



MÓDULO PROYECTO

Ciclo Superior Desarrollo de Aplicaciones Multiplataforma

Departamento: Informática y Comunicaciones

IES “María Moliner”

Curso: 2022/2023

Grupo: S2MD

Proyecto: Lista de la compra

Roberto Rodríguez Jiménez

Email: roberto.rodjim.1@educa.jcyl.es

Tutor individual: Esther Álvarez Cortés

Tutor colectivo: Rocío Manso Gil

Fecha de presentación: (18 de mayo de 2023)

Contenido

1	Descripción general del proyecto	6
1.1	Justificación.....	6
1.2	Introducción.....	8
1.3	Objetivos	11
1.4	Metodología	13
1.4.1	Esquema.....	13
1.4.2	Fases	14
1.5	Entorno.....	15
2	Descripción general del producto	17
2.1	Funcionalidades	18
2.1.1	Lista de la compra.....	18
2.1.2	Listado de listas de la compra.....	19
2.1.3	Almacén de productos	19
2.1.4	Otras funcionalidades	21
2.2	Límites de la aplicación	22
2.3	Estructura de la aplicación	23
2.3.1	Pantallas principales	23
2.3.2	Componentes principales de la interfaz de usuario	25
2.3.2.1	Inicio	27
2.3.2.2	Lista.....	28
2.3.2.3	Compras.....	29
2.3.2.4	Almacén	29
3	Documentación técnica	31
3.1	Creación del proyecto	31
3.2	Estructura.....	31
3.3	Despliegue	36

3.3.1	Generar APK.....	36
3.3.2	Instalar el APK	38
3.4	Origen de los datos	41
3.4.1	Elección del SGBD	41
3.4.2	Datos del producto.....	43
3.4.3	Datos de la compra.....	44
3.4.4	Normalización 3FN	45
3.4.5	Otros datos	47
3.4.6	Room database.....	48
3.4.6.1	Entidades	48
3.4.6.1.1	VistaCompra	49
3.4.6.2	DAO	50
3.4.6.3	Conexión con la base de datos	51
3.5	Backup remoto	52
3.5.1	Exportar datos.....	52
3.5.2	Importar datos.....	54
3.6	Interfaz de usuario.....	55
3.6.1	Inicio	56
3.6.2	Lista abierta	58
3.6.2.1	Lista.....	58
3.6.2.2	Detalle de la compra	59
3.6.2.2.1	Información	59
3.6.2.2.2	Comparativa.....	60
3.6.2.2.3	Evolución	61
3.6.2.3	Comparativa de precios por comercio	62
3.6.3	Listado de compras.....	63
3.6.4	Almacén.....	64

3.6.4.1	Almacén lista	64
3.6.4.2	Almacén detalle	65
3.6.5	Ayuda	66
3.7	Funcionamiento	67
3.7.1	Inicio	67
3.7.2	Lista de la compra (lista abierta) 	68
3.7.2.1	Cargar la lista de la compra	69
3.7.2.2	Adaptador de la lista abierta	70
3.7.2.3	Añadir un producto a la lista	71
3.7.2.4	Eliminar producto de la lista	72
3.7.2.5	Detalle	73
3.7.2.5.1	Producto Info	74
3.7.2.5.2	Producto comparativa	77
3.7.2.5.3	Producto evolución	80
3.7.3	Listados	83
3.7.3.1	Crear un listado	83
3.7.3.2	Duplicar un listado 	84
3.7.3.3	Eliminar un listado	85
3.7.3.4	Cambiar la fecha de una lista 	86
3.7.4	Almacén	88
3.7.4.1	Agregar un producto	88
3.7.4.1.1	Insertar un producto a granel	89
3.7.4.1.2	Insertar otros datos 	90
3.7.4.1.3	Proceso de guardado	92
3.7.4.2	Eliminar un producto del almacén	94
3.7.4.3	Asociar etiquetas a productos	95
3.7.5	Envío de la lista por mensajería	96

3.7.6 Utilidades	98
3.7.7 Sistema de ayuda	101
3.7.7.1 Video-tutoriales	103
3.8 Diseño multi-Screen	105
4 Manuales de usuario	107
4.1 Manual de usuario	107
4.1.1 Inicio de la aplicación	108
4.1.2 Listas	109
4.1.2.1 Crear una lista	109
4.1.2.2 Borrar una lista	109
4.1.2.3 Reutilizar una lista (duplicar)	110
4.1.2.4 Modificar la fecha de una lista	110
4.1.3 Productos (almacén)	111
4.1.3.1 Agregar un producto	111
4.1.3.2 Buscar un producto	112
4.1.3.3 Editar un producto	112
4.1.3.4 Eliminar un producto	112
4.1.4 Lista de la compra	113
4.1.4.1 Crear una lista de la compra	113
4.1.4.2 Abrir una lista de la compra	113
4.1.4.3 Añadir un producto a la lista	113
4.1.4.4 Eliminar un producto de la lista	113
4.1.4.5 Editar la cantidad y el precio del producto	114
4.1.4.5.1 Aplicar ofertas	114
4.1.4.6 Comparativa de precios	114
4.1.4.7 Evolución del precio	115
4.1.4.8 Compartir una lista de la compra	116

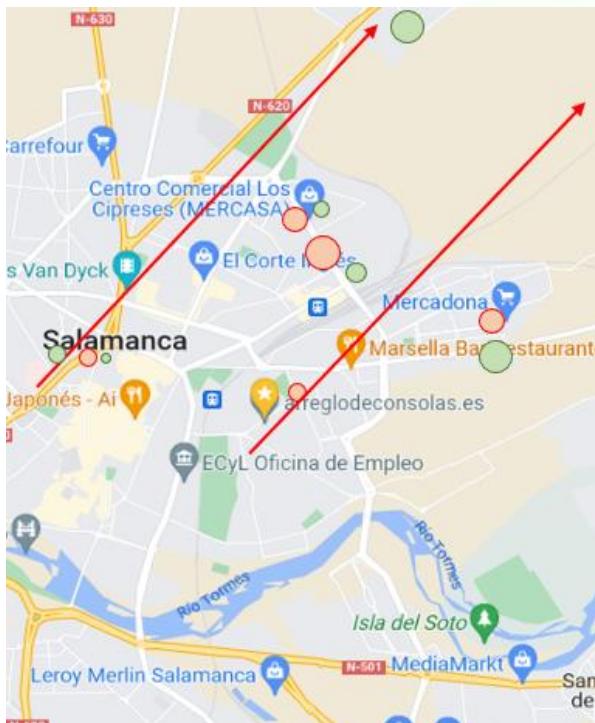
4.1.4.9	Comparar la lista en otros comercios	116
4.1.5	Crear un nuevo comercio.....	117
4.1.6	Asociar comercio – marca blanca	117
4.1.7	Copias de los datos	118
4.1.7.1	Guardar una copia de los datos	118
4.1.7.2	Recuperar una copia de los datos.....	119
4.1.7.3	Importar datos de ejemplo.....	120
4.1.7.4	Exportar datos de ejemplo.....	121
4.2	Manual de instalación.....	122
5	Conclusiones y posibles ampliaciones	124
5.1	Ampliaciones	125
6	Referencias	126

1 Descripción general del proyecto

1.1 Justificación



El rápido encarecimiento de los precios, durante los últimos años, nos lleva a buscar la forma de optimizar el gasto a la hora de comprar los productos habituales.



Asiduidad a los comercios habituales

- Comercios habituales antes del uso de la comparativa.
- Comercios actuales, una vez habituados a los precios.
- Ruta natural.

Normalmente, la compra se realiza en un mismo establecimiento, ya sea por precio, calidad, cercanía o, simplemente, capricho.

Un rápido vistazo a la asiduidad a los comercios nos hace ver como repetimos las compras un día tras otro.

Necesitamos una forma de tener presente los precios de los productos y en qué establecimiento se compraron, para poder decidir si podemos realizar una compra sin importarnos dónde estemos, sin la duda de si un comercio será más rentable que otro.

Del mismo modo que repetimos establecimiento, los productos que compramos para el consumo diario suelen ser los mismos, de una determinada marca y, probablemente, formato.

Los precios de los productos pueden resultar confusos debido al formato en el que se

presentan: diferentes tamaños o cantidad de un mismo producto a diferentes precios.

Es posible ver "packs de ahorro" a precio superior a su versión individual.



4,58 €

1,53 €/100ml

Champú anticaspa Citrus Fresh H&S 300 ml.



8,68 €

1,45 €/100ml

Champú anticaspa Citrus Fresh H&S 600 ml.



13,95 €

1,39 €/100ml

Champú anticaspa Citrus Fresh H&S 1000 ml.

Algo confuso lo podemos encontrar el en champú de la marca H&S, con envases de 90, 300, 360, 400, 480, 540, 600, 700, 1000ml ... con precios que hemos visto desde los 11.80€ hasta 15.30€ para el mismo producto y la misma variedad.

Nos vendría bien llevar un listado de

Los precios varían dependiendo de la presentación del producto

los precios relativos a su cantidad.

1.2 Introducción



Echo en falta una aplicación que me recuerde a qué precios he comprado últimamente los productos de mi cesta habitual, y en qué comercio lo hice. No puedo recordar si el pollo lo compré a 6 €/kg aquí o allí.

Necesito un poco de perejil, picado, en un pequeño tarro.

Acabo de encontrar uno: 1,99€, bien barato, aunque, mirándolo bien, son 40gr. El kilo de perejil sale a ¡49,75 €/kg!



ESPECIA ALTEZA PEREJIL HOJA 40 GRS.

1,99 €

El Kilo sale a: 49,75 €

1

Añadir al Carrito



Un precio aparentemente bajo puede ocultar el precio real del producto.

Existen muchas aplicaciones que guardan una lista de productos para comprar e, incluso, de los propios establecimientos en las cuales se marcan los precios. La necesidad de una aplicación personal, que incluya mis productos en mis comercios, motiva la creación de la aplicación.



AJO FINDUS TROCEADO 75 GRS

2,20 €

El Kilo sale a: 29,33 €



AJOS GRANEL, KILO

4,79 € /Kg

aprox. 7 unidades/Kg

El ajo troceado puede resultar cómodo, pero, por este precio, es mejor picarlo uno mismo.

Aunque el proyecto se llame “Lista de la compra”, realmente pretende monitorizar el gasto que se hace en la compra diaria. Saber en qué gastamos de más y en qué podemos ahorrar. Es sorprendente ver cómo artículos que nos parece tener un precio asequible, son un auténtico despilfarro.

Una aplicación de este tipo no tiene por qué limitarse a ser un listado de productos y sus precios, debe tratarse de una colección de datos con los que será posible optimizar las compras habituales. Se guardarán productos, fechas, precios, comercios, cantidades...



Precio confuso en el que se destaca la versión más cara sin que apenas pueda verse su precio

Sería una buena opción poder elegir que tipo de producto voy a comprar en función de la relación precio/cantidad, pues solemos comprar fijándonos tan solo en precio. Los comercios no ayudan y colocan los precios relativos a la cantidad en tamaños que apenas pueden verse.

Otra ayuda importante a la hora de elaborar un listado de productos es poder repetir cualquier otra lista ya realizada, editando y actualizando los precios o la cantidad comprada. Sería posible añadir o eliminar otros productos.

Tampoco estaría de más poder comparar la lista de la compra, realizada para un establecimiento, con los precios, ya almacenados, de los mismos productos, pero en otros comercios.

Aunque, aparentemente, no son demasiados los datos guardados, podemos aún sacar más provecho y mostrar los precios de productos relacionados, para poder optar por otros diferentes a los habituales.

El análisis de los datos da para mucho más, por lo que la aplicación es tan solo un primer paso para, principalmente, obtener los datos. Estos datos son usados para ofrecer opciones en la elaboración de una lista de la compra, pero pueden usarse para realizar informes sobre la evolución (real) de los precios, obtener las cadenas más económicas, comprobar las diferencias entre los precios de los comercios tradicionales y las cadenas de supermercados, ...

Una idea, que quedará en el aire, y que es la que más me gusta, es la de calcular los precios de los productos en relación con los ingresos regulares. En lugar de que un litro de leche cueste 1€ (p.ej.), pasaría costar un 0,1% para unos ingresos de 1.000€/mes.

La opción de compartir la lista entre diferentes usuarios está abierta, pero, al tratarse de una aplicación de uso personal, he optado por un almacenamiento local de los datos para garantizar su disponibilidad aún sin conexión a internet. Además, el acceso de otros usuarios a los datos podría desvirtuarlos.

1.3 Objetivos



- Almacenar productos comprados habitualmente, indicando la cantidad de producto que contiene, además de la magnitud de su medida, ya sea en kilogramos, litros, unidades, etc.

Los productos se referencian por el código de barras EAN-13 o EAN-8, dejando que la aplicación asigne códigos automáticamente a los productos que se compran a granel o de los que se desconoce el código de barras.

Cada producto tiene como etiqueta asociada, al menos, las palabras que forman su denominación, pudiendo agregarse etiquetas adicionales.

También llevará, cada uno de los productos, asociada la marca. Las marcas se registrarán en una tabla auxiliar de la base de datos, de modo que no puedan repetirse. Es posible guardar productos sin marca como, por ejemplo, los comprados a granel.

Podrá asignarse una imagen del producto, que tendrá por nombre el código de barras y como extensión “jpg”, pero deberá obtenerse y alojarse en el servidor de forma manual. Las imágenes no se almacenarán en el dispositivo, por razones de espacio.

El único requisito para almacenar un producto es la denominación, pues prevalece la agilidad en el uso de la aplicación, pudiendo completarse posteriormente.

- Crear una lista de productos para poder comprar en un establecimiento.

Esta lista debe incluir el precio, la cantidad y el total de cada producto, así como el total de la compra.

El listado guarda la fecha y el establecimiento.

La lista podrá duplicarse para ser reutilizada.

- Compartir la lista a través de aplicaciones de mensajería instantánea, como WhatsApp o Telegram.

- Mostrar una comparativa de la compra con la compra de los mismos productos en otros establecimientos.

Para realizar la comparativa, se usará la última compra del producto en cada comercio. Como es probable que las listas no estén completas, ya que hay marcas que no pueden ser compradas en establecimientos diferentes al suyo, ha de indicarse el número de productos que se incluyen en la lista, además de mostrarlos.

También se mostrará la fecha en la que fue comprado cada artículo para poder tener una referencia temporal.

- Aplicar diferentes, predefinidas, a los precios de productos.

El precio resultante de aplicar la oferta debe reflejarse en el precio de la compra del producto. Además, debe registrarse que esa compra se hizo bajo una oferta.

- Comparar el producto comprado con la compra en otro establecimiento. Debe compararse, además, con otros productos con los que comparte la etiqueta principal (primera).

- En la comparativa de los productos relacionados, el precio debe mostrarse relativo a la cantidad.

- Mostrar, mediante un gráfico, la evolución del precio de un producto a lo largo del tiempo.

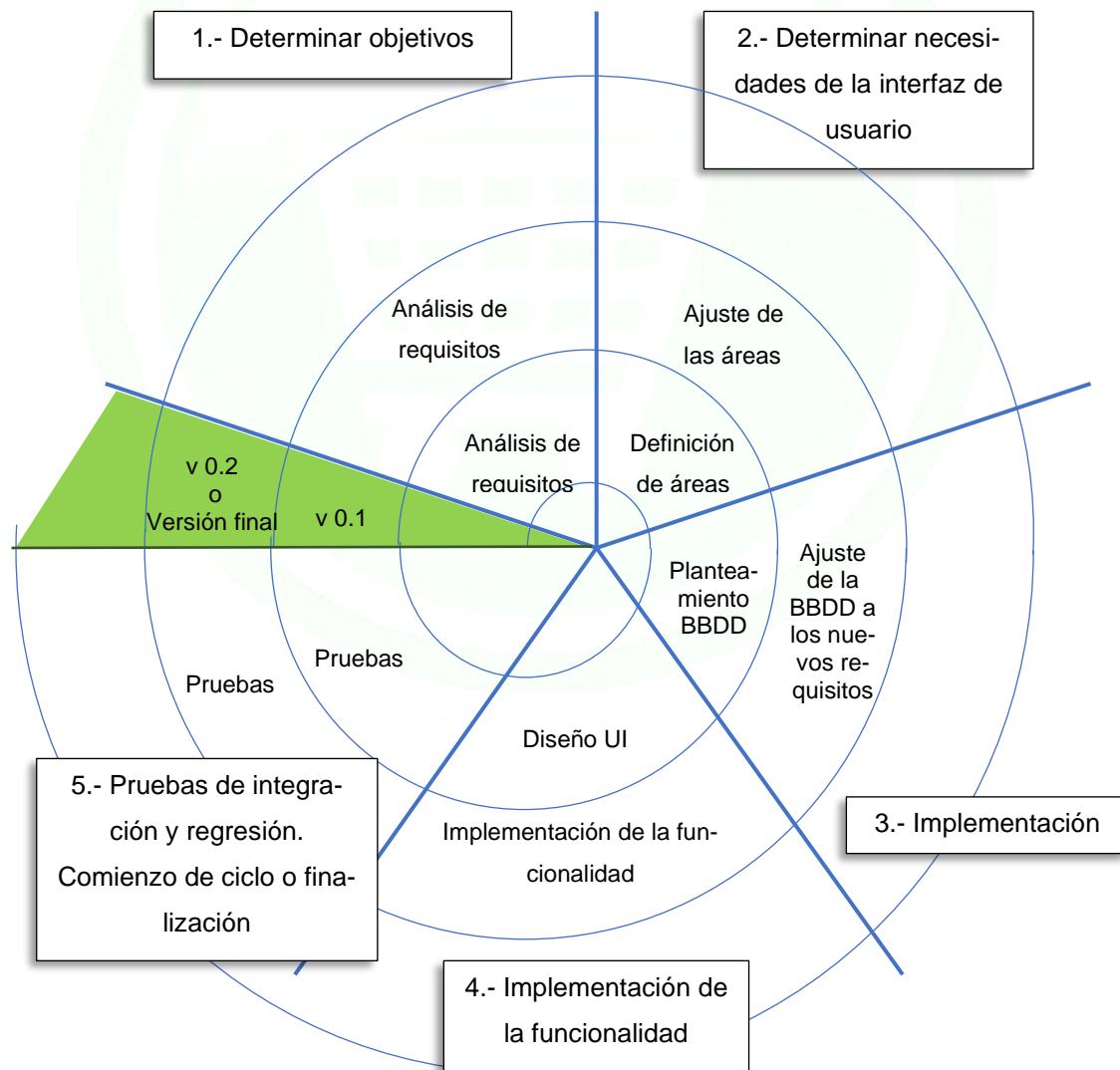
- Insertar el código de barras de un producto mediante la captura a través de la cámara del dispositivo.

1.4 Metodología



Se va a usar un desarrollo en espiral, con ciclos en cascada, de forma que, al final de cada ciclo, tengamos una aplicación funcional. En cada uno de los ciclos se añadirá el objetivo siguiente, pudiendo añadir alguna funcionalidad extra que pudiera necesitarse.

1.4.1 Esquema



1.4.2 Fases



En cada uno de los ciclos se repetirán las siguientes fases:

- Definición de requisitos.

Debemos tener claro qué debe hacer la aplicación en la fase.

- Determinar las áreas en las que se va a trabajar.

Hay dos áreas independientes fundamentales: el almacén y la lista.

- El almacén gestiona los productos y sus detalles.
- La zona de listas lleva a cabo la creación de las listas y su gestión.

En ciclos sucesivos aparecerán nuevas áreas, como puede ser la actividad de ayuda o la de créditos.

- Desarrollo de la base de datos.

En función de las necesidades que vayamos encontrando, deberemos ajustar la base de datos hasta quedar completada.

Toda la gestión de la base de datos debe hacerse desde la aplicación.

- Implementación de la funcionalidad de la base de datos.

La base de datos debe proporcionar los métodos necesarios para obtener los datos.

En cada ciclo habrá que implementar nuevas funciones dependiendo de las necesidades.

- Implementación de los requisitos.

En esta fase la aplicación debe responder correctamente a los requisitos definidos al inicio del ciclo.

Al finalizar cada ciclo deben realizarse las pruebas necesarias.

1.5 Entorno

Plataforma: Android (SDK 32, mínimo 24) Android 7.0 (diciembre 2016)

Lenguaje: Java

SO: Windows 10

Equipo: Intel Core i7, 1.30 GHz, RAM 32GB, 4 procesadores (8 lógicos)

Máquina virtual: VirtualBox, 4 hilos de ejecución, 16GB RAM

IDE: Android Studio Dolphin 2021.3.1 / Flamingo 2022.2.1

Orientación de la pantalla: se ha optado por una visualización en *portrait* debido a que la aplicación está orientada a dispositivos de mano, normalmente no superiores a 7". La vista en *landscape* para menos de 7" dificulta la visualización de la información. Aún así, se ha implementado el sistema de ayuda en *multiscreen*, adaptándose a al tipo de dispositivo y la orientación de la pantalla.

Equipos utilizados: Durante todo el desarrollo del proyecto se han usado dispositivos físicos.

	Samsung Galaxy S21	Samsung Galaxy S7	Samsung Galaxy Tab S6
S.O.	Android 13	Android 8.0	Android 13
Pantalla	6.2"	5.1"	10.5"
Resolución	2400x1080	1440x2560	2560x1600
Procesador	3x ARM-A78 Cortex @ 2,81 GHz 4x ARM Cortex-A55 @ 2,21 GHz 1x ARM Cortex-X1 @ 2,91 GHz	4x Samsung Exynos M1 @ 2,60 GHz 4x ARM Cortex-A53 @ 1,59 GHz	4x ARM Cortex-A73 @ 2,31 GHz 4x ARM Cortex-A53 @ 1,74 GHz
RAM	8GB	4GB	6GB

Tabla de especificaciones de los dispositivos usados durante el desarrollo del proyecto.

Nomenclatura del APK: 1.1.2023.mm.ddd

1.1: Número de versión que se considera estable.

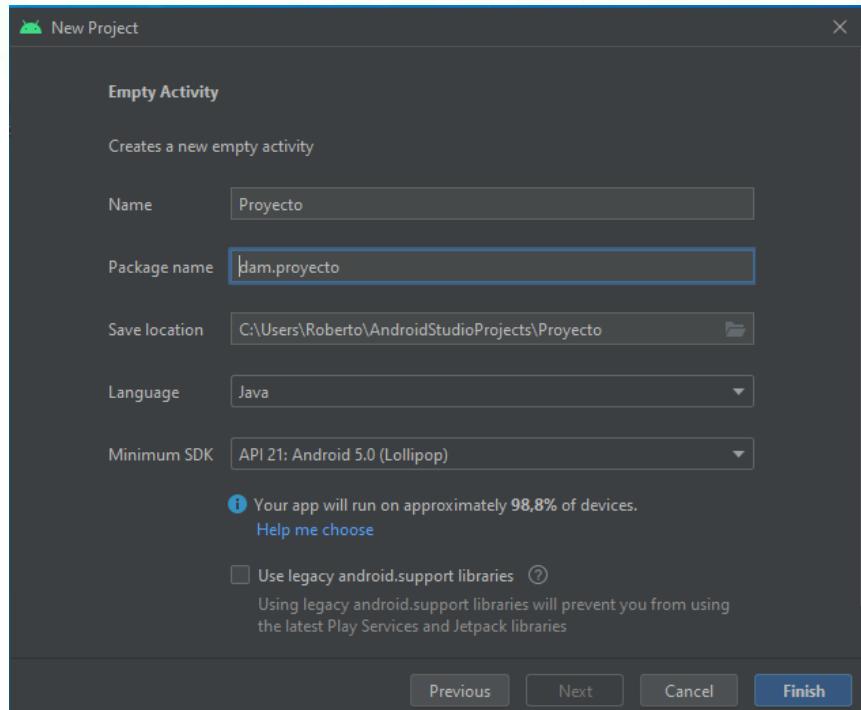
La versión 1 se creó cuando se consideró que la aplicación era funcional y cumplía los requisitos básicos.

Los números siguientes se corresponden con la fecha en la que lanza la actualización. En un mismo día se pueden generar dos APK diferentes, por lo que se añadiría una letra minúscula correlativa a la anterior, empezando por la b, pues la primera versión del mismo día no tendría letra.



Diferentes archivos APK generados.

El SDK inicial era el 21, pero se actualiza a 32 y se requiere mínimo 24 para poder hacer uso de algunos componentes.

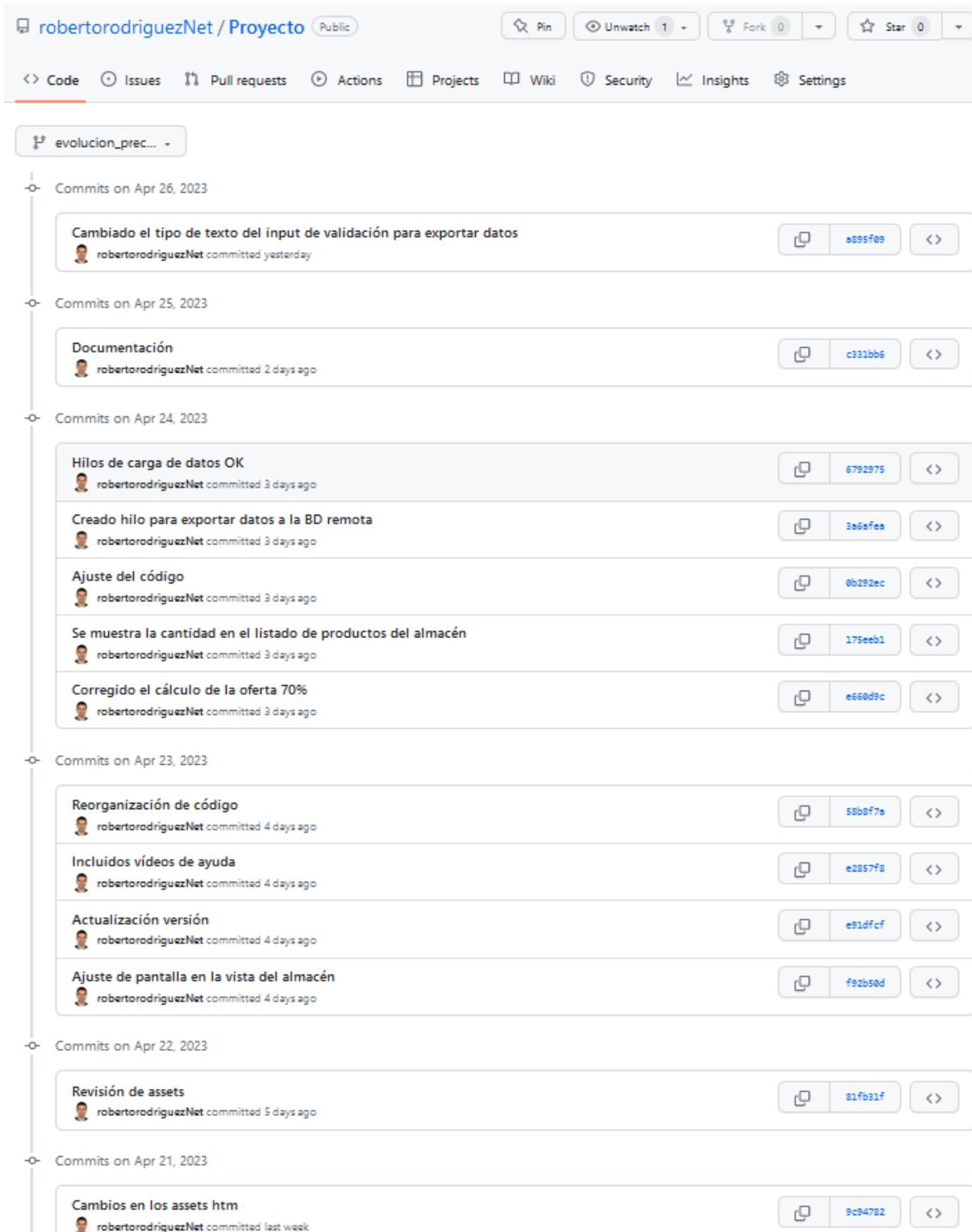


Creación del proyecto, en Java, para Android 5.0

1.6 El repositorio en GitHub

En GitHub puede seguirse la evolución del proyecto día a día.

<https://github.com/robertorodriguezNet/Proyecto>



The screenshot shows the GitHub repository interface for 'robertorodriguezNet/Proyecto'. The 'Code' tab is selected. A search bar at the top contains the text 'evolucion_prevencion'. The commit history is displayed in a tree view:

- o- Commits on Apr 26, 2023
 - Cambiado el tipo de texto del input de validación para exportar datos
robertorodriguezNet committed yesterday
- o- Commits on Apr 25, 2023
 - Documentación
robertorodriguezNet committed 2 days ago
- o- Commits on Apr 24, 2023
 - Hilos de carga de datos OK
robertorodriguezNet committed 3 days ago
 - Creado hilo para exportar datos a la BD remota
robertorodriguezNet committed 3 days ago
 - Ajuste del código
robertorodriguezNet committed 3 days ago
 - Se muestra la cantidad en el listado de productos del almacén
robertorodriguezNet committed 3 days ago
 - Corregido el cálculo de la oferta 70%
robertorodriguezNet committed 3 days ago
- o- Commits on Apr 23, 2023
 - Reorganización de código
robertorodriguezNet committed 4 days ago
 - Incluidos vídeos de ayuda
robertorodriguezNet committed 4 days ago
 - Actualización versión
robertorodriguezNet committed 4 days ago
 - Ajuste de pantalla en la vista del almacén
robertorodriguezNet committed 4 days ago
- o- Commits on Apr 22, 2023
 - Revisión de assets
robertorodriguezNet committed 5 days ago
- o- Commits on Apr 21, 2023
 - Cambios en los assets html
robertorodriguezNet committed last week

Captura de pantalla del repositorio en GitHub

2 Descripción general del producto



2.1 Funcionalidades

2.1.1 Lista de la compra

La primera de las funcionalidades, pero no más importante que cualquier otra, es poder crear una lista de la compra, a partir de los productos guardados, en la que mostrar los precios y desde la que acceder a otros detalles relacionados con los productos.

Funcionalidades relacionadas con los productos de la lista:

- Mostrar el listado de productos que se asocian a una lista abierta y a un comercio.
- El comercio asociado a la compra puede cambiarse en cualquier momento.
- Cada producto de la lista muestra la denominación, las unidades compradas, el precio y subtotal. También muestra una imagen descriptiva del producto, si ha sido guardada.
- Eliminar el producto de la lista, con click largo sobre el mismo.
- Aplicar, sobre el precio marcado, algunas de las ofertas más comunes (2x1, 3x2, ...).
- Comparar las últimas compras del producto seleccionado en diferentes comercios.
- Comparar las últimas compras de productos relacionados con el producto seleccionado, sin importar el comercio. El precio mostrado es el relativo a la cantidad, para facilitar la comparación.
- Mostrar en gráfico con la evolución del precio del producto, en cualquier comercio.
- Envío de la lista de la compra por mensajería (WhatsApp).
- Comparativa de la lista de la compra realizada en diferentes comercios. Para que los productos aparezcan en otros comercios, deben existir las compras. Le listado de cada comercio informa de la cantidad de productos que se han encontrado y de la fecha de su compra.

2.1.2 Listado de listas de la compra

El listado de listas es otra de las tres actividades principales. En ella se muestran las diferentes listas de las compras que se han ido creando. Desde esta actividad se puede:

- Cada una de las listas muestra el nombre (o identificador si no se le dio), la fecha y hora, incluido el día de la semana, y el comercio.
- Crear una nueva lista, asignándole, o no, un nombre para su identificación.
- Eliminar una lista guardada.
- Duplicar una lista ya guardada. La lista que se crea nueva es idéntica a la original, pero con el identificador creado a partir de la fecha y hora actual. Este identificador puede ser modificado para ajustar la fecha y la hora de la compra.

2.1.3 Almacén de productos

La tercera de las actividades principales es el almacén de productos, cuyo cometido es la gestión de los productos que se guardan en la aplicación.

Al igual que la actividad de la lista de la compra, tiene una zona de lista y otra de detalle.

Funcionalidades:

- Muestra el listado de los productos almacenados junto con algún detalle (precio de la última compra, precio por unidad de medida, último comercio en el que se compró, fecha, etc...) y una imagen representativa del producto, si está disponible.
- Buscar un producto. La búsqueda puede realizarse por su identificador (código de barras), la denominación o las etiquetas que se hayan asociado al producto.
- Si existe una lista abierta, no se muestran los productos de marca blanca que no pertenezcan al comercio de la lista. Para mostrar todos los productos, sin importar el comercio, ha que dejar el comercio de la lista abierta en blanco.

- Captura del código de barras (para la búsqueda) a través de la cámara del dispositivo. Si se captura un código de barras que no está registrado en la base de datos, se abre la pantalla para poder guardarlo.
- Se puede mostrar un listado completo de los productos dejando la casilla del texto a buscar en blanco.
- Se añade a la lista abierta con un click.
- Al añadirse a la lista abierta, se puede seleccionar el último precio conocido, el último precio conocido para el comercio de la lista, o no poner un precio.

Funcionalidades del detalle (click largo sobre el producto):

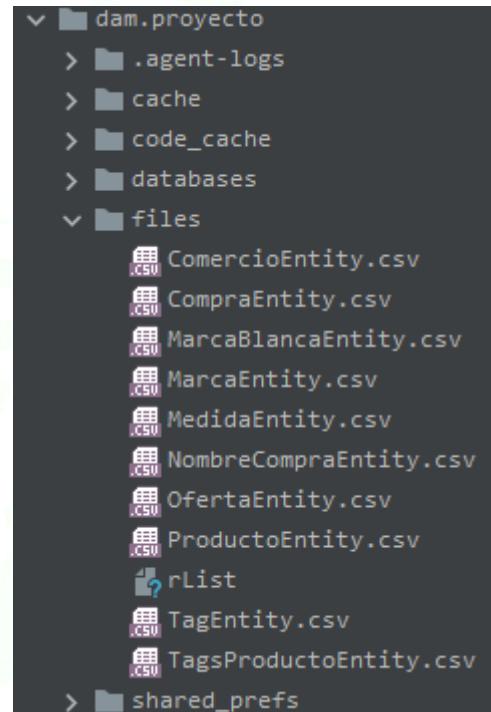
- Se muestra la información básica del producto: identificador, denominación, marca, unidades que componen el producto, cantidad, medida y etiquetas.
- Se puede cambiar la marca del producto.
- Se pueden añadir etiquetas.
- Se puede cambiar la información del producto.
- Se puede eliminar el producto.
- Se muestra un listado con las últimas compras del producto.
- Se muestra, si se ha guardado, una imagen del producto.
- Se puede modificar el código de barras (identificador del producto) capturándolo con la cámara del dispositivo.

2.1.4 Otras funcionalidades

RL

Funcionalidades accesibles desde el menú del *ToolBar*, en la pantalla de inicio.

- Insertar comercios (nombres de entre 3 y 16 caracteres).
- Asociar comercios y sus marcas blancas.
Se muestra un listado de las marcas asociadas al comercio.
- Exportar la base de datos.
Al exportar la base de datos, se generan tantos archivos csv como tablas hay.
El almacenamiento realiza en la carpeta de la propia aplicación *files*. Esta carpeta es creada exportar la base de datos y se elimina al desinstalar la aplicación.
- Importar la base de datos.
Con un click se puede recuperar la base de datos previamente exportada. Existe otra opción, siempre que se pueda acceder a los directorios del sistema, que consiste en copiar los archivos, que hayan sido guardados, al directorio *files* de la aplicación y, posteriormente, importar la base de datos. Esto sirve para insertar datos en una instalación nueva.
La importación de la base de datos supone el borrado de la anterior.



Archivos de respaldo de la base de datos.

2.2 Límites de la aplicación



La aplicación tiene algunos límites, bien porque no estén previstos, bien por no ser viables técnicamente:

- La aplicación no captura las imágenes de los productos directamente. La aplicación trata de ser una herramienta ágil, de uso sencillo. No es imprescindible la imagen del producto, aunque ayude a identificarlo. Hacer uso de la cámara para capturar una imagen, darle nombre, procesarla para darle el tamaño correcto (80x80px) y subirla al servidor, es un trabajo extra excesivo.
- La aplicación no comparte la base de datos por wi-fi, bluetooth o en la red. Es una aplicación personal, que maneja eficazmente los productos usados con frecuencia. Se nombran y etiquetan de una forma personal. El almacenamiento de la información es interno para una mayor agilidad a la hora de procesar las listas.
Si se comparten las listas, se deberían ajustar los identificadores de los productos, comercio, etiquetas, etc. Algunos productos pueden tener códigos de barras diferentes, por lo que es decisión del usuario igualarlos o no.
- La aplicación no actualiza los precios de forma automática, ni los obtiene de terceros.
Es el usuario quién debe actualizar los precios de forma personal. Aunque limita la capacidad de la información, asegura la integridad y veracidad de los datos.

2.3 Estructura de la aplicación



2.3.1 Pantallas principales

La aplicación se compone de tres actividades independientes, aunque necesarias para su correcto funcionamiento.

La actividad *Lista* es la encargada de la gestión de una lista de la compra. Muestra los producto y los detalles. Puede ser compartida.

