



<title> Objetivos del día </title>

- Repaso de conceptos MVVM
- Retrofit con Coroutines
- Revisar concepto de LiveData.
- Revisar concepto de ViewModel.
- Ejercicio de consolidación (Retrofit + Room + mvvm + RV)



LiveData



LiveData

En resumen, se trata de un objeto que contiene datos y tiene la capacidad de poder ser observado.

Lo más importante de LiveData y algo que lo distingue de otros tipos de datos observables, es que se adapta al uso y al ciclo de vida de android. Un objeto LiveData solo notificará a los elementos que estén suscritos si es que están activos(activos en un ciclo de vida).

- **Perfecta sincronización con el ciclo de vida de la UI.**
- **Adiós a los Memory Leaks (se libera cuando debe).**
- **Adiós al control manual del ciclo de vida.**
- **Datos actualizados en todo momento.**
- **No se ven afectados por los cambios de configuración.**

Crear un objeto LiveData

En los ejemplos de esta semana hemos creado objetos **LiveData** que provienen directamente desde Room (base de datos) y también desde un servicio web, bastó con utilizar la referencia y ya tenemos un LiveData para observar.

Cuando utilizamos Retrofit, tuvimos que generar un nuevo objeto LiveData y luego entregarle la información, La clase que utilizamos fue **MutableLiveData**.

```
private LiveData<List<Todo>> listLiveData;  
listLiveData = todoDao.getallTodoFromDB();
```

```
.....
```

```
private MutableLiveData<List<MarsObject>>  
listMutableLiveData = new MutableLiveData<>();
```

```
listMutableLiveData.setValue(marsObjectList);
```

MutableLiveData

LiveData es un objeto inmutable, por lo que no podemos directamente cambiar el contenido de estos datos. Cuando necesitamos crear este tipo de objetos Utilizaremos **MutableLiveData**, la cual contiene métodos que nos permitirán añadir información.

- `setValue(T value)` Con este método, podemos añadir datos a un **MutableLivedata**, los datos se añaden de forma sincrónica en el main Thread.
- `postValue(T value)` Con este método también agregamos añadimos datos, pero lo hacemos de forma asíncrona y no en el main Thread.

Ejemplo LiveData

LiveData es un wrapper(envoltorio) que se puede usar con cualquier dato, incluidos los objetos que implementan **Collections**, cómo **List**.

Por lo general, un objeto **LiveData** se almacena dentro de un objeto **ViewModel**, y se puede acceder a este a través de un método get, como se muestra en el siguiente ejemplo:

```
fun exposeLiveDataFromDatabase():  
    LiveData<List<Terrain>> {  
        return repository.mLiveData  
    }
```

LiveData recomendaciones

Asegúrate de almacenar objetos **LiveData** que actualicen la IU en objetos **ViewModel**, en lugar de una actividad o un fragmento, por los siguientes motivos:

- Para evitar las actividades y fragmentos demasiado cargados (ahora, estos controladores de IU son responsables de mostrar los datos, pero no retienen el estado de los datos)
- Para desacoplar las instancias **LiveData** de instancias de fragmentos o actividades específicas, y permitir que los objetos **LiveData** sobrevivan a los cambios de configuración.

Observando LiveData

En la mayoría de los casos, el método `onCreateView()` o `onViewCreated()` de un componente de la aplicación es el lugar adecuado para comenzar a observar un objeto `LivData`, por los siguientes motivos:

- Para garantizar que el sistema no realice llamadas redundantes desde el método `onResume()` de una actividad o un fragmento.
- Para garantizar que la actividad o el fragmento tenga datos que pueda mostrar tan pronto como quede activo.

```
//3. Observo la funcion que retornara LiveData desde el  
viewModel  
mViewModel.exposeLiveDataFromDatabase().observe(vi  
ewLifecycleOwner, Observer {  
    Log.d("VIEW", it.toString())  
    adapter.updateAdapter(it)  
})  
.....
```

ViewModel

ViewModel

Se diseñó la clase `ViewModel` a fin de almacenar y administrar datos relacionados con la IU de manera optimizada para los ciclos de vida. La clase `ViewModel` permite que los datos sobrevivan a cambios de configuración, como las rotaciones de pantallas.

```
// ViewModel
implementation "androidx.lifecycle:lifecycle-
viewmodel:2.2.0"
// LiveData
implementation "androidx.lifecycle:lifecycle-
livedata:2.2.0"
```

Ejemplo ViewModel

El ejemplo más simple de viewModel extendiendo de viewModel, no es necesario utilizar un constructor.

