l'ImageView en el moment que es carrega. S'obté el URL de descàrrega en el OnSuccess i en el OnComplete de l'obtenció d'aquest, si ha sigut exitosa es pinta directament l'ImageView amb la imatge i es crida als mètodes adients per actualitzar la base de dades.

ChangePassword: només s'han introduït les noves normes de contrasenya i que es mostrin en vermell els camps erronis en prémer el botó. No s'ha obtat per afegir ProgressBar, ja que és instantani.

No entrarem en els dialogs, en comptar aquests amb una implementació i estructura bastant simple i similar. No són més que Fragments dels quals omplim els Recyclers de les vistes que 'inflen', amb informació carregada de la base de dades a través del Model amb el ViewModel com a intermediari.

Pel que fa als ViewModels, son les classes que s'encarreguen de realitzar la comunicació Vista-Model, en el nostre cas, en la anterior entrega comptàvem amb quatre: GroupViewModel, BillViewModel, BilMembersViewModel i UserViewModel. A

continuació mencionarem els canvis que han patit i el nou ViewModel `HistorialViewModel`.

No entrarem gaire en detall, ja que la majoria actuen com a intermediaris entre la Vista i el Model implementant listeners del Model per tasques de segon pla, principalment la càrrega de dades de Firebase.

GroupViewModel: s'ha afegit un `MutableLiveData` per les notificacions de nous grups de l'usuari i mètodes que implementant els listeners necessaris, realitzen tasques com escoltar els canvis en els grups de l'usuari de `GroupRepository`. *AddListenerFirebaseChanges*, que al listener que es crida en haver canvis recarrega l'`ArrayList` dels grups. En tenim un altre similar per escoltar els canvis en les notificacions dels grups nous d'un usuari, un mètode per eliminar grup, un per afegir un usuari a un grup que crida el corresponent de `GroupRepository` per les notificacions de nous grups i també tenim els mètodes estàndards amb una implementació similar a la que s'ha fet en altres casos (carregar les notificacions d'un usuari, eliminar una notificació). També hem afegit un control que en el `MutableLiveData` de Groups no hi hagi dos grups amb el mateix ID, solucionant el problema de duplicació dels ítems del `RecyclerView` (que venia un cop canviava la referència de l'array de `MutableLiveData` respecte del Adapter).

UserViewModel, no ha patit gaires canvis, principalment mètodes intermediaris però obtenir el nom del contacte a partir del id de l'usuari o viceversa, que fan d'intermediaris entre la Vista i `UserContactsLocal`. S'ha afegit també el mètode `updateURL` per actualitzar la URL de la imatge de perfil de la base de dades.

BillViewModel, s'han afegit `MutableLiveData` pel balanç dels membres d'un grup, per la llista de pagaments a rebre i una altra pels pagaments a realitzar i per les despeses noves d'un usuari, que es mostren als dialogs. Tots compten amb les funcions estàndards requerides, (carregar les dades de Firebase fent servir els mètodes corresponents de `GroupBillRepository` o `SettleGroupBills`), amb un listener passat per paràmetre per actualitzar l'`ArrayList` un cop s'hagi completat l'operació en segon pla.

Tenim mètodes per escoltar si hi ha noves despeses d'un grup, perquè en cas que n'hi hagi cridem al mètode per recarregar els elements del `MutableLiveData` de les despeses i del balanç dels membres del grup. També té el mètode que es crida per escoltar cada despesa del grup, que crida el mètode corresponent de `GroupBillRepository`, per escoltar les notifiacions d'un usuari, tant de les noves despeses com si s'han saldat nous deutes (escoltant `settledNotifications` de l'usuari a `settleGroupsBills`). També un per escoltar els canvis en un grup concret, que si recordem es feia servir als `Fragments` del `GroupActivity` per saber quan un membre ha acceptat entrar en un grup, i actualitzar-ho automàticament.

Es poden trobar mètodes per eliminar despeses, per crear despeses, crear notifiacions de deutes, eliminar notifiacions de deutes (tant pels pagaments a rebuts i fets), de despeses, etc. Tots aquests mètodes actuen purament com a intermediaris.

Cal mencionar que ara també controlem que no hi hagi dos `Bills` amb el mateix id solucionant el problema de duplicació dels ítems del `RecyclerView`, com amb el `MutableLiveData` de grups.

`BillMembersViewModel`, no ha patit cap canvi respecte la versió anterior.