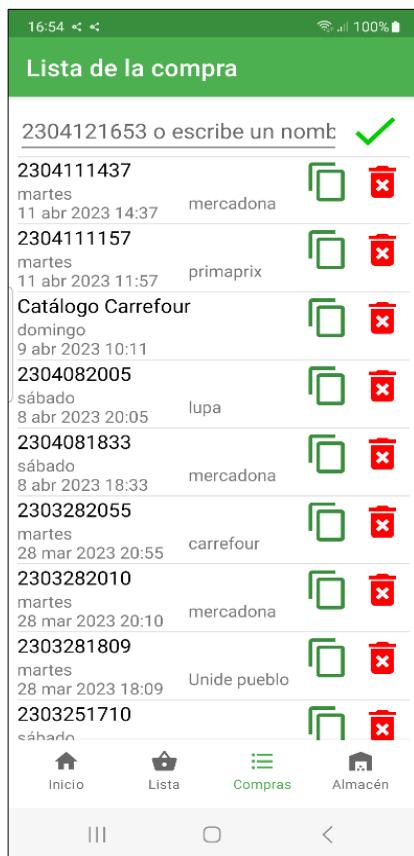
Su principal cometido es poder comparar la lista en un momento y/o comercio dado, con la lista en otro momento u otros comercios.



Lista abierta			
2304111437	mercadona		
Total:			27,89
PRODUCTO	UD.	PRECIO	PAGADO
Naranja	5,00	0,99	4,95
Tomate pera	2,30	2,29	5,27
Ensalada 4 estaciones	3,00	0,72	2,16
Cebolla	1,00	2,48	2,48
Zanahoria	1,00	1,15	1,15

Below the table are navigation icons for Inicio, Lista, Compras, and Almacén, along with standard Android navigation buttons.

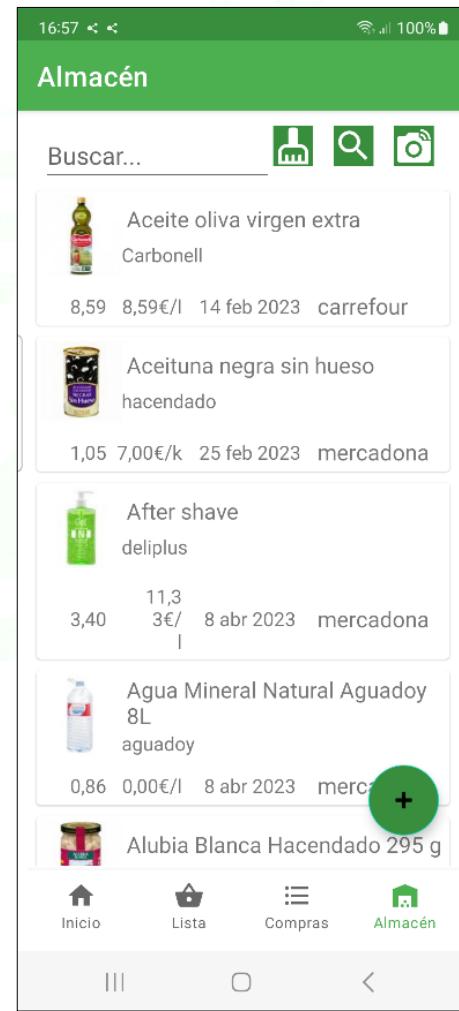
Actividad *Lista*



Actividad Compras

Compras es la actividad dedicada a la creación, modificación, duplicado o borrado de la colección de las listas de la compra guardadas.

Se limita a sus funciones. No conoce en absoluto nada acerca de los productos o los precios.



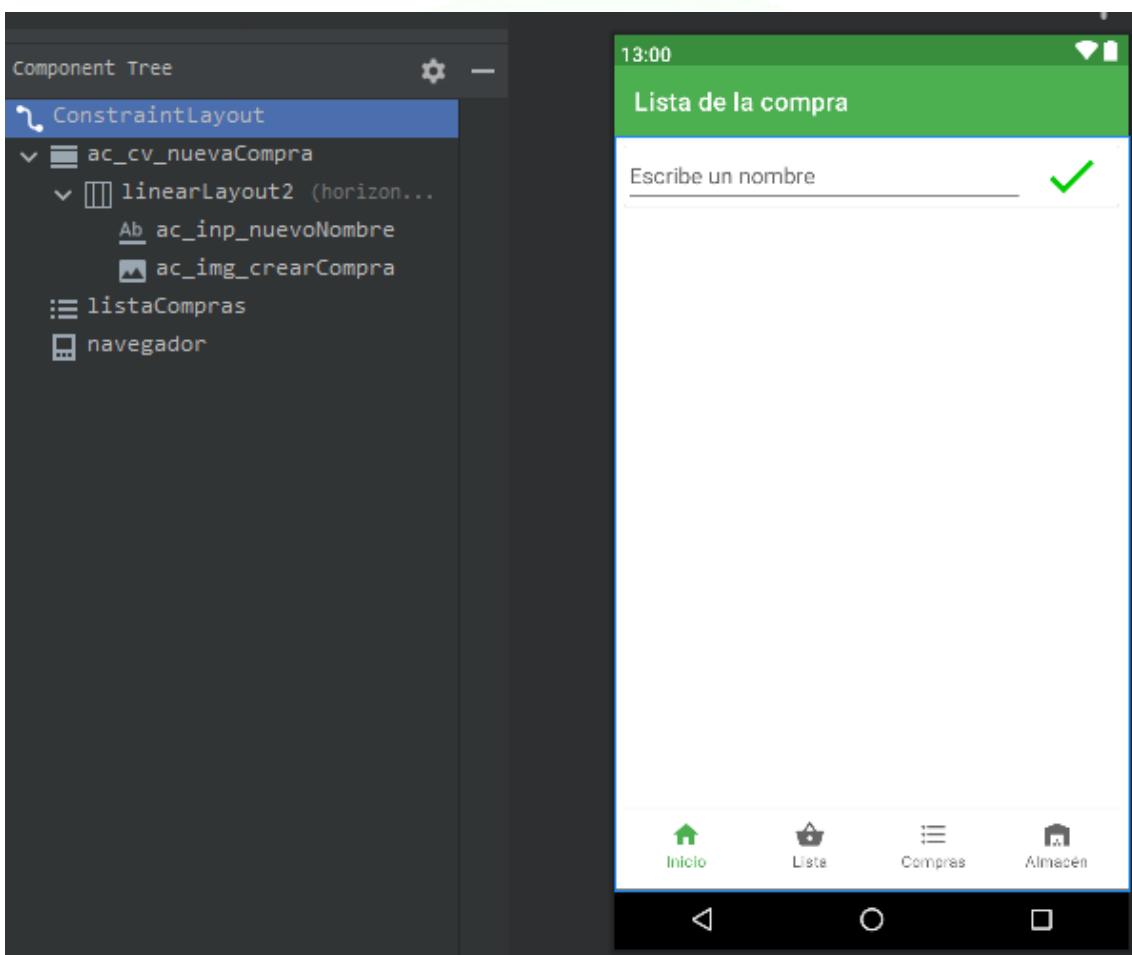
Actividad Almacén

2.3.2 Componentes principales de la interfaz de usuario



Cada una de las cuatro actividades iniciales (las tres principales más la de inicio) están compuestas por un contenedor para mostrar la información de cada una de ellas y por un *BottomNavigationView* que muestra los botones de navegación entre las actividades.

Cada una de las actividades tiene su propio navegador, no es único para todas ellas.



Árbol de componentes de la actividad Compras.

Las actividades *Lista* y *Compra* cargan fragments de lista y de detalle.

La vista del menú está definida en el archivo menu/bottom_nav_menu.xml y este archivo sí es aprovechado por cada actividad que lo usa.

Lista de la compra	
	Inicio
	Lista
	Compras
	Almacén

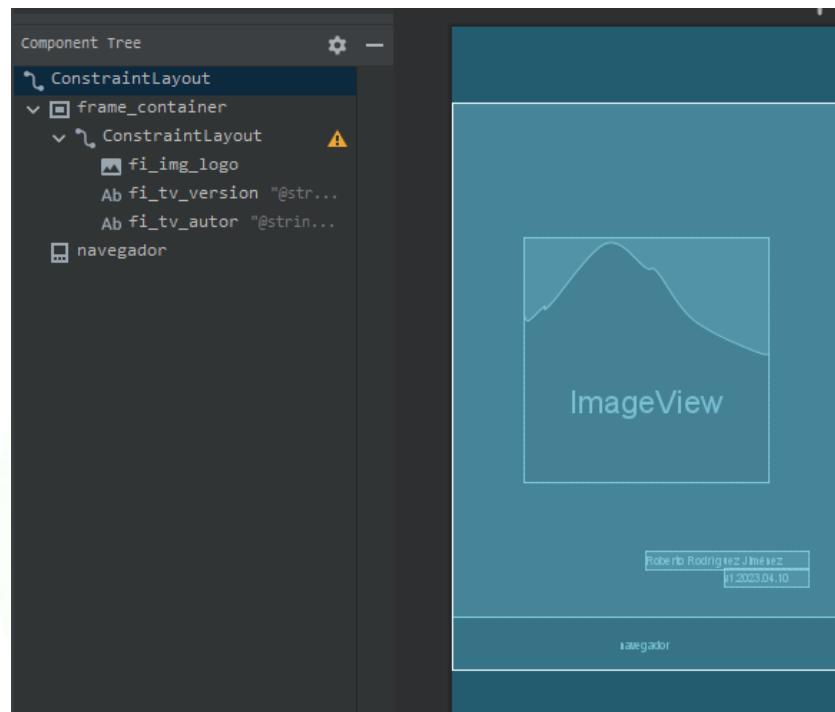
Menú usado en el BottomNavigationView

2.3.2.1 Inicio

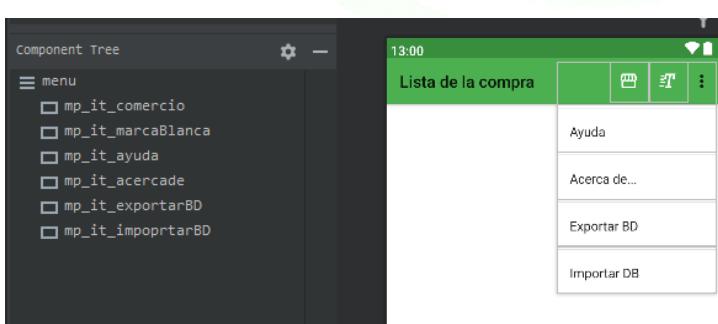
~L~

La estructura de la pantalla de inicio se diferencia del resto de pantallas principales en que incorpora un menú en el *ToolBar*.

El layout que carga la vista es `activity_main.xml`.



Vista del diseño de `activity_main.xml`



Diseño del menú principal.

El menú del *ToolBar* se declara en `MainActivity.class` con la función heredada `onCreateOptionsMenu(Menu menu)`. El archivo con las opciones del menú se ha declarado en `res/menu/menu_principal.xml`.

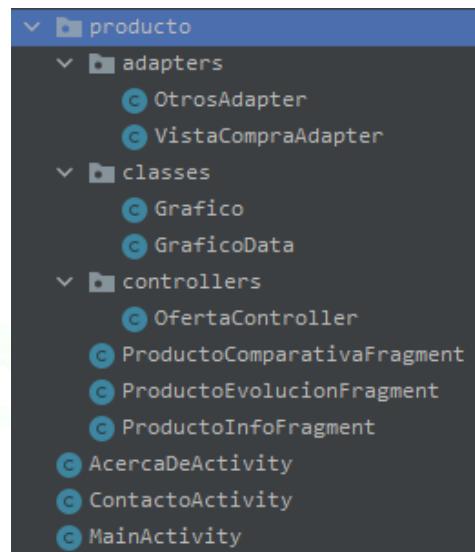
2.3.2.2 Lista

~

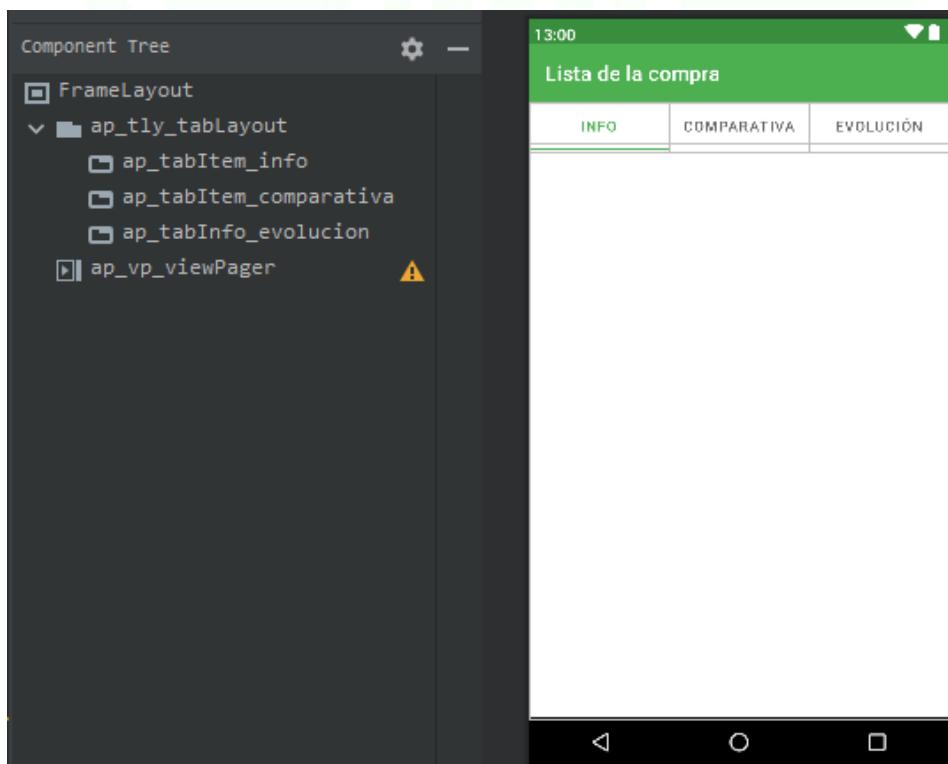
La lista de la compra tiene un *FragmentContainerView* que carga la lista abierta (*fragment_lista_list.xml*) o el detalle de los productos (*fragment_detalle_list.xml*).

La vista del detalle de los productos contiene, a su vez, un contenedor *ViewPager2* que permite visualizar varias actividades a través de pestaña.

Las actividades mostradas se obtienen del paquete `dam/proyecto/activities/producto`. Cada una de las clases son fragmentos que se cargan en el *ViewPager2*.



Archivos de "Producto" que se muestra en el detalle de la lista de la compra



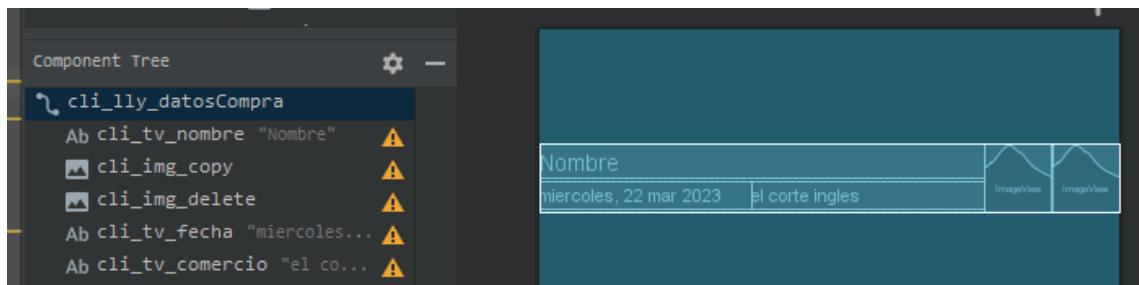
Fragment del detalle de la actividad Lista

2.3.2.3 Compras



La actividad *Compras* tiene la estructura más sencilla de todas (exceptuando la pantalla de inicio).

El componente principal, el que muestra el listado, es un *ListView* que recibirá los registros desde el adaptador `compras.adapters.AdaptadorCompras`. El adaptador hace uso de la vista `item_compra.xml` para presentar los datos de cada uno de los registros.



Vista de la plantilla usada para cargar la información de cada listado

2.3.2.4 Almacén

El almacén usa dos fragmentos para mostrar la información: uno para la vista y otro para el detalle.

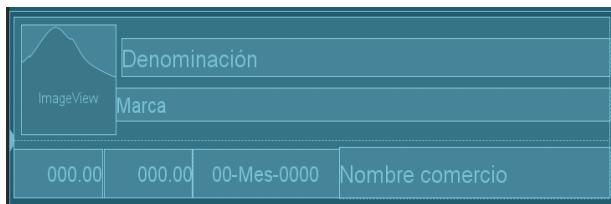
La vista de lista hace uso de un *RecyclerView* para mostrar los productos. El *RecyclerView* carga tan solo los registros que van a mostrarse en la pantalla, de esta forma, cuando uno de los registro sale de la pantalla, su vista es reciclada para el siguiente elemento. Este componente es útil cuando debe mostrarse un alto número de registro o estos son muy pesados. En el caso del almacén, parece adecuado para cargar las imágenes tan solo cuando van a ser mostradas, pues el listado de productos puede llegar a ser muy largo.

El layout que carga la lista es `fragment_lista_productos.xml`.



Diseño del fragment de la lista

La vista de cada producto, y que es pasada al adaptador para cargarla en el RecyclerView, es cardview_producto.xml.



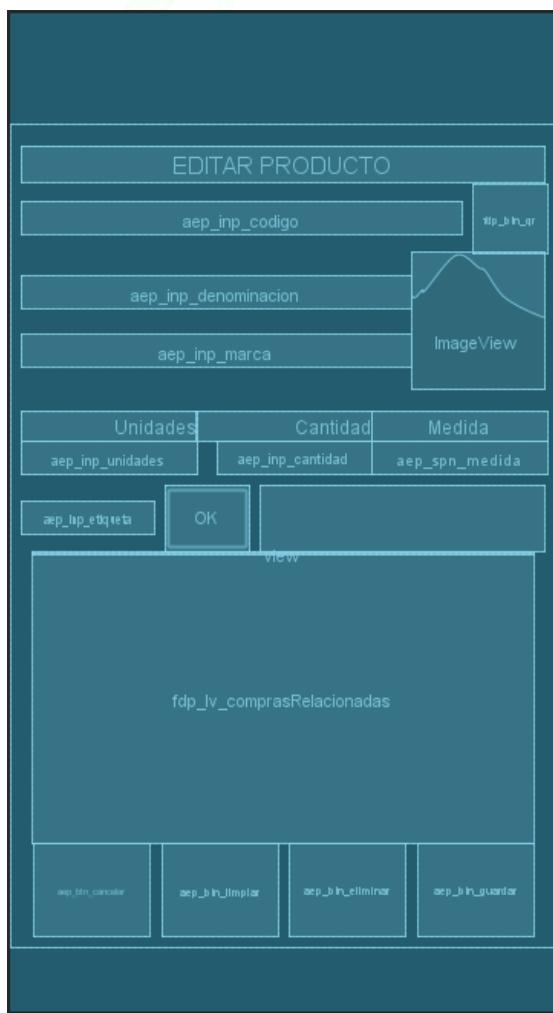
CardView para cada registro

Del detalle del producto se encarga fragment_detalle_productos.xml
Esta es una vista sencilla, pero que contiene un *ListView* para poder mostrar las compras del producto que se está mostrando. La vista que utiliza el

ListView es item_vista_compra.xml.



Vista usada en ListView para mostrar las compras.



Layout para mostrar el detalle del producto.

3 Documentación técnica

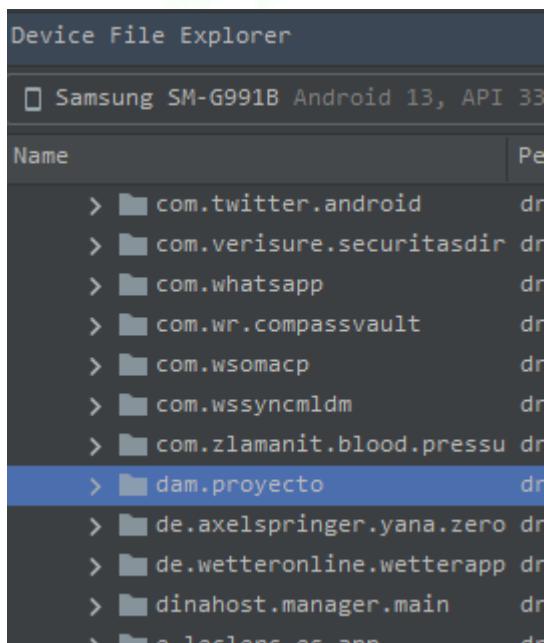


3.1 Creación del proyecto



Para la creación del proyecto se crea un nuevo proyecto vacío (*Empty Activity*) llamado *Proyecto*. La API inicial era la 21 (Android 5.0), pero los recursos que se han ido incluyendo durante el desarrollo han obligado a subir la API hasta la 27 (Android 7.0 Nougat).

Como nombre del paquete se le da *dam.proyecto*.



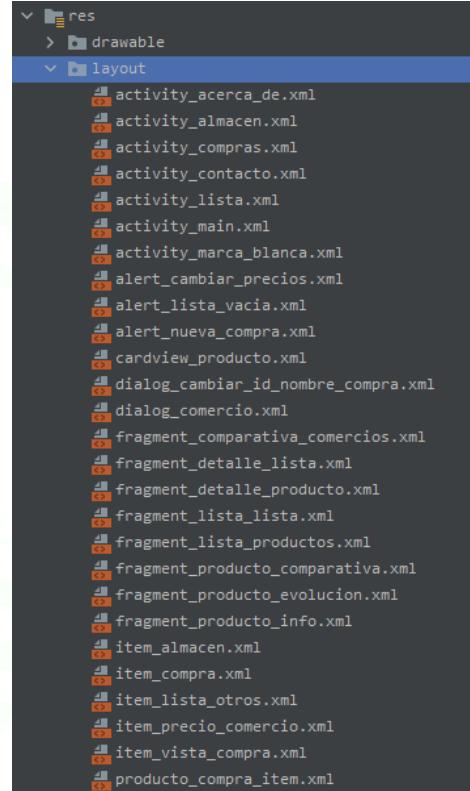
Directorio de la aplicación en el dispositivo

3.2 Estructura



Se ha intentado separar utilizar un patrón MVC, de forma que la aplicación sea escalable y no depender del origen de los datos.

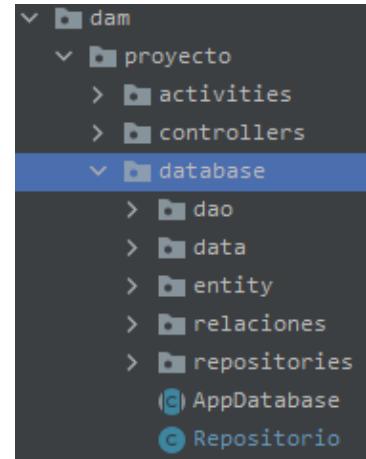
Las vista se especifican en res/layout. Cada *Activity*, *Fragment* o vista de los ítems de los *ListView* tiene su propia vista. La excepción es la vista del fragment *producto.ProductoEvolucionFragment.class* que, aunque tiene su propio layout, el componente *Grafico* está diseñado en tiempo de ejecución.



Directorio de vistas

Los datos se obtienen de la base de datos instalada en propio dispositivo. El SGBD es SQLite y está manejada por la librería Room database. Todos los archivos relacionados con los datos se agrupan en el paquete database.

- database: contiene todos los archivos necesarios la gestión de los datos.
- dao: paquete con los archivos DAO.
- data: diferentes archivos para cargar datos de ejemplo.
- entity: paquete en el que se agrupan los archivos la definición de las entidades (POJO's).
- relaciones: los archivos con relaciones son entidades, con su propio DAO, pero que no suponen una tabla, sino una relación. Por lo tanto, no suponen un archivo Entity.
- repositories: paquete para los repositorios.
- AppDatabase: archivo de configuración de la base de datos.
- Repositorio: superclase de los repositorios.

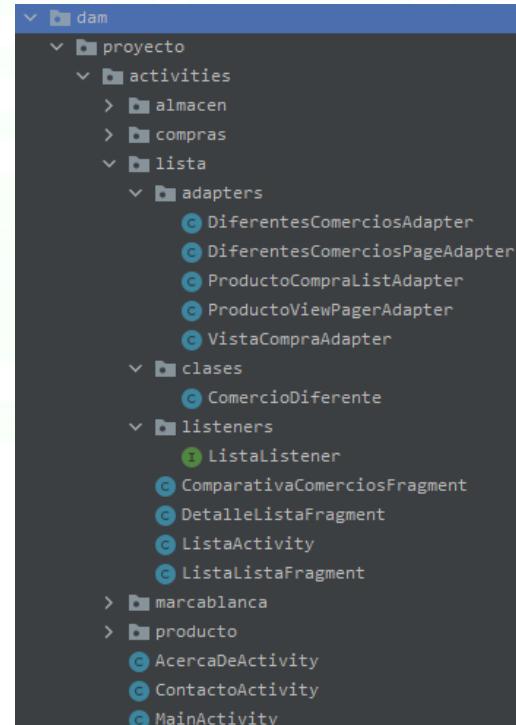


para
con
una

Lógica de la aplicación.

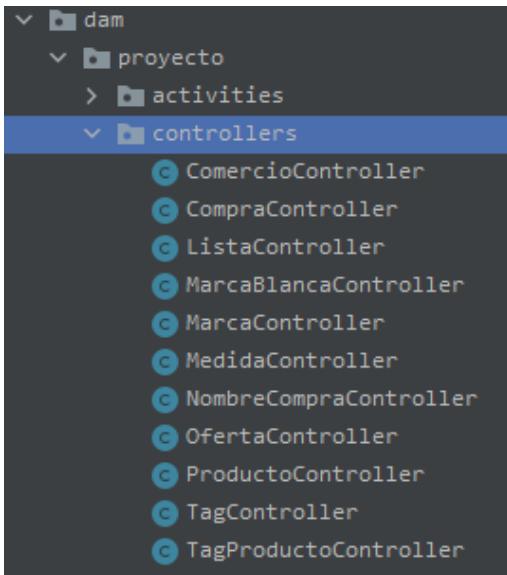
Para la lógica de la aplicación hay dos paquetes: *activities* y *controllers*.

Activities contiene las clases creadas cada una de las actividades de la aplicación. Si la actividad contiene fragments, éstos se encuentran dentro del paquete de la actividad. las clases que son requeridas por la actividad, y no por otras, se empaquetan de la actividad, como pueden ser los adaptadores, los listeners u otras clases usadas específicamente por la actividad.



para
Todas
dentro

Vista de las actividades y sus paquetes



Directorio de controladores

Los *controladores* son parte de la lógica común. Son usado por cualquier actividad. Su uso, principalmente, se centra en la petición de datos y el procesamiento éstos antes de entregarlos a la actividad que los ha solicitado, que es la clase que finalmente los muestra.

Cada repositorio cuenta con un controlador, de esta forma evitamos acceder a la base de datos cada vez que queramos modificar los datos, ya que se ha preferido realizar los cálculos por código en lugar de pedirlos al SGBD.

Algunas funciones de los controladores simplemente replican funciones de los repositorios, pero permiten poder modificar las respuestas en cualquier momento.

Un ejemplo: mostrar la última compra de un producto

El adaptador carga los datos en la vista

```
ultimaFecha.setText((mapa == null) ? "" : mapa.get("fecha"));
ultimoComercio.setText((mapa == null) ? "" : mapa.get("comercio"));
ultimoPrecio.setText((mapa == null) ? "" : mapa.get("precio"));
ultimoPrecioM.setText((mapa == null) ? "" : mapa.get("precioM"));
```

Desde Almacén se listan los productos y se muestra la última compra. Se la pedimos al controlador de compras.

```
CompraController controller = new CompraController(CONTEXT);
HashMap<String, String> mapa =
    (HashMap<String, String>) controller.getUltimaCompraDe(producto.getId());
```



Para mostrar el producto se usa el adaptador AdaptadorProductos

getUltimaCompraDe(producto) necesita el nombre del comercio en el que se realizó la compra

```
public Map<String, String> getUltimaCompraDe(String id) {
    try {
        // Obtener el producto para poder conocer las unidades de medida
        ProductoEntity producto = ProductoController.getById(id, CONTEXT);

        // Obtener la última compra del producto
        CompraEntity compra = REPOSITORY.getUltimaCompraByProducto(id);

        // Obtener el comercio en el que se realizó la compra
        String comercio = getNombreComercioByCompra(compra.getFecha());

        // Ya tenemos todos los datos
    } catch (Exception e) {
        producto.setId();
        compra.setFecha();
    }
}

public String getNombreComercioByCompra(String idCompra) {
    NombreCompraController nombreCompraController =
        new NombreCompraController(CONTEXT);
    return nombreCompraController.getNombreComercioByCompra(idCompra);
}

Map<String, String> obtenerMapa(ProductoEntity producto, CompraEntity compra) {
    Map<String, String> mapa = new HashMap<String, String>();
    mapa.put("producto", String.format("%.2f", producto.getPrecioMedido()) +
        "€/" + producto.getMedida());
    mapa.put("fecha", Fecha.getFechaFormatoada(compra.getFecha()));
    mapa.put("comercio", comercio);

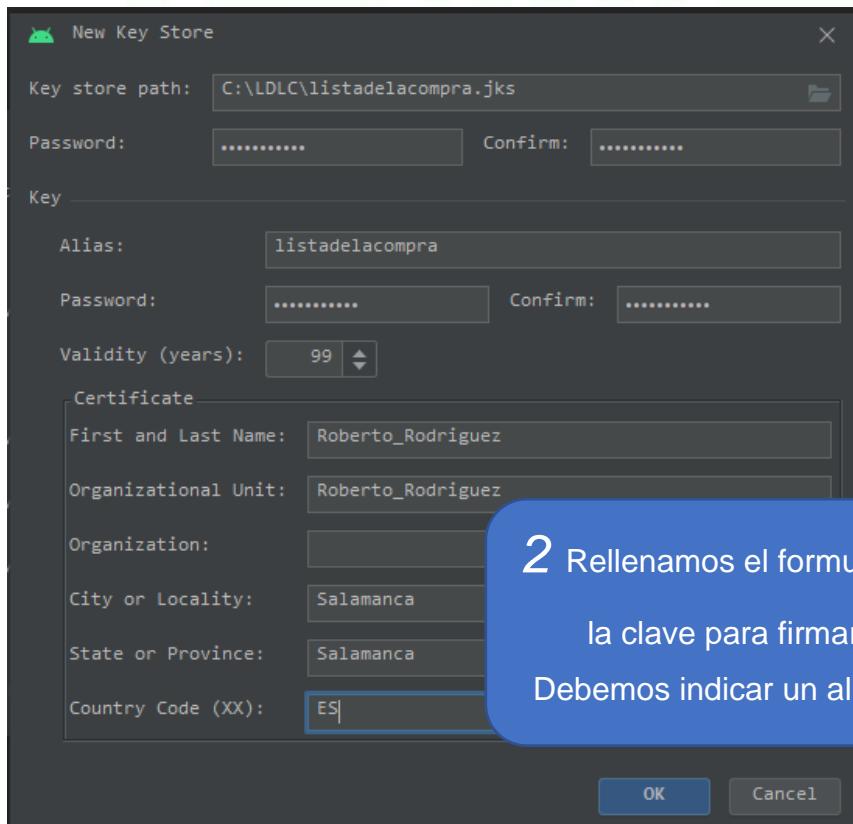
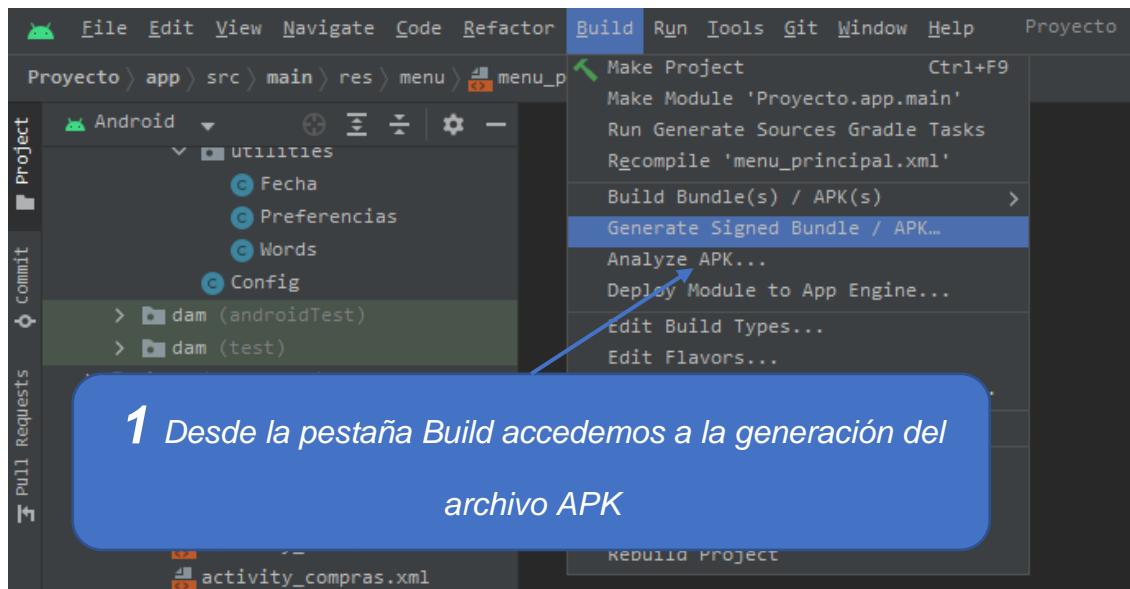
    return mapa;
}
```

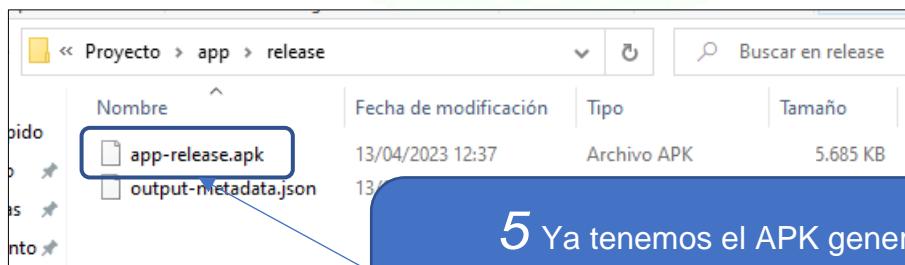
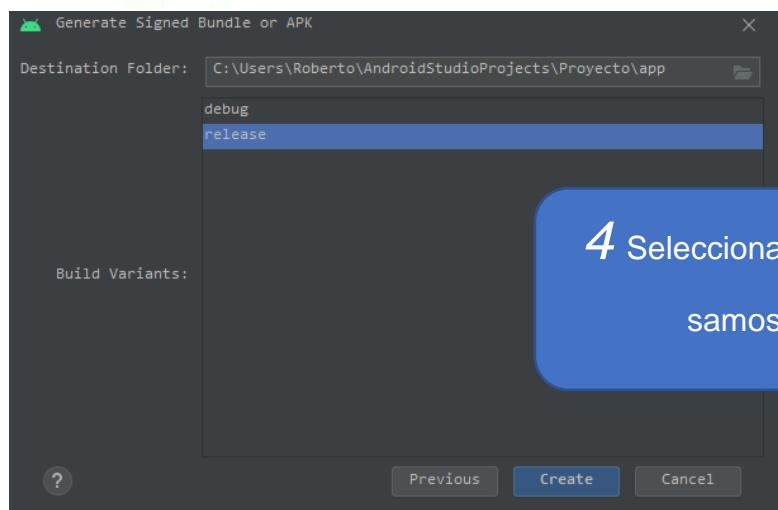
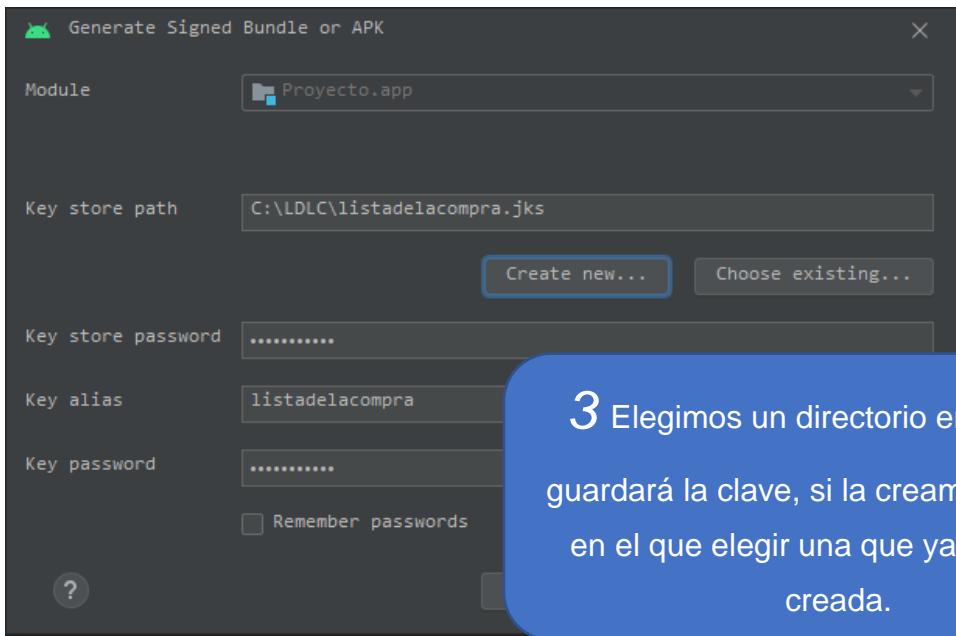
A su vez, el controlador de la compra tiene que llamar al controlador de NombreCompra que termina consultando al repositorio

3.3 Despliegue



3.3.1 Generar APK

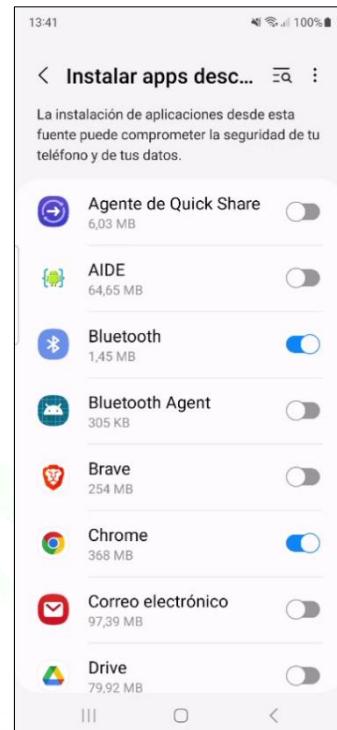




3.3.2 Instalar el APK

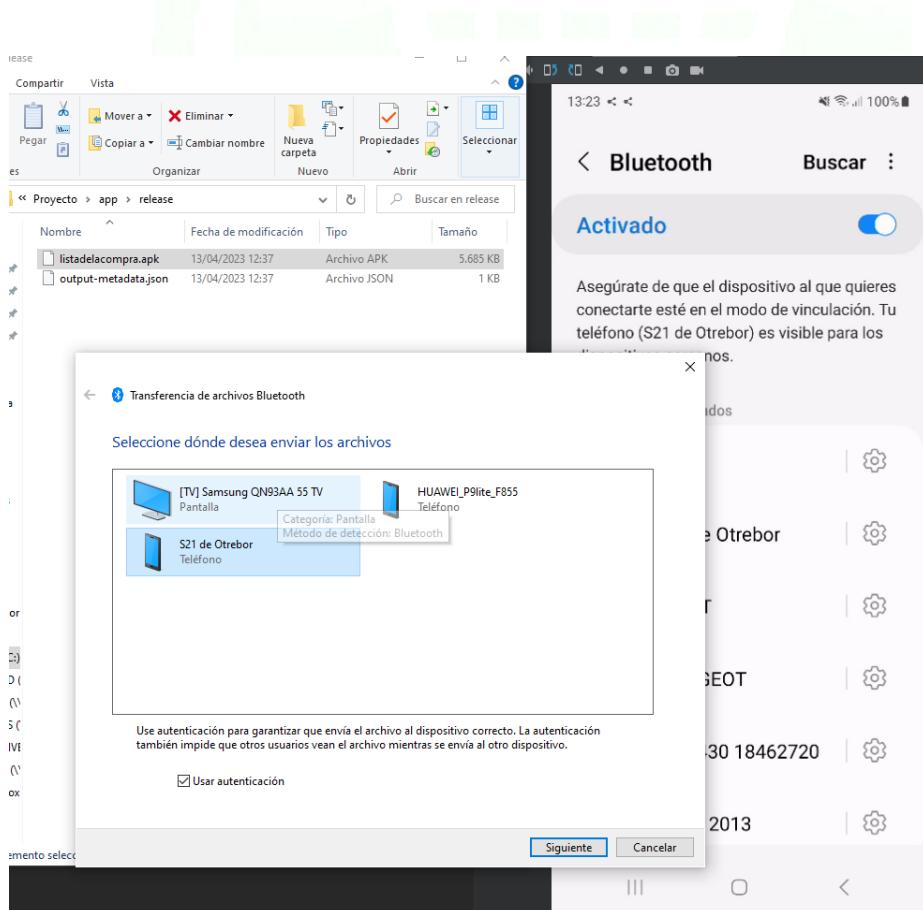


Se va a instalar la aplicación a través de bluetooth. Para que nos lo permita, debemos habilitar el permiso en *Instalar apps desconocidas* → *Bluetooth*. Al finalizar, volvemos a deshabilitar esta opción.

