Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación para la gestión de compras con dispositivos móviles

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Rodrigo Ballesteros Cabañas

Tutor: Juan Vicente Capella

Curso 2015/2016

Resumen

Este proyecto consiste en la creación de una aplicación móvil desarrollada para sistemas operativos Android cuya principal funcionalidad es que cada usuario puede crear sus listas de la compra y, una vez registrado, compartirlas con sus amigos que previamente son agregados al aceptar su solicitud de amistad. Cada lista se puede guardar en una de las cinco categorías posibles, junto con su nombre y el precio total. En ellas encontramos los productos con sus precios añadidos por el usuario.

El proceso de desarrollo de la aplicación se ha llevado a cabo en tres fases. Un primer proceso de análisis para especificar los requisitos de la aplicación mediante los casos de uso, un segundo proceso de diseño centrado en el usuario, atendiendo sobretodo al contexto de uso, y posteriormente, se ha realizado la implementación y desarrollo de la aplicación utilizando tecnologías como Java y XML, además de los servicios de Google Play Services necesarios para la autenticación de los usuarios.

Por último, la aplicación utiliza una base de datos NoSQL alojada en la nube llamada Firebase Realtime Database, donde se almacenan los usuarios, sus listas de la compra, sus solicitudes de amistad y sus amigos. Los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada usuario conectado.

Palabras clave: Aplicación móvil, listas de la compra, categorías, diseño centrado en el usuario, autenticación, nube.

Abstract

This project involves the creation of a mobile app developed for Android operating systems whose main feature is that each user can create shopping lists and, once registered, share them with friends which they are previously added to accept your friend request. Each list can be saved in one of five possible categories, along with its name and the total price. Among them, we can find the products with their prices added by the user.

The process of app development has been carried out in three phases. A first analysis process for specific application requirements through use cases, a second process design user-centered, taking especially the context of use, and subsequently carried out the implementation and app development using technologies such as Java and XML, as well as Google Play Services required for the users authentication.

Finally, the app uses a NoSQL database, located in the cloud called Firebase Realtime Database, where users, their shopping lists, their friend requests and their friends are stored. Data is stored in JSON format and it's synchronized in real time with each connected user.

Keywords: Mobile App, shopping lists, categories, user-centered, authentication, cloud.

Tabla de contenidos

1.Introducción 9

1.1 Motivación 9

1.2 Objetivos 9

1.3 Estructura 10

2. Estado del arte 11

2.1 Listas de compras 11

2.2 Lista Compra - ListOn Free 12

3. Contexto tecnológico 13

3.1 Entornos de desarrollo 13

3.1.1 OS X El Capitan 13

3.1.2 Android Studio 13

3.1.2.1 Gradle 14

3.2 Java 15

3.3 Android 15

3.3.1 Dalvik 16

3.3.2 XML 16

3.3.3 SQLite 16

3.3.4 Google Play Services 17

3.3.5 Técnicas de debugging 17

3.3.6 Control de versiones 18

3.4 Firebase - Google 19

3.4.1 Autenticación 20

3.4.2 Base de datos en tiempo real 20

3.4.2.1 Estructuración de los datos 20

4. Especificación de requisitos 21

4.1 Introducción 21

4.1.1 Propósito 21

4.1.2 Ámbito 21
4.2 Descripción general 22
4.2.1 Perspectiva del producto 22
4.2.2 Funciones del producto 22
4.2.3 Características de usuario 23
4.2.4 Restricciones generales 23
4.3 Requisitos específicos 24
4.3.1 Requisitos funcionales 24
4.3.2 Requisitos de interfaz 29
4.3.3 Restricciones de diseño 29
4.3.4 Atributos 30
4.3.5 Otros requisitos 30
5. Análisis 31
5.1 Casos de uso 31
5.1.1 Tipos de relaciones 31
5.1.2 Ventajas 31
6. Diseño 33
6.1 API de Android 33
6.1.1 AndroidManifest 33
6.1.2 Views y Layouts 34
6.1.3 Activities 34
6.1.4 Fragments 35
6.1.5 Fragment Pager Adapter 35
6.1.6 Intents 36
6.1.7 Adapters 36
6.1.8 Action Bar y Toolbar 37
6.1.9 Navigation Drawer 37
6.1.10 View Pager 38
6.1.11 Mensajes Toast 38
6.2 Interfaz gráfica de usuario 39

6.2.1 Prototipos....39
6.2.1.1 Pantalla de bienvenida....39
6.2.1.2 Pantalla de login y registro....40
6.2.1.3 Pantalla de nueva lista....40
6.2.1.4 Pantalla de los tablones....41
6.2.1.5 Pantalla de compartir la lista....41
6.2.1.6 Pantalla de seleccionar amigos o peticiones....42
6.2.1.7 Pantalla de los amigos....42
6.2.1.8 Pantalla de las solicitudes....43
6.2.1.9 Pantalla sobre nosotros....43
6.2.1.10 Diálogo para el nombre de la lista....44
6.2.1.11 Diálogo para el precio del producto....44
6.2.1.12 Diálogo para compartir la lista....45
7. Implementación....47
7.1 Almacenamiento externo de ficheros....47
7.2 Bases de datos de los usuarios en SQLite....49
7.3 Clase Product....51
7.4 Clase Share....52
7.5 Conexión a Firebase....53
7.5.1 Autenticación....53
7.5.2 Compartir lista....55
7.5.3 Aceptar solicitud de amistad....56
8. Evaluación....59
9. Conclusiones....63
10. Bibliografía....65
Anexo 1. Interfaces gráficas de usuario en formato XML....67

1.Introducción

Una aplicación para la gestión de compras es una herramienta que permite el control de las listas personales de la compra en varias categorías, así como la gestión del precio de cada producto y del total de cada lista. La aplicación está destinada a dispositivos móviles smartphones que utilicen el sistema operativo Android.

Las acciones que pueden realizarse mediante el uso de esta aplicación consisten en crear y administrar las listas personales, así como modificar los precios de cada producto. También es posible compartir las listas, pero es necesario estar registrado y tener amigos agregados.

Por otro lado, existe un apartado web en Firebase destinado a la gestión de la base de datos mediante el cual se pueden realizar diversos cambios, con el fin de poder completar la funcionalidad que ofrece la aplicación Android. De esta forma, se pueden administrar los usuarios, introducir datos y modificarlos.

1.1 Motivación

Uno de los motivos por los que se decidió desarrollar el proyecto en la plataforma Android está relacionado con el auge actual de la tecnología utilizada para el desarrollo de la aplicación, el éxito de los smartphones y las posibilidades que ofrecen. De esta forma, se han podido adquirir conocimientos del desarrollo destinado a estas plataformas en oposición a otras tecnologías como los ordenadores de sobremesa.

Otro de los motivos consiste en el sistema operativo al que va dirigida la aplicación, en este caso Android, un sistema operativo que ya lleva unos años en el mercado y que cada vez está más extendido, tanto en smartphones como en tablets. Gran parte de su popularidad es debido a que es de código abierto lo que permite un sin fin de posibilidades nuevas. Debido a esto, hay que agradecer el gran apoyo que recibimos todos los desarrolladores de Google, con nuevos métodos para conectar toda la aplicación con todo el mundo.

Por último, la posibilidad de desarrollar una aplicación para dispositivos móviles en un ámbito que no se estudia ampliamente a lo largo de la ingeniería, por lo que era una buena oportunidad para obtener conocimiento en dicho ámbito y poder desarrollar una aplicación para el mercado actual.

1.2 Objetivos

El objetivo principal de este Trabajo de Final de Grado es la creación de una aplicación móvil para la gestión de compras. La aplicación tiene que estar dirigida a cualquier persona del planeta que desee administrar sus listas de la compra con un diseño novedoso y sencillo.

Desde el punto de vista académico tiene los siguientes objetivos:

- Diseñar, implementar y evaluar una aplicación móvil de gestión de compras desde cero.
- Adquirir experiencia en la ejecución de aplicaciones en entornos de depuración.
- Adquirir experiencia en la configuración de sistemas de gestión de bases de datos como SQLite.

• Adquirir nociones para diseñar la aplicación centrándose en el usuario, obteniendo como resultado un diseño sencillo e intuitivo.

• Adquirir nociones sobre la plataforma Firebase, para llevar un control sobre los usuarios, sobre los amigos y sobre las listas compartidas.

1.3 Estructura

Este apartado está diseñado para describir con claridad las fases que ha atravesado la aplicación durante su ciclo de vida. Está estructurado en las siguientes partes:

• **Introducción:** En este apartado se introduce el proyecto, así como la motivación, objetivos y estructura.

• **Estado del arte:** En este punto se hará un análisis de dos aplicaciones similares en Google Play Store, resumiendo las características comunes.

• **Contexto tecnológico:** Este capítulo describirá los entornos de desarrollo utilizados para realizar la aplicación y las tecnologías empleadas como Java, Android y Firebase, una potente plataforma de desarrollo móvil en la nube de Google.

• **Especificación de Requisitos:** Durante este apartado se hará una descripción completa del comportamiento del sistema que se va a desarrollar. Por una parte, se hablará de la funcionalidad que el cliente desea tener en su aplicación, y por otra, la funcionalidad que va a implementarse.

• **Análisis:** Este punto describirá las fases de análisis necesarias para obtener el modelo conceptual, concretamente se utilizaran los casos de uso.

• **Diseño:** Capítulo que describirá cuáles son los diseños a los que tendrá que ajustarse la aplicación, así como las metodologías que se han utilizado.

• **Implementación:** En este apartado se mencionarán las herramientas utilizadas durante la fase de implementación, y se tratarán temas como el almacenamiento externo, la gestión de bases de datos en SQLite, las clases necesarias para manejar correctamente los datos y el uso y manejo de firebase para autentificar, añadir amigos y compartir listas personales en la nube.

• **Evaluación:** Punto en el que se describirán las técnicas de evaluación de la aplicación y en el que se resumirán los resultados obtenidos en varios móviles con pantallas de diferente tamaño.

• **Conclusiones:** En este capítulo se hará una reflexión sobre el trabajo realizado, analizando si se han cumplido los objetivos propuestos. Además, de una valoración personal sobre los conocimientos adquiridos y sobre el desarrollo del proyecto.

• **Bibliografía:** Este apartado indicará las fuentes consultadas para comprender los conceptos usados durante el desarrollo del proyecto.

• **Anexo:** Punto en el que se adjuntan documentos independientes que aportaran información relevante para entender mejor el diseño y funcionamiento de la aplicación.

2. Estado del arte

Actualmente en el mercado existen varias aplicaciones con una funcionalidad parecida a la de nuestra aplicación. Algunas de ellas comparten varios objetivos perseguidos en este proyecto como la posibilidad de guardar tus listas personales y compartirlas con tus amigos. Debido al tiempo que llevan las aplicaciones en Google Play Store, muchas funcionalidades han quedado desfasadas y hoy en día, se pueden incluir de forma más fácil y con un coste menor.

De todas las aplicaciones que existen ahora mismo en el mercado se han elegido las dos con más similitud a nuestros objetivos. Por lo tanto, vamos a realizar un análisis con las características más importantes y sobretodo, con las funcionalidades que se van a mejorar.

2.1 Listas de compras

Esta aplicación de origen ruso se puede descargar gratuitamente para sistemas operativos Android y cuenta con más de 1.000.000 de usuarios desde 2013. Su objetivo principal es crear listas rápidamente y compartirlas con cualquier número de teléfono móvil mediante sms tradicional con su correspondiente pago.

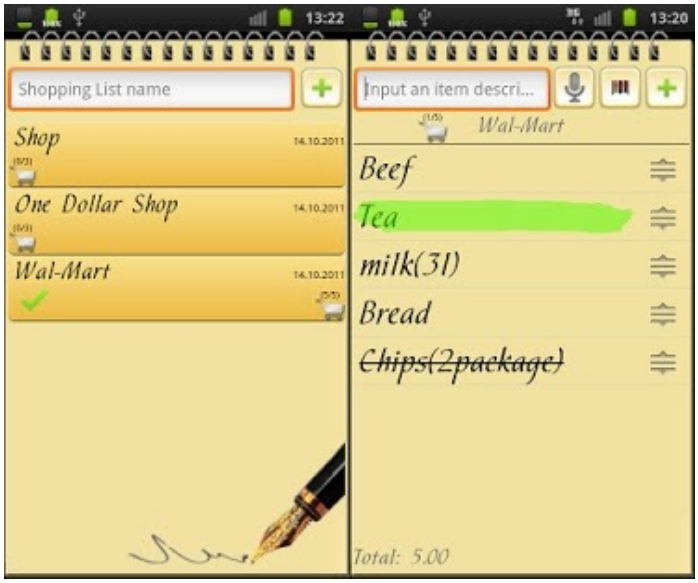

Figura nº 1. Interfaz de la aplicación Listas de compras

Entre las características más interesantes destacan:

- La similitud en el diseño con una lista tradicional ya que simula muy bien la tarea de rellenarla e incluso la forma en tachar los productos, como se observa en la figura 1.

- La opción de compartir las listas con cualquiera introduciendo el número de teléfono de la persona a la que deseas enviárselo por sms.

- La facilidad a la hora de crear las listas y de introducir los productos, ya que se pueden organizar fácilmente mediante el uso de arrastrar y soltar.

- La falta de un sistema para autentificar a los usuarios de la aplicación ya sea por correo anónimo o mediante Google o Facebook.

Como se trata de una aplicación que lleva en el mercado 3 años, muchas de sus funcionalidades ya están atrasadas en el mercado actual. Una de ellas sería la opción de introducir el producto mediante voz, actualmente está disponible desde el teclado de Google. La segunda sería una de las opciones más importantes e interesantes en la aplicación, como es compartir tus listas personales con todo el mundo. Para poder utilizarlo es necesario realizarlo mediante sms a un número de teléfono, por lo tanto hay que pagar el coste del mensaje lo que al usuario no le beneficia en ningún caso. Esta funcionalidad va a ser mejorada ya que actualmente, casi nadie utiliza la opción de enviar por sms tradicional, se puede enviar de cualquier otra forma o compartir en la misma aplicación como es nuestro caso.

2.2 Lista Compra - ListOn Free

En este caso la aplicación es de origen español y también se puede descargar gratuitamente para sistemas operativos Android contando con más de 500.000 usuarios desde 2014. Esta aplicación cuenta con un diseño gráfico muy trabajado en comparación con la aplicación anterior, lo que es muy importante a la hora de realizar una aplicación de este tipo. Su objetivo principal es crear listas, pero en este caso, se da la opción al introducir cada producto de elegir una de las muchas categorías disponibles. Más adelante, es añadida a tu lista y también a la categoría elegida para llevar la cuenta de todos los productos que has ido añadiendo cuando estabas haciéndola. También añade la opción de compartirlas por muchas plataformas como puede ser Gmail o WhatsApp, pero no desde la propia aplicación.

Figura nº 2. Interfaz de la aplicación ListOn Free

Las características más importantes serían:

- Su diseño novedoso e intuitivo, incluyendo la inmensa cantidad de categorías disponibles en las que puedes clasificar casi todos los productos (véase figura 2).

- La falta de un sistema de autenticación para poder compartir las listas entre los usuarios correctamente.

En este caso, la aplicación nos permite compartir las listas mediante otras plataformas pero sigue sin poder compartirse desde ella. Esta opción va a ser mejorada para tener un mayor control y una mejor organización utilizando la plataforma Firebase.

3. Contexto tecnológico

En este punto se van a exponer los distintos entornos de programación, lenguajes, y tecnologías que se han utilizado para desarrollar la aplicación, sin entrar en detalles de implementación. También se van a describir las técnicas de debugging y el control de versiones que se ha llevado a lo largo del ciclo de vida de la aplicación. Para acabar se introducirá la plataforma Firebase y se explicarán las diferentes opciones que se pueden habilitar al diseñar nuestra aplicación utilizando esta plataforma.

3.1 Entornos de desarrollo

En este primer apartado se van a mencionar cuáles han sido los entornos que se han utilizado para desarrollar la aplicación, tanto a nivel de sistema operativo como de entorno de programación.

3.1.1 OS X El Capitan

Es un entorno operativo basado en Unix, un sistema operativo portable, multitarea y multiusuario. Ideal para el desarrollo de aplicaciones móviles ya que, gracias a su rapidez, robustez y seguridad, se consigue ser más productivo que si se estuviese realizando en otro sistema operativo como puede ser Windows.

3.1.2 Android Studio

Es una plataforma Software que abstrae el Hardware y facilita el desarrollo de apps para dispositivos con recursos limitados. Sus características más importantes serían:

- Su licencia Apache 2.0, es decir, permite al usuario del software la libertad de usarlo para cualquier propósito, distribuirlo, modificarlo, y distribuir versiones modificadas de ese software.

- Su máquina virtual propia, Android Virtual Machine (AVD), donde podemos emular las aplicaciones con todos los tamaños de pantallas disponibles en el mercado.

- La gran cantidad de APIs y herramientas de desarrollo, compilación, depuración y emulación.

- Su renderizado de vistas en tiempo real, esto permite que podamos visualizar cada pantalla antes de implementarla en el código.

- Un soporte para la construcción de la aplicación basado en Gradle, hablaremos de él a continuación.

3.1.2.1 Gradle

Es una herramienta de construcción multiplataforma, basada en Apache Ant y Apache Maven. Introduce un lenguaje basado en Groovy para declarar las dependencias del proyecto en vez del tradicional XML. Podemos manejar de manera fácil las dependencias, tener varios entornos para un mismo proyecto o poder generar diferentes APKs de un mismo proyecto con diferentes configuraciones de UI, entre otras muchas cosas. A continuación, se pueden observar las dependencias del proyecto desarrollado tanto a nivel de proyecto, como a nivel de módulo (véase figura 3 y 4).

