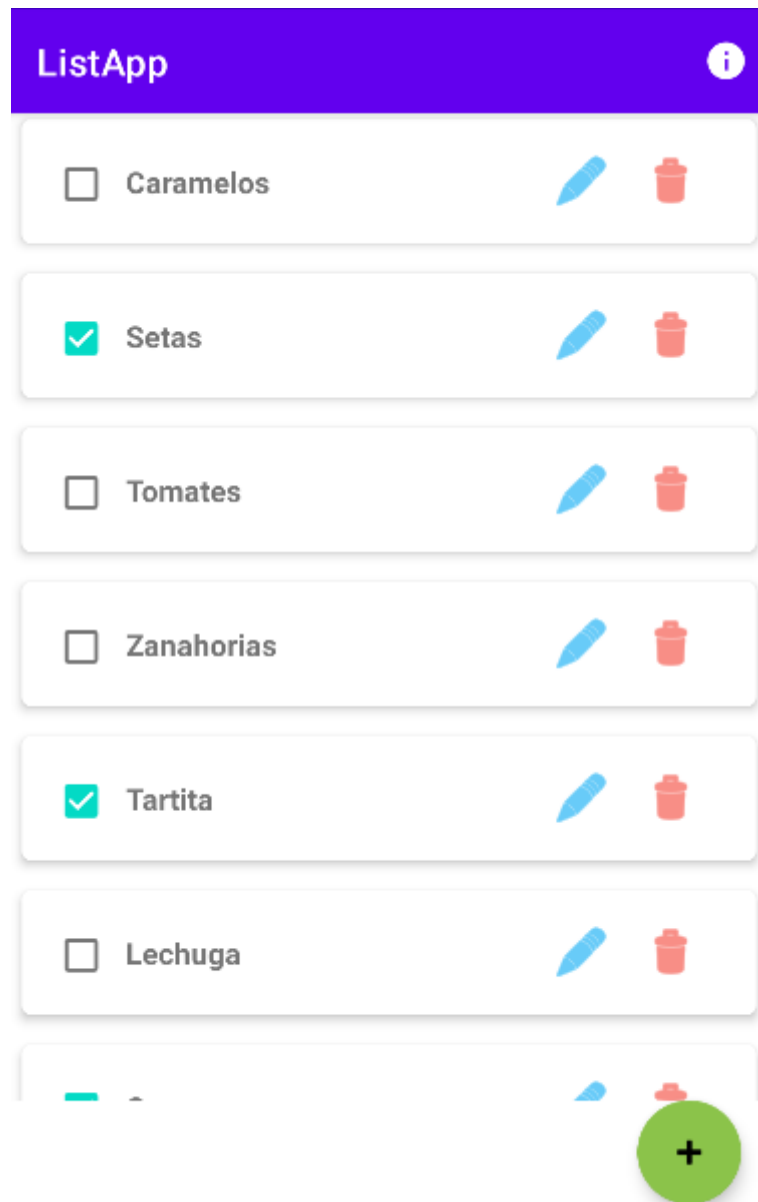


# ListApp



Img. 1 Pantalla principal

## **Índice**

0. Portada	1
1. Funcionamiento	3
2. Justificación del código	9
3. Estructura de la app	10

## Funcionamiento

La aplicación consiste en una pantalla principal que es la Activity principal (“ListActivity” que contiene la navegación entre Fragments y el Fragment inicial “ListFragment”), una segunda pantalla con los detalles del elemento seleccionado (botón Editar) o con los detalles del elemento a crear (botón de Añadir: +) (que no es otra Activity, sino que es un Fragment, aunque visualmente no se aprecia) y una tercera pantalla que contiene información sobre la aplicación, accesible desde cualquiera de las otras dos mediante un menú de opciones.

