Chuleta de Room

jueves, 7 de marzo de 2024 17:55

NO ABRIR PROYECTOS DESDE ANDROID STUDIO PARA COPIAR Y PEGAR

1. Puesta en marcha

Añadir el permiso de internet y el viewbinding. Importar las librerías de Picasso, Retrofit, Room y Corrutinas

En el manifest `<uses-permission android:name="android.permission.INTERNET"/>`

Arriba (fuera) de dependencies:

```
buildFeatures{
    viewBinding = true   //Además, hay que darle a sincronizar
}
```

En dependencies:

```
//Retrofit
implementation("com.squareup.retrofit2:retrofit:2.9.0")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")

//Picasso
implementation ("com.squareup.picasso:picasso:2.8")

//Room
implementation ("androidx.room:room-ktx:2.6.1")
kapt ("androidx.room:room-compiler:2.6.1")

// Corrutinas
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.4.2")
```

En Kotlin, `mutableListOf<String>` es una colección mutable que puede contener elementos de tipo String. La palabra clave mutableListOf indica que esta lista es mutable, lo que significa que puedes agregar, eliminar o modificar elementos después de que se haya creado la lista.

Por otro lado, una lista no mutable se declara con la palabra clave `listOf<String>`. Una lista no mutable es una colección de elementos que no se pueden modificar una vez que se ha creado la lista. Esto significa que no puedes agregar, eliminar o modificar elementos en una lista no mutable después de su creación. Esencialmente, es una lista de solo lectura.

2. View binding

```
private lateinit var binding: ActivityMainBinding (o el que sea tu main)

En el onCreate():
binding = ActivityMainbinding.inflate(layoutInflater)
setContentView(binding.root)
```

3. Crear el layout "activity_main.xml" con un SearchView y un RecyclerView, ponerles IDs

4. Definir un modelo de datos (DogResponse, SuperheroResponse…)

```
data class DogsResponse(
        @SerializedName("status") var status: String,
        @SerializedName("message") var message: List<String>
)
```

En este caso, la respuesta que recibimos de la API contiene un String y una lista de Strings
Es importantísimo que el nombre dentro de SerializedName sea el mismo al que recibimos del JSON

5. Crear una interfaz

Usamos @GET, consultamos la documentación para ver la forma del link de la API y vemos cual es el parámetro
Valoramos cuál es el parámetro de búsqueda. Hay tres formas principales de hacerlo:

1. @Path -> Si una parte del path es reemplazada por un parámetro

```
@GET("/api/10229233666327556/search/{name}")
suspend fun getDogsByBreed(@Path("name") dogBreed: String): Response<DogsResponse>
```

2.@Url

```
@GET
suspend fun getDogsByBreed(@Url url:String): Response<DogsResponse>
```

3.

@Query: Esta anotación se utiliza para añadir parámetros de consulta a la URL.
Por ejemplo, si necesitas pasar un parámetro de consulta como **?page=1**, puedes utilizar esta anotación.

```
@GET("/api/10229233666327556/search")
suspend fun getDogsByBreed(@Query("page") page: Int): Response<DogsResponse>
```

6. Creamos el código de retrofit y las corrutinas

1. getRetrofit()

```
private fun getRetrofit(): Retrofit {
    return Retrofit
        .Builder()
        .baseUrl("https://superheroapi.com/")   //la parte fija de la URL de la Api, debe terminar en "/"
        .addConverterFactory(GsonConverterFactory.create())
        .build()
}
```

2. Variable "retrofit"

Instanciar fuera del onCreate() -> `private lateinit var retrofit: Retrofit`
Declarar dentro del onCreate() -> `retrofit = getRetrofit()`

3. searchByName(query:String)

La variable query es la parte no fija de la URL de la api… hay que determinar la forma de plantearla
Por ejemplo, si después de la parte fija hay un "**breed/images**" y nosotros queremos buscar por **breed** -> `getDogsByBreed("$query/images")`
Si no se da el caso, simplemente lo dejaremos como `getDogsByBreed(query)`

```
private fun searchByName(query: String) {
    binding.progressBar.isVisible = true
    CoroutineScope(Dispatchers.IO).launch {
        val myResponse: Response<DogsResponse> = retrofit.create(ApiService::class.java).getDogsByBreed(query)
        if (myResponse.isSuccessful) {
            val response: DogsResponse? = myResponse.body()
            if (response != null) {
```

```
                // se mostrarán cositas del RecyclerView
                runOnUiThread {
                    adapter.updateList(response.superheroes)
                    binding.progressBar.isVisible = false
                }
            }

        } else {
            Log.i("Consulta", "No funciona :(")
        }
    }
}
```

7. Adapter (con los 3 métodos) , ViewHolder e item_dog.xml

Prácticamente copiar y adaptar alguno que ya tenga hecho

- En "item_dog.xml" tiene que haber un ImageView con un id "**ivDog**" y con android:scaleType="centerCrop"
- En el ViewHolder tiene que haber un

```
private val binding = ItemDogBinding.bind(view)
```

y un

```
fun bind(superheroItemResponse: SuperheroItemResponse) {
    binding.tvSuperheroName.text = superheroItemResponse.name    //pinta el nombre dentro del textview "tvSuperheroName" del item
    Picasso.get().load(superheroItemResponse.superheroImage.url).into(binding.ivDog)
            //Picasso.get().load(url) -> coge la url del response (su estructura y nombre estará definida en el DogResponse9, la convierte en una imagen y la mete en ivDog del item
}
```

