CICLO FORMATIVO DE GRADO SUPERIOR
DESARROLLO DE APLICACIONES MULTIPLATAFORMA

PROYECTO FIN DE CICLO

Howse

Daniel Sabbagh Pastor

Ricardo Martínez Ingelmo

CURSO 2018-19

TÍTULO : Howse

AUTORES: RICARDO MARTINEZ INGELMO

YOSEPH DANIEL SABBAGH PASTOR

TUTOR DEL PROYECTO: Ernesto Ramiro Córdoba

FECHA DE LECTURA: 07 / 06 / 2019

En Madrid

Ernesto Ramiro Córdoba
Tutor del PFC

RESUMEN:

Howse surge a partir de la necesidad de mejorar la comunicación y la convivencia en el ecosistema de los pisos compartidos. Durante la última década se ha hecho mucho hincapié en agilizar el trámite de alquiler tanto de habitaciones como de pisos, pero no en mejorar las relaciones humanas entre inquilinos y arrendador.
Las personas que tienen que hacer uso de viviendas compartidas, suelen tener problemas a la hora de comunicar incidencias al arrendador y de organizarse, ya sea para comprar productos de uso común o para organizar tareas de limpieza. Cuando se formuló la idea de la app, se hizo énfasis en la necesidad de centrarse en la experiencia del inquilino en una casa compartida, con el objetivo de que mejorase notablemente mediante una buena comunicación con el casero en caso de necesidad, y agilizando la organización y el trabajo que supone vivir en este tipo de viviendas.
Llegamos a la conclusión de que crear un sistema de organización de tareas y tener una lista de la compra común a los que todos los inquilinos puedan acceder de manera rápida y sencilla además de tener un chat en el que poder comunicarse con el arrendador y entre los inquilinos resolvería el problema.
A parte de lo anterior mencionado creemos que si el arrendador tiene una mejor relación con sus inquilinos será más fácil para él/ella saber en qué estado está su propiedad lo que supondrá una mayor tranquilidad.

ABSTRACT:

Howse arises from the need to improve the communication and the convivence in the shared apartment ecosystem. During the last decade much emphasis has been placed on streamlining the rental process of both rooms and flats, but not in improving human relations between tenants and landlords.
People who have to use shared housing tend to have problems when communicating incidents to the landlord and to organize themselves, either to buy products of common use and to organize cleaning tasks. When we think about the idea of the app, we wanted to focus on the fact that the experience of the tenant in a shared house improved notably through good communication with the landlord in case of need, and streamlining the organization and the work involved in living in this type of households.
We came to the conclusion that create a system of task organization and have a list of common purchases to which all tenants can access quickly and easily in addition to having a chat in which to communicate with the landlord and between Tenants solves the problem.
Apart from the aforementioned, we believe that if the landlord has a better relationship with his tenants it will be easier for him / her to know in what state his / her property is, which will suppose a greater tranquility.

AGRADECIMIENTOS

Queríamos agradecer a María Pilar Martín por el apoyo que nos ha brindado este curso, gracias a ella hemos sabido superarnos en cada momento de complicación.

Agradecer a nuestras familias por hacer el esfuerzo económico y apoyarnos en cada decisión difícil.

Creemos que gracias a nuestro esfuerzo hemos conseguido muchas metas tanto personales como profesionales durante el desarrollo de Howse.

Agradecer a Ernesto Ramiro por tener tanta paciencia con nosotros y a la Universidad Europea por darnos esta oportunidad de realizar este proyecto en el extranjero.

Agradecer a Carla Gomez Sanz, estudiante de Animación 3D, diseño de Videojuegos y Entornos Interactivos en la Escuela Superior de Comunicación, Imagen y Sonido de Madrid (CEV) por ayudarnos en algunos detalles gráficos, tanto de la aplicación como en el diseño del logo.

ÍNDICE

1.INTRODUCCIÓN

El propósito principal del PFC es el de realizar una app que resuelva el problema de la la convivencia y la organización en el ámbito de la vivienda compartida. Se observó que la comunicación en este ecosistema era prácticamente nula, y era necesario abrir un canal de comunicación en forma de herramienta para organizar las tareas habituales que existen en una vivienda compartida con el fin de mejorar la coordinación entre los usuarios evitando los roces de convivencia que derivan en problemas más graves.

Además, una vez observado este problema, se estudiaron otras funcionalidades que podrían ser útiles, se decidió añadir una lista de la compra común, ya que aporta organización a la vivienda.

Otro de los propósitos es el de mejorar la comunicación entre inquilino y casero mediante un canal de comunicación exclusivo, para que en caso de cualquier imprevisto en la vivienda pueda ser solventado lo más rápido posible.

Howse nace para mejorar, agilizar y organizar el uso de las viviendas compartidas y la comunicación entre los los individuos que la forman.

1.1. Objetivos

Para alcanzar los objetivos que se han propuesto son necesarias varias herramientas. A continuación se explica el por qué de la elección de cada herramienta y su uso o propósito.

- Una base de datos en tiempo real. La herramienta más afín a lo que se necesita para este proyecto es Firebase. Las principales características por las que se ha elegido son: BB.DD. a tiempo real, ya que se quiere implementar un servicio de mensajería instantánea. Tiene un apartado para la autentificación de usuarios, por lo que la tarea de login y registro se facilita en gran medida. Con la opción Storage de firebase se han podido guardar imágenes de perfil.

- Android Studio: Se eligió esta herramienta dado que el propósito es hacer una app y en concreto que fuera nativa de Android. Esta herramienta proporciona las estructuras y las facilidades necesarias para que no sea tan tedioso construir un app desde cero.

A continuación se enumeran los objetivos y subsiguientemente su explicación.

- Innovar en el ecosistema de los pisos compartidos desde un punto de vista distinto.

- Proporcionar una plataforma en la que tanto inquilinos como arrendadores puedan mantener una comunicación mucho más efectiva.

- Hacer más eficiente las viviendas compartidas.

- Desarrollar una versión alfa de una plataforma que consiga agrupar todos los servicios de alquiler de vivienda compartida.

El principal objetivo que se propuso al empezar el proyecto era el de innovar en el ecosistema de los pisos de alquiler.

Se planteó cómo se podía mejorar de una manera efectiva y certera desde el punto de vista de la tecnología y en concreto a través de una app, pero también desde una perspectiva más humana en la que primara la relación entre las personas.

Gracias a proponer este primer objetivo surgió de inmediato el segundo. Tener una plataforma para hacer una comunicación efectiva.

Tener un chat específico para comunicarse parece esencial si se quiere dar la importancia necesaria a las relaciones entre inquilinos y arrendador.