Si necesitamos la referencia por ejemplo de un objeto application, podemos extender desde AndroidViewModel.

```
class MarsViewModel(application: Application) :  
    AndroidViewModel(application){  
  
    // una variable referencia al Repositorio  
    private var repository : Repository  
    init {  
        val terrainDao =  
            MarsDataBase.getDatabase(application).getTerrainDao()  
        repository = Repository(terrainDao)  
        repository.getDataFromServer()  
    }  
  
    fun exposeLiveDataFromDatabase(): LiveData<List<Terrain>>  
    {  
        return repository.mLiveData  
    }  
}
```

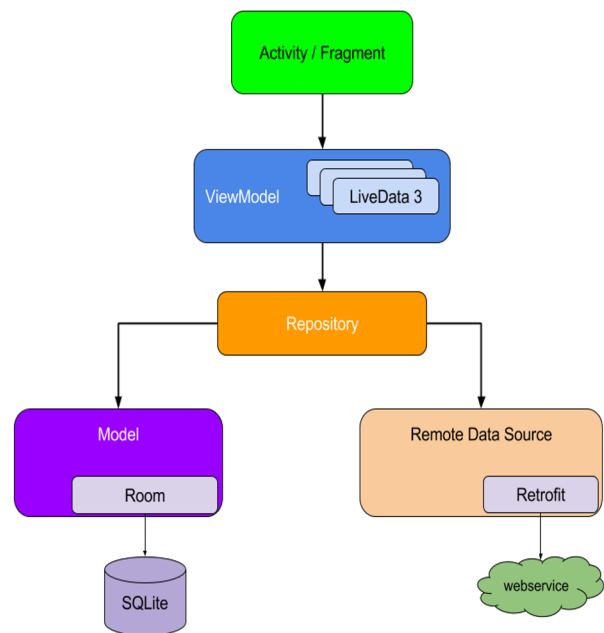
Ejemplo ViewModel

Asociación de un ViewModel en un fragmento.

```
class FirstFragment : Fragment(),  
MarsAdapter.CallbackInterface {  
    //1- declaro variable del viewmodel  
    lateinit var mViewModel: MarsViewModel  
    lateinit var adapter: MarsAdapter  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        //2- Asigno el viewmodel a esta vista  
        mViewModel =  
        ViewModelProvider(this).get(MarsViewModel::class.java)  
        adapter = MarsAdapter(this)
```

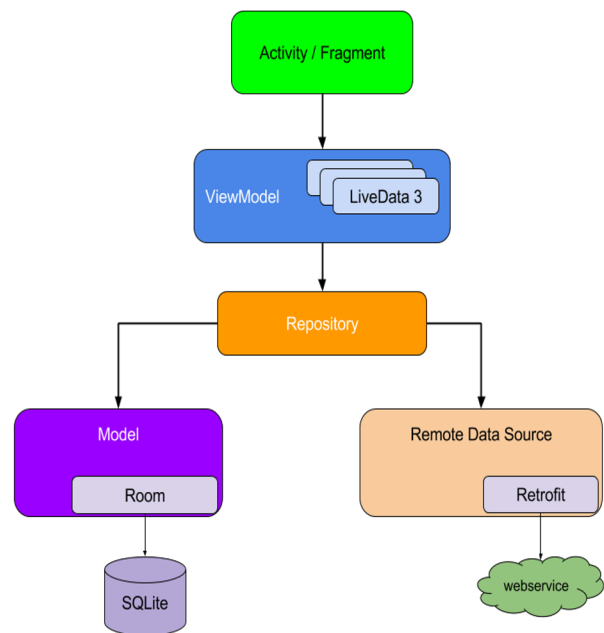
Ejercicio de consolidación (Requerimientos generales)

1. El proyecto debe estar en un repositorio en la plataforma GitHub.
2. Debe construir el proyecto con base en los requerimientos específicos solicitados, pero puede añadir su capa de customización.
3. Está considerado construirlo en horas de clases, para responder las consultas, pero si trabaja fuera del horario no existe problema.
4. En cada clase debe subir un commit y push hasta donde haya avanzado.