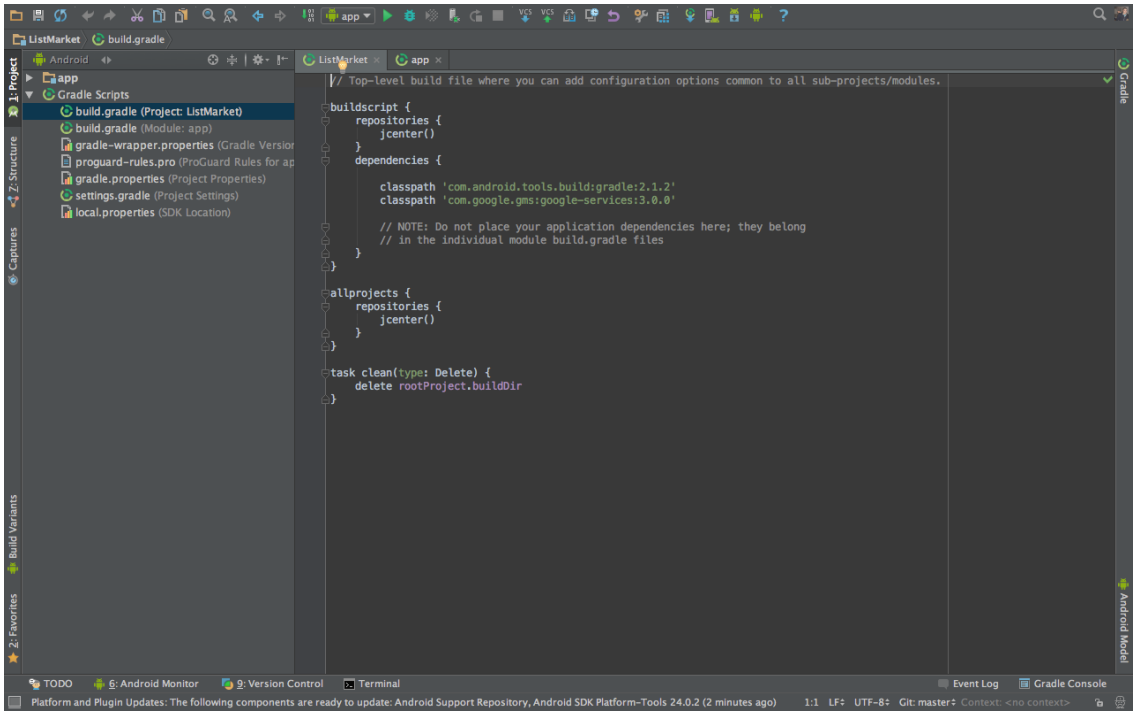

Figura nº 3. Fichero build.gradle (Project)

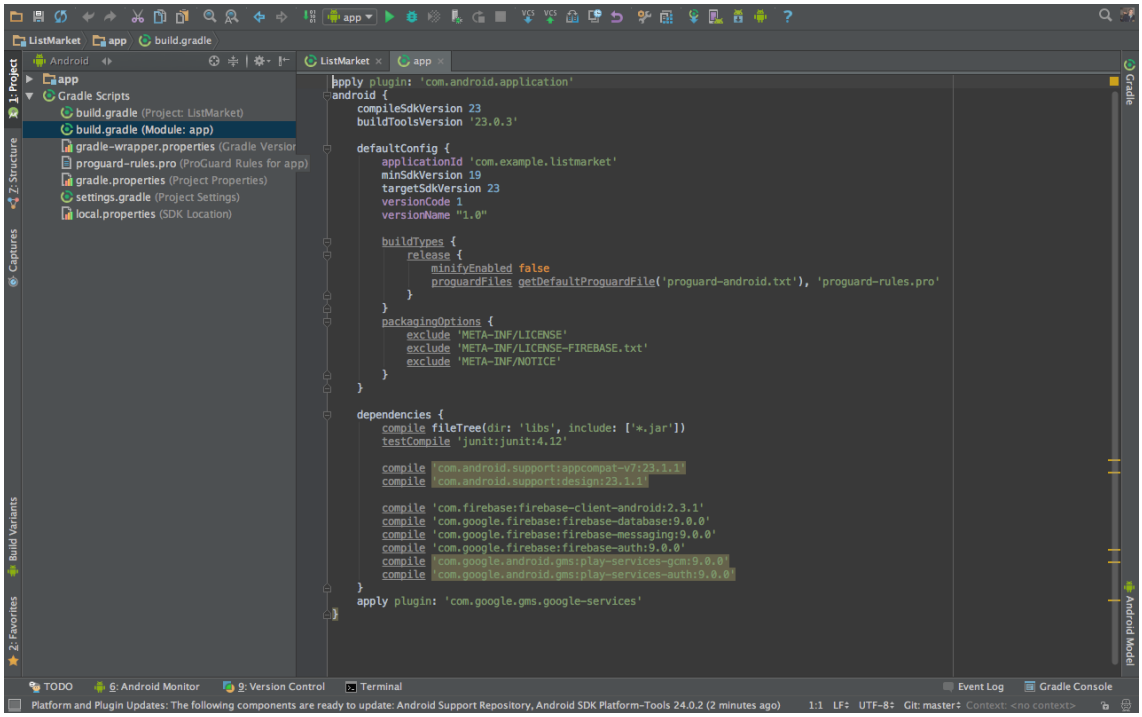

Figura nº 4. Fichero build.gradle (Module)

3.2 Java

Es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para funcionar en otra. Desde 2012, es uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web.

3.3 Android

Android es un sistema operativo inicialmente pensado para teléfonos móviles, al igual que iOS, Symbian y Blackberry OS. Lo que lo hace diferente es que está basado en Linux, un núcleo de sistema operativo libre, gratuito y multiplataforma.

El sistema permite programar aplicaciones en una variación de Java llamada Dalvik que veremos a continuación. El sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla en un lenguaje de programación muy conocido como es Java.

Esta sencillez, junto a la existencia de herramientas de programación gratuitas, hacen que una de las cosas más importantes de este sistema operativo sea la cantidad de aplicaciones disponibles, que extienden casi sin límites la experiencia del usuario. En la figura 5 se observan los componentes por los que está formado el sistema de Android.

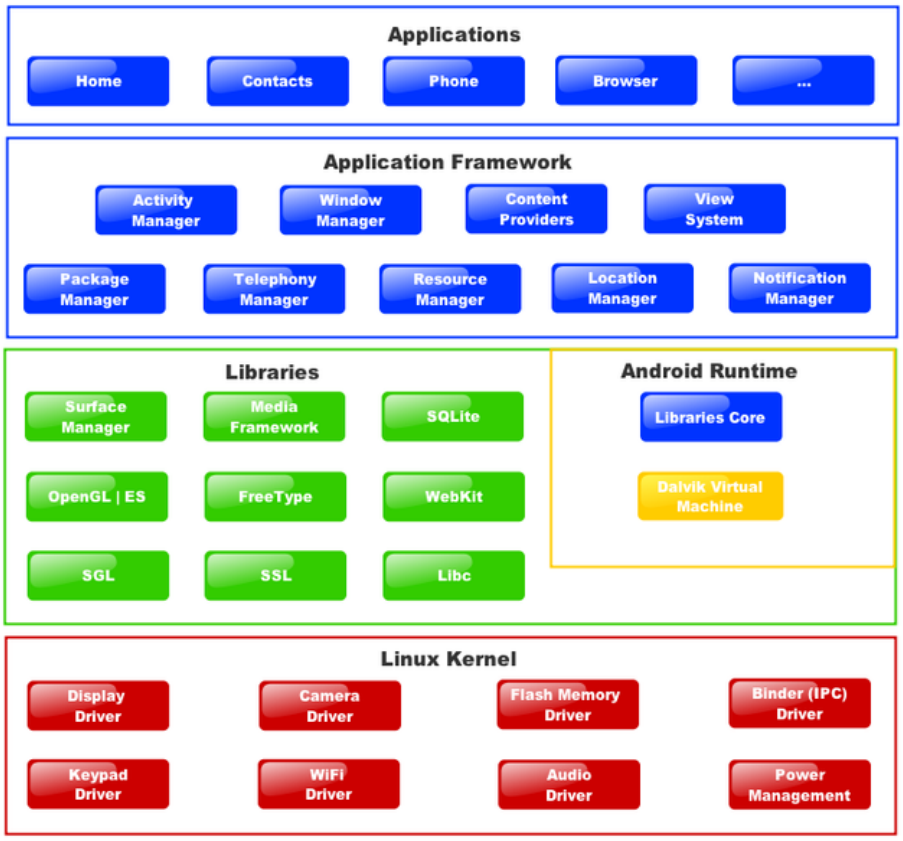

Figura nº 5. Ilustración de los componentes de un sistema Android

3.3.1 Dalvik

Se trata de la máquina virtual usada en los dispositivos Android, la cual tiene una serie de características que la diferencian de las máquinas virtuales de java. Todas las aplicaciones Android están hechas en el lenguaje de programación java, pero usan un bytecode diferente al resto de aplicaciones creadas con las máquinas virtuales anteriores.

Las características más importantes serían:

- El bytecode generado, primero es transformado al bytecode de Java y posteriormente se transforma en el usado por Android. Es decir, primero son generados los .class típicos de java, y posteriormente se transforman al tipo de archivo .dex (dalvik executable). Estos archivos dex son comprimidos en los conocidos APK (Android Package).

- Esta diseñada para ejecutar varias instancias de la propia máquina simultáneamente y optimizada para necesitar poca memoria.

- Se basa en registros en lugar de pilas, aprovechando así mejor el rendimiento de los móviles con estos.

- También esta distribuida como software libre usando la licencia Apache.

3.3.2 XML

XML proviene de Extensible Markup Language (Lenguaje de Marcas Extensible). Se trata de un metalenguaje (un lenguaje que se utiliza para decir algo acerca de otro) extensible de etiquetas que fue desarrollado por el Word Wide Web Consortium (W3C), una sociedad mercantil internacional que elabora recomendaciones para la World Wide Web.

Dado que en gran parte, la utilidad de una herramienta depende de la creatividad de quien la utiliza, resulta imposible resumir todas las aplicaciones de XML. En pocas palabras, se puede decir que ofrece la posibilidad de estructurar y representar datos. Pero además de facilitar la organización de los recursos y la configuración de un programa, cumple un papel muy importante que es, sin lugar a dudas, su punto fuerte: le permite comunicarse con otras aplicaciones, de diferentes plataformas y sin que importe el origen de la información en común.

3.3.3 SQLite

Es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y por supuesto ser de código libre. A diferencia de los sistema de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo.

Android incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos SQLite, y entre ellas una completa API para llevar a cabo de manera sencilla todas las tareas necesarias.

3.3.4 Google Play Services

Es una librería que contiene las interfaces con los servicios individuales de Google y que permite obtener la autorización de los usuarios para obtener acceso a estos servicios con sus credenciales. También contiene las API que permiten resolver cualquier problema en tiempo de ejecución.

Utilizar los servicios de Google Play permite la libertad de usar las nuevas APIs para utilizar las herramientas populares sin preocuparse de la compatibilidad en los dispositivos. Los cambios se distribuyen automáticamente por Google Play Store, y las nuevas versiones de la librería se obtienen a través del SDK de Android, lo que hace que sea más fácil desarrollar la aplicación sin tener que preocuparse de estos servicios.

3.3.5 Técnicas de debugging

Para desarrollar una aplicación Android de este tipo, hay que realizar muchas tareas de debugging (depuración). Es muy importante disponer de un entorno que nos muestre la mayor información posible acerca de los errores y los valores que tiene asignados cada variable en tiempo real. Android Studio ofrece un entorno muy bien pensado para la depuración de código Java, debido a esto, ha sido una de las técnicas que más me ha ayudado a lo largo del desarrollo.

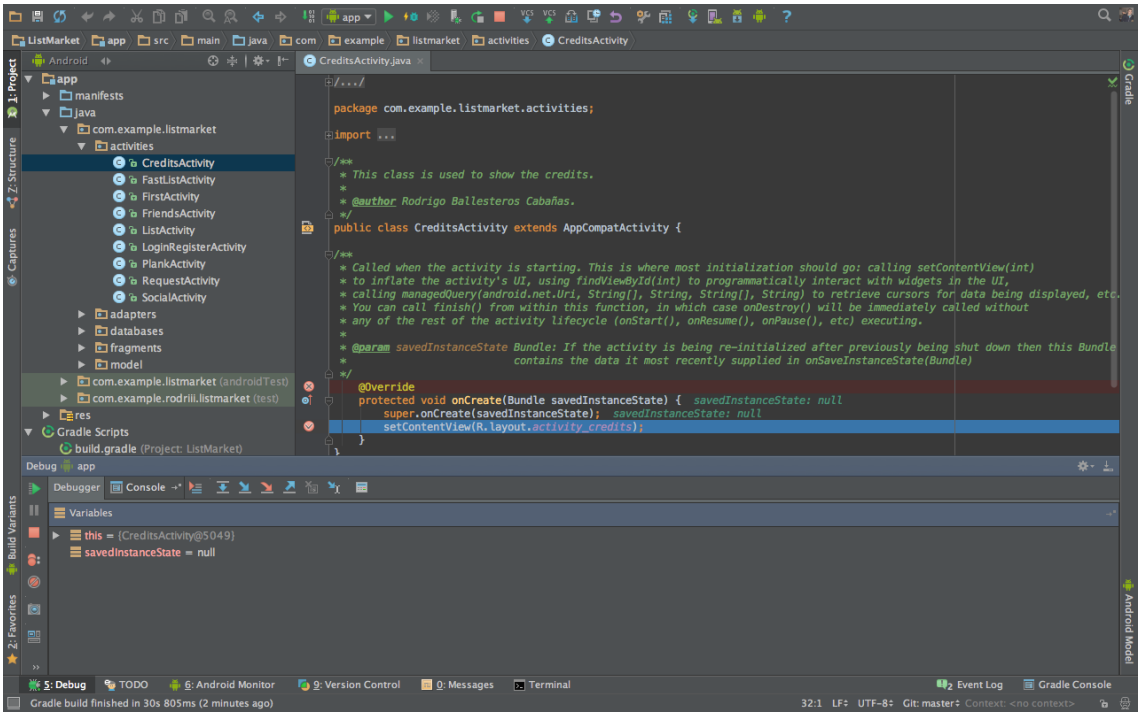

Figura nº 6. Actividad créditos utilizando la opción debug

3.3.6 Control de versiones

Es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. Una versión o revisión de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación. Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. Ejemplos de este tipo de herramientas son entre otros: CVS, Subversion, GitHub, etc. Este último ha sido el que he utilizado para el control de versiones en mi aplicación ya que tiene muy buena integración en Android Studio.

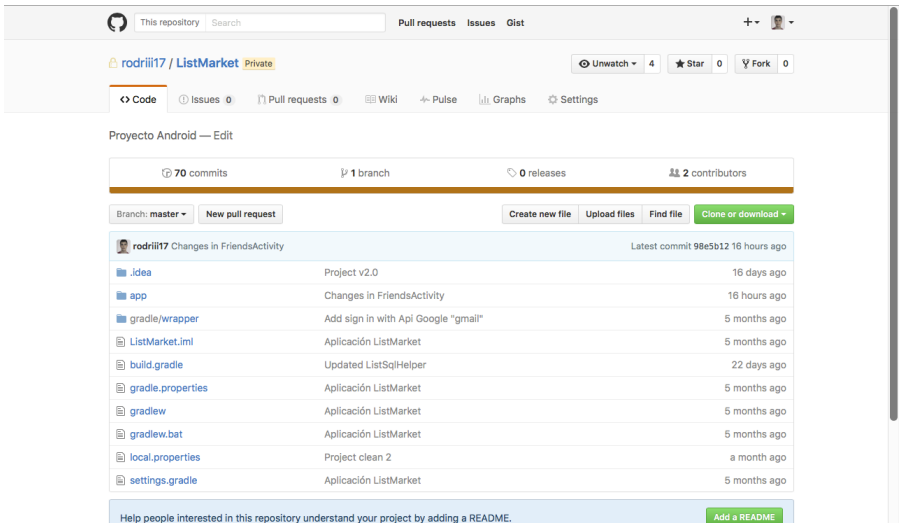

Figura nº 7. Aplicación web de GitHub

Además, Android Studio trabaja de manera excepcional con este tipo de software, ya que incluye un plugin por defecto que incorpora toda la funcionalidad de Git. Es por tanto un sistema que gestiona el trabajo en grupo, integrando los cambios realizados por cada usuario en el proyecto.

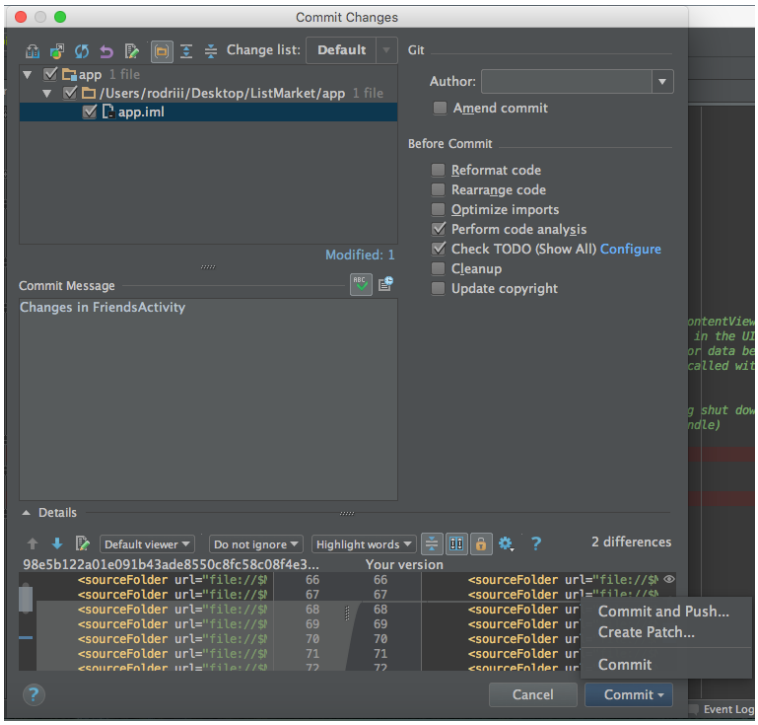

Figura nº 8. Plugin de GitHub en Android Studio

3.4 Firebase - Google

Firebase es la nueva y mejorada plataforma de desarrollo móvil en la nube de Google. Se trata de una plataforma disponible para diferentes plataformas (Android, iOS, web). En muchas ocasiones nos planteamos cómo acceder a un servicio web para tener nuestra aplicación trabajando con datos en la nube. Por ello surgió Firebase, para proveer una API donde guardar y sincronizar los datos en la nube en tiempo real. Uno de los aspectos que más hay que destacar es la asombrosa documentación que se puede consultar cuando accedemos a la plataforma. Hay un gran cantidad de información interesante y necesaria disponible para todo aquel desarrollador que quiera probar suerte en esta plataforma.

Sus características fundamentales están divididas en varios grupos, podemos agruparlas en:

- Analíticas: Provee una solución gratuita para tener todo tipo de medidas (hasta 500 tipos de eventos), para gestionarlo todo desde un único panel.

- Desarrollo: Permite construir mejores apps, permitiendo delegar determinadas operaciones en Firebase, para poder ahorrar tiempo, evitar bugs y obtener un aceptable nivel de calidad. Entre sus características destacan el almacenamiento, testeo, configuración remota, mensajería en la nube o autenticación, entre otras.