El objetivo de hacer más eficiente las viviendas compartidas, se alcanzará al conseguir los objetivos anteriores ya que este es resultado de Innovar en este campo y en proporcionar una comunicación más efectiva.

Por último el objetivo de que el TFC fuera una versión alfa de una app con mucho más alcance y recorrido para poder tener una visión más amplia del futuro de las viviendas de alquiler. Es un objetivo a medio-largo plazo.

1.2. Motivación

La idea de hacer la app surgió durante el tiempo de prácticas formativas en centros de trabajo en Londres,donde tuvimos que alquilar un piso.

Nos dimos cuenta de que no había una aplicación que nos proporciona facilidad alguna para que pudiéramos comunicarnos con el casero y organizarnos con los inquilinos, la mayoría de los servicios que necesitábamos se encontraban muy dispersos en las tiendas de aplicaciones de android e iOs(1.3 Antecedentes) . Por ese motivo decidimos embarcarnos en el proyecto teniendo en mente que una vez finalizado el TFC queríamos seguir desarrollándola.

1.3. Antecedente

A continuación se darán algunos ejemplos de las apps con las que debemos competir:

Trello, que es un sistema de gestión de tareas, aunque no está enfocado al ámbito de las viviendas sino de tareas en general.

OurGroceries, es una app para hacer una lista de la compra común ya que informa que productos se están acabando o es necesario comprar.

WunderList, es una app para organizar tareas y hacer una lista de la compra con tus compañeros de piso y notifica cuando se han realizado cualquier acción.

Howse se encarga de agrupar todas estas utilidades (tareas, lista de la compra, chat) en una sola app con el objetivo de hacer el uso de una vivienda compartida más fácil. La Innovación viene a partir de la necesidad de tener una comunicación más efectiva con el arrendador y tener agrupadas en una sola app los servicios que hacen más fácil la vida en una vivienda compartida.

2.DESARROLLO DEL PROYECTO

En este punto vamos a describir las tecnologías usadas en Howse:

- Tanto tecnologías de programación, Bases de Datos noSQL, como las tecnologías de comunicación usadas (GitHub, Slack...).

- Como se ha gestionado la planificación a lo largo de 2 meses, con una variada repartición de tareas.

- Descripción detallada del funcionamiento de todas las pestañas, incluyendo partes del código con gran relevancia en Howse.

- Resultados de tests realizados, junto con errores de gran importancia que se han ido solucionando según avanzaba la aplicación.

2.1. Herramientas tecnológicas

Para realizar este proyecto hemos tenido que lidiar con varios tipos de tecnologías diferentes:

- Android Studio: Ha sido nuestra herramienta principal, donde hemos construido nuestra aplicación desde cero.
  Android Studio es el entorno de desarrollo integrado para la plataforma android, dicha herramienta incluye también su propio sistema de emulación que permite ver los cambios realizados en nuestra aplicación cuando ejecutemos la app.
  En primer lugar podemos dividir la programación de esta herramienta en tres partes:

  - XML: Toda la parte visual de la app está programada en XML (Frontend) mediante una interfaz gráfica que genera el código, el cual se puede modificar manualmente.
    XML (Lenguaje de Marcado Extensible), es un lenguaje de marcas parecido a HTML pero con etiquetas determinadas por el mismo programador, en este caso determinadas por Android ya que tiene que estar controlado por la lógica del código en Java.

  - JAVA: Es el código predominante en Android Studio, la mayoría de librerías y clases internas están programadas en Java, estas clases, son las que le dan una gran funcionalidad a este entorno de desarrollo ya que facilitan su uso y hacen mucho más legible a la hora de generar el código de la aplicación.
    Los archivos ".java" (Backend) son los que dan funcionalidad a los archivos ".xml"
    Java es un lenguaje de programación orientado a objetos y concurrente que permite a los programadores reutilizar código previamente hecho sin necesidad de escribir constantemente lo mismo.

  - Gradle: Es un sistema de automatización para la construcción de código abierto que consigue una configuración adaptada a los gustos del desarrollador en su proyecto.

- Firebase: Es una base de datos NoSql en la nube, a tiempo real dividida en tres partes importantes, dichas partes se manejan mediante librerías ya programadas para facilitar la consulta e inserción de datos desde Android Studio:

    - Authentication (autentificación): Aquí es donde se guardan todos los datos de autentificación (existen muchos métodos para realizar la autentificación)

    - RealtimeDatabase: Una vez guardado los datos de autentificación , es posible que sea necesario almacenar datos. Mediante una organización en forma de árbol registramos cada dato con su clave principal y sus datos ramificados.

    - Storage (almacenamiento): Su uso principal es para guardar archivos y poder acceder a ellos posteriormente.

- Adobe XD: Es un editor de gráficos desarrollado por Adobe Inc, para crear el diseño y el prototipo (mokup) de la experiencia de usuario para la aplicación.

- GitHub: Es una herramienta con muchas funcionalidades, tanto para compartir código con diferentes desarrolladores, como para permitir programar en equipo.
  Se suele utilizar en equipos de programadores para que todos los integrantes tengan acceso al mismo código y puedan actualizarlo.

2.2. Planificación

REPARTICIÓN DE TAREAS POR SEMANAS

SEMANA 1 (1 - 7 Abril)

- RICARDO y DANIEL:
  - Nombre y Diseño (paleta colores) de Howse
  - Mokup (Adobe XD)
  - Estructuración Howse

SEMANA 2-3 (8 - 21 Abril)

- RICARDO:
  - Estructuración Firebase
  - Diseño Login registro
  - Diseño y funcionalidad del SplashScreen, clasificación casero/inquilino y validación del código de casa
- DANIEL:
  - Funcionalidad login
  - Funcionalidad registro, diferenciación de casero/inquilino en Firebase
  - Diseño login registro

SEMANA 4-5 (21 Abril - 5 Mayo)

- RICARDO:
  - Diseño y funcionalidad del chat
  - Diseño del perfil
- DANIEL:
  - Funcionalidad del perfil
  - Diseño del perfil

SEMANA 6-7 (5 - 19 Mayo)

- RICARDO:
  - Diseño tareas
  - Funcionalidad visualización y borrado tareas
- DANIEL:
  - Funcionalidad de creación de tareas y actividades

SEMANA 8-9 (19 Mayo - 2 Junio)

- RICARDO:
  - Funcionalidad y diseño la Lista de la compra (creación ítems e historial de compra y visualización)
  - Funcionalidad y diseño del menú navigation bar inquilino
  - Funcionalidad y diseño menú superior
- DANIEL:
  - Solución de errores e implementación detalles (ProgressDialog, cambio de constraint a relative layout)
  - Funcionalidad del perfil-arrendador y menu navigation bar Arrendador