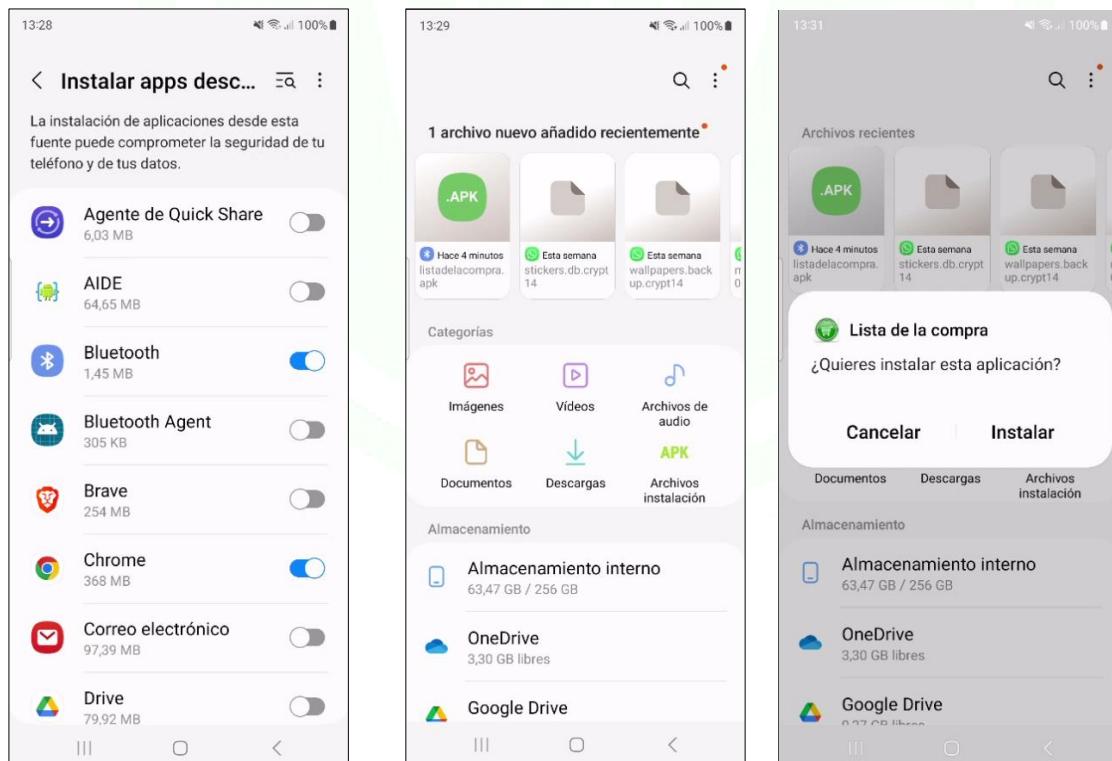
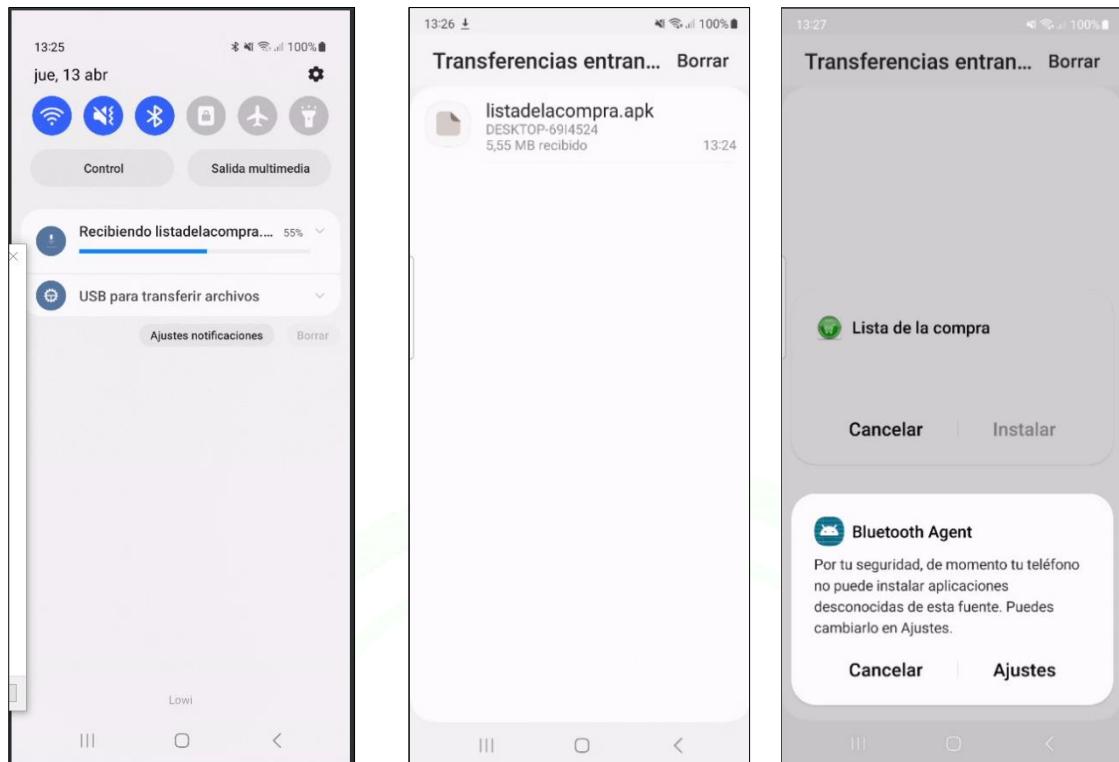


Habilitar la instalación desde Bluetooth

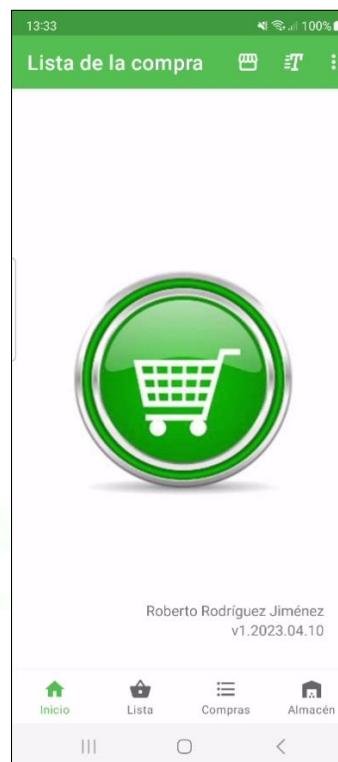
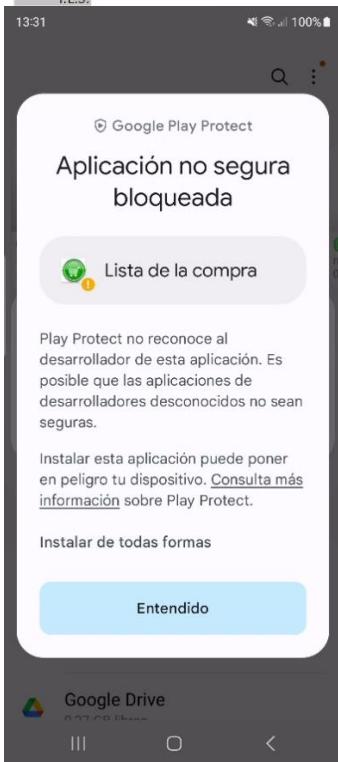
Se ha renombrado el archivo apk como *listadelacompra.apk* para que sea más fácilmente reconocible. Ahora, simplemente lo transferimos al dispositivo.



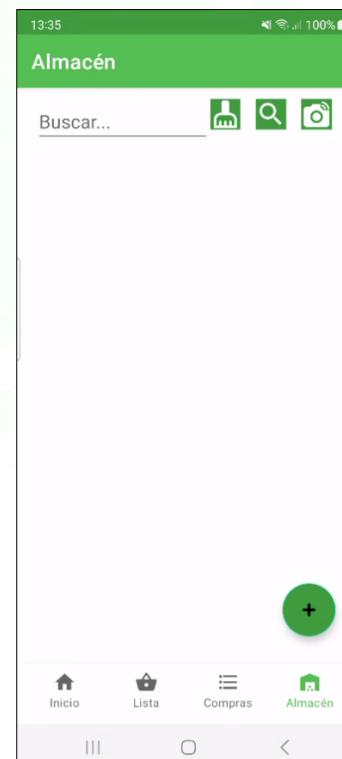
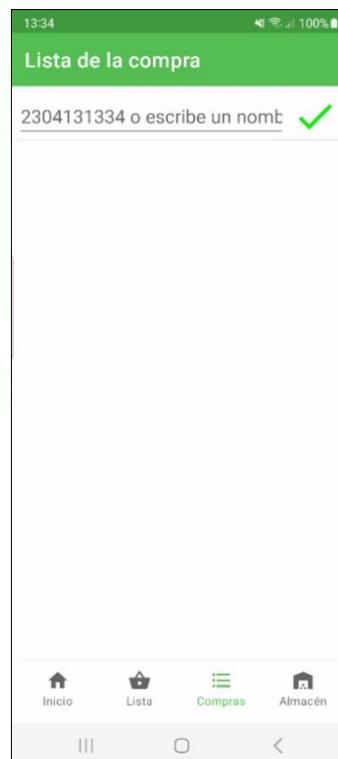
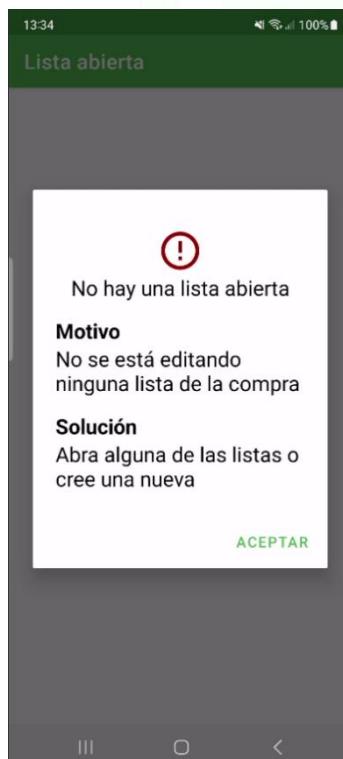
Transferencia del apk mediante Bluetooth



Secuencia de pasos para la transferencia del apk



Instalación de la aplicación en el dispositivo



La instalación se realiza en limpio, sin datos

3.4 Origen de los datos



3.4.1 Elección del SGBD



El almacenamiento de la información se hace de forma local al imperar la agilidad de la aplicación a la hora de obtener los datos y mostrarlos. Esta necesidad obliga a que dichos datos se almacenen en el mismo dispositivo para no depender de la conexión a internet. Tan solo las imágenes, debido al tamaño que ocupan en el disco, estarán almacenadas en un servidor remoto.

En lo que respecta al tipo de bases de datos, ya que se basa en la relación entre los productos, los comercios, las fechas de compras, etc, resulta obvio que ha de ser una base de datos relacional.

Para gestionar la base de datos he decidido usar la librería **room database** en lugar de realizar las consultas sobre **SQLLite** directamente. El motivo principal es que proporciona mecanismos para gestionar fácilmente el trabajo directo con objetos. Otra ventaja que aporta es la gestión de las llamadas a las conexiones de la base de datos, quitando al desarrollador la labor de crear y destruir las conexiones.

Room es una librería creada en Google para Android.

Conceptos:

- *Entity* es la clase que describe una tabla de la base de datos (POJO).
- *SQLLite database*: almacenamiento interno del dispositivo.
- *DAO* (Data Access Object) Objeto de Acceso a Datos. Asigna las consultas a las funciones.
- *Repository* es utilizado para proporcionar múltiples fuentes de datos.

- *ViewModel* actúa como un centro de comunicación entre el repositorio y la UI, sin preocuparse por la fuente de datos.

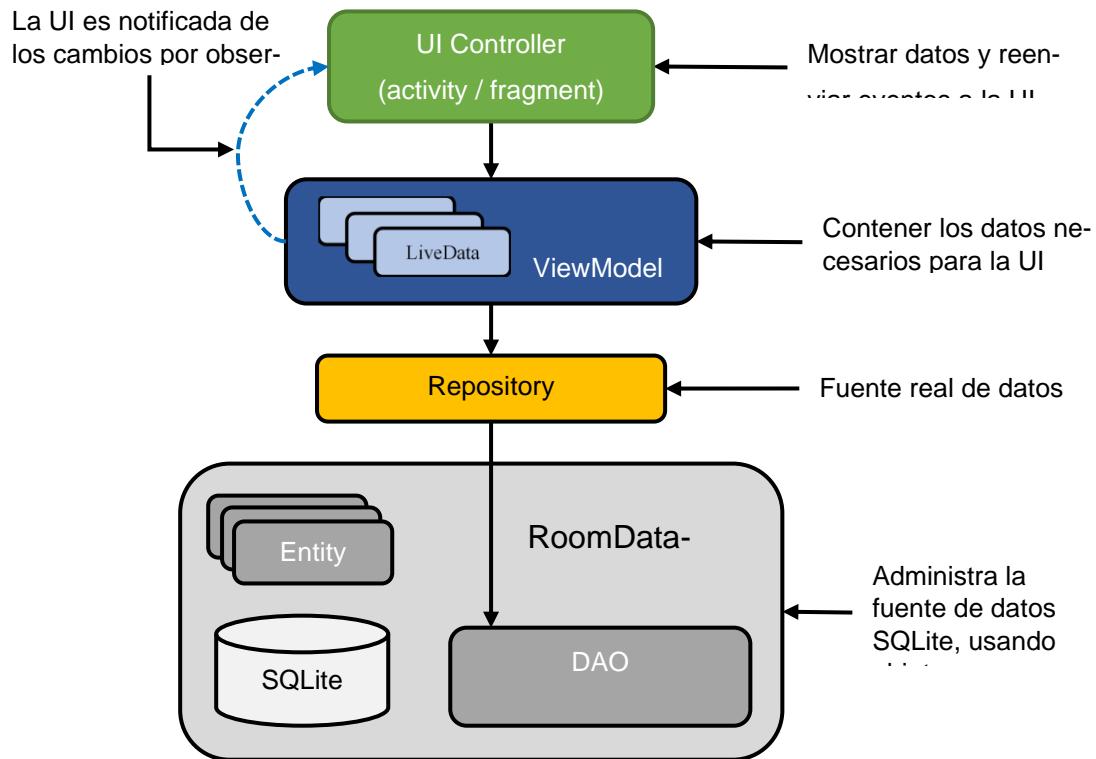


Diagrama básico de la arquitectura Room Database

3.4.2 Datos del producto



Partimos de los datos de un producto cualquiera.

Para cada producto debemos guardar:

- El *código de barras*.
- Un *denominador* para el producto.
- La *marca*.
- Cuántas *unidades* componen el producto.
- La *cantidad* de producto.
- En qué *magnitud* se mide la cantidad.
- Algunas *etiquetas*.



Producto compuesto por 4 unidades(cuatro yogures) y 0,48k de cantidad (4x120g).



Producto compuesto por 1 unidad (media docena) y cantidad 6u

Las medidas para las cantidades son:

- Kilo (k)
- Litro (l)
- Unidad (u)
- Metro (m)
- Granel (gr)

La denominación del producto genera etiquetas: cada una de las palabras que la componen genera una etiqueta.

3.4.3 Datos de la compra

Los datos de la compra se obtienen a partir de la compra de cada producto.

Los datos que guardamos para cada producto comprado son:

- Para la compra:
 - La fecha y hora (dd-mm- hh:mm) de la compra
 - Comercio

- Para el producto
 - El código de barras
 - Precio
 - Unidades compradas

Descripción	P. Unit	Imp.(€)
1 LECHE DESN P6		4,62
1 BIF 0% NAT		1,10
1 P.PAVO REDUC.SAL		3,15
1 LOMO ADOBADO		3,10
2 JAMON SERRANO 1/2 L.	3,54	7,08
2 QUESO HAVARTI LIGHT	2,65	5,30
1 SANDIA PARTIDA B/S		4,09
1 TORTILLAS MEXICANAS		1,30
2 ENS-TV ESTACIONES	0,69	1,38
1 MELON AMAR PARTIDO		1,92
1 ARÁNDANO VIOLETA		2,89
2 PICATAC		1,70
1 PAN M...		1,55

Un ticket común de la compra.

Tabla antes de normalizar

COMER-CIO	FECHA	Ud.	pre-cio	ID	DENOMINA-CIÓN	MARCA	tags	Ud	Cant.	M
Carrefour	2303301956	0.79	1.69	1000000000001	Plátano	Granel	Plátano, fruta	1	1	k
Carrefour	2303301956		3.99	8410500008871	Yogurt bífidus natural desnatado	Danone	Yogurt,bífidus,desnatado,natural	8	0.96	k
Carrefour	2303301956	1	2.25	8410128000004	Agua mineral	Bezoya	Agua, mineral	1	5	l
Carrefour	2303301956	1	2.15	8431876291674	Huevos	Carrefour	Huevos	1	6	u
Lupa	2303281345	1	2.85	8429687801212	Huevos	Lugareño	Huevos	1	12	u
Lupa	2303281345	1	2.19	8410128000004	Agua mineral	Bezoya	Agua, mineral	1	5	l
Lupa	2303281345	1.25	1.39	1000000000001	Plátano	Granel	Plátano, fruta	1	1	k
Merca-dona	2303271855	1	0.86	8437003018008	Aqua	Aguadoy	Aqua, mineral	1	8	l
Merca-dona	2303271855	1.05	1.99	1000000000001	Plátano	Granel	Plátano, fruta	1	1	k
Merca-dona	2303271855	1	4.25	8006540750889	Champú HS classic 360	H&S	champu	1	0.36	l
Alcampo	2303261234	0.85	1.89	1000000000001	Plátano	Granel	Plátano, fruta	1	1	k
Alcampo	2303261234	1	8.99	8006540746432	Champú HS classic 700	H&S	Champú,	1	0.7	l
Carrefour	2303221643	0.79	1.49	1000000000001	Plátano	Granel	Plátano, fruta	1	1	k

COMPRA

COMÚN

PRODUCTO

3.4.4 Normalización 3FN

→

id	marca
1	Granel
2	Danone
3	Bezoya
4	Carrefour
5	Lugareño
6	Aguadtoy
7	H&S

id	comercio
1	Carrefour
2	Lupa
3	Mercadona
4	Alcampo

id	tag
1	platano
2	fruta
3	yogurt
4	bifidus
5	natural
6	agua
7	mineral
8	huevos
9	champu

id	med.
gr	granel
k	kilo
l	litro
m	metro
u	unidad

Tabla de productos

ID	DENOMINACIÓN	MARCA	Ud	Cant.	M
10000000000001	Plátano	1	1	1	k
8410500008871	Yogurt bifidus natural desnatado	2	8	0.96	k
8410128000004	Agua mineral	3	1	5	l
8431876291674	Huevos	4	1	6	u
8429687801212	Huevos	5	1	12	u
8410128000004	Agua mineral	3	1	5	l
8437003018008	Agua mineral	6	1	8	l
8006540750889	Champú HS classic 360	7	1	0.36	l
8006540746432	Champú HS classic 700	7	1	0.7	l

Relación de compras

FECHA	COMERCIO	NOMBRE
2303301956	1	2303301956
2303281345	2	2303281345
2303271855	3	2303271855
2303261234	4	2303261234
2303221643	1	2303221643

Tabla de compras

COMPRA	ID	Ud.	precio
2303301956	10000000000001	0.79	1.69
2303301956	8410500008871	2	3.99
2303301956	8410128000004	1	2.25
2303301956	8431876291674	1	2.15
2303281345	8429687801212	1	2.85
2303281345	8410128000004	1	2.19
2303281345	10000000000001	1.25	1.39
2303271855	8437003018008	1	0.86
2303271855	10000000000001	1.05	1.99
2303271855	8006540750889	1	4.25
2303261234	10000000000001	0.85	1.89
2303261234	8006540746432	1	8.99
2303221643	10000000000001	0.79	1.49

Relación de tags

id	tag
10000000000001	1
10000000000001	2
8410500008871	3
8410500008871	4
8410500008871	5
8410128000004	6
8410128000004	7
8431876291674	8
8429687801212	8
8437003018008	6
8437003018008	7
8006540750889	9
8006540746432	9

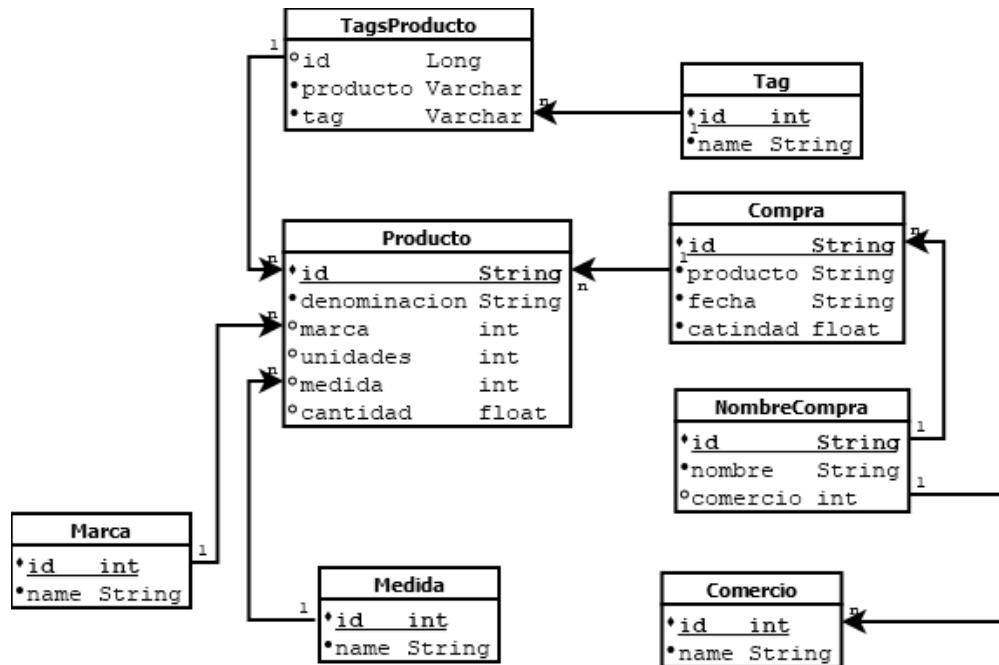


Diagrama de clases

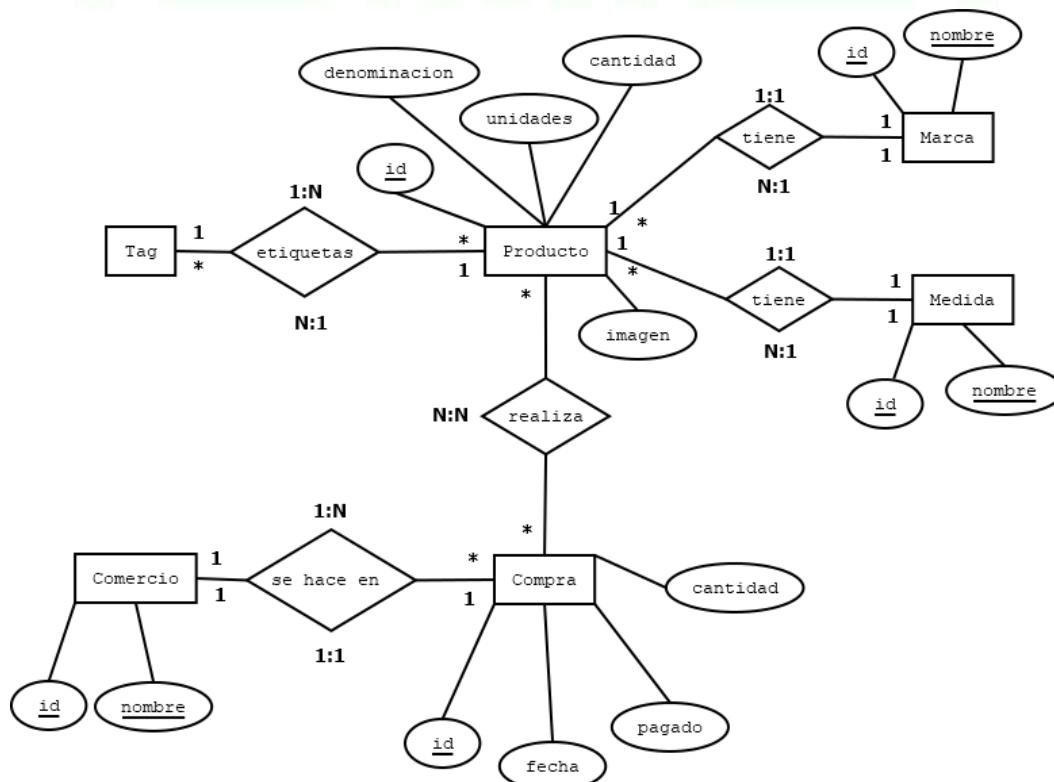


Diagrama Entidad-Relación

3.4.5 Otros datos



El desarrollo de la aplicación obliga a tener que restringir datos que en principio no se tenían en cuenta.

Un ejemplo son las marcas blancas de los comercios. A la hora de realizar la búsqueda de los productos, no debe mostrarse aquellas marcas que son propias de otros establecimientos.

Para dar solución a esta parte, creamos una tabla de relación que une las marcas blancas a los productos.

id	comercio
1	Carrefour
2	Lupa
3	Mercadona
4	Alcampo
5	Gadis

id	marca
1	Granel
2	Danone
3	Bezoya
4	Carrefour
5	Lugareño
6	Aguadoy
7	H&S
8	Hacendado
9	Alteza
10	Elige
11	Bosque Verde
12	Actuel
13	Qilive

marca	comercio
4	1
6	3
8	3
9	2
10	5
11	3
12	4
13	4

Otro dato que debemos añadir a la tabla de compra es la opción de que se haya hecho durante una oferta. Aunque el precio final marcado es el que se ha pagado (ejecutada la oferta), conviene recuperarla cuando la compra sea editada.

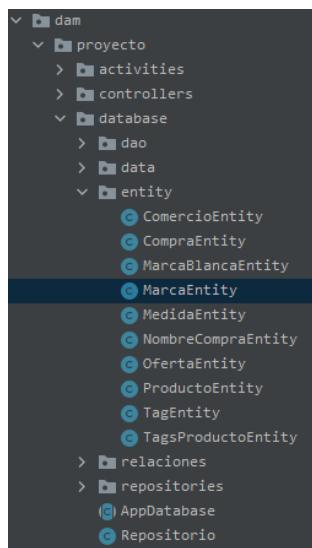
COMPRA	ID	Ud.	precio	oferta
2303301956	10000000000001	0.79	1.69	
2303301956	8410500008871	2	3.99	2-50
2303301956	841012800004	1	2.25	
2303301956	8431876291674	1	2.15	
2303281345	8429687801212	1	2.85	
2303281345	841012800004	1	2.19	
2303281345	10000000000001	1.25	1.39	
2303271855	8437003018008	1	0.86	
2303271855	10000000000001	1.05	1.99	
2303271855	8006540750889	1	4.25	
2303261234	10000000000001	0.85	1.89	
2303261234	8006540746432	1	8.99	
2303221643	10000000000001	0.79	1.49	

abrv texto
2x1 2x1
3x2 3x2
2-50 2ª 50%
2-70 2ª 70%

3.4.6 Room database



3.4.6.1 Entidades



Listado de *Entities* usadas.

Las *entities* se almacenan en `database/entity` con el nombre `NombreTablaEntity.java`. Para cada una de las tablas se crea una entidad cuyos campos serán las propiedades de la tabla.

La clase tiene un solo constructor.

Al declarar las propiedades de la clase (columnas de la tabla) se indica, mediante etiquetas, en nombre de la tabla y el resto de las propiedades.

La clase debe tener los getters y los setters de cada propiedad.

```
@Entity(tableName = "Marca")
public class MarcaEntity {

    @PrimaryKey(autoGenerate = true) private int id;
    @ColumnInfo(name = "name") private String name;

    public MarcaEntity( int id, String name ) {
        this.id = id;
        this.name = name;
    }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }
}
```

Entity para la tabla de marcas.

3.4.6.1.1 VistaCompra

RL

VistaCompra es una entidad de relación. Representa la vista de la compra de un producto en la que se recupera la denominación, el nombre del comercio, el precio, el precio medio, la medida y la fecha de la compra.

El DAO tiene dos operaciones: puede obtener las vistas por producto o por etiqueta.

```
@Dao
public interface VistaCompraDao {
    1 usage 1 implementation ▲ robertorodriguezNet
    @Query("select p.denominacion, m.name, c.precio, c.preciomedido, p.medida, c.fecha " +
        "from Compra as c, NombrecCompra as n, Comercio as m, Productos as p " +
        "where c.producto = :idProducto " +
        "and c.fecha = n.id " +
        "and m.id = n.comercio " +
        "and p.id = c.producto " +
        "order by c.fecha DESC")
    List<VistaCompra> getVistaCompraByProducto(String idProducto);

    1 usage 1 implementation ▲ robertorodriguezNet *
    @Query("select p.denominacion, m.name, c.precio, c.preciomedido, p.medida, c.fecha " +
        "from Compra as c, NombrecCompra as n, Comercio as m, Productos as p " +
        "where c.producto in (SELECT producto FROM TagsProducto WHERE tag = :idTag) " +
        "and c.fecha = n.id " +
        "and m.id = n.comercio " +
        "and p.id = c.producto " +
        "order by c.fecha DESC")
    List<VistaCompra> getVistaCompraByTag( int idTag );
}
```

DAO de VistaCompra

Los datos se recuperan tal y como están guardados en sus tablas, no hay campo calculados.

3.4.6.2 DAO

RL

Los DAO son los Data Access Objects (Objetos de Acceso a Datos). Son interfaces en las que se definan las operaciones que se harán contra la base de datos. Existen operaciones ya definidas y otras que pueden crearse mediante consultas personalizadas. Podemos pedir que se devuelvan datos individuales, en colecciones o ejecutar consultas relacionadas con tablas diferentes al propio DAO. Estos datos pueden ser datos primitivos u objetos.

```

@Dao
public interface MarcaDao {
    // CREATE -----
    @Insert
    void insert(MarcaEntity object);

    @Insert
    void insertAll(List<MarcaEntity> objects);

    // READ -----

    @Query("SELECT * FROM Marca WHERE id = :id")
    MarcaEntity findById(int id);
}

public interface VistaCompraDao {
    @Query("select p.denominacion, m.name, c.precio, c.preciomediado, p.medida, c.fecha " +
           "from Compra as c, Nombrecompra as n, Comercio as m, Productos as p " +
           "where c.producto = :idProducto " +
           "and c.fecha = n.id " +
           "and m.id = n.comercio " +
           "and p.id = c.producto " +
           "order by c.fecha DESC")
    List<VistaCompra> getVistaCompraByProducto(String idProducto);

    @Query("select p.denominacion, m.name, c.precio, c.preciomediado, p.medida, c.fecha " +
           "from Compra as c, Nombrecompra as n, Comercio as m, Productos as p " +
           "where c.producto in (SELECT producto FROM TagsProducto WHERE tag = :idTag) " +
           "and c.fecha = n.id " +
           "and m.id = n.comercio " +
           "and p.id = c.producto " +
           "order by c.fecha DESC")
    List<VistaCompra> getVistaCompraByTag( int idTag );
}

```

SEI DAO de VistaCompra devuelve objetos relacionales

```

    implementation & robertorodriguezNet
    try("SELECT * FROM Marca WHERE name = :name")
    MarcaEntity findByName(String name);

    implementation & robertorodriguezNet
    try( "SELECT MAX(id) FROM Marca")
    getMaxId();

    implementation & robertorodriguezNet
    try("SELECT * FROM Marca")
    <!MarcaEntity> getAll();

    implementation & robertorodriguezNet
    try("SELECT name FROM Marca ORDER BY name ASC")
    <String> getAllNames();

    UPDATE -----
    implementation & robertorodriguezNet
    date
    void update(MarcaEntity object);

    // DELETE -----
    implementation & robertorodriguezNet
    @Delete
    void delete(MarcaEntity object);

    implementation & robertorodriguezNet
    @Query("DELETE FROM marca")
    void clear();
}

```

DAO para la tabla "marcas"

↴

```
public class MarcaRepository extends Repository{

    9 usages
    private final MarcaDao DAO;

    3 usages  ↳ robertorodriguezNet
    public MarcaRepository(Context context) {
        super(context);
        DAO = db.marcaDao();
    }

    /** Borra los datos de la tabla */
    ↳ robertorodriguezNet
    public void clear() { DAO.clear(); }

    /** Devuelve el objeto a partir del nombre ...*/
    ↳ robertorodriguezNet
    public MarcaEntity findByName( String name ) { return DAO.findByName( name ); }

    /** Devuelve el objeto a partir del id ...*/
    ↳ robertorodriguezNet
    public MarcaEntity findById( int id ) { return DAO.findById( id ); }

    /** Inserta el objeto en la tabla ...*/
    ↳ robertorodriguezNet
    public void insert( MarcaEntity objeto ) { DAO.insert( objeto ); }
}
```

Muestra del repositorio de la tabla "Marca"

Los repositorios son las clases que se utilizan para acceder a las consultas definidas en los DAO's. De esta forma, podemos establecer diferentes orígenes de los datos al poder realizar las conexiones a diferentes DAO's.

3.4.6.3 Conexión con la base de datos

```
// Nombre de la base de datos
1 usage
private static final String DB_NAME = "listadelacompra.db";

1 usage  ↳ robertorodriguezNet
public static AppDatabase getInstance(Context context) {

    if (INSTANCE == null) {
        synchronized (AppDatabase.class) {
            if (INSTANCE == null) {
                INSTANCE = Room.databaseBuilder(context, AppDatabase.class, DB_NAME)
                    .allowMainThreadQueries()
                    .fallbackToDestructiveMigration()
                    .build();
            }
        }
    }
    return INSTANCE;
}
```

Método que devuelve un objeto con la conexión

```
1 usage 1 implementation  ↳ robertorodriguezNet
public abstract ComercioDao comercioDao();
1 usage 1 implementation  ↳ robertorodriguezNet
public abstract CompraDao compraDao();
1 usage 1 implementation  ↳ robertorodriguezNet
public abstract MarcaDao marcaDao();
```

Declaración de los DAO's

La conexión se declara en una clase abstracta `AppDatabase` que extiende de `RoomDatabase`. En esta clase hay que declarar las entidades y los DAO's.

```
@Database(
    entities = {
        ComercioEntity.class,
        CompraEntity.class,
        MarcaEntity.class,
        MedidaEntity.class,
        ProductoEntity.class,
        TagEntity.class,
        TagsProductoEntity.class,
        NombreCompraEntity.class,
        OfertaEntity.class,
        MarcaBlancaEntity.class
    },
    version = 1,
    exportSchema = false
)
```

Declaración de las entidades

El método estático `getIn-
stance()` de-
vuelve un ob-
jeto `AppData-
base` con la co-
nexión a la
base de datos.

