ComprApp Documentación

Programación de dispositivos móviles

Autor:

Alejandro Olivares del Rey Pierres

Granada, a 13 de abril 2022

Índice

1. Descripción de la aplicación

Este proyecto tiene como objetivo desarrollar una aplicación que permita llevar un seguimiento de una lista de la compra. Permitiendo escanear productos al terminarlos en casa y antes de tirarlos a la basura. Además, puede llevarse un conteo de los objetos que se han echado al carrito de la compra. Permitiendo así mostrar el precio total de la compra.

La aplicación mostrará una pantalla principal con la última compra, otra pantalla con la finalidad de crear nuevas listas y, adicionalmente, una última que tiene un historial de compras pasadas. Esta última permite ver los comestibles de dichas compras.

Además, la aplicación permitirá el uso de dos maneras diferentes: un modo de seguimiento para cuando se estén añadiendo objetos al carrito en el supermercado y otro modo para planificar compras futuras.

Los principales problemas que debemos resolver para el correcto funcionamiento de la aplicación se basan en los módulos que debemos implementar:

1. Lista de comestibles
2. Cámara
3. Escáner
4. Base de datos
5. Historial de compras

2. Requisitos funcionales

En este apartado de la documentación se definirán los diferentes requisitos funcionales que se necesitan para que la aplicación sea completamente funcional.

2.1. Requisitos Generales y de interfaz

RF1.1 Gestionar compra. Se debe poder gestionar una lista de la compra, como requisito básico de la aplicación.

- RF1.1.1 Abrir última compra. Será posible abrir la última compra realizada para seguir añadiendo objetos.
- RF1.1.2 Editar productos de la compra. Se debe permitir gestionar los objetos de la compra, tanto como para añadir uno nuevo como para eliminarlos o duplicarlos.
- RF1.1.3 Obtener precio total de la compra. La aplicación permitirá saber el precio total en el modo de "seguimiento"de la aplicación.
- RF1.1.4 Cambiar modo de seguimiento. Se debe permitir cambiar el seguimiento de la aplicación. Solo existen dos modos: Seguimiento de compra o futura compra

RF1.2 Listar productos. Se debe poder listar los productos de la lista de la compra, de una manera sencilla.

2.2. Requisitos del módulo de cámara

RF2.1 Abrir cámara. La aplicación debe implementar una cámara propia que sea capaz de detectar la información necesaria para guardar el objeto en la lista de la compra.

RF2.2 Detectar información del objeto. Será necesario y fundamental, detectar la información de una manera sencilla y eficiente para el usuario.

- RF2.2.1 Detectar código de barras. La aplicación debe detectar el código de barras para poder buscar la información del objeto, tal como: Nombre del producto e imagen. Es posible que algunos productos no presenten imagen.
- RF2.2.2 Detectar etiqueta de producto. La aplicación debe detectar de manera sencilla el precio de la etiqueta del objeto. En caso de que no sea posible detectarla, se podrá introducir manualmente

RF2.3 Convertir imágenes. La cámara capturará imágenes de tipo ImageProxy, pero la cámara usará imagenes en Bitmap para detectar la información.

RF2.4 Rotar imágenes. La cámara capturará la imagen, pero luego es necesario una transformación de plano, para permitir la detección de la información.

2.3. Requisitos del módulo de escáner

RF3.1 Analizar imágenes tomadas con la cámara. La aplicación debe permitir analizar las imágenes en busca de información del producto escaneado.

RF3.2 Detectar información

- RF3.1.1 Código de barras. La aplicación debe ser capaz de detectar un código de barras de un producto alimenticio.
- RF3.1.2 Texto en las etiquetas. La aplicación debe ser capaz de detectar el precio del producto en la etiqueta del producto.

2.4. Requisitos del módulo de base de datos

RF4.1 Buscar. Se debe poder buscar el código de barras en la API utilizada en este proyecto.

RF4.1.1 Código de barras

RF4.1.2 Imágen. La aplicación debe ser capaz de extraer la imágen de la base de datos del proyecto.

RF4.2 gestionar almacenamiento

RF4.2.1 Guardar compra. Se debe poder guardar la compra para su posterior visualización.

RF4.2.2 Obtener compra. Se debe poder obtener una compra en concreto de la base de datos.

RF4.2.3 Obtener última compra. Se puede disponer únicamente de la última compra realizada.

RF4.2.4 Obtener una lista de compras. Deberá ser posible obtener una lista de todas las compras realizadas.

RF4.2.5 Eliminar archivo. Por cualquier motivo, debe ser posible eliminar una compra.

2.5. Requisitos del módulo de historial

RF5.1 Listar compras. Se debe poder listar todas las compras realizadas con anterioridad.