SEMANA 10 (2 - 7 Junio)

- RICARDO y DANIEL:
  - Documentación
  - Arreglos de errores
  - Implementaciones de detalles

Los principales canales de comunicación usados han sido GitHub, Slack y Whatsapp, a continuación se hará una breve explicación de su uso:

- GitHub: Usado para poder programar varias personas en un mismo código mediante ramificaciones del código principal, donde cada persona actualiza su rama en la rama principal cuando finaliza lo requerido en dicha rama:

  Una vez creado el proyecto en GitHub la única rama por defecto que existe es "Master", para realizar una ramificación (Branch) se debe haber actualizado (Pull) lo que la persona tiene en local con lo que existe en la nube.
  Ahora que se ha creado una rama, se le da un nuevo nombre y cada vez que quieras mezclar tu contenido (nueva rama) con el contenido de "Master" (rama principal) se debe unir (Merge into current), para ello se debe haber actualizado nuestra rama en la nube (hacer commit and push de la nueva rama); en primer lugar, se debe actualizar lo modificado con un mensaje que diferenciará una modificación de otra y se guardará en la carpeta "Git" del ordenador (Commit) y por último debes actualizar tu rama en la nube (Push).

- Slack: Es una herramienta Online que se ha usado para entrar en contacto con nuestro tutor del PFC y para compartir tanto ideas como opiniones con los compañeros de 2º de DAM
- Whatsapp: Aplicación móvil básica para mantener conversaciones online con cualquier poseedor de un smartphone.
  Ha sido usada para compartir código e ideas entre los desarrolladores de Howse.

2.3. Descripción del trabajo realizado

Howse se ha desarrollado en su totalidad en Android Studio y FireBase. A continuación se hará una descripción detallada de cómo, de qué manera y por qué se han utilizado estas tecnologías.

FIREBASE

Se ha decidido utilizar Firebase ya que para el funcionamiento óptimo de la aplicación era necesario una base de datos en tiempo real. Destaca la facilidad de uso y la buena compenetración con Android Studio ya que no hace falta sentencias SQL para manejar datos.

La estructura de Howse en Firebase se basa principalmente en un atributo que sirve como hilo conductor para poder diferenciar a cada grupo de usuarios (inquilinos y arrendador) y todos los datos que acompañan a estos, este dato es el **Código de la casa** (CodCasa), se ha elegido esta manera de organizar los datos para minimizar la aparición de errores.

La estructura de datos en FireBase es la siguiente:

- USUARIOS: Los dos tipos de usuarios que existen son Inquilino y Arrendador, diferenciados por el atributo booleano (tipoUs) que indica si es de un tipo o de otro. Los demás atributos son los mismos para ambos usuarios
  Cuando se registra o hace login un usuario de tipo inquilino se requiere un código de casa, el cual se le asignará posteriormente como atributo y que es indispensable para el funcionamiento.

- CHATS: En él están los mensajes guardados de todos los chats que contienen el mensaje, el destinatario y el usuario que envía el mensaje.

- TAREAS: Se guardan las actividades semanales que tienen que hacer los usuarios. Cada tarea tiene asignada un usuario, un código de casa, un nombre (Actividad) y otros datos de interés para los usuarios.

- ACTIVIDADES: Howse ofrece la posibilidad de añadir tareas personalizadas por lo que el nombre de cada tarea (Actividad) debe ser guardado junto a código de casa para poder después consultarlo.

- ARTÍCULOS: Los artículos de compra contienen un usuario, un código de casa, un precio orientativo del precio y otros datos de interés para los usuarios.

- HISTORIAL DE COMPRA: En el historial de compra se guarda una lista con todos los elementos de la compra en el momento de hacer una nueva lista la fecha actual en la que se ha guardado la lista y el código de casa al que pertenece.

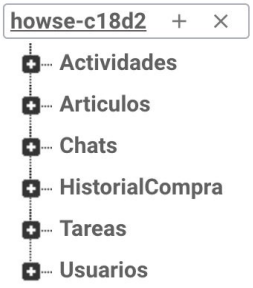


Figura 1

A continuación se dará una descripción detallada del código en Android Studio, explicando la estructura y el flujo de datos de Howse:

Se harán tres casos de uso desde el inicio de la aplicación hasta que se haya iniciado sesión diferenciando el tipo de usuario para ilustrar de manera efectiva el completo funcionamiento de la app; en el primer caso, un nuevo usuario de tipo inquilino, en el segundo un nuevo usuario de tipo arrendador y por último un usuario (sin diferenciar el tipo) que ya se ha registrado y que haya hecho log in previamente.

1. En el primer caso de uso después del SplashScreen se le requerirá al usuario, si es inquilino o arrendador. Cuando se seleccione la opción de inquilino (en este momento se le asigna el booleano tipoUs=false, en el caso de inquilino tipoUs=true) se pasará a una pantalla en la que se tendrá que introducir un código de casa válido para poder avanzar.
Una vez introducido un código se avanza a la pantalla del login en la que se tendrá hacer uso del botón para crear una nueva cuenta.
En la pantalla de registro, se solicita el nombre y el apellido junto con el correo y la contraseña, existe la opción de subir una imagen de perfil aunque no es obligatorio (los campos mencionados anteriormente si lo son), una vez que el inquilino lleva a cabo el registro, le lleva al log in para que pueda introducir sus datos previamente introducidos en Firebase (Autentificación).

En este caso de uso se arrastra el tipo de usuario y el código de casa por las activities antes mencionadas hasta el momento en que se haga login.