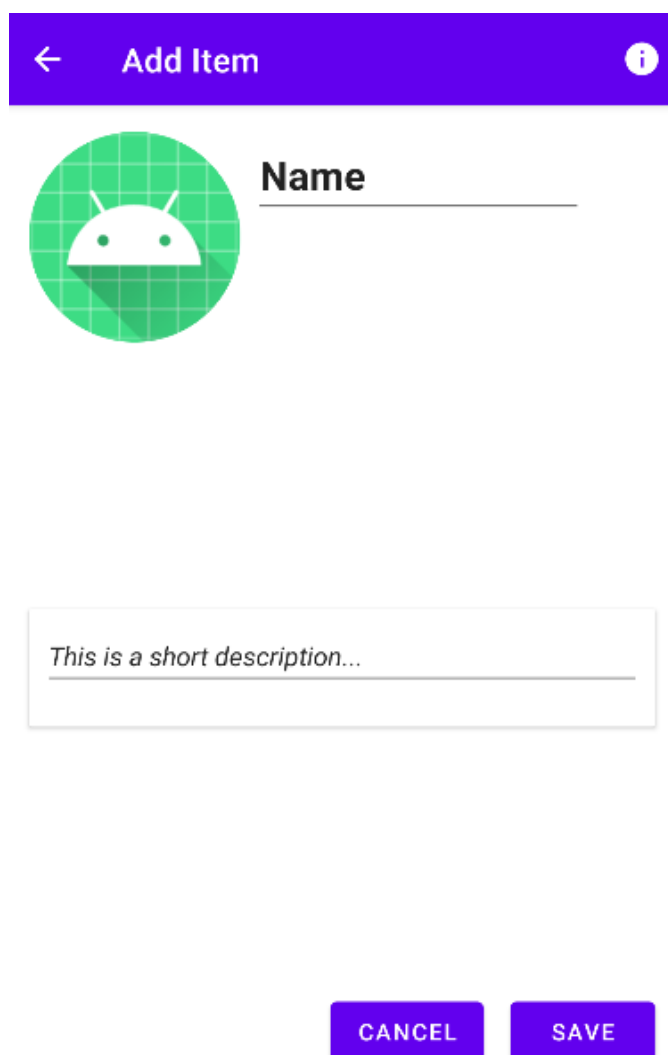
Al abrir la aplicación, lo primero que veremos será una pantalla en blanco y un FAB (Floating Action Button) para añadir elementos, ya que no hay elementos creados en la base de datos Room.



*Img. 2 Pantalla inicial*

Al presionar el FAB se abrirá la segunda pantalla: DetailFragment(un Fragment reutilizado tanto para Crear como para Editar un elemento de la lista. Dentro tendremos una imagen que representa al elemento, un campo de texto con el nombre y una breve descripción del producto en otro campo de texto.

Al final tenemos dos botones (Cancelar y Guardar). Si no queremos crear el elemento, cancelaremos; en caso contrario, al darle a “Save” el elemento se añadirá a la base de datos y se mostrará en la lista de la primera pantalla.



Img. 3 Pantalla Añadir elemento (add item)

## Cancelar y Guardar

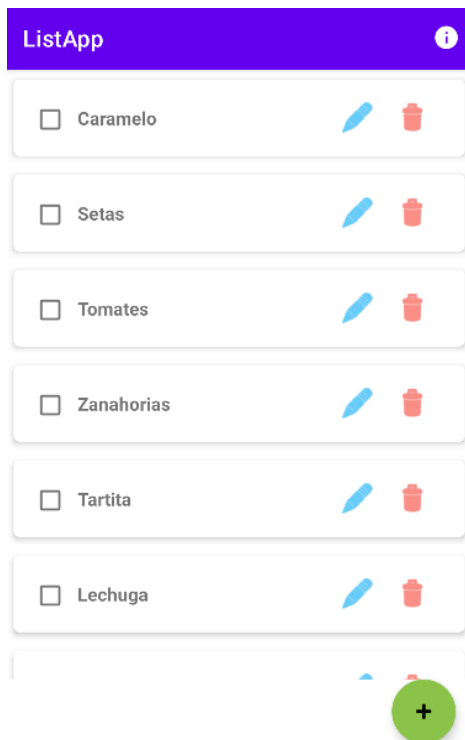


Img. 4 Pantalla tras cancelar



Img. 5 Al guardar, se añade el elemento

Tras añadir varios elementos, la lista quedaría así:



Como se puede apreciar, se han añadido varios elementos:

Caramelo, Setas, Tomates, Zanahorias, Tartita, Lechuga...