`HistorialViewModel`, conté la informació de 3 col·leccions de dades, la que mostra l'historial de pagaments, el llistat de despeses i el total de quantitat de despeses de cada grup d'un usuari concret en les quals ha participat. Ho fa amb `MutableLiveData`, similars als que hem vist en la entrega anterior i en la actual, per tant, no entrarem en detalls. Cadascun té un mètode, per carregar la informació amb `ArrayList` del `MutableLiveData`, que criden als mètodes del Model corresponents. Prèviament es defienixen els listeners que es passen per paràmetre al cridar aquestss mètodes, però on rebrem el nou `ArrayList` i el posem al `MutableLiveData` respectiu de cada un. Per la quantitat de despeses per grup filtrades per data cridem un listener definit i implementat fora (*`mOnGetExpensesTotalListener`*) cada cop que es crida el mètode handler del listener que és cridat un cop es carreguen els totals de cada grup filtrats

(*OnGroupsExpenses*), i calcula el total, que s'envia a través del listener *mOnGetExpensesTotalListener*. Amb això s'obté el total a *GlobalAnalysisFragment*.

3.3. Base de dades

Les bases de dades representen un element fonamental d'una aplicació, ja que permeten emmagatzemar grans volums de dades de manera eficient, evitant la necessitat de mantenir-les carregades constantment. La seva principal funció és proporcionar una persistència de dades fiable.

Les Firebase Database és on han estat emmagatzemada la base de dades que fa servir el model.

No s'ha modificat cap aspecte respecte la metodologia que se segueix guardant la informació del registre i inici de sessió. La col·lecció que recopila els usuaris tampoc ha patit canvis.

Sobre la col·lecció dels grups s'ha afegit un Array amb els membres del grup, per reduir el nombre d'accesos per obtenir els membres d'un grup i, així, tenir un millor rendiment de l'APP.

Pel que fa a les despeses, la col·lecció anomenada Bills, hem afegit en un map els membres que la despesa i quant havien d'haver pagat, un String amb l'ID de l'usuari que ha pagat així com un booleà de si el deute de la despesa ha estat saldat. També hem inclòs dos atribus, un pel nom del grup de la despesa i un pel seu ID, així reduïm el nombre d'accesos per obtenir aquesta informació, altrament hauríem de recórrer *GroupBills* fins trobar un grup que tinguit com a despesa la despesa actual i, després, buscar la informació del grup, cosa que vam notar que ralentitza la informació de l'APP.

A la col·lecció de notificacions dels grups (notifications) tenim un document per cada usuari i els grups nous que se li ha intentat afegir en un Array de Maps, on cada un

conté la data del nou grup, l'ID (per si accepta poder cridar la funció per afegir-ho) i el nom del grup per mostrar-ho.

Per la col·lecció de notificacions de noves despeses (billNotifications), com amb les de grups tenim un document per cada usuari i guardem l'ID i nom de la nova despesa i del grup associat en un Array de Maps.

En relació a la col·lecció de notificacions de deutes saldat, guardem per cada usuari en un document un array, on cada element d'aquest és el deute que ha de pagar on guardem la quantitat, l'ID del destinatari i l'ID del deute. Aquest últim és el nombre del grup + el moment on s'ha donat en milisegons, permetent així que l'ID sigui únic, ja que mai es podran saldar deutes en un grup de forma tan seguida.

La col·lecció payments guarda tots els deutes entre membres (un per document), amb l'ID de qui ha pagat, a qui li ho a pagat, la data i la quantitat pagada, però un cop l'usuari que rep el pagament o qui l'ha de realitzar indiquen que s'ha donat ja. Així, podem mantenir un historial dels pagaments donats, i per l'historial de payments filtrem pels que involucren a l'usuari corresponent.

Pel que fa a les relacions entre grups-usuaris i grup-despeses, hem implementat una opció que barreja la simulació de la estructura de les taules SQL i tenir als camps de cada grup o despesa alguna informació com els membres. Internament mantenim una coordinació entre ambdues llistes, cosa que es senzilla. Així tenim els avantatges de totes dues opcions.

Per emmagatzemar les imatges dels grups, les despeses i els usuaris s'ha fet servir l'eina Firebase Storage.

S'ha creat un bucket on hi ha una carpeta anomenada images/. Dins d'aquesta hi ha tres carpetes, una per cada àmbit, anomenades profiles/, bills/ i groups/. En cadascuna d'elles, les imatges es guarden amb els identificadors únics als quals estan associades. En el cas dels perfils, tenen l'identificador de l'usuari, en el cas dels grups, el del grup i

en despeses, la despesa. Així, no es poden sobre escriure imatges de grups per imatges de despeses o de perfils.

3.4. 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.

Mencionar que hem evitat posar listeners i classes internes en general per evitar saturar més el diagrama de classes, i que pugui ser llegible.

