Módulo 4:
Room 03. MVVM, LiveData, Coroutines.

<title> Objetivos del día </title>

- Terminar ejercicio anterior.
- Base de datos y Room.
  - MVVM
  - Livedata
  - Coroutines
- Observadores en Vistas.
  - Adapter y RecyclerView
  - Listener y callback

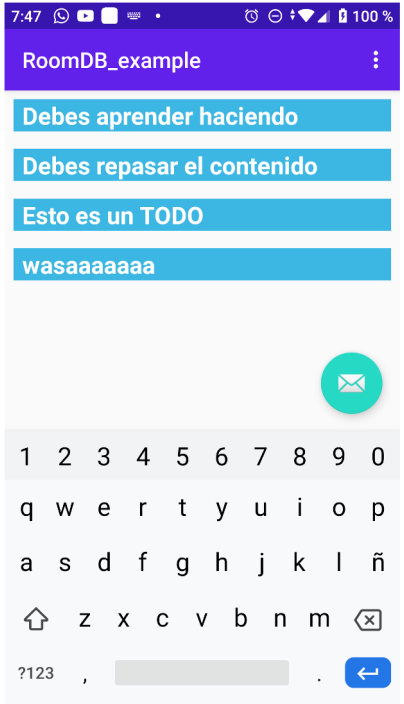
Añadir Room a un proyecto

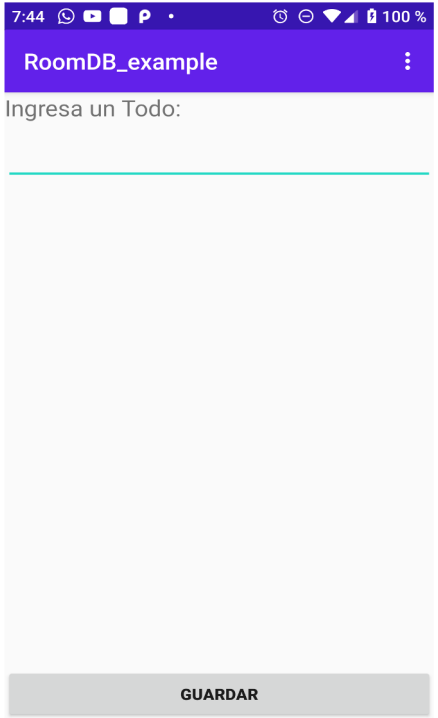
Ejercicio: Implementar Room + RV+ ¿? (To-do App)

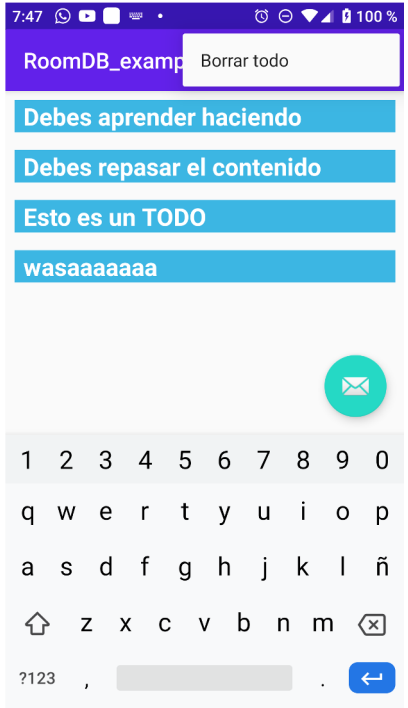
Vamos a crear una app e implementar Room database.

Además consolidar algunos conocimientos por ejemplo:

- RecyclerView
- Consultas SQL
- ¿ MVP o MVVM ?
- Callbacks
- Fragmentos





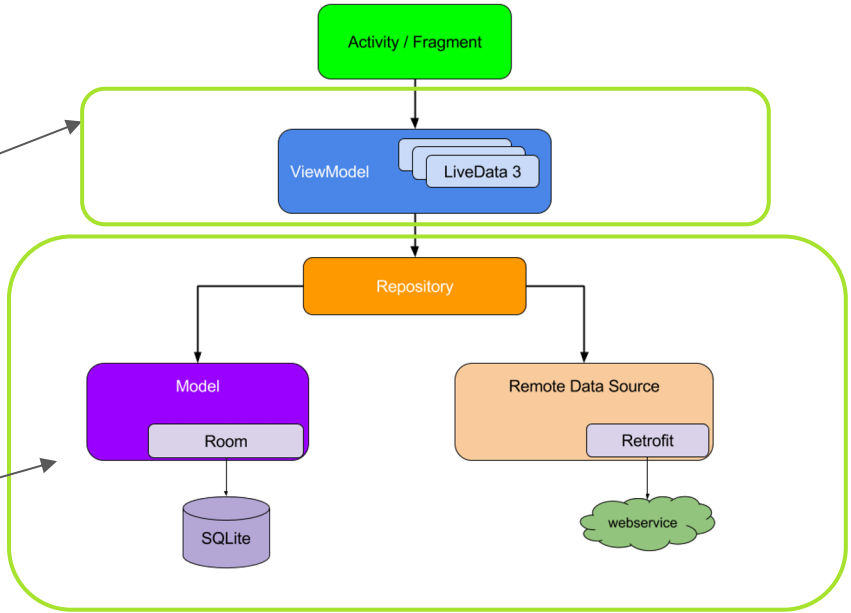


¿Que tenemos construido ?

Ayer construimos las capas de ViewModel y Modelo. Y parte de las vistas.

ViewModel, clase que se encargará de proporcionar la información a las vistas.

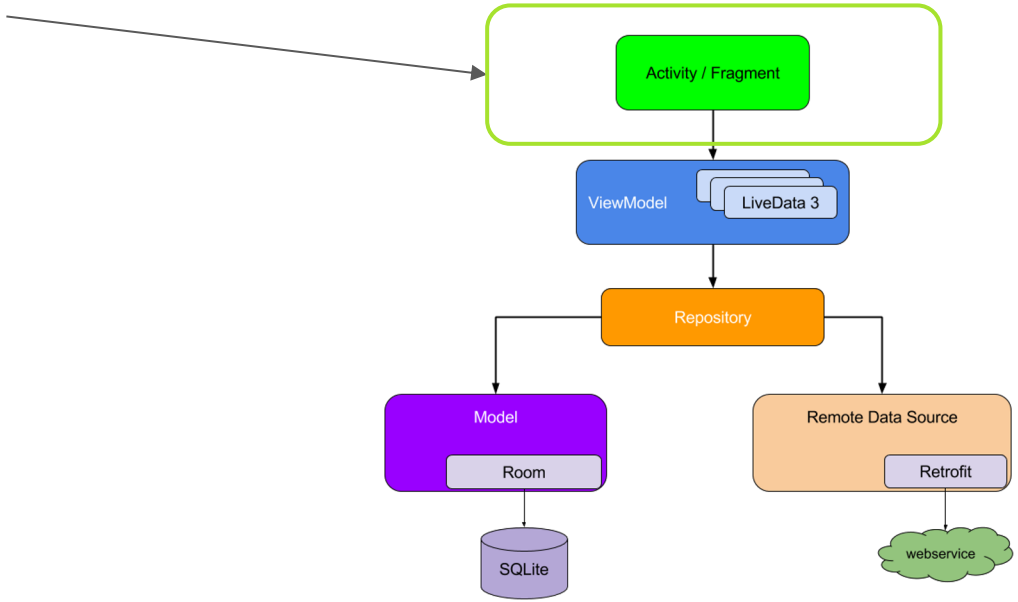
Modelo, constituido por un repositorio quien se encarga de entregar datos al viewModel. Estos datos provienen de la base de datos administrada por Room.



Ahora vamos a construir las vistas

Las vistas van a observar los datos Livedata que el viewModel le indique y se mantendrán actualizadas.

Construir todos los elementos visuales que mostraran los datos.

