UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA MECÁNICA

"DESARROLLO DE UNA APLICACIÓN ANDROID PARA LA GESTIÓN EFICIENTE DE LOS ALIMENTOS EN EL HOGAR"

TRABAJO FIN DE GRADO

Febrero - 2021

AUTOR: Jaime Miralles Zaragoza

DIRECTOR/ES: César Fernández Peris

ÍNDICE

TABLA DE ILUSTRACIONES

1. AGRADECIMIENTOS

Para comenzar me gustaría mencionar a mi tutor del TFG, César Fernández Peris. Gracias por guiarme durante la realización de mi Trabajo de Fin de Grado y por la facilidad de comunicación que he recibido.

También agradecer a la Universidad Miguel Hernández de Elche por toda la ayuda recibida tanto dentro del campus como fuera de la universidad.

En especial, a mi familia por el increíble apoyo recibido y la paciencia mostrada en momentos difíciles.

Para finalizar, mencionar a los amigos que he ganado y perdido durante esta experiencia que ha sido mi vida universitaria, por todo lo que me han aportado, enseñado y compartido conmigo.

2. RESUMEN

En el presente proyecto, “Desarrollo de una aplicación Android para la gestión eficiente de los alimentos en el hogar”, se explicará de manera detallada el funcionamiento de la aplicación, cuyo nombre es FridgeTracker, así como el código utilizado para montar la app en Android Studio.

Con este Trabajo de Fin de Grado se pretende adquirir los conocimientos necesarios para desarrollar una aplicación para dispositivos móviles desde cero. Consiguiendo suficientes conocimientos en JAVA y Android Studio para iniciarse en la programación y asentar unas bases robustas para entrar en el mundo laboral.

3. INTRODUCCIÓN

El presente documento recoge el proceso de desarrollo realizado por el alumno Jaime Miralles Zaragoza para completar el Trabajo de Fin de Grado (TFG), concluyendo así el Grado en Ingeniería Mecánica en la Universidad Miguel Hernández de Elche.

FridgeTracker es una aplicación para dispositivos Android desarrollada en Android Studio utilizando JAVA como lenguaje de programación. La aplicación tiene como objetivo facilitar la gestión de alimentos en el hogar para cada usuario, permitiendo contabilizar los productos adquiridos, así como su fecha de caducidad para evitar el malgasto innecesario de alimentos.

FridgeTracker funciona además como una lista de la compra, permitiendo así tener a mano los productos ya adquiridos y organizar eficientemente la adquisición y mantenimiento de los alimentos.

3.1 MOTIVACIÓN Y OBJETIVOS

Durante mi carrera universitaria he estado trabajando en diversos proyectos personales que incluían el diseño de aplicaciones móvil y páginas web desde el punto de vista de la interfaz del usuario. Por ello me resultó interesante aprender como implementar un diseño y reflejarlo en un código para una aplicación.

De esta manera podría trabajar en una app desde cero, aprendiendo de manera robusta las bases de la programación y el lenguaje JAVA con el objetivo de ampliar mis conocimientos para que al finalizar contara con un currículum más versátil que me permita optar a un mayor número de puestos de trabajo.

La idea detrás de la aplicación desarrollada en este TFG surgió debido a un concurso de diseño al que me presenté. La temática era la de desarrollar una

idea que ayudara a los hogares a reducir el número de alimentos que se desperdician a diario. Por ello planteé una aplicación cuya función principal es la de gestionar y organizar los alimentos que cada hogar almacena en la nevera, congelador y despensa, así como la cantidad de cada producto adquirido y su fecha de caducidad. Además, la app cuenta con una lista de la compra para poder controlar todos los alimentos del usuario desde el móvil. Todo en conjunto pretende eliminar la compra de productos innecesarios, o que ya se han adquirido, pero están olvidados, así como tirar alimentos por que se haya pasado la fecha de caducidad.

3.2 CONTEXTO DE DESARROLLO

La aplicación desarrollada en este Trabajo de Fin de Grado ha sido creada para el sistema Android. Este sistema operativo estaba basado en el núcleo Linux además de otros softwares de código abierto. Android fue diseñado para dispositivos móviles, tabletas y relojes equipados con el sistema operativo, además el software también se utiliza en automóviles, televisores y otras máquinas.

Inicialmente fue desarrollado por Android Inc. y adquirido por Google en 2005. El código fuente principal de Android se conoce como Android Open Source Project (AOSP) y se licencia principalmente bajo la Licencia Apache.

En los últimos años, Android ha crecido de manera rápida en el mercado de los teléfonos inteligentes, que comenzó como un mercado multi-plataforma, y se ha ido convirtiendo un duopolio formado por los sistemas operativos iOS de Apple y Android de Google, desplazando a las demás plataformas como Windows Phone de Microsoft y el sistema operativo BlackBerry.

Según datos de la consultora de tecnología IDC, los dispositivos Android representaron más del 86% de las unidades distribuidas en 2019, Sin embargo,

hace tan solo una década, en 2010, la cuota de mercado de Android e iOS combinada era inferior al 40%, con BlackBerry, Windows Phone y otros sistemas operativos compartiendo el resto del mercado.

A continuación, se muestra la cuota de mercado de smartphones por sistema operativo, basado en unidades distribuidas.

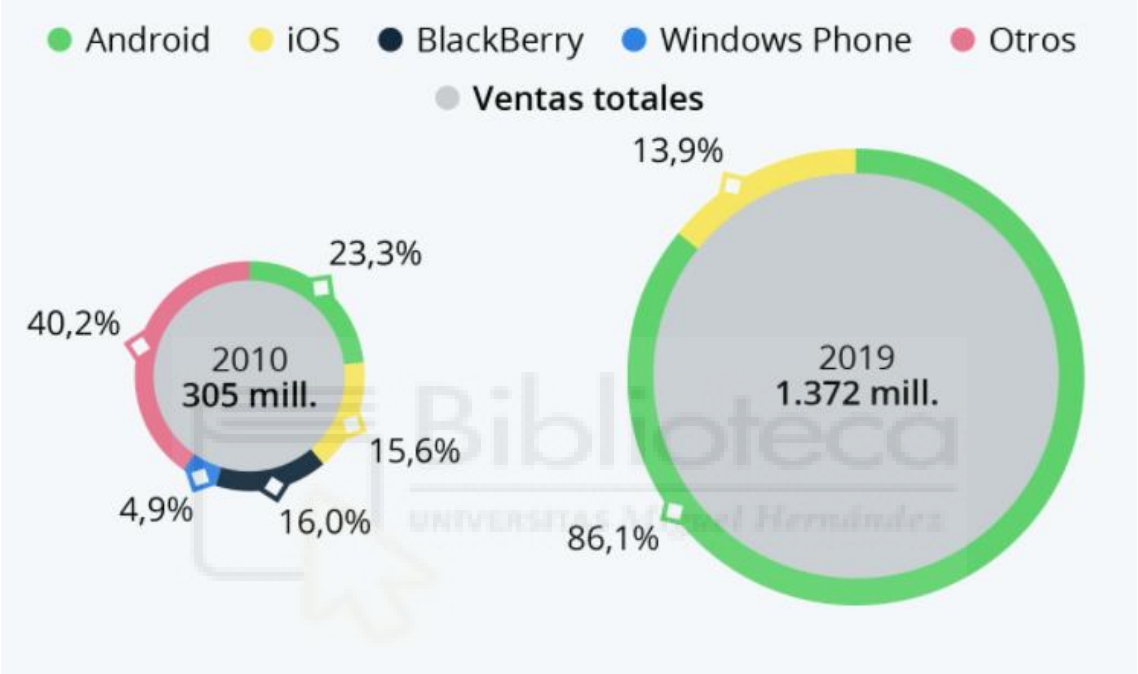

Ilustración 1 - Cuota de Mercado de Smartphones

El sistema operativo Android se inició en 2007 con el lanzamiento de la primera versión beta. En 2008, fue lanzada la primera versión comercial, Android 1.0, y desde entonces el software ha tenido numerosas versiones y constantes actualizaciones cuyo objetivo es el de corregir fallos y añadir nuevas funcionalidades, entre otros.

Dado el número de versiones desarrolladas hay algunos problemas técnicos a la hora de trabajar con Android. No todos los dispositivos en el mercado pueden actualizarse a la última versión, y debido a ello, a la hora de programar con este sistema operativo se ha de tener en cuenta el nivel de API, Application Programming Interface (interfaz de programación de aplicaciones).

A la hora de desarrollar una aplicación se ha de estudiar inicialmente que dispositivos serán capaces de instalar y ejecutar la aplicación diseñada. Utilizando un nivel menor de API se obtendrá un mayor número de dispositivos que podrán instalar la app ya que admite versiones más antiguas, sin embargo, con un mayor nivel de API se puede obtener un desarrollo de una aplicación más complejo, con menos errores y una programación más eficaz.

Se muestran en la tabla siguiente el historial de versiones de Android.

Nombre código	Número de versión	Fecha de lanzamiento	Nivel de API
Apple Pie	1.0	23 de septiembre de 2008	1
Banana Bread	1.1	9 de febrero de 2009	2
Cupcake	1.5	25 de abril de 2009	3
Donut	1.6	15 de septiembre de 2009	4
Eclair	2.0 – 2.1	26 de octubre de 2009	5 – 7
Froyo	2.2 – 2.2.3	20 de mayo de 2010	8
Gingerbread	2.3 – 2.3.7	6 de diciembre de 2010	9 – 10
Honeycomb	3.0 – 3.2.6	22 de febrero de 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.5	18 de octubre de 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	9 de julio de 2012	16 – 18
KitKat	4.4 – 4.4.4	31 de octubre de 2013	19 – 20
Lollipop	5.0 – 5.1.1	12 de noviembre de 2014	21 – 22
Marshmallow	6.0 – 6.0.1	5 de octubre de 2015	23
Nougat	7.0 – 7.1.2	15 de junio de 2016	24 – 25
Oreo	8.0 – 8.1	21 de agosto de 2017	26 – 27
Pie	9.0	6 de agosto de 2018	28
10	10.0	3 de septiembre de 2019	29
11	11.0	8 de septiembre de 2020	30

Versión descontinuada Versión antigua pero vigente Última versión

Ilustración 2 - Historial de Versiones

4. APLICACIONES SIMILARES

En esta sección se mostrarán varias aplicaciones con funciones similares a las de la app desarrollada en este proyecto. Estas aplicaciones han servido como modelo e inspiración para poder desarrollar el diseño, la interfaz y algunas de las funcionalidades de FridgeTracker.

