

Tutorial sobre como conectarse a la API Google Books

(por Leo 🐶)

* Se aceptan sugerencias y correcciones

Como en las últimas clases del módulo 4 del curso Desarrollo de aplicaciones móviles Android Trainee / BOTIC-SOFOF-20-14-13-0016 impartido por AIEP nos dejaron a todos los alumnos abandonados a nuestra suerte (esto después de que el profesor anterior renunció a su cargo) y más encima con un ejercicio hasta el momento imposible de realizar gracias a los vacíos en el aprendizaje generados por una mala enseñanza y coordinación del curso, es que en el presente tutorial voy a intentar explicar todo lo que pueda sobre el ejercicio y avanzar de manera pausada y explicando paso a paso lo que alcance a hacer.

Agradecimientos a Ignacio Cavallo (su [Github](#)) que me ayudó mucho a aclarar dudas.



ADVERTENCIA:

Yo no soy y estoy lejos de ser un experto en la materia, por lo que acá esta presentado es parte mi propio proceso de aprendizaje y esfuerzo, si alguien intenta seguirlo, bienvenid@, si no le resulta, podemos comentarlo e intentar buscar soluciones, pero si algo falla o se les hecha a perder algo que ya tenían, es 100% de su responsabilidad (nadie los manda a seguir a un tipo con una advertencia entre bandas amarillas-negras al inicio de su tutorial o no?)



Bueno, para comenzar que se va a intentara acá es dar solución al siguiente ejercicio:

Ejercicio

Se requiere desarrollar una aplicación móvil para Android en el lenguaje Kotlin o Java, esta aplicación móvil debe realizar las siguientes acciones:

- La aplicación se conectará a un api rest del cual obtendrá un listado de libros mostrando su título, autor y una miniatura de su portada.
- La aplicación permitirá seleccionar libros y mostrar su detalle.
- La aplicación permitirá ver los libros favoritos sin conexión.
- La aplicación contará con un botón que permite enviar un correo electrónico utilizando alguna aplicación de correo.

Agregar 4 nuevas funcionalidades a la aplicación.
Elaborar diagramas de casos de uso y detallar requerimientos funcionales.

Partiendo con la desventaja de que nada de lo que vimos para hacer esto fue enseñado, el punto de partida inicial a mi parecer es...

¿Que es una API?

Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones (Application Programming Interface). Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados.

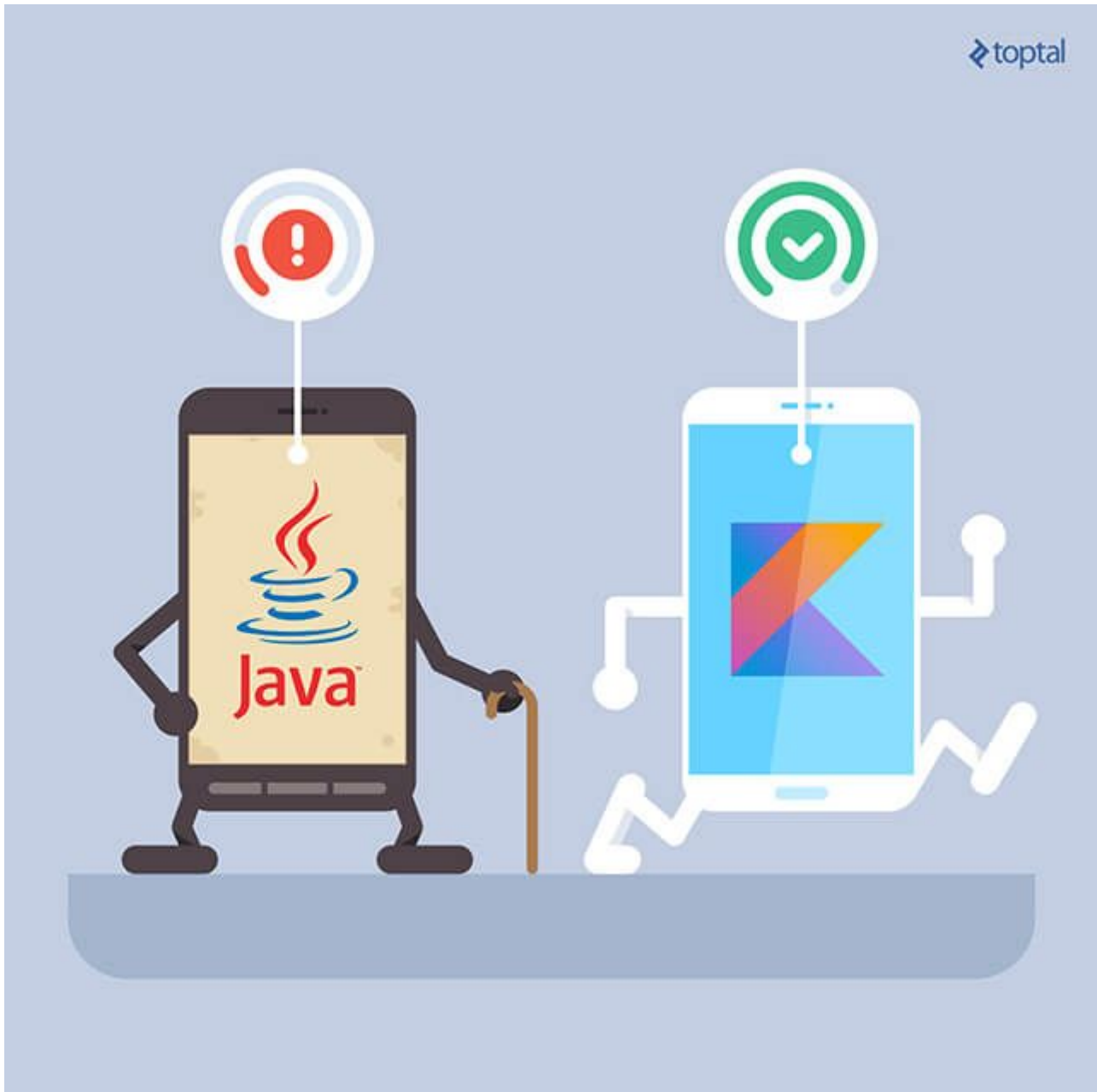
Fuente: [RedHat](#)

Para explicarlo simple y con una analogía (que escuche en algun video que no recuerdo mientras investigaba todo esto). Imaginate que quieres hacer una aplicación tipo "Uber", para eso tienes 2 opciones:

- 1.- Gastar un "chilión" de dolares en una empresa que registre y realice un mapa de todas las ciudades objetivos y más e inventar un sistema de geolocalización eficiente que permita rastrear los autos y al cliente... lo cual suena poco factible, en terminos de dinero y tiempo.
- 2.- Pedirle al tio Google que te preste sus mapas y sistema de geolocalizacion... en el momento que escojes esta, escojes usar la API de Google Maps y por tanto, usar una API

Para el ejercicio acá presentado, intentaremos usar la API Google Books ([link](#)), la cual permite acceder a un listado de miles de libros (con sus datos, imagenes, preview y mas) de manera gratuita, lo cual se nos pide al inicio del ejercicio.

Preparativos iniciales



Ojo → Voy a intentar ir paso a paso y siguiendo el orden en que se realizaron las cosas, por eso dependiente van a existir saltos de una clase a otra o dejando cosas a medio concluir, que se finalizaran después al ir siguiendo el tutorial (dicho de otro modo, parece medio desordenado, pero así se hace esto, no es mi culpa (-_-))

Preparativos iniciales:

- a) Iniciamos abriendo Android Studio y creando un proyecto en blanco desde 0 (Empty activity, lenguaje Kotlin, SDK mínimo 23 en mi caso y con nombre de proyecto lo que se te ocurra, yo no juzgo los nombres, así que ponle algo que te recuerdes después).
- b) Ir al Manifest y colocar los permisos de Internet en las líneas 5 y 6 (las API requieren conexión a internet para traer los datos, si no es así, no se puede usar)

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

c) Luego de esto ir al Módulo Build.Gradle del proyecto y añadir las dependencias que se van a usar para trabajar la API, además de configurar el método View Binding y los plugins de Kotlin que pudieran faltar. Para esto, vamos a añadir lo siguiente:

- línea 4 → id

```
'kotlin-android-extensions'
```

- línea 34 a 36 →

```
buildFeatures{  
    viewBinding = true  
}
```

- línea 46 a 49 →

```
implementation "com.squareup.retrofit2:retrofit:2.9.0"  
implementation "com.squareup.retrofit2:converter-gson:2.9.0"  
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.6'  
implementation 'com.github.bumptech.glide:glide:4.12.0'
```

La utilidad de estas librerías es la siguiente:

- **Retrofit** = cliente HTTP seguro para Java y Kotlin que permite consumir los datos de las APIs
- **Converter Gson** = Convertidor que permite pasar los datos de archivos Json de la API a algo más legible por la app
- **Kotlinx Coroutines** = para implementar corrutinas y ejecuciones en segundo hilo que no ralenticen la app
- **Glide** = transformar las Url de las imágenes de los libros en imágenes reales que se puedan ver, no solo una String

Después de poner todo lo anterior el Módulo del Build.Gradle debería de quedar algo así:

```

1  plugins {
2      id 'com.android.application'
3      id 'kotlin-android'
4      id 'kotlin-android-extensions'
5  }
6
7  android {
8      compileSdkVersion 30
9      buildToolsVersion "30.0.3"
10
11     defaultConfig {
12         applicationId "com.example.librosapp"
13         minSdkVersion 23
14         targetSdkVersion 30
15         versionCode 1
16         versionName "1.0"
17
18         testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
19     }
20
21     buildTypes {
22         release {
23             minifyEnabled false
24             proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
25         }
26     }
27     compileOptions {
28         sourceCompatibility JavaVersion.VERSION_1_8
29         targetCompatibility JavaVersion.VERSION_1_8
30     }
31     kotlinOptions {
32         jvmTarget = '1.8'
33     }
34     buildFeatures{
35         viewBinding = true
36     }
37 }
38
39 dependencies {
40
41     implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
42     implementation 'androidx.core:core-ktx:1.6.0'
43     implementation 'androidx.appcompat:appcompat:1.3.1'
44     implementation 'com.google.android.material:material:1.4.0'
45     implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
46     implementation "com.squareup.retrofit2:retrofit:2.9.0"
47     implementation "com.squareup.retrofit2:converter-gson:2.9.0"
48     implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.6'
49     implementation 'com.github.bumptech.glide:glide:4.12.0'
50     testImplementation 'junit:junit:4.+'
51     androidTestImplementation 'androidx.test.ext:junit:1.1.3'
52     androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
53 }