3.5 Backup remoto

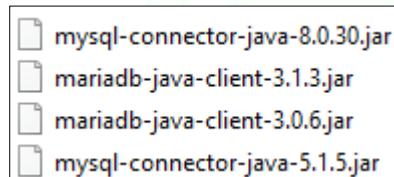


El almacenamiento de la información de forma remota es, de momento, inviable debido a la posible pérdida de información al manipular los datos. Implementar esta opción está presente, pero necesita replantear la aplicación.

Como alternativa temporal se ha optado por crear un repositorio de datos completo y real, con acceso restringido a los administradores. El base de datos remoto se encuentra en un servidor compartido de *Dinahosting*. El motor de la base de datos es MariaDB, a pesar de que el conector JDBC usado es MySQL, ya que los conectores de MariaDB provocaban errores de compatibilidad.

La implementación del Driver JDBC se ha hecho copiando el archivo *.jar* directamente al directorio *app/libs* del proyecto. Una vez copiado el archivo, en su menú contextual, se selecciona *Add As Library...* para añadir el *.jar* como librería. Automáticamente se agrega la línea de implementación en el *build.gradle*.

Driver JDBC-MYSQL: 5.1.5



3.5.1 Exportar datos

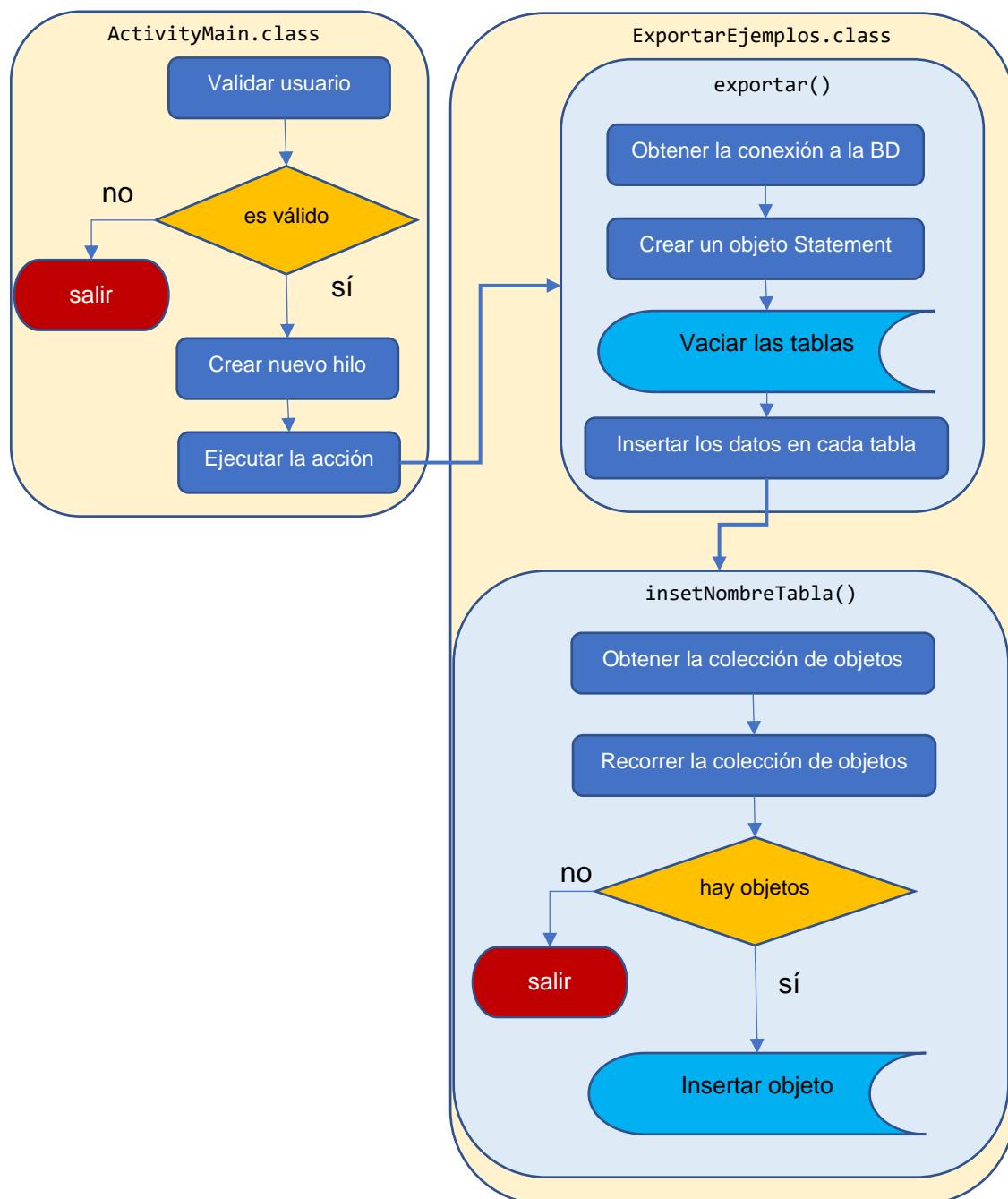
dam/proyecto/database/data/ExportarEjemplos.java

La llamada a la clase encargada de exportar los datos a la base de datos remota se realiza desde *ActivityMain.HiloExportarEjemplos.run()*. *HiloExportarEjemplos* es una clase interna que extiende *Thread* para poder ejecutar la consulta en un hilo diferente del principal, ya que la operación puede durar el tiempo suficiente para que Android lance un mensaje diciendo que la aplicación no responde.

Desde `run()` se llama al método estático `ExportarEjemplo.exportar()`, que se encarga de realizar toda la operación para cargar los datos en la base de datos remota.

Después de realizar la conexión a la base de datos, ejecuta una instrucción para vaciar las tablas y, a continuación, para cada una de ellas, ejecuta un método `insertNombreTabla()` para insertar cada uno de los registros de cada tabla.

El método `insertNombreTabla()` pide al controlador de la tabla toda la colección de registros, la recorrer y, para cada registro, ejecuta una instrucción `INSERT`.



3.5.2 Importar datos

dam/proyecto/database/data/ExportarEjemplos.java

El proceso se realiza en un hilo secundario creado en `ActivityMain.HiloImportarEjemplos()`.

Importar datos presenta dos opciones:

- Importar solo los productos.
- Importar todos los datos.

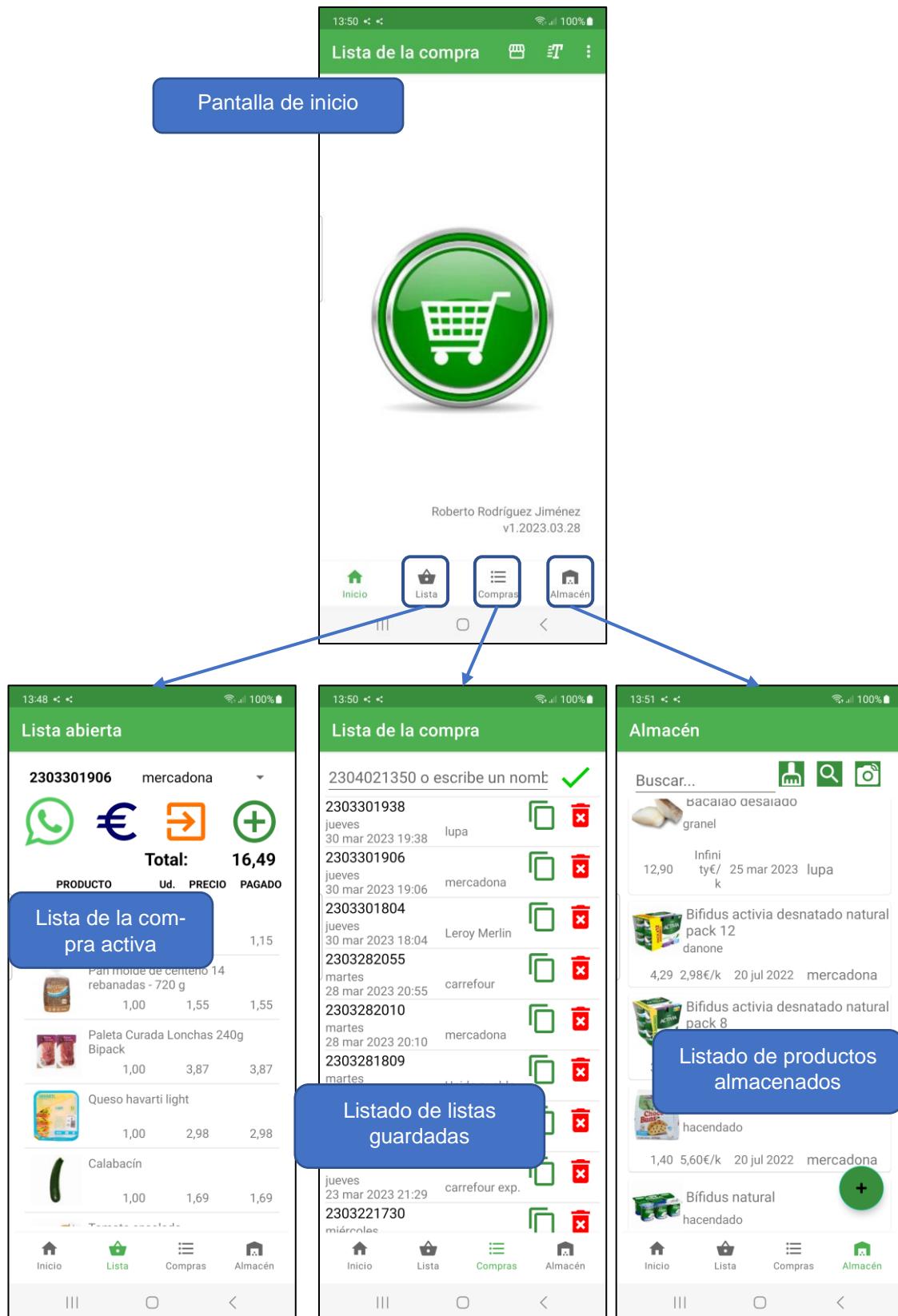
Importar solo los productos supone que se importan los registros guardados de los productos, sin ninguna relación. No se cargan, por tanto, ni marca ni etiquetas.

Si queremos importar todos los datos, la información almacenada en el dispositivo será eliminada, pues debe mantenerse la integridad de los datos.

Los productos guardados localmente, no son eliminados, en ningún caso. Para conseguirlo, simplemente se ha comentado la línea que llama al vaciado de la tabla `productos` en la clase `dam.proyecto.database.data.ImportDB#importarProductoEntity()`.

El método para importar los datos aprovecha la clase `ImportDB.class`, que se usa para la recuperación del backup local. Desde `ImportarEjemplos.importarTabla()` se cargan los registros de Tabla y se escriben en el fichero `TablaEntity.csv`. A continuación, de llama a `ImportDB.importarTabla()` que termina de realizar el trabajo, extrayendo los registros del fichero y escribiéndolos en la base de datos de *room*.

3.6 Interfaz de usuario

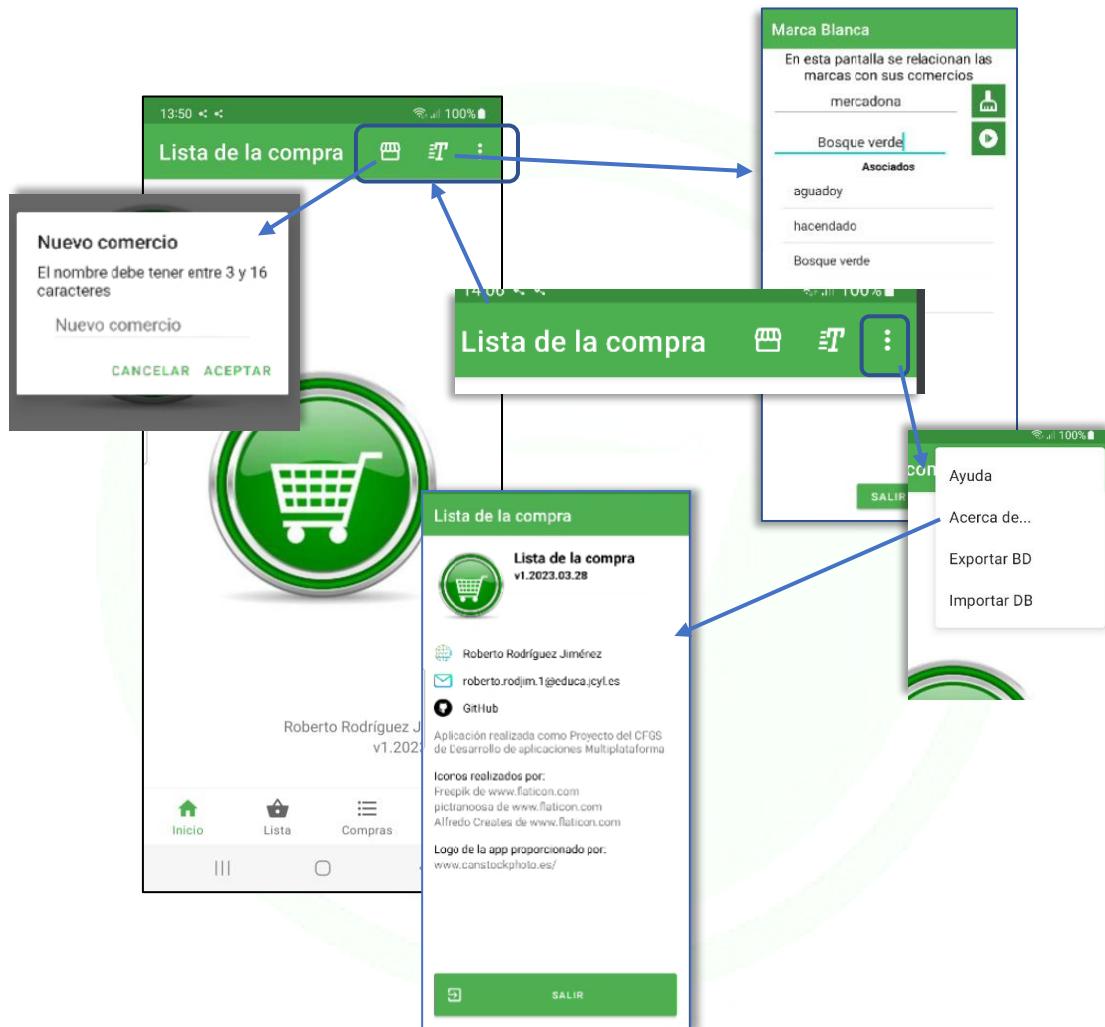


3.6.1 Inicio

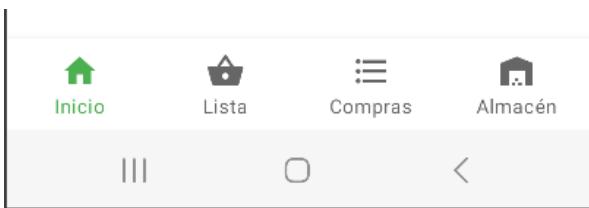


La pantalla de inicio tan solo muestra el logo de la aplicación, a modo de bienvenida.

Como elemento exclusivo de esta actividad, tenemos el menú del *toolbar*. Este menú no se muestra en ninguna otra pantalla.



Las opciones *Importar* y *Exportar* crean e importan el código para guardar y recuperar la base de datos. No tienen asociada una vista, ya que las operaciones se realizan de manera totalmente automatizadas.



BottomNavigationView

En la parte inferior de la pantalla se encuentra la barra de navegación entre las diferentes pantallas.

Las opciones nos llevan a las tres actividades principales de la aplicación: la lista de la compra activa , un listado de listas creadas  y el almacén con los productos guardados . El panel de navegación es visible en cada una de las cuatro actividades.

3.6.2 Lista abierta



Esta interfaz gestiona la lista de la compra que está siendo usada (lista abierta).

3.6.2.1 Lista

Nombre dado a la lista.
Si no se ha dado un nombre, coincide con la fecha y hora, que es, a su vez, el id de la lista.

Selector para el comercio. Puede cambiarse en cualquier momento.

Añade un producto a la lista.

Control para compartir la lista por WhatsApp.

Vuelve al listado de listas.

Muestra una comparativa de la lista comprada en otros comercios.

Clic largo:
Elimina el producto.

Vista de un producto:
Unidades, precio y total.
A efectos de la base de datos, cada registro se considera una fila.

Clic:
Edita la compra del producto.

PRODUCTO	Ud.	PRECIO	PAGADO
Naranja	1,00	1,15	1,15
Pan molde de centeno 14 rebanadas - 720 g	1,00	1,55	1,55
Paleta Curada Lonchas 240g Bipack	1,00	3,87	3,87
Queso havarti light	1,00	2,98	2,98
Calabacín	1,00	1,69	1,69

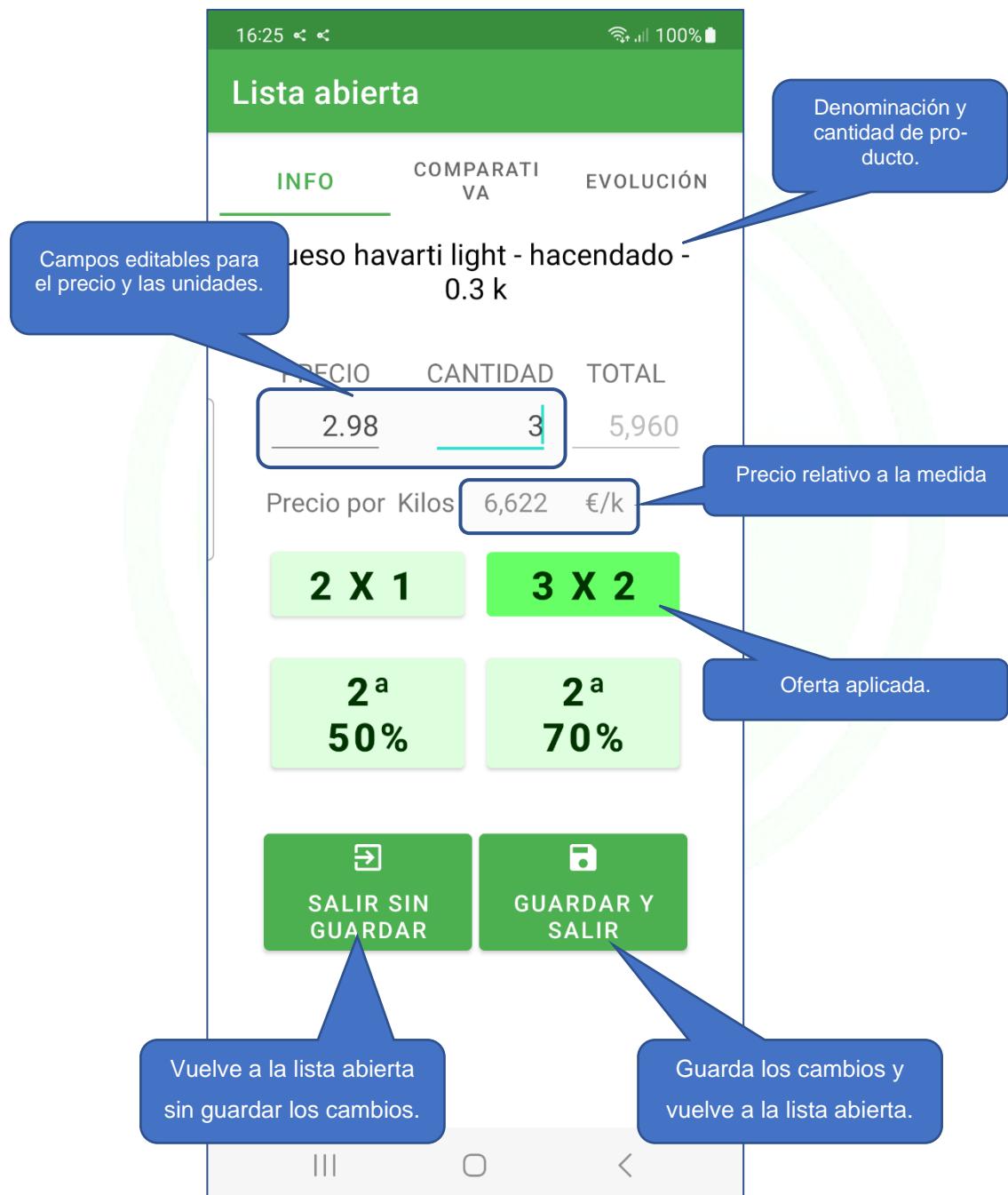
3.6.2.2 Detalle de la compra



3.6.2.2.1 *Información*

Pantalla de información desde la que indicar el precio, la cantidad o si se aplica alguna oferta. En esta captura se está aplicando una oferta de 3x2.

La vista muestra el precio del producto relativo a su unidad de medida.



3.6.2.2. Comparativa

Esta vista está dividida en dos zonas: una primera en la que comparan las últimas compras del mismo producto (mismo código de barras) y en la que el precio indicado es el precio del producto y una segunda en la que la comparativa se realiza entre productos que comparten la primera etiqueta. En esta zona, el precio mostrado es el relativo a su unidad de medida y se muestran todas las compras en cualquier comercio.

The image shows a smartphone screen displaying a mobile application interface for comparing prices. The top navigation bar includes icons for back, forward, and search, along with the time (10:35), signal strength, battery level (100%), and a notification icon. The main title is "Lista abierta". Below it, there are three tabs: "INFO", "COMPARATIVA" (which is currently selected and highlighted in green), and "EVOLUCIÓN".

COMPARATIVA

Naranja

Comercio	Precio	Fecha
mercadona	1,15	30 mar 2023
carrefour	1,14	22 mar 2023

Productos relacionados con naranja

Producto	Comercio	Fecha	Precio
naranja	mercadona	30 mar 2023	0 €/k
naranja	carrefour	22 mar 2023	inf €/k
naranja	mercadona	21 mar 2023	0 €/k
naranja	mercadona	16 mar 2023	inf €/k
naranja	mercadona	25 feb 2023	0,99 €/k

Below the table are navigation icons: three vertical dots, a square, and a left arrow.

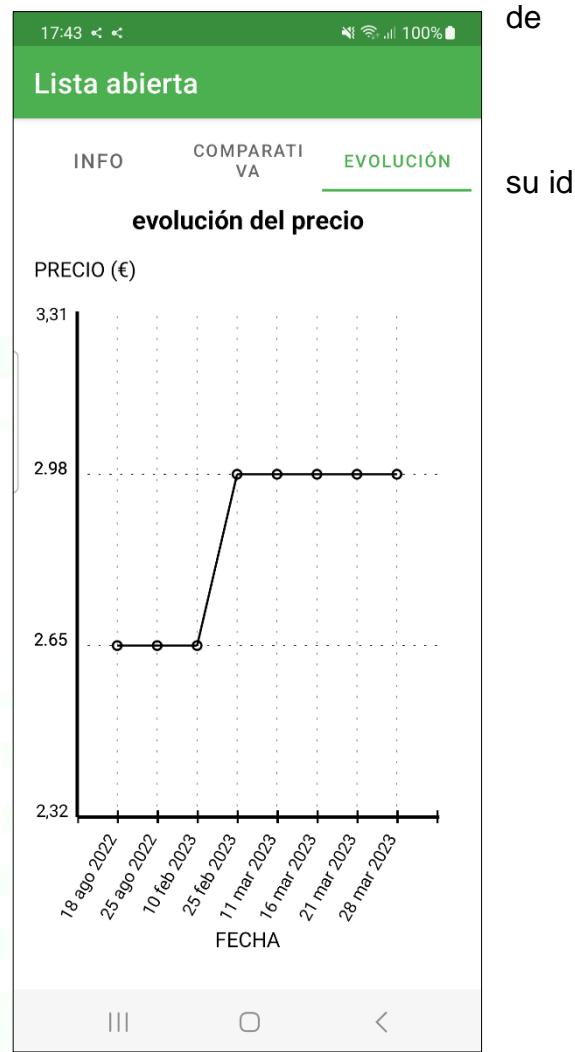
Annotations:

- A blue speech bubble points to the first table: "Se comparan las compras del mismo producto en diferentes comercios".
- A blue speech bubble points to the second table: "Se comparan diferentes productos que comparten la etiqueta principal (primera)".

3.6.2.2.3 Evolución

Muestra la evolución de un precio a través las compras realizadas. No se tiene en cuenta el comercio en el que se ha realizado y el producto se selecciona por (código de barras).

El gráfico está pintado sobre un *canvas*.



Evolución de los precios

3.6.2.3 Comparativa de precios por comercio

→

Una de las funcionalidades de la app es la de comparar los precios de una lista con esa misma lista en otros comercios.

Con una lista abierta, pulsamos sobre el símbolo €

Número de artículos coincidentes

Se carga una actividad en la que se muestra la lista en diferentes comercios.
Para ver los demás comercios, hay deslizar la pantalla a izquierda y derecha.

abierta

Comparativa de comercios

ALDI

Artículos:	2	Total:	8,86
Desde	3 mar 2023	Hasta	13 mar 2023
Cebolla	2,21	3 mar 2023	
Lomo adobado	6,65	13 mar 2023	

Rango de fechas en las que se compraron los productos

Deslizar la pantalla

Lista abierta

Comparativa de comercios

CARREFOUR

Artículos:	1	Total:	2,49
Desde	22 mar 2023	Hasta	22 mar 2023
Cebolla	2,49	22 mar 2023	

MERCADONA

Artículos:	22	Total:	50,87
Desde	8 abr 2023	Hasta	11 abr 2023
Huevos camperos	2,69	8 abr 2023	
Bifidus natural 0.0	1,45	8 abr 2023	
Queso havarti light	2,98	11 abr 2023	
Pechuga de pavo baja sal	3,37	8 abr 2023	
Servilleta papel	1	8 abr 2023	
Agua Mineral Natural Aguadoy 8L	0,86	8 abr 2023	
Ensalada 4 estaciones	0,72	11 abr 2023	
Pimiento asado tiras	1,1	8 abr 2023	
Leche desnatada	5,34	8 abr 2023	
Pai molde de centeno 14 rebanadas - 720 g	1,55	8 abr 2023	
Pañecillo horno paquete 11 u - 495 g	1,1	11 abr 2023	
Zumo de melocotón y uva 6x200ml	1,6	8 abr 2023	

DIA

Artículos:	1	Total:	2,29
Desde	22 mar 2023	Hasta	22 mar 2023
Cebolla	2,29	22 mar 2023	

LIDL

Artículos:	1	Total:	6,85
Desde	13 mar 2023	Hasta	13 mar 2023
Lomo adobado	6,85	13 mar 2023	

Lista abierta

SALIR

[Inicio](#) [Lista](#) [Compras](#) [Almacén](#)

Otras opciones de la misma lista

3.6.3 Listado de compras

≡

Esta pantalla muestra la relación de listas que se han ido creando.

Para cada uno de los registros se da la opción de *clonar* o eliminar la lista.

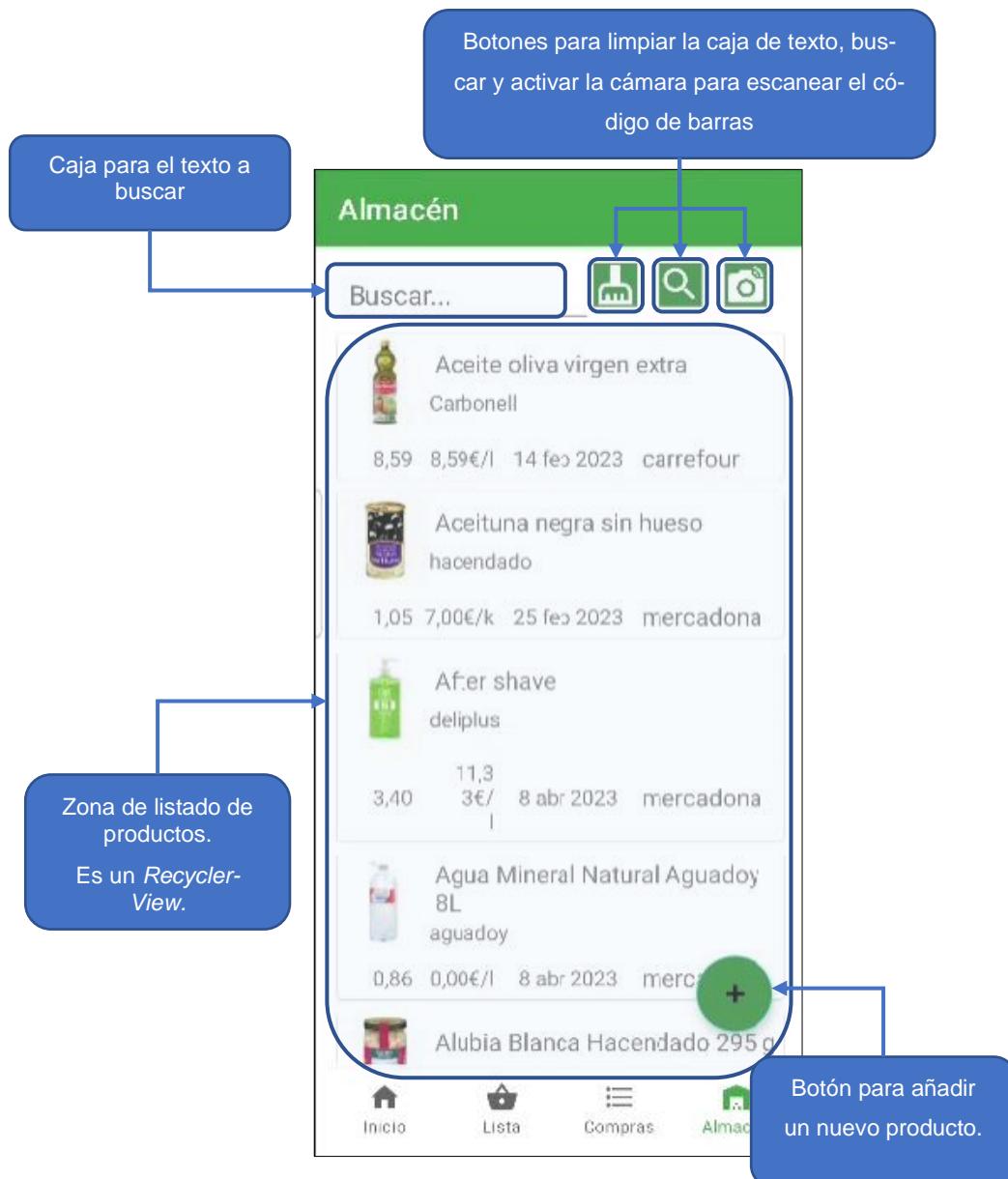
Clonar significa que se guarda una lista exacta a la clonada, pudiendo ser editada posteriormente.



3.6.4 Almacén

~

3.6.4.1 Almacén lista



3.6.4.2 Almacén detalle

RL

Al detalle de un producto se accede con un LongClick sobre dicho producto en la lista.



3.6.5 Ayuda

Desde el menú del *ToolBar*, en la activity de *Inicio*, se accede al sistema de ayuda. El sistema de ayuda se compone de diferentes archivos *html*, uno para cada tema de ayuda. El componente encargado de su visualización en un *WebView*.



Menú de ayuda desde el *ToolBar*

La interfaz se compone de un fragment de lista y de otro de detalle, en el cual se carga el contenido del tema seleccionado.

Ayuda

Lista

- Crear una lista de la compra
- Abrir una lista de la compra
- Añadir un producto a la lista
- Eliminar un producto de la lista
- Editar la cantidad y el precio
- Aplicar ofertas
- Comparativa de precios
- Evolución del precio
- Comparar la lista en otros comercios
- Compartir una lista de la compra

Compras

- Modificar una lista
- Crear una lista
- Eliminar una lista
- Reutilizar una lista (duplicar)

Almacén

- Agregar un producto al almacén
- Editar un producto
- Eliminar un producto

Otros

-
-
-
-

Ayuda

Agregar un producto a la lista

- Pulsar sobre el botón de añadir que nos lleva al almacén.
- Desde el almacén buscamos el producto que queremos añadir. Ver [Buscar producto](#).
- Un click sobre el producto buscado abre un diálogo que nos permite *prefijar* un precio:
 - Ningún precio,
 - Último precio conocido.
 - Último precio en el comercio seleccionado en la lista.
 - Al pulsar *Aceptar* se carga el producto en la lista con el precio seleccionado.

La vista del detalle cuenta con un botón flotante para volver al listado en cualquier momento.

Vista de la lista y el detalle de los temas de ayuda

roberto.rodjim.1@educa.jcyl.es

66

Roberto Rodríguez Jiménez

3.7 Funcionamiento

~

3.7.1 Inicio

Clase que carga la actividad: `activities/MainActivity.java`

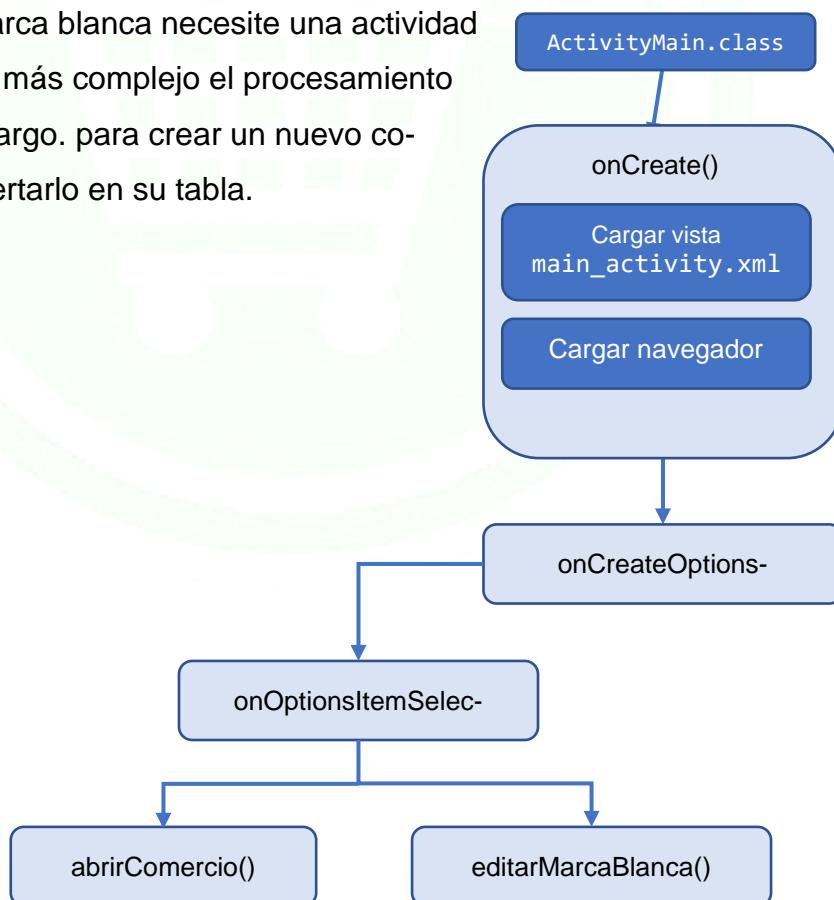
Vista: `layout/activity_main.xml`

Esta clase tan solo carga el `BottomNavigationView` y el `MenuToolBar`.

A parte de los métodos para controlar los dos menús, se han declarado otros dos para abrir sendas opciones del menú principal: crear un nuevo comercio y asociar las marcas blancas.

Para crear un nuevo comercio se crea un `AlertDialog` y se llama a la actividad `MarcaBlancaActivity.class` para asociar la marca blanca con su comercio.

