# App de Lista de la compra v2 (Enunciado)

Elaborar una versión algo más completa de una app tipo lista de la compra. Leer bien todo el enunciado hasta el final antes de comenzar, una vez hecho esto, partir de un proyecto vacío o un proyecto con una `activity` sencilla de Compose.

La aplicación debe tener un `datasource` dentro del paquete **data** llamado `ProductData.kt`. El contenido de este archivo está adjunto al enunciado del examen para que podáis añadirlo directamente en el proyecto, pero debéis crear el paquete en su lugar correspondiente.

La aplicación debe tener también un **Modelo** de datos para los Productos en un archivo `Product.kt` también en su paquete **model** correspondiente. El contenido de este archivo también se adjunta al enunciado del examen y también debe posicionarse en un lugar adecuado.

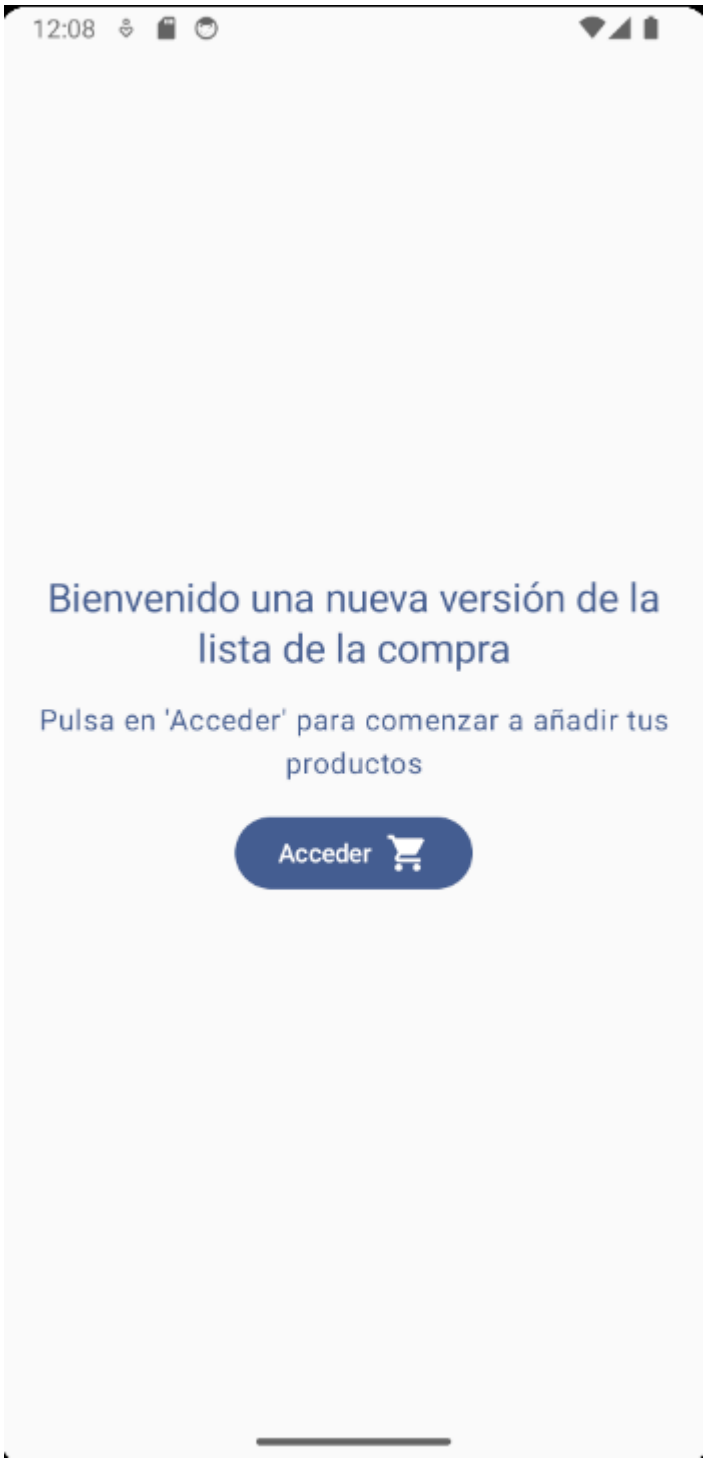
En este último archivo encontraréis la definición de qué propiedades tiene un Producto y dos métodos interesantes para la implementación de la app:

- *getFakeProducts(size: Int = 10): List*
    - Genera una lista de Productos aleatorios.
- *getFakeProduct(): Product*
    - Genera un producto aleatorio.