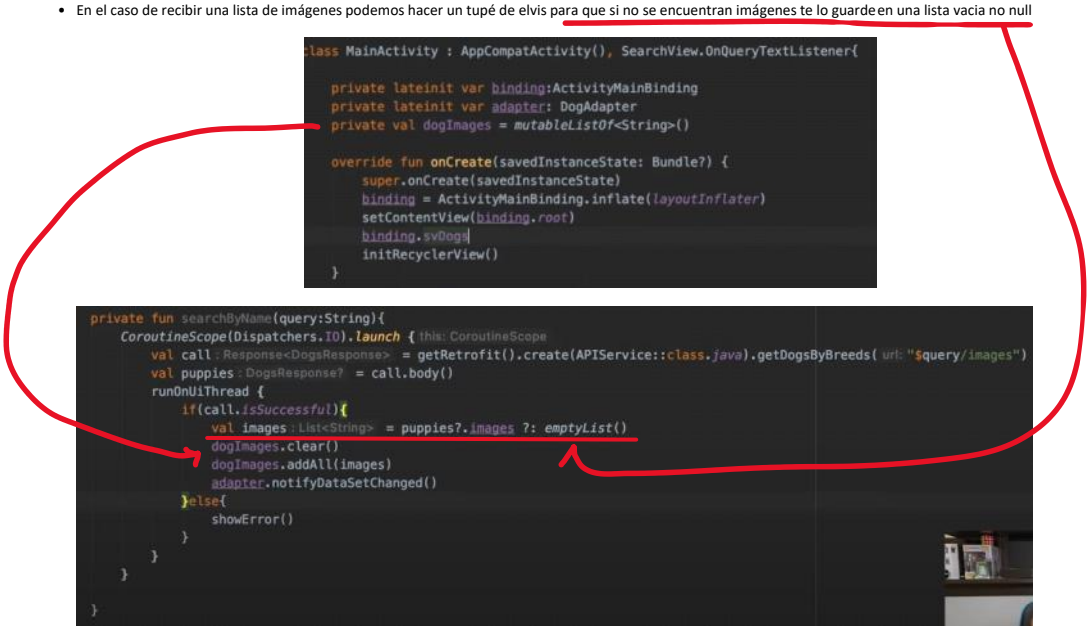
8. Inicializamos cosas en el Main Activity y rematamos el searchByName()

Inicializar el RecyclerView y el adapter
Ojo si tenemos que pasar listados, en ese caso vamos a tener que instanciar (lo más seguro) una mustableListOf<String>

Como la corrutina está tomando la respuesta de la API en segundo plano, queremos que si la respuesta no es null, se cargue en el hilo principal:

```
if (response != null) {
    Log.i("Cuerpo de la consulta", response.toString())
    runOnUiThread {
        adapter.updateList(response.superheroes)
        binding.progressBar.isVisible = false
    }
}
```

- En el caso de recibir una lista de imágenes podemos hacer un tupé de elvis para que si no se encuentran imágenes te lo guarde en una lista vacia no null



```
class MainActivity : AppCompatActivity(), SearchView.OnQueryTextListener{

    private lateinit var binding:ActivityMainBinding
    private lateinit var adapter: DogAdapter
    private val dogImages = mutableListOf<String>()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.svDogs
        initRecyclerView()
    }
```

```
private fun searchByName(query:String){
    CoroutineScope(Dispatchers.IO).launch { this: CoroutineScope
        val call : Response<DogsResponse> = getRetrofit().create(APIService::class.java).getDogsByBreeds( url: "$query/images")
        val puppies : DogsResponse? = call.body()
        runOnUiThread {
            if(call.isSuccessful){
                val images : List<String> = puppies?.images ?: emptyList()
                dogImages.clear()
                dogImages.addAll(images)
                adapter.notifyDataSetChanged()
            }else{
                showError()
            }
        }
    }

}
```

9. Añadir el SearchView.OnQueryTextListener

En nuestra clase MainActivity añadimos ", SearchView.OnQueryTextListener"

```
class MainActivity : AppCompatActivity(), SearchView.OnQueryTextListener{
```

Para que funcione dberemos implementar dos métodos en esa clase:

1. *OnQueryTextChange()*

Nos avisará de cara 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 lo dejamos como está y no hacemos nada.

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

2. *OnQueryTextSubmit()*

Este método será llamado cuando el usuario pulse en enter al terminar de buscar y ahí es cuando tendremos el texto que hemosescrito y se lo pasaremos a retrofit para que haga la petición a internet.

```
override fun onQueryTextSubmit(query: String?): Boolean {
    if(!query.isNullOrEmpty()){
        searchByName(query.toLowerCase())
    }
    return true
}
```

Volvemos a la función onCreate() y hacemos que nuestro searchview implemente el listener que hemos creado

```
binding.svDogs.setOnQueryTextListener(this)
```