```

d) Luego, para terminar el ViewBinding en la Main activity, cambiar el método on create para dejarlo de la siguiente manera (en amarillo lo nuevo):

```

package com.example.librosapp

import ...

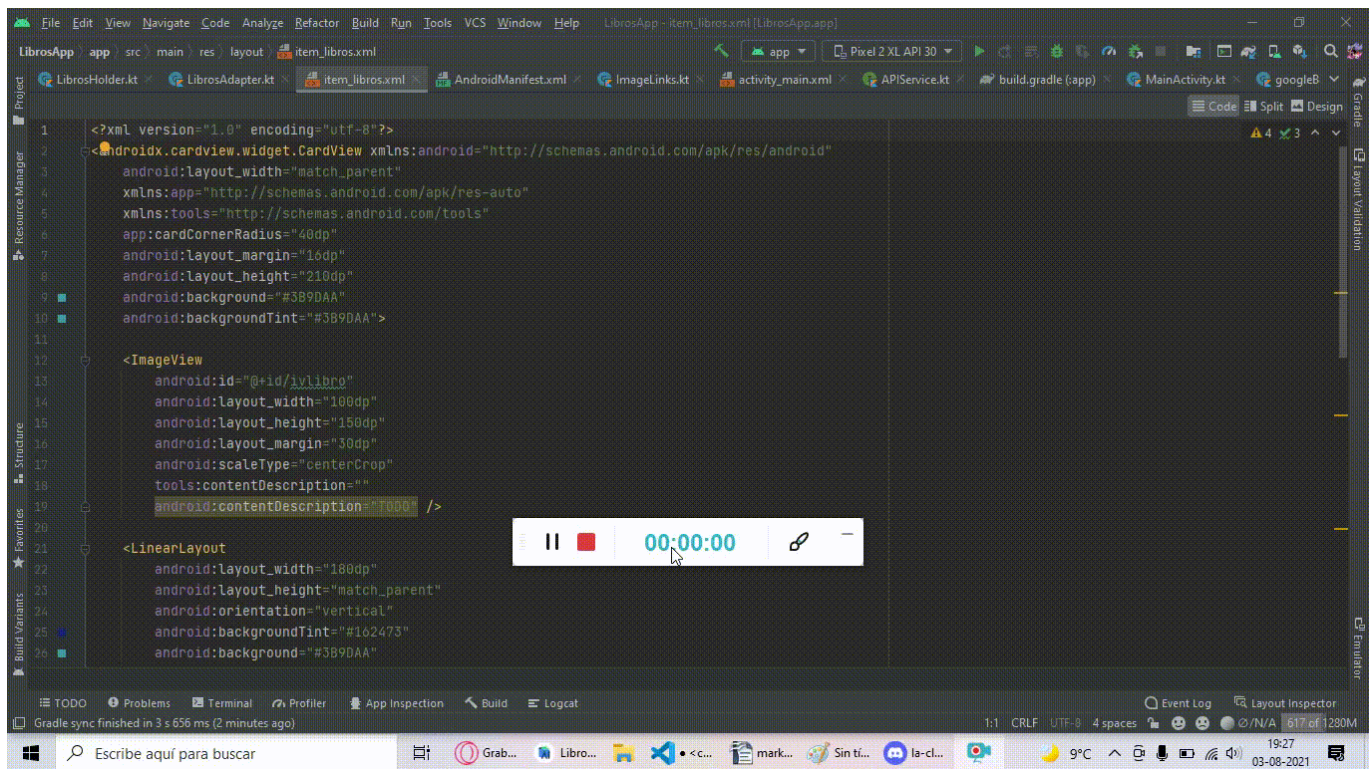
```

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding // Agregar esta variable

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding= ActivityMainBinding.inflate(layoutInflater) //Agregar esta línea
        setContentView(binding.root) //Modificar esta línea
    }
}
```

e) Finalmente, para ahorrar tiempo al crear las data class (Una data class es una clase que contiene solamente atributos que quedemos guardar, por ejemplo datos de un superhéroe. Con esto conseguimos por ejemplo no tener varias variables «nombre1», «nombre2» para almacenar todos los superhéroes con los que vamos a trabajar.) que se usan en las API es que vamos a instlar un Plugin llamado "Json To Kotlin Class" (yo lo tengo instalado de antes, pero dejo un gif que muestra como instalarlo).



Con este Plugin lo que nos permite es pasar la informacion del Json de la API a data class de Kotlin ahorrando mucho tiempo en el proceso (son varias, así que en realidad es bastante mas sencillo).

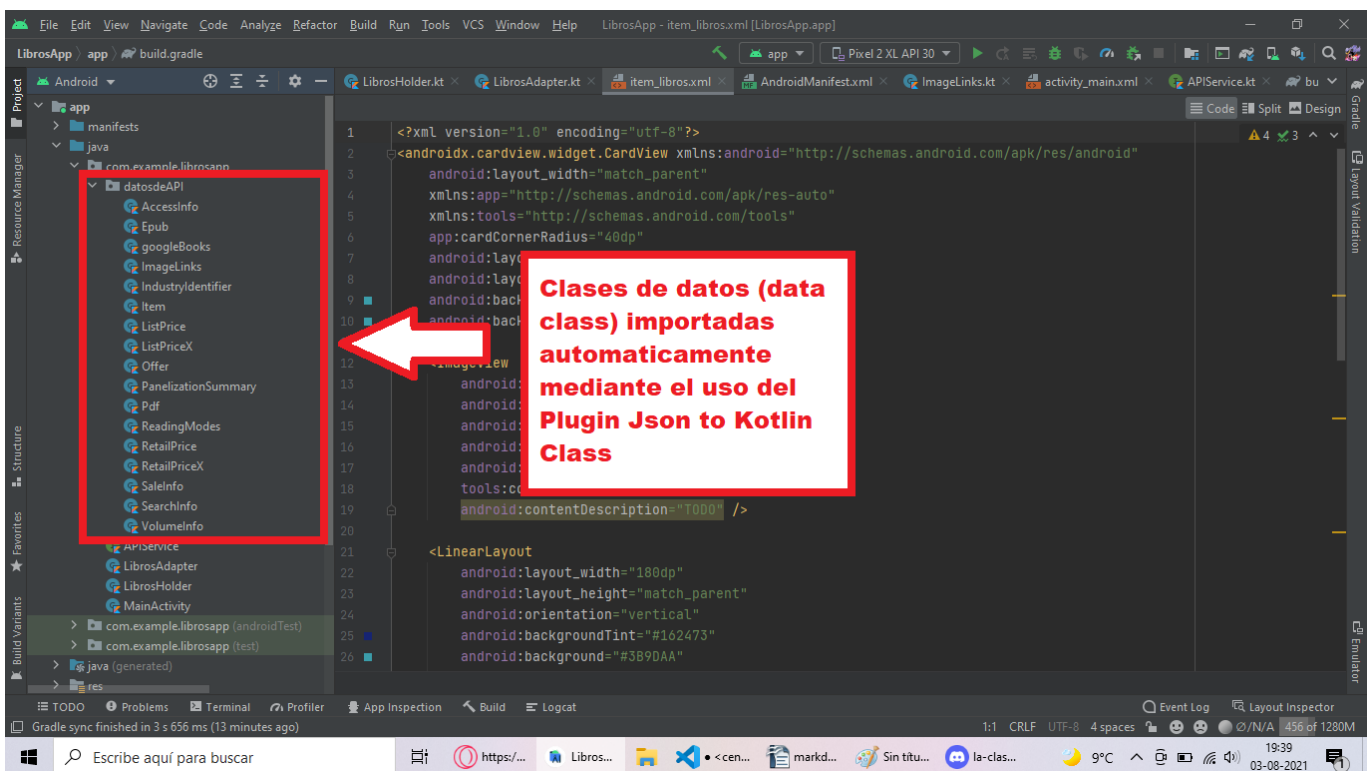
Dicho de otro modo, pasamos de esto (en la web):