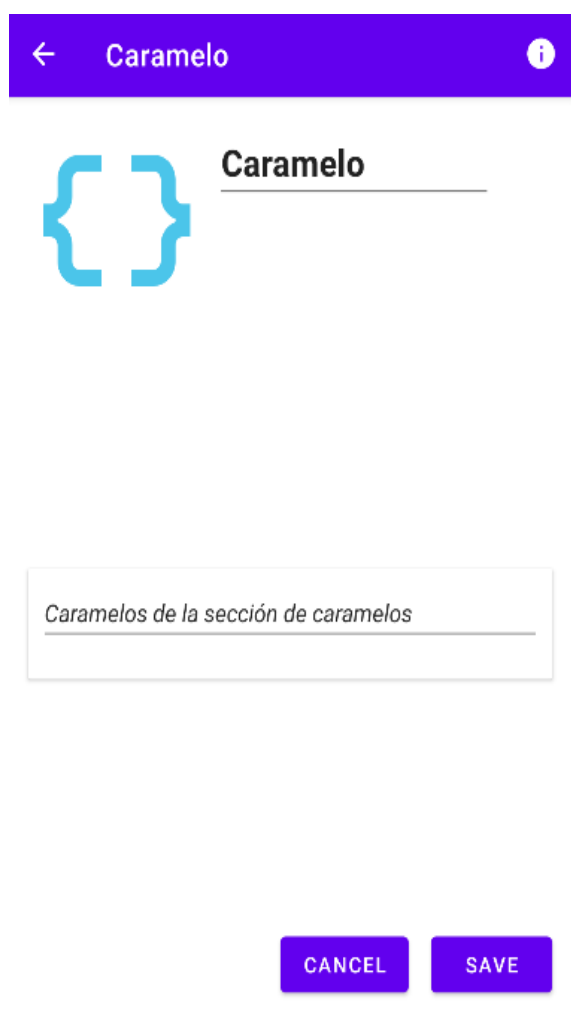
Y aunque no se ve, se puede intuir que hay otro elemento bajo Lechuga. Bueno, esto es gracias a que se usa una RecyclerView para la lista, lo que permite mostrar una lista "infinita" reciclando los elementos que se muestran.

Si hacemos scroll hacia abajo, veremos que hay dos elementos más: Carne y Té verde.

Veremos este detalle en mayor profundidad en el vídeo explicativo.

Img. 6 Pantalla Añadir elemento (add item)

Ahora, si editamos cualquiera de los elementos, por ejemplo “Caramelo” (que hemos creado previamente), veremos la siguiente pantalla al pulsar en el lápiz azul (botón editar del elemento):



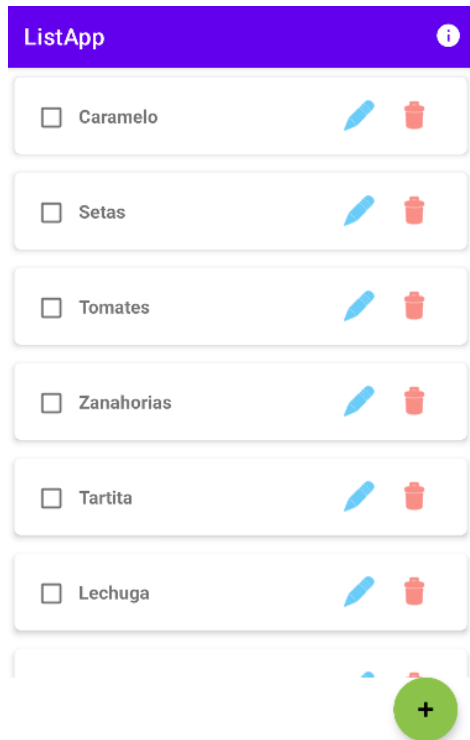
El texto del menú de opciones pasa a ser el nombre del elemento, como se puede apreciar en la imagen 7.

Además, se ha editado la imagen y la descripción.

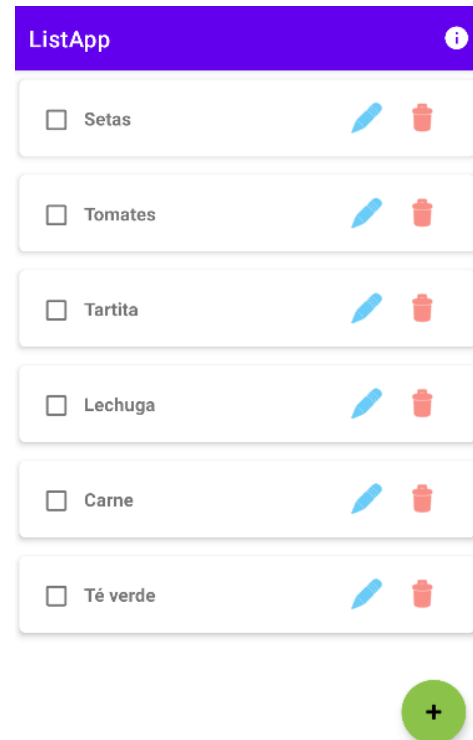
Cualquier cambio realizado se guardará nuevamente si pulsamos “Save” y volverá a la lista, pero se descartará y volverá a la lista igualmente si pulsamos “Cancel”.