4. TESTING PLAN

4.1. Unit testing

Les proves a nivell unitari es basen en tests enfocats a classes, funcions i components. Validen de forma molt exhaustiva cada aspecte de una classe concreta, portant-la fins a l'extrem per a intentar trobar “bugs” que no hagin estat tinguts en compte a la codificació. És un error fer poques proves unitaries, ja que acaben suposant més defectes en el codi en el futur i es tornen més difícils de trobar.

Nosaltres, durant el desenvolupament d'una classe o funcionalitat hem intentat tenir en compte tots els casos possibles que podrien generar error i els hem anat preveient.

Per posar un exemple, en la nostra classe User tenim uns quants atributs que es poden testejar:

- Comprovar que el telèfon introduït existeixi i que tingui el format correcte.
- Comprovar que el prefix introduït correspongui a algún país.
- Comprovar que el mail introduït existeixi i que tingui un format vàlid.
- Comprovar que la contrasenya segueixi els paràmetres de seguretat mínims.

Un altre exemple, podria per la funcionalitat de crear un nou deute, on podem testejar com a casos extrems:

- Comprovar que s'hagi introduït un concepte de la despesa.
- Comprovar que el preu de la despesa no sigui negatiu.
- Comprovar que no hi hagi problemes en que pagui un usuari i la despesa sigui d'un altre
- Comprovar que no hi hagi problemes en que un usuari sigui l'únic membre d'una despesa.

No entrarem en aquests casos per totes les funcionalitats i totes les classes del codi, però el procés es similar, testejar amb funcions especials alguns casos generals i anar provant casos extrems.

4.2. Integration testing

El test per integració avalua la forma de comunicar-se de diversos mòduls del software, que poden haver estat creats per diferents programadors, per tal de comprovar que es comporten com a una única entitat correctament.

El test per integració s'ha de dur a terme després de fer el test unitari, ja que es més fàcil detectar errors si abans centrem els test en punts més petits. Per a dur a terme aquests test per integració, es poden utilitzar dos mètodes: utilitzant totes les classes a la vegada, o agafant dues classes (o funcionalitats), després tres, després quatre, fins a integrar totes les funcionalitats del codi i fer un test general de tot el projecte.

En el nostre cas, es pot posar com a exemple la interacció de les classes User i Bill. Es té que un objecte de la classe Bill ha de tenir un string amb un ID igual al ID d'un objecte de la classe User. Havent avaluat el codi i provat amb diferents casos, podem concloure que aquestes dues classes funcionen correctament juntes.

4.3. Performance Testing

El test de rendiment avalua numèricament com rendeix l'APP en quant als resultats esperats del nostre software i la seva estabilitat sota un espai de treball concret.

En el nostre cas podem fer un test de memòria i CPU mitjançant l'Android Studio. Volem fer el màxim de tasques seguides que puguin generar molt de rendiment per a comprovar com actuen sobre el dispositiu.

Hem fet una prova durant uns 7 minuts de temps on el rendiment de la CPU màxim voltava el 49%, mentre que el rendiment de la memòria és una mica més variant. Hem extret que en el primer llançament de l'APP es consumeixen 161,6 Mb de memòria, Mentre estem al login de l'APP ens trobem a uns 64,9 Mb d'ús. A l'iniciar sessió, el consum de memòria augmenta fins als 88,6 Mb i així es manté aproximadament fins acabar l'execució.

4.4. Usability testing

En aquest test ens interessa obtenir feedback de diferents tipus d'usuaris per a obtenir un ventall suficientment gran com per a arribar a unes conclusions reals de la nostra APP que ens ajudin a millorar-la.

La nostra aplicació està destinada a tot el món, però, com ja s'ha mencionat diverses vegades, està centrada en persones joves, i no s'espera el seu ús en persones d'edat avançada. Així doncs, definirem el nostre target com a les persones d'entre 18-30 anys. Descartem les persones menors d'edat, en tenir poques despeses grans com viatjes per donar un ús realista a la APP. Enquestarem 50 persones ja que, segons diversos estudis a partir de 20 - 30 persones s'obté un amb un 95% aproximat de coneixement l'opinió dels usuaris respecte l'APP. Els rangs seran els següents:

- 5 persones entre 14 i 17 anys
- 30 persones entre 18 i 30 anys
- 10 persones entre 31 i 40 anys
- 5 persones entre 41 i 60 anys

No ens és important respecte a les persones enquestades cap dada sobre els seus nivells d'estudis ni econòmics, així com tampoc ens cal fer una distinció de gènere, ja que la classe social a la que pertanyin els usuaris i el seu sexe no haurien d'influir a l'hora d'escollir a qui li fem. Si que ens importaria que els usuaris visquessin a Catalunya, doncs seria el lloc on treuriem l'APP inicialment.