2. El segundo caso de uso en el que se escoge la opción de arrendador, directamente se pasa a la pantalla de log in. debido a la naturaleza del tipo de usuario no es necesario que introduzca un código de casa ya que en el momento del registro se le asignará uno.
En este caso también se debe arrastrar el tipo de usuario.
Cuando se haya hecho efectivo el registro en Firebase se generará un código de casa único y aleatorio que permitirá a los inquilinos hacer log in y registrarse con ese código (Listado 1)

```
474     public String crearCodCasa(){
475
476         char[] conjunto = new char[6];
477         char[] elementos={'0','1','2','3','4','5','6','7','8','9' ,'a',
478                 'b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t',
479                 'u','v','w','x','y','z'};
480
481         for(int i=0;i<6;i++){
482             int el = (int)(Math.random()*36);
483             conjunto[i] = (char)elementos[el];
484         }
485         return codigoCasa = new String(conjunto);
486     }
```

Listado 1: Generación única y aleatoria del código de casa

3. En el tercer y último caso de uso se utiliza el método onStart () situado en el activity de elección del tipo de usuario. Dado que dependiendo del tipo de usuario se debe dirigir a un Activity u otro, en el SplashScreen se hace uso de la función de CurrentUser de FireBase, para subsiguientemente hacer una consulta y poder obtener el tipo de usuario que está iniciando la aplicación. La razón por la cual se hace uso este método en el splashScreen es porque si se ejecutase en la misma pantalla que el método onStart () daría un error (nullPointerExcepcion) ya que no da tiempo a que se recojan los datos de FireBase. Una vez se obtiene este dato (tipoUs) se decide mediante un if () a qué activity debe dirigirse (Listado 2).

```
35      @Override
36      protected void onStart() {
37          super.onStart();
38          tipoUsuario=getIntent().getBooleanExtra( name: "tipoUs", defaultValue: true);
39
40          if(firebaseUser!=null){
41              if(tipoUsuario){
42                  Intent i= new Intent( packageContext: ActivityCaseroInquilino.this, PreviewDelChat.class);
43                  startActivity(i);
44                  finish();
45              }else{
46                  Intent i= new Intent( packageContext: ActivityCaseroInquilino.this, PreviewChatArrendador.class);
47                  startActivity(i);
48                  finish();
49              }
50          }
51      }
52
```

Listado 2: Método "onStart ()"

Con respecto al Activity Login, se solicita el correo y la contraseña del usuario ya registrado, que se valida haciendo una consulta en la parte de autentificación de Firebase usando el método "signInWithEmailAndPassword ()" (Listado 3).

Una vez efectuado el login dependiendo del tipo de usuario, se iniciará sesión:
- Arrendador: Se muestra un menú con dos opciones: chat y perfil .
- Inquilino: Muestra un menú con cuatro opciones: tareas, lista de la compra, chat y perfil.

```
auth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener( activity: Login.this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()) {

                    if (tipoUs){
                        Intent i = new Intent( packageContext: Login.this, PreviewDelChat.class );
                        startActivity( i );
                    }else{
                        Intent i = new Intent( packageContext: Login.this, PreviewChatArrendador.class );
                        startActivity( i );
                    }
                } else {
                    Toast.makeText( context: Login.this, text: "El email o la contraseña no es correcta", Toast.LENGTH_LONG ).show();
                }
            }
        });
```

Listado 3: Método "signInWithEmailAndPassword ()"

MENÚS

Como se ha aclarado anteriormente existen dos menús inferiores que cambian dependiendo del tipo de usuario:

En caso del inquilino una vez se ha efectuado el login se pueden observar dos menús, uno inferior estático (bottom navigation bar) en el que se diferencian cuatro opciones desde donde se navegará por las utilidades de la aplicación.

En lo que se refiere al menú superior se observa que va cambiando según se va moviendo por las pestañas del menú inferior mostrando diferente información.

Existe una opción constante en todas las modalidades del menú superior que es la opción de poder hacer log out, la cual sirve para desconectar el usuario logueado y llevarle a la pantalla del login.

En caso del arrendador el menú superior es el mismo que en el caso anterior, la diferencia se sitúa en el menú inferior en el que sólo hay dos opciones por las que poder navegar.

FUNCIONALIDADES DE HOWSE

En este apartado se explicarán toda la funcionalidad de la aplicación. Se dividirá en cuatro partes (una por cada funcionalidad).

TAREAS: Es solo accesible para los inquilinos, consiste en la organización de tareas semanales. Visualmente es simple, consistiendo en un recyclerView formado cards. Cada una de las cards contiene un título con cada día de la semana (en los que haya tareas) y un recyclerView con un listado de las tareas que se deben hacer en ese día.
En cada elemento de la lista de tareas se puede observar cual es la tarea el nombre del usuario que debe realizarla y una foto la foto de perfil de este ya que puede que existan dos inquilinos con el mismo nombre, por último situado el la parte derecha se encuentra un botón con el cual se puede eliminar la tarea.

Se ha elegido la posición vertical en el recyclerView principal ya que es muy probable que la parte inferior de la pantalla se quede vacía.

En lo que se refiere al código es la parte más compleja de la aplicación ya que es necesario tener dos recyclerView anidados.

Primero se consulta en FireBase las tareas que pertenecen a ese código de casa. Una vez se obtengan las tareas se añaden y clasifican en Arraylist distintos (por días de la semana, cada día de la semana es una lista diferente).
Una vez están separadas por día de la semana se crean objetos de tipo "DiasTareas" que están formados por un título (día de la semana) y un Arraylist de objetos de "TareasVisual" (Se creado este tipo de objeto con el único objetivo que sea más manejable a la hora de crear el recyclerView).

En el caso de que alguna de las listas de tareas de un cierto día esté vacía no se creará el objeto "DiasTareas" con el objetivo que no aparezcan días de la semana sin tareas. se debe mencionar que los días de la semana siempre aparecen ordenados ya que en el código se le ha forzado a ello (Listado 4).

```
if(tareasLunes.size()!=0){

    DiasTareas diasTareasL = new DiasTareas( dia: "Lunes", tareasLunes);
    tareasDias.add(diasTareasL);

}
if(tareasMartes.size()!=0){

    DiasTareas diasTareasM = new DiasTareas( dia: "Martes", tareasMartes);
    tareasDias.add(diasTareasM);

}
if(tareasMiercoles.size()!=0){

    DiasTareas diasTareasX = new DiasTareas( dia: "Miercoles", tareasMiercoles);
    tareasDias.add(diasTareasX);


}
if(tareasJueves.size()!=0){
    DiasTareas diasTareasJ = new DiasTareas( dia: "Jueves", tareasJueves);
    tareasDias.add(diasTareasJ);


}
if(tareasViernes.size()!=0){
    DiasTareas diasTareasJ = new DiasTareas( dia: "Viernes", tareasViernes);
    tareasDias.add(diasTareasJ);


}
if(tareasSabado.size()!=0){
    DiasTareas diasTareasS = new DiasTareas( dia: "Sabado", tareasSabado);
    tareasDias.add(diasTareasS);


}
if(tareasDomingo.size()!=0){
    DiasTareas diasTareasD = new DiasTareas( dia: "Domingo", tareasDomingo);
    tareasDias.add(diasTareasD);


}
```

Listado 4. Clasificación semanal

Por otra parte en la esquina inferior derecha se encuentra un menú flotante (librería externa) el cual tiene dos opciones. La primera consiste en añadir una tarea, esta opción te dirige a un activity que está formada por tres spinners: día de la semana, inquilino al que se le asigna la tarea y por último la actividad a realizar y tres botones: añadir tarea, añadir actividad y guardar cambios. El botón de añadir tarea solo se habilita en caso de haber seleccionado opciones válidas cada uno de los spinners, el botón de añadir actividad muestra un "popUpActivity" en el cual se puede añadir una actividad personalizada. Por último el botón guardar cambios devuelve al usuario a la visualización de las tareas. En la segunda opción del botón flotante se puede borrar todas las tareas que han sido asignadas hasta el momento (esta opción solo se ejecuta en caso de que haya alguna tarea).