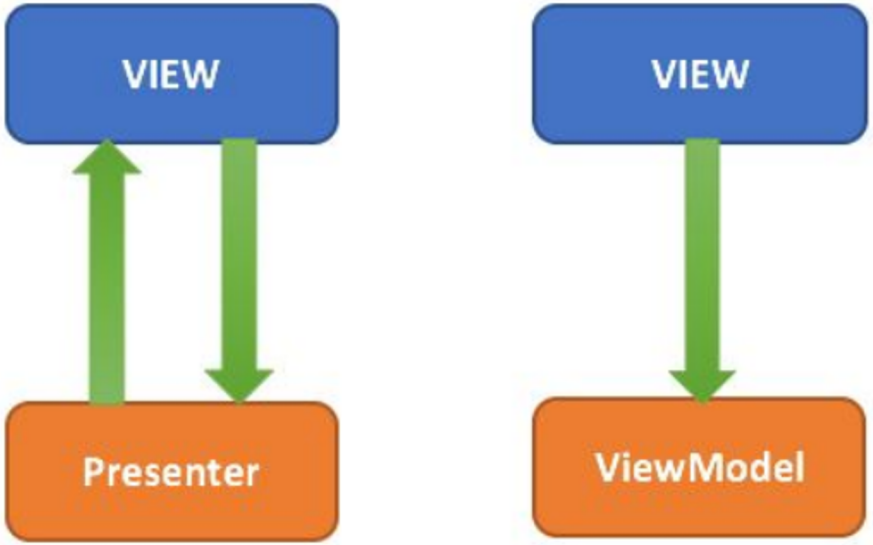


¿Que necesitamos conocer?

Para construir nuestra App Todo vamos a tener que conocer algunos nuevos elementos por ejemplo:

- Patrones de arquitectura recomendados.
- MVVM.
- Patrón Observador y LiveData.
- Arquitectura recomendada.
- Repositorio.
- Coroutines.

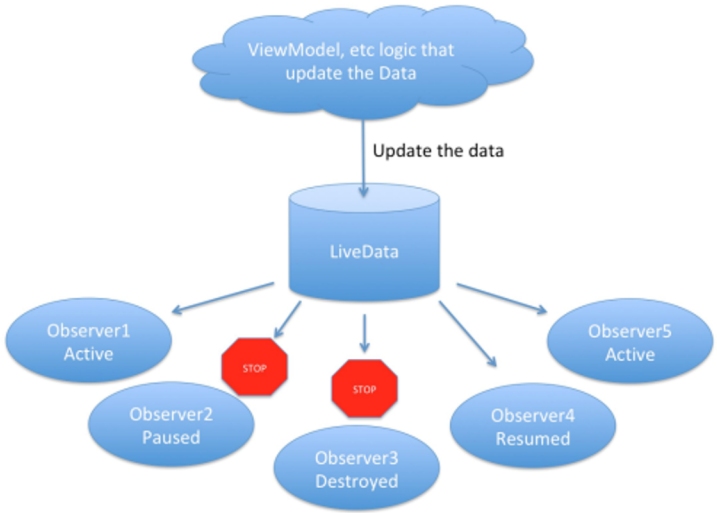
Diferencias entre MVP y ViewModel



Fuente: miro.medium.com

LiveData

- UI siempre actualizada.
- Manejo automático del ciclo de vida.
- Manejo automático de cambios de configuración.
- No más pérdidas de datos o memory Leaks



MVVM

MVVM

En el patrón de arquitectura modelo vista Vista-Modelo (MVVM) se incorporó como con la llegada de los componentes de arquitectura proporcionados por google (JetPack).

La idea detrás de estos componentes y las recomendaciones de arquitectura, es facilitar la vida del desarrollador.

Poniendo en perspectiva esta recomendación de arquitectura descansa mucho en elementos que manejan de forma automática el ciclo de vida, y también componentes modernos como Live Data, que sirven para observar cambios y reaccionar de forma automática, sumandole la buena simbiosis que tienen estos nuevos componentes.

ViewModel

El principal componente de este patrón recomendado de arquitectura es un objeto ViewModel. El cual se encargará de proporcionar los datos para componentes de vista específicos como Fragmentos o actividades. Además este objeto incluye la lógica de negocio para comunicarse con el modelo, o la fuente de estos datos.

La principal diferencia de este elemento con un presentador es que ViewModel no conoce los componentes de IU, de manera que no se ve afectado por los cambios de configuración, como la recreación de una actividad debido a la rotación del dispositivo.

Ciclo de vida de un ViewModel

Por lo general, solicitas un ViewModel la primera vez que el sistema llama al método onCreate().

El sistema puede llamar a onCreate() varias veces durante la vida de una actividad, como cuando rota la pantalla de un dispositivo.

El ViewModel existe desde la primera vez que solicitas un ViewModel hasta que finaliza la actividad y se destruye.