Ambos deben usarse en su momento para el correcto funcionamiento de la app.

La aplicación debe constar de 3 pantallas:

- Una pantalla de **Home**:




La pantalla de **Home** simplemente debe contener esos tres elementos tal y como aparecen en la imagen.

Para conseguir un resultado idéntico se puede usar la tipografía `titleLarge` por defecto para el título y la tipografía `bodyLarge` para el subtítulo. El color de los textos es `primary`.

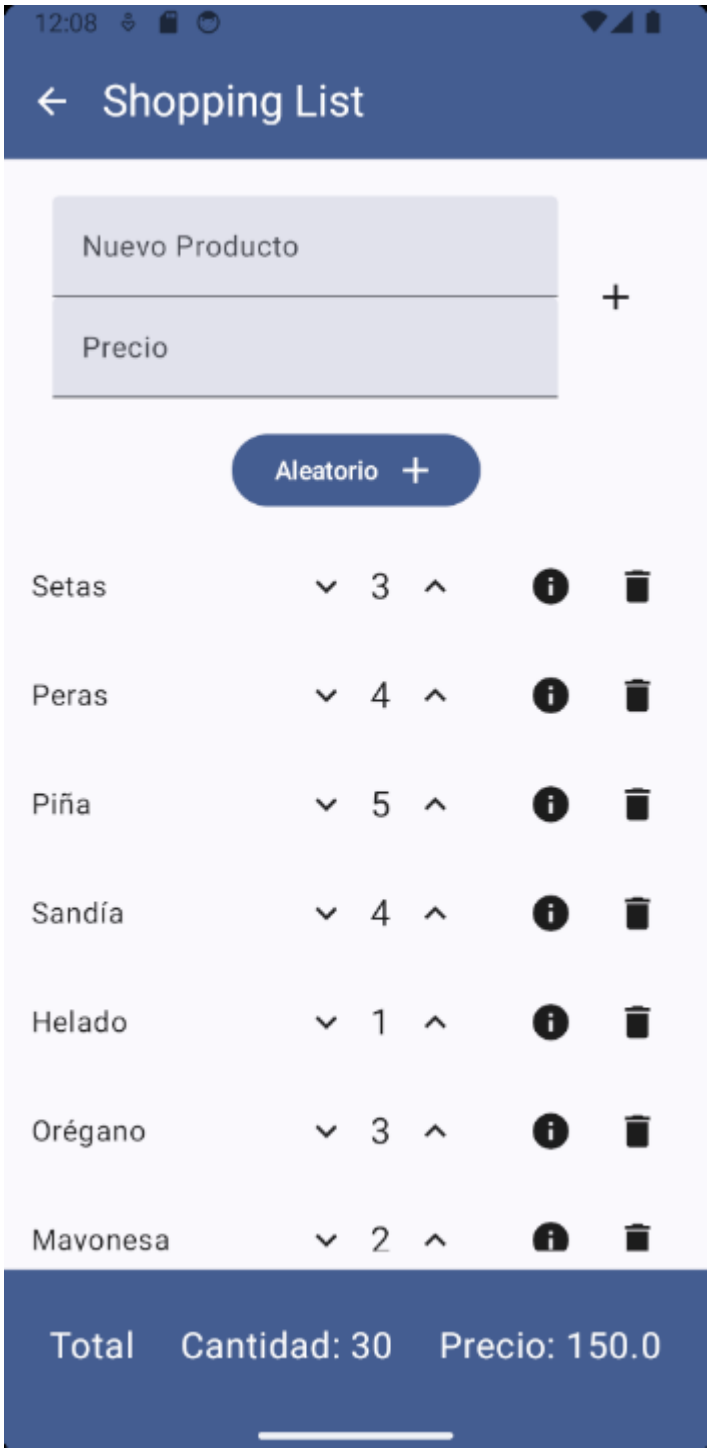
Ambos textos están alineados al **centro**.

Los elementos están separados por un `Spacer` de `16.dp`.

El botón tiene un `Icon` de las librerías de iconos por defecto de Material llamado `ShoppingCart` y una separación con respecto al texto de `8.dp`.

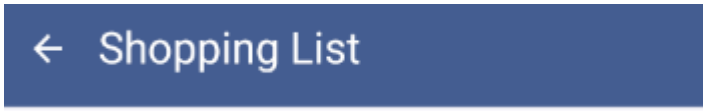
Es importante intentar mantener las `Strings` en recursos para no hardcodear.