Food Checklist

Esta aplicación permite monitorear suministros de alimentos en el hogar, observando las fechas de caducidad de los productos y utiliza la lista de compras para abastecer las existencias de alimentos.

Puede usar el código de barras de un alimento para agilizar el proceso de insertar nuevos elementos a las listas. Se sirve de la cámara del teléfono para leer los códigos y, además, puede asociar varios de estos códigos de barras con un mismo producto.

Permite agrupar los productos en categorías, personalizar la vista y asignar colores a los lugares o carpetas de almacenamiento para visualizar fácilmente e intuitivamente.

Esta aplicación tiene la capacidad de compartir las listas creadas con más usuarios. Mediante el correo electrónico otros usuarios pueden acceder a estas listas. También permite sincronizar todos los datos automáticamente si el usuario se registra en su cuenta con otro dispositivo.

A continuación se muestran varias de las pantallas de esta aplicación.

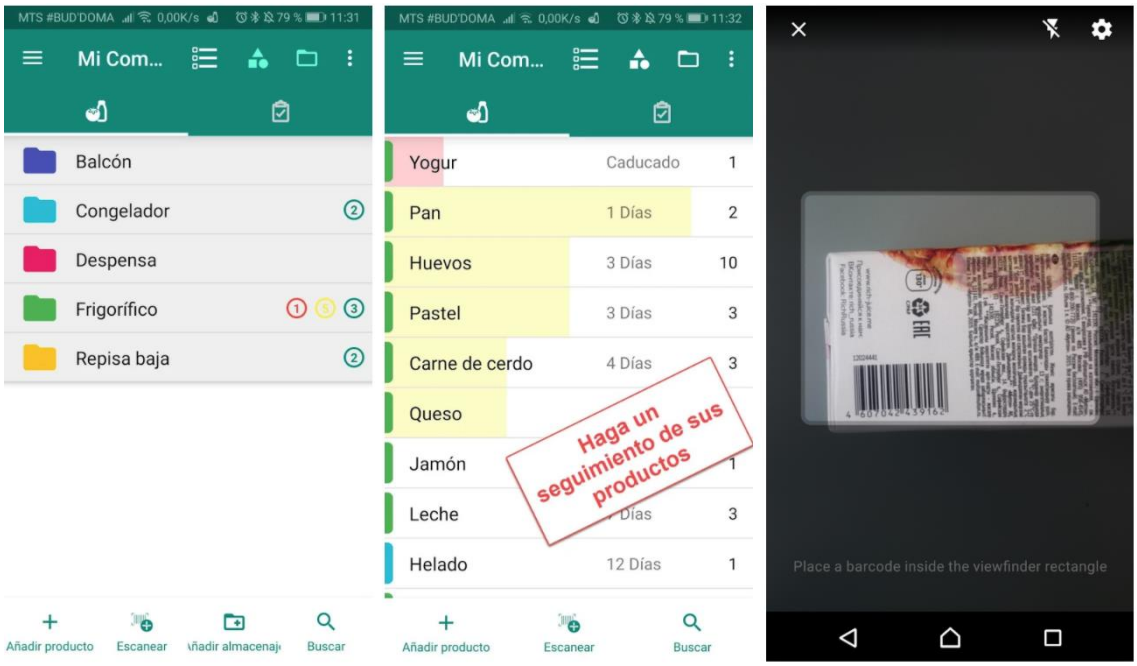

Ilustración 3 - Food Checklist

Pantrify

Es una aplicación que permite gestionar los productos que se almacenan en la despensa, nevera, lista de la compra o en cualquier otro lugar de forma sencilla, rápido e inteligente.

Las listas creadas están sincronizadas en la nube, posibilitando acceder a ellas desde cualquier dispositivo además de por compartirlas. Se pueden crear, además, varios grupos llevando así un control separado de hogares o eventos.

Algunas de las características de esta aplicación son:

- Escáner de códigos de barras.
- Entrada por voz.
- Adición de fotos personalizadas de los productos.
- Permite anotar el precio de los alimentos.
- Crea etiquetas para categorizar y filtrar elementos.
- Sincronización de listas en tiempo real.

En la siguiente ilustración se muestran diversas pantallas de Pantrify.

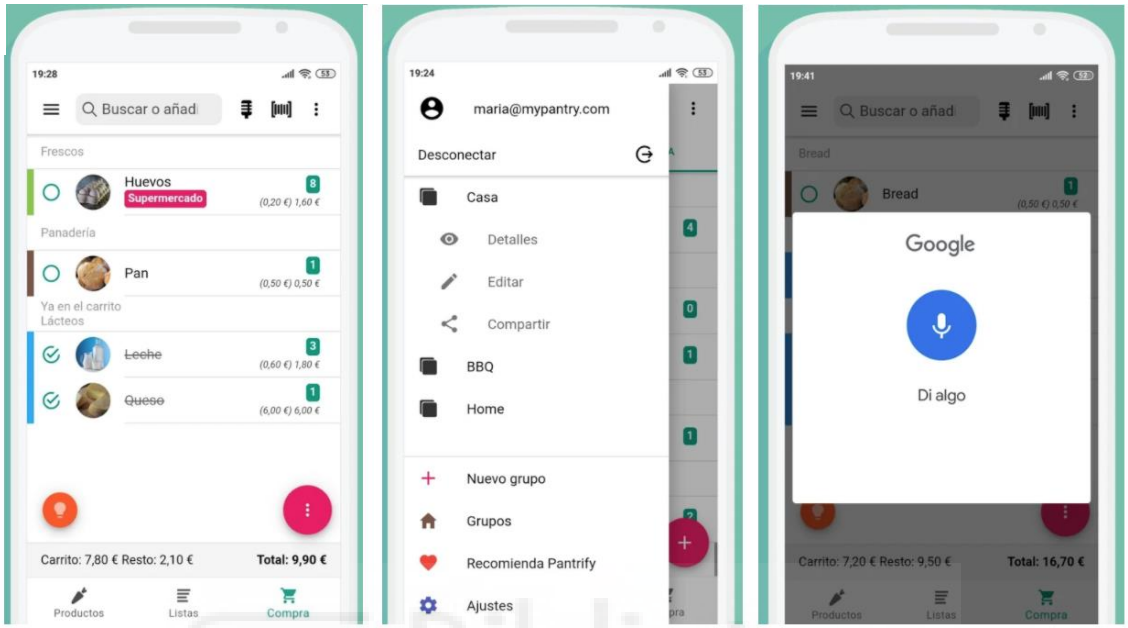

Ilustración 4 - Pantrify

5. DESCRIPCIÓN DE LA APLICACIÓN

La aplicación como se ha mencionado anteriormente se llama FridgeTracker y ha sido programada para poder añadir elementos a varias listas en diferentes actividades y transferir datos entre ellas para poder gestionar de manera eficiente los alimentos que se encuentran en la nevera, congelador y/o despensa del usuario.

FridgeTracker ha sido desarrollada en Android Studio y cuenta con cuatro actividades diferentes donde cada una es formada por un *RecyclerView* que contiene a las listas que se pueden crear y que están definidas por el archivo *.xml* que les da forma (*layout*).

Al lanzar la aplicación, esta se inicia con un *Splash Activity* que muestra el logo de la app por un breve periodo de tiempo antes de mostrar la actividad principal, la Lista de la Compra.

Ilustración 5 - Splash Activity

Ilustración 6 - Actividad Principal

En la actividad principal se muestran cinco elementos. Cuatro botones en la parte superior de la pantalla, que corresponden a las diferentes actividades de la aplicación, Lista de la Compra, Nevera, Congelador y Despensa. Cada botón permite navegar a cada una de dichas actividades.

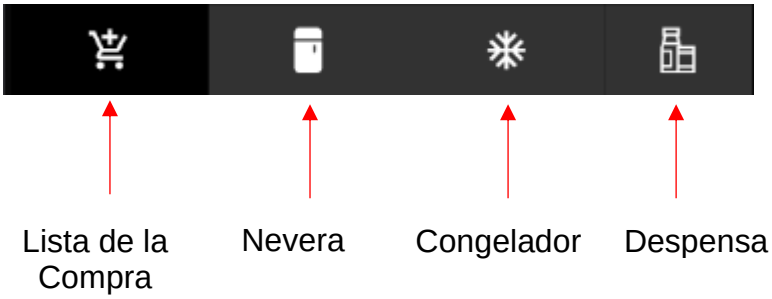

Ilustración 7 - Barra de Navegación Superior

En la parte inferior de la pantalla se encuentra el botón con el símbolo más, "+", que permite añadir un nuevo producto a la lista. Este botón aparece en todas las actividades y su funcionamiento es el mismo.

Ilustración 8 - Botón Añadir Producto

No obstante, al pulsar el botón de añadir producto se mostrarán dos tipos de ventanas para insertar los datos. El primer tipo corresponde a las actividades Lista de la Compra y Congelador, donde se pedirá al usuario escribir el nombre del producto y la cantidad de artículos a añadir. El segundo caso se muestra en las actividades Nevera y Despensa, donde además del nombre y la cantidad, se dará la opción de insertar la fecha de caducidad del alimento. Véase Ilustraciones 4 y 5.

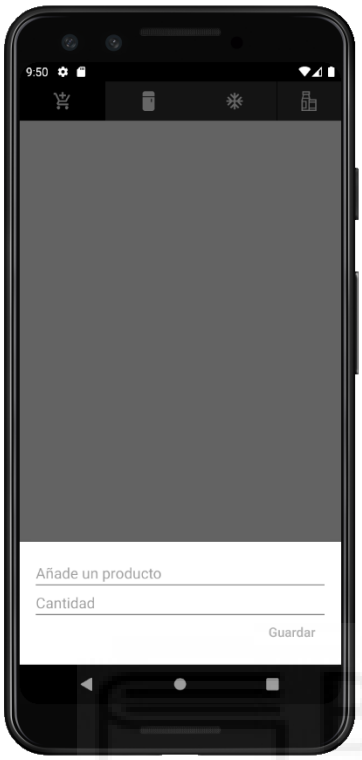

Ilustración 9 - Añadir Producto 1

Ilustración 10 - Añadir Producto 2

Para poder insertar un alimento en la lista se debe, al menos, rellenar el apartado “Añadir un producto”, tras escribir el nombre el botón que aparece en la parte inferior derecha, el botón “Guardar” se activará y al pulsarlo se podrá guardar el producto en la lista. En este caso el producto se mostrará en la actividad sin ningún otro dato adicional, sin embargo, se podrá editar posteriormente para añadir la cantidad y/o fecha de caducidad.