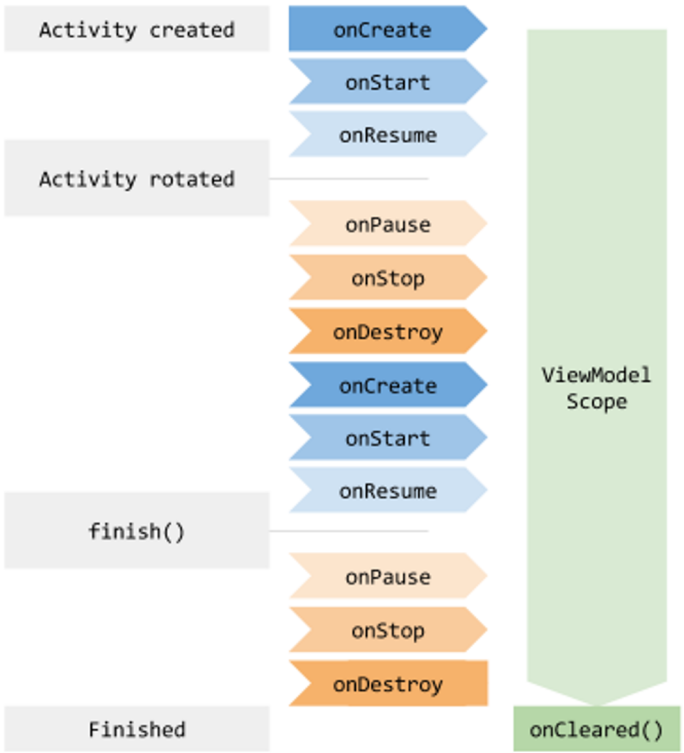
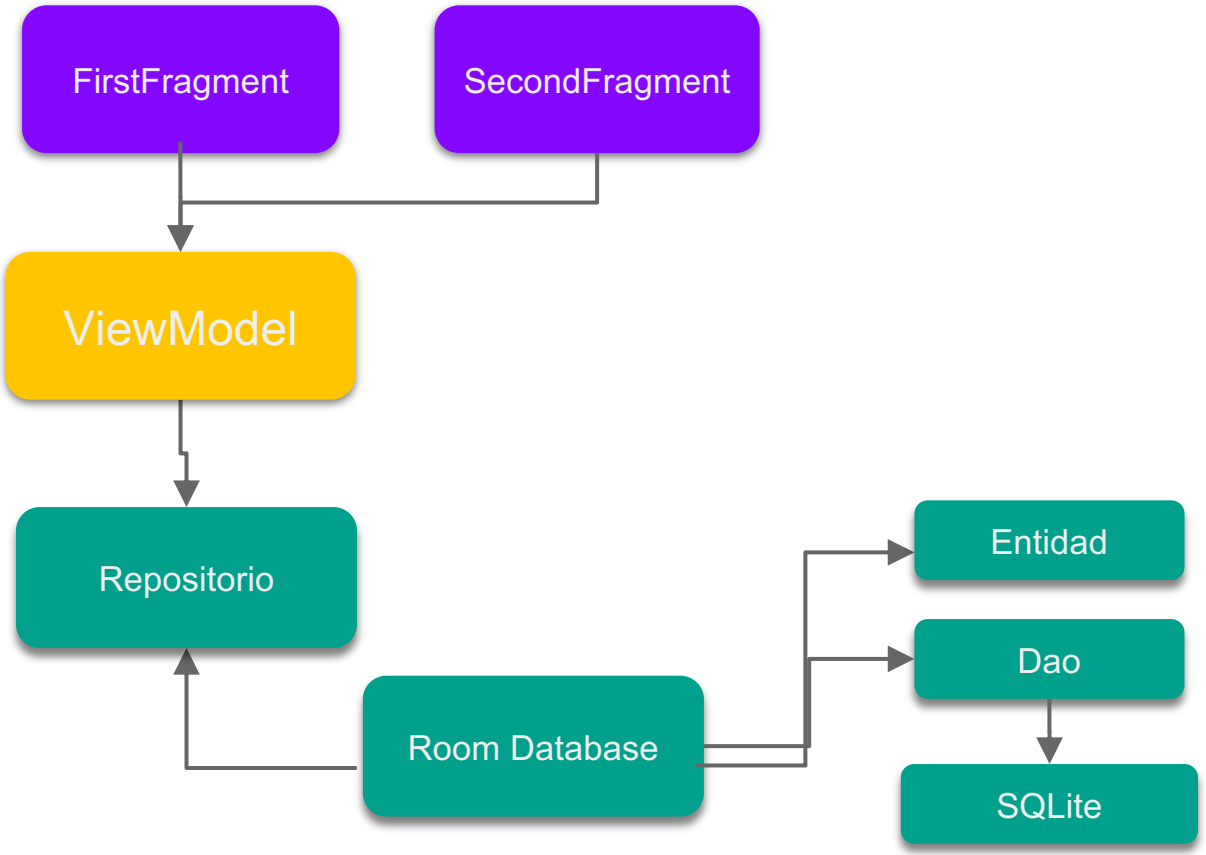