Respectet a les restriccions que posarem a la nostra mostra seràn les següents:

- Voldriem que hagi una equitat a nivell de gènere.
- Tenir una varietat respecte el seu estatus economic.
- Tenir un volum que sigui estudiants, altres que treballin, etc.

Una manera molt ràpida d'arribar a molta gent seria a través de les xarxes socials. Per Instagram mateix, els 5 de l'equip sumem més de 3000 seguidors, amb un percentatge per rangs d'edat similar als desitjats. Com no tenim recursos econòmics per contractar

els serveis d'una empresa dedicada als estudis de mercat hauríem de fer ús dels nostres contactes per arribar a la quota d'usuaris desitjada. A més a més, en el cas de que ens faltessin usuaris, sobretot en el rang entre 18 i 30 anys, d'on són la majoria, un bon entorn per escollir persones a l'atzar seria en diferents facultats i de diferents estudis.

Una consideració a fer a l'hora de seleccionar els usuaris és que ens interessa trobar gent sense tant interès per a les noves tecnologies, a qui li pugui costar una mica més usar l'APP per a poder saber si la app és intuïtiva i fàcil d'utilitzar, per això s'han escollit persones d'un rang d'edat més avançada.

Per a fer els test, donaríem fins a 15 minuts perquè els enquestats fessin ús de l'aplicació i es familiaritzessin una mica amb ella i després es farien unes preguntes, incloent les següents:

1. L'aplicació és fàcil/intuïtiva de fer servir per a vostè? Valora-ho del 1 al 10.
2. Creu que l'aplicació funciona amb la suficient rapidesa per poder usar-la amb comoditat? Valora-ho del 1 al 10.
3. El disseny de l'aplicació és adequat? Valora-ho del 1 al 10.
4. Com valora la seva experiència general al fer servir l'aplicació? Valora-ho del 1 al 10.
5. Creu que necessitaria ajuda per a trobar les coses (per exemple pop-ups que expliquessin que fan els botons o on el duen)?
6. Que troba més difícil de l'aplicació?
7. Hi ha hagut algun aspecte que l'hagi pogut frustrar durant la seva prova?
8. Que troba més fàcil o l'ha sorprès positivament de l'aplicació?
9. Canviaria algun aspecte visual de l'aplicació?
10. Preferiria que la pròpia APP permetés fer els pagaments, que el dugués a una altra aplicació per a pagar (com pot ser la de dintre de l'app o que només li notifiqués quant ha de pagar o rebre)?
11. Confia en la APP i la gestió podria arribar a fer de les teves dades, sobretot si l'APP permetés fer els pagaments?

12. Utilitzaria l'aplicació de forma freqüent?

13. Hi ha algún aspecte extra que vulgui comentar?

Les respostes de les preguntes de l'1 a la 4 aporten informació numèrica sobre la facilitat d'ús de la APP, la rapidesa, el disseny i la UX general.

Les respostes de les preguntes de la 5 a la 9 aporten informació concreta sobre aspectes a millorar per aconseguir una millor UX.

Les respostes de la pregunta 10 i 11 ens ajudaran a decidir el futur de l'APP, les de la pregunta 12 si tindria èxit al mercat i les de la pregunta 13 ens podrien donar alguna informació adicional.

Al fer totes aquestes preguntes, en el cas de ser presencials, s'ha de donar també importància a les expressions facials o comentaris que faci l'usuari a l'hora d'usar l'APP i que ens indiquin si està gaudint o no de l'experiència amb l'APP.

5. PROBLEMES TROBATS

Durant la implementació d'aquesta pràctica ens hem anat trobant amb diversos problemes. La majoria els hem anat solucionant buscant informació al respecte a fòrums a internet, llegint la documentació relacionada amb les parts que ens donava problemes i depurant el nostre codi fins a trobar les fonts d'errors.

Però no ens posarem a tractar aquests problemes i ens centrarem en la problemàtica de les notificacions en segon pla i de registrar-se.

A l'hora de registrar-se a l'aplicació, ens hem topat amb un problema intern del reCAPTCHA, que hem estat incapaços de solucionar. La part estranya és que sí que funciona en alguns dispositius, però no en altres. Per exemple, funciona a l'emulador de l'integranent del grup David Fernández i al mòbil personal del Soufiane Lyazidi (com vam veure en la sessió de laboratori), però no en el mòbil personal del David Fernández ni a l'emulador del Víctor Sort. L'error és el següent:

- E/FirebaseAuth: [SmsRetrieverHelper] SMS verification code request failed: unknown status code: 18002 Invalid PlayIntegrity token; app not Recognized by Play Store.
- D/FirebaseAuth: Re-triggering phone verification with Recaptcha flow forced for phone number +34625208277

...