Al pulsar el botón “Fecha de caducidad” aparecerá un calendario que permitirá añadir la fecha.

En caso de no querer añadir ningún producto, basta con pulsar fuera de la ventana emergente para salir de ella.

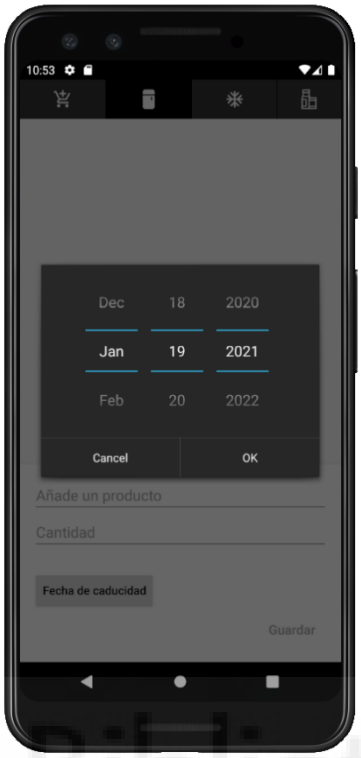

Ilustración 11 - Fecha de Caducidad

Una vez añadidos los productos, estos se muestran en las actividades en forma de lista vertical descendente como se muestra en las siguientes ilustraciones.

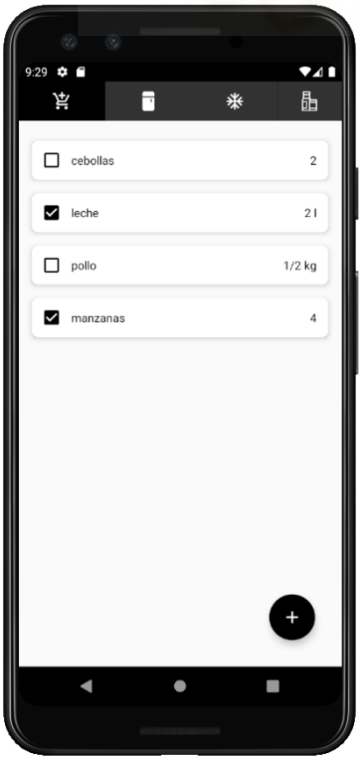

Ilustración 12 - Lista de la Compra

Ilustración 13 - Nevera

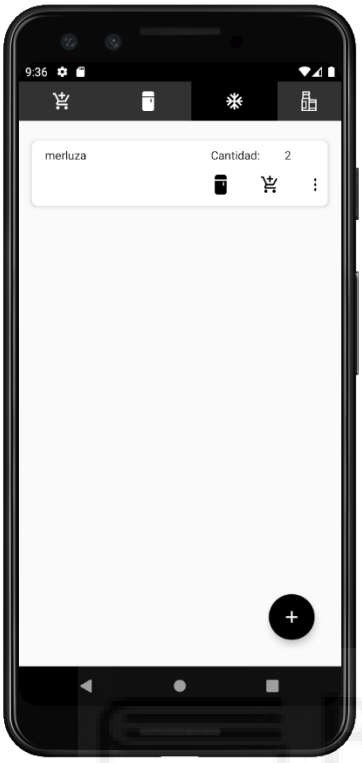

Ilustración 14 - Congelador

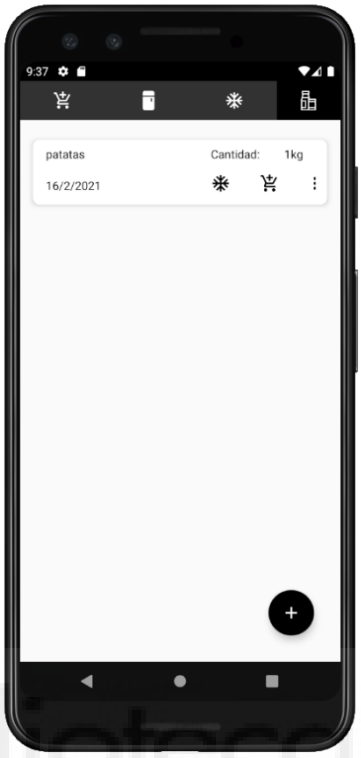

Ilustración 15 - Despensa

Una vez añadidos los productos, cada actividad muestra diferentes funcionalidades.

En la actividad Lista de la Compra, a la izquierda de los productos se mostrará una casilla en blanco, que se podrá marcar y desmarcar y que sirve para indicar que un alimento ha sido adquirido o puesto en el carro de la compra.

Además, los productos mostrados en la lista pueden ser editados o eliminados deslizando el dedo hacia la izquierda o derecha.

Al deslizar el producto hacia la izquierda, se muestra un menú que da la opción al usuario de eliminar el producto o cancelar la operación.

Al deslizar hacia la derecha se permite editar el producto. Aparece la ventana emergente mostrada en las ilustraciones 4 y 5 con los datos del producto que se desea modificar.

Ilustración 16 - Deslizar Izquierda

Ilustración 17 - Eliminar Producto

Ilustración 18 - Deslizar Derecha

Ilustración 19 - Editar Producto

En las actividades Nevera, Congelador y Despensa se muestran los productos de una manera diferente a la Lista de la Compra. Estas actividades cuentan con un número de botones que permiten modificar los elementos de la lista, eliminarlos y enviar datos entre las cuatro actividades.

El botón formado por tres puntos verticales despliega un menú que permite eliminar o editar el producto, similar a la función de deslizamiento mencionada en la actividad Lista de la Compra.

El funcionamiento de el botón con forma de carrito con un símbolo “+” muestra la opción de añadir el elemento clicado a la Lista de la Compra. El propósito es el de volver añadir un producto que se esté agotando o se desee adquirir de nuevo a la actividad sin necesidad de insertar el producto desde cero.

Ilustración 20 - Editar o Eliminar

Ilustración 21 - Añadir a Lista de la Compra

El botón situado a la izquierda del carrito, en el caso de las actividades Nevera y Despensa, permite mover un producto a la actividad Congelador, simula que el usuario ha decidido congelar un alimento, ya sea porque no va a ser consumido en los próximos días o se desea evitar que caduque.

En el caso de la actividad Congelador, el botón mencionado arriba es sustituido por el de una nevera. Que permite mandar el alimento del congelador a la actividad Nevera, simulando que se ha sacado un producto de congelador para descongelarlo.

Ilustración 22 - Mover al Congelador

Ilustración 23 - Mover a la Nevera

6. HERRAMIENTAS DE DESARROLLO EN EL PROYECTO

En esta sección se definirán los programas que se han utilizado para desarrollar el Trabajo de Fin de Grado, en este caso, Adobe XD para el diseño de la estructura y apariencia de la app, y Android Studio, para la implementación del diseño en el lenguaje de programación Java.

5.1 ADOBE XD

Adobe XD es un constructor de interfaz gráfica y editor de gráficos vectoriales desarrollado por Adobe Inc. cuya función principal es la de diseñar y crear prototipos para la experiencia de usuario en el entorno de las páginas web y aplicaciones móvil.

Este programa es capaz de crear prototipos interactivos de alta fidelidad que simulan el funcionamiento de una página web o aplicación sin necesidad de programarla primero. Se posibilita de esta manera mostrar una app a los usuarios de pruebas con la función de corregir errores de diseño, así como funcionalidades y crear una versión más pulida que será la que se programe tras finalizar la fase de diseño.

Las características principales de Adobe XD son:

- Permite importar archivos desde Photoshop, Illustrator y Sketch simplificando la interoperabilidad desde varios programas de diseño.
- Es capaz de crear componentes reutilizables, como botones o barras de navegación para evitar crear duplicados y agilizar todo el proceso.
- Puede desarrollar microinteracciones y animaciones mostrando como sería una versión final de una página web o aplicación.
- También posibilita la edición colaborativa de documentos y publicación de los diseños creados mediante enlaces.

Se pueden dividir las funcionalidades de Adobe XD en tres interfaces. La primera consiste en la interfaz de Diseño, donde se incluyen los componentes del diseño,

las imágenes, botones y los demás elementos que forman la aplicación. En la interfaz de Prototipado se conectan las diversas pantallas que componen la app y es la que permite simular el funcionamiento de la misma. La última interfaz, Compartir, es la encargada de generar un enlace para que se pueda compartir la aplicación la cual se puede ver y probar aun no teniendo Adobe XD instalado.

A continuación se muestran las tres interfaces mencionadas.