```

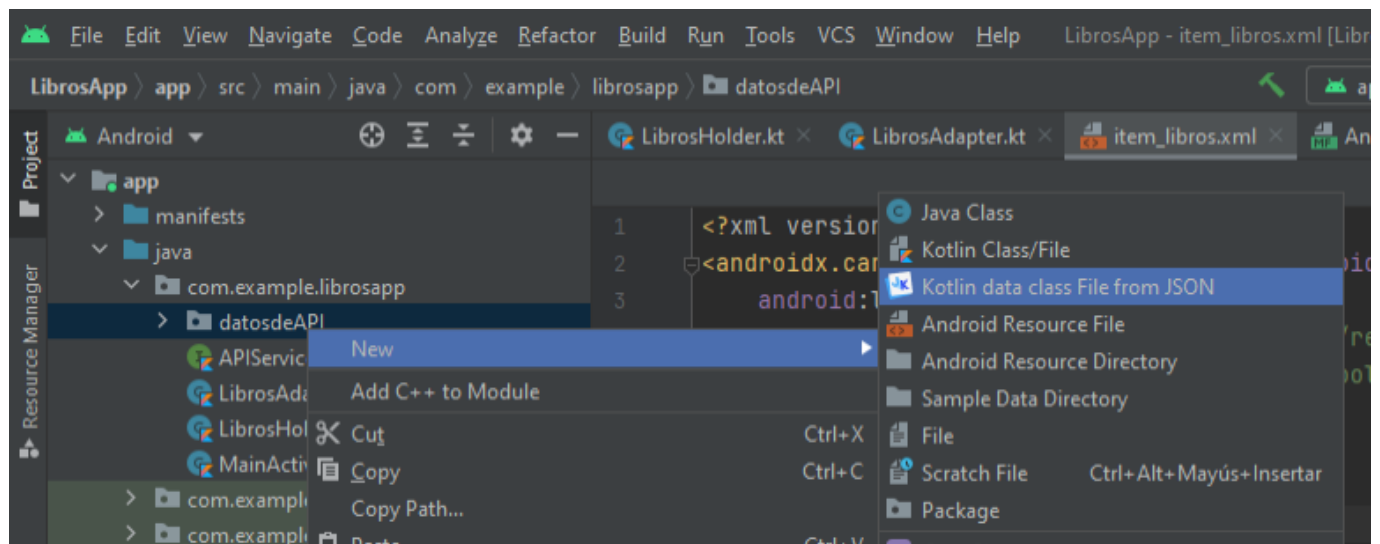
{
  "kind": "books#volumes",
  "totalItems": 900,
  "items": [
    {
      "kind": "books#volume",
      "id": "gqDf_ULmR8C",
      "etag": "YmEz13yCpAE",
      "selfLink": "https://www.googleapis.com/books/v1/volumes/gqDf_ULmR8C",
      "volumeInfo": {
        "title": "Flexible Query Answering Systems",
        "subtitle": "8th International Conference, FQAS 2009, Roskilde, Denmark, October 26-28, 2009, Proceedings",
        "authors": [
          "Troels Andreasen",
          "Ronald R. Yager",
          "Henrik Bulskov",
          "Henning Christiansen",
          "Henrik Legind Larsen"
        ],
        "publisher": "Springer Science & Business Media",
        "publishedDate": "2009-10-15",
        "description": "This volume constitutes the Proceedings of the 8th International Conference on Flexible Query Answering Systems, FQAS 2009, held in Roskilde, Denmark, October 26-28, 2009. FQAS 2009 was preceded by the 1994, 1996 and 1998 editions held in Roskilde, Denmark, the FQAS 2000 held in Warsaw, Poland, the 2002 held in Copenhagen, Denmark, and the 2004 and 2006 editions held in Lyon, France, and in Milan, Italy, respectively. FQAS is the premier conference concerned with the very important issue of providing users of information systems with flexible querying capabilities, and with easy and intuitive access to information. The main objective is to achieve more expressive, informative, cooperative, and productive systems which facilitate retrieval from information repositories such as databases, libraries, heterogeneous archives and the World-Wide Web. In targeting this objective, the conference draws on several research areas, such as information retrieval, database management, information filtering, knowledge representation, soft computing, management of multimedia information, and human-computer interaction. The conference provides a unique opportunity for researchers, developers and practitioners to explore new ideas and approaches in a multidisciplinary forum. The overall topic of the FQAS conferences is innovative query systems aimed at providing easy, flexible and human-friendly access to information. Such systems are becoming increasingly important due to the huge and always growing number of users as well as the growing amount of available information.",
        "industryIdentifiers": [
          {
            "type": "ISBN_13",
            "identifier": "9783642049569"
          },
          {
            "type": "ISBN_10",
            "identifier": "3642049567"
          }
        ],
        "readingModes": {
          "text": false,
          "image": true
        }
      }
    }
  ]
}

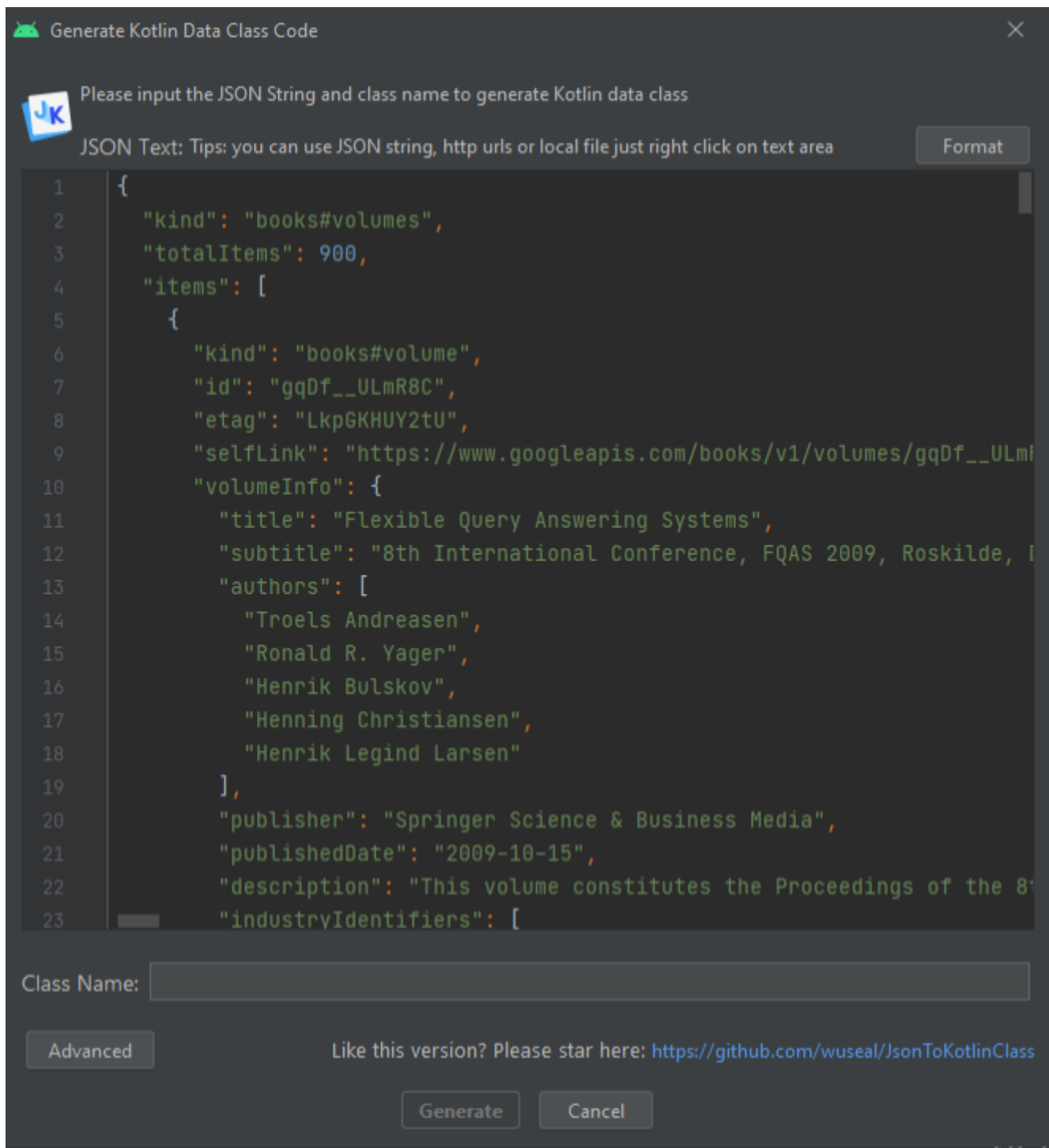
```

A esto (todos los archivos data class generados automáticamente):



Para esto lo que hay que hacer es copiar lo del siguiente [link](#) (que representa la orden de búsqueda dentro de la API), llevarlo a Android Studio y dentro de la siguiente dirección `Nombre_del_proyecto\app\src\main\java\com\example` crear un nuevo paquete (New → Package (poner el nombre que quieras)) y después sobre ese paquete (la carpeta nueva creada) dar click derecho y seleccionar la nueva opción "Kotlin data class File from JSON", para que aparezca la siguiente pantalla





Acá copiar el código copiado de la API, dar click en Class Name y darle un nombre (en mi caso googleBooks) y luego seleccionar "Generate" (notar que en el package creado van a aparecer muchos archivos, es lo normal, esas son las data class que traen los datos de la API y que nos ahorramos escribir uno a uno).

A meterle código



1. Una vez concluidos los preparativos iniciales, vamos a partir con lo que sería crear la parte visual de la MainActivity (que contendrá el RecyclerView y el Search View para las búsquedas) y otro archivo .xml llamado item_libros, que no es más que una Card View que tendrá en su interior los datos de Título, Autor y la Imagen de la portada del libro (esta Card View será lo que se va a utilizar para llenar la RecyclerView y mostrar los datos en una misma activity). Como es un tema visual, dejo mi código de ejemplo pero quien intente este tutorial, (es importante notar las id, que en su caso puede poner otras, pero yo haré referencia a estas mismas más adelante en el tutorial)

- **En MainActivity:**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="730dp"
    android:layout_width="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:background="#0AC2DC"
    android:backgroundTint="#0AC2DC"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <ImageView
        android:id="@+id/logo"
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:layout_margin="25dp"
        android:contentDescription="@string/titulo"
        android:src="@mipmap/logo"
        tools:ignore="ImageContrastCheck" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="30dp"
        android:layout_marginStart="25dp"
        android:layout_marginEnd="25dp"
        tools:text="Buscador de Libros"
        android:gravity="end"
```

```

        android:textStyle="bold"
        android:textSize="20sp"/>
<androidx.appcompat.widget.SearchView
    android:id="@+id/svlibros"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="5dp"
    android:layout_marginStart="25dp"
    android:layout_marginBottom="5dp"
    android:layout_marginEnd="25dp"
    android:background="@drawable/custom_search_background"
    android:queryBackground="@android:color/transparent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintVertical_bias="0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:queryHint="@string/buscar"
/>
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/rvlibros"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="25dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/svlibros"
    tools:listitem="@layout/item_libros" />
</LinearLayout>

```

- **En item_libros.xml:** (este archivo fue creado expandiendo la carpeta res, dando click derecho en la carpeta layout y seleccionando New → Layout Resource File, dándole el nombre item_libros y seleccionando OK).