- Crecimiento: Permite gestionar los usuarios de las aplicaciones, pudiendo además captar nuevos. Para ello dispondremos de funcionalidades como las de invitaciones, indexación o notificaciones.

- Monetización: Permite ganar dinero gracias a AdMob.

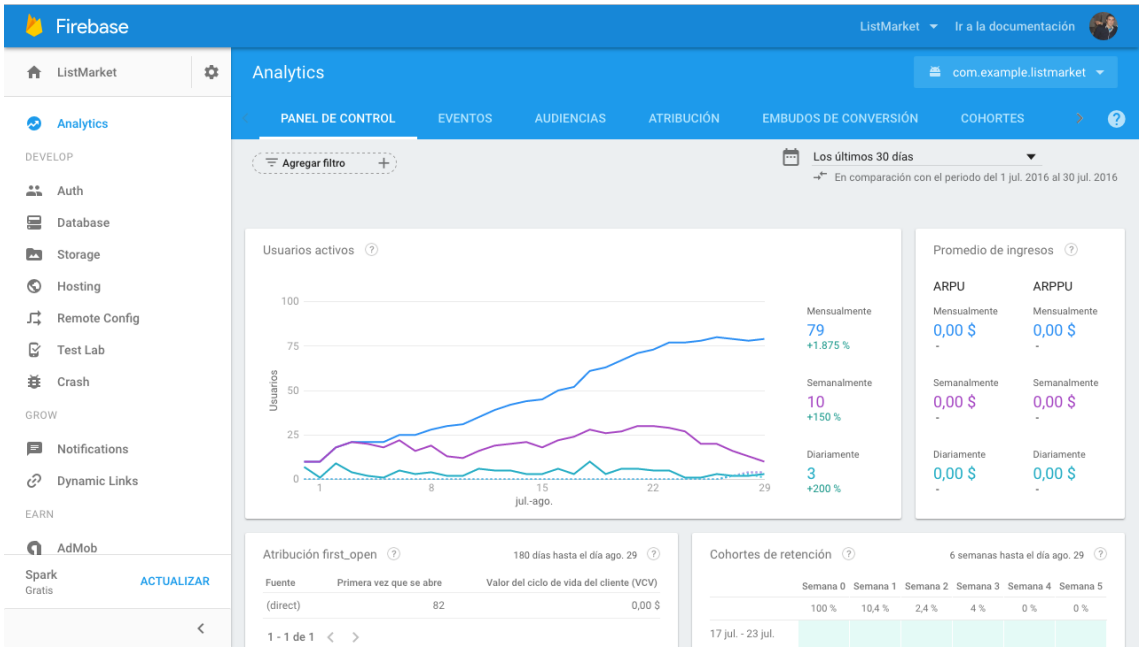

Figura nº 9. Aplicación web de Firebase

3.4.1 Autenticación

Muchas de la apps que existen actualmente reconocen la identidad de un usuario. El reconocimiento de la identidad de un usuario permite guardar los de datos de este de manera segura en la nube, ya sea para consultar sus datos o para añadir nuevos.

Firebase Authentication proporciona servicios de backend, SDK fáciles de usar y bibliotecas de IU ya hechas para autenticar usuarios en tu app. Admite autenticación con contraseñas y proveedores de identidades federadas populares, como Google y Facebook. Se integra estrechamente con otros servicios de Firebase y aprovecha estándares industriales como OAuth 2.0 y OpenID Connect. Por lo tanto, se puede integrar fácilmente con tu backend personalizado.

Puedes iniciar sesiones usando FirebaseUI como una solución de autenticación directa completa, o bien usar el Firebase Authentication SDK para integrar manualmente uno o varios métodos de inicio de sesión a tu app. En nuestro caso, hemos elegido la segunda opción dado que tenemos varios métodos de inicio de sesión.

3.4.2 Base de datos en tiempo real

Firebase dispone de una base de datos NoSQL alojada en la nube donde se pueden almacenar y sincronizar los datos que vamos generando en nuestra aplicación. Estos datos son sincronizados con todos los clientes en tiempo real y siguen estando disponible cuando la aplicación pierde la conexión.

La Firebase Realtime Database es una base de datos alojada en la nube. Los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada cliente conectado. También proporciona un lenguaje de reglas flexibles basadas en expresiones llamado Security Rules de Firebase Realtime Database, donde se define el modo en que los datos se deben estructurar y el momento en que se pueden someter a lectura y escritura. Es importante pensar en el modo en que los usuarios necesitan acceder a los datos y luego estructurarlos de forma adecuada.

3.4.2.1 Estructuración de los datos

La construcción de una base de datos estructurada de manera apropiada requiere un poco de previsión. Lo más importante es que necesitas planear cómo guardar los datos y luego recuperarlos para que ese proceso sea lo más sencillo posible.

Todos los datos de Firebase Realtime Database se almacenan como objetos JSON. Se plantea la base de datos como un árbol JSON alojado en la nube. A diferencia de la base de datos SQL, no existen tablas ni registros. Cuando se agregan datos al árbol JSON, estos se convierten en un nodo en la estructura JSON existente.

Los datos pueden anidarse hasta 32 niveles de profundidad, no obstante no hay que excederse. Cuando obtienes datos en una ubicación de tu base de datos, también recuperas todos los nodos secundarios. En la práctica, es mejor mantener la estructura de datos con la mayor simpleza posible. Si los datos, en cambio, se dividen en rutas separadas, también llamadas no normalizadas, se pueden descargar de manera eficiente en llamadas separadas, a este tipo de estructura se le llama compactada.

4. Especificación de requisitos

En este apartado se va dar una descripción completa del comportamiento del sistema que se va a desarrollar. Más adelante definiremos el conjunto de casos de uso que describe todas las interacciones que van a tener los usuarios con el software. Los casos de uso también son conocidos como requisitos funcionales. Además de los casos de uso, también contiene requisitos no funcionales (o complementarios). Los requisitos no funcionales son requisitos que imponen restricciones en el diseño o la implementación.

Está dirigida tanto al cliente como al equipo de desarrollo. El lenguaje utilizado para la redacción debe ser informal, de forma que sea fácilmente comprensible para todas las partes involucradas en el desarrollo.

4.1 Introducción

En primer lugar es necesario definir algunas características del proyecto, tales como el propósito y el ámbito, requerimientos técnicos por los que ha de regirse el desarrollo de la aplicación

4.1.1 Propósito

El propósito de este capítulo es definir los requerimientos que debe tener y cumplir la aplicación desarrollada. Esta especificación de requisitos tiene como objetivo formalizar las funcionalidades, de forma que haya una base con la que contrastar el desarrollo de la aplicación, y así poder realizar el desarrollo de una forma más sencilla y guiada.

También se harán varias entrevistas con los usuarios finales para obtener la descripción funcional de la aplicación. Estas entrevistas se programarán de tal forma que permitan a los analistas reflexionar sobre la información recibida, de esta manera se evitará recibir información de forma ambigua o contradictoria.

4.1.2 Ámbito

La aplicación está destinada a dispositivos móviles Android en la cual se realizarán todas y cada una de las distintas acciones posibles, ya sea crear listas clasificándolas en diversas categorías como enviar y recibir peticiones de amistad.

Dentro de las características que implementa la aplicación, el usuario puede registrarse o identificarse para habilitar las funciones de los amigos y de compartir sus listas personales. Si decide no registrarse, puede probar la aplicación en modo offline, de esta manera solo se pueden crear listas y clasificarlas.

La funcionalidad principal podría resumirse en permitir la gestión de peticiones de amistad y decidir quien es tu amigo y quien no. Esta función es clave ya que se pueden compartir listas solo con tus amigos, aquellas personas que no sean aceptadas o se elimine su petición no podrán recibir nuestras listas.

Cabe mencionar que esta aplicación se complementa con otro proyecto web donde se irán almacenando los datos de los usuarios cuando se registren, los datos de nuestros amigos y los de las listas compartidas con ellos.

4.2 Descripción general

A continuación, se detallan una serie de apartados que describen los objetivos del producto, así como su funcionalidad y requisitos.

4.2.1 Perspectiva del producto

El acceso a la aplicación se puede realizar desde cualquier parte del plantea mediante un dispositivo móvil. Si se dispone de acceso a Internet se podrán hacer amigos, enviar solicitudes de amistad y compartir listas, en caso contrario servirá a modo de almacén de listas. Está dirigida a cualquier persona que desee compartir datos de sus compras ya sean amigos, familiares o compañeros de un piso de estudiantes.

La aplicación está diseñada en Android Studio. Utiliza Java como lenguaje de programación y la API de Android, para habilitar los distintos componentes que usamos en la aplicación. Además de gestionar una base de datos interna en SQLite que permite administrar las diferentes listas y sus respectivas categorías.

Debido a que la aplicación contiene una serie de características propias, es necesario usar el servicio de alojamiento web que proporciona Firebase.

4.2.2 Funciones del producto

Este apartado describe cuáles son las funcionalidades que debe proporcionar la aplicación. Las funciones principales que la aplicación debe permitir son las siguientes:

1. Gestión de registro e identificación:
   a) Registrar anónimamente.
   b) Registrar mediante Google.
   c) Identificarse con un usuario registrado.
   d) Cerrar sesión.

2. Gestión de listas:
   a) Asignar nombre a una lista.
   b) Asignar categoría a una lista.
   c) Restablecer una lista.
   d) Guardar una lista.
   e) Consultar una lista.
   f) Eliminar una lista.
   g) Compartir una lista.

3. Gestión de productos:

   a) Añadir un producto con su precio.

   b) Modificar precio de un producto.

   c) Eliminar un producto.

   d) Consultar precio total.

4. Gestión de amigos:

   a) Enviar petición de amistad.

   b) Aceptar petición de amistad.

   c) Refrescar los amigos y las peticiones.

   d) Eliminar una petición.

   e) Eliminar un amigo.

4.2.3 Características de usuario

La aplicación puede ser manejada por un usuario sin identificar. Este usuario debe ser capaz de realizar todas las operaciones y funciones comentadas en el apartado anterior, excepto la función de compartir listas, enviar solicitudes de amistad y gestionar los amigos, que estarán vetadas hasta que se registren.

Por otro lado, los usuarios que se hayan registrado podrán gozar de los servicios online, requerirán acceso a internet. Un detalle importante a tener en cuenta es que los usuarios, solo podrán compartir sus listas con aquellas solicitudes que hayan decidido aceptar, ya que pasarán a ser amigos.

4.2.4 Restricciones generales

En cuanto a las restricciones, la necesidad de trabajar con una base de datos con una estructura preestablecida, puede hacer que funcione de manera incorrecta si se modifica dicha estructura o se elimina parcialmente. En caso de tener que modificar la estructura de la base de datos, será necesario realizar correcciones en la aplicación.

A la hora de usar la aplicación, hay que tener en cuenta que es necesario disponer de acceso a internet. Esto puede realizarse mediante conexión Wi-Fi o mediante conexiones móviles 4G. Hay que tener en cuenta que la velocidad con la que la aplicación responda a las peticiones dependerá tanto de la velocidad de la conexión como del rendimiento del servidor de Google en ese momento, así pues, siempre será mejor disponer de una conexión de alta velocidad.

4.3 Requisitos específicos

En este apartado se hace una descripción más exhaustiva de todos aquellos requerimientos que debe poseer la aplicación, tanto a nivel funcional como de interfaz y diseño. Para concluir, se mencionarán algunos atributos presentes en la aplicación.

4.3.1 Requisitos funcionales

Registrar anónimamente o mediante Google.

Introducción: Los usuarios pueden elegir registrarse en la aplicación o bien, continuar sin hacerlo. Si deciden registrarse hay dos métodos: anónimamente o mediante Google.

Entrada: Botón registrarse en la pantalla de login, para hacerlo anónimo o botón de Google en la pantalla de bienvenida, para hacerlo con la cuenta de Google.

Proceso: El usuario accede a la pantalla de login, introduce sus datos y presiona el botón de registrarse o en la pantalla de bienvenida, presiona el botón de Google y selecciona su cuenta.

Salida: Si no ocurre ningún error la aplicación añade el email a la pantalla de bienvenida, a la base de datos en la nube de Firebase y a la base de datos de SQLite interna, concretamente a la tabla usuarios.

Identificarse.

Introducción: Los usuarios pueden elegir identificarse en la aplicación o bien, continuar sin hacerlo. Si deciden identificarse hay dos métodos: anónimamente o mediante Google.

Entrada: Botón login en la pantalla de login, para hacerlo anónimo o botón de Google en la pantalla de bienvenida, para hacerlo con la cuenta de Google.

Proceso: El usuario accede a la pantalla de login, introduce sus datos y presiona el botón de login o en la pantalla de bienvenida, presiona el botón de Google y selecciona su cuenta.

Salida: Si no ocurre ningún error la aplicación añade el email a la pantalla de bienvenida y a la base de datos de SQLite interna, concretamente a la tabla usuarios.

Cerrar sesión.

Introducción: Solo los usuarios que se han identificado en la aplicación pueden elegir esta opción, sirve para salir de la sesión.

Entrada: Opción cerrar sesión en la pantalla de nueva lista.

Proceso: El usuario accede a la pantalla de nueva lista, presiona el icono para desplegar el panel del lateral izquierdo y selecciona la opción de cerrar sesión.

Salida: La aplicación borra el email de la pantalla de bienvenida y de la base de datos de SQLite interna, concretamente de la tabla usuarios.

Asignar nombre y categoría a una lista.

Introducción: Estos diálogos aparecen cuando se accede a crear una nueva lista, con ellos se puede elegir el nombre y categoría de la lista que se va a crear. Si ya hay una lista en curso no aparecerán hasta que se haya guardado.

Entrada: Diálogo nombre de lista y diálogo categoría en la pantalla de nueva lista.

Proceso: El usuario accede a la pantalla de nueva lista, rellena los datos y presiona el botón de ok tanto en un diálogo como en otro.

Salida: La aplicación guarda el valor introducido tanto en un diálogo como en otro a una variable, y con la ayuda de la clase Shared Preferences, se podrá recuperar el valor aunque el usuario se salga de aplicación. Estas variables no se reiniciarán hasta que el usuario decida guardar la lista o eliminarla.

Restablecer una lista.

Introducción: Esta acción solo puede realizarse cuando el usuario ha introducido un nombre para la lista y ha elegido una categoría. Con esta opción se restablece la lista a su estado inicial, es decir, vacía.

Entrada: Botón de borrar en la barra superior de la pantalla nueva lista.

Proceso: El usuario accede a la pantalla de nueva lista, presiona el icono de borrar en la barra superior y el botón ok del diálogo emergente para aceptar la acción.

Salida: La aplicación vacía la lista, su nombre y su categoría.

Guardar una lista.

Introducción: Esta acción solo puede realizarse cuando el usuario ha introducido un nombre para la lista y ha elegido una categoría. Con esta opción se pueden organizar todas las listas que se guardan en diferentes carpetas según la categoría.

Entrada: Botón de guardar en la barra superior de la pantalla nueva lista.

Proceso: El usuario accede a la pantalla de nueva lista, presiona el icono de guardar en la barra superior y el botón ok del diálogo emergente para aceptar la acción.

Salida: La aplicación añade la nueva lista al tablón de la categoría que se haya elegido, junto con su nombre y el precio total de la lista. También se añaden estos valores a la base de datos de SQLite interna, concretamente a la tabla de la categoría seleccionada. Por último, genera un fichero en el dispositivo móvil con el nombre de la lista, que será utilizado cuando el usuario quiera acceder a visualizarla y será modificado cuando el usuario altere los productos y precios.

Cargar una lista.

Introducción: Esta acción solo puede realizarse cuando el usuario ha guardado previamente una lista. Con esta opción se pueden consultar las listas de la compra que se han ido almacenando en la aplicación.

Entrada: Presionar la lista en la pantalla de los tablones.

Proceso: El usuario accede a la pantalla de los tablones, navega por las diferentes categorías y presiona la lista que quiere consultar.

Salida: La aplicación busca el archivo guardado en el almacenamiento externo del dispositivo móvil y carga los datos en la pantalla de compartir la lista. Para llevar acabo la acción correctamente, los datos del archivo son añadidos a la base de datos de SQLite interna, concretamente a la tabla productotxt.

Eliminar una lista.

Introducción: Esta acción solo puede realizarse cuando el usuario ha guardado previamente una lista. Con esta opción se elimina la lista para siempre.

Entrada: Mantener presionada la lista a borrar en la pantalla de los tablones.

Proceso: El usuario accede a la pantalla de los tablones, navega por las diferentes categorías, mantiene presionada la lista y acepta eliminarla en el diálogo emergente.

Salida: La aplicación borra la lista del tablón y de la base de datos de SQLite interna, concretamente de la tabla de la categoría. Además, borra el fichero del dispositivo móvil que se había generado al guardarla y, en caso de estar en la categoría online, también la borra de la base de datos en la nube de Firebase.

Compartir una lista.

Introducción: Esta acción solo puede realizarse cuando el usuario se ha identificado en la aplicación y ha guardado previamente una lista. Con esta opción se pueden compartir las listas de la compra con los amigos que hay agregados.

Entrada: Botón de compartir en la barra superior de la pantalla compartir la lista.

Proceso: El usuario accede a la pantalla de compartir la lista, presiona el icono de compartir en la barra superior, introduce el email de un amigo agregado y pulsa el botón ok del diálogo para realizar la acción.

Salida: La aplicación recoge los datos de la lista y la envía al tablón online de nuestro amigo, es decir, a la base de datos en la nube, concretamente a su rama.

Añadir un producto con su precio y modificarlo.

Introducción: Esta acción solo puede realizarse cuando el usuario ha introducido un nombre para la lista y ha elegido una categoría. Con esta opción se puede añadir el producto a nuestra lista junto con su precio y más adelante, si es necesario, modificarlo con el nuevo precio.

Entrada: Botón flotante de añadir en la pantalla de nueva lista, para insertar un producto o presionarlo en la lista, para modificarlo.

Proceso: El usuario accede a la pantalla de nueva lista, rellena los datos, pulsa el botón flotante de añadir, elige un precio y pulsa el botón ok del diálogo para añadirlo, o el usuario accede a la pantalla de nueva lista, presiona un producto, elige un nuevo precio y pulsa el botón ok del diálogo para modificarlo.

Salida: La aplicación añade el producto junto con su precio a la lista y, a la base de datos de SQLite interna, concretamente a la tabla producto. Si solamente se modifica, busca el producto en la base de datos y sustituye el precio por el nuevo elegido.

Eliminar un producto.

Introducción: Esta acción solo puede realizarse cuando el usuario ha introducido un nombre para la lista y ha elegido una categoría y, además, ha añadido algún producto. Con esta opción se elimina el producto de la lista y de la base de datos.