Ilustración 24 - Interfaz de Diseño

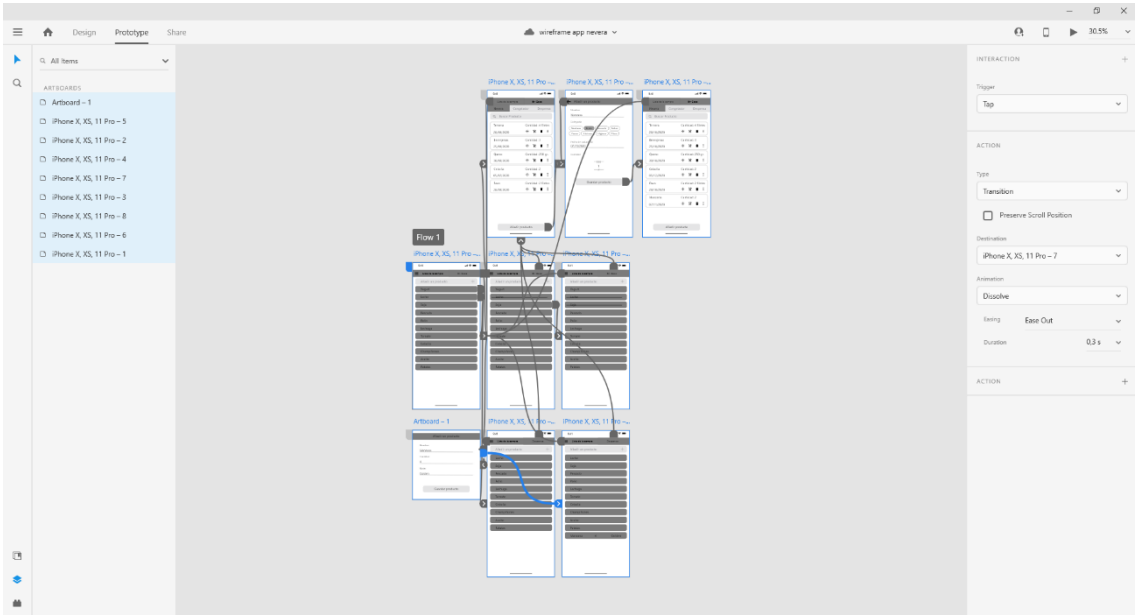

Ilustración 25 - Interfaz de Prototipado

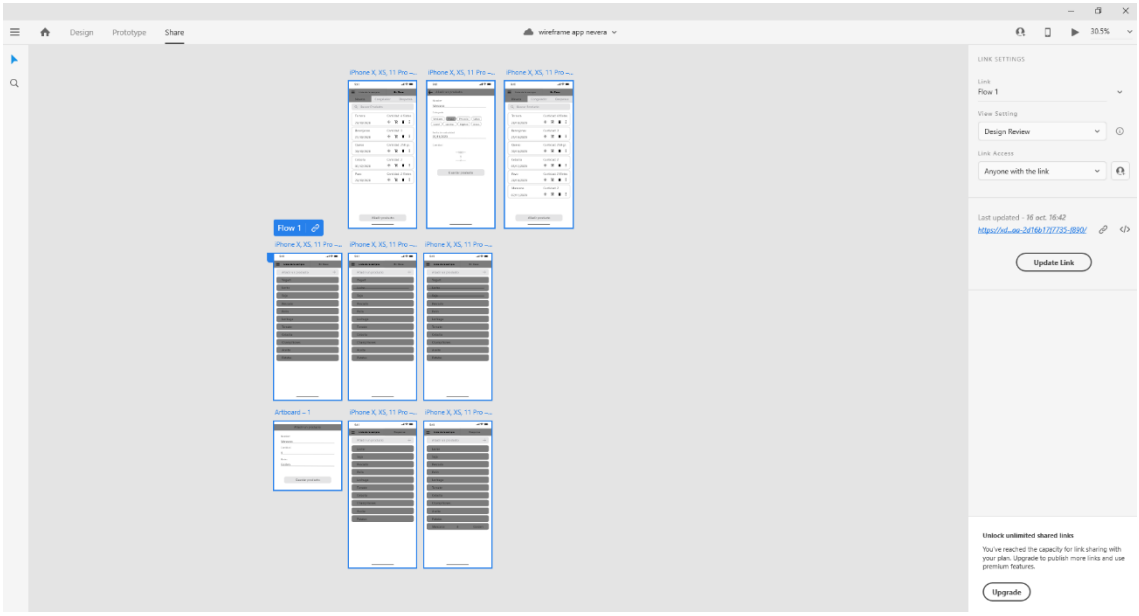

Ilustración 26 - Interfaz para Compartir

En cuanto a los requisitos mínimos del sistema, recomendados por Adobe, para su instalación y uso:

Para macOS

	Requisitos mínimos
Sistema operativo	macOS X v10.14 o posterior
Pantalla	Pantalla de 13 pulgadas o mayor, resolución de 1400 x 900, se recomienda Retina.
Internet	Es preciso disponer de conexión a Internet y estar registrado para poder activar el software, validar las suscripciones y acceder a los servicios en línea. Las posibilidades de voz requieren que los usuarios estén conectados a Internet para obtener una vista previa de sus prototipos.
RAM	4 GB de RAM
Almacenamiento	Recomendamos usar documentos en la nube (almacenados en Creative Cloud), pero también se admiten documentos almacenados en el almacenamiento local. Para almacenar documentos de XD, no recomendamos el uso de almacenamientos en la nube sincronizados (por ejemplo, CC Files, Dropbox, Google Drive, etc.) o unidades de red.

Ilustración 27 - Requerimientos macOS Adobe XD

Para Windows

	Requisitos mínimos
Sistema operativo	Windows 10 (64 bits): versión 1803 (compilación 10.0.17134) o posterior.
Pantalla	Pantalla de 13 pulgadas o mayor, resolución de 1280 x 800.
Internet	Es preciso disponer de conexión a Internet y estar registrado para poder activar el software, validar las suscripciones y acceder a los servicios en línea. Las posibilidades de voz requieren que los usuarios estén conectados a Internet para obtener una vista previa de sus prototipos.
RAM	4 GB de RAM
Gráficos	Conjunto mínimo de funciones Direct 3D DDI Feature Set: 10. Para la GPU de Intel, son necesarios los controladores distribuidos en 2014 o posteriormente. Para encontrar esta información, ejecute "dxdiag" en el menú Ejecutar y seleccione la pestaña "Pantalla".
Entrada de lápiz y de función táctil	XD en Windows 10 es compatible con las funciones nativas de lápiz y táctil de Windows. Para obtener más información sobre cómo trabajar con lápiz y táctil en XD, consulte Preguntas frecuentes.
Almacenamiento	Recomendamos usar documentos en la nube (almacenados en Creative Cloud), pero también se admiten documentos almacenados en el almacenamiento local. Para almacenar documentos de XD, no recomendamos el uso de almacenamientos en la nube sincronizados (por ejemplo, CC Files, Dropbox, Google Drive, etc.) o unidades de red.

Ilustración 28 - Requerimientos Windows Adobe XD

5.2 ANDROID STUDIO

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android, basado en IntelliJ IDEA. Es un editor de códigos potente cuyas herramientas para desarrolladores permiten aumentar la productividad al trabajar con Android, además de que admite varios lenguajes de programación como Kotlin, Java y C++. Entre las principales funcionalidades de Android Studio encontramos:

- Compatibilidad integrada con Google Cloud Platform.
- Compatibilidad con C++ y NDK
- Emulador virtual de Android utilizado para ejecutar y probar aplicaciones.
- Plantillas para crear diseños utilizados por Android.
- Integración con GitHub y plantillas de código.
- Renderizado en tiempo real.
- Cuenta con herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de la versión, entre otros.

La interfaz principal de trabajo de Android Studio se divide en varias áreas lógicas.

- La barra de herramientas que permite ejecutar la app o iniciar herramientas en Android.
- La barra de navegación que sirve para explorar el proyecto o abrir archivos y editarlos, entre otras funciones.
- La ventana del editor, es el área central donde se crea y modifica el código.
- La barra de la ventana de herramientas contiene los botones que permiten expandir o contraer ventanas de herramientas individuales.
- Las ventanas de herramientas, desde donde se puede acceder a la administración del proyecto, la búsqueda de elementos y el control de versiones.
- La barra de estado, en esta se muestran mensajes y advertencias, así como el estado del proyecto.

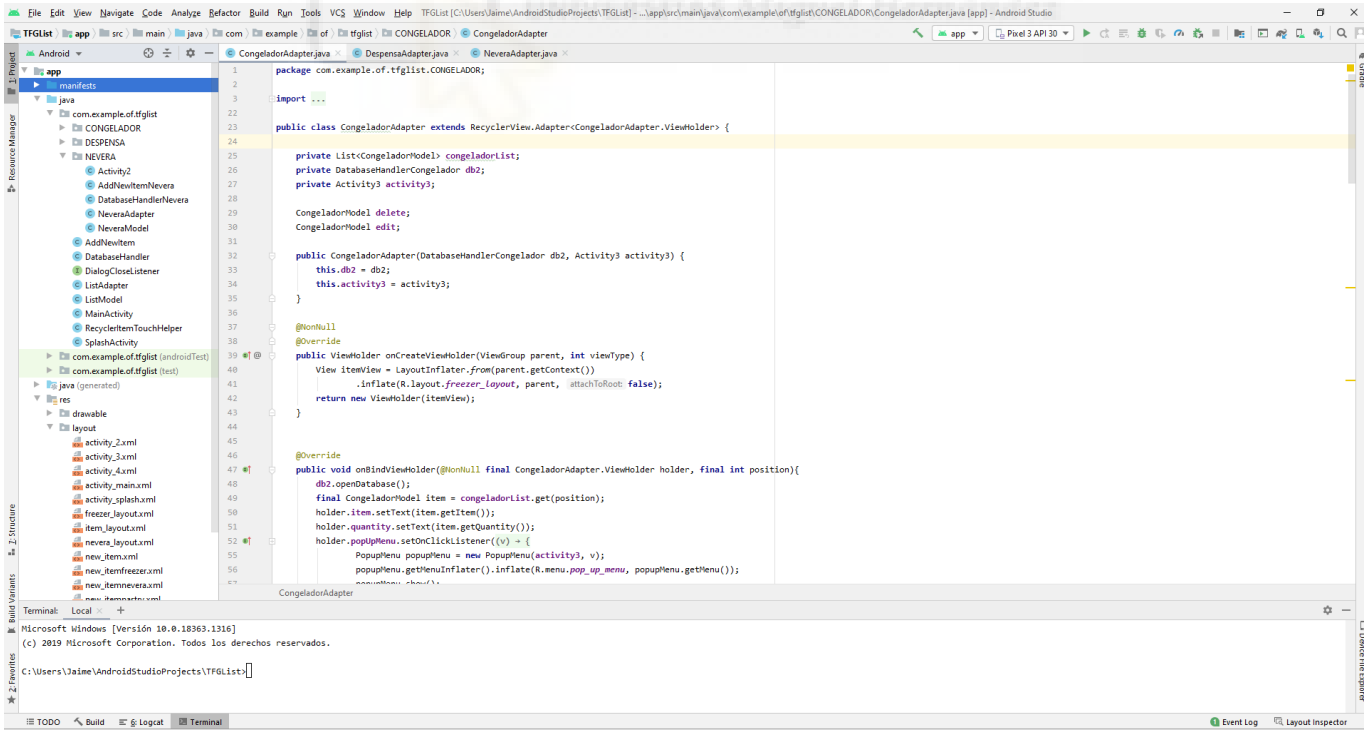

Ilustración 29 - Interfaz de Android Studio

Android Studio incorpora un emulador que simula dispositivos Android en una computadora en la cual se puede probar la app en diferentes dispositivos y niveles de API. Este emulador proporciona casi todas las funciones de un dispositivo Android, se pueden simular llamadas y mensajes, dar la ubicación del dispositivo y acceder a Google Play Store entre otras funciones.