- Una pantalla de **ListaCompra**:




**IMPORTANTE**: Al entrar en esta pantalla **se deben generar automáticamente una lista de la compra con 10 productos aleatorios** de entre los posibles del `datasource` haciendo uso del método *getFakeProducts()* disponible en el archivo adjunto antes mencionado.

En la pantalla de lista de la compra debemos añadir una barra superior con el título correspondiente y un botón para navegar de vuelta a la pantalla de **Home**. Los colores de la barra son `primary` para el fondo y `onPrimary` para todo lo demás.



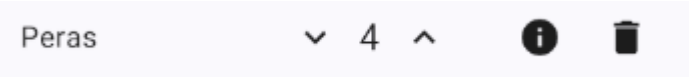
En el contenido de la pantalla debemos tener una primera sección con dos `TextField` dónde el usuario podrá introducir el nombre de un nuevo producto a añadir y su precio, junto con un `Button` a la derecha de estos elementos para añadir un nuevo elemento a la lista. Pero para añadir debemos de respetar lo siguiente:

- El producto que añadamos no debe estar vacío, en caso de que no tenga nombre o el precio sea incorrecto debemos indicarlo usando un mensaje `Toast` correspondiente.
- El producto no debe existir ya en la lista, en caso de que así sea y el usuario intente introducir un producto ya existente debemos lanzar de nuevo un aviso usando un `Toast` que indique el problema y vacíar los `TextField` de nuevo para una **UX** correcta.
- El producto que añadamos debe posicionarse en la primera posición de nuestra lista de la compra.
- En el `TextField` del *Precio* es importante añadir el parámetro `keyboardOptions` y pasarle el valor: `KeyboardOptions(keyboardType = KeyboardType.Number)` para que cuándo el usuario en su teléfono pulse en ese campo del formulario se le abra únicamente el teclado numérico.


En esta sección también existirá debajo del pequeño formulario, otro botón para añadir un nuevo producto *Aleatorio*. Esto funcionará de la siguiente forma:

1. Generará un producto aleatorio y comprobará si este ya existe en la lista.
2. Si el producto ya existe, genera otro hasta encontrar uno que no exista.
3. Añade el producto generado en la primera posición de la lista de la compra.

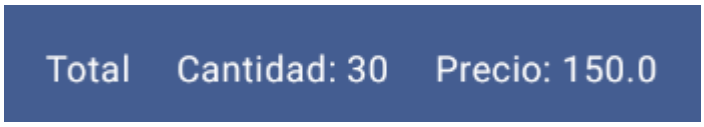
Además de este bloque de añadir nuevos productos, la pantalla debe contener también la propia lista de productos, esta lista debe estar compuesta por elementos con la siguiente **estructura**, de izquierda a derecha:




1. Un texto con el nombre del producto.
2. Un `IconButton` para disminuir en 1 la cantidad a comprar de ese producto en concreto.
    - Al pulsarlo debe disminuir en uno la cantidad pero si ya solo quedaba 1 producto de este elemento debe de borrar el producto de la lista.
    - El icono a utilizar para este botón es `KeyboardArrowDown`.
3. Un texto que muestre la cantidad de este producto.
4. Un `IconButton` para aumentar en 1 la cantidad a comprar de ese producto en concreto.
    - El icono a utilizar para este botón es `KeyboardArrowUp`.