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    app:cardCornerRadius="40dp"
    android:layout_margin="16dp"
    android:layout_height="210dp"
    android:background="#3B9DAA"
    android:backgroundTint="#3B9DAA">

    <ImageView
        android:id="@+id/ivlibro"
        android:layout_width="100dp"

```

```
        android:layout_height="150dp"
        android:layout_margin="30dp"
        android:scaleType="centerCrop"
        tools:contentDescription=""
        android:contentDescription="TODO" />

<LinearLayout
    android:layout_width="180dp"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:backgroundTint="#162473"
    android:background="#3B9DAA"
    android:layout_gravity="end">

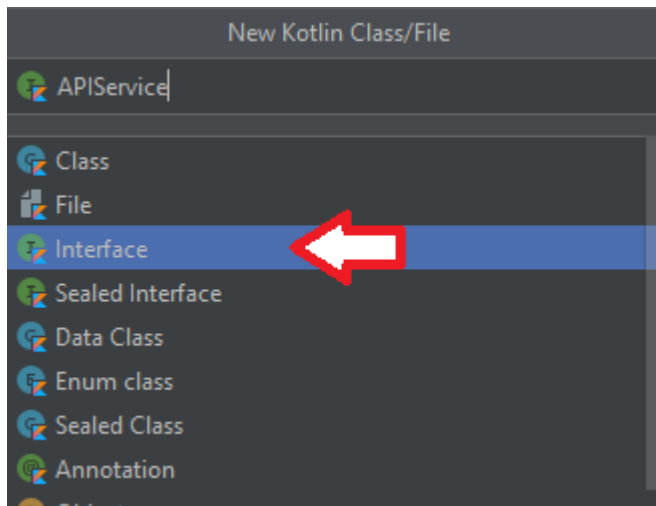
    <TextView
        android:id="@+id/tvtitulo"
        android:layout_width="200dp"
        android:layout_height="150dp"
        android:layout_gravity="end"
        android:textSize="32dp"
        android:textColor="@color/white"
        android:textStyle="bold"
        android:paddingTop="30dp"
        android:paddingBottom="30dp"
        android:gravity="center_horizontal"
        tools:text="TITULO LIBRO"
        />

    <TextView
        android:id="@+id/tvautor"
        android:layout_width="200dp"
        android:layout_height="match_parent"
        android:layout_gravity="end"
        android:textSize="20dp"
        android:textColor="@color/white"
        android:textStyle="italic"
        android:fontFamily="serif"
        android:paddingTop="15dp"
        android:gravity="center_horizontal"
        tools:text="AUTOR"
        />

</LinearLayout>

</androidx.cardview.widget.CardView>
```

2. Crear una Interfaz de Kotlin, para esto lo que se hace es en Android Studio ir a la siguiente ruta Nombre_del_proyecto\app\src\main\java\com\example y con click derecho sobre example crear una "nueva Clase/Archivo de Kotlin" (New → "Kotlin Class/File" (poner el nombre que quieras, en mi caso ApiService)) y al momento antes de crearlo, seleccionar la que tiene un circulo verde con una I (de Interface) y apretar ENTER.



La función de esta nueva Interfaz es crear el método con el que vamos a acceder al servicio de la API de Google.

Notar que acá todo lo que se importa (que son varias cosas) viene de Retrofit2, por lo que si se importa algo distinto no va a funcionar (para esto se implemento Retrofit en las dependencias del Modulo Build,Gradle en los preparativos iniciales del tutorial).

Para hacer la consulta, se va a llamar a la Url Base de la API de Google Books, que es la asiguiente dirección

```
URL base de la API --> https://www.googleapis.com/books/v1/volumes?q=search+terms
```

Y en las siguientes líneas se intentará codificar esa dirección en Kotlin para que se pueda establecer la conexión API-Aplicación

Para esto se comienza con "@GET" que es método de llamada a la API (en este caso usamos @GET porque trabajamos con una Url, pero existen otros tipo de llamadas no consideradas en este tutorial, pero que se explican brevemente [acá](#)). Este @GET recibe como parámetro "volumes" que es lo que estamos llamando en la API y que es parte de la Url base que no cambia al hacer las consultas.

Luego del @GET se crea la función buscarLibroPorTitulo, la que como va a ser trabajada en segundo plano, es decir asíncrona, se le pone al inicio el "suspend" (que es parte de las corrutinas, importada en las librerías al principio del proyecto) para que no interfiera con el hilo en primer plano. Se sigue con la función dándole una "@Query" que tendrá la "q" (que viene de la Url base e inicia la consulta) y encoded = true (encoded da el formato de enlace web, sin espacio entre palabras (lo llena con un + o ? según corresponda)), los atributos de la @Query se finalizan con lbuscado:String, que como su nombre lo indica es lo que el usuario va a buscar en el Searchiew y que será distinto en cada ocasión (como es una cadena de letras = String). Para aacabar esta parte, a la función se que va a recibir una respuesta... pero, ¿que tipo de respuesta?, una de la clase googleBooks creada antes (esta fue creada cuando usamos el Plugin Json to Kotlin Class, e incluso ay una imagen de ella. Esta junto a los archivos creados por el plugin, con ese mismo nombre).

De lo anterior, se puede ver que la Interfaz queda compuesta de la siguiente manera (mira los import, todos de retrofit2 y compara la Url base que estamos creando con una que tiene como ejemplo "lbuscado" a el

libro El hobbit):

```
package com.example.librosapp

import com.example.librosapp.datosdeAPI.googleBooks
import retrofit2.Response
import retrofit2.http.GET
import retrofit2.http.Query
import retrofit2.http.Url

interface APIService {

    @GET("volumes")
    suspend fun buscarLibroPorTitulo (@Query ("q", encoded = true)
lobuscado:String): Response<googleBooks>

    //COMPARAR URL QUE ESTOY FORMANDO CON -->
    https://www.googleapis.com/books/v1/volumes?q="el+hobbit" (el hobbit =
lobuscado:String)

}
```

Acá termina esta parte, más breve pero media "extraña" de entender en un comienzo

3. Después de crear lo de antes (las datas class, los .xml, la interface APIService y lo del archivo googleBooks.tk) en la MainActivity se crea una instancia de un objeto retrofit. Esta instancia es la que va a contener la URL base, el conversor de Gson a el modelo de datos de Kotlin y toda la configuración para hacer las llamadas a internet (recordadndo que así lo creado antes, lo instancio e inicializo en la MainActivity, no dejandolo solo en la clase de la Interfaz = APIService). Lo que se busca en estos pasos es de cierta manera (analogía mode ON) crear "un teléfono" que nos permita a nosotros (usuario) llamar a la API, mediante su número telefónico (en este caso la Interfaz APIService es el número)



Este "teléfono" se inicia creando una función privada con nombre `getRetrofit`, el cual devuelve un tipo de variable `Retrofit` (de la librería importada, no confundir los nombres similares, el primero es el nombre de la función, y el segundo es el tipo de variable de esa función). Dentro de las llaves `{ }` lo que recibe la función es obtener como retorno (`return`) a `Retrofit` que tiene un constructor que lo cree (de ahí viene el `Builder`), luego una URL base (`baseUrl`, la que venimos usando hace rato) la cual es la parte que NO cambia de la llamada que hacemos a la API (OJO = la `baseUrl` siempre tiene que terminar con una barra diagonal `/`). Después se le añade (`.add`) un convertidor del modelo de datos, el cual es `GsonConverterFactory.create`, que resulta ser una de las librerías que se añaden al principio del proyecto. Finalmente se le dice que con todo lo que le pasó, construya (`.build()`) el objeto de la instancia `retrofit`.

Todo lo anterior en código quedaría de la siguiente manera.

```
private fun getRetrofit(): Retrofit {  
    return Retrofit.Builder()  
        .baseUrl("https://www.googleapis.com/books/v1/")  
        .addConverterFactory(GsonConverterFactory.create())  
        .build()  
}
```


4. Como la llamada anterior puede a veces ser un poco lenta en mandar su respuesta, esto puede llegar a bloquear el hilo (thread) principal de la app, lo que se hace para solucionar esto es trabajar asíncrono (síncrono seria en el hilo principal), para esto se utilizan la corrutinas (otra de las librerías implementadas al inicio del proyecto). De este modo se crea una función la cual se llama consultaDelUsuario y se le da de parámetro la consulta (query, del tipo String) que el usuario va a realizar (notar que aún el usuario no hace la consulta, pero esta funcion se crea antes, para atrapar de antemano esa consulta que creará el usuario), dicha consulta va a ser respondida en una corrutina para no saturar el hilo principal de la app*/

Así, dicho esto, se crea la corrutina de la siguiente manera. Se parte con una CoroutineScope, la cual lleva como parámetros un Dispatcher.IO (Dispatcher = Las corrutinas de Kotlin utilizan despachadores para determinar qué subprocesos se utilizan para la ejecución de corrutinas || Dispatchers.IO: es un despachador que está optimizado para realizar operaciones fuera del subproceso principal. Algunos ejemplos incluyen usar el componente Room, leer desde archivos o escribir en ellos (el Json de la API) y ejecutar operaciones de red (la API también supongo). Al final se le da la orden que se lance (.launch) con todo lo que esa función va a contener en su interior

Es importante señalar cuando se trabaja con Corrutinas, dentro de estas todo se va a desarrollar en un hilo secundario, de esta manera la función implementada queda de la siguiente manera: constante que realiza un llamada (call) al objeto Retrofit del punto 3., y que le dice que sea igual a el retrofit que llamó (getRetrofit), creándole (create) además una interfaz (que la hicimos nosotros antes, la ApiService, pero ni idea por que lleva la parte de "::class.java") y que con eso obtenga los libros buscados por título en la query (query = parámetro que va a consultar el usuario, atributo al inicio de la función).

Pregunta...

Si todo esto está pasando en un hilo secundario.... ¿Cómo interactuará con ello el usuario?, después de todo estas corrutinas tienen por propiedad que todo lo que ve e interactúa de cara al usuario debe encontrarse en el hilo principal, por esa razón al estar en segundo plano con la corrutina, no podremos usarla y ver las imágenes y textos de los libros o no??

Exactamente!!!! para hacer esto y salir del hilo secundario se usa la función runOnUiThread (eso hace que dentro de la corrutina, todo lo que este dentro de las llaves de este método runOnUiThread corra en el hilo principal).

Así, al final de esto, solo queda poner entre las llaves de runOnUiThread un if, el cual como condicional comprueba si la llamada definida anteriormente (call) es exitosa (.isSuccessful), de ser así vamos a generar un Log.v (Log de registro, eso significa la "v") el cual contiene la palabra "hola" para poder filtrar con facilidad cuando se le busque (puede ser cualquier palabra, pero intenta que no sea una que se repita mucho durante la aplicación o los procesos de Android y del Gradle, así pues, palabras como error, information, Log o similares, no te convienen) e inicialice la función del RecyclerView (esta función aún no la creamos, pero dejémosla así por ahora con el error, el cual al seguir avanzando en el tutorial y al llegar a la instancia de creación de dicha función se solucionará solo)

!!!IMPORTANTE!!!

Como la función usada acá pide un parámetro de búsqueda (la query:String en su constructor) y aún no la tenemos, pues no está hecho el SearchView, se le va a dar de manera temporal en el método onCreate, una

busqueda de prueba que se ejecute al iniciar la aplicación, esta será simular que el usuario está buscando el libro "el hobbit". De este modo el método onCreate queda de la siguiente manera (agrega la línea 26 = consultaDelUsuario("el hobbit"))

```

1      package com.example.librosapp
2
3      import ...
16
17     class MainActivity : AppCompatActivity() {
18
19
20
21
22     override fun onCreate(savedInstanceState: Bundle?) {
23         super.onCreate(savedInstanceState)
24         binding= ActivityMainBinding.inflate(layoutInflater)
25         setContentView(binding.root)
26         consultaDelUsuario(query: "el hobbit")

```