La interfaz del emulador es la siguiente:

Ilustración 30 - Interfaz del Emulador

Los requisitos del sistema recomendados para Android son los siguientes:

Windows

- Microsoft® Windows® 7/8/10 (64-bit)
- 4 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum,
  4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

Mac

- Mac® OS X® 10.10 (Yosemite) or higher, up to 10.14 (macOS Mojave)
- 4 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum,
  4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

Linux

- GNOME or KDE desktop
  Tested on gLinux based on Debian.
- 64-bit distribution capable of running 32-bit applications
- GNU C Library (glibc) 2.19 or later
- 4 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum,
  4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

Chrome OS

- 8 GB RAM or more recommended
- 4 GB of available disk space minimum
- 1280 x 800 minimum screen resolution
- Intel i5 or higher (U series or higher) recommended

For more information on recommended devices as well as Android emulator support, visit chromeos.dev

Ilustración 31 - Requerimientos Android Studio

7. MATERIALES Y MÉTODOS

7.1 LENGUAJE DE PROGRAMACIÓN

Android Studio, como se ha mencionado anteriormente, es el software empleado para el desarrollo de la aplicación en este Trabajo de Fin de Grado. Se ha trabajado con dos tipos de archivo en este proyecto, archivos XML y Java.

7.1.1 ARCHIVOS XML

Un archivo *.xml* consiste en un archivo de lenguaje marcado extensible (XML) el cual es formado por un archivo de texto sin formato que utiliza una serie de etiquetas personalizadas con la finalidad de describir tanto la estructura como otras características del documento.

En este proyecto los archivos XML creados son los que definen el diseño de la interfaz (*layout*) que el usuario visualiza e interactúa.

7.1.2 ARCHIVOS JAVA

Un archivo JAVA es un archivo de código fuente escrito en el lenguaje de programación JAVA y que, utiliza un enfoque orientado a objectos donde los tipos de datos estructurados, llamados clases, se utilizan para crear instancias de objetos en tiempo de ejecución.

En estos archivos se programan las instrucciones para el comportamiento de los elementos declarados en los archivos XML.

7.2 PROGRAMACIÓN Y DISEÑO

7.2.1 ESTRUCTURA DEL PROYECTO

Cada proyecto de Android Studio incluye uno o más módulos con archivos de código fuente y archivos de recursos. Entre los tipos de módulos se incluyen los siguientes:

- Módulos de apps para Android
- Módulos de biblioteca

- Módulos de Google App Engine

De manera predeterminada, Android Studio muestra los archivos del proyecto abierto en la vista de proyecto de Android, mostrado en la siguiente ilustración. Esta vista está organizada en módulos que permiten acceder de manera sencilla a los archivos fuente clave en el proyecto que se está desarrollando.

Todos los archivos de compilación pueden ser observados en el nivel superior de Secuencias de comando Gradle, donde cada módulo contiene las siguientes carpetas:

- manifests: esta carpeta contiene el archivo AndroidManifest.xml. Este archivo es la raíz de la fuente del proyecto, describe información esencial de la aplicación para las herramientas de creación de Android.
- java: contiene los archivos de código fuente Java, incluido el código de prueba de JUnit.
- res: contiene todos los recursos usados por la aplicación, como diseños XML, strings de IU e imágenes de mapa de bits.

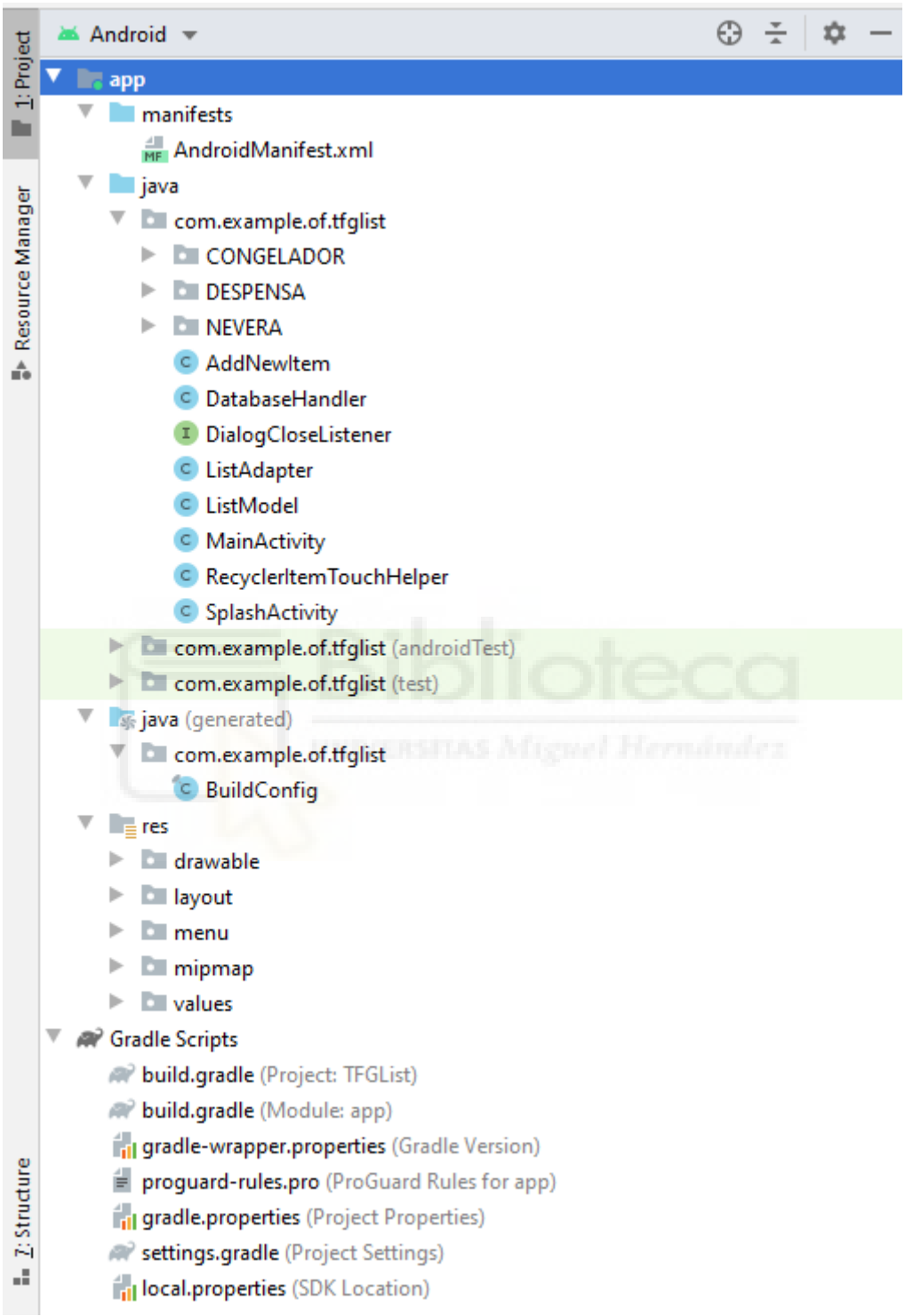

Ilustración 32 - Archivos del Proyecto

7.2.2 CÓDIGO DEL PROYECTO

En esta sección se describirá el código utilizado para desarrollar la aplicación FridgeTracker. Se explicarán los métodos utilizados y se mostrarán ilustraciones que contienen el código.

7.2.2.1 ANDROID MANIFEST

Toda aplicación cuenta con el archivo AndroidManifest.xml en el directorio raíz. Este archivo realiza diversas funciones esenciales para el funcionamiento de la app, entre otras, debe declarar lo siguiente:

- El nombre del paquete de la aplicación. Las herramientas de compilación de Android usan esto para determinar la ubicación de las entidades de código cuando se compila el proyecto.
- Los componentes de la aplicación, que incluyen todas las actividades, servicios, receptores de emisiones y proveedores de contenido.
- Los permisos que necesita la aplicación para acceder a las partes protegidas del sistema o a otras aplicaciones.
- Las funciones de hardware y software que requiere la aplicación afectan a los dispositivos que pueden instalar la aplicación desde Google Play.

Al comenzar un proyecto con Android Studio para desarrollar una aplicación se generará el archivo *Android Manifest* automáticamente, y la mayoría de los elementos esenciales de este se irán agregando a medida que se compila la aplicación.

En la siguiente ilustración se muestra el archivo *Android Manifest* correspondiente a este proyecto.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.of.tfglist">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="TFGList"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".DESPENSA.Activity4" />
        <activity android:name=".CONGELADOR.Activity3" />
        <activity android:name=".NEVERA.Activity2" />
        <activity android:name=".SplashActivity">
            <intent-filter>
                <category android:name="android.intent.category.LAUNCHER" />

                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>
        <activity android:name=".MainActivity" />
    </application>

</manifest>
```

Ilustración 33 - AndroidManifest

En el archivo *manifest* mostrado se pueden observar las diferentes actividades que forma la aplicación:

- *MainActivity*, que corresponde a la Lista de la Compra.
- *NEVERA.Activity2*, la Nevera
- *CONGELADOR.Activity3*, el Congelador
- *DESPENSA.Activity4*, la Despensa

7.2.2.2 SPLASH ACTIVITY

La actividad *SplashActivity.java*, funciona como un lanzador de la aplicación y sirve para mostrar el logo de la app al iniciarse esta.

```java
package com.example.of.tfglist;

import ...

public class SplashActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
        getSupportActionBar().hide();

        final Intent i = new Intent( packageContext: SplashActivity.this, MainActivity.class);
        new Handler().postDelayed(() -> {
                startActivity(i);
                finish();
        }, delayMillis: 2000);
    }
}
```

Ilustración 34 - SplashActivity

7.2.2.3 ACTIVIDAD PRINCIPAL – MainActiviy.java

Este archivo es la actividad principal y punto de entrada de interacción del usuario con una app, también es fundamental para el usuario que navega por la aplicación. Las actividades se implementan mediante la clase *Activity* y posibilitan interacciones clave entre el sistema y la aplicación, entre los cuales se encuentran:

- Realizar un seguimiento de lo que realmente le interesa al usuario (lo que está en pantalla) para garantizar que el sistema siga ejecutando el proceso que aloja la actividad.
- Saber que procesos usados con anterioridad contienen elementos a los que el usuario puede regresar y con ello priorizar dichos procesos.
- Ayudar a la aplicación a controlar la finalización de su proceso para que el usuario pueda regresar a las actividades con el estado anterior restaurado.
- Permitir que las aplicaciones implementen flujos de usuarios entre sí y que el sistema los coordine.

A continuación, se muestra el código de la actividad principal, este código ha sido dividido en dos secciones separadas, la cuales se encargan de diferentes funciones en la aplicación.

En la primera sección se definen las distintas variables que se utilizarán en la actividad principal, entre ella se encuentran la base de datos, la *RecyclerView* que formará la lista de los productos, la lista de array donde se encuentran los alimentos, los botones de cambio entre pantallas y el de añadir productos.

Tras definir las variables, la función *onCreate* inicializa la actividad y se define la interfaz que utilizará esta pantalla, en este caso definida por el archivo XML *activity_main*.

El resto de código en esta parte prepara la base de datos y los elementos en la lista, si hay alguno creado, además de funciones como la de deslizar el dedo para editar o eliminar elementos como hemos mencionado anteriormente.