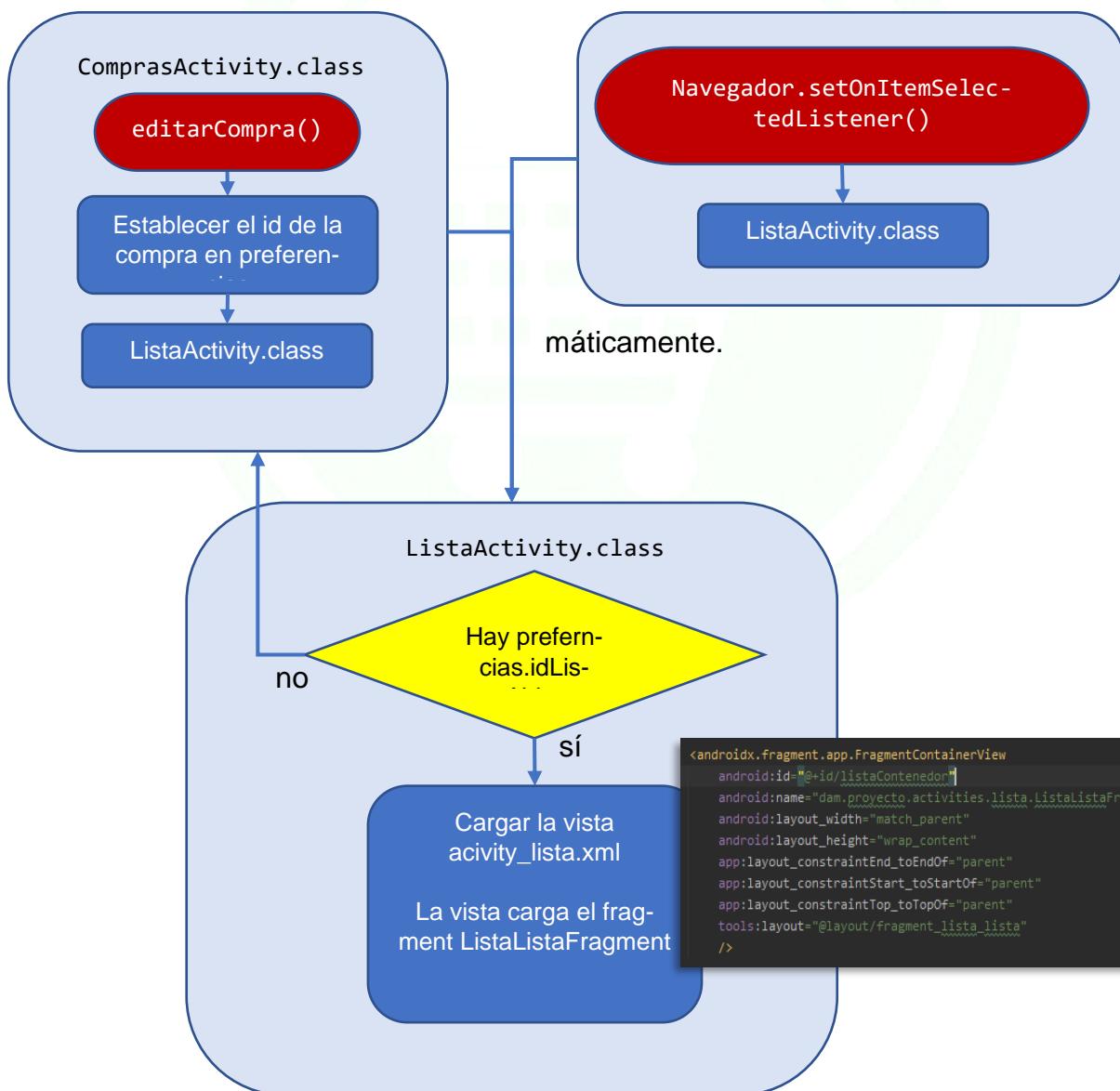
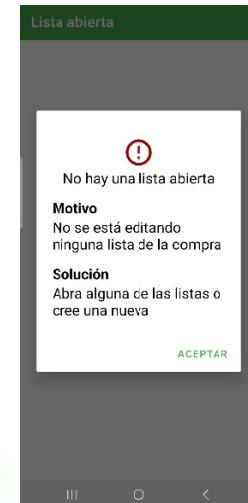
El hecho de que la marca blanca necesite una actividad se debe a que resulta más complejo el procesamiento de los datos. Sin embargo, para crear un nuevo comercio, basta con insertarlo en su tabla.



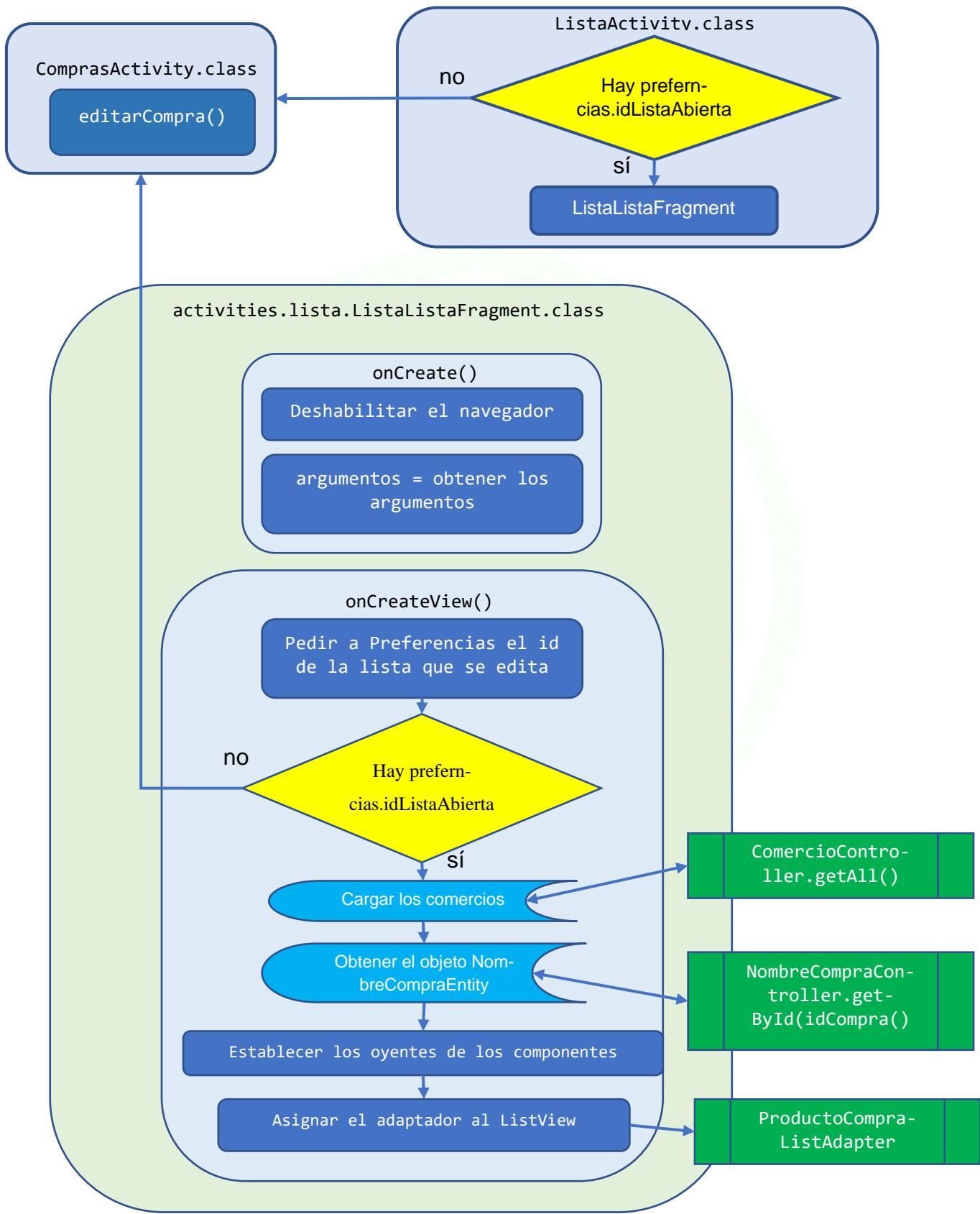
3.7.2 Lista de la compra (lista abierta)

Para abrir una lista de la compra hay varias formas: podemos abrirla directamente pulsando en el menú inferior o desde el listado de compras, pulsando sobre la lista que queramos abrir.

En el caso de querer entrar directamente en la lista abierta y que no hay ninguna disponible, se muestra un mensaje indicando la circunstancia y cargando, al aceptar el mensaje, la actividad de *Compras*. Cuando se crea o duplica una compra, se edita auto-

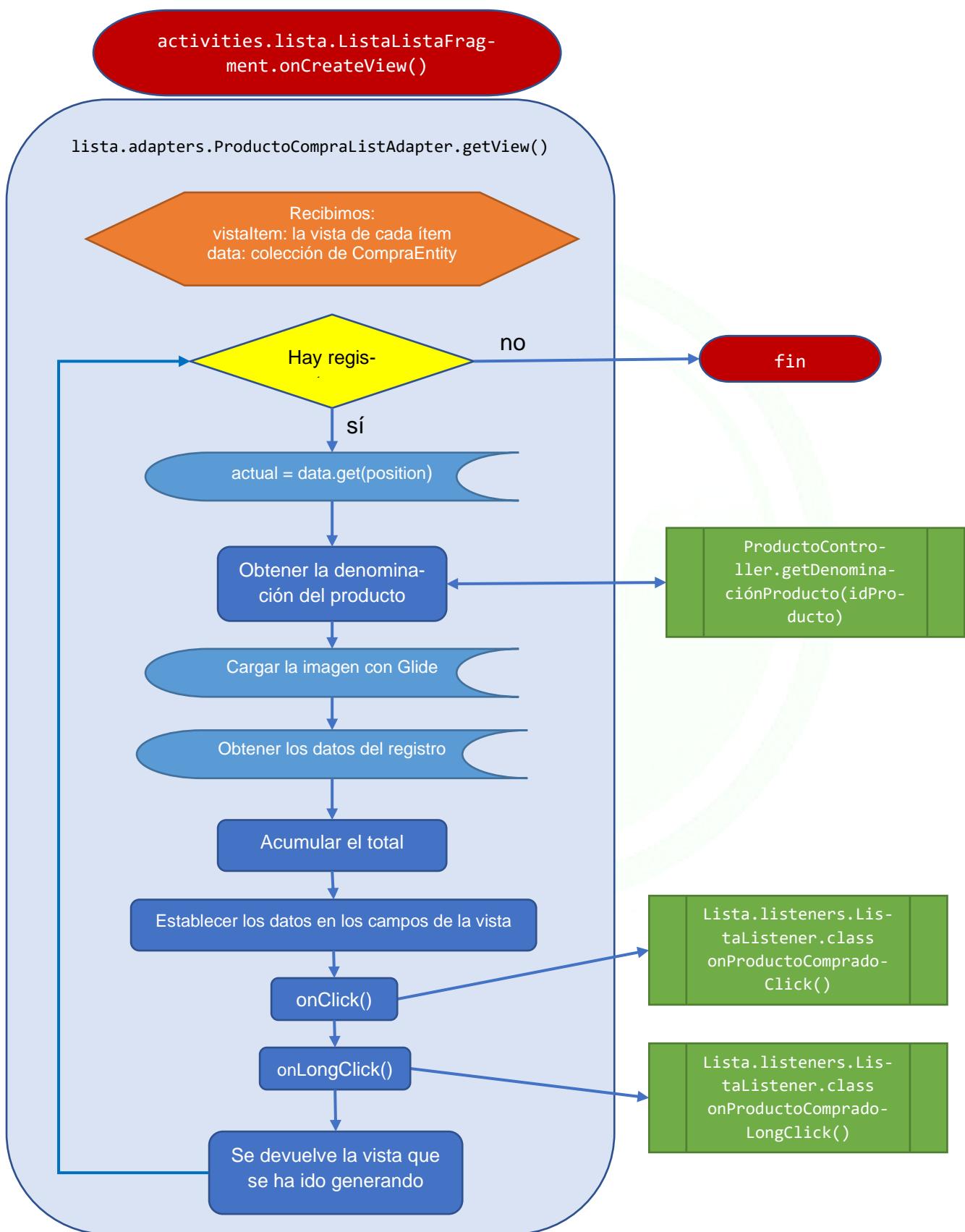


3.7.2.1 Cargar la lista de la compra



3.7.2.2 Adaptador de la lista abierta

22

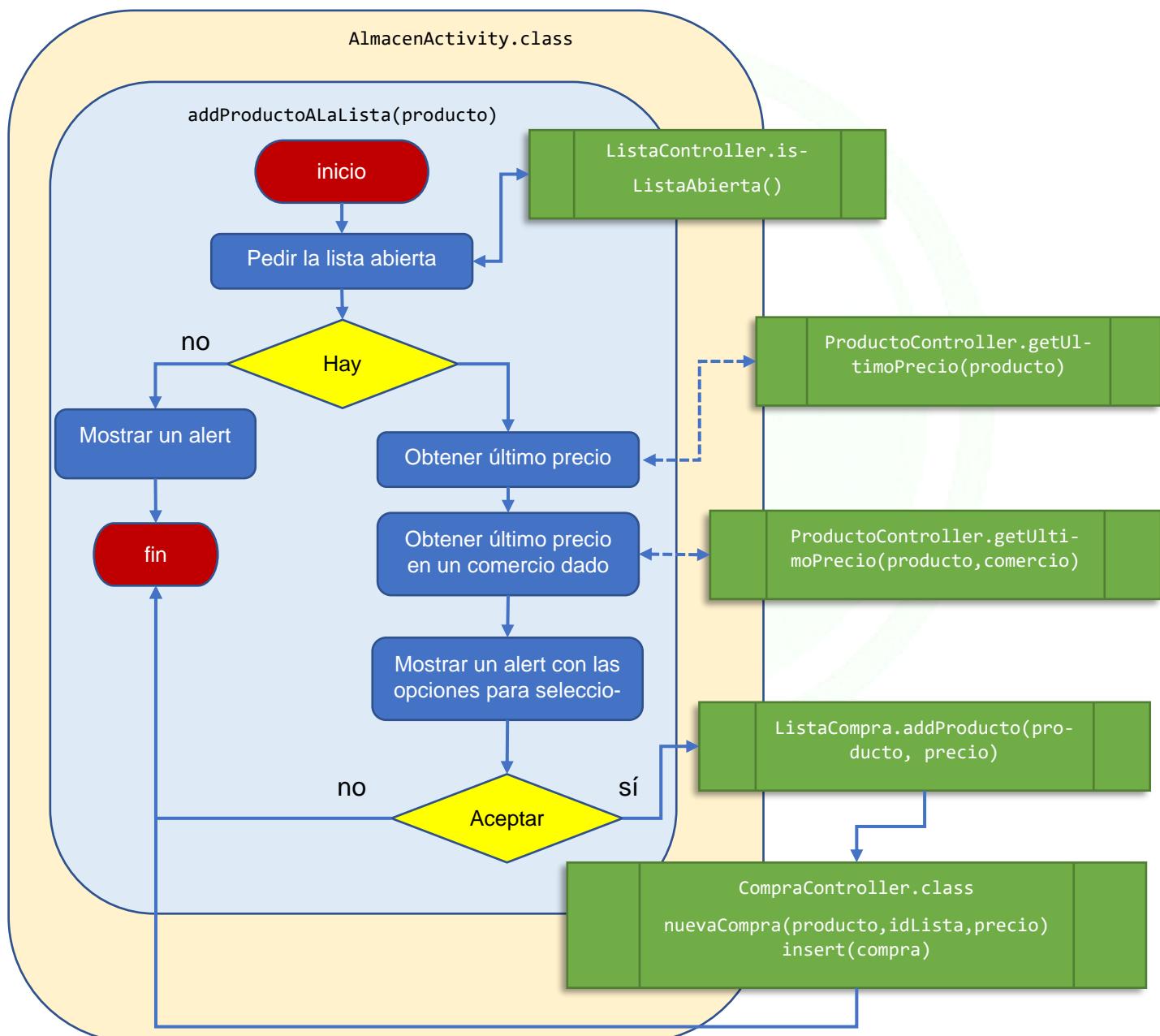


3.7.2.3 Añadir un producto a la lista

~

Desde la vista *Almacén*, mostrándose los producto, con un click sobre el producto elegido, se abre un diálogo para seleccionar el precio con el que se quiere guardar: sin precio, el último precio comprado o el último precio del producto en el comercio del a lista. En caso de no haber una lista, se muestra un mensaje.

Clase: dam/proyecto/activities/almacen/AlmacenActivity.java

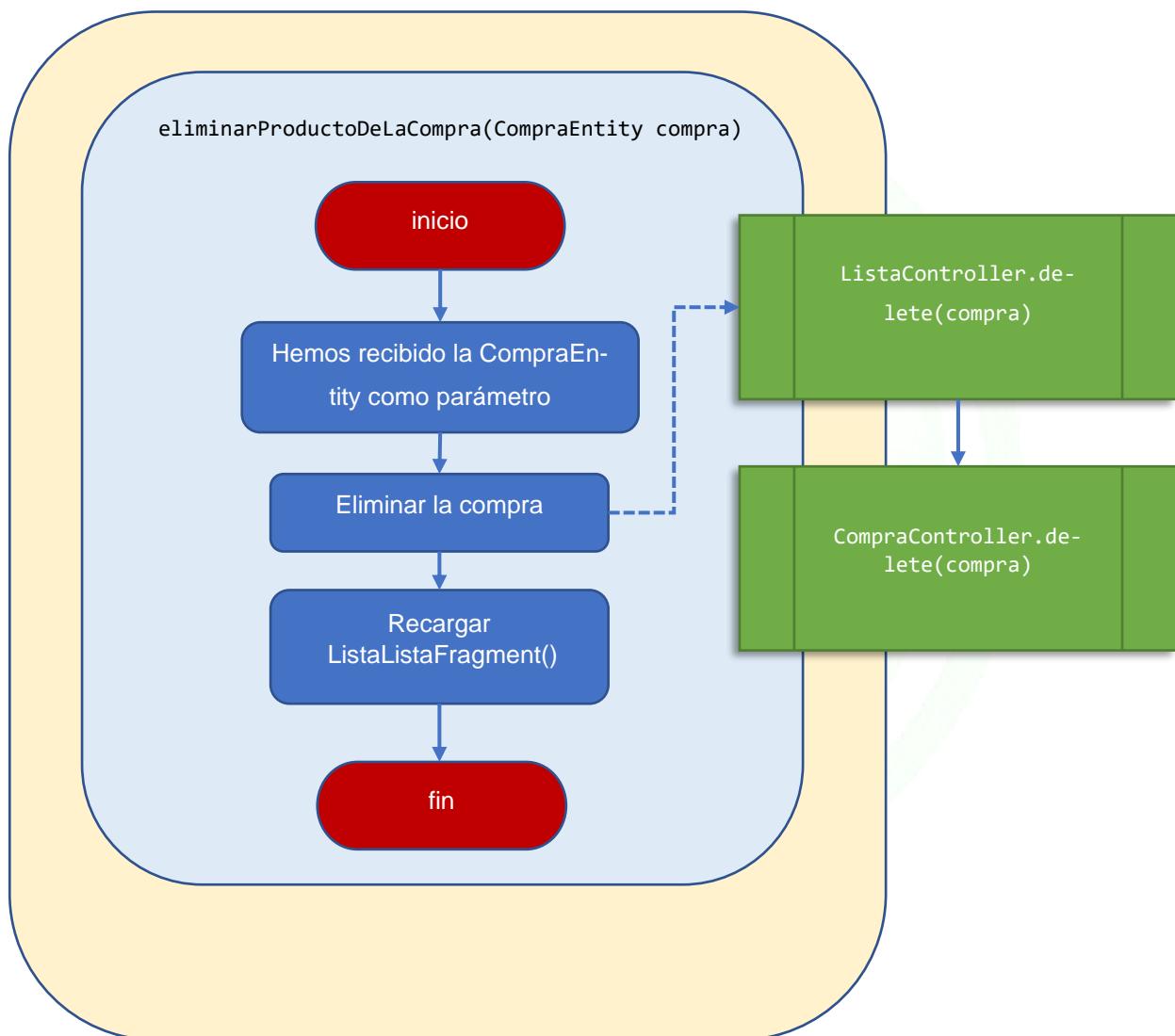


3.7.2.4 Eliminar producto de la lista

~ ~

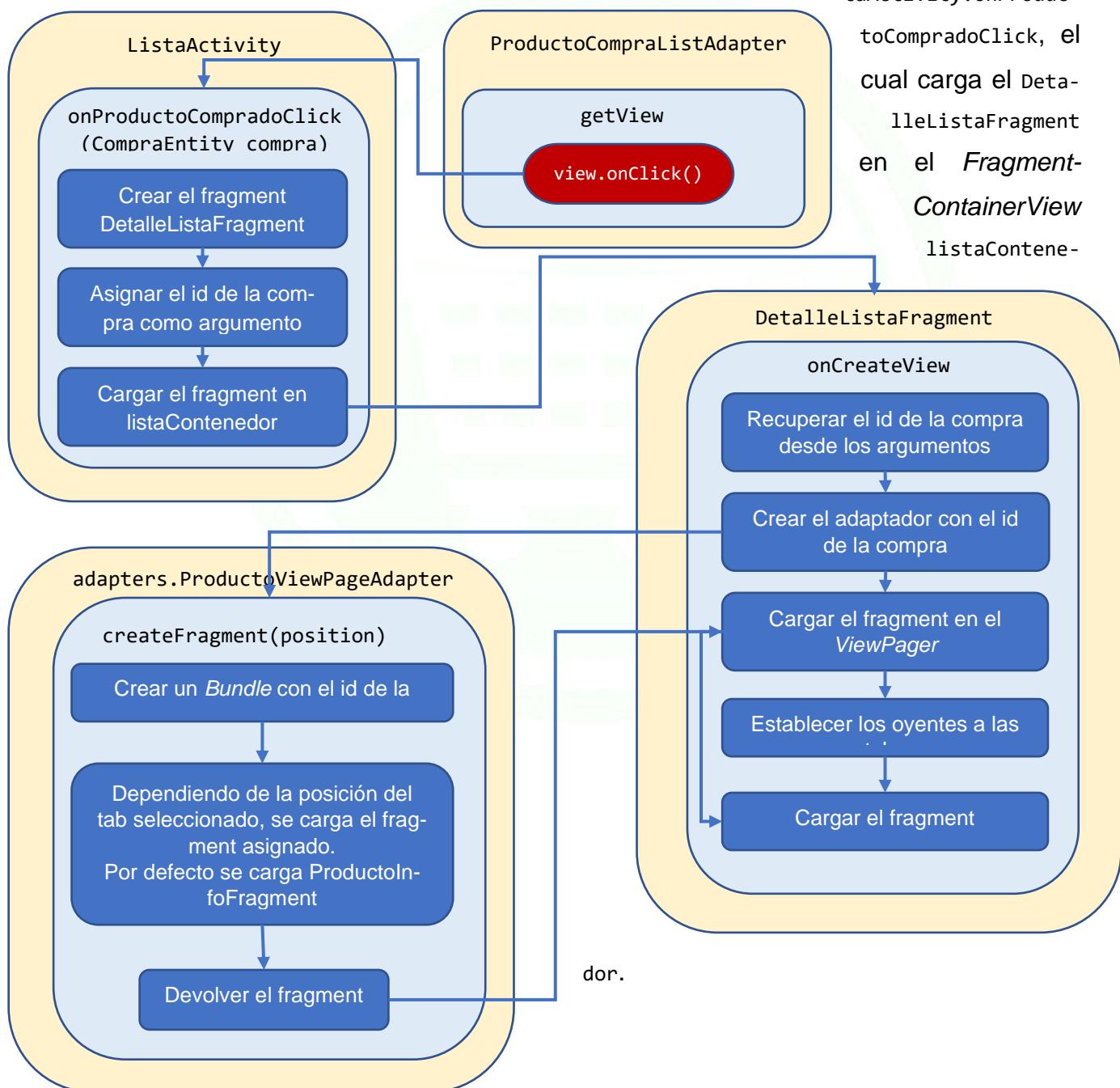
Al realizar un *Long Click* sobre el producto, se abre un *AlertDialog* preguntando si queremos eliminar el producto. El botón *Aceptar* desencadena el borrado llamando a `eliminarProductoDeLaCompra()`.

Clase: `dam/proyecto/activities/lista/ListaActivity.java`



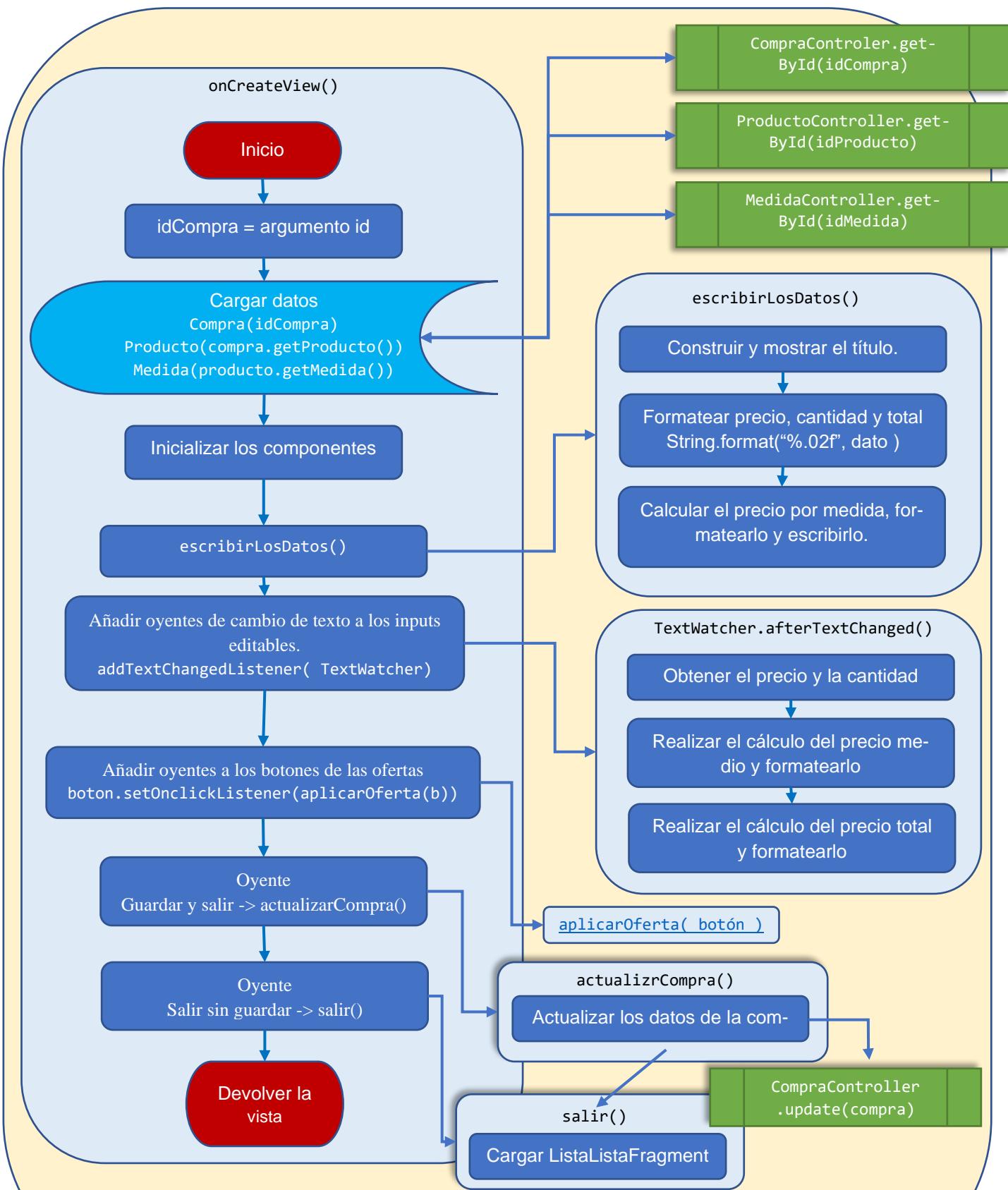
El detalle de la lista se compone de tres fragmentos: [información](#), [comparativa](#) y [evolución](#).

Desde `ListaActivity` se carga `DetalleListaFragment`. Este fragmento contiene un `TableLayout` que será el componente que carga cada uno de los fragmentos seleccionados. La carga del detalle se inicia en el adaptador `ProductoCompraListAdapter`, al clicar sobre uno de los productos de la lista. Se desencadena el evento `onClick` y se llama a `ListaActivity.onProductoCompradoClick`.

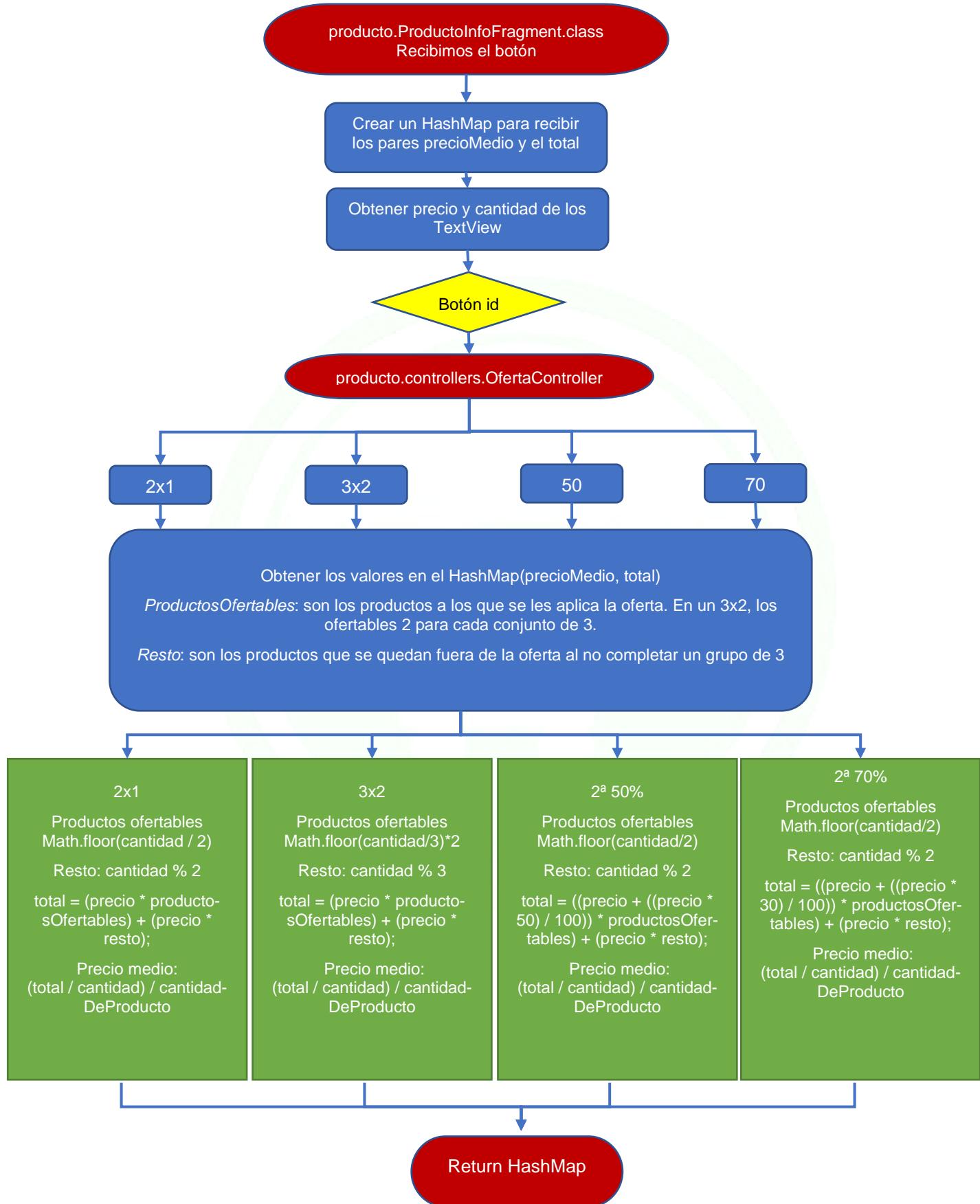


3.7.2.5.1 Producto Info

...



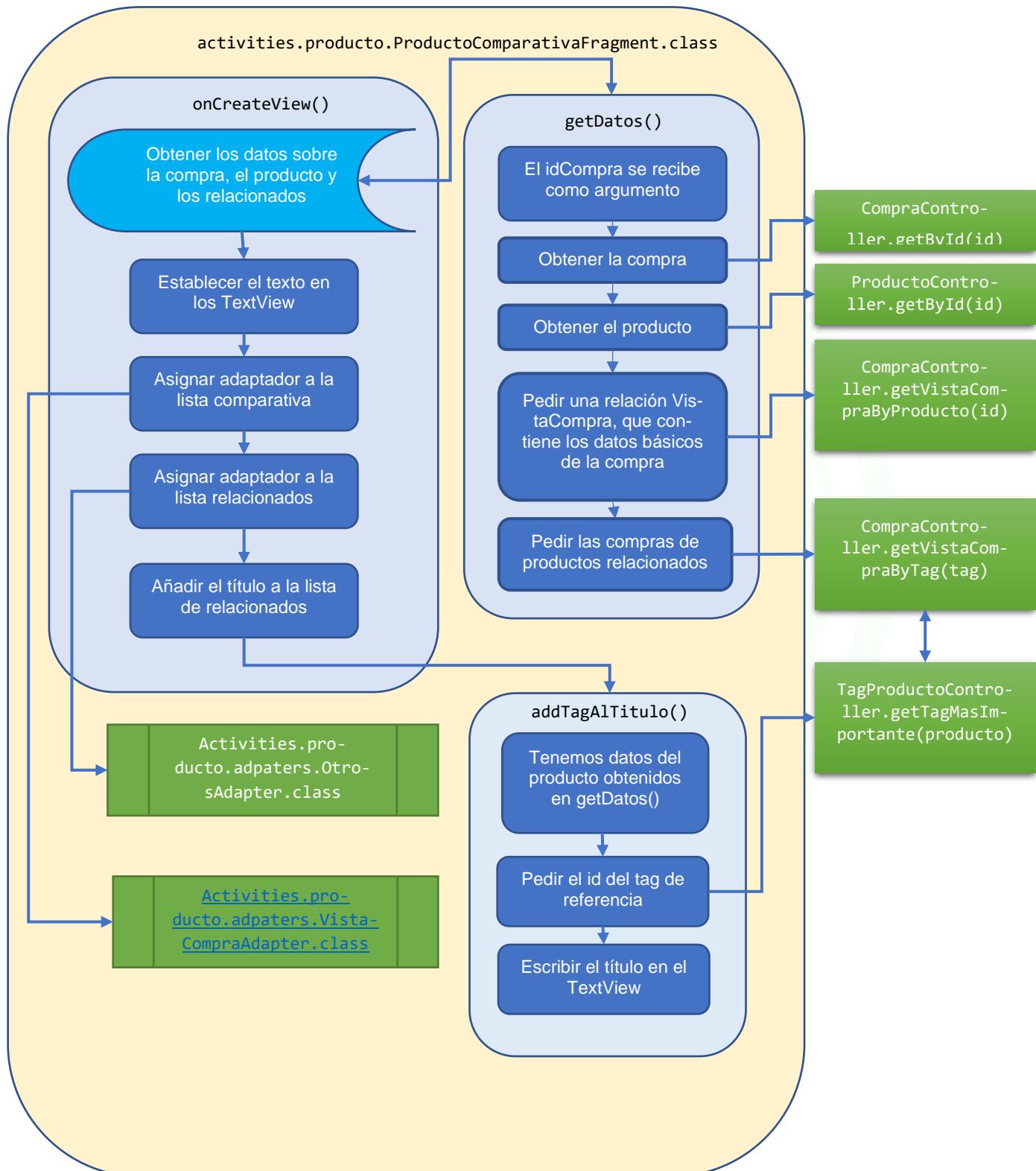
3.7.2.5.1.1 Aplicar oferta



The screenshot shows the Android Studio interface with the following details:

- Project Tab:** Shows the file structure with `InfoFragment.java` and `OfertaController.java` open.
- Code Editor:** Displays the Java code for `InfoFragment.java`. The code implements a `get3x2` method that calculates a discount for quantities divisible by 3. It logs the total products, quantity, and price, and returns a map with the results.
- Logcat Tab:** Shows log output from the device. It includes a process start message, several log entries, and a crash message starting with "----- beginning of crash".
- Device Manager Tab:** Shows a running device (samsung SM-G991B) with the app's interface displayed. The interface is a grocery store app showing a product detail screen for "Crema queso light 250 - Philadelphia - 0.5 k". It shows a table with columns PRECIO, CANTIDAD, and TOTAL. It also displays two offer options: "2 X 1" and "3 X 2", each with a 50% discount.
- Bottom Bar:** Includes tabs for Git, Run, Profiler, Logcat, App Quality Insights, Build, TODO, Problems, Terminal, Services, App Inspection, and Layout Inspector.