RF5.2 Obtener el precio total de la compra. Debe ser posible obtener el precio total de la compra.

3. Módulo de interfaz

La interfaz de la aplicación es muy sencilla y básica. En parte debido a que no cuento con la experiencia necesaria todavía para hacer florituras. El diseño elegido es “material design”, con una barra inferior que permite cambiar de tarea en la aplicación. Teniendo: Inicio, Añadir compra e historial.

3.1. Implementación del módulo

La implementación es muy sencilla, se ha utilizado una actividad principal con un contenedor para fragmentos y cada “tarea” es un fragmento. Para la lista de inicio (última compra) y la de añadir elementos, se ha utilizado el mismo RecycleView, ya que se tiene el mismo modelo de datos, un ítem comestible. Para almacenar los datos, se ha utilizado un modelo “PurchaseModel”. Que permite guardar nombre, precio, imagen y código de barras del producto.

3.2. Diagrama UML del módulo Interaz

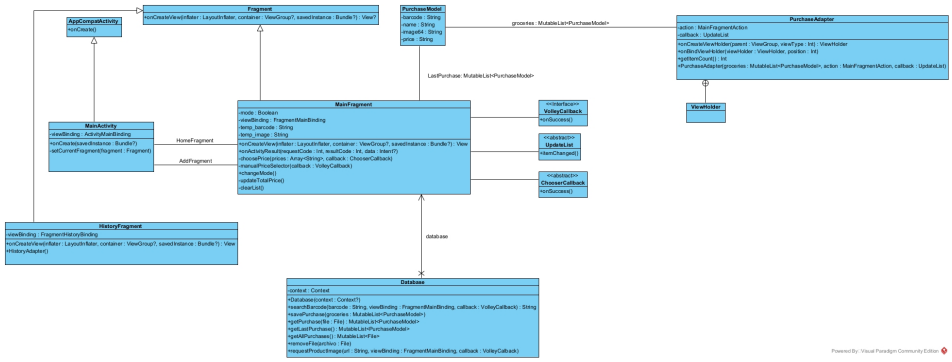

Figura 1: Diagrama UML del módulo de interfaz

4. Módulo de cámara

Se ha creado una actividad que contiene una cámara, permitiendo que sea posible tomar fotos de los códigos de barras y las etiquetas. Esta actividad es muy básica, ya que su funcionalidad principal es la de tomar las imágenes y pasarlas al escáner para sacar la información esencial del producto. Además, es la encargada de enviar la foto al escáner y recoger los datos del mismo. Para continuar con la adición del nuevo producto a la lista.

4.1. Implementación del módulo

Anteriormente se ha explicado una pequeña parte de la funcionalidad de este mismo módulo. Más concretamente, se ha implementado una actividad que inicia una cámara haciendo uso de la librería CameraX. Cuya documentación se encuentra disponible en https://developer.android.com

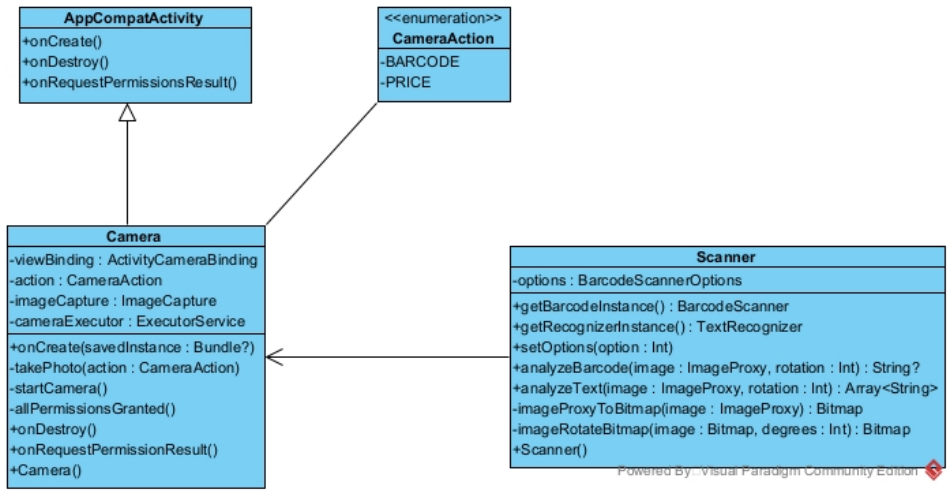

Figura 2: Diagrama UML del módulo de cámara

4.2. Diagrama UML del módulo Cámara

5. Módulo de escáner

Para resolver el problema del escaneo de barras y detectar precios, se ha creado una clase que permite detectar esta información, a través de la librería ofrecida por Google: MLKit. Esta es una clase sin ninguna vista, simplemente se utiliza para detectar información en segundo plano.