- E/FirebaseCryptoHelper: Exception encountered during crypto setup:
 - Keystore cannot load the key with ID: firebear_master_key_id.W0RFRkFVTFRd+MToyNDM3OTk4NDMwMDM6YW5kcm9pZDowYjIhYWJhMjQxZGJkZTI5Yjk4OTIi
- E/FirebaseCryptoHelper: KeysetManager failed to initialize - unable to get Public key
- E/RecaptchaActivity: Could not generate an encryption key for reCAPTCHA - cancelling flow.

...

- E/zzh: Failed to get reCAPTCHA token with error [An internal error has occurred. [Failed to generate/retrieve public encryption key for reCAPTCHA flow.]]- calling backend without app verification

...

- E/FirebaseAuth: [SmsRetrieverHelper] SMS verification code request failed: unknown status code: 17093 null

Aquest error indica que com l'aplicació no és registrada al Google Play, cal verificar mitjançant reCAPTCHA, però no troba la clau pública del KeysetManager.

En un primer moment, tal i com diversos fòrums explicaven, vam pensar que era problema de les claus SHA-1 i SHA-256 del nostre projecte, però vam veure que sí estaven ben afegides. Més tard, vam descarregar el .json per comprovar que tinguéssim l'última versió, sense cap canvi. Vam provar d'afegir dependències noves, més actualitzades, etc. i tampoc va funcionar. Aquí presentem una solució no satisfactòria, però que almenys permet registrar-se sense fer servir el mecanisme intern del Firebase Firestore de rebre SMS amb un codi.

Si s'accedeix a la consola del Firebase i s'accedeix al projecte MoneySplitter. En l'apartat d'Authentication, a sign-in-method, premem a l'apartat de "Teléfono".