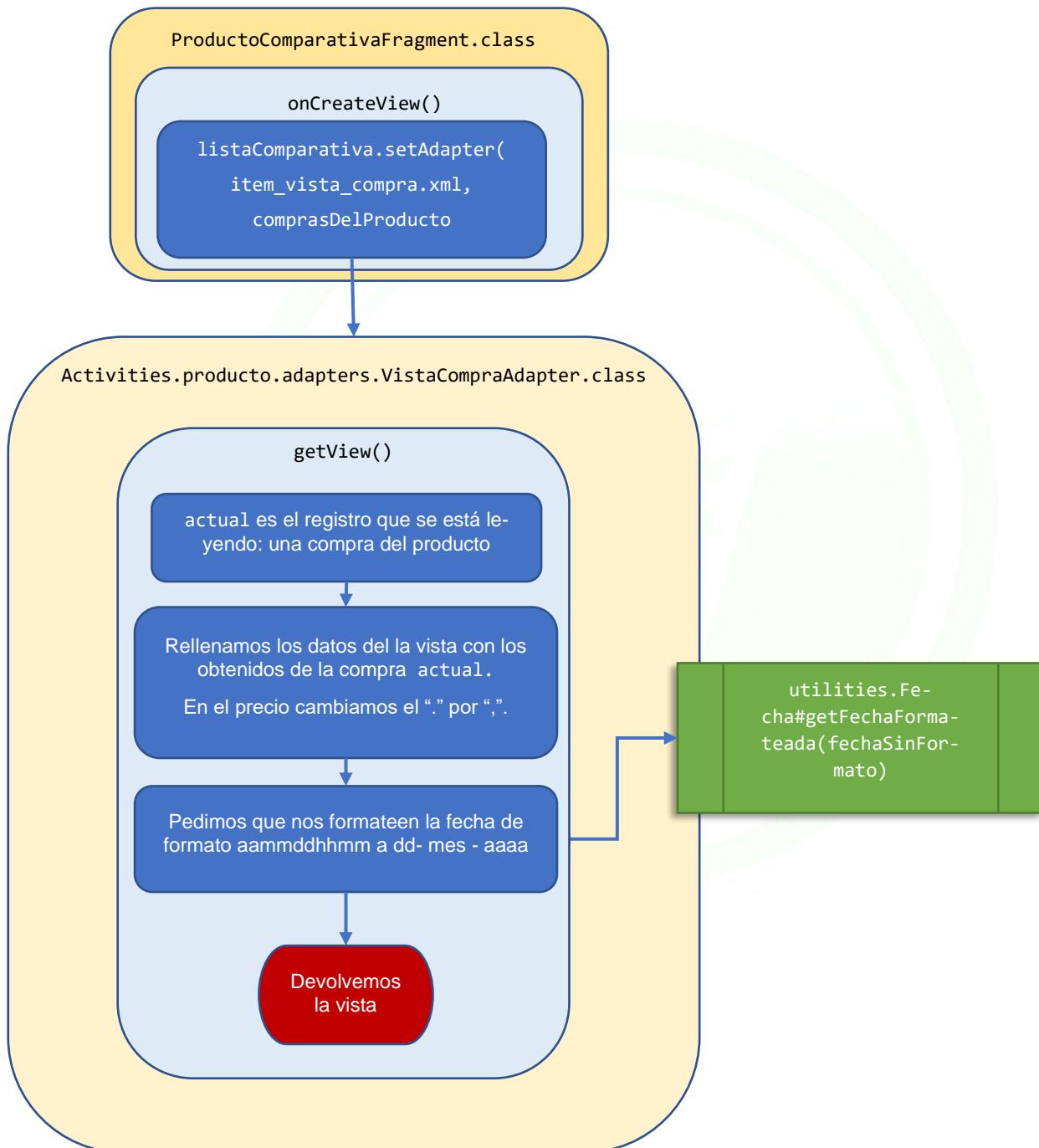
Captura de pantalla de la función `get3x2` que aplica los cálculos a la oferta


[Ver la interfaz de usuario](#)


3.7.2.5.2.1 VistaCompraAdapter

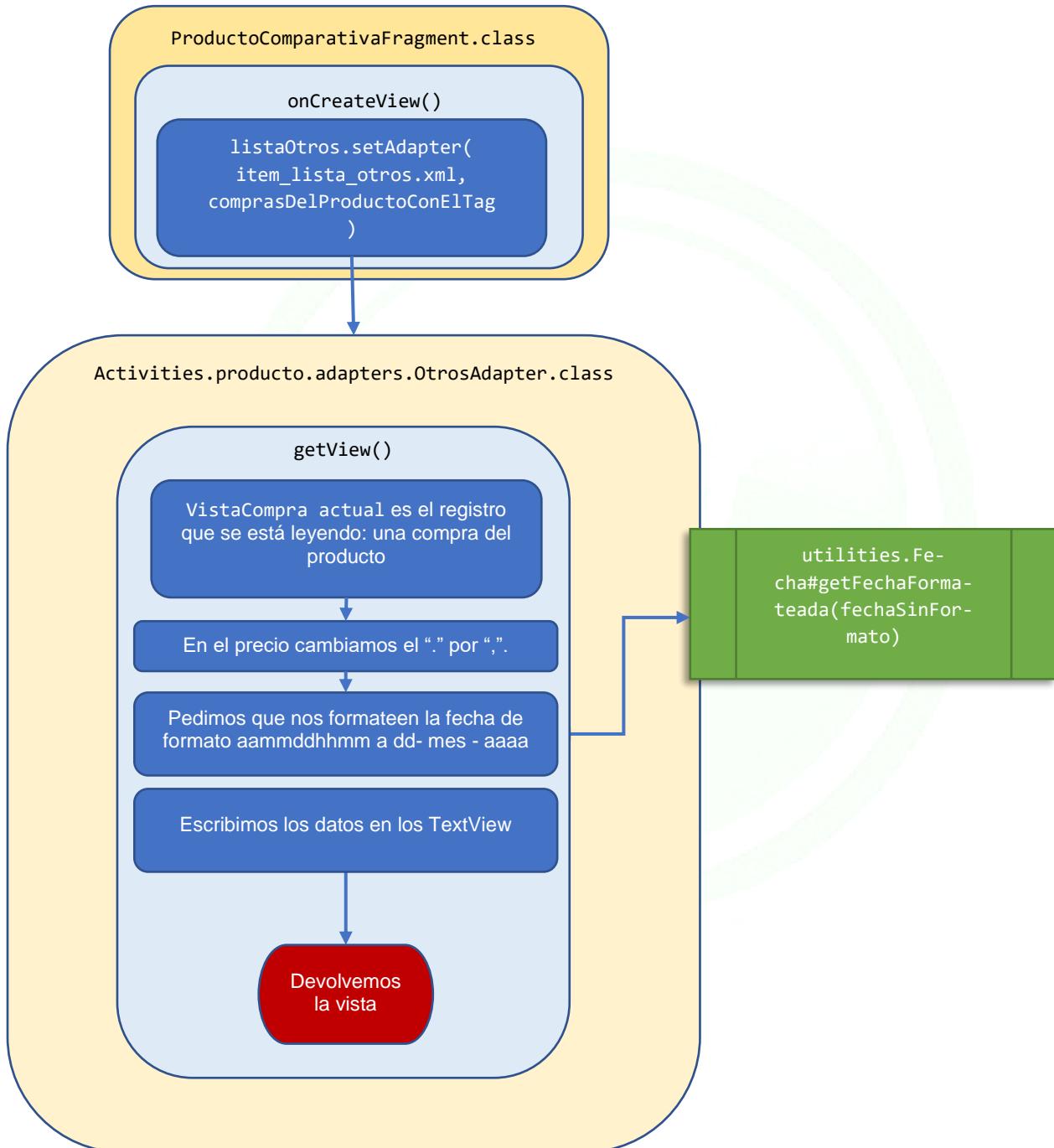
↖↖

Muestra un listado de las compras realizadas de un mismo producto. Por producto se entiende que tienen un mismo identificador único (código de barras).





El listado muestra las compras realizadas de productos que comparten el tag más importante (el primero).




[Ver Interfaz gráfica](#)

Clase: dam/proyecto/activities/producto/ProductoEvolucionFragment.java

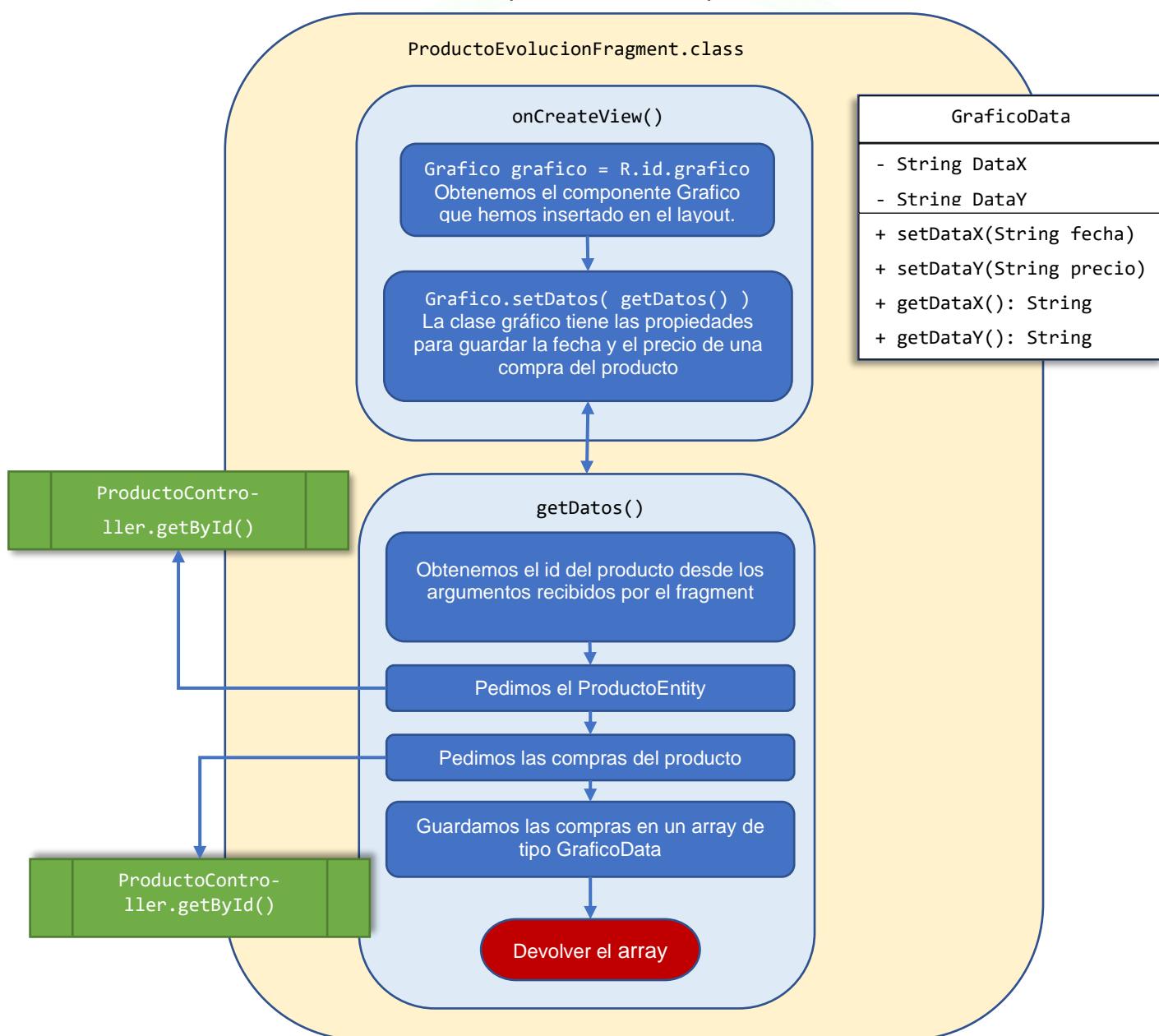
Clase gráfico: dam/proyecto/activities/producto/classes/Grafico.java

Clase para los datos: dam/proyecto/activities/producto/classes/GraficoData.java

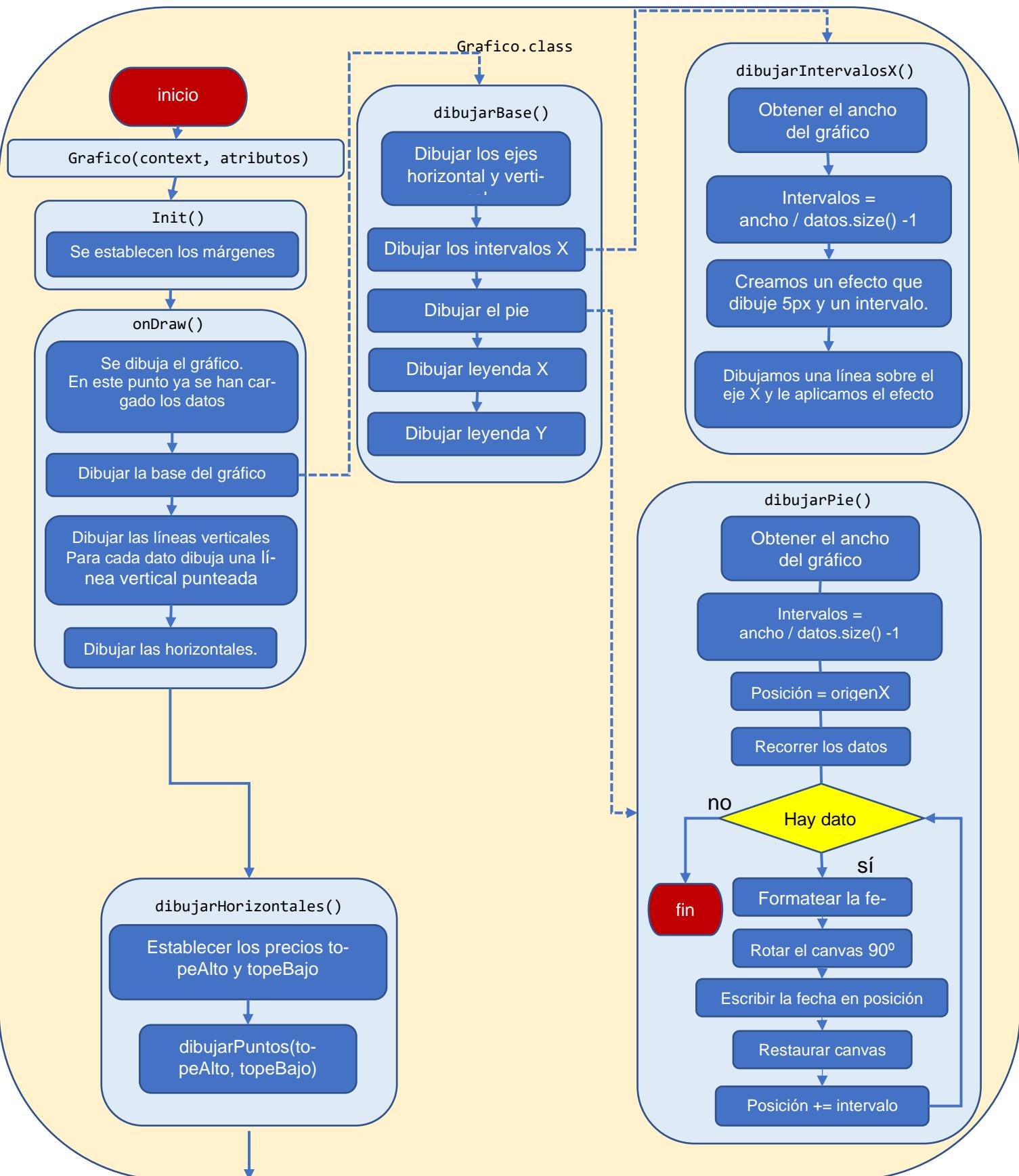
Vista: res/layout/fragment_producto_evolucion.xml

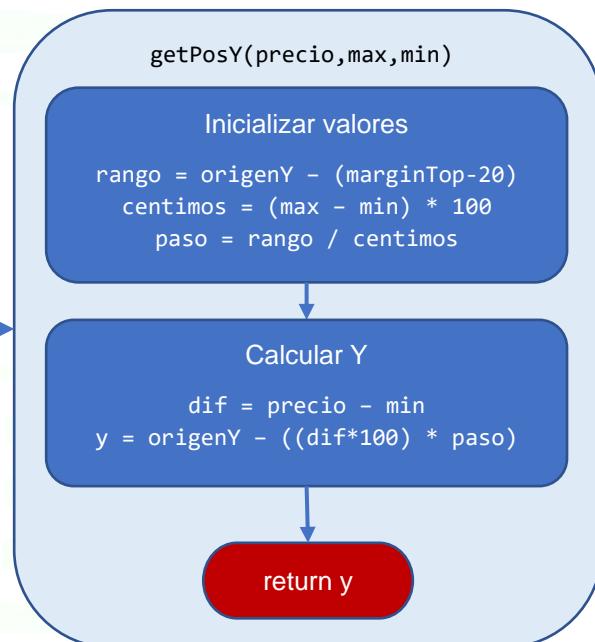
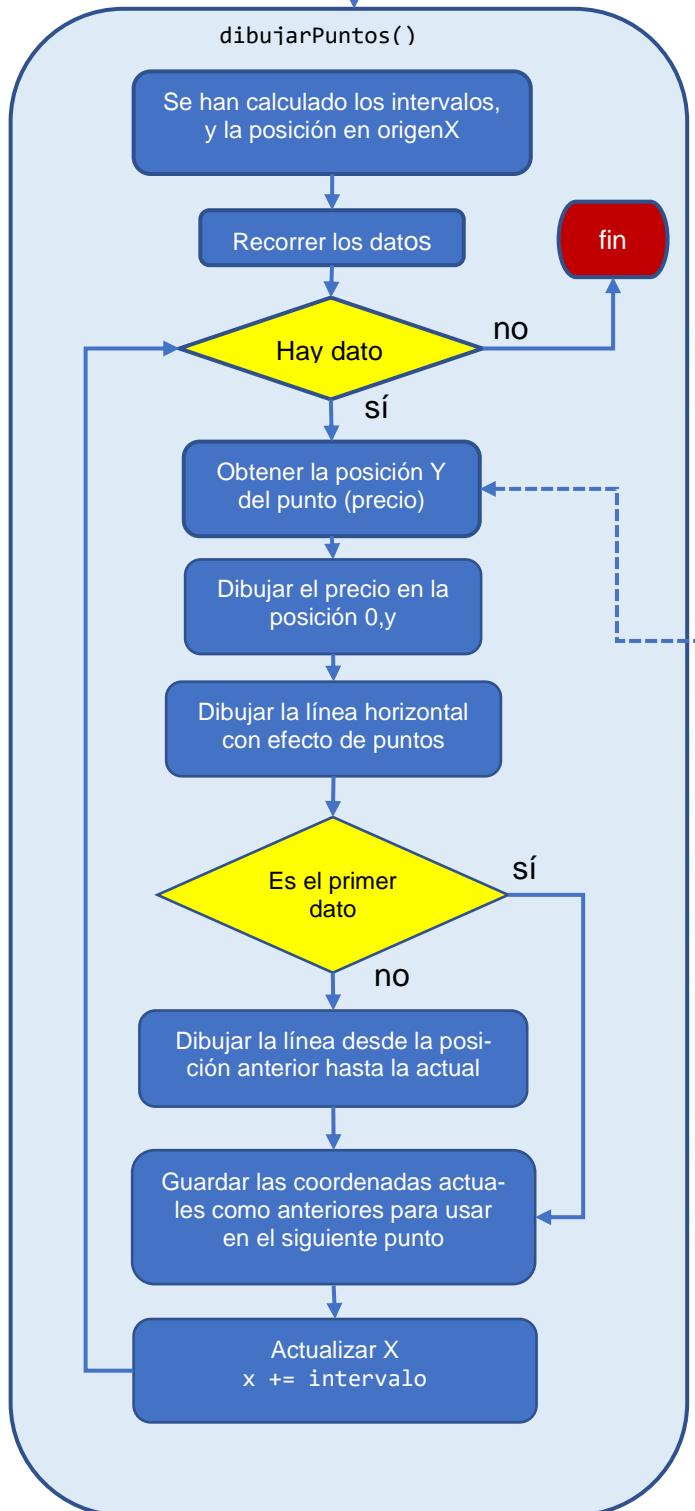
La gráfica trata de mostrar la evolución del precio de un producto concreto.

La vista es mostrada desde la opción *Lista* en la pestaña *Evolución*.



El componente que carga el gráfico es un *canvas* y toda la vista está mostrada por código. El gráfico está representado por la clase *Grafico.class*, que extiende de *View*.



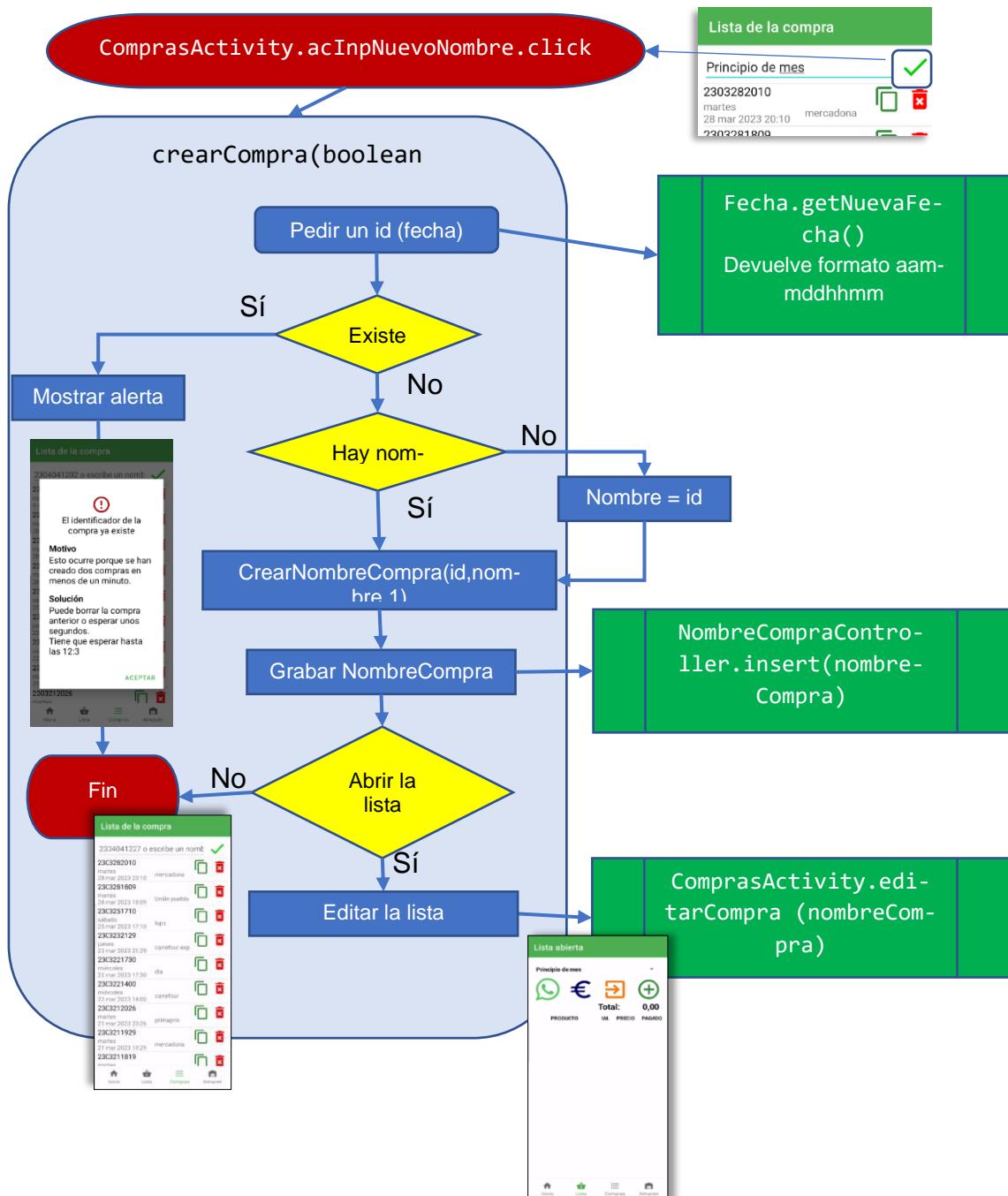


3.7.3 Listados

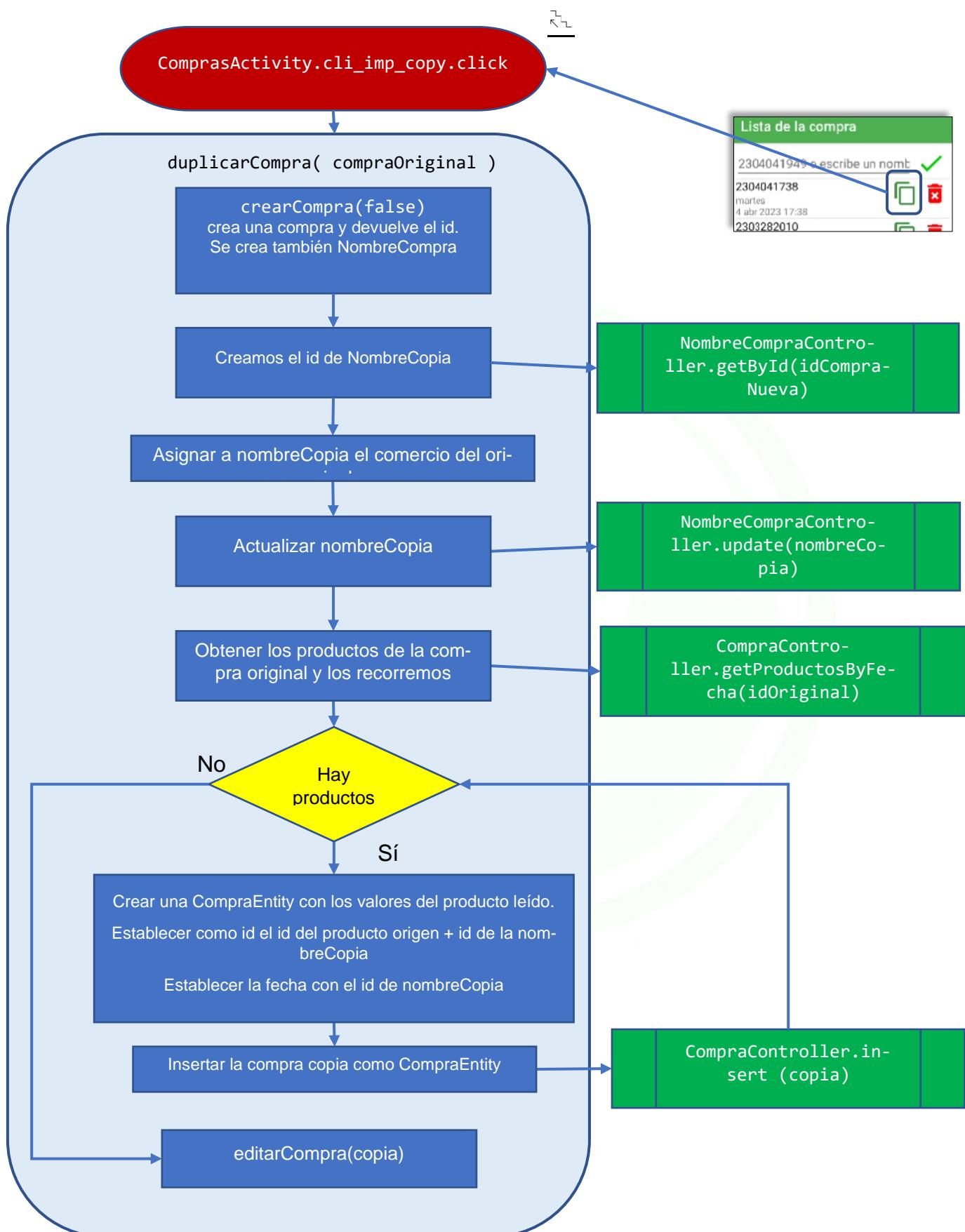
→

3.7.3.1 Crear un listado

Se crea el listado nuevo con el nombre que se ha introducido en la caja de texto. En caso de no haberse introducido ningún nombre, el nombre dado será el formado por aammddhhmm, dato que sirve como identificador de la lista de la compra que se está creando.

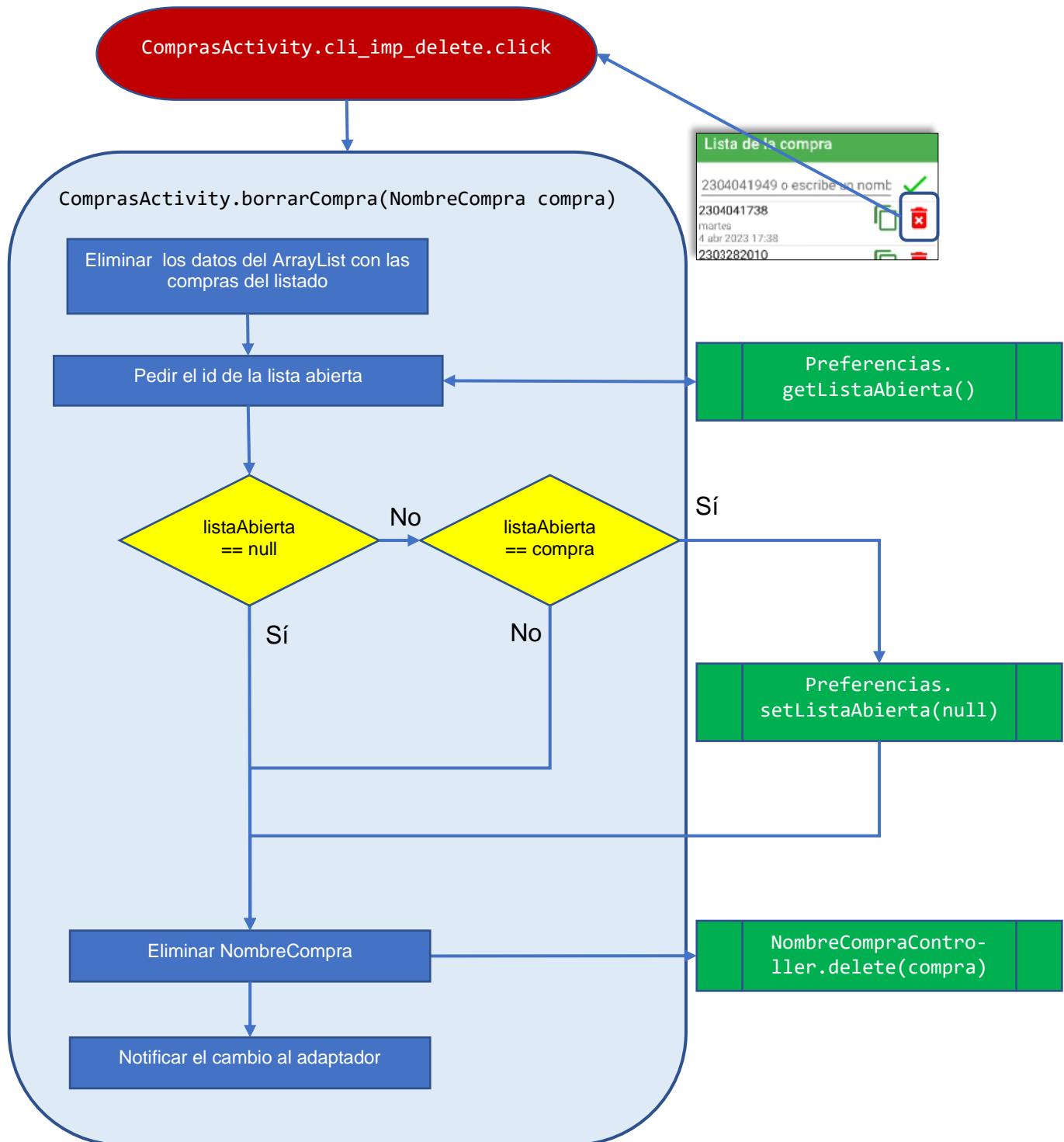


3.7.3.2 Duplicar un listado



3.7.3.3 Eliminar un listado

...



3.7.3.4 Cambiar la fecha de una lista



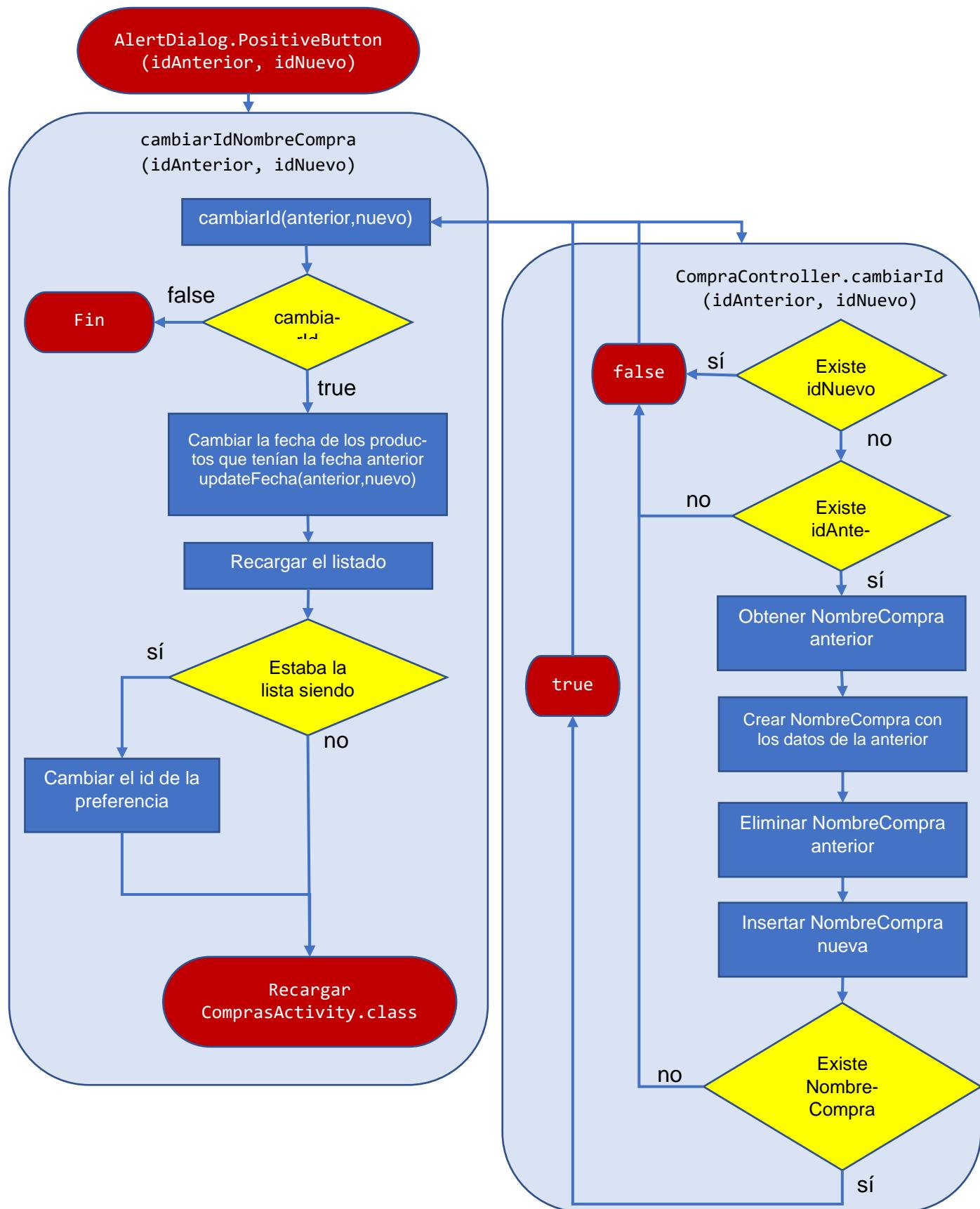
Se puede crear una lista en cualquier momento, incluso días o meses después de la compra cuando, por ejemplo, se encuentra un antiguo ticket. La lista crea un identificador único compuesto por la fecha, hora y minutos del momento en el que se crea, por lo que la lista guardada aparecerá en primer lugar, con la fecha actual.

Para solucionar este problema, se da la opción de cambiar el identificador de la compra y, con ello, su fecha.

The screenshot illustrates the steps to change the date of a receipt:

- 1 encontramos un recibo de hace un año.** (We find a receipt from a year ago.) A blue callout points to a screenshot of a receipt from May 2, 2022, at 17:47:46.
- 2 creamos la lista.** (We create the list.) A blue callout points to a screenshot of the "Lista de la compra" screen where a new list is being created.
- 3 se ha creado la lista con la fecha actual** (The list has been created with the current date) A blue callout points to a screenshot of the "Lista de la compra" screen showing the newly created list with the current date.
- 4 con un clic largo sobre la compra que queremos modificar, se abre el diálogo** (With a long click on the purchase we want to modify, the dialog opens) A blue callout points to a screenshot of a receipt with a long press on the date, which opens a dialog for changing the ID.

The screenshots show the user interface of the Mercadona app, including the receipt details, the list creation screen, the list view, and the receipt modification dialog.



3.7.4 Almacén

3.7.4.1 Agregar un producto

RL

Se puede escanear el código de barras y, si no está registrado, se abre la pantalla de edición con él.

Marca del producto.
Es un campo autocompletado

Unidad y cantidad
Unidades que componen el producto y cantidad de producto.
Una docena de huevos es una unidad con la cantidad 12

Etiquetas para identificar el producto.
La denominación del producto forma etiquetas.

Aceite oliva virgen extra
Carbonell

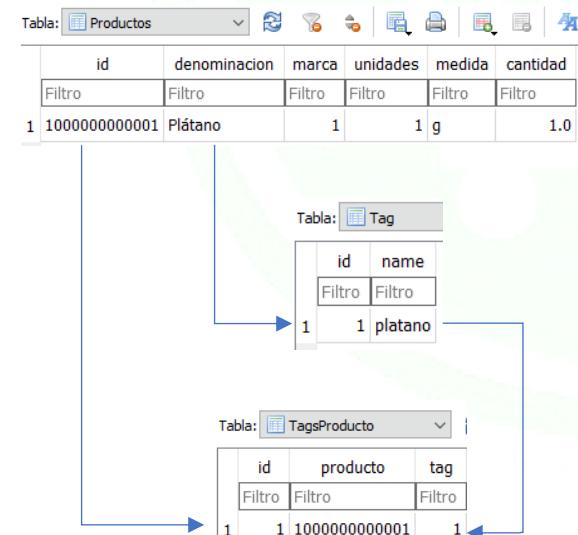
3.7.4.1.1 Insertar un producto a granel

...

Insertamos un producto *plátano* para que se pueda comprar a granel. Todos los campos están en blanco, incluidos los de unidades y cantidad. La etiqueta no aparece porque el campo *denominación* no pierde el foco, así que será en el momento de guardarse el producto cuando se genere.

Las unidades y la cantidad se deben guardar como 1, ya que lo normal será calcular el precio de 1k/l/m.

La
etiqueta es *normalizada* para eliminar
tildes, mayúsculas espacios en blanco.



Relación entre el producto y la etiqueta



Producto insertado a granel

3.7.4.1.2 Insertar otros datos

Si queremos insertar un producto de una marca determinada, podemos insertar, o no, el código barras. El código de barras puede modificarse posteriormente.

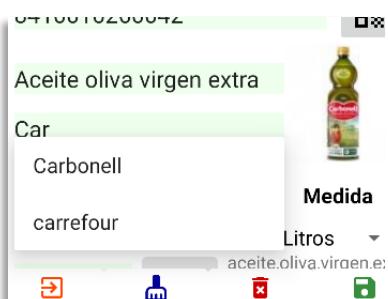
Para insertar el código de barras podemos usar cámara del dispositivo para capturarlo, pulsando el icono de la cámara



Las etiquetas se generan de forma automática cuando el campo de la denominación

Edición de un producto

La marca del producto no necesita estar guardada previamente. Podemos escribirla en su recuadro de manera que éste nos proporciona opciones a medida que escribimos.