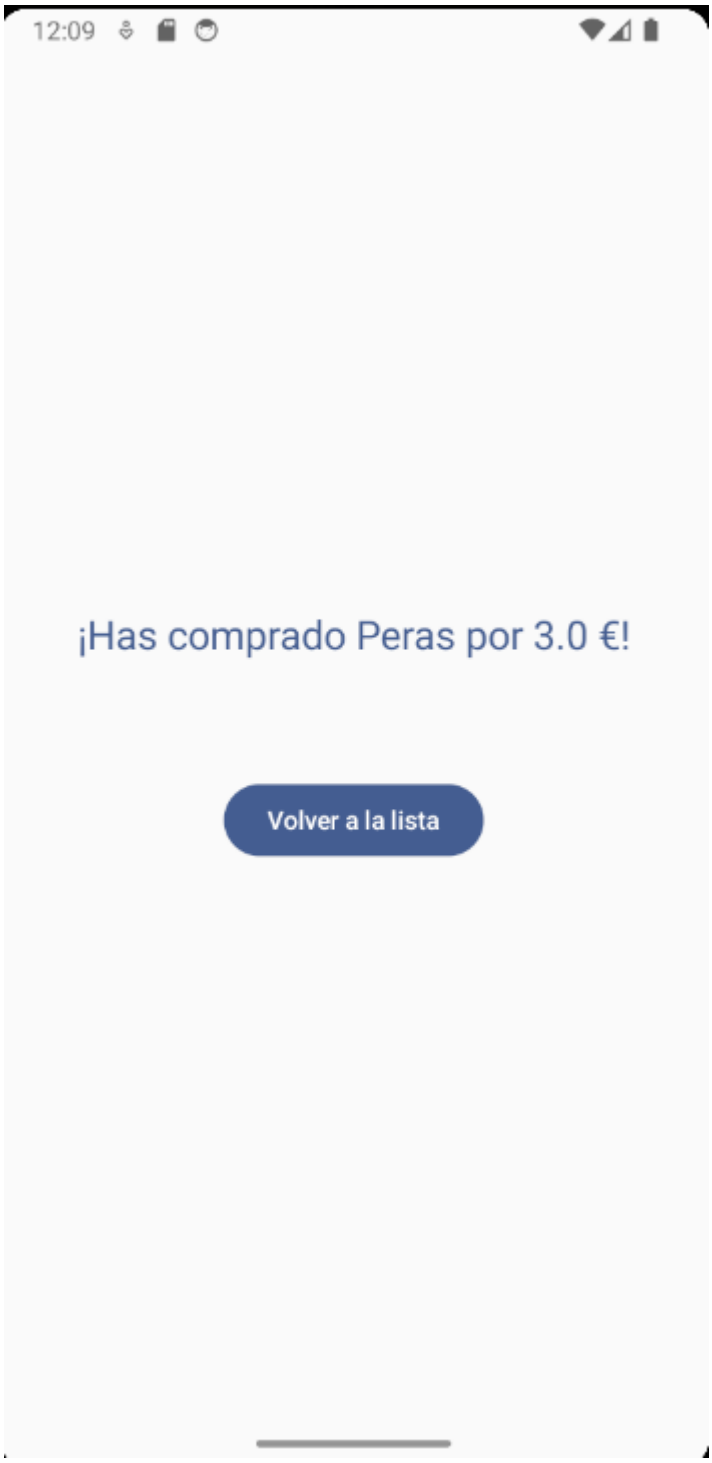
5. Un `Spacer` de `16.dp` de **ancho**.
6. Un `IconButton` que nos lleve a la pantalla de **Detalle** para ese producto en concreto y así poder consultar su precio.
    - El icono a utilizar para este botón es `Info`.
7. Un `IconButton` para eliminar este producto de la lista completamente.
    - El icono a utilizar para este botón es `Delete`.

Por último, en esta pantalla tenemos una **barra inferior** que simplemente es una `Row` con tres elementos `Text`, en el primero tendremos el texto fijo "*Total*" y en los dos siguientes tendremos la información de la **Cantidad** total de productos que hay en la lista, y el **Precio Total** que nos va a costar nuestro carrito de la compra. Estos valores deben actualizarse de forma dinámica según vamos añadiendo o elminando productos, o bien modificando la cantidad que compramos de cada uno de ellos.




Para que nos sea más **fácil posicionar esta barra inferior** podemos hacer uso del atributo `bottomBar` del elemento `Scaffold` y situar nuestro **Composable** correspondiente dentro del mismo. Para darle el color de fondo podemos introducirlo dentro de un elemento de **Material** llamado `BottomAppBar` y modificar sus atributos `containerColor` y `contentColor` para conseguir el diseño de las capturas son `primary` para el primero y `onPrimary` para el segundo.

- Una pantalla de **Detalle**:




En esta pantalla tan solo tendremos un pequeño texto que nos indique el **precio unitario** del producto concreto desde el que hayamos navegado, es decir, su **precio para una única unidad de cantidad**.

También tendremos un botón para volver a la pantalla de la lista.

El estilo del texto es `tittleLarge` y su color es `primary`.

La separación entre el texto y el botón es un `Spacer` de `64.dp`.

## A tener en cuenta

- La aplicación debe gestionar la navegación enviando lambdas desde el `NavHost` y no propagando el `navController`, ya que cada pantalla solo puede ir a 1 o 2 pantallas como máximo.

- Desde la pantalla de **Lista** a la de **Detalle** es necesario enviar dos argumentos al navegar con la información necesaria para poder mostrarla correctamente.
- El estado de la pantalla de la lista debe de controlarse de forma aislada, es decir, deberíamos gestionar todos los eventos desde un `ViewModel` y definir muy bien el **model** de la **UI** para esta pantalla. Es la única de las 3 que lo necesita. El ejercicio puede resolverse sin hacer esto pero tendrá penalización en la calificación ya que el correcto uso de la arquitectura **MVVM** es una parte de la nota.