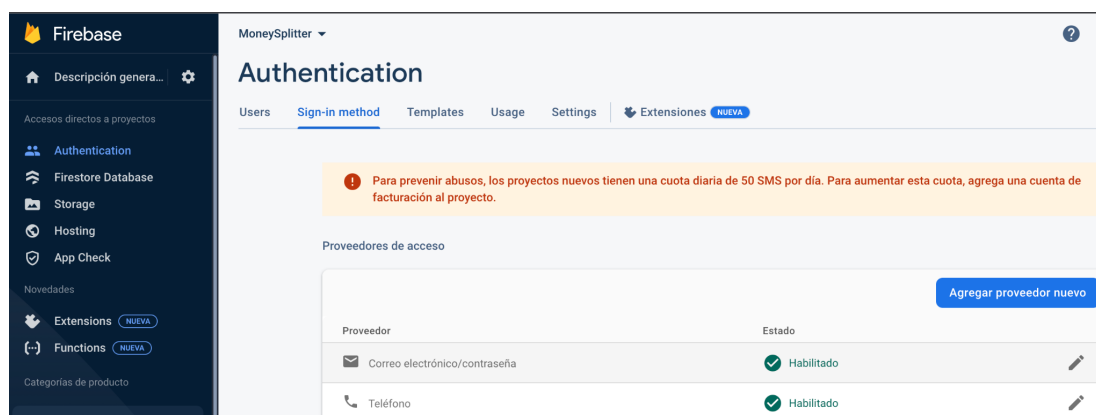
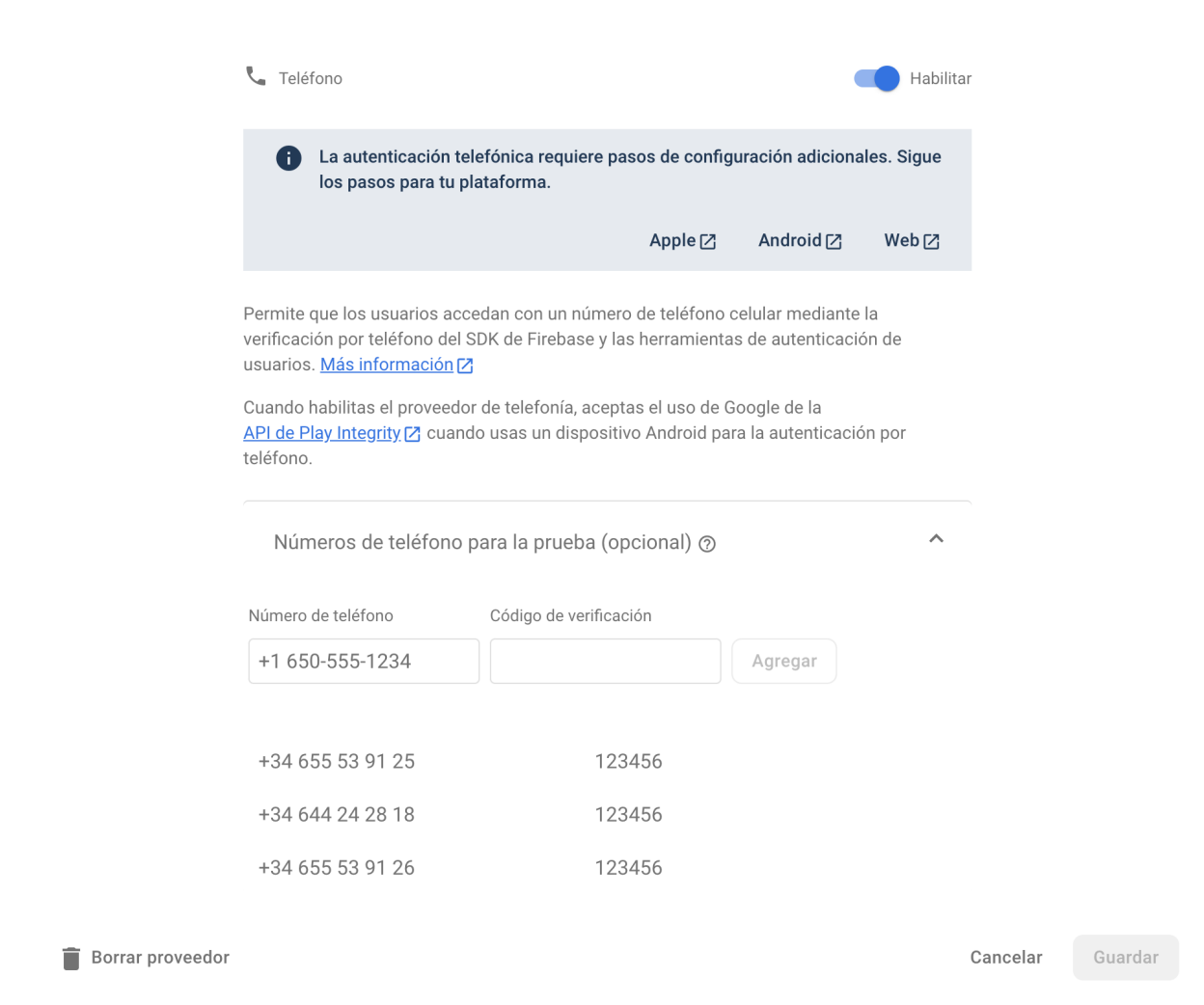


Figura 24: Apartat on s'ha d'accedir de la Firebase Console

Un cop fet, es pot afegir el número que volem registrar i el codi de verificació que voldríem que arribés per SMS.



Teléfono Habilitar

i La autenticación telefónica requiere pasos de configuración adicionales. Sigue los pasos para tu plataforma.

[Apple](#) [Android](#) [Web](#)

Permite que los usuarios accedan con un número de teléfono celular mediante la verificación por teléfono del SDK de Firebase y las herramientas de autenticación de usuarios. [Más información](#)

Cuando habilitas el proveedor de telefonía, aceptas el uso de Google de la [API de Play Integrity](#) cuando usas un dispositivo Android para la autenticación por teléfono.

Números de teléfono para la prueba (opcional)

Número de teléfono	Código de verificación	
<input type="text" value="+1 650-555-1234"/>	<input type="text"/>	<input type="button" value="Agregar"/>
+34 655 53 91 25	123456	
+34 644 24 28 18	123456	
+34 655 53 91 26	123456	

Borrar proveedor Cancelar Guardar

Figura 25: Menú on es poden afegir els telèfons i els codis de verificació

Pel que fa al tema de les notificacions en segon pla, com vam mencionar diversos cops a la primera entrega al ser un usuari afegit en un nou grup, una nova despesa, s'hagi saldat un deute on està involucrat se li enviarà una notificació en segon pla. Ara bé, a l'hora d'intentar implementar-ho la opció que vam trobar per enviar notificacions entre dispositius en segon pla va ser Firebase Messaging. En intentar traslladar aquesta opció al nostre codi, tant a la documentació de Firebase com als diversos tutorials a YouTube i fòrums incloïen disposar d'un servidor per enviar les notificacions a Firebase

Messaging a través d'ell, ja que per crear el que Firebase anomena sol·licituds d'enviament, que requereixen de l'ús de la classe Message del Firebase admin sdk, però aquest admin sdk només s'en pot fer ús en un servidor.