```

private fun consultaDelUsuario(query: String) {

    CoroutineScope(Dispatchers.IO).launch {

        val call =
        getRetrofit().create(APIService::class.java).buscarLibroPorTitulo(query)
        val libro = call.body()!!

        runOnUiThread {

            if (call.isSuccessful){

                Log.v("hola", call.body()!!.toString())
                initRecyclerView(libro)

            }

        }

    }

}

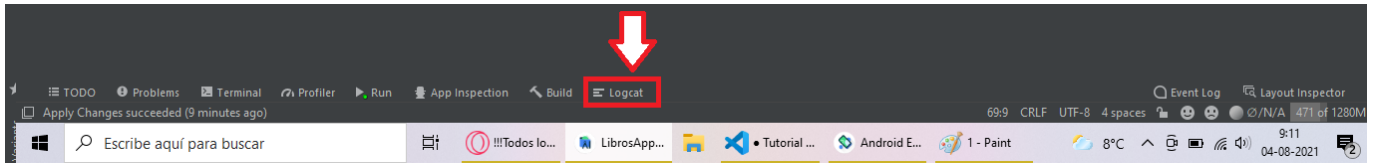
```

¿Podemos probar ahora si existe comunicación con la API?

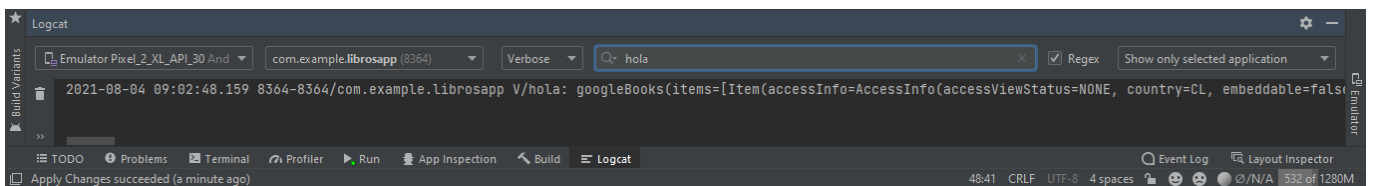
Obvio, para eso pusimos el Logcat.v, lo que tienes que hacer para probarlo es lo siguiente:

- Comenta (//) la función initRecyclerView() para que no genere errores por ahora.
- Ejecuta la aplicación (no esperes ver nada aún en ella, solamente las vistas que dejaste al inicio, con el Search View aún no funcional y con el RecyclerView sin nada).

- Cuando este lista y ya ejecutada, ingresa a la ventana del "Logcat" en la parte de abajo de Android Studio (ligeramente arriba de la barra de inicio de Windows).



- En el Logcat, en el buscador de la parte superior ingresa la palabra que pusiste en el mensaje Logcat del código (en mi caso "hola", sin las comillas). Si haciendo esto te aparece un mensaje, es que la aplicación entró en el "if" y por tanto hay respuesta de la API (la misma respuesta arroja en este caso los datos de la query de prueba configurada, es decir El Hobbit).



Acá se deja por ahora a medio hacer esta parte, y se salta a configurar el recyclerview, pues se necesita implementar este último para poder continuar este apartado.

OJO --> Al hacer esto es posible que la Aplicación se caiga, eso puede suceder por tres razones:

- a) Las imágenes de las portadas estan fallando al cargarse (no aparecen o se llega a un Nullpoint, o espacio donde no hay nada que mostrar y que confunde al emulador).
- b) Los tipos de variables generados por el Json to Kotlin converter fueron erróneas (pasa en algunos casos).
- c) Falta algún permiso.

Si esto sucede, dejarlo así, por ahora solo importa ver si existe respuesta por parte de la API o no

5. Para iniciar a configurar el RecyclerView, hay que notar que... MOMENTO!!!, antes al crear la apariencia de la MainActivity metí este componente y las CardView, pero que son esos??, Breve explicación en camino.... 😊

- **RecyclerView:** La clase RecyclerView nos permite mostrar un listado de elementos, lleva este nombre porque a medida que se renderizan los elementos de la lista, los elementos que dejan de observarse se reciclan para mostrar los elementos siguientes. Funciona en base a un Adaptador (Adapter = es el encargado de traducir datos en UI, posee) y un Holder (Holder = permite obtener referencias de los componentes visuales (views) de cada elemento de la lista).
- **CardView:** Si bien un RecyclerView representa una lista de elementos, cada elemento debe tener una UI (User Interface = Interfaz de Usuario) definida, pero con Material Design se suele usar la clase CardView para definir la apariencia de cada elemento del listado del Recycler (dicho simple, cada tarjeta dentro del listado es la CardView).

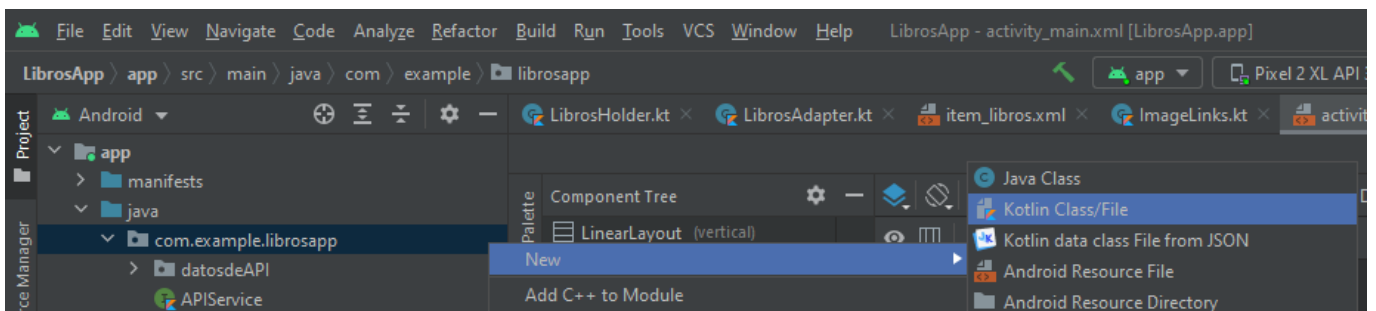


RecyclerView = Cuadro rojo || CardView = Cuadros morados

Dicho esto, podemos ver que las CardView al ser un componente interno que se mostrará en el RecyclerView ya las tenemos creadas, por lo que falta implementar el Adapter y el Holder, pero como la creación del primero depende del segundo, partiremos creando el Holder para luego meterlo en el Adapter.

Holder

Con esto, iniciamos creando una nueva clase para el Holder como se muestra en la imagen (en este caso la clase fue llamada LibrosHolder, recordar este nombre pues en breve se va a hacer referencia a ella cuando se crea el Adapter)



Una vez hecho esto, se codifica la clase LibrosHolder, la que contendrá entre paréntesis los atributos de una constante binding: `ItemLibrosBinding` (para así decirle a que esta enlazada y sobre que va a inflar las vistas (en este caso la CardView item libros)), esta va a obtener el RecyclerView unido a a su ViewHolder (la clase general de ViewHolder, no el que estamos creando nosotros) enlazado al `binding.root` (esto de nuevo, es del `ViewBinding`). Lo anterior dicho queda en código (tomando en cuenta las importaciones que se realizan) queda de la siguiente manera.


```
package com.example.librosapp

import androidx.recyclerview.widget.RecyclerView
import com.example.librosapp.databinding.ItemLibrosBinding

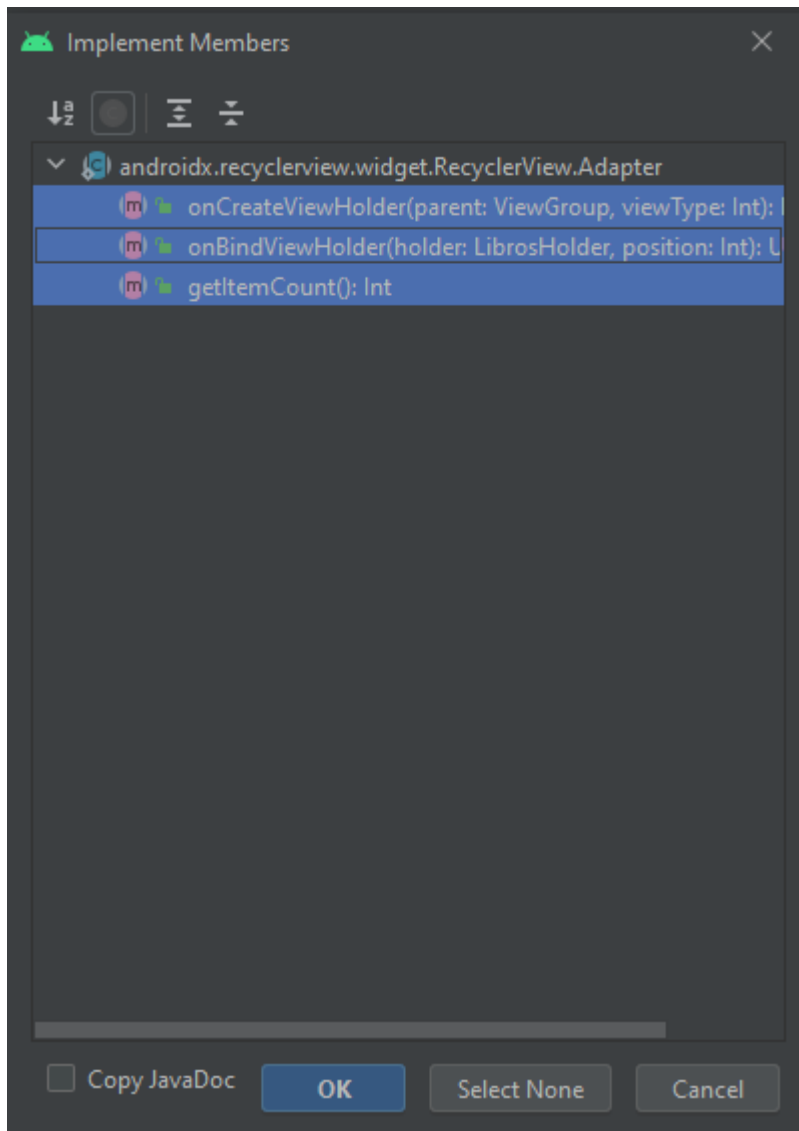
class LibrosHolder(val binding: ItemLibrosBinding):
    RecyclerView.ViewHolder(binding.root) {}
```

Adapter

Implementado el Holder, ahora crearemos el Adapter. Para esto creamos una nueva clase como se explico al inicio del Holder, pero esta vez la llamaremos LibrosAdapter, a esta misma se le entrega como parámetros entre paréntesis una constante privada llamada libros, de la data class googleBooks y fuera de los paréntesis, esta recibira como retorno a la clase RecyclerView.Adapter (esta clase no la creamos nosotros, es una clase del sistema creada al usar los RecyclerView y que trae su implementacion del adapter), finalmente esto entre los signos "<>" recibe LibrosHolder que nosotros mismos creamos en el paso anterior. Esto quedaria de la siguiente manera

