



viu

**Universidad
Internacional
de Valencia**

Aplicación Android hecha en Kotlin con inyección de dependencias, acceso a internet y guardado de datos

Titulación:

Máster Universitario en Desarrollo de
Aplicaciones y Servicios Web

Curso académico: 2022-2023

Alumno/a: Gómez Olivencia, Rubén

D.N.I.: 78910013-A

Asignatura: Programación en
dispositivos móviles (wearables)

Índice general

1. Introducción	3
2. Requisitos del proyecto	3
3. Creación del proyecto	4
3.1. Inyección de dependencias	4
3.2. Conexión a internet y obtención de datos	5
3.3. Data Store y Protocol Buffers	6
3.3.1. Estructurar los datos	7
3.3.2. Guardar, recuperar y borrar los datos	8
3.4. Visualizar datos con RecyclerView	9
4. Conclusiones	11

1. Introducción

A la hora de crear una aplicación de móvil moderna podemos utilizar como fuente de inspiración cualquiera de las que utilizamos en nuestro día a día.

Si realizamos un pequeño análisis de las mismas, veremos que todas ellas hacen uso de la conectividad de red para obtener (o enviar) datos a internet, y que parte de esos datos serán almacenados en una base de datos interna en el propio móvil.

En esta documentación se va a explicar los pasos realizados para la creación de una aplicación que funcionará en el sistema operativo Android y creada bajo el lenguaje de programación Kotlin. Se hará uso de un sistema de inyección de dependencias, acceso a internet, gestión de una base de datos propia y la visualización de datos se hará de manera eficiente.

2. Requisitos del proyecto

Como todo proyecto, debemos ceñirnos a unos requisitos que asegurarán que el resultado final cumple con las expectativas esperadas. En este caso, el objetivo es profundizar en ciertas características que cualquier aplicación moderna para Android cumple a día de hoy.

Los requisitos establecidos son:

- Obtener un listado de películas de un servicio HTTPS, que recibirá un fichero JSON. Para ello se utilizará la librería [retrofit](#).
- Se creará un modelo de datos para el correcto *parseo* de los datos recibidos.
- Estos datos se guardarán en una base de datos interna en el móvil utilizando [Protocol Buffers](#) y [DataStore](#). También se podrán eliminar las películas de la base de datos.
- La visualización de los datos se hará mediante el componente [RecyclerView](#). Una manera eficiente de visualizar datos en un listado.
- Al seleccionar una película del listado, nos enviará a otra pantalla para ver

más información de la misma.

- Se debe hacer uso de un sistema de inyección de dependencias, como por ejemplo [Koin](#).

3. Creación del proyecto

Para la creación del proyecto se ha hecho uso del entorno de desarrollo [Android Studio](#), que es el IDE oficial que nos permite programar teniendo todos los recursos necesarios.

Por otro lado, se ha hecho uso del lenguaje de programación [Kotlin](#), ya que desde el año 2019 Google lo ha puesto como el lenguaje de programación preferido a la hora de realizar los desarrollos para la plataforma Android.

A continuación se detallan los aspectos más destacados del proyecto.

3.1. Inyección de dependencias

A la hora de crear un proyecto es posible que utilicemos distintos componentes o clases que se refieran unos a otros. Para facilitar que estas referencias se cumplan se hará uso de la inyección de dependencias utilizando la librería [Koin](#).

</> Inicialización de Koin en App.kt

```
startKoin {  
    androidLogger(Level.ERROR)  
    androidContext(this@App)  
    modules(  
        module { single { moviesDataStore } },  
        mainModule,  
        mainActivity  
    )  
}
```

Aunque no es obligatorio el uso de librerías externas para la inyección de dependencias, Koin nos va a ayudar asegurando que lo realizamos de forma correcta y de esta manera evitaremos bugs de implementación.

3.2. Conexión a internet y obtención de datos

La gran mayoría de aplicaciones que utilizamos hoy en día hacen uso de conexión a internet, por lo que es necesario entender cómo funciona este sistema dentro del ecosistema Android.

En primer lugar la aplicación debe de contar con los permisos necesarios para poder acceder a internet. Para ellos se deben especificar los permisos en el fichero  **AndroidManifest.xml**

</> Permisos de red en **AndroidManifest.xml**

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Con los permisos configurados es momento de saber cómo realizar la conexión para obtener ficheros y/o información. En nuestro caso se va a utilizar la librería [retrofit](#) para la descarga de un fichero [json](#) que contiene un listado de películas.

La configuración cuenta con varias partes:

- Construcción de la instancia del servidor, en la que indicamos cuál es el servidor al que vamos a conectarnos

</> Instancia de Retrofit

```
Retrofit.Builder()
    .baseUrl("https://gist.githubusercontent.com/")
    .addConverterFactory(
        MoshiConverterFactory.create(
            Moshi.Builder()
                .add(KotlinJsonAdapterFactory())
                .build()
        )
    )
    .build()
    .create(MoviesInterface::class.java)
```

- Interfaz de conexión, en la que se indica el resto de la URL a la que queremos acceder.