Entrada: Mantener presionado el producto a borrar en la pantalla de nueva lista.

Proceso: El usuario accede a la pantalla de nueva lista, mantiene presionado el producto y acepta eliminarlo en el diálogo emergente.

Salida: La aplicación borra el producto de la lista y de la base de datos de SQLite interna, concretamente de la tabla producto.

Consultar el precio total.

Introducción: Con esta opción consultamos el precio total de la lista. Si la lista no se ha creado aparecerá como precio total 0,0€.

Entrada: Botón flotante de consultar precio total en la pantalla de nueva lista.

Proceso: El usuario accede a la pantalla de nueva lista y presiona el botón flotante para consultar el precio.

Salida: La aplicación suma todos los productos añadidos en la lista y aparece un mensaje Toast en la pantalla, indicándonos el total de nuestra lista.

Enviar una petición de amistad.

Introducción: Esta acción solo puede realizarse cuando el usuario se ha identificado en la aplicación. Con esta opción se pueden enviar peticiones de amistad a nuestros amigos para que sea posible compartir listas con ellos.

Entrada: Botón flotante de enviar petición en la pantalla de peticiones.

Proceso: El usuario accede a la pantalla de peticiones, rellena los datos con el email al que enviar la petición y pulsa el botón flotante.

Salida: La aplicación recoge los datos y envía una petición con nuestro email a su pantalla de solicitudes, es decir, a la base de datos en la nube, concretamente a su rama.

Refrescar los amigos y las peticiones.

Introducción: Esta acción solo puede realizarse cuando el usuario se ha identificado en la aplicación. Con esta opción se refresca la pantalla para actualizar los datos.

Entrada: Botón de refrescar en la barra superior de la pantalla amigos o peticiones.

Proceso: El usuario accede a la pantalla de amigos o peticiones y presiona el icono de refrescar en la barra superior.

Salida: La aplicación reinicia la pantalla y la inicia con los datos actualizados tanto de solicitudes como de amigos.

Aceptar o eliminar una petición de amistad.

Introducción: Esta acción solo puede realizarse cuando el usuario se ha identificado en la aplicación. Con esta opción se pueden aceptar o denegar las peticiones de amistad de nuestros amigos ya sea para compartir listas con ellos o no ser amigos, respectivamente.

Entrada: Presionar una solicitud de la lista en la pantalla de solicitudes, para aceptarla o mantenerla presionada, para eliminarla.

Proceso: El usuario accede a la pantalla de peticiones, presiona una solicitud y pulsa el botón ok del diálogo emergente o el usuario accede a la misma pantalla, mantiene presionada la petición y pulsa el botón ok del diálogo emergente.

Salida: Si la solicitud es aceptada, la aplicación elimina la solicitud de la lista y de la base de datos en la nube, y añade al usuario a nuestra pantalla de amigos y a la suya, debido a que los datos de los respectivos emails son añadidos también a la rama correspondiente en la nube. Por el contrario si la solicitud es denegada, se elimina de la lista de peticiones y de la base de datos en la nube.

Eliminar un amigo.

Introducción: Esta acción solo puede realizarse cuando el usuario se ha identificado en la aplicación. Con esta opción se eliminan nuestros amigos y ya no se podrán compartir listas con ellos hasta que no vuelvan a ser agregados.

Entrada: Mantener presionado el email del amigo en la pantalla de los amigos.

Proceso: El usuario accede a la pantalla de amigos, mantiene presionado uno y pulsa el botón ok del diálogo emergente.

Salida: La aplicación elimina el email de la lista y por tanto, de la base de datos en la nube.

4.3.2 Requisitos de interfaz

Con respecto a los requisitos de interfaz podemos diferenciar tres tipos: interfaz de usuario, interfaz software e interfaz hardware.

En cuanto a la interfaz de usuario, el principal objetivo es conseguir una interfaz simple y sencilla de manejar. Puesto que la aplicación está destinada a dispositivos móviles y con diversas funciones, no existe un patrón básico que compartan las distintas interfaces. Aún así, debido al carácter de la aplicación y su funcionalidad orientada al uso de compras y por tanto la necesidad del uso de listas, la interfaz más común es la empleada por las funciones de añadir productos y precios, la cual consiste en una lista de productos junto con sus precios. Por otro lado, a la hora de registrar un usuario la interfaz consiste en diversos campos a rellenar. También se encuentra la interfaz donde se cargan las listas, formada por el nombre y el total. Además, para la funcionalidad de las solicitudes y los amigos, se dispone de una lista para agregarlas/os o eliminarlas/os. Por último, para acceder a todas estas opciones descritas se utiliza el panel del lateral izquierdo llamado Navigation Drawer.

En referencia a la interfaz software, el proyecto se desarrolla empleando el sistema operativo OS X El Capitan, con el entorno de desarrollo Android Studio y la plataforma Firebase como servidor. Java es el lenguaje principal puesto que Android hace uso de dicho lenguaje. Para el sistema de gestión de base de datos se emplea NoSQL.

Como requisito o interfaz hardware es importante mencionar la necesidad de emplear un dispositivo móvil con el sistema operativo Android y conexión a Internet, además de un servidor, en este caso de Google, con soporte para la tecnología NoSQL.

4.3.3 Restricciones de diseño

A diferencia de una página web en la que sería adecuado seguir los estándares marcados por el W3C, a la hora de desarrollar una aplicación móvil no existen unos estándares básicos que seguir. Aún así, se han seguido en la medida de lo posible una serie de mejores prácticas sugeridas por la guía de desarrolladores Android ofrecida en su página web, mediante la cual se ofrece información por ejemplo, de qué hacer y cómo realizarlo para adaptar la interfaz a distintos tamaños de pantalla.

4.3.4 Atributos

La aplicación creada no debería necesitar mantenimiento puesto que funciona mediante los datos almacenados en la base de datos en la nube, así pues, el mantenimiento de los datos recae sobre el usuario ya que será el encargado de actualizarlos o modificarlos mediante el uso de la aplicación.

Por otro lado, ya que la aplicación hace uso de un sistema de acceso de usuarios, para poder acceder será necesario disponer de un email y una contraseña.

4.3.5 Otros requisitos

Dentro de los requisitos para la aplicación, es esencial que la aplicación funcione en la mayoría de dispositivos Android posibles pero debido al gran avance de la tecnología muchos de los recursos utilizados en la aplicación necesitan una alta versión de Android para mostrarse correctamente tanto en diseño como en implementación. Por este motivo, se establece como requisito, que la aplicación funcione en dispositivos que dispongan de Android 4.0 o superior con API 19 como mínimo.

5. Análisis

En este capítulo se va a proceder a realizar la especificación mediante los casos de uso, una técnica que permite expresar de forma simplificada el comportamiento de un sistema ante la interacción de los usuarios. Su utilización está indicada especialmente para sistemas interactivos debido a que reflejan las acciones de un usuario cuando hace uso de una funcionalidad en la aplicación.

5.1 Casos de uso

Son una descripción de los pasos que deberán realizarse para llevar a cabo algún proceso. Los personajes o entidades que participan en un caso de uso se denominan actores. Un caso de uso es una secuencia de interacciones que se desarrollará entre un sistema y sus actores, en respuesta a un evento que inicia un actor principal sobre el propio sistema. Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema.

5.1.1 Tipos de relaciones

A la hora de completar los casos de uso, se pueden utilizar varias relaciones. Las más importantes son:

- <<communicates>>: Relación de asociación entre un actor y un caso de uso que denota la participación del actor en dicho caso de uso. En la práctica se suele omitir.

- <<include>>: Relación de dependencia entre dos casos de uso que denota la inclusión del comportamiento de un escenario en otro.

- <<extends>>: Relación de dependencia entre dos casos de uso que denota que un caso de uso es una especialización de otro.

Se utiliza una relación de tipo <<extends>> entre casos de uso cuando nos encontramos con un caso de uso similar a otro pero que hace algo más que éste. Por contra, utilizaremos una relación tipo <<include>> cuando nos encontramos con una parte de comportamiento similar en dos casos de uso y no queremos repetir la descripción de dicho comportamiento común.

5.1.2 Ventajas

Como técnica de extracción de requerimiento permite que el analista se centre en las necesidades del usuario, qué espera éste lograr al utilizar el sistema. Aunque comúnmente se asocian a la fase de Test de una aplicación, esta idea es errónea, y su uso se extiende mayormente a las primeras fases de un desarrollo. A continuación, se pueden observar los casos de uso de un usuario (véase figura 10).

Figura nº 10. Caso de Uso - Usuario

6. Diseño

La etapa de diseño permite describir como el sistema va a satisfacer los requisitos. Esta etapa a menudo tiene diferentes niveles de detalle. Los niveles más altos de detalle, generalmente describen los componentes o módulos que formarán el software a ser producido. Los niveles más bajos describen con mucho detalle cada módulo que contendrá el sistema.

Es importante anotar que la fase de diseño no solo aporta a la definición del proyecto las especificaciones sobre los requisitos funcionales, sino que también provee las especificaciones sobre distintas áreas, tales como la escalabilidad, disponibilidad, portabilidad, flexibilidad, etc.

La arquitectura de la aplicación es una arquitectura típica cliente-servidor. Por un lado el cliente es la propia aplicación Android y por otro, el servidor se corresponde con una base de datos NoSQL alojada en la nube que ofrece soporte para la persistencia.

En el siguiente apartado se va a describir el software utilizado para desarrollar el proyecto.

6.1 API de Android

Android puede ser considerado como un framework del lenguaje de programación Java, es decir, un esquema de desarrollo específico creado para la implementación de una aplicación sobre dicho lenguaje. De manera general, la ventaja de utilizar un framework radica en que el desarrollador no tiene que definir una estructura global de la aplicación, sino que tiene que ir añadiendo estructuras y servicios que le ofrece el propio framework. A continuación se van a describir los elementos y estructuras utilizados a la hora de programar en Android.

6.1.1 AndroidManifest

Está situado en la raíz de nuestras aplicaciones como AndroidManifest.xml, es un archivo de configuración donde podemos aplicar las configuraciones básicas de nuestra app. Su configuración puede realizarse a través de una interfaz gráfica, pero es recomendable hacerlo desde el propio xml. En él se declaran todas las actividades que se han utilizado en la aplicación y en cada una de ella se declaran valores como el nombre, el estilo y el punto de entrada a la aplicación. En la figura 11 se observa el manifiesto de la aplicación desarrollada.

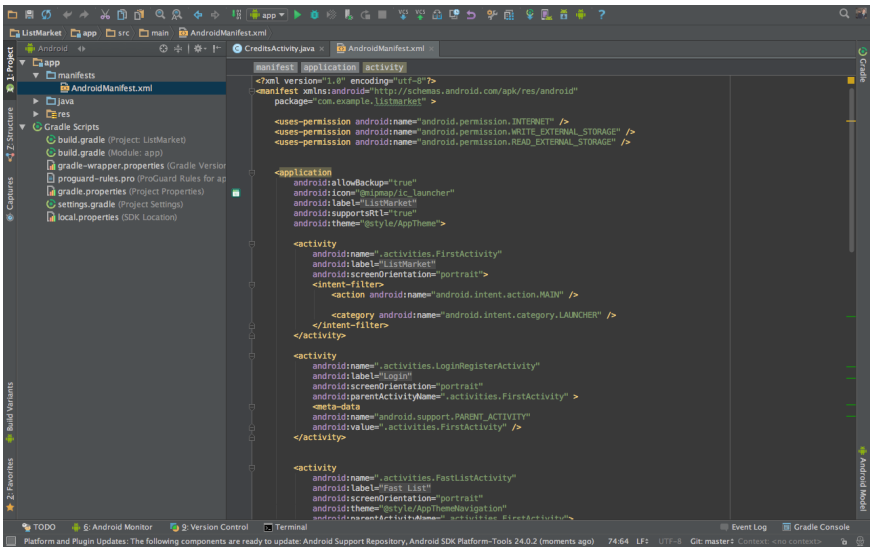

Figura nº 11. AndroidManifest.xml

6.1.2 Views y Layouts

La interfaz de usuario se define en los archivos XML del directorio res/layout. Cada pantalla tiene un código XML diferente. Diseñar una pantalla usando Java puede resultar complejo y poco eficiente, sin embargo, Android soporta XML para diseñar pantallas y define elementos personalizados, cada uno representando a una "subclase" especifica de view.

Cada fichero describe un layout y cada layout a su vez puede contener otros elementos. Estos elementos pueden estar formados por view. Un view es un objeto cuya clase es android.view.View. Es una estructura de datos cuyas propiedades contienen los datos de la capa, la información específica del área rectangular de la pantalla y permite establecer el layout. Es útil como clase base para los widgets, que son unas subclases ya implementadas que dibujan los elementos en la pantalla. Los widgets contienen sus propias medidas, pero puedes usarlas para construir tu interfaz más rápidamente. La lista de widgets que puedes utilizar incluyen Text, EditText, Button, RadioButton, CheckBox, etc. A continuación, se puede observar de forma esquemática en la figura 12 una construcción de layouts con sus respectivas views.

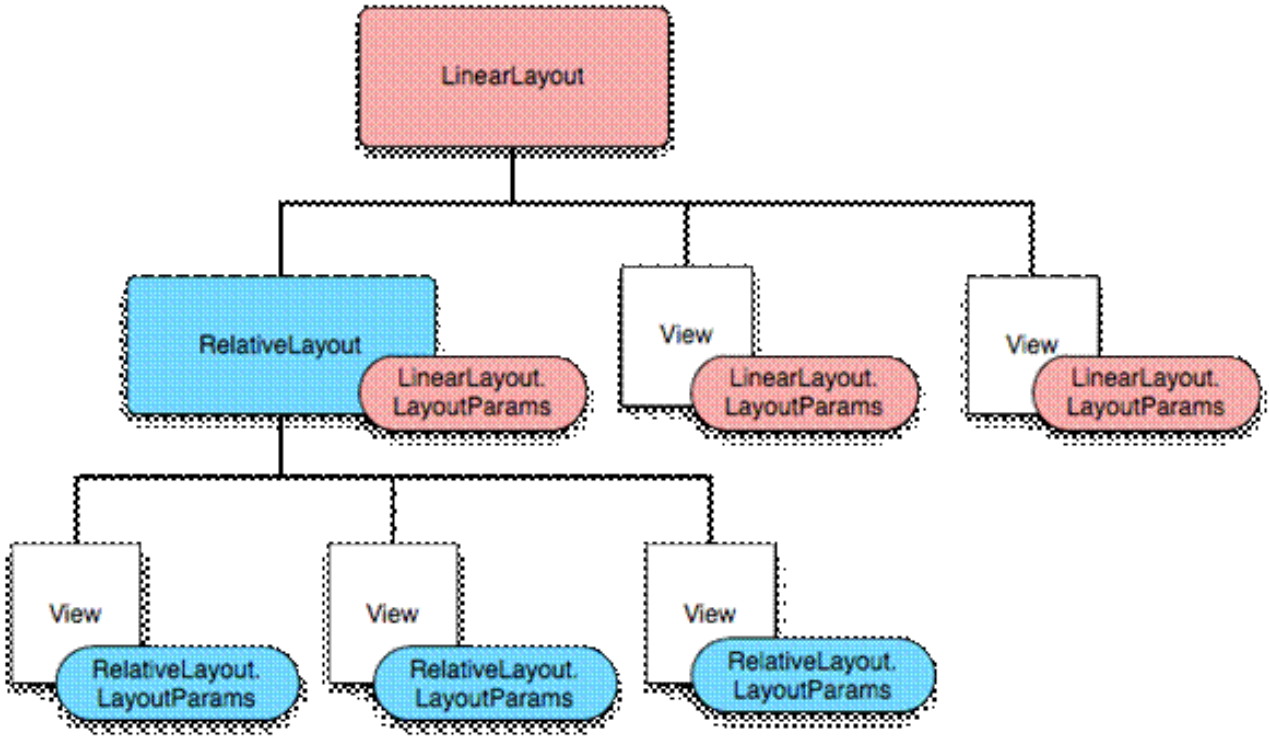

Figura nº 12. Ejemplo de Layouts y Views

6.1.3 Activities

Se usa el termino de Activity para denominar a un tipo de clases java que heredan de Activity. Una actividad, como su propio nombre indica, es algo que el usuario puede hacer. En Android, una actividad es un conjunto de acciones (tocar la pantalla para apretar un botón, para escribir con el teclado, etc) que son una iteración directa con el usuario y que afectan a una parte de la aplicación. También representa la lógica de negocio de una pantalla de la aplicación visualizada gracias a una interfaz gráfica de usuario (layouts).

El método más importante en las activities es OnCreate, en él se inicializan todas las referencias a las views de nuestros layouts, además de añadir la funcionalidad de cada uno de ellos, los eventos de escucha, etc.

6.1.4 Fragments

Un fragmento podría definirse como una porción de la interfaz de usuario que puede añadirse o eliminarse de la interfaz de forma independiente al resto de elementos de la actividad, y que por supuesto puede reutilizarse en otras actividades. Son como pequeñas actividades contenidas dentro de una actividad anfitriona, manejando su propio diseño y ciclo de vida. Los fragmentos facilitan el despliegue de las aplicaciones en cualquier tipo de tamaño de pantalla y orientación.

Otra ventaja de usarlos es que permiten crear diseños de interfaces de usuario de múltiples vistas ya que los fragmentos son imprescindibles para generar actividades con diseños dinámicos, como por ejemplo el uso de pestañas en un View Pager.

6.1.5 Fragment Pager Adapter

Representa cada vista como un fragmento que se mantiene persistente en el gestor de fragmentos para que el usuario siempre pueda volver a cada página. De esta forma, puede ser muy útil cuando existen varios fragmentos que van a utilizarse de manera similar. En nuestro caso se va a utilizar para cargar las listas en diferentes categorías junto con su nombre y precio total de la compra.

Como se aprecia en la figura 13, cada View Pager tiene asociado un Fragment Pager Adapter con los diferentes fragmentos a mostrar en él. Para que el adaptador funcione correctamente solo necesita utilizar las subclases getItem (int) y getCount (), donde la primera se utiliza para devolver el fragmento asociado a cada posición y la segundo para llevar la cuenta del número de fragmentos disponibles. También se puede utilizar la subclase getPagerTitle (int) para asociar un título a cada fragmento.