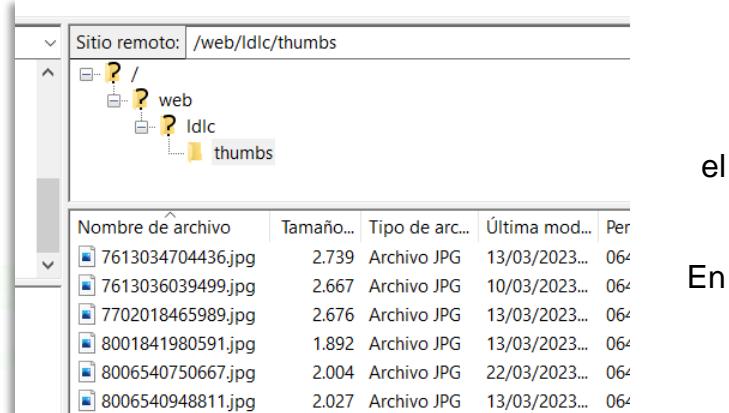


Autocompletado de la marca

Es posible guardar etiquetas, diferentes a las creadas de forma automática, escribiéndolas en el recuadro *tags*, el cual se activa al recibir un texto.

La imagen del producto hay que guardarla manualmente y subirla a algún tipo de alojamiento en red. No se guardan en dispositivo por motivos de espacio.

el archivo Config.java existe una constante estática PATH_PRODUCTS_THUMB que guarda el directorio remoto en el que se guardan las imágenes. Las imágenes deben nombrarse con el id (código de barras) del producto y llevar la extensión jpg.



Directorio remoto para guardar las imágenes

3.7.4.1.3 Proceso de guardado

→

- Se busca un código de barras en caso de que lo hayamos dejado en blanco. El proceso se lo pedimos al controlador `ProductoController` que, a su vez, llama a

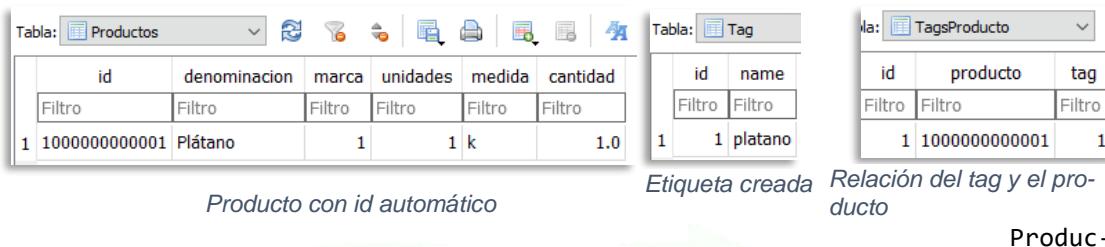


Tabla: Productos					
id	denominacion	marca	unidades	medida	cantidad
Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	1000000000001	Plátano	1	1	k
					1.0

Producto con id automático

Tabla: Tag	
id	name
Filtro	Filtro
1	platano

Etiqueta creada

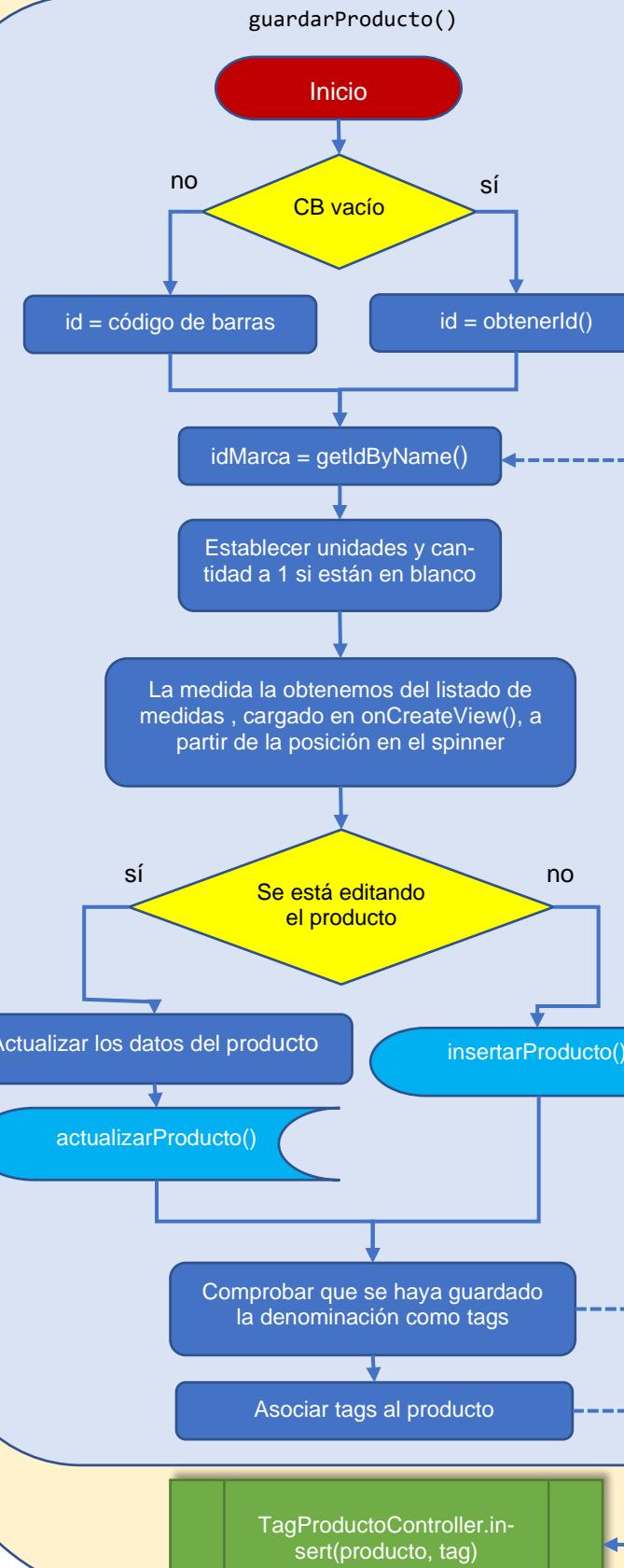
Tabla: TagsProducto		
id	producto	tag
Filtro	Filtro	Filtro
1	1000000000001	1

Relación del tag y el producto

toRepository para devolver las consultas que se ejecutan en `ProductoDao`.

- Se crea un nuevo `ProductoEntity` con los valores recogidos en el formulario.
 - Marca en blanco: se guarda el valor 1, ya que *room database* comienza los autoincrement en 1.
 - Cantidad: si están en blanco, se guarda 1.
 - Unidades: 1 si está en blanco.
 - Medida: el listado de medidas está precargado, por lo que se guarda el que corresponde.
- Las etiquetas se asocian una vez guardado el producto.
 - Es posible que al introducir un producto sin nada más que la denominación, el campo de etiquetas quede vacío, lo que dejaría el producto inco-
 - nexo.
 - Para evitarlo, antes de asociar la etiqueta, en el mismo método de guardarProducto() se llama a `addTagDesdeDenominación()`, que es un método destinado a guardar las etiquetas cuando el campo `denominacion` pierde el foco.
 - El método que asocia etiquetas a productos es `asociarTagsAlProducto(tag)`

activities/almacen/DetalleProductoFragment.class



ProductoController.getIdAutomático()

MarcaController.getIdByName(marca)

ProductoController.insertProducto()

addTagDesdeDenominación()

Dividir la denominación en palabras

Recorrer las palabras

Hav palabra

Normalizar()

addTag()

return null

asociarTagsAlProducto()

Separar las palabras de los tags

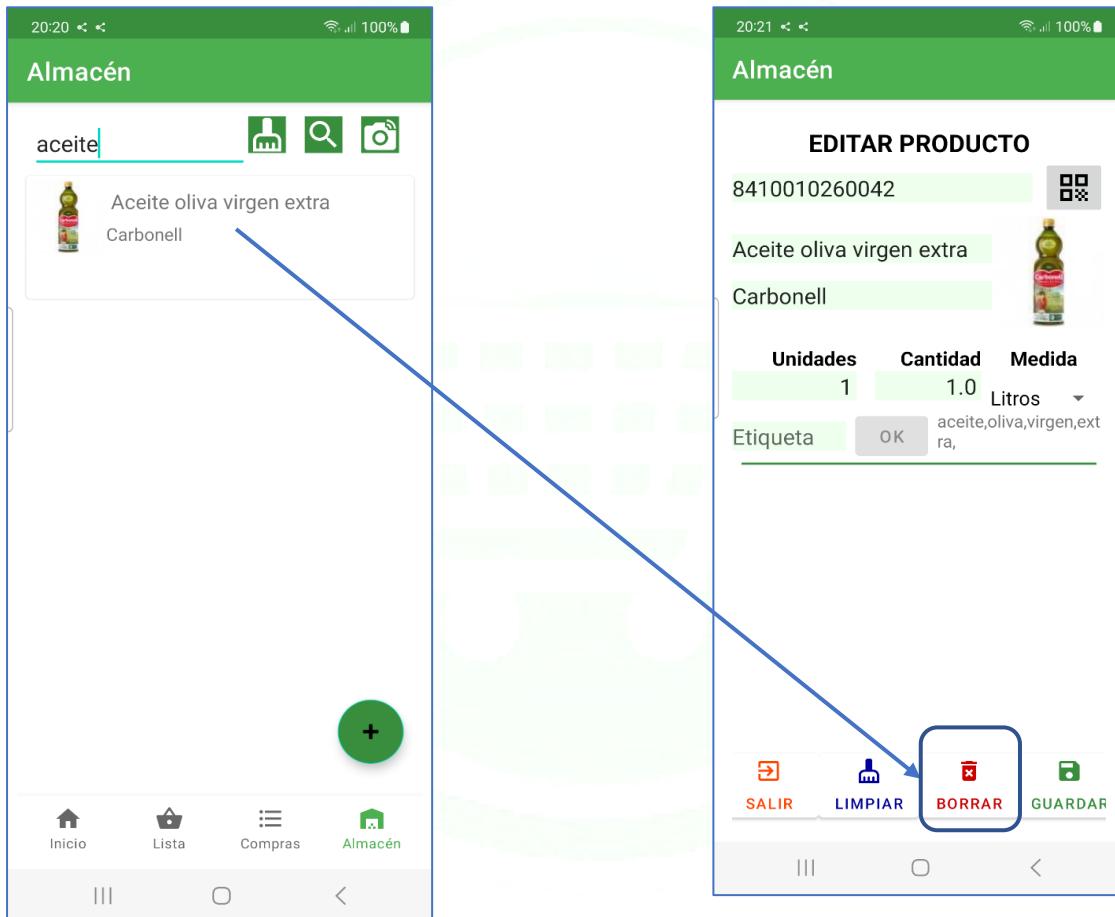
Asociar cada palabra al producto

3.7.4.2 Eliminar un producto del almacén

~

Para eliminar un producto del almacén, hay que seleccionarlo de la lista de productos, buscándolo por la etiqueta, el código de barras, o mostrando todos los productos.

Una vez encontrado, para acceder a la edición, hay que hacer una presión larga (*LongClick*). Un click simple añade el producto a la lista abierta, si la hubiera.



Desde `activities.almacen.DetalleProductoFragment.eliminar()` se llama a `database.repositories.ProductoRepository.deleteById(idProducto)` para que elimine el producto recibido como parámetro.

El id del producto se obtiene del campo de texto del código de barras.

3.7.4.3 Asociar etiquetas a productos

RL

Cada producto puede tener asociadas varias etiquetas que los identifiquen. Estas etiquetas sirven para relacionar o buscar productos. La denominación del producto genera las primeras etiquetas de manera automática, de forma que cada palabra supone una etiqueta.

Table: Productos		
id	denominacion	marca
8410010260042	Aceite oliva virgen extra	20
8480000332493	Aceituna negra sin hueso	9
8480024738455	Aceituna negra sin hueso	3
8437003018008	Agua Mineral Natural Agua...	2

Table: Tag	
id	name
51	aceite
52	oliva
53	virgen
54	extra
55	champu
56	colutorio
57	jabon
58	salabasic

Table: TagsProducto		
id	producto	tag
76	8411384002009	50
77	8411384002009	19
78	7613036039499	9
79	8410010260042	51
80	8410010260042	52
81	8410010260042	53
82	8410010260042	54
83	8437014689501	57
84	5410076230068	55

Tabla de relación entre los productos y las etiquetas

Método que inicia el proceso:

```
activities.almacen.DetalleProductoFragment.asociarTagsAlProducto(idProducto)
```

Llegamos a este método desde `activities.almacen.DetalleProductoFragment.guardarProducto()`, por lo que disponemos de un id del producto.

Lo primero que se hace es extraer el texto del campo de etiquetas y crear un array de String al separar el texto por las comas.

A continuación, recorremos la colección y para cada etiqueta obtenemos su id a través de `controllers.TagController.getIdByName(name)` (las etiquetas son guardadas en el mismo instante en que se crean, desde el método `addTag(tag)`).

Una vez que tenemos el id de la etiqueta, le pedimos a `controllers.TagProductoController.insert(etiqueta, idProducto)` que guarde el registro.

3.7.5 Envío de la lista por mensajería



Clase: dam/proyecto/activities/lista/ListaListaFragment.java

Vista: res/layout/fragment_lista_list.xml

La aplicación puede enviar la lista abierta a contactos que tengamos en el dispositivo mediante el uso de aplicaciones de terceros. Para versión de la aplicación se utiliza *WhatsApp*.

La acción se desencadena al presionar sobre ícono fla_img_share , que invoca al método compartir().

En este método se crea un *Intent* en el que se configuran algunos parámetros y se pasa su ejecución al paquete *com.whatsapp*, el cual se encarga de la ejecución: elección del contacto y envío.

En el intent que llama a *WhatsApp* se añade un extra con el mensaje compartido. Este mensaje se configura en *getListaParaCompartir()*.

En *getListaParaCompartir()* se crea un *StringBuilder* para contener el mensaje que se va a mostrar. También se crea un float para acumular el total.



esta
el

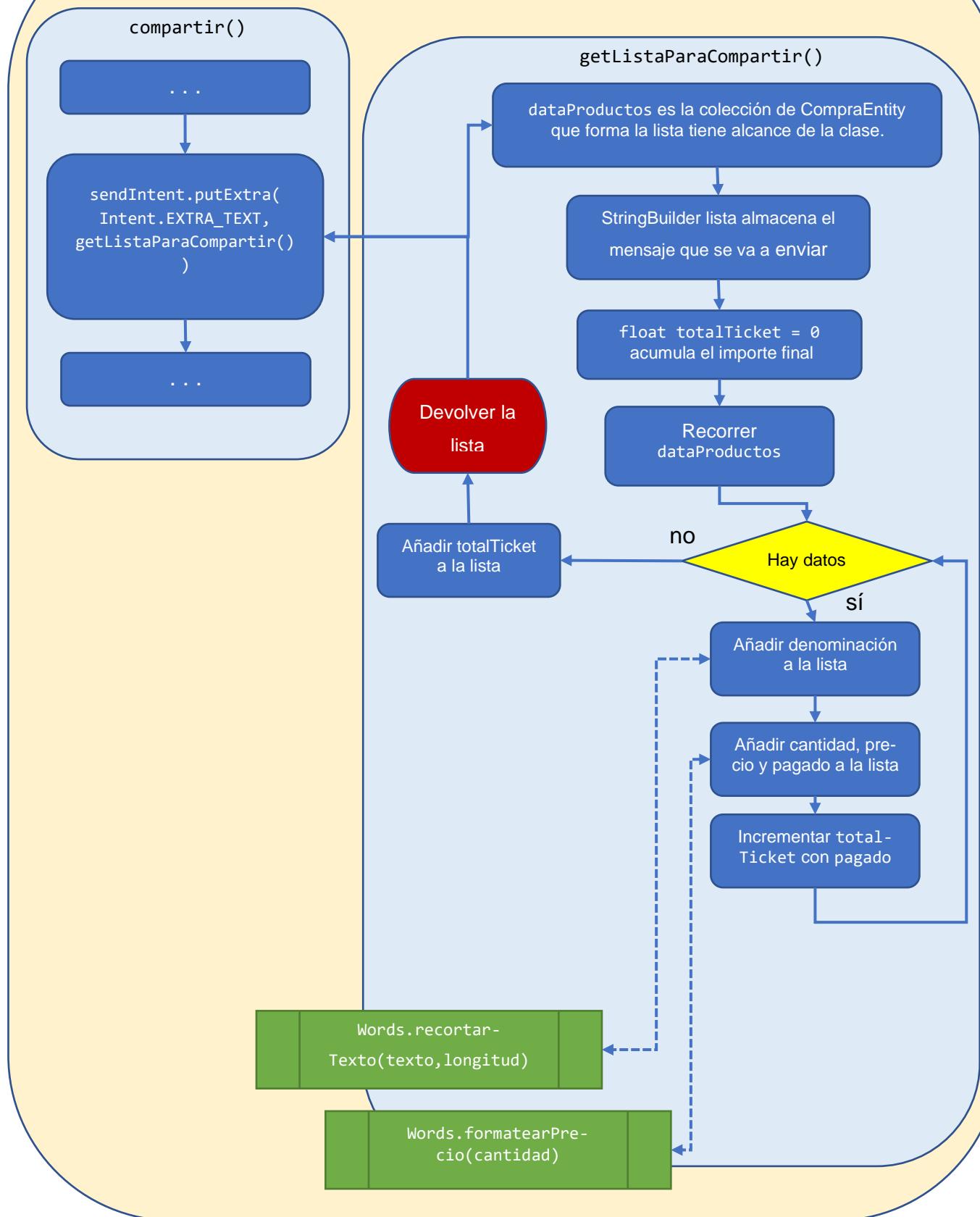
La botonera incluye el botón para el envío de la lista por WhatsApp.

```
public void compartir() {
    Intent sendIntent = new Intent();
    sendIntent.setAction(Intent.ACTION_SEND);
    sendIntent.putExtra(Intent.EXTRA_TEXT, getListadoParaCompartir());
    sendIntent.setType("text/plain");
    sendIntent.setPackage("com.whatsapp");

    try{
        startActivity(sendIntent);
    }catch ( android.content.ActivityNotFoundException ex){
        ex.printStackTrace();
        Toast.makeText(context, text: "No se ha encontrado WhatsApp", Toast.LENGTH_SHORT).show();
    }
}
```

Código del método Compartir()

ListaListaFragment.class



3.7.6 Utilidades



Se ha creado el paquete *utilities* en cual se guardan clases auxiliares.

Fecha.class

La fecha es un dato fundamental en la identificación de las compras. Es la clave primaria de la entidad `NombreCompraEntity` y forma parte de la clave primaria, junto al identificador del producto, de `CompraEntity`.

Lista de la compra	
2304111824 o escribe un nom	
2304111437	
martes	
11 abr 2023 14:37	mercadona

Formato de fecha como código y con año completo

El formato usado como clave es `aammddhhmm`, todo en números.

Entre las funciones más importantes están:

`getNuevaFecha()` que devuelve la fecha del instante en formato `yyMMddhhmm` y se usa para obtener los identificadores de la compra.

`getNuevaFecha(String id)` es una sobre carga del método anterior. En este caso se recibe una fecha `yyMMddhhmm` y se devuelve incrementada en 1 minuto.

En las versiones actuales no se utiliza, pues si se crea un identificador que ya exista, se pide que se vuelva a intentar pasado un minuto.

`getFecha(String fecha)` recibe la fecha en formato de identificador y la devuelve con formato `dd-mm-aaaa hh:mm`

`getFechaFormateada(String fecha)` recibe la fecha en formato identificador y devuelve `dd – mes – aaaa`, donde mes está completo.

`getFechaFormateada()` admite un segundo parámetro de tipo boolean que indica (true) si queremos que la fecha se devuelva en dos dígitos.

Preferencias.class

La aplicación guarda información en la *shared preferences*, como puede ser el id de la lista que se está editando, ya que se debe poder acceder a ella desde cualquier actividad.

Hasta el momento de escribir esta parte de la documentación, tan solo hay dos datos almacenados: el identificado de la compra y una variable de control para asegurarnos de no cargar datos de ejemplo si y han sido cargados.

La ventaja del uso de esta clase es la simplificación de la escritura y la lectura de las preferencias, ya que se han creado sendos setter y getter.

Existe, también, un método para borrar todas las preferencias.

Words.class

La clase *Words* provee de métodos para modificar datos de tipo String.

El método `capitalizar(String texto)` devuelve el texto con la primera letra en mayúscula. Este método es muy importante a la hora de guardar un nuevo producto, ya que el ordenamiento alfabético de *room* distingue mayúsculas y minúsculas.

`destildar(String palabra)` devuelve la palabra sin caracteres especiales, exceptuando algunos de uso común como la *ñ*. Su uso principal se da a la hora de guardar las etiquetas que identifican a los productos, para que *Atún* sea lo mismo que *atún*, *Atun* o *atun*.

`normalizar(String palabra)` destilda, convierte a minúsculas y limpia de espacio en blanco al principio y al final de la palabra. Es el método llamado al guardar una etiqueta.

```
private void addTag() {
    // Texto que se va a guardar
    String tag = tv_etiqueta.getText().toString().trim();
    String normal = Words.normalizar(tag);
    addTag(normal);
}
```

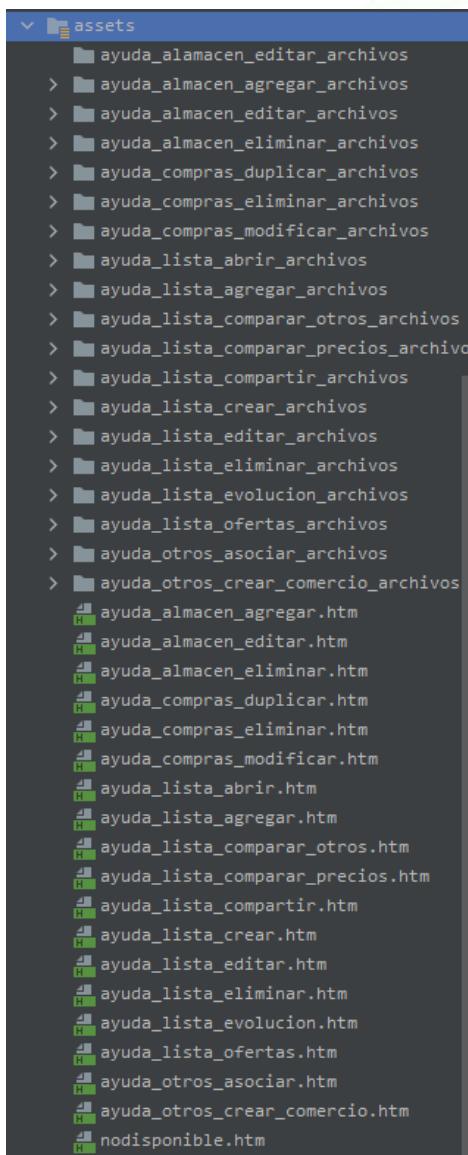
DetalleProductoFragment.addTag() normaliza la etiqueta antes de añadirla.



3.7.7 Sistema de ayuda

El sistema de ayuda está formado por un conjunto de archivos html, uno para cada tema de la ayuda.

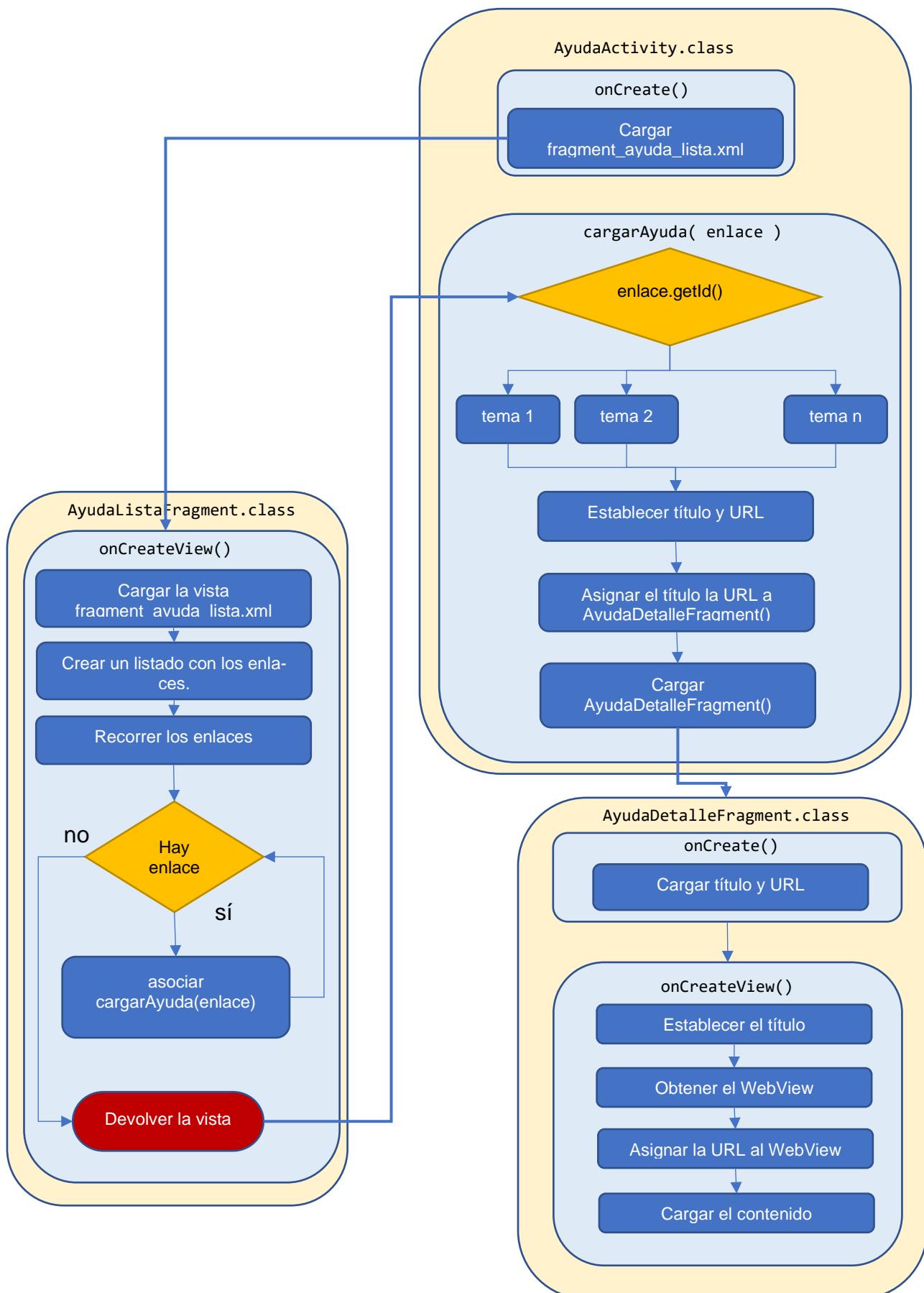
La actividad encargada de su gestión es `dam/proyecto/activities/ayuda/AyudaActivity.class`. Esta clase carga, en el layout `res/layout/activity_ayuda.xml`, un *Fragment-Container*, en el que cargar los fragments, y el *BottomNavigationView* en que visualizar el menú de navegación entre actividades. Además, se carga un *FloatingActionButton* para permitir volver al listado de temas de ayuda desde cualquier vista del detalle.



Sistema de ficheros de la ayuda

`AyudaActivity` carga por defecto `fragment_ayuda_lista.xml` para mostrar el listado de temas de ayuda. `AyudaListaFragment.class` añade el listener para ejecutar `cargarAyuda()`, que se ha implementado en `AyudaActivity`, a cada uno de los enlaces al contenido.

En `AyudaActivity.cargarAyuda(View view)` se establece un título una URL para cada view. Finalmente, se carga el fragment `AyudaDetalleFragment()` al que se le han añadido, como argumento, el título y la URL previamente establecidos.



3.7.7.1 Video-tutoriales

A última hora, y debido a la gran demanda de los usuarios (dos de los dos usuarios así lo han pedido), he incluido alguna ayuda en formato de vídeo. El motivo principal ha sido probar el componente *VideoView*. El principio de este componente es exactamente el mismo que el del *WebView*: se carga una URL y se establece el componente que lo ejecuta, en este caso es un *MediaController*.

En cuanto al *Layout*, se ha duplicado *fragment_ayuda_detalle.xml*, renombrándolo como *ayuda_fragment_video.xml* y cambiando el componente *WebView* por uno *VideoWeb*.

La forma para diferenciar qué elemento de la ayuda es un video o una web es aprovechar el título del tema, que en el caso del vídeo lleva el sufijo “(Vídeo)”.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getArguments() != null) {
        file = getArguments().getString("file");
        title = getArguments().getString("title");
    } else {
        file = "blanco";
        title = "";
    }
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {

    int lauyout = (title.contains("(Vídeo))") ?
        R.layout.fragment_ayuda_video : R.layout.fragment_ayuda_detalle;

    View view = inflater.inflate(lauyout, container, false);

    TextView titulo = view.findViewById(R.id.ayuda_detalle_titulo);
    titulo.setText(title);

    if (lauyout == R.layout.fragment_ayuda_video) {
        Uri uri = Uri.parse("http://robertorodriguez.net/ldlc/videotutoriales/" +
file + ".mp4");
        VideoView contenido = (VideoView) view.findViewById(R.id.ayuda_detalle_contenido);
        contenido.setMediaController(new MediaController(getContext()));
        contenido.setVideoURI(uri);
        contenido.requestFocus();
        contenido.start();
    }
}
```

```
        } else {
            WebView contenido = (WebView) view.findViewById(R.id.ayuda_detalle_conte-
nido);
            contenido.loadUrl("file:///android_asset/" + file + ".htm");
            contenido.setWebViewClient(new WebViewClient());
        }