```java
package com.example.of.tfglist;

import ...

public class MainActivity extends AppCompatActivity implements DialogCloseListener {

    private DatabaseHandler db;

    private RecyclerView listRecyclerView;
    private ListAdapter listAdapter;
    private FloatingActionButton fab;

    private List<ListModel> itemList;

    private ImageButton button2;
    private ImageButton button3;
    private ImageButton button4;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Objects.requireNonNull(getSupportActionBar()).hide();

        db = new DatabaseHandler( context: this);
        db.openDatabase();

        listRecyclerView = findViewById(R.id.listRecyclerView);
        listRecyclerView.setLayoutManager(new LinearLayoutManager( context: this));
        listAdapter = new ListAdapter(db, activity: MainActivity.this);
        listRecyclerView.setAdapter(listAdapter);

        ItemTouchHelper itemTouchHelper = new
                ItemTouchHelper(new RecyclerItemTouchHelper(listAdapter));
        itemTouchHelper.attachToRecyclerView(listRecyclerView);

        fab = findViewById(R.id.fab);

        itemList = db.getAllItems();
        Collections.reverse(itemList);

        listAdapter.setItems(itemList);

        fab.setOnClickListener((v) -> {
                AddNewItem.newInstance().show(getSupportFragmentManager(), AddNewItem.TAG);
        });
```

Ilustración 35 - MainActivity - Parte 1

En la segunda sección de la actividad principal se programan los botones creados, definiéndolos mediante el método *findViewById*, para que inicien las demás actividades. Para ello se utiliza la función *Intent* y *startActivity*.

```
70
71          //buttons to change activity
72          button2 = (ImageButton) findViewById(R.id.button2);
73          button2.setOnClickListener((v) → { openActivity2(); });
79
80          button3 = (ImageButton) findViewById(R.id.button3);
81          button3.setOnClickListener((v) → { openActivity3(); });
87
88          button4 = (ImageButton) findViewById(R.id.button4);
89          button4.setOnClickListener((v) → { openActivity4(); });
95
96          SharedPreferences preferences = getSharedPreferences( name: "ListaCompra", MODE_PRIVATE);
97          String item = preferences.getString( key: "item",  defValue: "");
98          String xcompra = preferences.getString( key: "quantity",  defValue: "");
99
100         SharedPreferences.Editor editor = preferences.edit();
101         editor.clear();
102         editor.commit();
103
104         if (!item.isEmpty()) {
105             ListModel listModel = new ListModel(item, xcompra);
106             db.insertItem(listModel);
107             itemList.add(listModel);
108
109             itemList = db.getAllItems();
110             Collections.reverse(itemList);
111             listAdapter.setItems(itemList);
112             listAdapter.notifyDataSetChanged();
113
114             preferences.edit().commit();
115         }
116     }
117     @Override
118     public void handleDialogClose(DialogInterface dialog){
119         itemList = db.getAllItems();
120         Collections.reverse(itemList);
121         listAdapter.setItems(itemList);
122         listAdapter.notifyDataSetChanged();
123     }
124     //create intent to open the different activities
125     public void openActivity2(){
126         Intent intent = new Intent( packageContext: this, Activity2.class);
127         startActivity(intent);
128     }
129     public void openActivity3(){
130         Intent intent = new Intent( packageContext: this, Activity3.class);
131         startActivity(intent);
132     }
133     public void openActivity4(){
134         Intent intent = new Intent( packageContext: this, Activity4.class);
135         startActivity(intent);
136     }
137 }
```

Ilustración 36 - MainActivity - Parte 2

Las actividades Nevera (*Activity2*), Congelador (*Activity3*) y Despensa (*Activity4*) muestran una estructura similar a la actividad principal, diferenciada por los archivos que definen sus interfaces.

7.2.2.4 ReclyerItemTouchHelper

Este archivo JAVA es el encargado de realizar las funciones de deslizar a izquierda o derecha un producto. Y el que genera los diálogos emergentes de editar y eliminar un producto de la lista. En la siguiente ilustración se muestra la parte clave del código donde se han programado estas funciones.

```
32      @Override
33      public void onSwiped(@NonNull final RecyclerView.ViewHolder viewHolder, int direction) {
34          final int position = viewHolder.getAdapterPosition();
35          if (direction == ItemTouchHelper.LEFT) {
36              AlertDialog.Builder builder = new AlertDialog.Builder(adapter.getContext());
37              builder.setCancelable(false);
38              builder.setTitle("Eliminar Producto");
39              builder.setMessage("Seguro que deseas eliminar este producto?");
40              builder.setPositiveButton( text: "Confirmar",
41                      new DialogInterface.OnClickListener() {
42                          @Override
43                          public void onClick(DialogInterface dialog, int which) {
44                              adapter.deleteItem(position);
45                          }
46                      });
47              builder.setNegativeButton( text: "Cancelar",
48                      (dialog, which) -> {
51                      adapter.notifyItemChanged(viewHolder.getAdapterPosition());
52              });
54              AlertDialog dialog = builder.create();
55              dialog.show();
56          } else {
57              adapter.editItem(position);
58          }
59      }
```

Ilustración 37 - RecyclerItemTouchHelper

7.2.2.5 ADAPTADOR – Adapter

Este archivo es el encargado de darle forma a la interfaz donde se encuentran los productos añadidos dentro de las actividades. El adaptador dará acceso a los archivos de información y es el responsable de mostrar una vista (*view*) para cada elemento.

En este proyecto los archivos *(…)Adapter.java* enlazan los *RecyclerView* de cada actividad con los *ArrayList* correspondientes para formar las diversas listas.

La siguiente ilustración muestra la parte esencial del código del adaptador.

```java
package com.example.of.tfglist;

import ...

public class ListAdapter extends RecyclerView.Adapter<ListAdapter.ViewHolder> {
    private List<ListModel> shopList;
    private DatabaseHandler db;
    private MainActivity activity;

    public ListAdapter(DatabaseHandler db, MainActivity activity) {
        this.db = db;
        this.activity = activity;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View itemView = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_layout, parent, attachToRoot: false);
        return new ViewHolder(itemView);
    }

    @Override
    public void onBindViewHolder(@NonNull final ViewHolder holder, int position) {
        db.openDatabase();

        final ListModel item = shopList.get(position);
        holder.item.setText(item.getItem());
        holder.compra.setText(item.getXcompra());

        holder.item.setChecked(toBoolean(item.getStatus()));
        holder.item.setOnCheckedChangeListener((buttonView, isChecked) -> {
                if (isChecked) {
                    db.updateStatus(item.getId(), status: 1);
                } else {
                    db.updateStatus(item.getId(), status: 0);
                }
        });
    }

```

Ilustración 38 - Adapter

7.2.2.6 BASE DE DATOS – DatabaseHandler

La base de datos almacena y guarda los datos insertados en la actividad para que no se eliminen y pierdan al moverse entre actividades o al cerrar la aplicación. En el presente proyecto, este archivo crea una tabla donde se alojan los datos introducidos al añadir un producto, en este caso los datos que puede almacenar son el nombre del producto, su *id* o nombre de recurso único, su estado (*status*), la cantidad y la fecha de caducidad.

A continuación se muestra el código para crear la tabla de la base de datos:

```java
package com.example.of.tfglist;

import ...

public class DatabaseHandler extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String NAME = "toListDatabase";
    private static final String LIST_TABLE = "list";
    private static final String ID = "id";
    private static final String ITEM = "item";
    private static final String STATUS = "status";
    private static final String XCOMPRA = "xcompra";


    private static final String CREATE_LIST_TABLE = "CREATE TABLE " + LIST_TABLE + "(" + ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + ITEM + " TEXT, "
            + XCOMPRA + " TEXT, " + STATUS + " INTEGER)";


    private SQLiteDatabase db;

    public DatabaseHandler(Context context) { super(context, NAME,  factory: null, VERSION); }

    @Override
    public void onCreate(SQLiteDatabase db) { db.execSQL(CREATE_LIST_TABLE); }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Drop older table if existed
        db.execSQL("DROP TABLE IF EXISTS " + LIST_TABLE);
        // Create tables again
        onCreate(db);
```

Ilustración 39 - DataBaseHandler

7.2.2.7 AÑADIR UN NUEVO PRODUCTO – AddNewItem

El archivo *AddNewItem.java* genera un diálogo emergente en el cual se pide al usuario que inserte varios datos relacionados con el alimento que se quiere añadir a la lista. El usuario puede darle nombre al producto, insertar la cantidad y añadir la fecha de caducidad del producto.

La siguiente ilustración representa el código utilizado para añadir un producto.