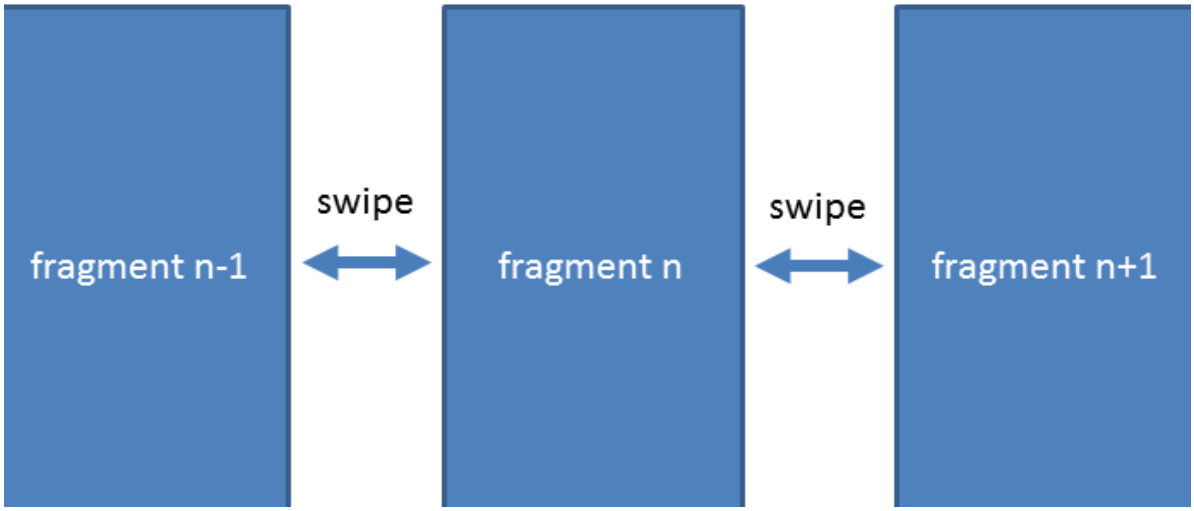

Figura nº 13. Ilustración del Fragment Pager Adapter

6.1.6 Intents

Es un mensaje asíncrono que solicita acciones de otros componentes. Básicamente, sirve para invocar componentes, en android entendemos por componentes las Activities, que representan una única pantalla con interfaz de usuario; los Services, que no disponen de interfaz gráfica, y realizan tareas costosas en segundo plano; los ContentProviders, que permiten compartir datos entre aplicaciones; y por último, los BroadcastReceiver que permiten responder a notificaciones del sistema.

Sus métodos principales son:

- startActivity() para lanzar una actividad.
- startService() para lanzar un servicio.
- sendBroadcast() para lanzar una notificación.
- bindService() para comunicar un servicio.

En este proyecto se van a utilizar para lanzar actividades y en algunas ocasiones, a la hora de crearlos, para añadir información adicional mediante la función extras, que utiliza la técnica pares (clave-valor). Con esto se va a conseguir pasar información entre las diferentes pantallas de la aplicación.

6.1.7 Adapters

Un adaptador es un objeto que comunica a un ListView (lista de objetos) los datos necesarios para crear las filas de la lista. Es decir, conecta la lista con una fuente de información como si se tratase de un adaptador de corriente que alimenta a un televisor. Además de proveer la información, también genera los Views para cada elemento de la lista. Se puede apreciar el funcionamiento de un adapter en la figura 14.

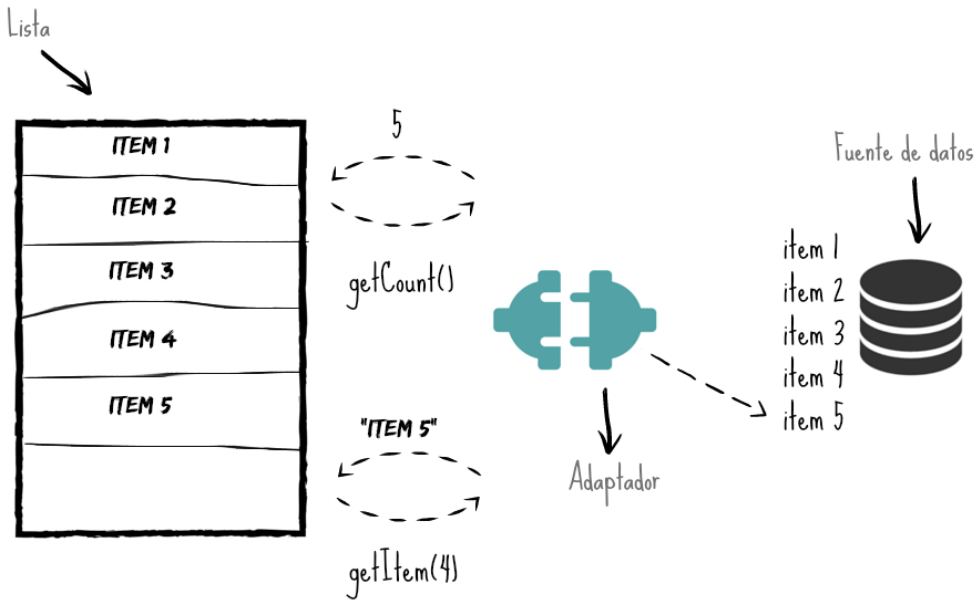

Figura nº 14. Funcionamiento del Adapter

Cuando se relaciona un adaptador a una lista, inmediatamente comienza un proceso de comunicación interno para poblarla con una fuente de datos. Dicha comunicación se basa principalmente en los siguientes métodos del adaptador:

- getCount(), el cual devuelve la cantidad de elementos que hay en la fuente de datos. Con este valor la lista ya puede establecer un límite para añadir items.

- getItem(), con el que se obtiene un elemento de la fuente de datos asignada al adaptador en una posición establecida. Normalmente la fuente de datos es una lista de objetos.

Aunque estos métodos no son los únicos que existen para establecer la relación, son los más significativos para entender el concepto de un adaptador.

6.1.8 Action Bar y Toolbar

Action Bar es la barra de título y herramientas que aparece en la parte superior de muchas de las aplicaciones actuales. Normalmente muestra un icono, el título de la actividad en la que nos encontramos, una serie de botones de acción, y un menú desplegable (menú de overflow) donde se incluyen más acciones que no tienen espacio para mostrarse como botón o simplemente no se quieren mostrar como tal.

Puede implementarse de dos formas:

- Haciendo uso de la funcionalidad básica incluida por defecto en las actividades al utilizar uno de los temas de la librería de soporte y extenderlas de AppCompatActivity (librería de Android).

- Utilizar el nuevo componente Toolbar proporcionado por la librería appcompat. De esta forma se puede incluir de forma explícita la action bar en los layouts XML como si fuera cualquier otro control, y no sólo en la parte superior de la pantalla a modo de app bar, sino también en cualquier otro lugar de la aplicación donde se quiera utilizar esta funcionalidad de barra de acciones.

6.1.9 Navigation Drawer

Es un menú lateral deslizante, el cual aparece en muchas aplicaciones al deslizar el dedo desde el borde izquierdo de la pantalla hacia el lado opuesto (también puede aparecer en el lado derecho, pero es menos frecuente).

Para añadir el navigation drawer a una actividad hay que hacer que el elemento raíz del layout XML sea del tipo <android.support.v4.widget.DrawerLayout>. Y dentro de este elemento colocar únicamente 2 componentes principales: el layout real de la actividad y el layout del menú lateral, que entre otras cosas hace de contenedor de las distintas opciones del menú lateral.

El primero de estos elementos se añade en forma de <include>. Para el segundo se va a utilizar otro de los nuevos componentes incluidos con la nueva librería de diseño de Android (Design Support Library). El componente en cuestión es el llamado NavigationView, que nos ayuda bastante en la construcción del layout del menú lateral.

6.1.10 View Pager

Está constituida por una serie de páginas que contienen los elementos que se quieren mostrar. Para mostrar cada una de las páginas se van pasando hacia adelante o atrás con nuestro dedo, mostrando una animación cada vez que se pasa de una página a otra. Para usar la vista ViewPager hay que tener instalada la librería de soporte de Android (Support Library). También es necesario que tenga un Fragment Pager Adapter asociado para manejar los fragmentos correctamente como se puede observar en la figura 15.

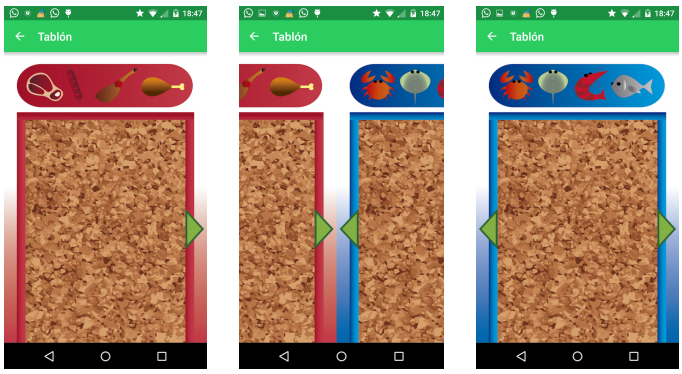

Figura nº 15. Ejemplo de los fragmentos en el View Pager

6.1.11 Mensajes Toast

Son muy útiles para ofrecer información extra en forma de eventos o sucesos específicos. Tienen un tiempo determinado de duración que cuando expira desaparecen automáticamente. Estos mensajes se han utilizado para indicarle al usuario información sobre los siguientes eventos:

- Errores de autenticación de los usuarios.
- Cuando se añaden productos y precios en las listas.
- Al cargar y compartir las listas.

Estos mensajes ayudan a que el usuario no se sienta perdido cuando ocurre algún error y le dan seguridad cuando se introducen datos correctamente. Este tipo de mensajes realimentan la comunicación entre la aplicación y los usuarios, además son sutiles, ocupan poco espacio y duran pocos segundos, con lo que a los usuarios avanzados no les incomoda. En la figura 16 se encuentra un ejemplo de un mensaje Toast que se utiliza en la pantalla de login.

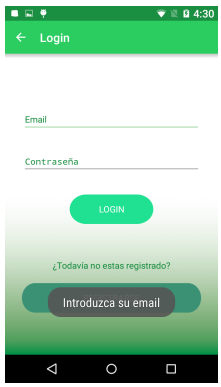

Figura nº 16. Mensaje Toast en la pantalla de login

6.2 Interfaz gráfica de usuario

También conocida como GUI (graphical user interface), es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo.

Uno de los aspectos a cuidar durante todo el desarrollo ha sido el diseño gráfico de la aplicación ya que todas las imágenes se han escalado a los diferentes tamaños de pantalla que hay en el mercado. Según las pulgadas de cada pantalla, los elementos visuales deben de estar en uno de los siguientes tamaños:

- xxxhdpi: 1280x1920 px.
- xxhdpi: 960x1600 px
- xhdpi: 640x960 px
- hdpi: 480x800 px
- mdpi: 320x480 px

6.2.1 Prototipos

En este apartado se van a describir y mostrar los diseños de los prototipos de la interfaz, los cuales deben facilitar el uso de la aplicación a los usuarios. Los prototipos que se van a mostrar a continuación tienen la finalidad de representar los aspectos interactivos de la aplicación con un cierto nivel de precisión, y así poder evaluar su usabilidad y funcionalidad, pero sin entrar en detalles de implementación.

6.2.1.1 Pantalla de bienvenida

La pantalla de bienvenida se ha diseñado pensando en que el usuario acceda a crear una nueva lista sin necesidad de identificarse en la aplicación. Si por el contrario, decide autenticarse, puede hacerlo accediendo a la ventana de identificación y registro, pulsando el botón de login. También se puede acceder pulsando el botón de Google (véase figura 17).

Figura nº 17. Boceto de la pantalla de bienvenida

6.2.1.2 Pantalla de login y registro

Esta pantalla se ha diseñado pensando en que el usuario introduzca sus credenciales, email y contraseña, y rápidamente pulse el botón de login. Si no se ha registrado todavía, puede hacerlo pero esta vez pulsando el botón de registro (véase figura 18).

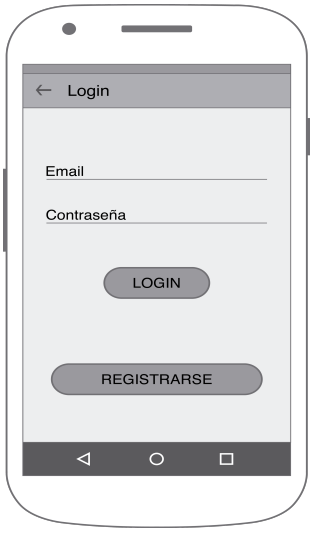

Figura nº 18. Boceto de la pantalla de login y registro

6.2.1.3 Pantalla de nueva lista

Esta pantalla se ha diseñado básicamente para que el usuario pueda crear sus listas. También actúa como punto de entrada a varias pantallas de la aplicación, gracias al desplegable del lateral izquierdo. Además, se pueden realizar varias acciones como guardar la lista, restablecerla y consultar el precio total (véase figura 19).

Figura nº 19. Boceto de la pantalla de nueva lista

6.2.1.4 Pantalla de los tablones

Esta pantalla se ha diseñado para que el usuario pueda navegar por los diferentes tablones distribuidos en categorías, en ellos se podrá elegir una lista o eliminarla. El último de los tablones está dedicado a recibir listas compartidas (véase figura 20).

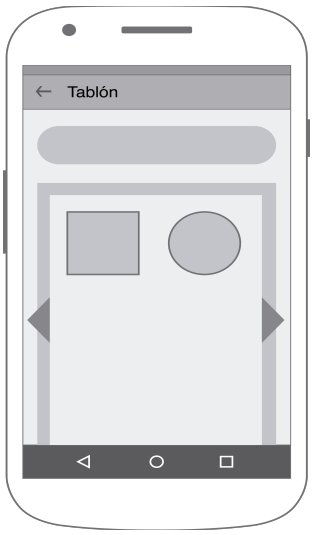

Figura nº 20. Boceto de la pantalla de los tablones

6.2.1.5 Pantalla de compartir la lista

Esta pantalla se ha diseñado para que el usuario pueda compartir y cargar sus listas. Además, se pueden realizar varias acciones como restablecer la lista y consultar el precio total (véase figura 21).

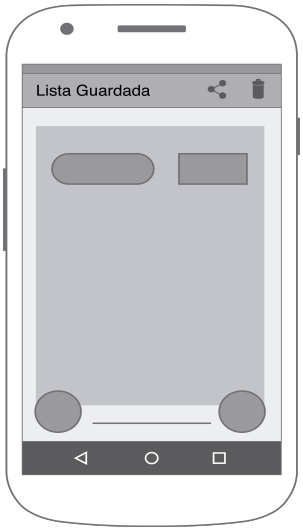

Figura nº 21. Boceto de la pantalla de compartir la lista

6.2.1.6 Pantalla de seleccionar amigos o peticiones

La pantalla de seleccionar amigos o peticiones se ha diseñado para que el usuario pueda acceder a sus amigos o a sus peticiones (véase figura 22).

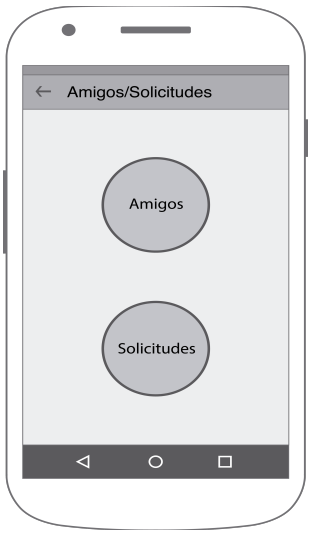

Figura nº 22. Boceto de la pantalla de seleccionar amigos o peticiones

6.2.1.7 Pantalla de los amigos

La pantalla de los amigos está formada por una lista donde se muestran los amigos que se han agregado en la aplicación. Solo se puede consultar el email de tu amigo o eliminarlo (véase figura 23).

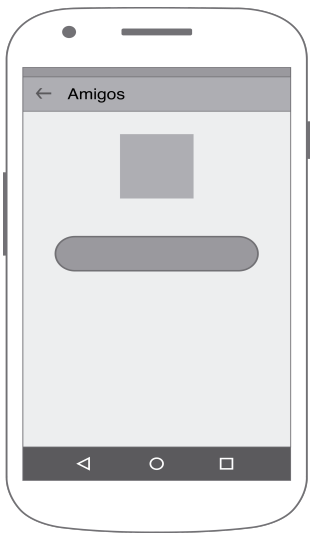

Figura nº 23. Boceto de la pantalla de los amigos

6.2.1.8 Pantalla de las solicitudes

La pantalla de las peticiones se ha diseñado para que el usuario pueda enviar, aceptar y eliminar las peticiones de amistad que otros usuarios le han enviado (véase figura 24).

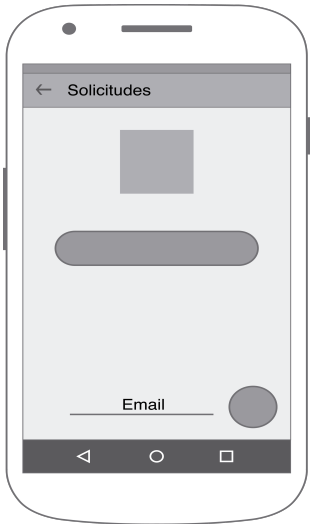

Figura nº 24. Boceto de la pantalla de las peticiones

6.2.1.9 Pantalla sobre nosotros

La pantalla de créditos se ha diseñado para que el usuario consulte los datos personales del desarrollador de la aplicación (véase figura 25).

Figura nº 25. Boceto de la pantalla sobre nosotros

6.2.1.10 Diálogo para el nombre de la lista

El diálogo para el nombre de la lista aparece siempre a la hora de crear una lista nueva. Está formado por un campo de texto donde irá el nombre y un botón para aceptar la acción (véase figura 26).

Figura nº 26. Boceto del diálogo para el nombre de la lista

6.2.1.11 Diálogo para el precio del producto

El diálogo para el precio aparece siempre a la hora de introducir un producto nuevo en la lista. Está formado por un conjunto de botones que se utilizarán para introducir el precio, además de un botón para borrarlo y un botón para asociarlo al producto y que aparezca en la lista (véase figura 27).

Figura nº 27. Boceto del diálogo para el precio del producto

6.2.1.12 Diálogo para compartir la lista

El diálogo para compartir la lista se usa para enviar la lista visualizada a un amigo agregado en la aplicación. Está formado por un campo de texto donde irá el email de la persona que va a recibir la lista y un botón para aceptar la acción (véase figura 28).

