


Ejercicio Propuesto Tienda Room

[Descargar estos apuntes](#)

Vamos a crear una aplicación para profundizar más en la **gestión de Bases de Datos con Room**. Se tratará de implementar una mini tienda para que los clientes puedan hacer pedidos de una serie de artículos que ofrece esta. En la siguiente imagen se mostrará la estructura de paquetes que podeis seguir, para que los componentes de la aplicación estén más organizados:

```
java
├── com.ejemplo.b11.ejerciciopropuestotiendaroom
│   ├── Adaptadores
│   ├── DataBase
│   ├── modelo
│   ├── View
│   └── ViewModel
```

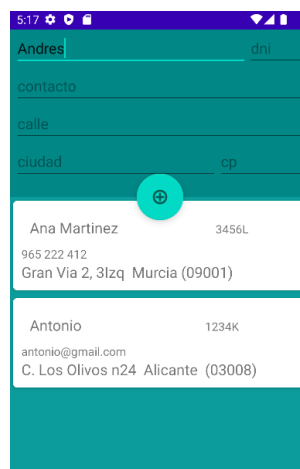
Además para no perder tiempo en la implementación de las vistas y los adaptadores, se os pasan como recursos adjuntos todos los ficheros necesarios, aunque podéis personalizar la aplicación a vuestro gusto. *Importante alojar los adaptadores en la carpeta correspondiente.*

 **Importante:** Para mejorar la independencia del código, en el paquete ViewModel se alojará:

- una clase object que contendrá la inicialización de los artículos (se comentará después del primer uso para evitar duplicados)
- la instanciación de la BD para poder realizar su uso.
- además de los métodos necesarios para comunicarse con las funcionalidades de Dao, de forma que está será la clase que tiene relación directa con las incluidas en el paquete de la arquitectura Room, database.

FragmentCliente

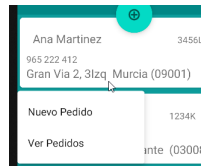
La aplicación se presentará con un fragment principal que permitirá la gestión de clientes: añadir, mostrar y eliminar. La eliminación se hará con el click largo, añadir mediante el FAB y los elementos añadidos se mostrarán con el recycler.



Todos los fragments y las activitys se alojarán en el paquete View.

Para la gestión anterior solo necesitaremos la entidad Cliente con los campos que se visualizan en la imagen y con el dni como clave primaria.

Si pulsamos de forma corta en uno de los elementos del recycler, se mostrará un popup menú con el siguiente aspecto:



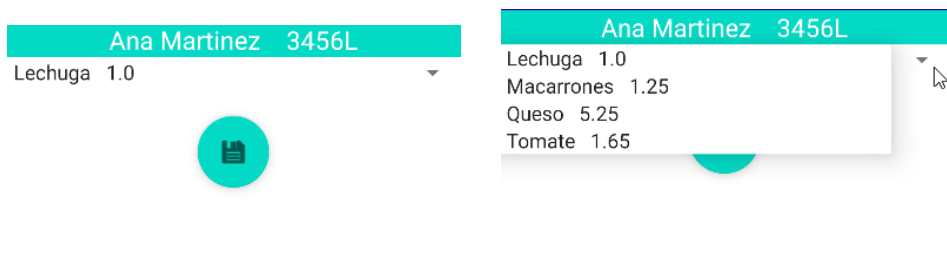
La selección de cada opción del menú, cargará el fragment correspondiente.

FragmentPedido

Servirá para realizar el pedido del cliente pulsado, lo primero que hará este fragment es cargar en un spinner los articulos a la venta de la tienda. Estos artículos se cargarán de forma automática al crear la aplicación. Podemos usar una clase object alojada en ViewModel que devuelva la siguiente lista de artículos, por ejemplo:

```
object InicializarArticulos {  
    fun listaArticulos():MutableList<ArticuloEntity>  
    {  
        var lista= mutableListOf<ArticuloEntity>()  
        lista.add(ArticuloEntity(0,"Macarrones",1.25f))  
        lista.add(ArticuloEntity(0,"Queso",5.25f))  
        lista.add(ArticuloEntity(0,"Tomate",1.65f))  
        lista.add(ArticuloEntity(0,"Lechuga",1.00f))  
        return lista  
    }  
}
```

Los elementos de esta lista se insertarán en la entidad Articulos para posteriormente relacionarlos con los pedidos. El id del artículo será autoincrementable, por eso se inserta como 0 en el constructor.



Al seleccionar un artículo del Spinner se mostrará en el recycler inferior, podremos incrementar o decrementar la cantidad con los iconos correspondientes, cuando se llegue a 0 el artículo dejará de mostrarse en el recycler. Para esto utilizaremos una lista de ArticulosPedido, formada por una data clase con los mismos campos que la de articulos entity, más la cantidad que solo será necesaria a la hora de realizar los pedidos. *Esta data class se alojará en el paquete modelo.*

Ana Martínez 3456L	
Queso	5.25
2 Macarrones	1.25
1 Tomate	1.65
1 Queso	5.25

Al pulsar sobre el FAB se procederá a guardar los datos generales del pedido en la tabla Pedido, para ello crearemos una entidad como la siguiente:

```
data class PedidoEntity(
    @PrimaryKey(autoGenerate = true) // El id será autogenerado.
    @ColumnInfo(name = "pedidoId")
    val pedidoId: Long,
    @ColumnInfo(name = "dniCliente")
    val dniCliente: String,
    @ColumnInfo(name = "total")
    val total: Float,
    @NonNull
    @ColumnInfo(name = "fecha")
    // Indicamos que siempre debe tener una fecha.
    val fecha: Long
)
```

En esta tabla no se guardarán los artículos, este proceso lo haremos posteriormente, una vez que nos funcione bien la inserción del pedido.

Entre Cliente y Pedido hay una relación uno a muchos que deberás de crear, tienes la información en los apuntes y en la página oficial. Puedes comprobar que la BD está guardando los datos correctamente usando App Inspection.

Una vez que tengamos los datos del pedido guardado, pasaremos a guardar los artículos que lo componen. Para ello deberemos crear una relación **muchos a muchos entre Pedido y Artículo**, encontraremos la información en la página principal. En este caso se crea una entidad intermedia ArticulosConPedido con los dos id como clave primaria, y además añadiremos el campo cantidad. Por tanto, al pulsar sobre el FAB de guardar pedido, tendremos que insertar los datos en la tabla Pedido y además insertar en la tabla ArticulosConPedido el id de los Articulos de la lista de la compra con sus cantidades, junto con el id del pedido.

FragmentListaPedidos

Este fragment se cargará con la otra opción del menú Popup, y en este caso se visualizará la fecha y el coste de todos los pedidos del cliente seleccionado.

Ana Martínez 3456L	
11/01/2023 18:19	9.25
11/01/2023 18:18	14.35

Al pulsar sobre un pedido, se mostrará un dialogo con los artículos (cantidad, nombre y precio) que componían el pedido.

En el dialogo se puede mostrar la información como una cadena en la descripción.

Ana Martinez 3456L	
11/01/2023 18:19	9.25
11/01/2023 18:18	14.35
<div>CONTENIDO DEL PEDIDO</div> <div>2 Macarrones 1.25 1 Queso 5.25 4 Tomate 1.65</div> <div>OK</div>	