## Rúbrica orientativa de calificación - no definitiva, podría sufrir cambios

Total de puntos 100

| Criterio | Puntos | Descripción del nivel de cumplimiento |
|---|---|---|
| **Funcionamiento general de la app (30%)** | | |
| Correctitud de la lógica principal | 10 | **Alta (10)**: Todas las funcionalidades descritas en el enunciado funcionan correctamente.<br>**Media (5-9)**: Algunas funcionalidades presentan errores menores.<br>**Baja (0-4)**: Múltiples errores graves. |
| Gestión de errores | 10 | **Alta (10)**: La app maneja correctamente escenarios como entrada de datos inválidos o listas vacías.<br>**Media (5-9)**: Manejo parcial de errores.<br>**Baja (0-4)**: No hay manejo de errores. |
| Rendimiento | 5 | **Alta (5)**: La app responde fluidamente y sin problemas.<br>**Media (3-4)**: Leves ralentizaciones.<br>**Baja (0-2)**: Problemas serios de rendimiento. |
| Generación y manejo de productos aleatorios | 5 | **Alta (5)**: Generación consistente y sin errores.<br>**Media (3-4)**: Algunos problemas menores.<br>**Baja (0-2)**: La generación falla frecuentemente o no se implementa. |
| **UI y UX (20%)** | | |
| Diseño visual | 5 | **Alta (5)**: La app sigue principios de diseño claros y utiliza correctamente Material Design.<br>**Media (3-4)**: Diseño funcional pero básico.<br>**Baja (0-2)**: Diseño desordenado o confuso. |
| Usabilidad | 5 | **Alta (5)**: La app es fácil de usar y los elementos interactivos son intuitivos.<br>**Media (3-4)**: Usabilidad aceptable con algunos problemas.<br>**Baja (0-2)**: Dificultades claras en la usabilidad. |

| Criterio | Puntos | Descripción del nivel de cumplimiento |
|---|---|---|
| Implementación de la TopBar y BottomBar | 5 | **Alta (5)**: Barras implementadas y funcionales según lo descrito.<br>**Media (3-4)**: Barras con fallos menores.<br>**Baja (0-2)**: Barras faltantes o mal implementadas. |
| Experiencia en la pantalla de detalles | 5 | **Alta (5)**: La pantalla de detalles se muestra correctamente y contiene los datos esperados.<br>**Media (3-4)**: Detalles incompletos o errores leves.<br>**Baja (0-2)**: Falla en su implementación. |
| **Navegación entre pantallas (25%)** | | |
| Fluidez de navegación | 10 | **Alta (10)**: Navegación consistente y sin errores.<br>**Media (5-9)**: Navegación funcional con algunos problemas menores.<br>**Baja (0-4)**: Navegación confusa o con errores frecuentes. |
| Paso de parámetros entre pantallas | 10 | **Alta (10)**: Parámetros pasados correctamente en todas las transiciones.<br>**Media (5-9)**: Algunos parámetros faltan o tienen problemas.<br>**Baja (0-4)**: Parámetros no se pasan correctamente. |
| Manejo del botón "atrás" | 5 | **Alta (5)**: Funciona correctamente en todas las pantallas.<br>**Media (3-4)**: Fallos menores en el comportamiento.<br>**Baja (0-2)**: El botón "atrás" no funciona o tiene problemas serios. |
| **Uso correcto del estado y MVVM (25%)** | | |
| Gestión del estado | 10 | **Alta (10)**: El estado está bien gestionado con recomposición eficiente.<br>**Media (5-9)**: Algunos problemas en la gestión del estado.<br>**Baja (0-4)**: Gestión del estado incorrecta o ausente. |
| Separación de responsabilidades | 5 | **Alta (5)**: Arquitectura MVVM bien implementada.<br>**Media (3-4)**: Separación de capas con problemas menores.<br>**Baja (0-2)**: MVVM no implementada correctamente. |
| Uso de ViewModel | 5 | **Alta (5)**: ViewModels usados de manera eficiente y correctamente.<br>**Media (3-4)**: Uso limitado o incorrecto.<br>**Baja (0-2)**: No se usa ViewModel o se usa de forma inapropiada. |
| Recomposición eficiente | 5 | **Alta (5)**: Recomposición controlada y eficiente.<br>**Media (3-4)**: Recomposición innecesaria o poco optimizada.<br>**Baja (0-2)**: Problemas graves de recomposición o bloqueos frecuentes. |