Img. 7 Pantalla Editar elemento (add item)

## Eliminar un elemento



Img. 8 Lista de elementos de ejemplo



Img. 9 Al borrar, se elimina el elemento

Al pulsar en el botón de borrar (icono de papelera roja), se elimina el elemento directamente de la base de datos y, por tanto, de la lista. Utilizando el ejemplo anterior, borramos el elemento Caramelo. Vamos a borrar también el elemento Zanahoria para se aprecie mejor. Finalmente, la lista queda de la siguiente forma (imagen 9).

Como podemos apreciar en la imagen 9, los elementos se han eliminado de la lista y ahora Carne y Té verde son visibles sin hacer scroll.

Además, se puede hacer click en el checkbox y el estado del mismo se guarda directamente en la base de datos y aunque cerremos la aplicación, se mantiene.

El último caso es que tanto en ListFragment como en DetailFragment se pulse el icono info del menú de opciones (arriba a la derecha).

Al pulsarlo, se abrirá una nueva Activity (InfoActivity) que carece de Layout propio y está implementada mediante código, ya que únicamente tiene el texto siguiente (imagen 10):



Como la UI es muy sencilla, se ha optado por no crearla mediante Layouts, ya que luego hay que inflar la vista y vincular los elementos visuales (un TextView en este caso).

Con unas pocas líneas de código en la Activity se consigue el resultado deseado.

El texto dice “Aplicación creada por Roldán Sanchis Martínez 2º de DAM.

Aplicación creada por  
Roldán Sanchis Martínez  
2º de DAM

*Img. 10 Pantalla Info con la información*



## Justificación del código

Se ha optado por separar el código según la capa que ocupa/utiliza (p.e. UI para componentes visuales). Y se han implementado varios patrones de arquitectura y diseño de software.

El patrón principal y que mejor se aprecia es el patrón arquitectónico MVVM (Model-View-ViewModel) que permite separar las capas del modelo (datos), las vistas (UI) y el modelo de vista (o ViewModel) que actúa como intermediario entre ambos mediante datos observables (LiveData) que notifican de forma automática estos cambios y a los que se puede asociar eventos. Como podemos ver en el siguiente fragmento de código (imagen 11):

```
/* Muestra todos los elementos de la base de datos y los actualiza cuando cambian */  
viewModel.getAllItems().observe(viewLifecycleOwner) { list ->  
    listAdapter.itemList = list  
    listAdapter.notifyDataSetChanged()  
}
```

Img. 11 ViewModel devolviendo LiveData observada para mantener la UI actualizada.

El fragmento mostrado se asegura de que la lista (UI) se actualice cada vez que la lista de elementos cambie, lo que permite que la lista mostrada siempre esté actualizada.

Otro patrón empleado y muy común en programación y desarrollo es el patrón de diseño Singleton, que se asegura de que una clase tenga una única instancia y proporciona un punto de acceso global a ésta, como por ejemplo en una base de datos. Como podemos apreciar en la siguiente imagen (imagen 12):

```
/** Patrón Singleton de la base de datos */  
companion object {  
    @Volatile  
    private var INSTANCE: ItemRoomDatabase? = null  
  
    /** Devuelve la instancia de la base de datos de Room */  
    fun getDatabase(context: Context): ItemRoomDatabase {  
        return INSTANCE ?: synchronized(lock: this) {  
            val instance = Room.databaseBuilder(  
                context.applicationContext,  
                ItemRoomDatabase::class.java,  
                name: "item_database"  
            )  
                .fallbackToDestructiveMigration()  
                .build()  
            INSTANCE = instance  
            instance  
        }  
    }  
}
```