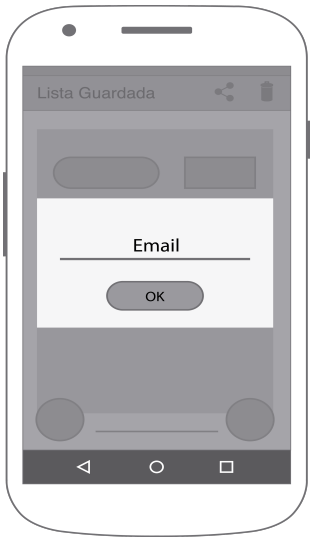

Figura nº 28. Boceto del diálogo para compartir la lista

7. Implementación

En este apartado se describen varios de los métodos que hemos utilizado para la fase de integración, tales como: el almacenamiento de ficheros, las bases de datos en SQLite y las clases Product y Share.

Para finalizar, se va a describir como conectarse a Firebase y utilizar sus servicios tanto de autenticación como de gestión de datos para cada usuario.

7.1 Almacenamiento externo de ficheros

Primeramente, antes de usar el almacenamiento externo, hay que añadir en el manifiesto de nuestra aplicación los permisos necesarios tanto de lectura como de escritura, sin ellos, los métodos que se van a describir a continuación no funcionarían.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Para poder escribir los archivos en la memoria del teléfono, es necesario un método boolean que compruebe si hay la memoria externa está presente y accesible. Si se puede escribir el archivo, devolverá verdadero; sino, devolverá falso.

```
public boolean isExternalMemoryWritable() {
    String state = Environment.getExternalStorageState();

    return (Environment.MEDIA_MOUNTED.equals(state));
}
```

Los dos métodos siguientes se encargan de comprobar que se hayan añadido los permisos anteriormente comentados. Si están incluidos en el manifiesto y existe una memoria donde escribir el archivo, ambos métodos conducirán al método encargado de almacenar el fichero, hablaremos de él a continuación.

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
    if ((grantResults.length > 0) && (PackageManager.PERMISSION_GRANTED
            == grantResults[0])) {
                switch (requestCode) {
                    case WRITING_PERMISSION:

                    // It's the method to load the list
                    saveFile2();
                    break;
                }
    }
}
```

```
public void saveFile() {
   if (isExternalMemoryWritable()) {
      int checkWriteablePermission = ContextCompat.checkSelfPermission(this,
            Manifest.permission.WRITE_EXTERNAL_STORAGE);
      if (PackageManager.PERMISSION_GRANTED == checkWriteablePermission) {

         // It's the method to save the list
         saveFile2();
      } else {
         ActivityCompat.requestPermissions(this, new String[]{
               Manifest.permission.WRITE_EXTERNAL_STORAGE},
               WRITING_PERMISSION);
      }
   }
}
```

En este método se crea una carpeta con el nombre de la categoría de la lista y se comprueba que no exista. También se crea el archivo que va a contener la lista, el método writeObject() es el encargado de escribirlo. Por último, se cierra el archivo para finalizar la escritura del archivo.

```
private void saveFile2() {
   try {

      // Creates the online folder
      File f = new File(Environment.getExternalStorageDirectory() + "/"
            + getString(R.string.app_name), category);

      //Checks if folder exists
      if (!f.exists()) {
         f.mkdirs();
      }

      // Creates the file to save the list
      FileOutputStream file = new FileOutputStream(Environment
            .getExternalStorageDirectory().getPath()
            + File.separatorChar + getString(R.string.app_name) + File.separatorChar
            + category + File.separatorChar + listName);

      // Add file to ObjectOutputStream to write
      ObjectOutputStream fos = new ObjectOutputStream(file);

      // Write the list in the file
      fos.writeObject(ListSqlHelper.getInstance(context).getProducts());

      // Closes this stream
      fos.flush();
      fos.close();

   } catch (FileNotFoundException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

7.2 Bases de datos de los usuarios en SQLite

En este apartado se van a describir cuales han sido las bases de datos y métodos que se utilizan cuando un usuario se identifica en la aplicación. En primer lugar, se crea la base de datos encargada de almacenar el email del usuario.

```
@Override
public void onCreate(SQLiteDatabase db) {

    //  SQL query to create a UserTable table with
    //  autoincremental integer primary key: id
    //  String: email
    db.execSQL("CREATE TABLE UserTable " +
            "(id INTEGER PRIMARY KEY AUTOINCREMENT, email TEXT);");

}
```

Más adelante, se usarán métodos para añadir el usuario a la base de datos cuando se identifique y para eliminarlo, cuando se cierre sesión en la aplicación. El primero, necesitará el email del usuario y lo añadirá a la base de datos, concretamente a la tabla UserTable.

```
public void addUser(String email) {

    // Get access to the database in write mode
    SQLiteDatabase database = getWritableDatabase();

    // Insert the new user into the table (autoincremental id)
    ContentValues values = new ContentValues();
    values.put("email", email);
    database.insert("UserTable", null, values);

    // Close the database helper
    database.close();
}
```

El segundo borrará de la tabla anterior todos las filas dejando la tabla vacía.

```
public void deleteAllUsers() {

    // Get access to the database in write mode
    SQLiteDatabase database = getWritableDatabase();

    // Delete from the table any entry with the given text
    database.delete("UserTable", null, null);

    // Close the database helper
    database.close();
}
```

Para finalizar este apartado se va a describir el método encargado de devolver el usuario que actualmente se encuentra identificado en la aplicación. Primero se declara una variable de tipo String, se encarga de devolver el usuario actual. También declara una variable de tipo SQLiteDatabase, para acceder a la base de datos y otra de tipo Cursor, para recorrer la tabla donde se encuentra el usuario. Si existe un usuario lo asigna a la variable y cierra la base de datos y el cursor, sino devuelve la variable sin usuario asignado y cierra también el cursor y la base de datos.

```
public String getUser() {

    // Initialize String
    String result="";

    // Get access to the database in read mode
    SQLiteDatabase database = getReadableDatabase();

    // Query the table to get the user
    Cursor cursor = database.query(
            "UserTable", new String[]{"email"}, null, null, null, null, null);

    // Check that there is any user in the database
    if(!cursor.moveToNext()){

        // Close cursor and database helper
        cursor.close();
        database.close();
    }
    else{

        // String equals with the registered user
        result = cursor.getString(0);

        // Close cursor and database helper
        cursor.close();
        database.close();
    }

    return result;
}
```

7.3 Clase Product

En este punto se va a describir la clase Product, esta clase se va a utilizar para enviar correctamente los datos a la base de datos de la nube en Firebase.

La clase esta compuesta por tres variables, dos de ellas son de tipo String y la otra, de tipo List. Además se añaden los métodos constructores con variables y vacío, ambos necesarios en la conexión a Firebase. También se incluyen los métodos get() y set() para cada variable. Para acabar se añade el método toMap(), se encarga de añadir las variables a otra de tipo HashMap, este método es necesario para añadir correctamente los datos a Firebase.

```
@IgnoreExtraProperties
public class Product {

    // String total price
    public String priceTot;

    // String friend's email
    public String emailShare;

    // List with products and prices
    public List<HashMap<String, String>> list;

    public Product() {}

    public Product(String priceTot,  List<HashMap<String, String>> list, String emailShare) {
        this.priceTot = priceTot;
        this.list = list;
        this.emailShare = emailShare;
    }

    public String getEmailShare() { return emailShare;}

    public void setEmailShare(String emailShare) { this.emailShare = emailShare;}

    public String getPriceTot() { return priceTot;}

    public void setPriceTot(String priceTot) { this.priceTot = priceTot;}

    public List<HashMap<String, String>> getList() { return list;}

    public void setList(List<HashMap<String, String>> list) { this.list = list;}

    @Exclude
    public Map<String, Object> toMap() {

        // Initialize HashMap
        HashMap<String, Object> result = new HashMap<>();

        // Add values to result
        result.put("priceTot", priceTot);
        result.put("list", list);
        result.put("emailShare", emailShare);

        return result;
    }
}
```

7.4 Clase Share

En este punto se va a describir la clase Share, esta clase se va a utilizar para enviar correctamente los datos a la base de datos de la nube en Firebase.

La clase esta compuesta solamente por una variable de tipo String. Además se añaden los métodos constructores con variables y vacío, ambos necesarios en la conexión a Firebase. También se incluyen los métodos get() y set(). Para acabar se añade el método toMap(), se encarga de añadir la variable a otra de tipo HashMap, este método es necesario para añadir correctamente los datos a Firebase.

```
@IgnoreExtraProperties
public class Share {

    // String yes or no
    public String share;

    public Share() {}

    public Share(String share) { this.share = share;}

    public String getShare() {return share;}

    public void setShare(String share) {this.share = share;}

    @Exclude
    public Map<String, Object> toShare() {

        // Initialize HashMap
        HashMap<String, Object> result = new HashMap<>();

        // Add values to result
        result.put("share", share);

        return result;
    }
}
```

7.5 Conexión a Firebase

En este apartado se van a describir los métodos necesarios para autenticarse en la aplicación mediante los servicios de Firebase, y para compartir y eliminar las listas que los usuarios van añadiendo a la base de datos cuando deciden compartir las listas con sus amigos.

7.5.1 Autenticación

En este punto se va a explicar el método que se utiliza en la pantalla de login cuando nos identificamos en la aplicación. Primeramente, hay que declarar las variables encargadas de la conexión con Firebase y de permitir la comunicación entre nuestra aplicación y el servidor de Google.

```
// Defining firebaseauth object
FirebaseAuth mAuth;
FirebaseAuth.AuthStateListener mAuthListener;
```

En el método principal onCreate() se instancian ambas variables y se declara una variable FirebaseUser para comprobar si hay un usuario autenticado en Firebase.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        …

        // Firebase auth
        mAuth = FirebaseAuth.getInstance();

        // Firebase login
        mAuthListener = new FirebaseAuth.AuthStateListener() {
        @Override
        public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {

                // Reference to the user in firebaseAuth
                FirebaseUser user = firebaseAuth.getCurrentUser();

                if (user != null) {

                    // User is signed in
                    Log.d(TAG, "onAuthStateChanged:signed_in:" + user.getUid());

                } else {

                    // User is signed out
                    Log.d(TAG, "onAuthStateChanged:signed_out");
                }
        };
}
```

No hay que olvidarse de los métodos necesarios para empezar y finalizar ambas variables cuando se comprueba el usuario en Firebase.

```
@Override
public void onStart() {
    super.onStart();
    mAuth.addAuthStateListener(mAuthListener);
}

@Override
public void onStop() {
    super.onStop();
    if (mAuthListener != null) {
        mAuth.removeAuthStateListener(mAuthListener);
    }
}
```

Este método es el encargado de comprobar si el usuario esta registrado en Firebase. Se recoge el email escrito por el usuario y se modifica el valor para que funcione correctamente la base de datos en la nube. Más adelante, si el usuario esta registrado se borra la tabla de los usuarios y se añade el nuevo valor.

```
private void loginUser() {
    // Getting email and password from edit texts
    final String email = editTextEmail.getText().toString().trim();
    String password = editTextPassword.getText().toString().trim();

    // Checking the user
    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if(task.isSuccessful()) {
                        Log.d(TAG, "signInWithEmail:onComplete:" + task.isSuccessful());
                        // Delete @xxx.com/es from the string user to use it correctly in the database
                        emailShare = email;
                        emailShare = emailShare.replace("@gmail.com", "");
                        emailShare = emailShare.replace("@hotmail.com","");
                        emailShare = emailShare.replace("@hotmail.es","");
                        emailShare = emailShare.replace("@outlook.com","");
                        emailShare = emailShare.replace("@outlook.es","");
                        emailShare = emailShare.replace("@yahoo.com","");
                        emailShare = emailShare.replace("@yahoo.es","");

                        // All users are deleted from the database
                        ListSqlHelper.getInstance(context).deleteAllUsers();
                        // Add user from database
                        ListSqlHelper.getInstance(context).addUser(emailShare);

                    }else{
                        // If sign in fails, display a message to the user. I
                        Log.w(TAG, "signInWithEmail", task.getException());
                    }
                }
            });
}
```

7.5.2 Compartir lista

En este punto se va a explicar el método que se utiliza para compartir las listas con nuestros amigos. Primeramente, hay que declarar las variables encargadas de la conexión con la base de datos en la nube y de permitir la comunicación entre nuestra aplicación y el servidor de Google.

```
// Reference to Firebase Database
DatabaseReference mDatabase = FirebaseDatabase.getInstance().getReference();
```

El método encargado de compartir la lista es shareList(), el cual necesita cuatro parámetros: el nombre de la lista, el precio total, la lista con los productos y el email del usuario. Cada vez que se añade información a la base de datos en la nube y se sabe que más adelante será consultada o modificada, es necesario crear una clave única para que de esta manera se pueda acceder a la información más fácilmente.

Para añadir la información a la base de datos es necesario crear las clases pertinentes, en nuestro caso utilizamos la clase Product formada por el precio total, la lista con los productos y el email del usuario. Se añaden a una variable de tipo Map mediante el método toMap() y en ese momento, se actualiza la base de datos con la ruta del email de nuestro amigo y la información de la lista a compartir.

```
private void shareList(String listName, String priceTot,  List<HashMap<String, String>> list,
                       String email) {

    // Get key to add firebase
    String key = mDatabase.child("users").child(emailShare).child("Online")
            .child(listName).push().getKey();

    // Creates new Product object
    Product product = new Product(priceTot, list, email);

    // Values to store in firebase
    Map<String, Object> productValues = product.toMap();

    // Creates childUpdates to put new values
    Map<String, Object> childUpdates = new HashMap<>();

    // Update firebase with new values
    childUpdates.put("/users/" + emailShare + "/" + "Online" + "/" + key + "/"
            + listName, productValues);
    mDatabase.updateChildren(childUpdates);
}
```

7.5.3 Aceptar solicitud de amistad

En este punto se va a explicar el método que se utiliza en la pantalla de solicitudes a la hora de aceptarlas. Primeramente, hay que declarar las variables encargadas de la conexión con Firebase y de permitir la comunicación entre nuestra aplicación y el servidor de Google.

```
// Reference to Firebase Database
DatabaseReference mDatabase = FirebaseDatabase.getInstance().getReference();
```

Cuando aceptamos una solicitud de amistad el primer método en ejecutarse es updateShareFire(). Este método se encarga de actualizar la rama correspondiente de amigos en cada usuario, eliminando la solicitud y añadiendo el usuario a su sección de amigos. Los métodos necesarios para añadir los datos a Firebase son los mismos que se han usado en el punto anterior, la única diferencia es que se utiliza la clase Share para actualizar el estado de los usuarios a si, ya que al enviar la petición, el estado se encuentra en no.

```
private void updateShareFirebaseRequest() {
    mDatabase.child("users").child(email).child("Friends").addListenerForSingleValueEvent(
            new ValueEventListener() {
                @Override
                public void onDataChange(DataSnapshot dataSnapshot) {
                    dataSnapshot.getKey();

                    // Initialize shareYesNo
                    shareYesNo = "yes";

                    // Create new Share object
                    Share share = new Share(shareYesNo);

                    // Values to store in firebase
                    Map<String, Object> productValues = share.toShare();

                    // Creates childUpdates to put new values
                    Map<String, Object> childUpdates = new HashMap<>();

                    // Creates friend key
                    key = mDatabase.child("users").child(email).child("Friends")
                            .child(emailList).push().getKey();

                    // Update firebase with new values
                    childUpdates.put("/users/" + email + "/Friends/" + key + "/"
                            + emailList, productValues);
                    mDatabase.updateChildren(childUpdates);
                }

                @Override
                public void onCancelled(DatabaseError databaseError) {
                    Log.w(TAG, "getUser:onCancelled", databaseError.toException());
                }
            });
}
```

En segundo lugar, se ejecuta el método updateShareFirebaseRequest2(). Este método tiene la misma función que el anterior, si antes se ha usado para actualizar el estado de los amigos agregados, ahora se encargará de actualizar los datos de la persona que ha enviado la petición de amistad.

```
private void updateShareFirebaseRequest2() {

mDatabase.child("users").child(emailList).child("Friends").addListenerForSingleValueEvent(
        new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                dataSnapshot.getKey();

                // Initialize shareYesNo
                shareYesNo = "yes";

                // Create new Share object
                Share share = new Share(shareYesNo);

                // Values to store in firebase
                Map<String, Object> productValues = share.toShare();

                // Creates childUpdates to put new values
                Map<String, Object> childUpdates = new HashMap<>();

                // Creates friend key
                key = mDatabase.child("users").child(emailList).child("Friends").
                        child(email).push().getKey();

                // Update firebase with new values
                childUpdates.put("/users/" + emailList + "/Friends/" + key + "/"
                        + email, productValues);
                mDatabase.updateChildren(childUpdates);
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                Log.w(TAG, "getUser:onCancelled", databaseError.toException());
            }
        });
}
```

Por último, se ejecuta el método deleteEmailFirebaseRequest(). Este método tiene la función de eliminar la petición de amistad de la bandeja de peticiones debido a que ya ha sido aceptada. Su uso será similar a lo que se ha visto anteriormente, accederá a la rama del usuario e irá recorriéndola hasta que coincidan los emails. Una vez se ha encontrado, se elimina la solicitud y automáticamente desaparecerá de nuestra bandeja ya que se actualiza la actividad.