🔗 Interfaz Retrofit en MoviesInterface.kt

```
import retrofit2.http.GET
interface MoviesInterface {
    // URL cortada
    @GET("skydoves/.../DisneyPosters2.json")
    suspend fun downloadMovies():
        List<com.rugoli.moviedb.dataclass.Movie>
}
```

- Junto con el código anterior, se indica que el resultado obtenido se va a parsear para obtener una lista de objetos “Movie”, que está especificado dentro del fichero  **dataclass/Movie.kt**. En este fichero se indica cómo es la clase y los distintos atributos que contiene.
- Ya sólo queda realizar la llamada para obtener los datos, que en este caso se hace desde el modelo de datos, que posteriormente se explicará con más detalle.

Con lo explicado la aplicación es capaz de realizar la descarga del fichero, parsear el json obtenido y crear un listado de objetos de una clase propia. De esta manera los datos se podrán utilizar posteriormente.

3.3. Data Store y Protocol Buffers

Una vez sabemos cómo realizar la descarga de datos, el siguiente paso es almacenarlos.

Es importante entender que **Data Store** se puede utilizar con datos estructurados y con datos en formato “clave-valor”. El primer caso es habitual utilizarlo como si fuese una base de datos SQL, mientras que el uso de “clave-valor” es más habitual a la hora de guardar configuración de la aplicación.

En nuestro caso, que vamos a utilizar datos estructurados, vamos a diferenciar el desarrollo en dos apartados:

- Estructurar los datos
- Guardar y recuperar los datos

3.3.1. Estructurar los datos

En el caso que nos ocupa, se va a crear una estructura de datos que forma la película (con los datos descargados), para posteriormente almacenarlos. Para realizar esta estructura se va a utilizar [Protocol Buffers](#).

La configuración está dividida en distintos apartados que detallaremos a continuación:

- Hay que indicar que se va a utilizar Protocol Buffers, para ello se configura en el fichero  `build.gradle`
- La **estructura de los datos** está dentro del fichero de configuración  `proto/movie.proto`. Este fichero sería el equivalente a un “schema” de creación de una tabla en MySQL.

Fichero `movie.proto`

```
syntax = "proto3";
option java_package = "com.rugoli.moviedb";
option java_multiple_files = true;
message MovieStore {
    repeated StoredMovie movies = 1;
}
message StoredMovie{
    int32 id = 1;
    string name = 2;
    int32 release = 3;
    string playtime = 4;
    string description = 5;
    string plot = 6;
    string posterUrl = 7;
    string gifUrl = 8;
}
```

- Creación de un **Serializer** que le indica a Datastore cómo leer y escribir el tipo de datos. Está creado en el fichero  [MovieStoreSerializer.kt](#)

3.3.2. Guardar, recuperar y borrar los datos

Antes de comenzar a guardar y recuperar datos, en la aplicación debemos indicar que vamos a hacer uso de DataStore y su configuración (indicando el fichero que debe crear y el *Serializer* a utilizar)

Creamos variable global para Datastore en [App.kt](#)

```
private val moviesDataStore: DataStore<MovieStore> by dataStore(  
    fileName = "movies.pb",  
    serializer = MovieStoreSerializer  
)
```

Tras esto, hay que decidir dónde se va a hacer uso de los datos, y para ello se ha creado el modelo de datos **Movie** dentro del fichero  [models/Movie.kt](#). Este modelo de datos se encarga de:

- **Guardar los datos** en el DataStore. Si la base de datos está vacía, descargará los datos de internet para después guardarlos. Se ha decidido que **en caso de borrar todas las películas del listado, se volverán a descargar**.

Parte del fichero [models/Movie.kt](#)

```
if (movieStore.moviesCount == 0) {  
    downloadMovies()  
}  
// ...  
// store downloaded movies  
moviesDataStore.updateData { movieStore ->  
    movieStore.toBuilder()  
        .addAllMovies(moviesToStore)  
        .build()  
}
```

- **Recuperar los datos** del DataStore al iniciar la aplicación. De esta manera se creará el RecyclerView que veremos posteriormente.
- **Borrar de datos** al pulsar el icono de borrado/papelera que se encuentra junto a una película.

3.4. Visualizar datos con RecyclerView

A la hora de visualizar un listado de elementos (conformados por datos), lo ideal es hacerlo con un sistema dinámico como RecyclerView, que está preparado para mostrar gran cantidad de elementos.

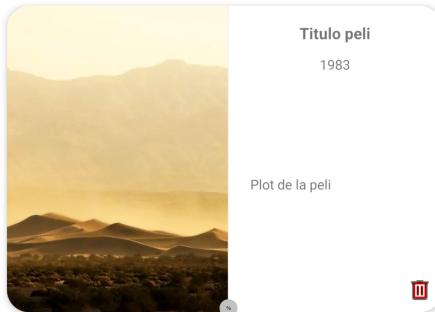
Este sistema, como su propio nombre indica, se va a encargar de reciclar los elementos que se deben mostrar, actualizando los datos de manera dinámica por datos nuevos antes de que se vuelvan a mostrar por pantalla.