A la documentació de Firebase menciona explícitament que els servidors de Firebase Cloud Messaging compten amb dues components el backend de FCM proporcionat per Google i el teu servidor d'apps o un entorn de servidor de confiança.

(<https://firebase.google.com/docs/cloud-messaging/manage-topics?hl=es-419> ,
<https://firebase.google.com/docs/cloud-messaging/android/topic-messaging?hl=es-419&authuser=0> i
<https://firebase.google.com/docs/cloud-messaging/server?authuser=0&hl=es-419>)

6. ANÀLISI DELS REQUERIMENTS INICIALS

En aquesta secció es troben els requeriments funcionals i no funcionals que vam proposar-nos a la primera entrega i s'avaluarà l'acompliment dels funcionals. No s'avaluen els no funcionals al no tenir les eines necessàries. Per fer-ho de manera visual, s'utilitzarà el següent codi de colors:

- **Verd**: requeriment acomplert.
- **Groc**: requeriment casi acomplert.
- **Taronja**: requeriment no acomplert.
- **Vermell**: requeriment rebutjat.

Requeriment funcional	Descripció
Registre	Els usuaris podran registrar-se fent servir el seu nom, cognoms i número de telèfon.
Inici de sessió amb telèfon	Els usuaris podran iniciar la sessió a l'aplicació introduint només el seu número de telèfon i contrasenya.
Inici de sessió amb correu electrònic	Els usuaris amb un correu electrònic vinculat al compte podran iniciar la sessió a l'aplicació introduint només el seu correu associat al compte i contrasenya.
Inici de sessió amb identificador	Els usuaris podran iniciar la sessió a l'aplicació introduint només el seu identificador associat al compte i contrasenya.
Llistar grups	Els usuaris podran veure una llista amb tots els grups on hi formen part.

Crear grup	Els usuaris podran crear un nou grup introduint un nom, una descripció, una imatge i els membres del grup.
Visualitzar balanç de grup	Els usuaris d'un grup podran veure el balanç de cada membre del grup.
Visualitzar despeses	Els usuaris d'un grup podran veure una llista amb les despeses que s'han donat al grup de forma detallada.
Imatge de grup	Els usuaris membres d'un grup podran canviar la imatge d'aquest grup.
Afegir despesa	Els usuaris membres d'un grup podran afegir una despesa introduint un concepte, quin membre ha pagat i com s'ha de repartir entre els membres la despesa.
Saldar deutes	Els usuaris membres d'un grup podran saldar els deutes, sempre que els altres membres del grup acceptin.
Perfil	Els usuaris podran veure al seu perfil les seves dades personals de les quals l'aplicació té accés.
Canvi de contrasenya	Els usuaris amb un correu vinculat al compte podran canviar la seva contrasenya.
Imatge de perfil	Els usuaris podran afegir i canviar la seva imatge de perfil.
Llista de	Els usuaris podran veure una llista de totes les transaccions

transaccions	que han realitzat mostrant a l'usuari a qui les han realitzades i la data de les mateixes.
Despeses per grup	Els usuaris podran veure una llista amb la quantitat de diners que han anat gastant en cada grup, mostrant el nom del grup i els diners invertits.
Històric de despeses	Els usuaris podran veure una llista amb l'històric de les seves despeses, mostrant el grup al qual pertanyen aquestes despeses, el seu context, la quantitat gastada i la data.

La justificació del perquè hem rebutjat no poder iniciar sessió amb el correu o l'identificador (de moment) no és pas per la complexitat que això té (doncs és completament anàlog a iniciar sessió amb el telèfon) sinó perquè de moment no verifiquem el correu electrònic, tal i com si fem amb el telèfon, i és una manera d'afegir seguretat a l'APP. Un cop en el futur s'implementi la verificació del correu, la implementació d'aquestes funcionalitats serà molt ràpida.

Respecte la funcionalitat de saldar deutes, al no haver implementat les notificacions externes per la seva excessiva dificultat hem omés la part d'acceptació dels usuaris d'un grup per saldar els deutes. A més a més, en aquesta versió de l'APP considerem que no és del tot necessari, doncs no gestionem diners. Aquest requeriment cobrarà molt més sentit per versions on implementem nosaltres mateixos un sistema de pagament.