CHAT: Es una funcionalidad común a los dos tipos de usuarios. La arquitectura de esta parte se compone de un TabLayout con dos opciones (fragments) y un menú superior que contiene la foto y el nombre del usuario actual conectado. Ambos fragments están formados por un recyclerView en el que aparecen un listado de usuarios con los que se pueden conversar (solo aparecen los usuarios con los cuales se comparte el mismo código de casa). La diferencia entre ambos fragments es la de que en el primero aparecen los usuarios con los que ya se ha mantenido una conversación mientras que en el segundo aparecen todos los usuarios con los que se puede chatear.

Esta clasificación de los usuarios se ha conseguido mediante una consulta en FireBase, en caso de que se encuentren mensajes cuyo destinatario y remitente coincidan con el usuario actual del sistema y otro usuario con el mismo código de casa entonces se añadirá dicho usuario al primer fragment. En el segundo fragment se ha hecho una consulta más simple, se ha filtrado por código de casa y se han añadido los usuarios.

Para identificar con claridad al arrendador se ha puesto un textView en la parte derecha del ítem del recyclerView en el que lo indica con la palabra "casero".
En ambos recyclerView se encuentra el método "onOptionsItemSelected ()" por el que cuando se pulsa el ítem de un usuario se abre la ventana del chat seleccionado.
Se cargan los mensajes correspondientes, en el caso del usuario con el que se esté manteniendo una conversación todos sus mensajes aparecerán junto a un CircleImageView (Librería externa) con su foto de perfil.
Los mensajes del destinatario aparecerán en blanco mientras que los del remitente se verán del color principal de la aplicación.
La transmisión de mensajes entre usuarios es instantánea gracias a FireBase.

En la parte superior de la conversación se puede observar que hay un menú en el que se encuentra una flecha con la que se puede volver a la previsualización del chat, un CircleImageView (Librería externa) y un textView con la imagen de perfil del destinatario y su nombre.

En lo referido a la disposición de los mensajes el layout es un recyclerView donde cada vez que se añade un mensaje a la conversación se añade un item de tipo "chat_item_right" o "chat_item_left" dependiendo si el usuario envía o recibe un mensaje.

```
if(viewType==MSG_TYPE_RIGHT){

    View view= LayoutInflater.from(mContext).inflate(R.layout.chat_item_right, parent,  attachToRoot: false);

    return new MessageAdapter.ViewHolder(view);
}else {

    View view= LayoutInflater.from(mContext).inflate(R.layout.chat_item_left, parent,  attachToRoot: false);

    return new MessageAdapter.ViewHolder(view);
}
```

Listado 5. método onCreateViewHolder del adaptador de los mensajes

LISTA DE LA COMPRA: Es una funcionalidad exclusiva de los inquilinos. La vista consiste en un recyclerView compuesto por cards formados por un título, un precio (opcional) y la foto y el nombre del usuario que ha añadido el elemento a la lista.

También se encuentra en la parte superior derecha de cada card un botón con el que se puede eliminar el elemento deseado de forma automática.

En la parte inferior derecha se encuentra un menú desplegable (FloatingactionMenu librería externa) en el que encontramos tres opciones:

Crear una nueva lista, añadir un elemento a la lista y el historial de compra.
Si se selecciona la primera opción (Crear una nueva lista) aparecerá un cuadro de diálogo que informa de que se borrará la lista pero se guardará en el historial de compra los elementos que hay actualmente en ella.
En caso de que se confirme la acción se cogerá el nombre de cada elemento y se guardará en un String junto a la fecha actual del sistema.

En el caso de la segunda opción aparecerá un "popUpActivity" donde se requiere el nombre del artículo (campo obligatorio) y un precio orientativo (opcional).
Cuando se añade un elemento a la lista, en la base de datos se guarda junto a los atributos antes mencionados, los datos del usuario que lo ha añadido y el código de casa.Para que se actualice automáticamente la lista y aparezca automáticamente se usa el método cargarArticulos () en el método onCreate () del activity principal (ListaCompraActivity).

En la tercera opción se encuentra el historial de compra, mediante un recyclerView donde se puede consultar todas las listas ya realizadas, ordenadas cronológicamente.

PERFIL

Con respecto al Activity de Perfil se cargan el nombre, el apellido, el correo y la foto de perfil.
Dicha foto es editable y pulsando en ella accedes a la galería para cambiarla.
Existe un botón flotante situado en la parte inferior derecha de la pantalla que siendo pulsado, habilita los campos de Nombre y Apellido (Editext Enabled), una vez editados los campos, se debe pulsar el botón (Guardar Cambios) para efectuar la actualización en Firebase y deshabilitar los campos nuevamente (Editext Disabled).

Por último en el perfil también se carga el código de la casa, por si el usuario necesita iniciar sesión en otro dispositivo o el arrendador quiere facilitárselo a sus inquilinos, a la derecha de este EditText (código de la casa) podemos pulsar el icono para copiarlo en el portapapeles del dispositivo.