5.1. Implementación del módulo

En este módulo debemos resolver el principal problema de la aplicación: Detectar la información de manera dinámica. Para detectar los códigos de barras se ha creado una función que obtiene una imagen de tipo ImageProxy, la convierte en Bitmap y la procesa. Ocurre lo mismo para la lista de precios detectados en la etiqueta. Se ha creado una función, a la que de la misma manera se le pasa una imagen de tipo ImageProxy y se convierte en Bitmap. Por último, se procesa y se devuelven los valores con los posibles precios del producto.

5.2. Diagrama UML del módulo de escáner

Debido a la simplicidad del diagrama UML, donde solo se encuentra la clase Scanner. Ya que no hace uso de otras dependencias en clases de la aplicación, se ha decidido no incluir el diagrama. Pero se animará al lector a comprobar la documentación incluida en el KDoc.

6. Módulo de base de datos

El corazón de la aplicación se basa en este módulo. Ya que necesitamos guardar y recuperar los datos de la lista, de otra manera la aplicación no serviría de mucho. Se ha creado una clase Database que básicamente se trata de un adaptador para las búsquedas en la API utilizada en el proyecto y que gestiona el guardado de listas de compra de la aplicación. Así como, la recuperación de las mismas compras guardadas.

6.1. Implementación del módulo

Esta clase puede que sea la que más funciones tiene, pero no la más complicada. En primer lugar, se han creado diversas funciones que permiten organizar la base de datos. Se han especificado en el apartado de requisitos funcionales de esta misma documentación. Para guardar las compras se ha resuelto el problema utilizando archivos JSON que permiten guardar una lista de objetos comestibles. Kotlin, el lenguaje por defecto de esta aplicación, no tiene una librería incluida para trabajar con JSON. Con lo cual se ha utilizado la librería GSON. Además, se han hecho otra serie de funciones que permiten obtener datos concretos de una compra. Por ejemplo, el precio total de la misma.

6.2. Diagrama UML del módulo

Debido a la simplicidad del diagrama UML, donde solo se encuentra la clase Database. Ya que no hace uso de otras clases ni dependencias, se ha decidido no incluir el diagrama. Pero se animará al lector a comprobar la documentación incluida en el KDoc.

7. Módulo del historial

El último bloque que se documentará será el de historial, que permite listar las compras guardadas en memoria. Además de observar de manera detallada cada compra, para llevar un conteo de los objetos de cada compra.

7.1. Implementación del módulo

La implementación es muy sencilla. El principal problema se ha resuelto en el apartado de bases de datos. Este se basaba en la manera que se guardan los datos. Para ello se ha utilizado una lista de objetos que se guarda con JSON. El fragmento utilizado en el historial contiene una lista de tipo RecycleView con objetos de tipo HistoryModel, es decir, hay una lista que contiene las anteriores compras. Además, si se mantiene pulsado uno de los objetos se lanza una ventana con los detalles de la compra.

7.2. Diagrama UML del módulo Historial

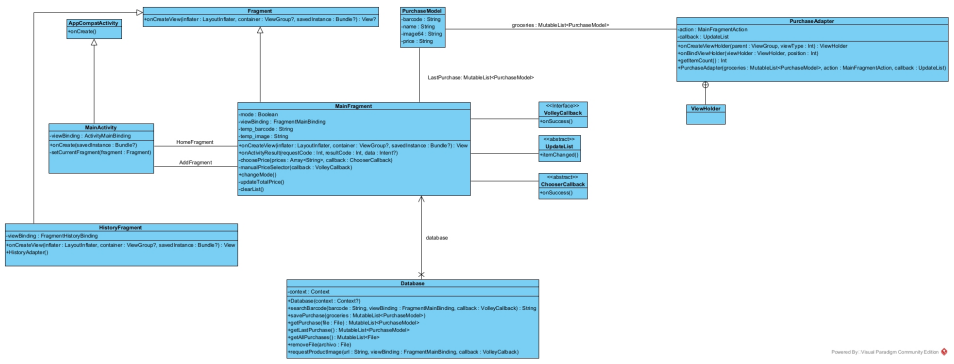

Figura 3: Diagrama UML del módulo de historial

8. Conclusiones

Para concluir la documentación, se quiere mencionar que esta aplicación se ha realizado con una fecha de entrega clara, un mes para ser exactos. Soy consciente de que la aplicación tiene muchas mejoras en base a la interfaz o casos de uso concretos que no haya podido comprobar por falta de tiempo. Además, pienso que tiene mucho potencial porque es algo, que personalmente usaría.