```
private void deleteEmailFirebaseRequest() {
    mDatabase.child("users").child(email).child("Requests").addListenerForSingleValueEvent(
            new ValueEventListener() {
                @Override
                public void onDataChange(DataSnapshot dataSnapshot) {
                    dataSnapshot.getKey();

                    // Creates childUpdates to put new values
                    Map<String, Object> childUpdates = new HashMap<>();

                    // Loop to obtein values and update the list
                    for (DataSnapshot snapshot : dataSnapshot.getChildren()){
                        key = snapshot.getKey();

                        // Friend email
                        emailPetition = snapshot.getChildren().iterator().next().getKey();

                        // If email list equals friend email
                        if(emailPetition.equals(emailList)){

                            // Get user key
                            key = snapshot.getKey();

                            // Update firebase with new values
                            childUpdates.put("/users/" + email + "/Requests/" + key + "/" +
                                    emailPetition, null);
                            mDatabase.updateChildren(childUpdates);

                            // Refresh activity
                            finish();
                            startActivity(getIntent());
                        }
                    }
                }

                @Override
                public void onCancelled(DatabaseError databaseError) {
                    Log.w(TAG, "getUser:onCancelled", databaseError.toException());
                }
            });
}
```

8. Evaluación

Una vez terminada la implementación se han realizado distintas pruebas para garantizar el correcto funcionamiento, probando cada funcionalidad una a una y corrigiendo aquellas en las que se presentaba algún error, de forma que se ha conseguido depurar la aplicación lo máximo posible con el fin de evitar fallos a la hora de realizar uso del proyecto desarrollado.

Una de las pruebas más importantes es probar la aplicación en funcionamiento en distintos dispositivos, para ello se ha empleado el simulador incorporado en el propio SDK de Android. Mediante el simulador se pueden configurar diversos dispositivos virtuales, de forma que simulen las capacidades técnicas de un dispositivo real. Se han configurado varios dispositivos virtuales para probar la aplicación funcionando en distintos tamaños de pantalla.

A continuación se muestran distintas imágenes de la aplicación desarrollada funcionando en la versión 5.0 de Android con API 22, concretamente en el smartphone Moto G de la marca Motorola (véase la figura 29, 30, 31, 32, 33, 34 y 35).

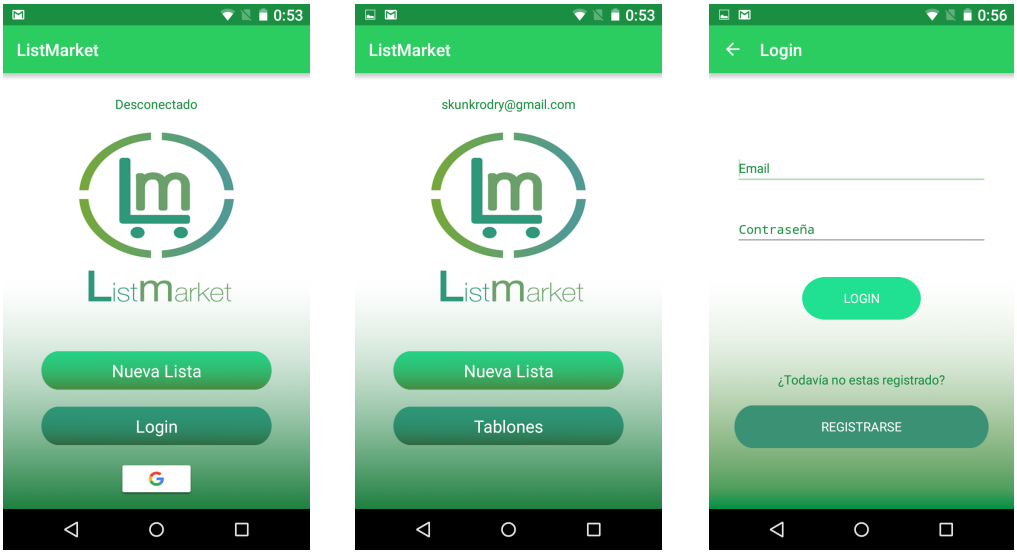

Figura nº 29. Pantalla de bienvenida y de login

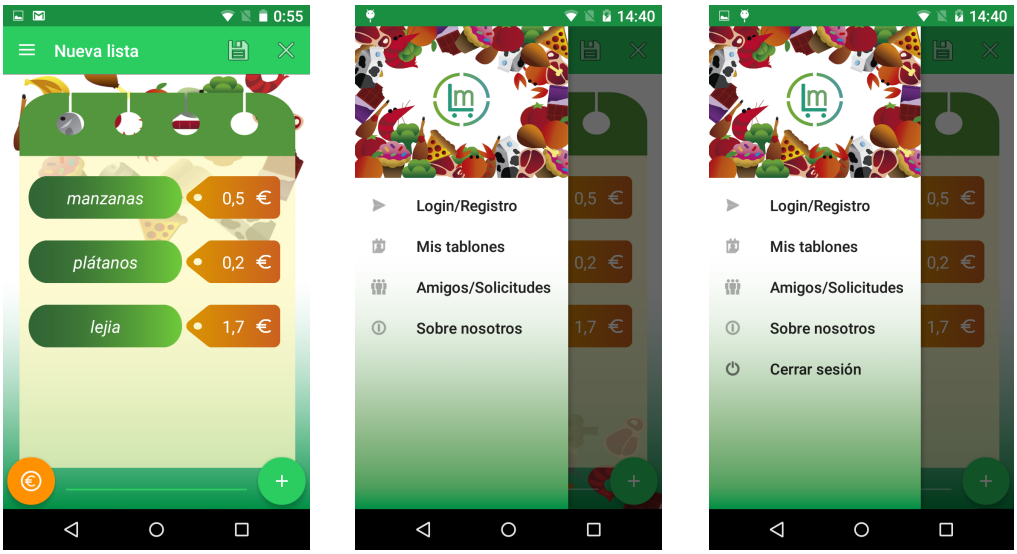

Figura nº 30. Pantalla de nueva lista

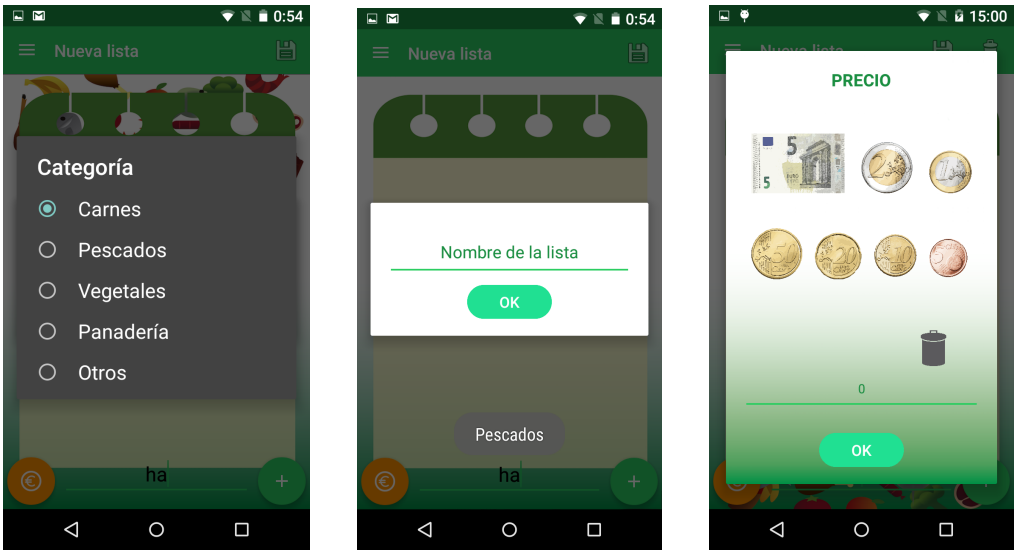

Figura nº 31. Diálogos de la pantalla de nueva lista

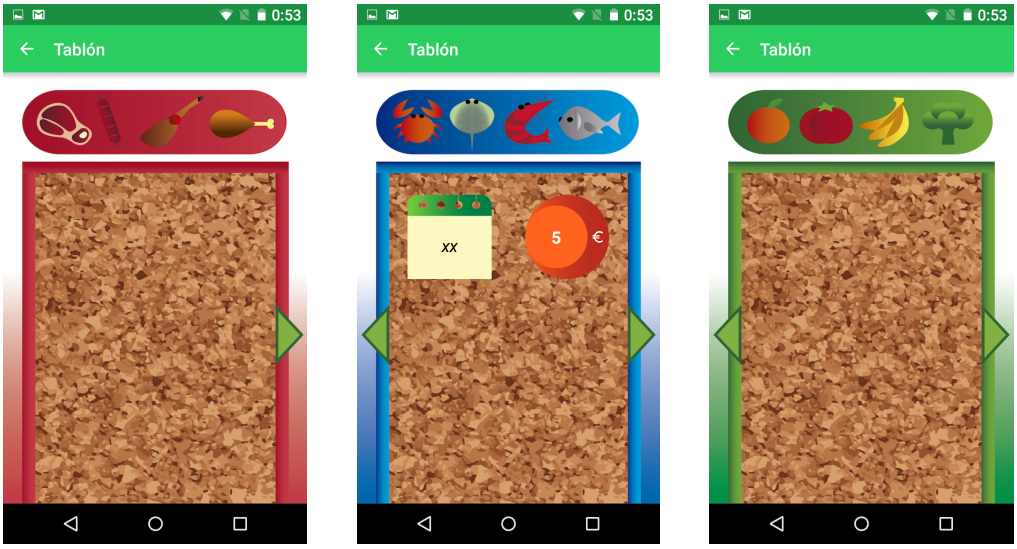

Figura nº 32. Pantalla de los tablones

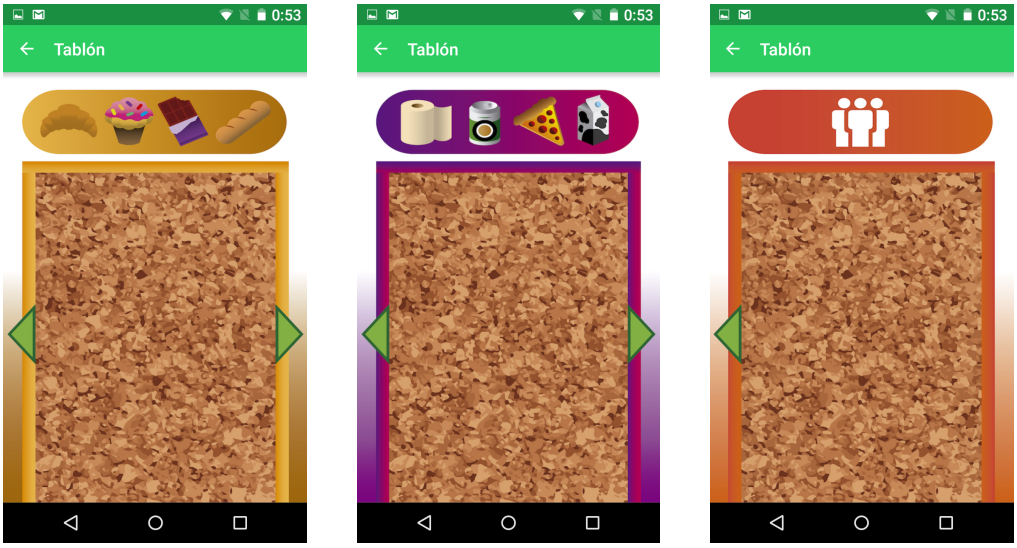

Figura nº 33. Pantalla de los tablones

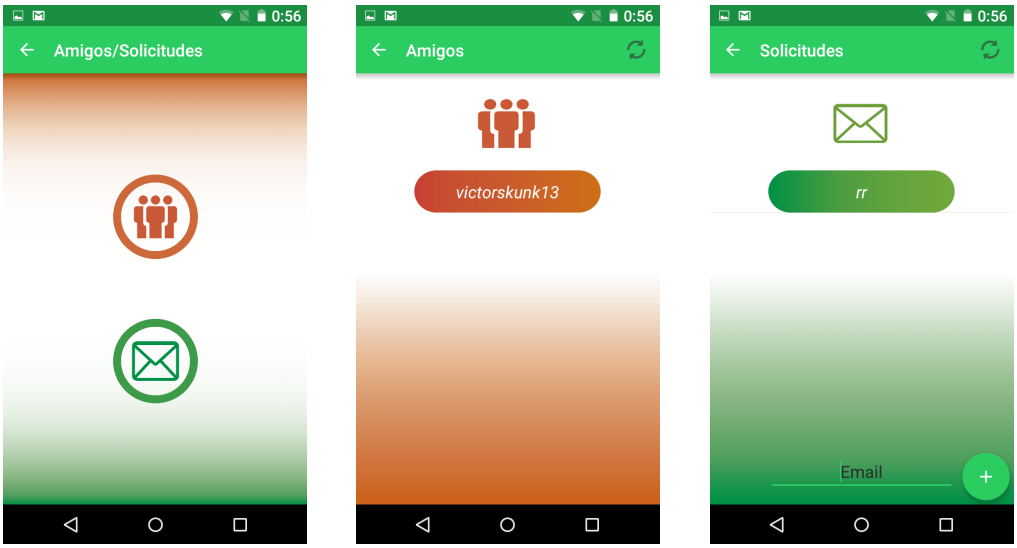

Figura nº 34. Pantalla de los amigos y las solicitudes

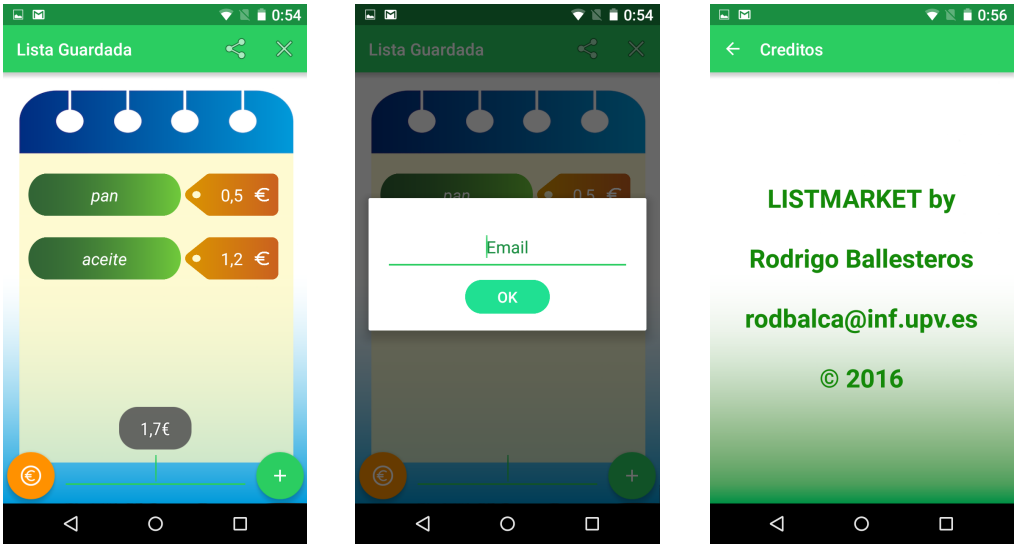

Figura nº 35. Pantalla de compartir la lista, su diálogo y la pantalla sobre nosotros

Para concluir este apartado se van a detallar los resultados de las pruebas realizadas. La aplicación ocupa 56,9 MB, un tamaño que hoy en día no resulta ningún problema para los dispositivos móviles actuales debido al gran avance de las tecnologías. Por otro lado, se han validado y verificado varios atributos de calidad del sistema, tales como la estabilidad, la escalabilidad, la fiabilidad y el uso de recursos. También se han comprobado los tiempos de carga a la hora de utilizar los servicios de internet y se han obtenido varios tiempos. Cuando nos autenticamos mediante la opción de Google se tarda, aproximadamente, de 2 a 5 segundos en acceder. Sin embargo, tanto al cargar las listas en el tablón online como las solicitudes de amistad y amigos, se reducen un poco los tiempos y pasan a ser de 1 a 3 segundos.

9. Conclusiones

El último apartado a tratar es el relacionado con las conclusiones del proyecto. Cabe mencionar que el proyecto, a lo largo de todo su desarrollo, ha logrado cumplir con los objetivos y motivaciones que se habían marcado al iniciar el proyecto.

He podido realizar el desarrollo de una aplicación destinada a dispositivos móviles desde cero hasta lograr una aplicación que funciona correctamente y sin errores. De esta forma, he adquirido más conocimientos en Android así como en Firebase. Además, se han empleado conocimientos ya aprendidos como el uso de bases de datos en SQLite y el almacenamiento externo de ficheros. Por otro lado, el hecho de tener que desarrollar la aplicación de forma individual, sin un código inicial o referencia ha hecho que consultar diferentes fuentes tales como libros o páginas web haya cobrado gran relevancia a la hora de abordar y solventar diversos problemas surgidos a lo largo del desarrollo. Sin embargo, el hecho de haber cursado la asignatura de Soluciones Informáticas para Dispositivos Móviles ha hecho partir con una gran base de conocimientos tanto en Android como en el manejo de bases de datos.

Una de las etapas más costosas fue el inicio del desarrollo y la creación de las interfaces, tanto a nivel de diseño como de implementación. Al tener un conocimiento bastante amplio del lenguaje Java, no ha habido graves problemas que no se hayan podido solventar consultando información en la web, contando también. Por otro lado, el diseño ha sido una de las etapas más importantes ya que todas las interfaces siguen una temática muy parecida y se ha intentado cuidar al máximo los detalles en todas las pantallas. También hay que mencionar la alta optimización a la hora de emplear los dispositivos virtuales gracias a las continuas actualizaciones de Android Studio, lo que ha hecho que las pruebas hayan sido un proceso rápido y no tan tedioso como habría resultado hace unos años.

También hay que añadir que el proceso más tedioso y que más tiempo he tenido que dedicar ha sido a la plataforma de Firebase, al ser unos servicios que no estaban tan destacados a la hora de construir un backend flexible y seguro, frente a otras tecnologías como pueden ser PHP y MySQL. Sin duda, hay que agradecerle a Google la inmensa cantidad de servicios que nos proporciona cuando hay que implementar un servidor para la aplicación en desarrollo. No solo incluyen clases, métodos y servicios para usar en Android; sino que también proporcionan sus servidores para contener a todos los usuarios que se registren con sus respectivos datos generados al usar la aplicación.

Como conclusión mencionar que el proyecto me ha servido para aprender aún más el uso de tecnologías que tienen mucho futuro como es la programación para dispositivos Android, así como para poner en práctica conocimientos adquiridos durante la carrera, como por ejemplo la especificación de requisitos en un proyecto software y la construcción de casos de uso que asientan los requisitos funcionales.

10. Bibliografía

- *Marcombo, **El Gran Libro de Android**, 2ª Edición, 2011*
- *Bruce Eckel, **Thinking in Java**, Edición Online, 1998*
- *Android developers: **https://developer.android.com** (accedido en junio de 2016)*
- *Firebase: **https://firebase.google.com** (accedido en julio de 2016)*
- *Simplified Coding: **https://www.simplifiedcoding.net** (accedido en agosto de 2016)*
- *Hermosa Programación: **https://www.hermosaprogramacion.com** (accedido en agosto de 2016)*
- *Tutorials Point: **https://www.tutorialspoint.com** (accedido en agosto de 2016)*
- *Android Hive: **https://www.androidhive.info** (accedido en junio de 2016)*
- *Site Point: **https://www.sitepoint.com** (accedido en agosto de 2016)*
- *El Androide Libre: **https://www.elandroidelibre.com** (accedido en julio de 2016)*

Anexo 1. Interfaces gráficas de usuario en formato XML

Pantalla de bienvenida: activity_first.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas - UPV,
  Development of apps for mobile devices.
  -->

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".activities.FirstActivity"
    android:orientation="vertical"
    android:weightSum="1"
    android:background="@drawable/first">

    <TextView
        android:id="@+id/status"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/signed_out"
        android:textColor="@color/primary"
        android:textSize="@dimen/status_text_size"
        android:layout_gravity="center_horizontal"
        android:paddingTop="@dimen/detail2"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <TextView
        android:id="@+id/detail"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fadeScrollbars="true"
        android:gravity="center"
        android:padding="@dimen/detail2"
        android:scrollbars="vertical"
        android:textColor="@android:color/white"
        android:textSize="@dimen/detail"
        android:layout_gravity="center_horizontal"
        android:textIsSelectable="false"
        android:maxLines="5"
        android:layout_centerHorizontal="true" />
```

```
<Button
    android:layout_width="@dimen/button_h"
    android:layout_height="@dimen/button_v"
    android:id="@+id/buttonFast"
    android:text="@string/new_list"
    android:onClick="buttonsList"
    android:background="@drawable/bfast"
    android:layout_row="10"
    android:layout_column="0"
    android:textSize="@dimen/size_text_size"
    android:textColor="@android:color/white"
    android:layout_gravity="center_horizontal"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="@dimen/space" />

<Button
    android:layout_width="@dimen/button_h"
    android:layout_height="@dimen/button_v"
    android:text="@string/login"
    android:onClick="buttonsList"
    android:layout_row="12"
    android:layout_column="0"
    android:textColor="@android:color/white"
    android:textSize="@dimen/size_text_size"
    android:background="@drawable/bpersonal"
    android:layout_gravity="center_horizontal"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:id="@+id/buttonLogin"
    android:layout_centerHorizontal="true"
    android:layout_alignParentTop="true"
    android:layout_marginTop="@dimen/space2" />

<RelativeLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="@dimen/space3"
    android:id="@+id/relativeLayout"
    android:layout_centerHorizontal="true"
    android:layout_alignParentTop="true">

    <com.google.android.gms.common.SignInButton
        android:id="@+id/sign_in_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="visible"
        tools:visibility="gone"
        android:layout_centerHorizontal="true"
        android:layout_alignWithParentIfMissing="true" />

    <LinearLayout
        android:id="@+id/sign_out_and_disconnect"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
```

```
        android:orientation="horizontal"
        android:visibility="gone"
        tools:visibility="visible">

        <Button
            android:id="@+id/sign_out_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/sign_out"
            android:theme="@style/ThemeOverlay.MyDarkButton" />

        <Button
            android:id="@+id/disconnect_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/disconnect"
            android:theme="@style/ThemeOverlay.MyDarkButton" />
    </LinearLayout>

  </RelativeLayout>

</RelativeLayout>
```