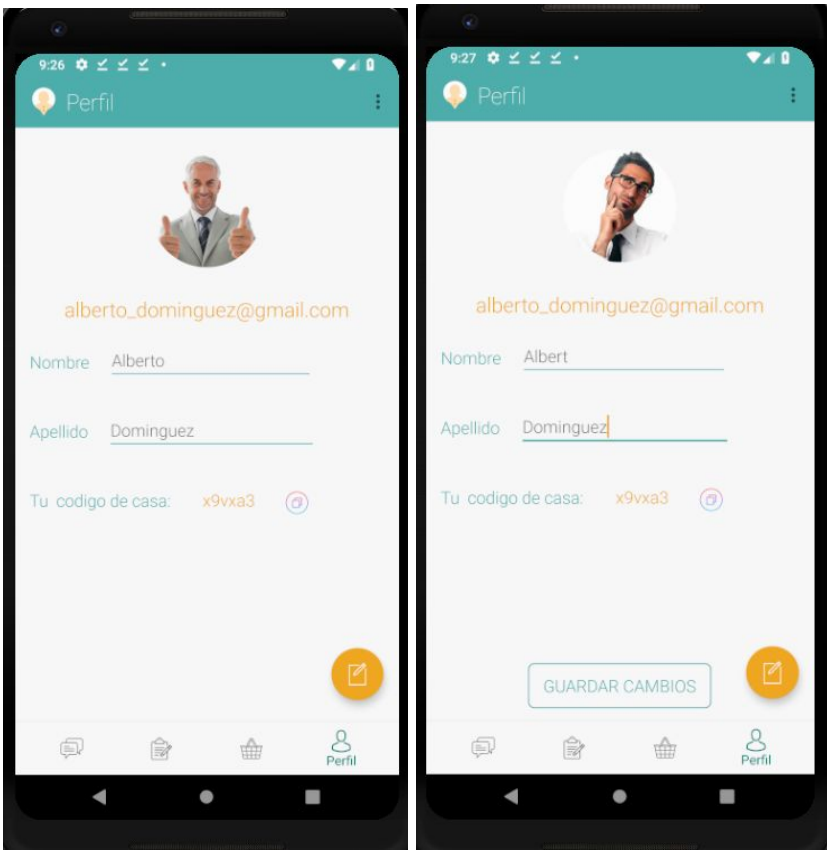
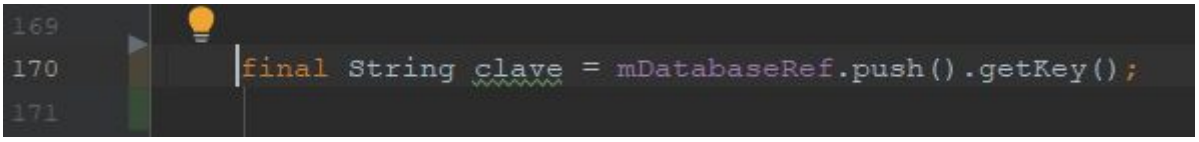


Figura 2 y 3

2.4. Resultados y Validación

Howse es una aplicación enfocada a la organización de viviendas de alquiler donde existen 2 piezas fundamentales, Inquilinos y Arrendador, por ello decidimos testearlo en nuestra casa ubicada en Londres, Reino Unido, donde durante un par de semanas se instaló la aplicación a todos los inquilinos (7 personas) con un smartphone y sistema operativo Android, aunque no pudimos testear el papel del arrendador, la prueba fue bastante útil.
Ambos les insistimos en que nos notificaran cualquier error, o nos sugieran cualquier cambio que ellos/as pudieran localizar en Howse, a continuación se mostrarán los resultados:

Los primeros días los errores eran comunes, por ejemplo, falta de botones que facilitaran la vuelta atrás en alguna pestaña, algunas imágenes no se cargaban correctamente, o los tiempos de espera eran demasiado grandes; cosas que fuimos solucionando según nos notificaban.
Durante el desarrollo de la aplicación nos dimos cuenta de que era mucho más útil utilizar el método "getUid ()" (método de Firebase) que el "getKey" (método de Firebase), esto trajo consigo distintos problemas ya que lo utilizabamos con mucha frecuencia. (Listado 6)
Tuvimos que actualizar los pojos y por tanto Firebase, añadir un dato y cambiar el anterior en todos los lugares. (Listado 7)



Listado 6: Key Antigua

```
ut.addOnSuccessListener( activity: Register.this, new OnSuccessListener<UploadTask.TaskSnapshot>() {
    @Override
    public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
        Task<Uri> task = taskSnapshot.getStorage().getDownloadUrl();
        task.addOnSuccessListener(new OnSuccessListener<Uri>() {
            @Override
            public void onSuccess(Uri uri) {
                Usuario usuario = new Usuario(clave,
                        etNombre.getText().toString(),
                        etApellido.getText().toString(),
                        etEmail.getText().toString().toLowerCase(),
                        uri.toString(),
                        codCasa,
                        tipoUs,
                        user.getUid());
                mDatabaseRef.child(user.getUid()).setValue(usuario);

                Intent i = new Intent( packageContext: Register.this, Login.class);
                i.putExtra("tipo", tipoUs);
                i.putExtra("codCasa", codCasa);
                startActivity(i);

            }
        });
```

Listado 7: Nueva Key llamada UID

El problema llega cuando olvidamos cambiarlo en algunos lugares de bastante importancia.
Varios de nuestros compañeros de piso se dieron cuenta que al actualizar los datos de su perfil, todo iba correcto, pero cuando lo consultaba de nuevo no se actualizaban y si volvía a actualizar dichos datos, el problema persistia. Cuando nos metimos de lleno con ese problema se descubrió que el usuario que se había cargado tenía una clave que era la Key actualizada con el método referido en el Listado X (key antigua) y en la modificación se estaba cargando el usuario actualizado con el UID referido en el Listado 9 (Nueva Key), por lo que en vez de actualizarse estaba generando otro nuevo usuario dentro de ese mismo usuario, a si que nos pusimos a repasar a fondo el código para resolver eso cuando antes.

Tras una semana de corrección de errores, mientras implementábamos la última parte del arrendador, les actualizamos la aplicación y se la volvimos a instalar para un segundo testeo.

A lo largo de la segunda semana los integrantes del piso nos notificaron que había ciertos fallos que no afectaban al funcionamiento directo de la aplicación;

- En la lista de la compra si no se especificaba el precio de un ítem no aparecía el nombre de usuario.
- En el perfil, cuando se actualiza la imagen y los datos al mismo tiempo y se procedía a guardar los cambios, se duplicaban los datos en firebase, porque el DatabaseReference de Firebase apuntaba a un sitio erróneo, provocando un duplicado de los datos.

3. CONCLUSIONES

Durante los dos meses y medio en los que hemos desarrollado nuestro trabajo de fin de ciclo hemos tenido que estudiar el ecosistema de los pisos compartidos, en un primer instante nos dimos cuenta de que era necesario una mejora en la calidad de vida de aquellas personas que viven en viviendas compartidas.

El desarrollo de esta aplicación nos ha llevado a entender de una manera mucho más amplia como funciona el alquiler de dichas viviendas.

La primera conclusión que sacamos, fue que implementar la tecnología de los smartphones en ese ecosistema iba a hacer que el nivel de vida de estas personas incrementase sustancialmente.

La falta de comunicación y de organización hacía mucho más difícil la convivencia ya que la comunicación no era efectiva y la organización era nula.

La situación en un piso compartido es bastante compleja ya que cada inquilino tiene un horario y una rutina distinta a los demás.