```
45 
46 class LibrosAdapter (private val libros:googleBooks):RecyclerView.Adapter<LibrosHolder>() {
```

Sin embargo, ¿Por qué queda con un error?.... aaaaah, eso es porque aún no se han implementado los tres métodos fundamentales que hacen funcionar el Adapter, por lo que para solucionarlo hay que poner el cursor sobre la palabra LibrosAdapter y dar click sobre la ampolleta roja, y seleccionanr en ella la opción "Implement members" y a continuación en la ventana que aparece, marcar los 3 y dar click en OK.



Estos métodos generados automáticamente por el IDE lo que hacen es lo siguiente:

- **override fun onCreateViewHolder** = Su función es inflar el layout (archivo .xml) que representa a nuestros elementos, y devuelve una instancia de la clase ViewHolder que antes definimos. Así la misma usa como parámetros el ItemLibrosBinding (usado en la viewBinding) y lo infla (inflar = La activity y el layout están conectados por un proceso llamado inflación. Cuando la activity inicia, las vistas que están definidas en el .xml (el layout) se convierten en (o se "inflan") objetos de vista de Kotlin en la memoria del dispositivo. Cuando esto pasa, la activity puede dibujar estos objetos en la pantalla y modificarlos si es necesario), esto lo logra a través de la clase LayoutInflater, que recibe de parámetros parent (el recycler view, donde se van a llenar las cardview) y context (la cardview)... el false ni idea porque va, pero es necesario ponerlo

```

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): LibrosHolder
{
    return LibrosHolder(
        ItemLibrosBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            false
        )
    )
}

```

```
)
}
```

- **override fun onBindViewHolder** = Esta función enlaza nuestros datos de la API con cada ViewHolder, esto lo logra diciéndose a sí misma (así lo imagino yo al menos XD) "estoy en la posición X (de hay el atributo position: Int), y tengo la instancia el ViewHolder (holder: LibrosHolder), por lo que con ello voy a pintar lo que me indiquen, en la posición y en los campos que tenga disponible para eso (se indica titulo, autor e imagen en los campos disponibles de tvtitulo, tvautor e ivlibro que estan vacíos hasta este momento y que se encuentran en la CardView, tomar en cuenta que las imagenes llegan como una String, que es la dirección web donde se encuentran, pero la libreria Glide usada en este método e importada al inicio las convierte en imagenes visibles para el usuario).

Pero que pasa si se le intenta dar a alguno de los Holder algún dato que no esté disponible en la API o que no exista??... pues adivina, lo que va a pasar es la función favorita de todos al hacer las app... caérse o crashearse XD jajajajaja. Pero eso tiene un solución, recuerdas cuando antes probamos la llamada poniendo el Log.v "hola" y veiamos la respuesta?, hay definimos los posibles errores que podrian aparecer, entre ellos que no aparecieran las imagenes. Por eso en este bloque que es el encargado de pintar (o mejor dicho inflar los layout, y que por tanto va a ser el que tenga el error), se le aplica una función de manejo de errores llamada "Try-Catch" o "Intenta-Atrapa" (Intenta poner los datos donde te dicen, y si no puedes producto de algún error, atrapa ese error y en vez de poner los titulos, autores e imagenes, coloca la frase "Sin Título", "Sin Autor" y la imagen substituta "Sin imagen disponible", que se encuentra en R.mipmap.noimagendisponible)

```
override fun onBindViewHolder(holder: LibrosHolder, position: Int) {
    holder.binding.apply {

        var autores = libros.items[position].volumeInfo.authors
        tvtitulo.text = libros.items[position].volumeInfo.title

        try {
            Glide.with(ivlibro.context)

                .load(libros.items.get(position).volumeInfo.imageLinks.thumbnail)
                    .fitCenter()
                    .into(ivlibro)
            tvautor.text = autores[0]
            tvtitulo.text = libros.items[position].volumeInfo.title
        } catch (e: NullPointerException) {
            ivlibro.setImageResource(R.mipmap.noimagendisponible)
            tvautor.text = "Sin autor"
            tvtitulo.text = "Sin título"
        }
    }
}
```


Sobre lo que se encuentra sobre estas líneas debo hacer notar 3 cosas:

a) La primera es que hay que darse cuenta que los datos van a venir de la dirección que se le indica en la API, la cual va teniendo sus datos ordenados de manera anidada, por eso se busca de esta manera (en el ejemplo se busca el título, pero si uno de manera manual entre los archivos data class generados sigue la misma dirección va a terminar encontrando el mismo parámetro en la misma ubicación, de hecho, intenta navegar entre los archivos y llegar a autor o las imágenes)

```
libros.items[position].volumeInfo.title // libros = la variable declarada en  
Adapter de tipo libros:googleBooks
```

b) La segunda es que pueden existir distintos Autores en un libro, por lo que al indicar la parte siguiente:

```
var autores = libros.items[position].volumeInfo.authors  
    .  
    .  
    .  
    .  
tvautor.text = autores[0]
```

Se le dice al sistema que coloque en las CardView solo el primero de los autores que encuentre en cada libro.


c) Y la tercera y última, que al parecer Glide trabaja con imágenes https (seguras, de hay la s) y que Google Books esta entregando solo imágenes http (sin la s, no seguras), por lo que aún a pesar de estos cambios la imagen no se va a mostrar. Para eso se incorpora el siguiente permiso en el Manifest del proyecto.

Copia y pega esto donde se muestra en el imagen --> `android:usesCleartextTraffic="true"`

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.librosapp">
4
5      <uses-permission android:name="android.permission.INTERNET"/>
6      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
7
8      <application
9          android:allowBackup="true"
10         android:icon="@mipmap/ic_launcher"
11         android:label="LibrosApp"
12         android:roundIcon="@mipmap/ic_launcher_round"
13         android:usesCleartextTraffic="true"
14         android:theme="@style/Theme.LibrosApp">
15         <activity android:name=".MainActivity">
16             <intent-filter>
17                 <action android:name="android.intent.action.MAIN" />
18
19                 <category android:name="android.intent.category.LAUNCHER" />
20             </intent-filter>
21         </activity>
22     </application>
23
24 </manifest>

```



- **override fun getItemCount** = retornar el numero de items que va a tener la lista (si son muchos, .size es el parámetro que identifica que la cantidad va a ser el tamaño de la lista)

```

override fun getItemCount(): Int {
    return libros.items.size
}

```

De todo los métodos y funciones anteriormente descritas, se desprende que el código en la clase LibrosAdapter queda de la siguiente manera:

```

package com.example.librosapp

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.example.librosapp.databinding.ItemLibrosBinding
import com.example.librosapp.datosdeAPI.googleBooks

class LibrosAdapter (private val

```

```

libros:googleBooks):RecyclerView.Adapter<LibrosHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
LibrosHolder {
        return LibrosHolder(
            ItemLibrosBinding.inflate(
                LayoutInflater.from(parent.context),
                parent,
                false
            )
        )
    }

    override fun onBindViewHolder(holder: LibrosHolder, position: Int) {
        holder.binding.apply {

            var autores = libros.items[position].volumeInfo.authors
            tvtitulo.text = libros.items[position].volumeInfo.title

            try {
                Glide.with(ivlibro.context)

.load(libros.items.get(position).volumeInfo.imageLinks.thumbnail)
                .fitCenter()
                .into(ivlibro)
                tvautor.text = autores[0]
                tvtitulo.text = libros.items[position].volumeInfo.title
            } catch (e: NullPointerException) {
                ivlibro.setImageResource(R.mipmap.noimagendisponible)
                tvautor.text = "Sin autor"
                tvtitulo.text = "Sin título"
            }
        }
    }

    override fun getItemCount(): Int {
        return libros.items.size
    }
}

```

-
6. Una vez completado lo anterior, y ya teniendo todo lo necesario para poder implementar el RecyclerView, es que podemos de una vez por todas inicializarlo dentro de la MainActivity para que así funcione e interactúe con el usuario. Es por esto que, para realizar esta labor lo primero a realizar es generar la función `initRecyclerView` (si es que la recuerdas de antes, es una función que llamamos en el punto 4. dentro del `if` donde pusimos el `Log.v` "hola" y que nos generaba error por llamar a una función que no existía, bueno ahora al crearla se te va a solucionar solo esos error).

La función `initRecyclerView` será del tipo privado y tendrá como argumentos ente paréntesis `libros:googleBooks` **(creado en el Adapter, y que ahora necesitas ir, descomentar y a colocar dentro de la función declarada en el `if` que tiene el `Log.v` para que quede de esta manera -->**

initRecyclerView(libros) <--) . En su interior esta función va a hacer que el Adapter se una al RecyclerView mediante la ViewBinding tal como se ve en el bloque de código siguiente.

```
private fun initRecyclerView(libros:googleBooks) {  
  
    val librosAdapter = LibrosAdapter(libros)  
    binding.rvlibros.adapter = librosAdapter  
    binding.rvlibros.layoutManager = LinearLayoutManager(this)  
}
```

Importante señalar que en la línea "binding.rvlibros.adapter = librosAdapter" yo personalmente me confundía con los dos adapter, pero luego descubrí que lo que hace es presentar el adapter al recycler, dicho de otra manera, en esa línea a la izquierda "adapter" es que esta pidiendo uno, y librosAdapter con minúscula (que es la variable que generé arriba de esa línea y que es igual a LibrosAdapter con Mayúscula, que es el Adapter creado por mí) es el adaptador que le entregamos para que use el Recycler.