Ejercicio de consolidación (Requerimientos generales)

1. Debe utilizar algun patron de diseño recomendado (MVP, MVVM).
2. Debe utilizar una clase Repositorio para tener una única fuente de datos.
3. Debe implementar persistencia de datos en una base de datos utilizando la biblioteca Room.
4. Debe conectarse a un servicio web y consumir datos, se recomienda un servicio específico.
5. Debe mostrar al menos tres pantallas, un RecyclerView con un listado de elementos.



Ejercicio de consolidación (Requerimientos generales)

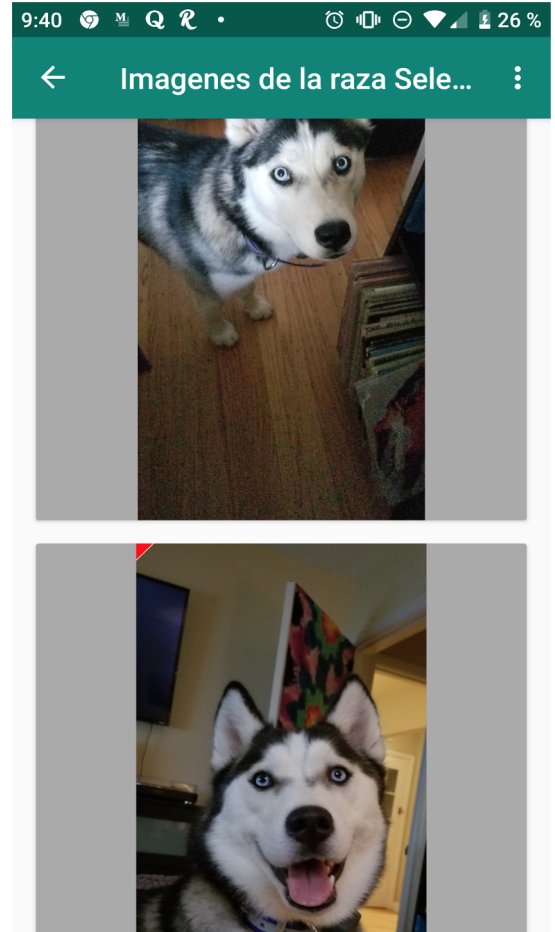
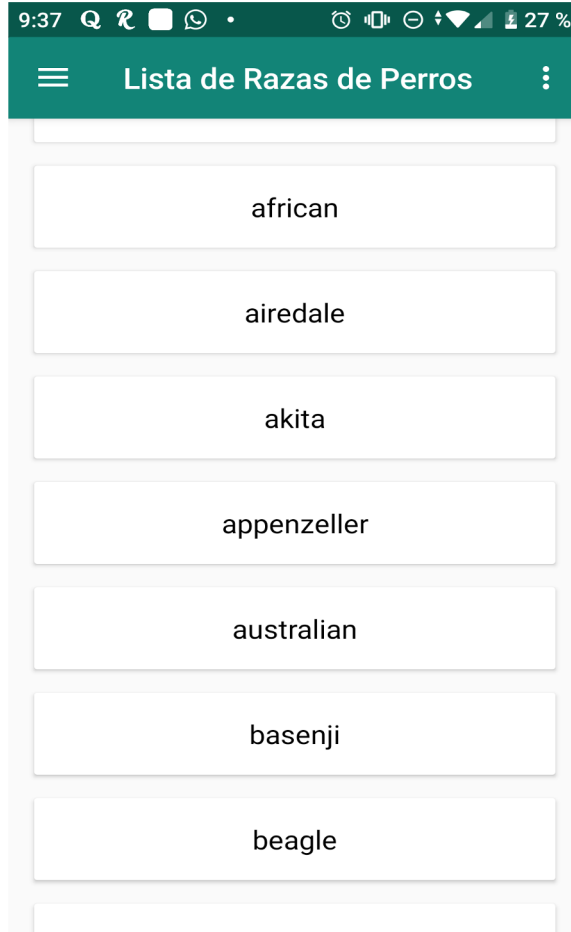
6. Al hacer click sobre un elemento, debe mostrar un nuevo Fragmento con otro RecyclerView mostrando imágenes.
7. Al hacer un onLongClick sobre un elemento, debe guardar el objeto en la persistencia de datos como favoritos.
8. Debe haber un fragmento que muestre los objetos favoritos seleccionados en un RecyclerView. (Implementar un fab o menu).
9. En el fragmento de favoritos, debe ser posible eliminar un elemento o todos los elementos.