Como no existe una relación afectiva entre ellos, la empatía es casi inexistente.

En el momento en el que hicimos los primeros tests nos dimos cuenta que las soluciones que habíamos pensado para estos problemas estaban siendo efectivas y estaba empezando surtir efecto en lo que nosotros mismos hemos denominado "Efecto Familia".

Hubo una notable mejora en la comunicación entre los inquilinos de nuestra propia casa, las tareas del hogar se realizaban con una mayor frecuencia y no había tantos roces entre los mismos

La curva de aprendizaje de la aplicación es leve ya que, por el feedback obtenido en los tests hechos en nuestra estancia en Londres no hubo dificultades en el uso de ésta, salvo algunos matices.

En lo que se refiere al proyecto en sí, la principal conclusión que hemos extraído es que implementar la tecnología en ámbitos donde nunca se había innovado de esta manera, requiere mucho más trabajo que el que nosotros hayamos podido realizar en estos dos meses y medio.

Una conclusión más personal es referida a la dificultad de hacer un proyecto en tan corto espacio de tiempo. Por este motivo hemos tenido que agilizar nuestro aprendizaje, tanto en adquirir nuevos conocimientos como en la solución de errores

3.1. Trabajo futuro

Howse es una aplicación que tiene mucho camino por recorrer, queríamos implementar infinidad de aspectos para continuar complementando y facilitando la vida tanto de los inquilinos como la del arrendador, pero debido al escaso tiempo tuvimos que intentar adaptarlo lo mejor posible a los verdaderos problemas dentro de un alquiler de viviendas compartidas.
A continuación se verán algunas de las implementaciones que nos gustaría haber hecho para mejorar ésta aplicación:

- Mejora del chat: Creación de chats grupales, donde puedan comunicar al arrendador opiniones conjuntas.
- Pestaña de facturación: Donde, cada Inquilino pueda ver de una forma sencilla todos los pagos por hacer, ya sean por semanas, por meses o por trimestres, incluso incluir una forma de pago dentro de la aplicación.
- Añadir una forma de pago conjunto en la lista de la compra para productos de primera necesidad.
- Implementar una opción en el registro para especificar el día de salida del piso, que él mismo lo pueda cambiar en el perfil bajo supervisión del arrendador.

4. BIBLIOGRAFÍA Y WEBGRAFÍA

Para completar nuestro trabajo, tanto en Howse, como en ésta memoria hemos utilizado los siguiente:

- **STACK OVERFLOW (2019).**

  https://www.stackoverflow.com/. Fecha de la consulta:

- Abril 3, 2019:

  https://stackoverflow.com/questions/5427195/how-to-change-the-minsdkversion-of-a-project

- Abril 16, 2019:

  https://stackoverflow.com/questions/44546849/unsupported-method-baseconfig-getapplicationidsuffix

- Abril 17, 2019:

  https://stackoverflow.com/questions/17054000/cannot-resolve-symbol-r-in-android-studio

- Abril 23, 2019:

  https://stackoverflow.com/questions/38671444/user-does-not-have-permission-to-access-this-object-firebase-storage-android

- Abril 26, 2019:

  https://stackoverflow.com/questions/50467814/tasksnapshot-getdownloadurl-is-deprecated/50468622#50468622

- Mayo 9, 2019:

  https://stackoverflow.com/questions/51659999/retrieve-corresponding-data-from-firebase-when-clicking-on-spinner-item-in-andro

- Mayo 10, 2019:

  https://stackoverflow.com/questions/16196444/text-layout-alignment-in-android-textalignment-gravity

- Mayo 17, 2019:

  https://stackoverflow.com/questions/3750903/how-can-getcontentresolver-be-called-in-android

- Mayo 21, 2019:

  https://stackoverflow.com/questions/7032070/what-is-the-difference-between-arraylist-clear-and-arraylist-removeall

- Mayo 22, 2019:

  https://stackoverflow.com/questions/34369469/nullpointerexception-attempt-to-invoke-interface-method-android-view-view-and

- **WIKIPEDIA (2019).**

  https://www.wikipedia.org/. Fecha de la consulta:

- Abril 15, 2019:

  https://es.wikipedia.org/wiki/Firebase

- Junio 3, 2019:

  https://es.wikipedia.org/wiki/Kotlin_(lenguaje_de_programaci%C3%B3n)

- Junio 4, 2019:

  https://es.wikipedia.org/wiki/Android_Studio

- Junio 5, 2019:

  https://es.wikipedia.org/wiki/Gradle

- Junio 6, 2019:

  https://es.wikipedia.org/wiki/GitHub

- **YOUTUBE (2019).**

  https://www.youtube.com/. Fecha de la consulta:

- Abril 6, 2019:
  https://www.youtube.com/watch?v=zVm4WKB4t2M
- Abril 10, 2019:
  https://www.youtube.com/watch?v=d13XebfE72o
- Abril 11, 2019:
  https://www.youtube.com/watch?v=KpWWXlvre78
- Abril 14, 2019:
  https://www.youtube.com/watch?v=bhhs4bwYyhc
- Abril 19, 2019:
  https://www.youtube.com/watch?v=lEacPYUB9cc&t=32s
- Abril 24, 2019:
  https://www.youtube.com/watch?v=3yMAK-zk--Q
- Abril 25, 2019:
  https://www.youtube.com/watch?v=rYfNRh0HjeI&t=3s
- Abril 26, 2019:
  https://www.youtube.com/watch?v=h8UNk6r-akk
- Abril 30, 2019:
  https://www.youtube.com/watch?v=DIAGb5V9Myo
- Mayo 3, 2019:
  https://www.youtube.com/watch?v=x-HljJg7IAg
- Mayo 9, 2019:
  https://www.youtube.com/watch?v=Ts5Iv_NfTkA

- Mayo 10, 2019:

  https://www.youtube.com/watch?v=eX-TdY6bLdg

- Mayo 12, 2019:

  https://www.youtube.com/watch?v=eX-TdY6bLdg&t=1s

- Mayo 14, 2019:

  https://www.youtube.com/watch?v=l_L21cKVuQ8

- Mayo 16, 2019:

  https://www.youtube.com/watch?v=KnUgUKJ9dmc

- Mayo 17, 2019:

  https://www.youtube.com/watch?v=ZHmVNst_Rnc&list=PLzLFqCABnRQftQQETzoVMuteXzNiXmnj8&index=11