        return view;
    }
}
```

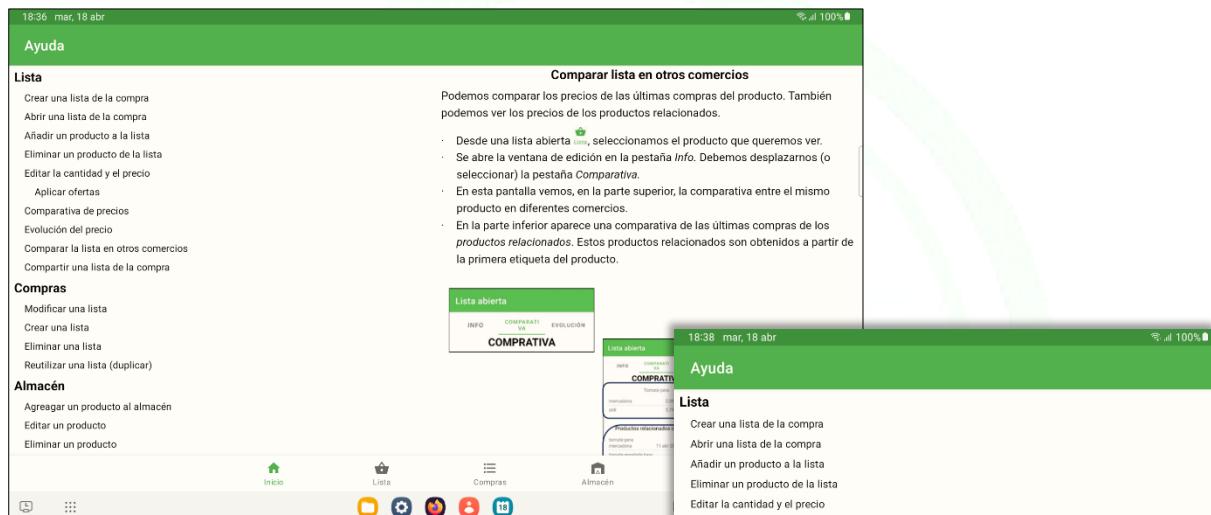


3.8 Diseño multi-Screen

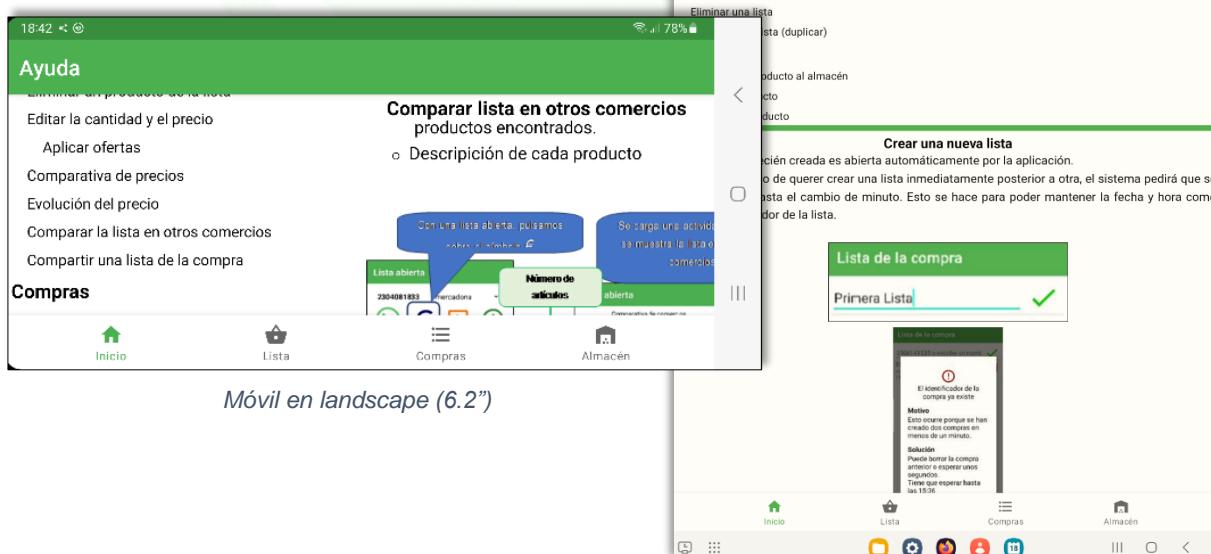


A pesar de que la aplicación se usa en modo *portrait* (vertical), he querido desarrollar alguna parte en multi-screen, de modo que se pueda comprobar la modificación de las vistas de acuerdo con el tipo u orientación de pantalla.

Se ha seleccionado la Actividad “Ayuda” porque su estructura se presta a ello y, en la vista *landscape* de un dispositivo pequeño, no se pierde información.



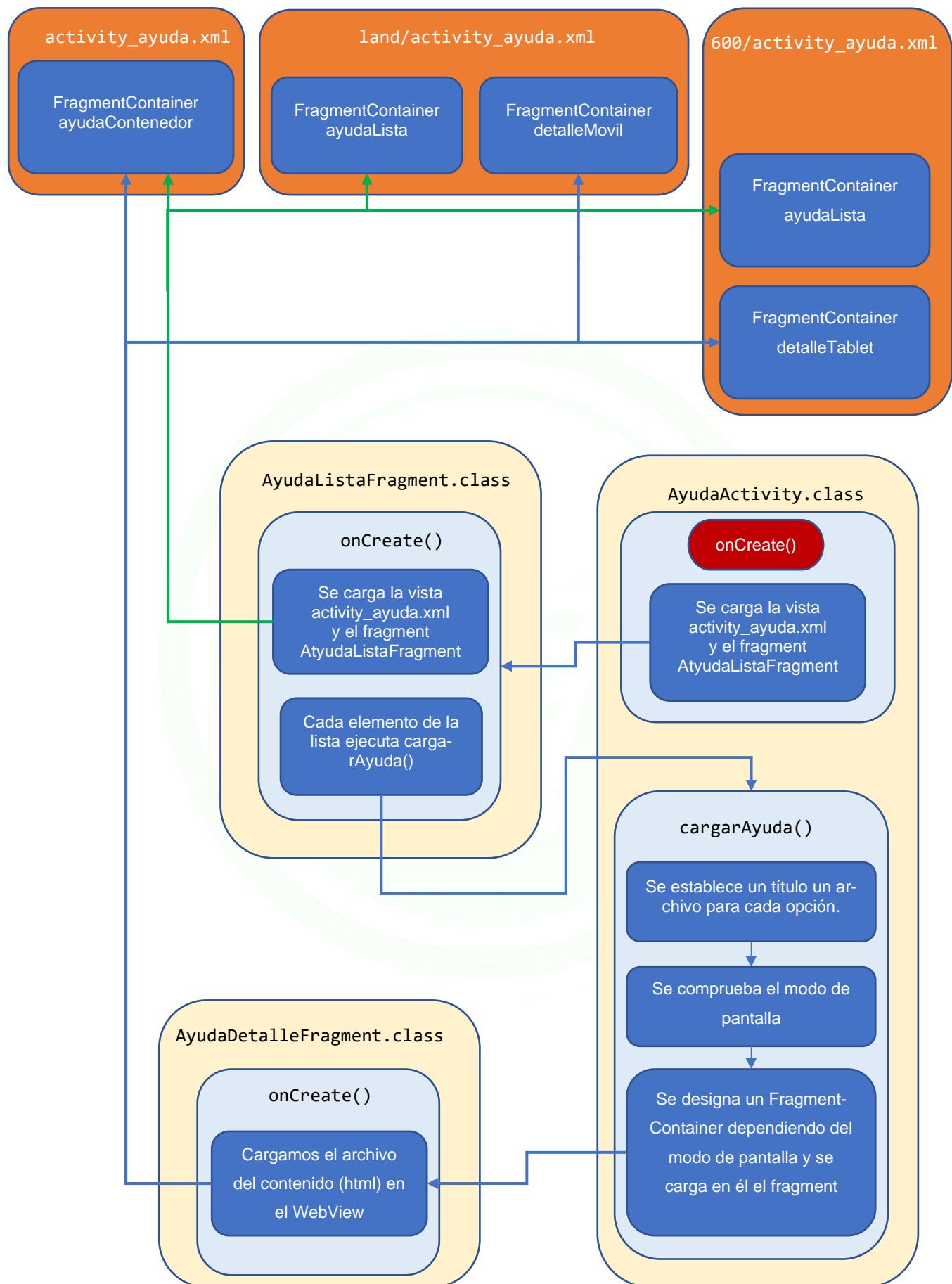
Tablet en landscape (10.5")



Móvil en landscape (6.2")



Tablet en portrait (10.5")



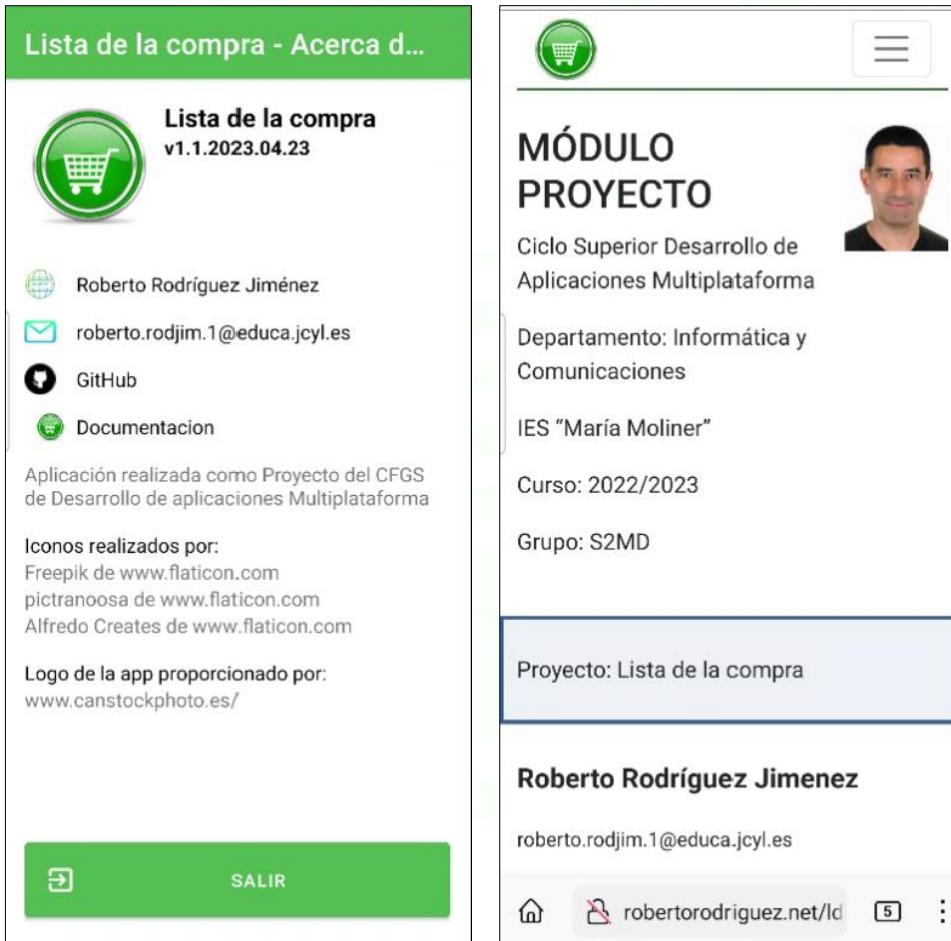
4 Manuales de usuario



4.1 Documentación

Se ha introducido la documentación del proyecto en el mismo proyecto a través de un enlace en la sección “Acerca de”. El enlace abre el navegador del dispositivo y carga la web con la documentación alojada en el servidor, bajo el dominio *robertorodriguez.net/ldlc*.

Página de documentación: <https://robertorodriguez.net/ldlc>



The screenshot displays two side-by-side web pages. The left page is titled "Lista de la compra - Acerca d..." and contains the following information:

- Lista de la compra** v1.1.2023.04.23
- Iconos realizados por:
 - Freepik de www.flaticon.com
 - pictranosa de www.flaticon.com
 - Alfredo Creates de www.flaticon.com
- Logo de la app proporcionado por:
 - www.canstockphoto.es/

The right page is titled "MÓDULO PROYECTO" and contains the following information:

- Ciclo Superior Desarrollo de Aplicaciones Multiplataforma
- Departamento: Informática y Comunicaciones
- IES “María Moliner”
- Curso: 2022/2023
- Grupo: S2MD

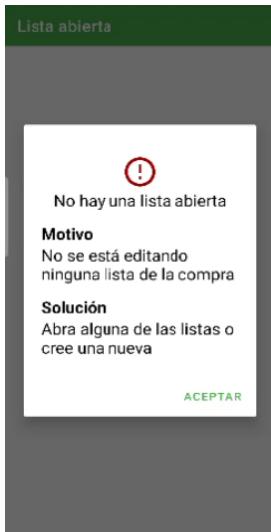
Both pages include a "SALIR" button at the bottom.

Documentación vía web.

4.2 Manual de usuario

4.2.1 Inicio de la aplicación

Cuando la aplicación se inicia por primera vez, no hay dato alguno guardado. La primera pantalla que aparece es la de inicio.



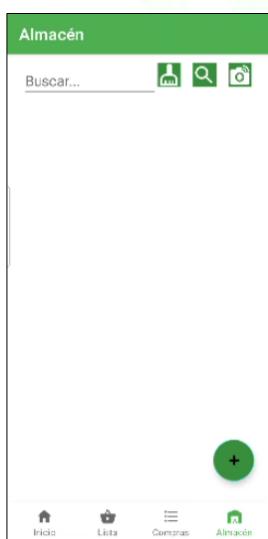
Pantalla de Lista, sin lista abierta

Si navegamos hasta la *Lista*, aparece un mensaje de que no hay aún una lista abierta y, al aceptar, nos lleva a la sección *Compras*, dónde poder crear la primera lista.



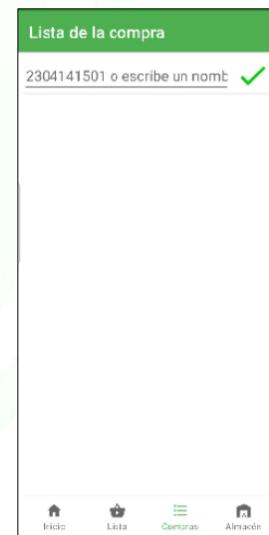
Pantalla de inicio

La sección de *Compras* también aparece vacía, pero, aunque aún no tengamos productos, ya se podría crear una lista.



Pantalla para el almacén.

El *Almacén*, al no tener ningún producto, se muestra completamente en blanco.



Sección de Compra, vacía

4.2.2 Listas

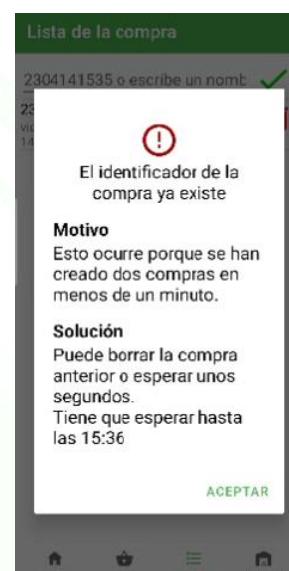
RL

4.2.2.1 Crear una lista

- Ir a la sección **Compras**
- Dar un nombre a lista. Esto es opcional y, en caso de no hacerlo, se nombrará con la fecha y hora del momento.
- Pulsamos sobre el botón *check* para que la lista sea creada.
- La lista recién creada es abierta automáticamente por la aplicación.
- En el caso de querer crear una lista inmediatamente posterior a otra, el sistema pedirá que se espere hasta el cambio de minuto. Esto se hace para poder mantener la fecha y hora como identificador de la lista.



Caja de texto y botón para crear una lista.

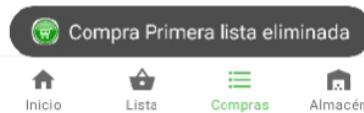


Mensaje de aviso de espera para crear una nueva lista.

4.2.2.2 Borrar una lista

Una lista puede ser borrada, aunque esté siendo editada (*lista abierta*).

- Desde la sección de **Compras** .
- Pulsamos sobre el ícono rojo que representa una papelera .
- Aparecerá un mensaje avisando de que la lista ha sido eliminada.



4.2.2.3 Reutilizar una lista (duplicar)

Si queremos volver a usar una lista ya creada, los precios asociados a la compra de cada producto se modificarán. Esto es útil si se ha creado una lista para usar más tarde, pero los precios han cambiado con respecto a los que tenemos.

Tenemos la opción de duplicar la lista y usarla libremente sin que la original se vea modificada.

- Desde la sección de **Compras** 
- Pulsamos en el ícono de duplicar .
- La lista se duplica y se le asigna el nombre de forma automática (fecha y hora), manteniéndose los mismos productos, con sus precios y la fecha de compra de la compra creada.
- Finalmente, la lista es abierta por la aplicación.

4.2.2.4 Modificar la fecha de una lista

Es posible que queramos agregar una compra realizada hace tiempo, de algún ticket que hayamos encontrado. Al crear la compra, se asigna la fecha y hora actual, lo que coloca la compra en primer lugar del listado. Es posible modificar la fecha de la compra.

- Desde la sección de **Compras** .
- Hacemos una pulsación larga sobre la compra que queremos modificar.
- Se abre un diálogo pidiéndonos la nueva fecha. La escribimos con formato aammddhhmm.
- Pulsamos en **Aceptar**.
- Hay que desplazarse hasta la fecha dada para encontrar, ahora, la lista.



4.2.3 Productos (almacén)

~

4.2.3.1 Agregar un producto

- Accedemos a la sección Almacén
- Pulsamos sobre el ícono de añadir y nos lleva a la pantalla de edición del producto.
- Escribimos la denominación del producto. Es el único campo obligatorio. Si no se escribe el código de barras, el sistema genera uno y podrá cambiarse posteriormente.
- El campo Marca es autocompletado, lo que significa que, si la marca ya ha sido guardada, aparecerá en un listado.
- Aunque no es obligatorio, para que los precios se puedan calcular por su unidad de medida, se recomienda llenar los campos para ello.
- Al introducir la denominación, se generan etiquetas a partir de ella. No obstante, pueden añadirse otras etiquetas.
- Se puede usar la cámara para capturar el código de barras pulsando sobre el ícono que representa un código QR . Se abre la cámara del dispositivo y puede leerse el código de barras.
- Si los datos introducidos son válidos, se activa el botón Guardar.

Pantalla de edición del producto.

4.2.3.2 Buscar un producto

- Accedemos a la sección Almacén
- Buscamos el producto desde la casilla de búsqueda o mediante la captura del código de barras.

4.2.3.3 Editar un producto

- Accedemos a la sección Almacén
- Buscamos el producto.
- Realizamos una pulsación larga sobre el producto que queremos eliminar para editarlo.
- Podemos modificar los campos que queramos, en caso de no ser válidos, el botón de Guardar se desactivará.
- Si no queremos guardar los cambios, simplemente pulsamos sobre Salir .

4.2.3.4 Eliminar un producto

Cuando se elimina un producto, no se elimina la compra, por lo que en los listados guardados siguen apareciendo los productos. Sin embargo, la compra de un producto borrado no puede editarse.

- Accedemos a la sección Almacén
- Editamos el producto.
- Una vez abierta la edición del producto, pulsamos en Borrar .



Intentamos editar un producto borrado.

4.2.4 Lista de la compra

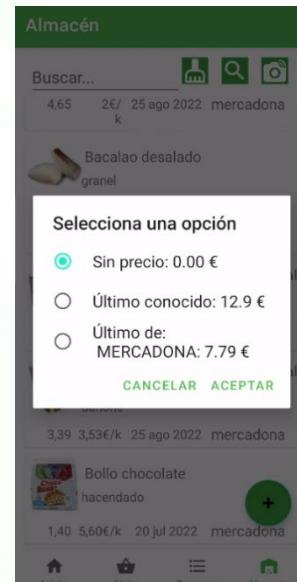
4.2.4.1 Crear una lista de la compra

4.2.4.2 Abrir una lista de la compra

- Acceder a *Compras* .
- Pulsar sobre la compra que queramos abrir.
- La aplicación carga la lista en la sección *Lista*.

4.2.4.3 Añadir un producto a la lista

- Pulsar sobre el botón de añadir  que nos lleva al almacén.
- Desde el almacén buscamos el producto que queremos añadir.
Ver [Buscar producto](#).
- Un click sobre el producto buscado abre un diálogo que nos permite *prefijar* un precio:
 - Ningún precio,
 - Último precio conocido.
 - Último precio en el comercio seleccionado en la lista.
 - Al pulsar *Aceptar* se carga el producto en la lista con el precio seleccionado.



Dialogo para fijar un precio



4 Eliminar producto.

4.2.4.5 Editar la cantidad y el precio del producto

- Un click sobre el producto abre la edición del producto.
- Los campos editables son *precio* y *cantidad*.
- El total y precio por unidad de medida son campos calculados por el sistema.

4.2.4.5.1 Aplicar ofertas

- Para aplicar una oferta, simplemente se pulsa sobre su botón.
- Los precios se calculan, pero no se guardan.



- Para cambiar de una oferta a otra, hay que *salir sin guardar*, ya que los campos de los precios son calculados y, si pedimos dos ofertas consecutivas, los precios son acumulados.
- Los precios finales, tras aplicar la oferta, no son almacenados, aunque la oferta sí es guardada para poder ser vista al editar el producto. Esto es así para no falsear los precios y tener una perspectiva a lo largo del tiempo.

4.2.4.6 Comparativa de precios

Podemos comparar los precios de las últimas compras del producto. También podemos ver los precios de los productos relacionados.

- Desde una lista abierta  *Lista*, seleccionamos el producto que queremos ver.
- Se abre la ventana de edición en la pestaña *Info*. Debemos desplazarnos (o seleccionar) la pestaña *Comparativa*.
- En esta pantalla vemos, en la parte superior, la comparativa entre el mismo producto en diferentes comercios.
- En la parte inferior aparece una comparativa de las últimas compras de los *productos relacionados*. Estos productos relacionados son obtenidos a partir de la primera etiqueta del producto.



Pestañas de la edición del producto

Lista abierta

INFO	COMPARATIVA	EVOLUCIÓN
COMPRATIVA		
Tomate pera		
mercadona	2,39	11 abr 2023
aldi	2,79	13 mar 2023
Productos relacionados con tomate		
tomate pera mercadona	11 abr 2023	2,39 €/k
tomate ensalada lupa lupa	8 abr 2023	2,69 €/k
tomate ensalada lupa	25 mar 2023	inf €/k
tomate ensalada dia	22 mar 2023	inf €/k
tomate carrefour	22 mar 2023	inf €/k
tomate pera mercadona	16 mar 2023	inf €/k
tomate pera aldi	13 mar 2023	2,79 €/k
tomate ensalada mercadona	25 feb 2023	2,39 €/k

Comparativa del mismo producto en diferentes comercios.

Comparativa de productos relacionados con *Tomate* en diferentes comercios.

4.2.4.7 Evolución del precio

- Desde una lista abierta  , seleccionamos el producto que queremos ver.
- Se abre la ventana de edición en la pestaña *Info*. Debemos desplazarnos (o seleccionar) la pestaña *Evolución*.
- Se muestra la gráfica de los precios sin tener en cuenta el comercio.

Lista abierta

INFO	COMPARATIVA	EVOLUCIÓN
------	-------------	-----------

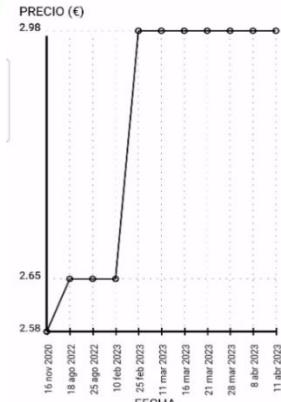
evolución del precio

Vista de la evolución del precio

Lista abierta

INFO	COMPARATIVA	EVOLUCIÓN
------	-------------	-----------

evolución del precio



Evolución de los precios

4.2.4.8 Compartir una lista de la compra

- Abrimos la lista de la compra que queremos compartir en la sección *Lista* .
- Pulsamos el ícono de WhatsApp .
- Ahora es WhatsApp la que se encarga del envío



4.2.4.9 Comparar la lista en otros comercios

- Abrimos la lista de la compra que queremos comparar en la sección *Lista* .
- Pulsamos el ícono  para acceder a la comparativa.
- En la pantalla aparece un comercio, para ver el resto, hay que deslizar la pantalla a un lado o a otro.
- La pantalla muestra:
 - Número de artículo de la lista original que aparecen en la lista.
 - Importe total los productos que aparecen en la lista.
 - Rango de fechas entre la que están los productos encontrados.
 - Descripción de cada producto.

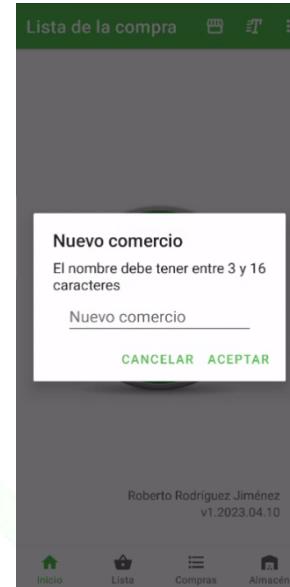
4.2.5 Crear un nuevo comercio



- Desde Inicio **Inicio** accedemos al *ToolBar*, en la parte superior de la pantalla, y seleccionamos el icono para agregar un nuevo comercio.
- En la ventana que se abre escribimos el nombre del comercio.
- En caso de que el comercio ya exista, se muestra un mensaje.



Mensaje avisando de que le comercio ya existe.



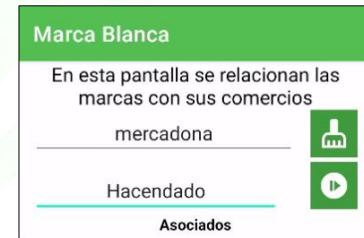
Diálogo para insertar un comercio.

- Si el comercio es guardado correctamente, aparecerá en el listado de comercios cuando se abra una lista.

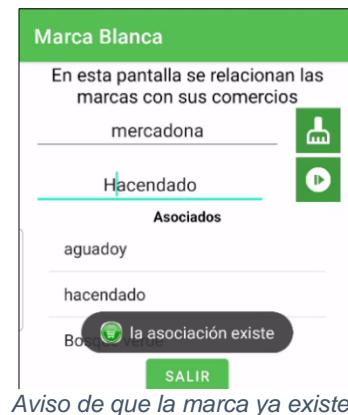
4.2.6 Asociar comercio – marca blanca



- Nos situamos en la página de inicio **Inicio**.
- En el *toolbar* seleccionamos el icono
- Se abre una nueva pantalla en la que escribiremos el comercio y la marca que queremos asociar.
- Una vez ingresados los datos, pulsamos sobre para grabar los datos.
- Se asocia la marca y se muestra un listado de las marcas asociadas al comercio.
- Si la marca ya existiese, se muestra un mensaje.



Formulario para el comercio y la marca

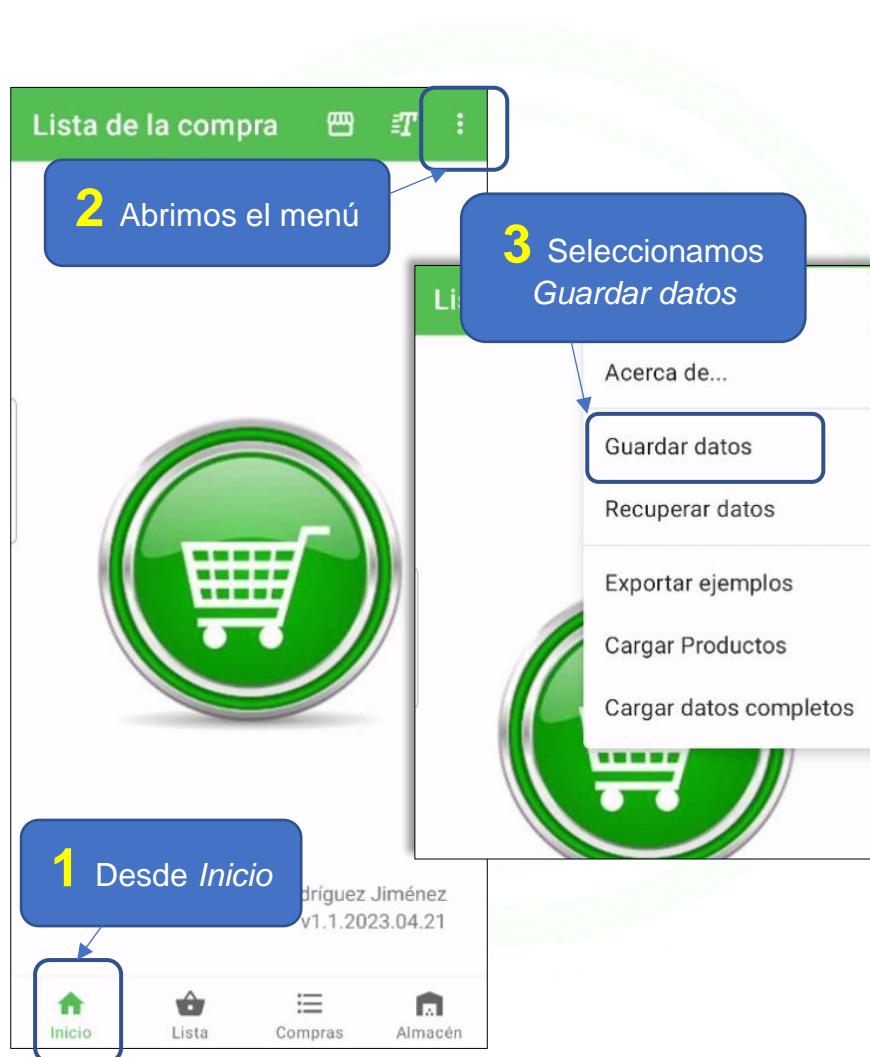


Aviso de que la marca ya existe

4.2.7 Copias de los datos

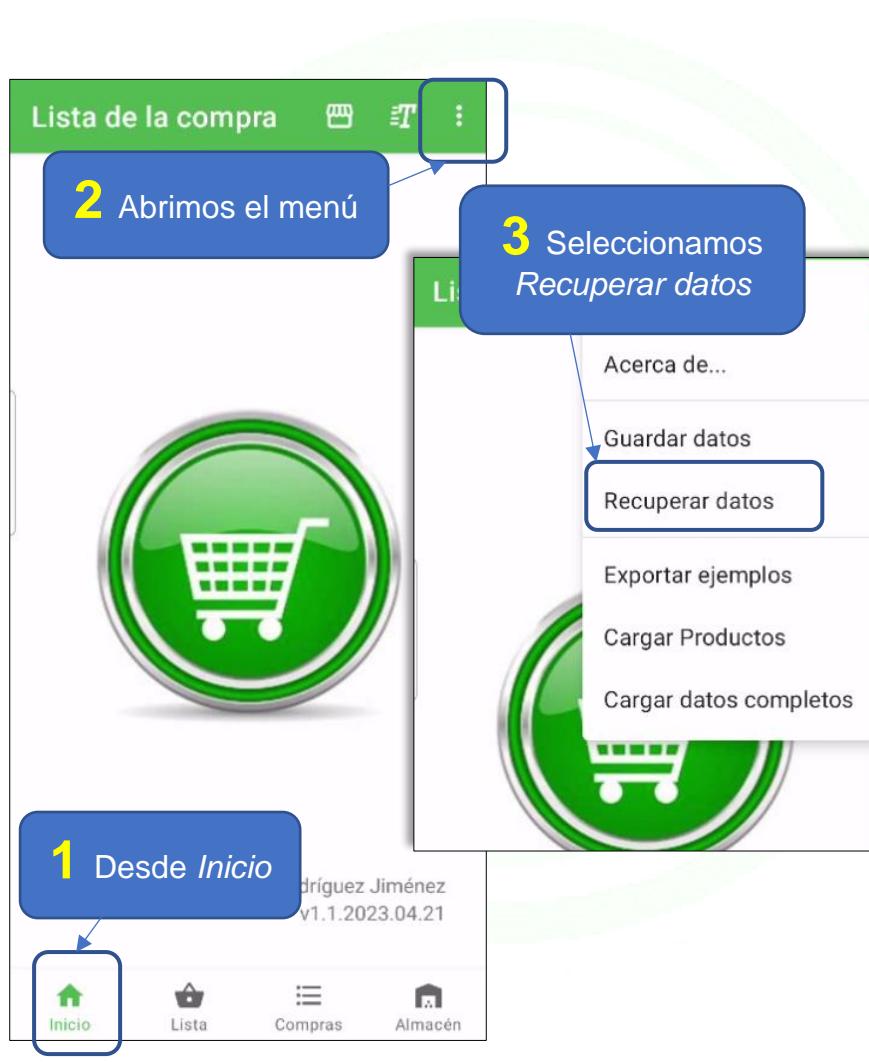
4.2.7.1 Guardar una copia de los datos

Es posible guardar una copia de los datos para poder recuperarla en cualquier momento. La copia queda en el directorio de la aplicación y se elimina si la aplicación es desinstalada.



4.2.7.2 Recuperar una copia de los datos

Para recuperar una copia de los datos, deben existir los archivos creados desde [Guardar una copia de los datos](#). Recuperar los datos guardados supone la eliminación de toda la información de la aplicación, a excepción de los productos. Los productos que no figuren en la copia de los datos perderán la información relacionada, como son la marca y las etiquetas.



4.2.7.3 Importar datos de ejemplo

Los datos de ejemplo son datos reales almacenados en un servidor remoto. Estos datos están pensados para ser cargados al instalarse la aplicación, pero pueden ser cargados en cualquier momento.

PRECAUCIÓN:

La carga de estos datos supone la pérdida de los datos almacenados en el dispositivo, así como los archivos de la copia de seguridad.

Como opción, podemos cargar únicamente los productos. Al cargar los productos, los datos almacenados se mantienen, aunque los productos cargados no tienen datos relacionados, como son la marca o las etiquetas.



4.2.7.4 Exportar datos de ejemplo

Esta opción figura en el menú ya que se debería implementar en futuras ampliaciones.

De momento está restringida al administrador de la aplicación (o sea, a mí 😊). Esto es así para mantener la integridad de los datos y que no se pierda información. Si se permitiese a los usuarios exportar sus datos a la base de datos, no coincidirían las claves de muchos de sus datos, por lo que la aplicación no sería eficiente.

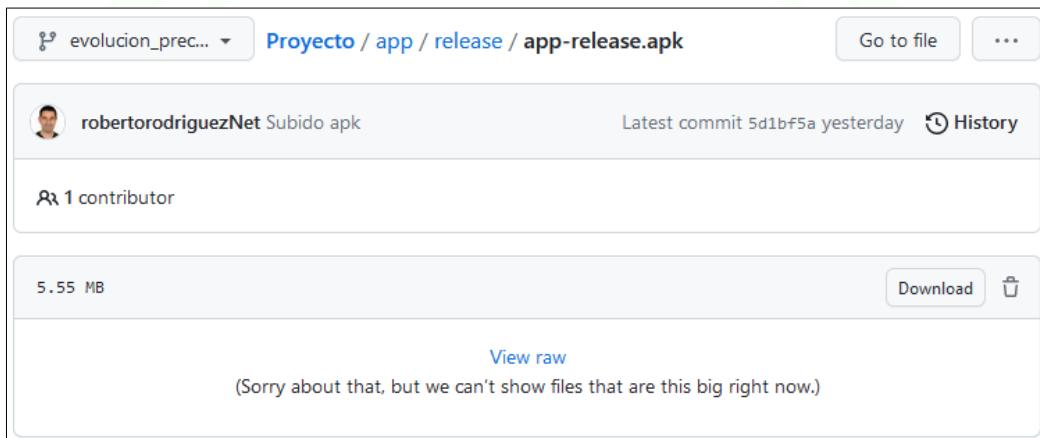
Se deja para una ampliación la opción de que los usuarios puedan colaborar con una base de datos general, pero eso implica un nuevo estudio de la estructuración de los datos.

4.3 Manual de instalación

El archivo instalable es un archivo comprimido en formato APK.

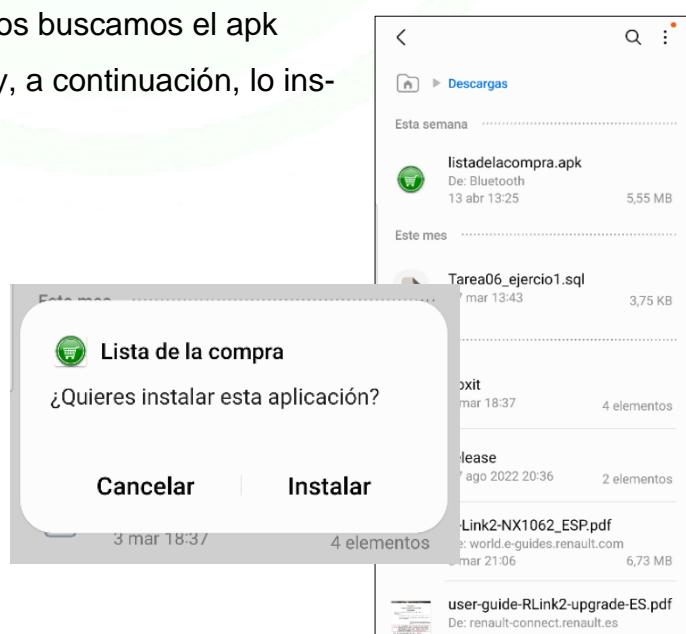
Pasos:

1. Obtención del paquete APK.
2. El archivo APK puede ser transferido al dispositivo de varias formas: descargado del repositorio, mediante el explorador de archivos del ordenador, por Bluetooth... Para poder acceder a la memoria del dispositivo desde el ordenador, es posible que se pida que se instalen los drivers del dispositivo.



Ventana de descarga del APK desde el repositorio en [GitHub](#)

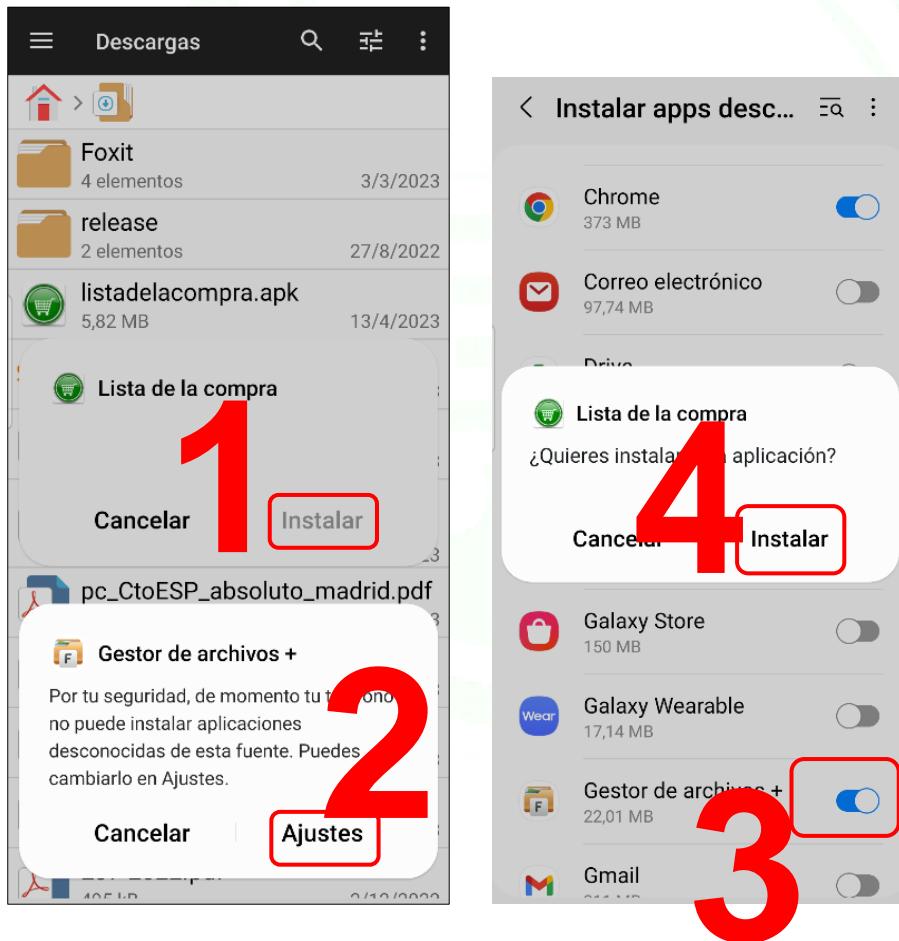
3. Desde un explorador de archivos buscamos el apk (normalmente en *Descargas*) y, a continuación, lo instalamos.



4. Si se instala desde un explorador de archivos que aún no tiene permisos, debemos dárselos. Lo mismo ocurre si la instalación se hace desde Bluetooth.

Al pulsar sobre *Instalar* (1) se abre la ventana para pedir permiso para instalar aplicaciones desconocidas. Vamos a *Ajustes* (2) y marcamos la opción que corresponde al explorador al que le queremos dar el permiso (3) y finalmente instalamos (4).

Tras las instalación, podemos revocar nuevamente el permiso para que no permita más instalaciones.



5 Conclusiones y posibles ampliaciones



El proyecto cumple con su función de una forma muy básica, no abarca todo lo que se esperaba, pero sí los objetivos iniciales.

La falta de experiencia ha hecho que el desarrollo avanzase en base al uso real que se hacía de la aplicación, no en lo especificado, pues han ido apareciendo algunos detalles con lo que no se contaba de antemano. Uno de estos detalles, por ejemplo, es la restricción de las marcas blancas de las cadena a la hora de la elaboración de las listas.

Implementar el control de las marcas blancas supuso la modificación de la base de datos, teniendo que crear nuevas tablas y propiedades en otras. Aunque, en principio, este tipo de contratiempo pudiera suponer un obstáculo para el desarrollo de la aplicación, realmente se aprovecha para el refinamiento del código, probando que el código tiene un bajo acoplamiento y una buena cohesión.

En lo que respecta a la aplicación como tal, específica para Android, me he encontrado con dificultades a la hora de elegir los componentes (*Views*) debido al rápido crecimiento del sistema y a la aparición constante de librerías. Esto ha obligado a que la versión mínima de Android requerida suba de la 4.0 a la 6.0.

Actualizarse a las nuevas versiones, con el trabajo comenzado y con muy poco tiempo para dedicar al estudio de las nuevas características, ha sido otro factor en contra. Como ejemplo de dificultad añadida, debido a los cambios de versión, está el almacenamiento de los archivos de backup de la base de datos en el almacenamiento interno pues, a partir de la versión 11 de Android, no está permitido el almacenamiento fuera del directorio de instalación de la aplicación. Ponerse al día en la nueva API supone un retraso considerable, teniendo en cuenta, además que habría que modificar la estructura de los archivos.

La aplicación comenzó a desarrollarse con unos conocimientos muy básico, por lo que el inicio fue muy lento, a veces parecía imposible avanzar ante los retos que iban apareciendo. El aprendizaje ha sido bastante amplio. Lo aprendido sobre Android durante el curso, a parte de demasiado elemental, resultaba algo (muy) obsoleto, lo que

me ha obligado a consultar fuentes externas constantemente, con casi todo el material nuevo en inglés. Esto vuelve a suponer un avance en lugar de un obstáculo.

5.1 Ampliaciones

Se quedan muchas cosas si implementar, no hay tiempo, ni medios, para más.

- *Almacenamiento interno*, fuera del directorio de instalación, para los archivos de backup de la aplicación.
A partir de Android 11 no es posible, y es una buena opción para poder compartir fácilmente estos archivos.
- *Compartir las listas entre diferentes usuarios*. Puede crearse una base de datos en un servidor remoto en la cual se almacenen todos los datos. Estos datos se relacionarían con cada usuario, de forma que cada uno de ellos sólo tuviera permiso para acceder a sus propios datos. La ventaja estaría en que un usuario podría dar permiso a otro(s), o a todos , para consultar su lista. Se podría, incluso, dar permiso para un control total.
- *Obtener las imágenes de los productos mediante la cámara del dispositivo*.
Esta opción no parece muy rentable en cuestión de facilidad de uso y de almacenamiento, pero no deja de ser una opción que se puede ofrecer.
- Búsqueda por voz. Estaría bien poder buscar y añadir los productos por voz. Ahora es posible activar el micrófono del teclado del dispositivo, si está configurado, pero lo ideal sería disponer del micrófono en la propia UI.

6 Referencias



Guía de referencia de Android

<https://developer.android.com/>

Generar APK

<https://developer.android.com/studio/run?hl=es-419>

Room Database

<https://developer.android.com/codelabs/android-room-with-a-view#0>

Room: relaciones entre objetos

<https://developer.android.com/training/data-storage/room/relationships#java>

Fragments dinámicos

<https://openwebinars.net/academia/aprende/android/2828/>

Blog sobre desarrollo de aplicaciones en general

<https://www.desarrollolibre.net>

Canvas. Efectos *drawLine*

[Código Eptadex](#)

Manejo de cambios en la configuración de Android Studio

<https://guides.codepath.com/android/Handling-Configuration-Changes>

Descarga de iconos e imágenes

<https://sp.depositphotos.com>

Imagen del logo de la aplicación

<https://www.canstockphoto.es/>

Integrar WhatsApp

https://faq.whatsapp.com/511210730860677/?locale=es_LA&cms_platform=android

Página de descarga de los conectores MariaDB

<https://mariadb.com/kb/en/mariadb-connector-j/>

Página para la descarga del driver JDBC-MySQL

<https://dbschema.com/jdbc-driver/MySql.html>

Sitio web para la descarga de diferentes drivers JDBC

<https://dbschema.com/>

Web MySQL Community

<https://dev.mysql.com/downloads/connector/j/>