Pantalla de login y registro: activity_loginregister.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas - UPV,
  Development of apps for mobile devices.
  -->

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".activities.LoginRegisterActivity"
    android:background="@drawable/degraded">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:id="@+id/linearLayout2"
        android:paddingTop="@dimen/edit2">
```

```
<EditText
    android:id="@+id/editTextEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/edit"
    android:hint="@string/enterM"
    android:inputType="textEmailAddress"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textSize="@dimen/status_text_size"
    android:textColorHint="@color/primary"
    android:textColor="@color/primary" />

<EditText
    android:id="@+id/editTextPassword"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/edit"
    android:hint="@string/enterPass"
    android:inputType="textPassword"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textSize="@dimen/status_text_size"
    android:textColor="@color/primary"
    android:textColorHint="@color/primary" />

<Button
    android:id="@+id/buttonSignIn"
    android:layout_width="@dimen/edit8"
    android:layout_height="@dimen/edit7"
    android:text="@string/access"
    android:layout_gravity="center_horizontal"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:background="@drawable/blogin"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textColor="@android:color/white"
    android:textSize="@dimen/status_text_size"
    android:layout_margin="@dimen/edit3" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="@string/tvReg"
    android:id="@+id/textView2"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true"
    android:layout_gravity="center_horizontal"
    android:textColor="@color/primary"
    android:textSize="@dimen/status_text_size"
    android:layout_marginTop="@dimen/edit5" />

<Button
    android:id="@+id/buttonRegister"
    android:layout_width="@dimen/edit6"
    android:layout_height="@dimen/edit7"
    android:text="@string/register"
    android:layout_gravity="center_horizontal"
```

```
        android:layout_below="@+id/textView2"
        android:layout_alignLeft="@+id/buttonSignIn"
        android:layout_centerHorizontal="true"
        android:background="@drawable/bregister"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="@android:color/white"
        android:textSize="@dimen/status_text_size"
        android:layout_marginTop="@dimen/edit3" />

  </LinearLayout>

</RelativeLayout>
```

Pantalla de nueva lista: activity_fastlist.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
 ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas - UPV,
 Development of apps for mobile devices.
 -->

<!--
 DrawerLayout enables to pull a drawer from an edge of the screen.
 It must declare from which edge the drawer could be pulled (start/end).
 -->
<android.support.v4.widget.DrawerLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/drawerLayout"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".activities.FastListActivity"
   tools:openDrawer="start">

   <!--The first component of a DrawerLayout is the container that will
   display elements on the screen-->
   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical"
      android:id="@+id/backgroundList">


      <!--ToolBar that will replace the default ActionBar.-->
      <android.support.v7.widget.Toolbar
         android:id="@+id/toolBar"
         android:layout_width="match_parent"
         android:layout_height="?attr/actionBarSize"
         android:background="@color/toolbar"
         android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
         app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

```
<!--Container that will display the rest of elements on the screen-->
<FrameLayout
    android:id="@+id/frameLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.listmarket.activities.FastListActivity"
    android:orientation="vertical">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        xmlns:android="http://schemas.android.com/apk/res/android">

        <ImageView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/imageView" />

        <android.support.design.widget.FloatingActionButton
            android:id="@+id/fabA"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/ic_add"
            android:theme="@style/AppThemeFloatButton2"
            android:layout_alignParentBottom="true"
            android:layout_alignParentRight="true"
            android:layout_alignParentEnd="true"
            android:layout_marginRight="@dimen/list6"
            android:layout_marginBottom="@dimen/list6" />

        <android.support.design.widget.FloatingActionButton
            android:id="@+id/fabPrice"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/ic_euro"
            android:theme="@style/AppThemeFloatButton"
            android:layout_marginBottom="@dimen/list6"
            android:importantForAccessibility="yes"
            android:focusable="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentBottom="true"
            android:layout_marginLeft="@dimen/list6" />

        <!--Container that will display the rest of elements on the screen-->

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:weightSum="1"
            android:layout_alignParentTop="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:id="@+id/linearLayout2"
            android:layout_above="@+id/editTextList"
            android:layout_alignParentRight="true"
            android:layout_alignParentEnd="true">
```

```
            <ListView
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:id="@+id/listFast"
                android:layout_marginTop="@dimen/fab6"
                android:paddingBottom="@dimen/rowList" />

        </LinearLayout>

        <EditText
            android:layout_width="@dimen/fab"
            android:layout_height="wrap_content"
            android:id="@+id/editTextList"
            android:textColor="@color/black"
            android:layout_gravity="center_vertical"
            android:layout_alignParentBottom="true"
            android:layout_marginBottom="@dimen/fab4"
            android:textAlignment="center"
            android:layout_centerHorizontal="true" />

      </RelativeLayout>

    </FrameLayout>

  </LinearLayout>

  <!--The second element of a DrawerLayout is the container displaying
  the options in the drawer-->
  <android.support.design.widget.NavigationView
    android:id="@+id/navView"
    android:layout_width="@dimen/navDraw"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:theme="@style/AppTheme"
    android:fitsSystemWindows="true"
    app:headerLayout="@layout/header_navigation_drawer"
    app:menu="@menu/menu_navigation_drawer_fastlist"
    android:background="@drawable/degraded" />

</android.support.v4.widget.DrawerLayout>
```

Pantalla de los tablones: activity_plank.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas - UPV,
  Development of apps for mobile devices.
  -->

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!--Component used to display pages and enable the lateral navigation (swipe) between
them-->
    <android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </android.support.v4.view.ViewPager>

</LinearLayout>
```

Pantalla de compartir lista: activity_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas - UPV,
  Development of apps for mobile devices.
  -->
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/BackgroundListTxt">

    <!--Container that will display the rest of elements on the screen-->

    <RelativeLayout android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        xmlns:android="http://schemas.android.com/apk/res/android">

        <android.support.design.widget.FloatingActionButton
            android:id="@+id/fabTxt"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/ic_add"
            android:theme="@style/AppThemeFloatButton2"
            android:layout_alignParentBottom="true"
```

```
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_marginBottom="@dimen/list6"
        android:layout_marginRight="@dimen/list6" />

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fabPriceTxt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_euro"
        android:theme="@style/AppThemeFloatButton"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="false"
        android:layout_alignParentEnd="false"
        android:layout_marginBottom="@dimen/list6"
        android:layout_alignParentLeft="false"
        android:layout_marginLeft="@dimen/list6" />

    <!--Container that will display the rest of elements on the screen-->

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:weightSum="1"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:id="@+id/linearLayout2"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_above="@+id/fabTxt">

        <ListView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/listTxt"
            android:layout_marginTop="95dp"
            android:layout_gravity="center_horizontal" />

    </LinearLayout>

    <EditText
        android:layout_width="@dimen/fab"
        android:layout_height="wrap_content"
        android:id="@+id/editTextListTxt"
        android:textColor="@color/black"
        android:layout_gravity="center_vertical"
        android:textAlignment="center"
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="@dimen/fab4" />

  </RelativeLayout>

</FrameLayout>
```

Pantalla de seleccionar amigos o peticiones: activity_social.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas - UPV,
  Development of apps for mobile devices.
  -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/socialdeg">

    <Button
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:id="@+id/buttonFriends"
        android:background="@drawable/bfriends"
        android:layout_centerHorizontal="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="@dimen/social" />

    <Button
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:id="@+id/buttonRequest"
        android:layout_centerHorizontal="true"
        android:layout_alignParentTop="false"
        android:background="@drawable/brequests"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="@dimen/social" />

</RelativeLayout>
```

Pantalla de los amigos: activity_friends.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas -
  UPV, Development of apps for mobile devices.
  -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/friends"
    android:theme="@style/AppThemeFloatButton">

    <ListView
        android:layout_width="match_parent"
```

```
        android:layout_height="match_parent"
        android:id="@+id/lvFriends"
        android:layout_marginTop="@dimen/friends" />

    <ProgressBar
        style="?android:attr/progressBarStyleLarge"
        android:id="@+id/progressBarFriends"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:visibility="invisible"
        android:textAlignment="gravity" />

</RelativeLayout>
```

Pantalla de las peticiones: activity_request.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas -
  UPV, Development of apps for mobile devices.
  -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/requests">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/lvRequests"
        android:layout_above="@+id/fabR"
        android:layout_marginTop="@dimen/friends" />

    <ProgressBar
        style="?android:attr/progressBarStyleLarge"
        android:id="@+id/progressBarRequests"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:visibility="invisible"
        android:textAlignment="gravity" />

    <EditText
        android:layout_width="@dimen/fab"
        android:layout_height="wrap_content"
        android:id="@+id/etRequests"
```

```
        android:textColor="@color/black"
        android:layout_gravity="center_vertical"
        android:textAlignment="center"
        android:layout_marginBottom="@dimen/fab4"
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true"
        android:hint="@string/email"
        android:textSize="@dimen/row_product_text_size"
        android:textColorHint="@color/text"
        android:theme="@style/AppThemeFloatButton2" />

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fabR"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_add"
        android:theme="@style/AppThemeFloatButton2"
        android:layout_marginRight="@dimen/list6"
        android:layout_alignParentBottom="true"
        android:layout_alignParentEnd="true"
        android:layout_marginBottom="@dimen/list6" />

</RelativeLayout>
```

Pantalla sobre nosotros : activity_credits.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas -
  UPV, Development of apps for mobile devices.
  -->
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.listmarket.activities.CreditsActivity"
    android:background="@drawable/degraded">

    <!--Display the credits centred on the screen-->
    <TextView
        android:id="@+id/tvABout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:gravity="center"
        android:text="@string/about_text"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textColor="@color/colorCredits"
```

```
        android:textSize="@dimen/activity_about_text_size"
        android:textStyle="bold" />

</ScrollView>
```

Diálogo para el nombre de la lista: dialog_listname.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas - UPV,
  Development of apps for mobile devices.
  -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/white"
    android:padding="@dimen/dialog_body">


    <EditText
        android:id="@+id/listNameEdit"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="@dimen/normal_padding"
        android:inputType="textPersonName"
        android:padding="@dimen/edit_text_padding"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:hint="@string/listName"
        android:gravity="center"
        android:textColorHint="@color/primary"
        android:textColor="@color/primary"
        android:textSize="@dimen/dialog_text_size" />

    <Button
        android:id="@+id/buttonDialogOk"
        android:layout_width="@dimen/dialog"
        android:layout_height="@dimen/dialog3"
        android:textColor="@android:color/white"
        android:text="@string/button_list"
        android:layout_below="@+id/listNameEdit"
        android:layout_centerHorizontal="true"
        android:background="@drawable/blogin"
        android:layout_marginTop="@dimen/dialog4" />

</RelativeLayout>
```

Diálogo para el precio del producto: dialog_price.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas - UPV,
  Development of apps for mobile devices.
  -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/degraded"
    android:padding="@dimen/dialog_body"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/price_dialog"
        android:id="@+id/textView"
        android:layout_gravity="center_horizontal"
        android:textSize="@dimen/size_text_size"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:layout_marginBottom="@dimen/dialogp3"
        android:textColor="@color/primary"
        android:textStyle="bold|normal" />

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="71dp"
        android:layout_marginTop="@dimen/dialogp4">

        <Button
            android:id="@+id/button500"
            android:layout_width="@dimen/dialogp24"
            android:layout_height="@dimen/dialogp2"
            android:textColor="@android:color/white"
            android:background="@drawable/one_e"
            android:layout_below="@+id/button20"
            android:layout_alignParentRight="true"
            android:layout_alignParentEnd="true"
            android:layout_marginLeft="@dimen/dialogp21" />

        <Button
            android:id="@+id/button200"
            android:layout_width="@dimen/dialogp20"
            android:layout_height="@dimen/dialogp20"
            android:textColor="@android:color/white"
            android:background="@drawable/three_e"
            android:layout_alignTop="@+id/button500"
            android:layout_toRightOf="@+id/button20"
            android:layout_toEndOf="@+id/button20"
            android:layout_marginTop="@dimen/dialogp23"
            android:layout_marginLeft="@dimen/dialogp22" />

        <Button
```

```
        android:id="@+id/button100"
        android:layout_width="@dimen/dialogp13"
        android:layout_height="@dimen/dialogp13"
        android:textColor="@android:color/white"
        android:background="@drawable/two_e"
        android:layout_alignBottom="@+id/button50"
        android:layout_toLeftOf="@+id/button20"
        android:layout_toStartOf="@+id/button20"
        android:layout_marginLeft="@dimen/dialogp22"
        android:layout_marginTop="@dimen/dialogp21" />

</LinearLayout>

<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/dialogp4">

    <Button
        android:id="@+id/button50"
        android:layout_width="@dimen/dialogp20"
        android:layout_height="@dimen/dialogp20"
        android:textColor="@android:color/white"
        android:background="@drawable/four_e"
        android:layout_alignTop="@+id/button200"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="@dimen/dialogp21"
        android:layout_marginTop="@dimen/dialogp22" />

    <Button
        android:id="@+id/button20"
        android:layout_width="@dimen/dialogp17"
        android:layout_height="@dimen/dialogp17"
        android:textColor="@android:color/white"
        android:background="@drawable/five_e"
        android:layout_alignTop="@+id/button10"
        android:layout_alignRight="@+id/buttonOkPrice"
        android:layout_alignEnd="@+id/buttonOkPrice"
        android:layout_marginLeft="@dimen/dialogp18"
        android:layout_marginTop="@dimen/dialogp19" />

    <Button
        android:id="@+id/button10"
        android:layout_width="@dimen/dialogp13"
        android:layout_height="@dimen/dialogp13"
        android:textColor="@android:color/white"
        android:background="@drawable/six_e"
        android:layout_alignParentTop="true"
        android:layout_alignRight="@+id/button500"
        android:layout_alignEnd="@+id/button500"
        android:layout_marginLeft="@dimen/dialogp15"
        android:layout_marginTop="@dimen/dialogp16"
```

```
        android:layout_gravity="top" />

    <Button
        android:id="@+id/button5"
        android:layout_width="@dimen/dialogp10"
        android:layout_height="@dimen/dialogp10"
        android:textColor="@android:color/white"
        android:background="@drawable/seven_e"
        android:layout_above="@+id/button500"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="@dimen/dialogp11"
        android:layout_marginTop="@dimen/dialogp12" />

</LinearLayout>

<Button
    android:layout_width="@dimen/dialogp6"
    android:layout_height="@dimen/dialogp7"
    android:id="@+id/buttonDelet"
    android:layout_alignTop="@+id/listPrice"
    android:layout_alignRight="@+id/listPrice"
    android:layout_alignEnd="@+id/listPrice"
    android:layout_gravity="right"
    android:layout_marginRight="@dimen/dialogp9"
    android:layout_marginTop="@dimen/dialogp8"
    android:theme="@style/AppTheme"
    android:background="@drawable/ic_action_delete" />

<EditText
    android:id="@+id/listPrice"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textPersonName"
    android:padding="@dimen/edit_text_padding"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:hint="@string/price"
    android:gravity="center"
    android:textColorHint="@color/primary"
    android:textColor="@color/primary"
    android:focusableInTouchMode="true"
    android:layout_above="@+id/buttonOkPrice"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="@dimen/dialogp5"
    android:textSize="@dimen/status_text_size" />

<Button
    android:id="@+id/buttonOkPrice"
    android:layout_width="@dimen/dialog"
    android:layout_height="@dimen/dialog3"
    android:textColor="@android:color/white"
    android:text="@string/button_list"
    android:background="@drawable/blogin"
```

```
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:layout_gravity="center_horizontal" />

</LinearLayout>
```

Diálogo para el email: dialog_email.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (c) 2016. Rodrigo Ballesteros Cabañas - UPV,
  Development of apps for mobile devices.
  -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/white"
    android:padding="@dimen/dialog_body">

    <EditText
        android:id="@+id/listEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="@dimen/normal_padding"
        android:inputType="textPersonName|textEmailAddress"
        android:padding="@dimen/edit_text_padding"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:hint="@string/email"
        android:gravity="center"
        android:textColorHint="@color/primary"
        android:textColor="@color/primary"
        android:textSize="@dimen/dialog_text_size" />

    <Button
        android:id="@+id/butDialogOk"
        android:layout_width="@dimen/dialog"
        android:layout_height="@dimen/dialog3"
        android:textColor="@android:color/white"
        android:text="@string/button_list"
        android:layout_below="@+id/listEmail"
        android:layout_centerHorizontal="true"
        android:background="@drawable/blogin"
        android:layout_marginTop="@dimen/dialog4" />

</RelativeLayout>
```