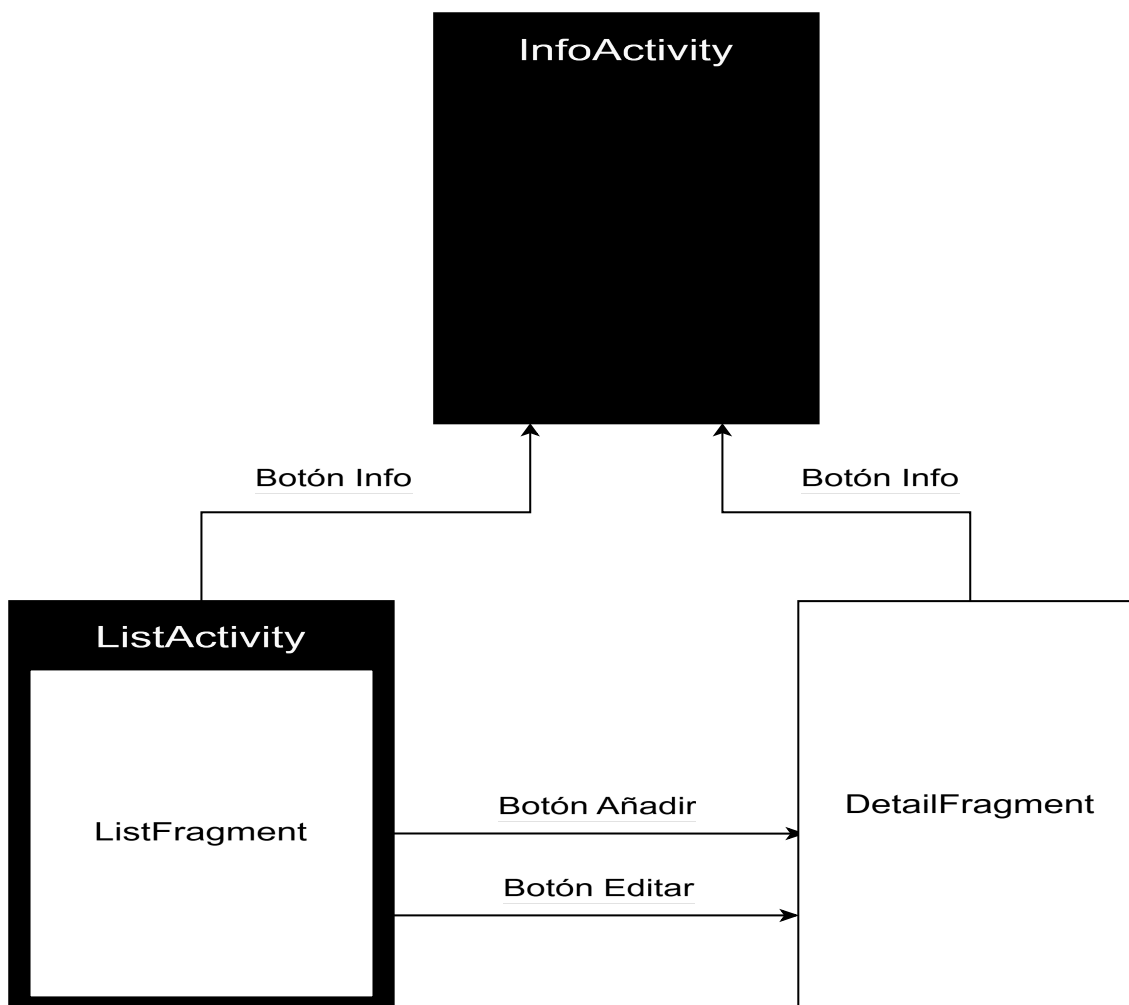
Img. 12 Ejemplo de patrón Singleton.

En cuanto a la decisión de usar Fragments en lugar de Activities, surge en base a que la vista principal es una lista mostrada con una RecyclerView y era más sencillo y eficiente de implementar mediante Fragments. Al final como en los requisitos de creación de la aplicación estaban las dos Activities, se optó por añadir un menú de opciones con un botón para abrir la pantalla de Info y hacer que fuera una Activity en lugar de otro Fragment.

El código está enteramente en inglés por decisión personal (de cara a futuro, prefiero programar en inglés) pero los comentarios están en español para mejorar la comprensión de los mismos de cara a un “público” de habla hispana.

Se ha intentado seguir las convenciones generales en cuanto a nombres de métodos, clases, variables, carpetas y demás elementos.

## Estructura de la app



Img. 13 Diagrama de la estructura de la aplicación