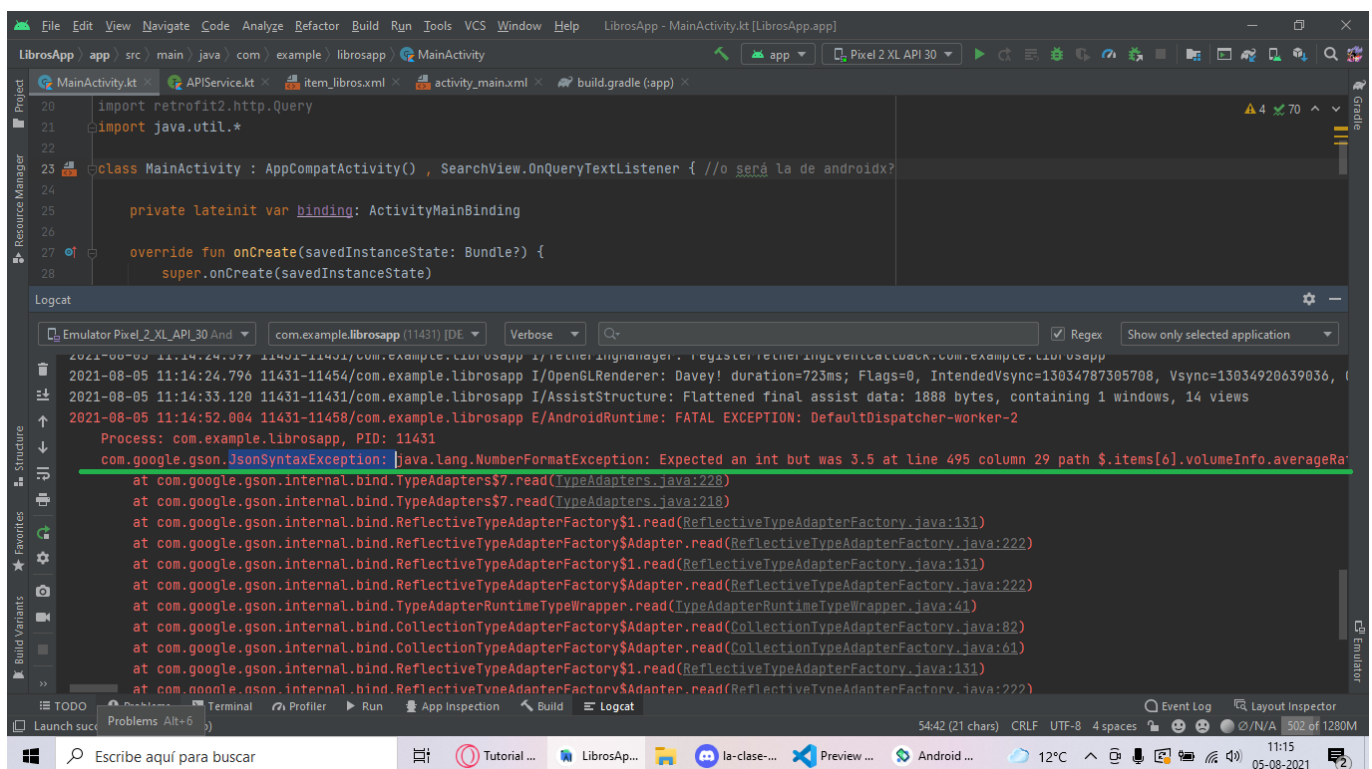
Con eso el adapter queda listo y ya podríamos probar si la app funciona sin el SearchView por ahora, solo un detallito breve... ¿Que pasa si no hay internet o si hay algún problema al momento de buscar?, para dar solución a eso completa el if (el único de la MainActivity) con su else, y hay dentro pon un Toast avisando el error en la siguiente posición del código (por ahora no veremos su efecto, pues no está listo el SearchView, pero es mejor dejarlo puesto de antemano).

```
private fun consultaDelUsuario(query: String) {  
  
    CoroutineScope(Dispatchers.IO).launch {  
  
        val call =  
        getRetrofit().create(APIService::class.java).buscarLibroPorTitulo(query)  
        val libro = call.body()!!  
  
        runOnUiThread {  
  
            if (call.isSuccessful){  
  
                Log.v("hola", call.body()!!.toString())  
                initRecyclerView(libro)  
                //Agrega las 2 líneas siguientes (desde else hasta show())  
            }else{  
                Toast.makeText(applicationContext, "Ha ocurrido un error",  
                Toast.LENGTH_SHORT).show()  
  
            }  
        }  
    }  
}
```

Probaste la app??... y?? Murío o no?? jajajajajaja XD

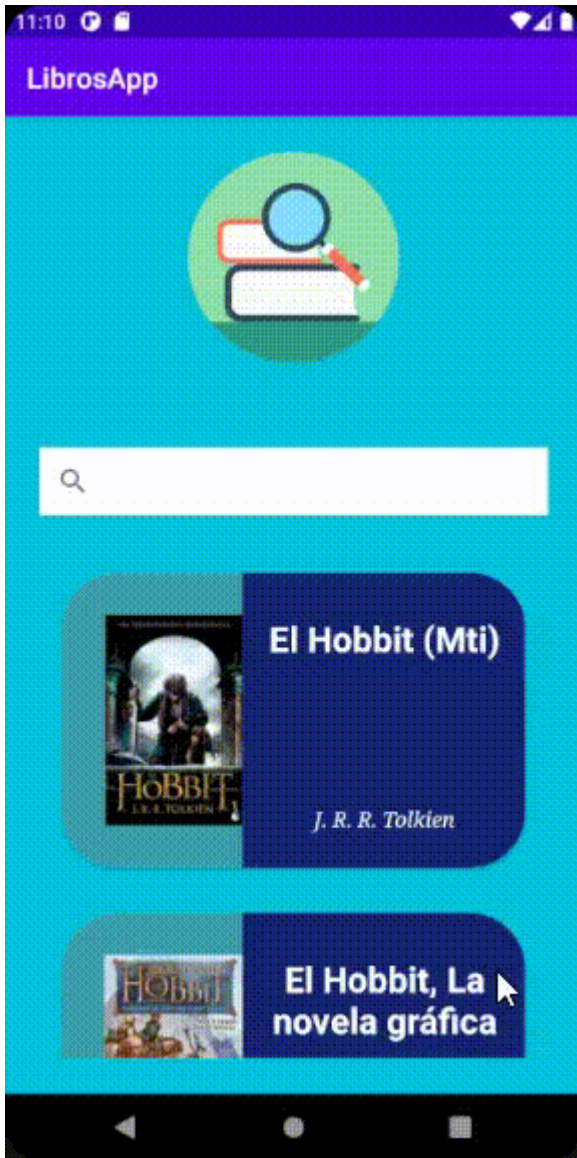
Eso es porque hay un último error que solucionar para poder dejarla más o menos estable (por ahora, cuando le empieces a meter mas cosas de seguro morirá otra vez, pero por ahora el tutorial te corrige esto XD)

Al punto, la razón por la que se te cae la app es porque cuando transformamos el archivo Json a clases de Kotlin con el plugin que instalamos al inicio del tutorial pueden existir errores, específicamente tienes que revisar los apartados de ListPrice, ListPriceX, RetailPrice y RetailPriceX, en donde producto de que el archivo que copiamos traía solo 10 libros de los millones que tiene Google Books, las variables dentro de estos archivos fueron identificadas como Int, cuando en realidad son Long, por lo que estas pasando un tipo de dato no soportado en esa variable. Para solucionarlo, anda al archivo con el problema (puedes revisar cual es ejecutando la app, esperando a que se caiga y luego revisando el Logcat para ver la razón y buscar el nombre del archivo) y cambia las variables Int por Long y si es que existe alguna otra que también de un error, cambiarla al tipo de variable correspondiente y eso debería bastar para corregir ESE error.



Como ejemplo en la imagen de arriba, lo subrayado en verde dice en el Logcat dice que "se espera un int pero fue 3.5" es decir espera un entero pero recibe un float, ¿Dónde?, pues al final de la línea lo dice items.volumeInfo.averageRating (esto está ubicado en las data class que creamos al inicio, y que se busca siguiendo ese mismo orden)

Ahora sí es el momento, prueba la app... (ahora no estoy trolleando, debería funcionar, de hecho te dejo un GIF de como se me ve a mi hasta este punto del tutorial).



Nota como en algunos casos, al no tener datos la CardView reemplaza con sin autor, sin titulo y la imagen asignada en el caso de que el libro no tenga una imagen de portada.

1. El último paso para terminar este tutorial (ahora sí) es implementar el SearchView, para que de esta manera podamos buscar los datos de libros a nuestra elección. Así que manos a la obra.