Ejemplo:

Ejemplo de App:

Web server :

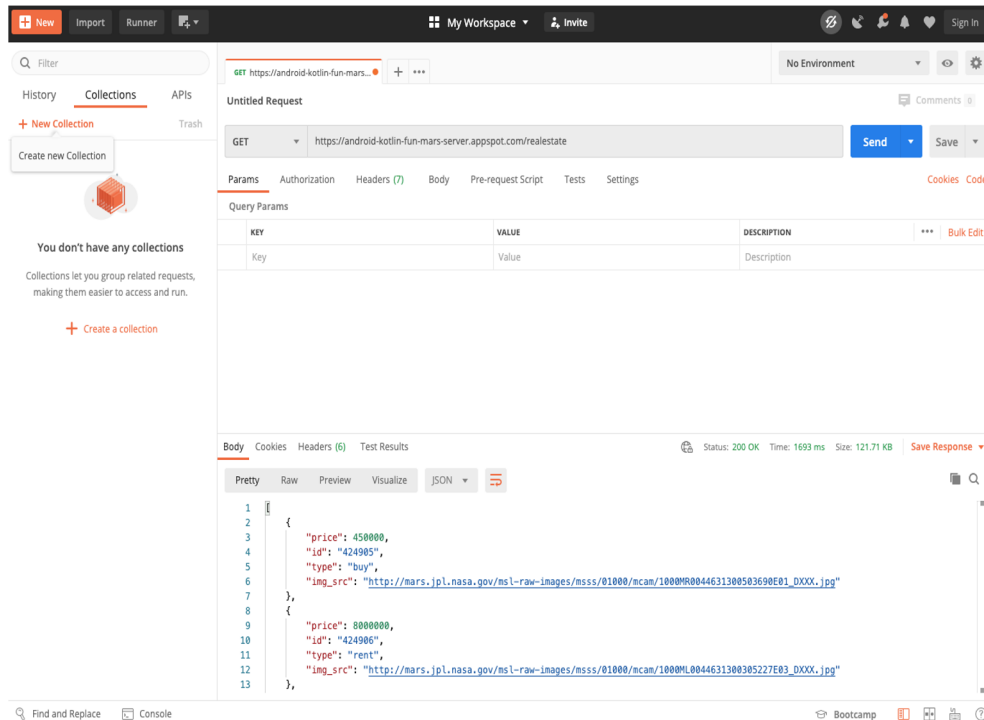
<https://dog.ceo/>



Recuerda subir tu avance a GitHub.

TIP: Analizar la respuesta de la API

1. Utiliza Postman para analizar los datos que recibiremos desde la API.
2. Utiliza el Plugin o convertirlo a mano



The screenshot displays the Postman application interface. On the left sidebar, the 'Collections' tab is active, showing a message: 'You don't have any collections. Collections let you group related requests, making them easier to access and run. + Create a collection'. The main workspace shows an 'Untitled Request' with the following details:

- Method: GET
- URL: `https://android-kotlin-fun-mars-server.appspot.com/realestate`
- Environment: No Environment

The 'Query Params' section is empty. The 'Body' tab is selected, showing a JSON response with a status of 200 OK, time of 1693 ms, and size of 121.71 KB. The response is formatted as JSON and contains two items:

```
1 {
2   {
3     "price": 450000,
4     "id": "424905",
5     "type": "buy",
6     "img_src": "http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631300503690E01_0XXX.jpg"
7   },
8   {
9     "price": 8000000,
10    "id": "424906",
11    "type": "rent",
12    "img_src": "http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631300385227E03_0XXX.jpg"
13  },
14 }
```