Requeriment no funcional	Descripció
Seguretat i privacitat	L'aplicació respectarà la normativa europea sobre el tractament de dades personals.
Transparència	Els usuaris seran informats, als termes i condicions de l'aplicació, sobre la informació del seu telèfon a la qual es tindrà accés i les raons per la qual aquesta es necessita.
Rendiment	L'aplicació no trigarà en carregar més de 5 segons cap activitat.
Usabilitat	L'aplicació serà fàcil d'usar i intuïtiva pels usuaris.
Internacionalització	L'aplicació es podrà utilitzar en diferents idiomes, per permetre l'ús de més usuaris.

Respecte els requeriments no aconseguits, no ho estan perquè no tenim les eines necessàries per assolir-los (assessoria legal, estudis de mercat, etc.). Això principalment per la Usabilitat i Seguretat i privacitat, encara que considerem que els hem aconseguit no és suficient, per exemple, per la Usabilitat faltaria realitzar un testeig d'usabilitat a una mostra de 20-30 persones com hem mencionat en l'apartat anterior.

Respecte el rendiment, hem provat l'APP en 3 mòvils i 3 emuladors i compleix el temps màxim de càrrega.

Respecte la internacionalització, tot i que l'APP només s'ha fet en castellà, en tenir tots els strings identificats en un sol document seria senzill canviar l'idioma, doncs només s'hauria de referenciar un altre document amb la traducció dels strings, en lloc d'haver de modificar un a un cada string.

7. FEINA PEL FUTUR

Tot i que en aquesta entrega hem aconseguit assolir pràcticament tots els requeriments inicials, trobem un parell de millores que es podrien fer en un futur.

Primer de tot, és evident que caldria implementar el registre amb un mètode de Firebase Firestore que fos compatible amb tots els dispositius i no donés errors. Això es podria solucionar registrant l'APP al Play Store, que evitaria l'ús del reCAPTCHA, però no hem tingut els mitjans per fer-ho.

Un aspecte a millorar seria el procés en esborrar grups. En aquesta entrega, si un membre d'un grup esborra un grup del seu Home, esborra el grup per tots els altres membres. Es podria implementar de forma més complexa però més pràctica, eliminant aquest usuari del grup automàticament obligant-lo a obtenir balanç nul.

Després, una millora que podríem implementar en un futur és la d'eliminar usuari, que de fet, ja hi ha un botó assignat a aquesta funció, però no hi és implementat. Aquesta funció esborraria l'usuari de tots els grups, només si salda tots els seus deutes abans.

Una millora respecte a l'històric de despeses seria mostrar el valor exacte que l'usuari ha gastat en cada despesa, no tot el valor de la despesa en la qual ha participat. Així, el valor de la despesa seria un reflex directe dels diners gastats per l'usuari.

Finalment, caldria poder rebre notificacions en segon pla, que ja ha estat intentat, però no aconseguit, a causa de la necessitat d'un servidor extern.

El fet de gestionar els pagaments des de l'aplicació quedaria subjecte també a l'opinió dels usuaris a l'estudi de mercat, a partir del qual també podríem avaluar qualsevol millora per l'aplicació.

8. CONCLUSIONS FINALS I AGRAÏMENTS

Com clarament s'ha anat podent veure en els punts anteriors, hem aconseguit crear una APP que implementa quasi la totalitat de funcionalitats que ens vam proposar en un inici. Tot i que quedaria bastanta feina per fer en el cas de voler treure l'APP al mercat en un futur (incloent tràmits legals, estudis de mercat, econòmics, estratègia de marketing..) estem molt satisfets en el producte final que presentem.

Respecte la quantitat de treball que hem invertit tant amb la realització de l'APP com en la seva concepció i la redacció dels informes podem garantir que ha sigut molt superior a la que ens esperàvem en un inici. Per una part, hem invertit molt de temps en intentar considerar tots els aspectes possibles per fer funcionar l'algorisme perfectament i de manera òptima. Per una altra part, deguda la complexitat de l'APP i totes les seves funcionalitats (que era creiem superior a l'esperada a l'assignatura) hem hagut d'invertir molt més temps a solventar diversos errors, generalment relacionats amb la base de dades.

Per finalitzar, el bon resultat del projecte i l'aplicació Money Splitter és la traducció del treball constant i la col·laboració de tots els membres del grup. Volem agrair-nos a nosaltres mateixos la bona predisposició a col·laborar en el projecte i la feina ben feta. També agrair la seva ajuda, feedback i suport al nostre professor de pràctiques Alex Dickson i al professor de teoria Karim Lekadir.

Esperem que Money Splitter pugui arribar a ser una eina útil i pràctica per a la gestió de pagaments entre amics.