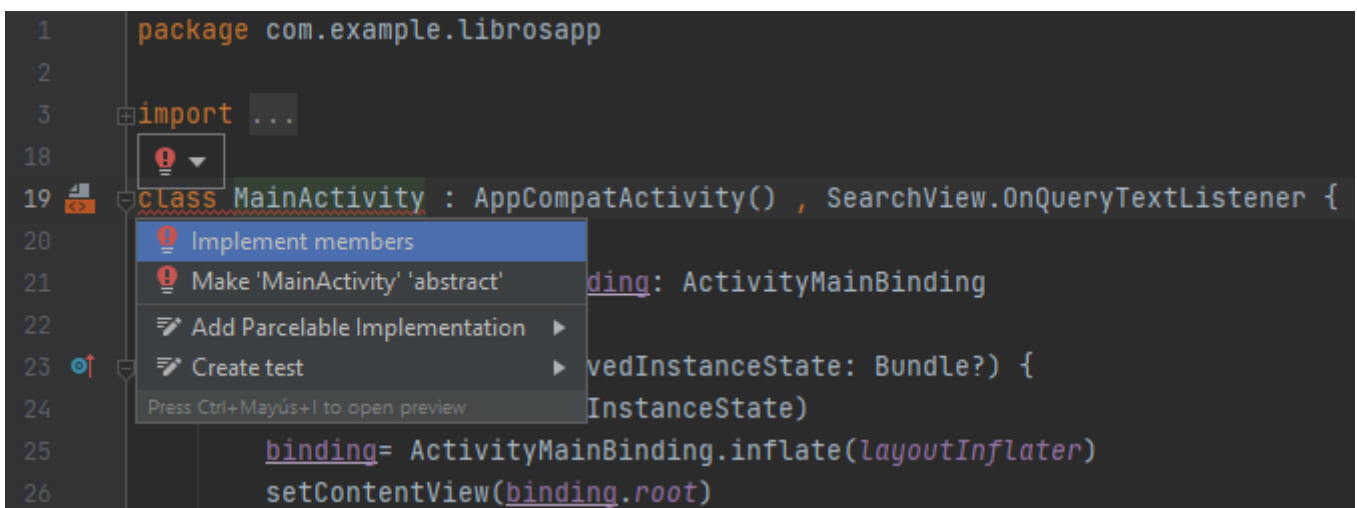
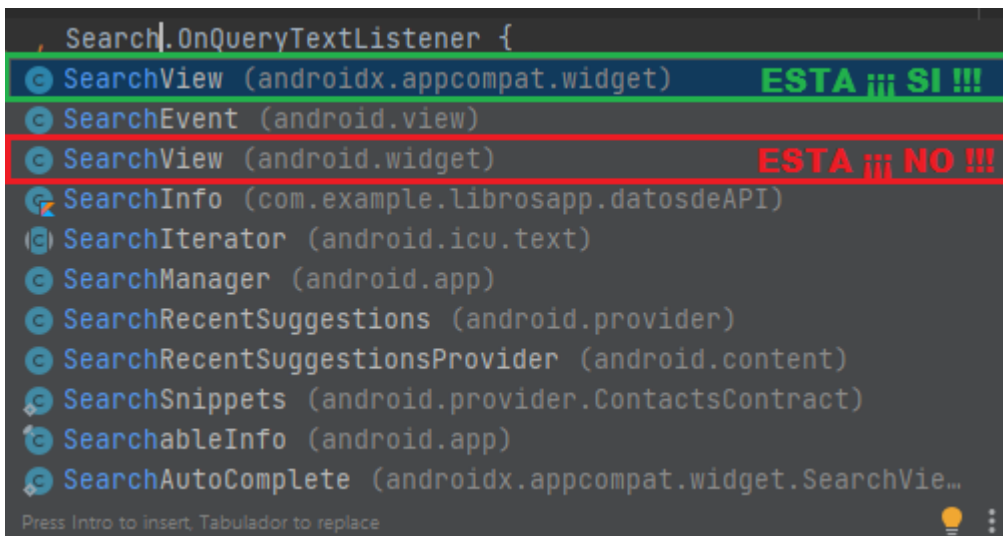
Primero, vuelve al inicio del proyecto, específicamente al inicio de la clase MainActivity y a esta línea:

```
class MainActivity : AppCompatActivity(){
```

Agregale " , SearchView.OnQueryTextListener " para que te quede algo así (con error incluido):

!!!MUY IMPORTANTE!!! Como desde un inicio estamos trabajando con un SearchView que viene de Android X (una version más nueva y mejor de Android, si no me crees anda a la Vista del SearchView en el ,xml de la Main Activity y revisa el nombre del componente) en este paso de arriba, el de "SearchView.OnQueryTextListener" presta mucha atención a importar la libreria de ese android, pues saldrán

dos que son iguales, y con una de ellas no va a funcionar. De aquí en más, siempre recuerda revisar mucho lo que importas 😊.



Como puedes en la segunda imagen de arriba, para solucionar ese error hay que implementar dos miembros que faltan a la clase, por lo que de manera similar a como lo hicimos antes con el Adapter (si no recuerdas como, ve al apartado adapter y revisa como los creamos) creamos de forma automática a través el IDE ambos miembros faltantes, obteniendo el siguiente resultado.

```
override fun onQueryTextSubmit(query: String?): Boolean {
    TODO("Not yet implemented")
}

override fun onQueryTextChange(newText: String?): Boolean {
    TODO("Not yet implemented")
}
```

Ahora toca completar estos dos métodos, partiremos con el segundo que es más sencillo:

- ***override fun onQueryTextChange*** = Este método nos avisará de cada carácter que se añada al buscador, pero nosotros no necesitamos eso, solo necesitamos que nos avise cuando el usuario haya terminado de escribir así que como el método espera el retorno de un Boleano, solo le pasamos entre sus llaves return true, quedando de la siguiente manera.

```
override fun onQueryTextChange(newText: String?): Boolean {  
    return true  
}
```

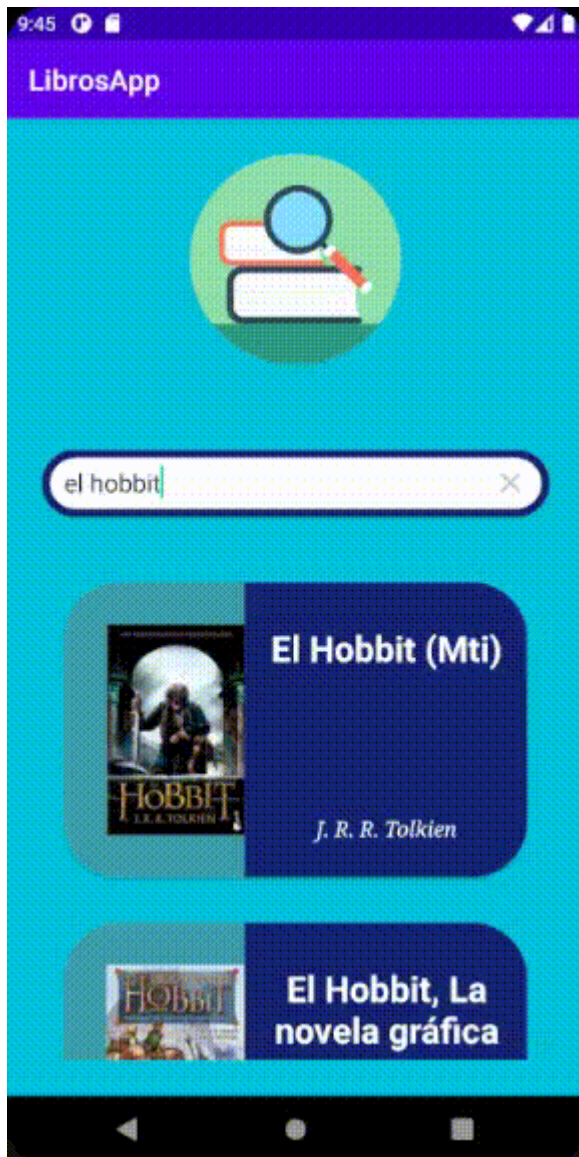
- ***override fun onQueryTextSubmit*** = Este método se trabaja con un if que indica que si la consulta del usuario NO es nula o esta vacia (por eso el " ! " al inicio y el metodo isEmpty()), llame a la funcion searchByName, la que recibe como parámetros la consulta (query) en minusculas (.toLowerCase(Locale.getDefault()))), esto se debe a que no sabemos como quiere la API que realizemos las consultas, por eso para asegurarnos, le pasamos todo en minusculas (antes se usaba una función llamada .toLowerCase que al día de hoy 04 de Agosto de 2021 está deprecada). Finalmente, como en el método anterior, se le pasa un return true pues espera de retorno un valor booleano.

```
override fun onQueryTextSubmit(query: String?): Boolean {  
    if (!query.isEmpty()) {  
        consultaDelUsuario(query.toLowerCase(Locale.getDefault()))  
    }  
    return true  
}
```

Ya para finalizar, con estos dos métodos listos, lo que se hace es hacer el ViewBinding en el método onCreate, agregando esta línea entre sus llaves.

```
binding.svlibros.setOnQueryTextListener(this)  
  
// Ojo aquí, el setOnQueryTextListener tambien debe ser de android x, que no  
sea otro
```

Con todo eso ya podriamos hacer correr la app y probar el SearchView, efectuando una busqueda de algun libro diferente a el hobbit para ver los resultado entregados.



EXTRAS

Lo siguiente no es necesario implementarlo, pero hace que se vea bonito, por lo mismo no entraré en detalle y sólo copiaré lo que encontré en un tutorial de internet (agradecimientos a AristiDevs - [Enlace a su tutorial completo](#))

Escondiendo el teclado

Si ejecutas la app ya va a funcionar, pero si me das un minuto más te enseñaré una función extra para dejar más bonita la app y es que al buscar en el buscador, una vez haya buscado no va a esconderse el teclado y visualmente queda bastante feo. Para ello simplemente añade esta función (recuerda importar el `InputMethodManager`).

```
private fun hideKeyboard() {  
    val imm = getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager  
    imm.hideSoftInputFromWindow(binding.root.windowToken, 0)  
}
```

No entraré en detalle porque es más avanzado pero ahora podemos llamar a `hideKeyboard()` desde `searchByName()` al terminar el `if`.

```
private fun consultaDelUsuario(query: String) {

    CoroutineScope(Dispatchers.IO).launch {

        val call =
getRetrofit().create(APIService::class.java).buscarLibroPorTitulo(query)
        val libro = call.body()!!

        runOnUiThread {

            if (call.isSuccessful){

                Log.v("hola", call.body()!!.toString())
                initRecyclerView(libro)

            }else{
                Toast.makeText(applicationContext, "Ha ocurrido un error",
Toast.LENGTH_SHORT).show()
            }
            hideKeyboard() // Agregar esta línea
        }
    }
}
```

De esta manera cada vez que ejecutemos una búsqueda, después de apretar enter (o buscar) el teclado se ocultará automáticamente y podremos ver los resultados sin problemas.

Customizando el fondo del SearchView

1. En la carpeta Drawable, dar click derecho y generar un New --> New Drawable Resource File, ponerle de nombre "custom_search_background"
2. En este nuevo archivo poner el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
android:shape="rectangle">

    <solid android:color="#FDFFFFFF"/>
    <corners
        android:topLeftRadius="25dp"
        android:topRightRadius="25dp"
        android:bottomLeftRadius="25dp"
        android:bottomRightRadius="25dp"/>
```

```
<stroke
    android:color="#162473"
    android:width="6dp"/>

</shape>
```

Notar que que todo está entre etiquetas <Shape> y que se le da una forma rectangular, además de un color sólido, unos márgenes azules y esquinas (corners redondeadas).

3. Una vez listo esto, y en el código de la vista SearchView, agregar el atributo android:background = "@drawable/custom_search_background", quedando de la siguiente manera.

```
<androidx.appcompat.widget.SearchView
    android:id="@+id/svlibros"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="5dp"
    android:layout_marginStart="25dp"
    android:layout_marginBottom="5dp"
    android:layout_marginEnd="25dp"
    android:background="@drawable/custom_search_background" //Agregar esta
línea
    android:queryBackground="@android:color/transparent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintVertical_bias="0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:queryHint="@string/buscar"
/>
```