- Mayo 19, 2019:

  https://www.youtube.com/watch?v=ZBEtt2rNS6M

- Mayo 21, 2019:

  https://www.youtube.com/watch?v=22UwqLznhYU

- Mayo 24, 2019:

  https://www.youtube.com/watch?v=gkh-5pmYEa0

- Mayo 25, 2019:

  https://www.youtube.com/watch?v=0aOn2mIRlCA

- Mayo 27, 2019:

  https://www.youtube.com/watch?v=jpaHMcQDaDg&t=409s

- Mayo 28, 2019:

  https://www.youtube.com/watch?v=DFnxY_PEnYY&t=3s

- Mayo 30, 2019:

  https://www.youtube.com/watch?v=wcE7IIHKfRg&t=2s

5. ANEXOS

En este apartado vamos a mostrar partes destacables de nuestro código:

- Código para copiar en el portapapeles:

```
btnClipBoard.setOnClickListener( new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String text = codigo.getText().toString();
        ClipboardManager clipboard = (ClipboardManager) getSystemService( Context.CLIPBOARD_SERVICE);
        ClipData clip = ClipData.newPlainText( label: "text",  text);
        clipboard.setPrimaryClip(clip);
        Toast.makeText( context: PerfilArrendador.this, text: "Copiado en el Portapapeles", Toast.LENGTH_LONG ).show();


    }
} );

```

Listado 8: "ClipBoardManager ()"

Código necesario para modificar una foto en Firebase:

```
private void uploadImage(){
    if(imageUri !=null){
        final StorageReference fileReference = storageReference.child(System.currentTimeMillis()
                +"."+getFileExtension( imageUri ));

        uploadTask = fileReference.putFile( imageUri );
        uploadTask.continueWithTask( new Continuation<UploadTask.TaskSnapshot, Task<Uri>>() {
            @Override
            public Task<Uri> then(@NonNull Task<UploadTask.TaskSnapshot> task) throws Exception {
                if (!task.isSuccessful()){
                    throw task.getException();
                }

                return fileReference.getDownloadUrl();

            }
        } ).addOnCompleteListener( new OnCompleteListener<Uri>() {
            @Override
            public void onComplete(@NonNull Task<Uri> task) {
                if (task.isSuccessful()){
                    Uri downloadUri = task.getResult();
                    String mUri = downloadUri.toString();
                    mDatabaseRefFotoUsuario = FirebaseDatabase.getInstance().getReference( s: "Usuarios" ).child( usuario.getUid() );
                    HashMap<String, Object> map = new HashMap<>();
                    map.put("fotoUsuario", mUri);
                    mDatabaseRefFotoUsuario.updateChildren( map );
                }else{
                    Toast.makeText( context: Perfil.this, text: "Failed!", Toast.LENGTH_SHORT ).show();
                }
            }
        } ).addOnFailureListener( new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Toast.makeText( context: Perfil.this, e.getMessage(), Toast.LENGTH_SHORT ).show();
            }
        } );


    }else {
        Toast.makeText( context: this, text: "Selecciona una imagen ", Toast.LENGTH_SHORT ).show();
```

Listado 9: "uploadImage ()"

Al realizar la visualización de las tareas recogidas en los días de la semana, hemos tenido que crear diferentes listas para poder anidar un recyclerview dentro de otro:

```
for(Tarea atarea: listaTareas){

        if(atarea.getDiaSemana().equals("Lunes")){

            tareasLunes.add(new TareasVisual(atarea.getKeyTarea(),atarea.getPersona().getFotoUsuario(),atarea.getPersona().getNombreUsuario(),atarea.getTipoTarea()));

        }else if (atarea.getDiaSemana().equals("Martes")){
            tareasMartes.add(new TareasVisual(atarea.getKeyTarea(),atarea.getPersona().getFotoUsuario(),atarea.getPersona().getNombreUsuario(),atarea.getTipoTarea()));


        }else if (atarea.getDiaSemana().equals("Miercoles")){
            tareasMiercoles.add(new TareasVisual(atarea.getKeyTarea(),atarea.getPersona().getFotoUsuario(),atarea.getPersona().getNombreUsuario(),atarea.getTipoTarea()));

        }else if (atarea.getDiaSemana().equals("Jueves")){

            tareasJueves.add(new TareasVisual(atarea.getKeyTarea(),atarea.getPersona().getFotoUsuario(),atarea.getPersona().getNombreUsuario(),atarea.getTipoTarea()));

        }else if (atarea.getDiaSemana().equals("Viernes")){
            tareasViernes.add(new TareasVisual(atarea.getKeyTarea(),atarea.getPersona().getFotoUsuario(),atarea.getPersona().getNombreUsuario(),atarea.getTipoTarea()));


        }else if (atarea.getDiaSemana().equals("Sabado")){
            tareasSabado.add(new TareasVisual(atarea.getKeyTarea(),atarea.getPersona().getFotoUsuario(),atarea.getPersona().getNombreUsuario(),atarea.getTipoTarea()));


        }else if (atarea.getDiaSemana().equals("Domingo")){

            tareasDomingo.add(new TareasVisual(atarea.getKeyTarea(),atarea.getPersona().getFotoUsuario(),atarea.getPersona().getNombreUsuario(),atarea.getTipoTarea()));

        }
```

```
if(tareasLunes.size()!=0){

    DiasTareas diasTareasL = new DiasTareas( dia: "Lunes", tareasLunes);
    tareasDias.add(diasTareasL);

}
if(tareasMartes.size()!=0){

    DiasTareas diasTareasM = new DiasTareas( dia: "Martes", tareasMartes);
    tareasDias.add(diasTareasM);

}
if(tareasMiercoles.size()!=0){

    DiasTareas diasTareasX = new DiasTareas( dia: "Miercoles", tareasMiercoles);
    tareasDias.add(diasTareasX);


}
if(tareasJueves.size()!=0){
    DiasTareas diasTareasJ = new DiasTareas( dia: "Jueves", tareasJueves);
    tareasDias.add(diasTareasJ);


}
if(tareasViernes.size()!=0){
    DiasTareas diasTareasJ = new DiasTareas( dia: "Viernes", tareasViernes);
    tareasDias.add(diasTareasJ);


}
if(tareasSabado.size()!=0){
    DiasTareas diasTareasS = new DiasTareas( dia: "Sabado", tareasSabado);
    tareasDias.add(diasTareasS);


}
if(tareasDomingo.size()!=0){
    DiasTareas diasTareasD = new DiasTareas( dia: "Domingo", tareasDomingo);
    tareasDias.add(diasTareasD);


}


adapter = new AdaptadorDiasSemana(getApplicationContext(), tareasDias);
rvDiasdelaSemana.setAdapter(adapter);
```

Listados 10 y 11: Cargando Listas