```java
public class AddNewItemNevera extends BottomSheetDialogFragment {
    public static final String TAG = "ActionBottomDialog";
    private EditText newItemText1;
    private EditText newItemQuantity1;
    private Button newItemSaveButton1;
    private TextView fecha;

    private Button caducidadBtn;
    private DatePickerDialog.OnDateSetListener mDateSetListener;

    private DatabaseHandlerNevera db1;

    public static AddNewItemNevera newInstance() { return new AddNewItemNevera(); }
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setStyle(STYLE_NORMAL, R.style.DialogStyle);
    }
```

```
56
57      @NonNull
58      @Override
59      public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
60
61          View view = inflater.inflate(R.layout.new_itemnevera, container,  attachToRoot: false);
62          getDialog().getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_ADJUST_RESIZE);
63
64          return view;
65      }
66
67      @Override
68      public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
69          super.onViewCreated(view, savedInstanceState);
70          newItemText1 = Objects.requireNonNull(getView()).findViewById(R.id.newItemText1);
71          newItemQuantity1 = Objects.requireNonNull(getView()).findViewById(R.id.newItemQuantity1);
72          newItemSaveButton1 = getView().findViewById(R.id.newItemButton1);
73          fecha = getView().findViewById(R.id.fechas);
74          caducidadBtn = getView().findViewById(R.id.caducidad);
75
76          caducidadBtn.setOnClickListener((v) → {
80                  Calendar cal = Calendar.getInstance();
81                  int year = cal.get(Calendar.YEAR);
82                  int month = cal.get(Calendar.MONTH);
83                  int day = cal.get(Calendar.DAY_OF_MONTH);
84
85                  DatePickerDialog dialog = new DatePickerDialog(Objects.requireNonNull(getContext()),
86                          android.R.style.Theme_Holo_Dialog_MinWidth,
87                          mDateSetListener, year, month, day);
88                  dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));
89                  dialog.show();
90          });
92          mDateSetListener = (OnDateSetListener) (view, year, month, dayOfMonth) → {
95                  month = month + 1;
96                  String date = dayOfMonth + "/" + month + "/" + year;
97                  fecha.setText(date);
98          };
100
101         boolean isUpdate = false;
102
103         final Bundle bundle = getArguments();
104         if(bundle != null) {
105             isUpdate = true;
106             String item = bundle.getString( key: "item");
107             String quantity = bundle.getString( key: "quantity");
108             String fechas = bundle.getString( key: "fechas");
109             newItemText1.setText(item);
110             newItemQuantity1.setText(quantity);
111             fecha.setText(fechas);
112             assert item != null;
113             if (item.length()>0)
114                 newItemSaveButton1.setTextColor(ContextCompat.getColor(Objects.requireNonNull(getContext()),R.color.colorPrimaryDark));
115         }
116
117         db1 = new DatabaseHandlerNevera(getActivity());
118         db1.openDatabase();
```

Ilustración 40 - AddNewItem

7.2.2.8 INTERFAZ – Layout

En la carpeta *res* > *layout* se encuentra los archivos *.xml* que definen las interfaces de la aplicación. El diseño de la app vendrá definido por estos archivos y será la parte por donde interactúe el usuario con todos los elementos.

Cada archivo de diseño debe contener un elemento de raíz, que debe ser un objecto *View* o *ViewGroup*. Una vez definido el elemento raíz, se pueden agregar *widgets* u objectos de diseño adicionales como elementos secundarios para crear gradualmente una jerarquía de vistas que defina el diseño. A continuación, se describirán los objetos y *widgets* utilizados en los archivos *.xml*.

- RelativeLayout: es un grupo de vistas que muestra vistas secundarias en posiciones relativas. La posición de cada vista puede especificarse como relativa a elementos del mismo nivel o en posiciones relativas al área *RelativeLayout* superior.
- EditText: permite al usuario añadir texto y además puede mostrar texto.
- TextView: es el elemento de la interfaz que muestra texto al usuario.
- CheckBox: esta casilla de verificación permite al usuario seleccionar una o más opciones de un conjunto.
- Button: elemento de la interfaz que tras tocarlo o hacer clic puede realizar una acción.
- ImageButton: muestra un botón con una imagen que el usuario puede presionar o hacer clic en él.
- FloatingActionButton: un botón de acción flotante (BAF) es un botón circular que activa la acción principal en la interfaz de usuario de la aplicación.
- RecyclerView: facilita la muestra de grandes conjuntos de datos de manera eficiente. Tras proporcionarle los datos y definir el aspecto de cada elemento la biblioteca *RecyclerView* creará los elementos de forma dinámica cuando los necesite.

Los objectos utilizados para diseñar la interfaz se componen de varios parámetros, los más utilizados en este proyecto se explicarán a continuación.

- Width: establece la anchura que ocupará un elemento en la interfaz.
- Height: define la altura que va ocupar un elemento en la pantalla.

- Orientation: define la orientación en la que se crearan los elementos, si se colocaran en filas o columnas.
- Padding: esta propiedad establece el espacio de relleno requerido por todos los lados de un elemento.
- LayoutManager: es un gestor de composición que permite desarrollar la interfaz de forma dinámica, con el objeto de ayudar a adaptar los diversos componentes que se desean incorporar a un panel.
- nestedScrollingEnabled: este parámetro ofrece la posibilidad al usuario de desplazarse verticalmente por la pantalla de la aplicación mostrando elementos que no caben en ella.
- Id: es el nombre de recurso único, cualquier objecto *View* puede tener un ID asociado para identificarse de forma única en la aplicación.
- Background: esta propiedad permite modificar el fondo de pantalla o el cambio de color en los botones.
- ButtonTint: permite cambiar el color de un botón o checkbox, entre otros.
- TextColor: permite cambiar el color de la letra del texto.
- TextSize: utilizado para modificar el tamaño de la letra del texto.

En las siguientes ilustraciones se mostrará el código de los archivos que definen la interfaz de usuario. Varios de los archivos *.xml* comparten notables similitudes, por ello, a continuación, están mostrados los más representativos.

Este archivo contiene el código de la actividad Splash que inicia la aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/black">

    <ImageView
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_centerInParent="true"
        android:src="@drawable/logo"/>

</RelativeLayout>
```

Ilustración 41 - activity_splash

El siguiente archivo muestra el código de la interfaz de la actividad principal.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageButton
        android:id="@+id/button1"
        android:layout_width="105dp"
        android:layout_height="48dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="0dp"
        android:layout_marginTop="0dp"
        android:background="@android:color/black"
        android:src="@drawable/ic_shopping_cart"/>

    <ImageButton
        android:id="@+id/button2"
        android:layout_width="105dp"
        android:layout_height="48dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="105dp"
        android:layout_marginTop="0dp"
        android:background="@color/colorAccent"
        android:src="@drawable/ic_refrigerador" />

    <ImageButton
        android:id="@+id/button3"
        android:layout_width="105dp"
        android:layout_height="48dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="210dp"
        android:layout_marginTop="0dp"
        android:background="@color/colorAccent"
        android:src="@drawable/ic_freezer" />

    <ImageButton
        android:id="@+id/button4"
        android:layout_width="105dp"
        android:layout_height="48dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:layout_marginStart="315dp"
        android:layout_marginTop="0dp"
        android:layout_marginEnd="0dp"
        android:background="@color/colorAccent"
        android:src="@drawable/ic_despensa"/>

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/listRecyclerView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="64dp"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        android:nestedScrollingEnabled="true"/>

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentEnd="true"
        android:layout_margin="32dp"
        android:backgroundTint="@android:color/black"
        android:src="@drawable/ic_add"/>

</RelativeLayout>
```

Ilustración 42 - activity_main

La siguiente ilustración pertenece al archivo que genera el diálogo emergente para añadir un producto nuevo, en este caso, en la actividad Despensa.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp">

    <EditText
        android:id="@+id/newItemText3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:hint="Añade un producto" />

    <EditText
        android:id="@+id/newItemQuantity3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/newItemText3"
        android:layout_centerInParent="true"
        android:hint="Cantidad"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium" />

    <TextView
        android:id="@+id/fechas3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/newItemQuantity3"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"/>

    <Button
        android:id="@+id/caducidad3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/fechas3"
        android:text="Fecha de caducidad"
        android:textAllCaps="false" />

    <Button
        android:id="@+id/newItemButton3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/caducidad3"
        android:textSize="16sp"
        android:layout_alignParentEnd="true"
        android:background="@android:color/transparent"
        android:text="Guardar"
        android:textAllCaps="false"
        android:textColor="@android:color/darker_gray" />

</RelativeLayout>
```

Ilustración 43 - new_itempastry

El siguiente archivo de código contiene los objectos que definen el diálogo que se muestra a la hora de añadir un producto, en este caso, sin la fecha de caducidad.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp">

    <EditText
        android:id="@+id/newItemText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:hint="Añade un producto" />

    <EditText
        android:id="@+id/newItemQuantity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:layout_marginBottom="25dp"
        android:hint="Cantidad"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium" />

    <Button
        android:id="@+id/newItemButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/newItemQuantity"
        android:textSize="16sp"
        android:layout_alignParentEnd="true"
        android:background="@android:color/transparent"
        android:text="Guardar"
        android:textAllCaps="false"
        android:textColor="@android:color/darker_gray" />

</RelativeLayout>
```

Ilustración 44 - new_item

La siguiente ilustración configura la disposición de los elementos en la actividad Lista de la Compra.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    app:cardElevation="4dp"
    app:cardCornerRadius="8dp"
    android:layout_marginHorizontal="16dp"
    android:layout_marginVertical="8dp">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="8dp">

        <CheckBox
            android:id="@+id/listCheckBox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingStart="8dp"
            android:buttonTint="@android:color/black"
            tools:text="Sample Text."/>

        <TextView
            android:id="@+id/quantitylist"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_alignParentEnd="true"
            android:layout_marginTop="6dp"
            android:layout_marginEnd="6dp"
            android:background="@android:color/transparent"
            android:textColor="@android:color/black"/>

    </RelativeLayout>

</androidx.cardview.widget.CardView>
```

Ilustración 45 - item_layout

El siguiente archivo define la interfaz de la pantalla Nevera.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    app:cardElevation="4dp"
    app:cardCornerRadius="8dp"
    android:layout_marginHorizontal="16dp"
    android:layout_marginVertical="8dp">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="80dp"
        android:padding="8dp">

        <TextView
            android:id="@+id/itemNevera"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingStart="8dp"
            android:textColor="@android:color/black"
            tools:text="Sample Text."/>

        <TextView
            android:id="@+id/quantityNevera"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="300dp"
            android:background="@android:color/transparent"
            android:textColor="@android:color/black"/>

        <TextView
            android:id="@+id/cantidad"
            android:layout_width="70dp"
            android:layout_height="20dp"
            android:text="Cantidad: "
            android:layout_marginStart="210dp"
            android:textColor="@color/colorPrimary"/>

        <TextView
            android:id="@+id/fechas"
            android:layout_width="180dp"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentBottom="true"
            android:paddingStart="8dp"
            android:layout_marginBottom="6dp"
            android:textColor="@android:color/black"/>

        <Button
            android:id="@+id/buttonfreezer"
            android:layout_width="24dp"
            android:layout_height="24dp"
            android:layout_alignParentStart="true"
            android:layout_alignParentBottom="true"
            android:layout_marginStart="210dp"
            android:layout_marginBottom="6dp"
            android:background="@drawable/ic_blackfreezer"/>
        <Button
            android:id="@+id/buttoncart"
            android:layout_width="24dp"
            android:layout_height="24dp"
            android:layout_alignParentBottom="true"
            android:layout_marginStart="270dp"
            android:layout_marginBottom="6dp"
            android:background="@drawable/ic_blackcart" />