Diagrama de Cómo debería quedar.

En este ejemplo estamos utilizando el mismo ViewModel para los fragmentos.



Coroutines

Las Coroutines o corrutinas son una forma elegante de trabajar con concurrencia dentro del lenguaje Kotlin.

Una corrutina es un patrón de diseño de simultaneidad que puede usar en Android para simplificar el código que se ejecuta de forma asincrónica.

Se agregaron corrutinas a Kotlin en la versión 1.3 y se basan en conceptos establecidos de otros lenguajes.

En Android, las corrutinas ayudan a administrar las tareas de larga duración que, de lo contrario, podrían bloquear el hilo principal y hacer que su aplicación deje de responder.

¿Por qué utilizar Coroutines?

Coroutines es la solución recomendada para la programación asincrónica en Android. Las características dignas de mención incluyen las siguientes:

- **Ligero**: puede ejecutar muchas corrutinas en un solo hilo debido al soporte para suspensión, que no bloquea el hilo donde se ejecuta la corrutina. La suspensión ahorra memoria sobre el bloqueo al tiempo que admite muchas operaciones simultáneas.
- **Menos pérdidas de memoria**: utilice la simultaneidad estructurada para ejecutar operaciones dentro de un ámbito.
- **Soporte de cancelación incorporado**: la cancelación se propaga automáticamente a través de la jerarquía de corrutinas en ejecución.
- **Integración de Jetpack**: muchas bibliotecas de Jetpack incluyen extensiones que brindan soporte completo para corrutinas. Algunas bibliotecas también proporcionan su propio alcance de corrutina que puede utilizar para la concurrencia estructurada.

Utilizando Coroutines con Room

- Utilizar la palabra reservada **suspend** en las funciones que estarán dentro del contexto de la corrutina.
- Utilizarlas dentro del contexto del ViewModel

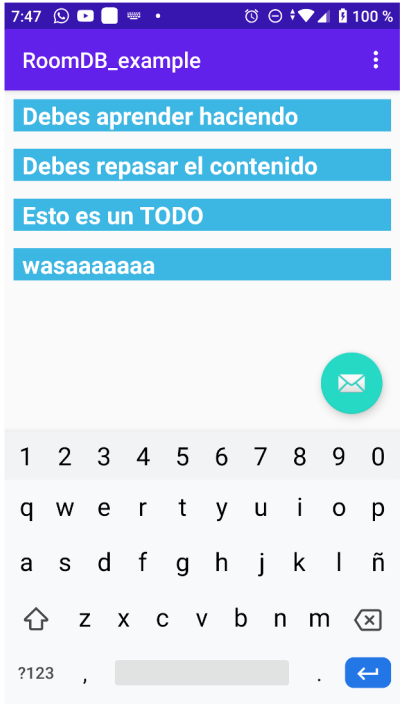
```
@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertTask(task: Task)


@Query("DELETE  FROM table_task")
suspend fun deleteAllTask()



fun insertTask(task: Task) = viewModelScope.launch {
   repository.insertTask(task)
}

//delete all task from database
 fun deleteAllTask() = viewModelScope.launch {
   repository.deleteAll()
}
```

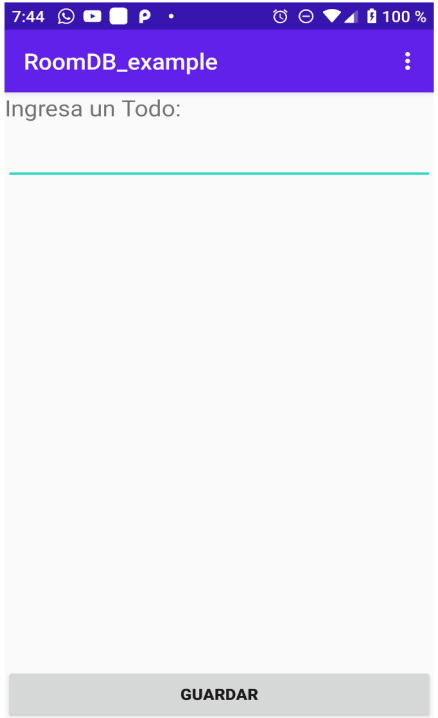
Ejercicio: Implementar Room + RV+ ¿? (To-do App)