De esta manera, en lugar de estar creando y destruyendo elementos de memoria, lo que se hará es cargar unos pocos elementos (los necesarios para ser mostrados por pantalla) que serán reutilizados, mejorando el uso de memoria y el rendimiento general.

A la hora de configurar el RecyclerView debemos especificar:

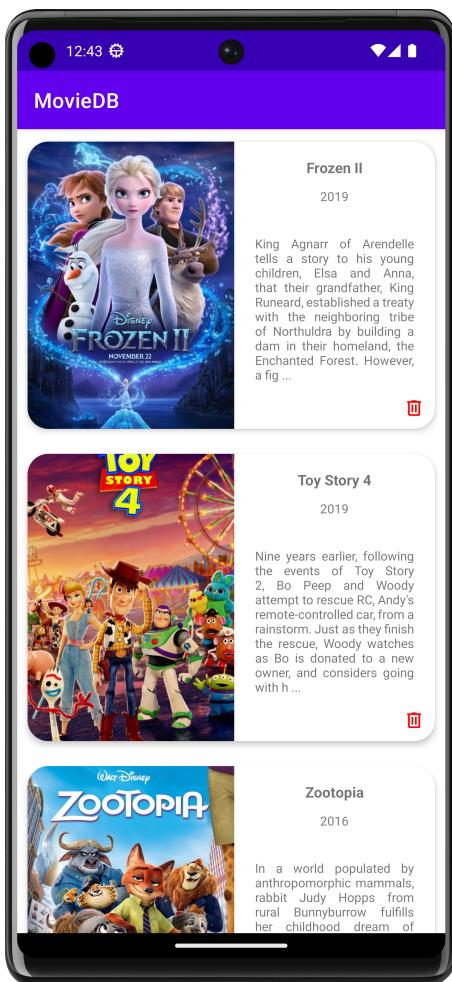
- El propio RecyclerView, que hemos añadido en la vista principal **activity_main.xml**.
- El **Adapter** que hemos creado en  **MovieAdapter.kt**, que se encarga de enlazar los datos que queremos visualizar con la vista correspondiente.

Para visualizar los datos de cada película, se ha creado un *layout* llamado **movie_item.xml**, que es de tipo *CardView* y que tiene el siguiente aspecto (con los datos de prueba en el diseñador de Android Studio):



Esta es la vista **movie_item.xml** que en el **MovieAdapter** se le indica al RecyclerView que debe de utilizar. De esta manera, RecyclerView creará los elementos necesarios que contendrán los datos de cada película y que mostrará por pantalla.

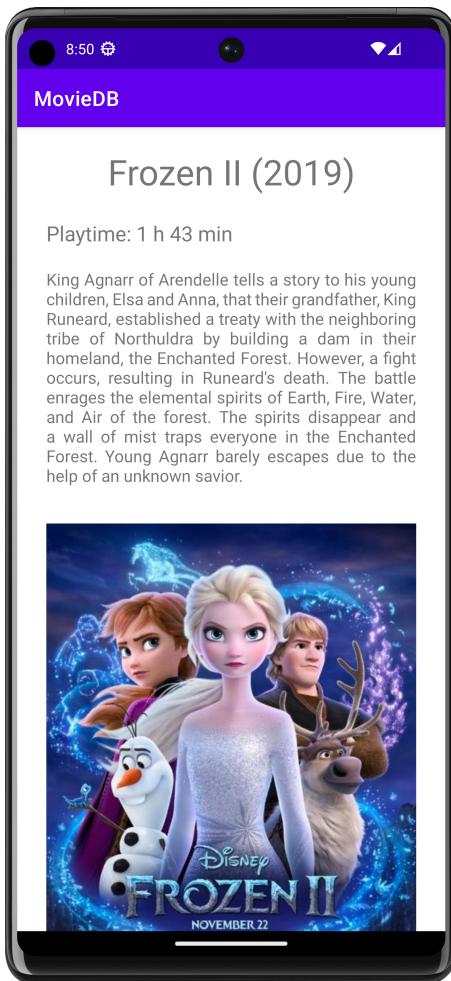
El resultado visual final será el siguiente:



Tal como se puede ver en la imagen, cada elemento de la lista tiene el mismo

aspecto visual, pero con los datos distintos, que son los obtenidos de la base de datos DataStore.

Al seleccionar un elemento de la lista, la aplicación nos enviará a otra pantalla donde veremos los datos completos de la película.



4. Conclusiones

A la hora de crear una aplicación para móvil es vital conocer las distintas características que nos proporciona la plataforma y el lenguaje de programación que estemos utilizando. De esta manera no sólo programaremos mejor, si no que como consecuencia conseguiremos una mejor experiencia de usuario.

También es importante conocer y hacer uso de librerías externas, que nos proporcionarán nuevas opciones que quizás la plataforma no nos ofrece, o

“simplemente” nos facilitará el trabajo a la hora de utilizar características complejas.