API RETROFIT

Lo primero es acordarse de las dependencias y el viewBinding

```
    buildFeatures{
        viewBinding  true
    }
}


dependencies {

    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.1'
    implementation 'com.google.android.material:material:1.0.0'

    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    //PICASSO
    implementation 'com.squareup.picasso:picasso:2.71828'

    //RETROFIT
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation "com.squareup.retrofit2:converter-gson:2.9.0"

    //Coroutines
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.4.1")
}
```

1-Lo primero es ver el MainActivity,vamos a ver paso por paso.
Tenemos la variable del Binding y unas listas donde guardaremos los distintos atributos que nos ofrece la api y extendemos la clase del SearchView para obtener sus metodos.

```
class MainActivity : AppCompatActivity(), androidx.appcompat.widget.SearchView.OnQueryTextListener


    lateinit var binding: ActivityMainBinding
    //initialize the character adapter
    private lateinit var adapter: PjAdapter
    //We create the mutableList of all the attributes that we will show
    private val dogImages = mutableListOf<String>()
    private val nombre = mutableListOf<String>()
    private val species = mutableListOf<String>()
    private val gender = mutableListOf<String>()
```

2- Ahora realizaremos el inflate del binding y también un par de cosas de un splascreen en el onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    //First we have added a sleep to show the Splash Screen
    Thread.sleep( millis: 3000)
    //We return to the normal theme
    setTheme(R.style.Theme_RestApiRuben)

    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
```

3-Aquí declaramos un boton el cual al clickar haremos un intent a la página web de la API

```
//We initialize the button with binding
val btn_click_me = binding.btnGoWeb as ImageButton

//We have created a listener which when we click it will bring up a sncackbar which if we continue it will take us to the officia
btn_click_me.setOnClickListener() { it: View!
    val snack = Snackbar.make(it, text: "¿Estas seguro de que quieres ir a la web?", Snackbar.LENGTH_LONG).setAction( text: "LOAD"){
        val url = "https://rickandmortyapi.com/"
        val i = Intent(Intent.ACTION_VIEW)
        i.data = Uri.parse(url)
        startActivity(i)
    }.show()

}
```

4-Por último en el oncreate tenemos un listener especial del searchview,no continuará hasta que le demos a buscar,después inicia el recycleviewer

```
//This is a special search listener which will not
binding.svDogs.setOnQueryTextListener(this)
initRecyvlerView()//we load everything
```

5-En el metodo para iniciar el recycleviewer básicamente creamos:
-Adapter → pasandole las listas vacias
-Layout manager
y por último le asignamos el adapter anteriormente creado al recycleviewer que tenemos en el XML(el cual luego veremos)

```
private fun initRecyvlerView() {
    adapter = PjAdapter(dogImages,nombre,species,gender)
    binding.rvDogs.layoutManager = LinearLayoutManager( context: this)
    binding.rvDogs.adapter = adapter

}
```

5-2 Antes de seguir con el metodo vamos a ver lo que son las distintas clases ,la primera que veremos sera el adaptador el cual le pasamos las listas.
Esta todo explicado en el códgio

```
class PjAdapter(val images:List<String>, val nombre: List<String>, val specie: List<String>, val gender: List<String>): RecyclerView.Adapter<PjViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PjViewHolder {
        val layoutInflate = LayoutInflater.from(parent.context)

        //Devuelve el PjViewHolder inflandole el layout del item que debe repetir
        return PjViewHolder(layoutInflate.inflate(R.layout.item_pj,parent,  attachToRoot: false))

    }

    //coge el tamaño de las imagenes?
    override fun getItemCount(): Int = images.size

    //coge de las listas los strings que toquen y lo pasan a unas variables las cual se las pasaremos al holder
    override fun onBindViewHolder(holder: PjViewHolder, position: Int) {
        val item:String = images[position]
        val nombre2:String= nombre[position]
        val especie:String = specie[position]
        val genero:String= gender[position]
        holder.bind(item,nombre2,especie,genero)
    }

}
```

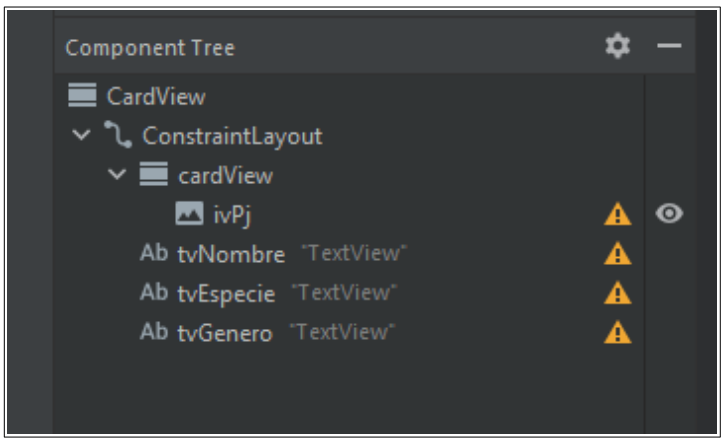
5-3 Ahora veremos el ViewHolder del recycleviewer,aquí simplemente tenemos un binding el cual pertenece al XML del ItemPJ que luego veremos(objeto repetible).
Como podemos ver estamos accediendo a los datos el ItemPJ y las variables que han llegado aquí las ponemos en los textos(estas variables son los atributos que solicitamos a la api)