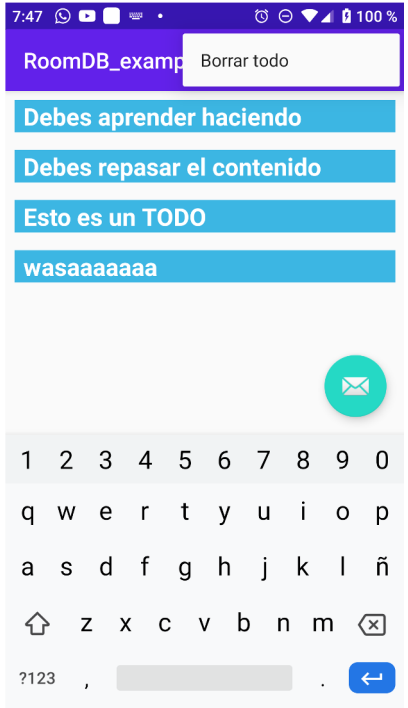
Vamos a crear una app e implementar Room database.

Además consolidar algunos conocimientos por ejemplo:

- RecyclerView
- Consultas SQL
- ¿ MVP o MVVM ?
- Callbacks
- Fragmentos







Ejercicio - Consolidación

- Objetivos Generales:
  - Construir una app siguiendo el patrón de arquitectura recomendado por google (MVVM).
  - Utilizar Room para otorgar persistencia de datos.
  - Utilizar **LiveData** para actualizar la información de las vistas.
  - Utilizar Coroutines para el trabajo asíncrono.
  - Mostrar los datos provenientes de la BBDD en una lista.
  - Utilizar control de versiones (Git) a lo largo del proceso de creación.
- Objetivos específicos:
  - Implementar los elementos necesarios para usar la biblioteca Room.
    - Entities, Dao’s, cliente Base de datos.
  - Utilizar una Clase Repositorio.
  - Utilizar clases ViewModel según corresponda.
  - Utilizar **LiveData** según corresponda y observar en UI.
  - Construir elementos visuales para mostrar la información.

App a construir. (Registro de consumos)

- Vamos a construir una app que registrara el consumo de un grupo de personas cuando están en un bar.
- Cada vez que se pidan nuevos Ítem, se irán anotando y guardando los consumos.
- Deberá tener mínimo dos pantallas (Fragmentos), Uno para ingresar los items y otro para mostrar los anteriormente ingresados.
- El Mínimo producto viable será parecido a las pantallas de al lado.
- Debe ser posible eliminar todos los registros ingresados
- Tiene libertad para mejorar la app una vez cumplidos los Hitos del MPV.

Hito 1

En primera instancia deberá construir el Modelo de la App.

- Los datos mínimos a recibir son:
  - Nombre del Item
  - Precio unitario
  - Cantidad
- Construya la entidad o entidades necesarias, dao’s necesarios y cliente base de datos.
- En base a los datos ingresados debe tener una clase o método para realizar el cálculo de cada ingreso.
- Cree una clase repositorio para exponer los datos necesarios o recibirlos.

Hito 2

En este Hito deberá construir el o los ViewModels necesarios para administrar los datos provenientes y hacia la UI.

- La app deberá recibir datos desde un fragmento.
- Deberá mostrar un listado de los elementos guardados en otro fragmento.
- Utilizar LiveData para exponer los datos hacia las vistas.
- Puede utilizar MutableLiveData según corresponda.

Hito 3

Unir las vistas con el resto de las capas de arquitectura.

- Crear adaptadores y recyclerView según corresponda.
- Indicar a las vista su ViewModel
- Comprobar métodos de borrar y guardar. asignados a botones.
- Comprobar el funcionamiento de la app.

Opcionales:

Realizar test unitario a la lógica de cálculo de ítems o modelo.