        <Button
            android:id="@+id/buttonmore"
            android:layout_width="24dp"
            android:layout_height="24dp"
            android:layout_alignParentBottom="true"
            android:layout_marginStart="330dp"
            android:layout_marginBottom="6dp"
            android:background="@drawable/ic_blackmorevert" />

    </RelativeLayout>

</androidx.cardview.widget.CardView>
```

Ilustración 46 - nevera_layout

Entre los archivos XML, en este proyecto se han definido además cuatro menus *PopUp*, que son los menus emergentes que aparecen al hacer clic en los botones de los tres puntos verticales, el carrito de la compra, el congelador y la nevera.

En las siguientes ilustraciones se muestra el código utilizado:

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/item_update"
        android:title="Editar" />

    <item
        android:id="@+id/item_del"
        android:title="Eliminar" />

</menu>
```

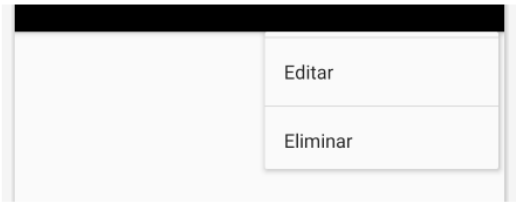

Ilustración 47 - pop_up_menu

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/item_tofreezer"
        android:title="Mover al Congelador" />

    <item
        android:id="@+id/item_cancelar"
        android:title="Cancelar" />

</menu>
```

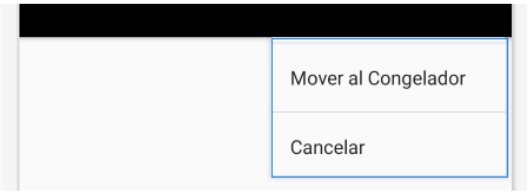

Ilustración 48 - popup_freezer

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/item_tolista"
        android:title="Añadir a la lista de la compra" />

    <item
        android:id="@+id/item_cancelar"
        android:title="Cancelar" />

</menu>
```

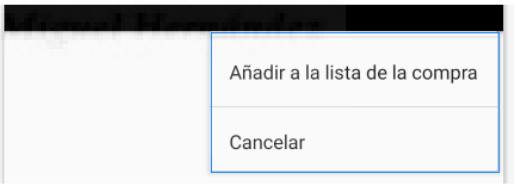

Ilustración 49 - popup_lista

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/item_tonevera"
        android:title="Mover a la Nevera" />

    <item
        android:id="@+id/item_cancelar"
        android:title="Cancelar" />

</menu>
```

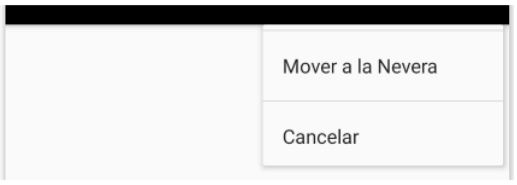

Ilustración 50 - popup_nevera

8. MONETIZACIÓN DE LA APLICACIÓN

En este apartado se enumeran posibles maneras de rentabilizar la aplicación. Cabe destacar que FridgeTracker se ha desarrollado sin ánimo de lucro y no se espera obtener beneficio alguno con esta aplicación.

A la hora de monetizar una aplicación existen diversas formas de lograrlo. Para comenzar, se podrían dividir las apps en dos tipos, las de descarga gratuita y las de pago. Una vez se decide qué tipo de app va ser, gratis o de pago, existen varios modelos para rentabilizar la aplicación según se ajuste al modelo de negocio establecido.

A continuación, se enumeran algunos de los modelos que podrían ajustarse al tipo de app desarrollada en este proyecto.

- Freemium: Gratis + Oportunidades de Monetización Adicional.
  Este modelo se basa en apps de descarga gratuita en el que una vez se fideliza a los usuarios, se ofrece contenido adicional al que solo se puede acceder mediante pago.
- Pago por descarga:
  Como indica el nombre se paga para poder descargar la app y automáticamente se puede acceder a todas sus funcionalidades. Sin embargo, se ha de tener en cuenta que este tipo de apps representan un pequeño porcentaje del mercado, ya que presentan una barrera de entrada respecto a las gratuitas.
- Paidmium: De pago + Oportunidades de Monetización Adicionales.
  Este tipo de apps requieren el pago por descarga y, además, ofrecen contenido adicional de pago. La monetización se centra en compras in-app o mejoras en la experiencia de usuario.
- Suscripción:
  Este modelo requiere de un pago regular, que suele ser mensual o anual, y que da acceso a los servicios, contenido o mejoras de la aplicación.

- Compras In-App:

  Estas apps suelen ser de descarga gratuita y posibilitan la compra de productos o servicios a través de ella.

- Publicidad In-App:

  La publicidad in-app es uno de los mecanismos que más terreno está ganando como modelo de monetización. Como su nombre muestra, consiste en mostrar anuncios o *ads* en la aplicación para rentabilizarla.

- Afiliate Marketing:

  Este método de monetización consiste en promocionar productos o servicios de otras empresas dentro de la app a cambio de recibir una comisión cuando un usuario se descarga la app, compra o se compromete con un producto o servicio.

- Sponsorship:

  Consiste en la búsqueda de inversores que financien la creación de la app para convertirla en un negocio, por ejemplo, se puede ofrecer a los inversores publicidad gratuita dentro de la app.

- Mobile Marketing Automation:

  Este método consiste en coordinar y enviar campañas de marketing a través de canales móviles como notificaciones push, mensajes in-app y campañas de e-mail. El objetivo principal es convencer al usuario para que realice compras in-app o se subscriba a un servicio que proporcione contenido adicional.

Para la aplicación desarrollada, FridgeTracker, las opciones de monetización que podrían ser más sencillas de implementar al principio serían la del modelo Freemium y la de publicidad in-app.

Utilizando la publicidad in-app se puede rentabilizar la aplicación introduciendo *ads* o anuncios de los comerciantes interesados. Además, este modelo puede trabajar en conjunto con el modelo Freemium, donde la versión de pago podría eliminar esos anuncios.

Con el modelo Freemium, la app sería de descarga gratuita y como parte de la versión de pago, se podrían añadir las siguientes funcionalidades o beneficios adicionales:

- Eliminación por completo de anuncios en la app.
- Poder compartir las listas creadas con un mayor número de usuarios.
- La opción de personalizar colores o ciertos iconos en las listas creadas.
- Personalizar las notificaciones cuando un alimento está cercano a su fecha de caducidad.

9. RESULTADOS Y DISCUSIÓN

Para finalizar, una vez terminada tanto la redacción de este documento escrito como el desarrollo de la aplicación de este trabajo, se pueden extraer varias conclusiones interesantes que ponen de manifiesto el estudio y trabajo llevado a cabo a la hora de realizar este proyecto, y que permitirán obtener una capacidad de desarrollo más precisa y ágil al enfrentarse a proyectos similares en un futuro.

Al comenzar el proyecto se ha tenido como primer objetivo familiarizarse con el lenguaje de programación JAVA y obtener un conocimiento robusto del sistema operativo sobre el que se ejecuta la aplicación desarrollada. Con este fin en mente se ha realizado un análisis sobre su arquitectura interna, de los elementos que constituyen el proyecto y de los componentes necesarios para una aplicación.

Durante el desarrollo de la aplicación se ha buscado siempre que la experiencia del usuario sea satisfactoria. Para ello la interfaz de la app ha sido diseñada para proporcionar un producto visual intuitivo, cuyos controles sean familiar a los que el usuario está acostumbrado a utilizar en Android, obteniendo así un producto sencillo de usar, que permita al usuario moverse por la aplicación de manera rápida y dinámica evitando que se pierda en numerosas pantallas o sobrecargando la interfaz con elementos innecesarios.

Desde mi punto de vista personal he de comentar que ha sido muy satisfactorio poder desarrollar un proyecto software partiendo desde cero y de estas características. Este trabajo me ha permitido poner en práctica varios conceptos y conocimientos que he ido adquiriendo durante los cursos de este grado y de mis proyectos personales y poder implementar dichos conceptos a un campo desconocido hasta ahora como es la programación.

10. PROPUESTAS DE MEJORA

En este apartado se detallan diferentes actualizaciones y mejoras que me gustaría implementar y que opino mejorarían la usabilidad y diseño de la aplicación. Con el objetivo de obtener un producto más trabajado, de mayor complejidad y más eficiente a la hora de realizar la tarea para la que esta inicialmente diseñado, la gestión eficiente de los alimentos en el hogar y evitar el malgasto de estos.

A continuación, se muestran mis propuestas de mejora:

- Añadir la opción de eliminar varios elementos seleccionados al mismo tiempo en lugar de individualmente.
- Crear una barra de búsqueda de productos para cada pantalla, permitiendo al usuario acceder a la información deseada de manera más rápida.
- Implementar filtros y dividir los productos en cada pantalla en secciones que agrupen los tipos de alimentos a los que pertenecen, para poder localizarlos y observar al instante los contenidos de la nevera, congelador o despensa.
- Añadir notificaciones cuando los alimentos se aproximen a su fecha de caducidad
- Crear una base de datos con recetas que proporcione ideas al usuario de como consumir productos que se acerquen a su fecha de caducidad evitando así que se desperdicien estos alimentos.
- Mejorar la interfaz gráfica, optimizando la información mostrada e incrementando la fluidez con la que se mueve el usuario entre pantallas e interactúa con los elementos de la aplicación.
- Posibilitar la opción de compartir las listas con miembros de la unidad familiar o amigos.

Durante el proceso de implementar estas mejoras se deberá realizar, no obstante, una investigación de como las funciones añadidas modifican la experiencia del usuario. Se requerirá probar la aplicación con diversos usuarios que tendrán que realizar varias tareas e indicar que les ha parecido complejo o difícil de usar, si están satisfechos con el producto, modificarían alguna parte, y sobre todo si utilizarían esta aplicación.

11. BIBLIOGRAFÍA Y REFERENCIAS

Desarrolladores de Android:
https://developer.android.com/

Abhi Android:
https://abhiandroid.com/

LCSI – Laboratorio de Computación Social e Interfaces Móviles:
http://lcsi.umh.es/androidstudio/

Stack Overflow:
https://stackoverflow.com/

Android:
https://www.android.com/

Adobe XD:
https://www.adobe.com/es/products/xd.html

Statista:
https://www.statista.com/

Wikipedia, Android:
https://es.wikipedia.org/wiki/Android

Wikipedia, Android Studio:
https://es.wikipedia.org/wiki/Android_Studio

Google Play
https://play.google.com/store