```
import android.view.View
import androidx.recyclerview.widget.RecyclerView
import com.rucech.restapiruben.databinding.ItemPjBinding
import fromUrl

class PjViewHolder(view: View): RecyclerView.ViewHolder(view) {
    private val binding = ItemPjBinding.bind(view)


    fun bind(image:String,nombre:String,genero:String,especie:String){
        //Extension function
        binding.ivPj.fromUrl(image)
        binding.tvNombre.text=nombre
        binding.tvGenero.text=genero
        binding.tvEspecie.text=especie
    }
}
```

5-4 Como podemos ver en las imagenes invoca un metodo que tenemos alejado en un .kt llamado extension.

```
fun ImageView.fromUrl(url:String){
    Picasso.get().load(url).into(this)
}
```

5-5Por últimoe estos son los componentes del CardView



6-También tenemos un metodo para esconder el teclado

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

7-Aquí lo que tenemos es el metodo que coge la URL base de nuestra api,la cual luego podremos añadirle un numero al final de la url para que coja un PJ determinado

```
private fun getRetrofit(): Retrofit {

    return Retrofit.Builder()
        .baseUrl( baseUrl: "https://rickandmortyapi.com/api/character/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
}
```

8-Aquí tenemos el metodo con más chicha el cual más tarde invocaremos en un metodo que reacciona cuando buscamos con el SearchView

```
private fun searchById(query: String) {
    CoroutineScope(Dispatchers.IO).launch {  this: CoroutineScope
        //Primero hacemos una llamadada con APIService la cual es una interfac en la que getDogs es el metodo
        val call = getRetrofit().create(APIService::class.java).getDogsByBreeds( url: "$query")
        //Luego el resultado de esa llamada lo guardamos en una variable
        val pjs = call.body()
        runOnUiThread {
            if (call.isSuccessful) {
                //Guardamos los datos de pjs en unos Strings
                val images: String = pjs?.images ?: String()
                val nombre2: String = pjs?.nombre ?: String()
                val especie: String = pjs?.Especie ?: String()
                val genero: String = pjs?.genero ?: String()
                //Limpiamos lo que había anteriormente
                dogImages.clear()
                nombre.clear()
                species.clear()
                gender.clear()

                //Los añadimos a las listas creadas arriba
                dogImages.add(images)
                nombre.add(nombre2)
                species.add(especie)
                gender.add(genero)

                //Y le decimos al adapter del recycleviwer que han cambiado los datos
                adapter.notifyDataSetChanged()
```

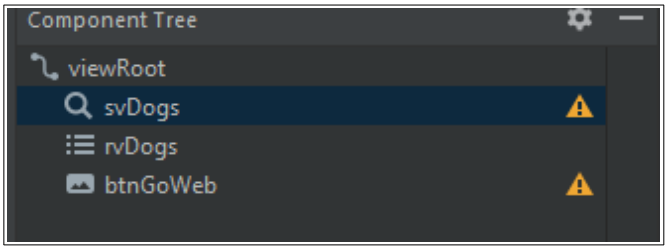
9-Por último,lo que queda en este metodo son un par de comprobadores de que le pasamos un número

```
} else {
    //show error
    var numeric = true
    try {
        val num = parseInt(query)
    } catch (e: NumberFormatException) {
        numeric = false
    }

    if (!numeric){ showError( error: "Tienes que poner un numero entero BOBO")

    }else{showError( error: "El numero debe estar entre 1-826")}

}
hideKeyboard()
```

10-Por último tenemos el metodo de error,el metodo de cuando buscamos con el searchView que acciona el Searchbyid y un metodo que no utilizamos.

```
private fun showError(error:String) {
    Toast.makeText( context: this, error, Toast.LENGTH_SHORT).show()
}

override fun onQueryTextSubmit(query: String?): Boolean {
    if (!query.isNullOrEmpty()) {
        searchById(query.lowercase())
    }else{showError( error: "Merluzo,introduce una ID de PJ")}
    return true
}

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

11-Aquí podemos ver la estructurá del XML del MainActivity , no tiene nada que resaltar en el código simplemente es copiarlo,tiene el searchview, el recycleviewer y un btn



12- Tal como se indica el paso 8 en la variable llamada se indica que se utiliza una interfaz,voy a seguir el recorrido,lo primero es la interfaz en si,la cual realiza un Get/Response de una clase que hemos creado

```
import retrofit2.Response
import retrofit2.http.GET
import retrofit2.http.Url

interface APIService {
    @GET
    suspend fun getDogsByBreeds (@Url url: String): Response<PjResponse>
}
```

13-Lo que hay dentro de esta clase es MUY simple,lo unico que utilizamos unas etiquetas @SerializedName ,dentro de este parentesis esta el nombre TAL CUAL de el campo que cogemos

```
import com.google.gson.annotations.SerializedName

data class PjResponse(
    @SerializedName( value: "status") var status: String,
    @SerializedName( value: "image") var images: String,
    @SerializedName( value: "name") var nombre: String,
    @SerializedName( value: "species") var Especie: String,
    @SerializedName( value: "gender") var genero: String
    )
```