Programación multimedia y dispositivos móviles

UT-4. Servicios API Rest. Recycler. Paginación

https://developer.android.com/guide/
https://github.com/stars/JulioHornos/lists/móviles-publico

Paginación en un Recycler

En muchas ocasiones cuando hacemos un Recycler, sobre todo si estamos accediendo a un servicio API a recuperar datos, nos puede interesar que haya paginación y no se recupere toda la información de golpe.

Para hacer la paginación en un Recycler necesitamos básicamente:

- Saber si la consulta que estamos haciendo incluye paginación, esto es, que hay elementos que cumplen con nuestras restricciones pero no los hemos recuperado.
- Saber cuántas páginas hay en total y en cual estamos.
- Saber cuando estamos viendo en el Recyler los últimos elementos del Recycler para así poder paginar.

Paginación en un Recycler

Normalmente cuando hay paginación es el propio servicio API consultado el que tiene la paginación, el que recupera un subconjunto de datos e indica cuántos datos hay en total, cuál es el total de páginas.

En este caso el API que tenemos no pagina por si mismo, así que voy a hacer yo la parte de paginación del API porque lo que realmente me interesa es ver la paginación del Recycler.

Paginación en un Recycler - Intercambio de datos

Me creo otro data class que además de la información que ya teníamos que nos mandaba el APi, status y el mensaje con las urls de las fotos, vamos a guardar la página actual y el total de páginas

Datos.kt × | MainViewModel.kt | MainActivity.kt | build.gradle.kts (:app) | activity_main.xml | MyAdapter.kt

```
package com.example.myapidogkotlin.model

class Datos (var status: String, var numPaginas: Int?, var paginaActual:Int?, var message: MutableList<String>? ) {
}
```

Paginación en un Recycler - State ó Model

Aquí voy a meter la lógica de la paginación (que como digo es algo que debería venirme dado desde el API). Me creo un atributo que va a ser la respuesta de la consulta:

```
class MainState {
    var cadena = "https://dog.ceo/api/breed/"
    lateinit var fotosPerrosCargado :  DogRespuesta
```

La idea es que este atributo se cargará al consultar con toda la información. Al hacer scroll iré incorporando a un objeto de tipo Datos las urls de 10 en 10.

Dentro del método que recupera las fotos, meto la lógica para calcular el número de páginas totales y siempre, al recuperar fotos, empezaremos por la página actual = 1.

Paginación en un Recycler - State ó Model

```
val call = retrofit.create(DogAPIService::class.java).getFotosPerros()
val fotosPerros = call.body()
if(fotosPerros!=null){
    fotosPerrosCargado = fotosPerros
    var misDatos : Datos? = null
    if (fotosPerrosCargado.message!!.size> 0){
        var numPaginas :Int  = fotosPerrosCargado.message!!.size/10
        if (fotosPerrosCargado.message!!.size%10!=0) numPaginas++
        misDatos = Datos(fotosPerrosCargado.status, numPaginas, paginaActual: 1, mutableListOf())
        var rango = Math.min(fotosPerrosCargado.message!!.size-1,9)
        for (i in 0 ≤ .. ≤ rango){
            misDatos.message!!.add (fotosPerrosCargado.message!!.get(i))
        }
    }
    return misDatos!!
}else{
    return Datos( status: "no success", numPaginas: null,  paginaActual: null,  message: null )
}
```

Guardo todo lo que recupera la consulta

Cálculo el total de páginas suponiendo que voy de 10 en 10

Si el resto no es 0, hay una página más para el pico

Al cargar fotos, siempre estoy en la página 1

Y cargo las que haya, que serán 10 si hay más de 10 y el número que haya si hay menos

Paginación en un Recycler - ViewModel

El cambio es muy sencillo, simplemente cambio el tipo de dato del LiveData, en lugar de un objeto de tipo DogRespuesta será un objeto de Datos.

```
class MainViewModel : ViewModel() {

    val myEstado = MainState()
    private val _datos  = MutableLiveData<Datos>(Datos(null.toString(), numPaginas: null, paginaActual: null, ArrayList()))
    val datos: LiveData<Datos> get() = _datos
```

Paginación en un Recycler - ViewModel

El cambio es muy sencillo, simplemente cambio el tipo de dato del LiveData, en lugar de un objeto de tipo DogRespuesta será un objeto de Datos.

```
class MainViewModel : ViewModel() {

    val myEstado = MainState()
    private val _datos  = MutableLiveData<Datos>(Datos(null.toString(), numPaginas: null, paginaActual: null, ArrayList()))
    val datos: LiveData<Datos> get() = _datos
```

Paginación en un Recycler - Adpater

El cambio vuelve a ser sencillo, simplemente cambio el tipo de dato de tipo DogRespuesta a objeto de Datos.

```
class MyAdapter (private val dataSet: Datos) : RecyclerView.Adapter<MyView>() {
```

Paginación en un Recycler

Ahora mismo, sin más cambios, tendríamos hecha la lógica de que se carguen en la consulta las 10 primeras (o las que haya si hay menos de 10 fotos). Nos queda implementar la lógica para que se pueda ir haciendo scroll.

Vamos a crear un servicio nuevo en el State o Model. Es bastante sencillo, será el sevicio al que se llame al cuando se quieran cargar más fotos. Simplemente debe saber en qué página está y cargar las 10 fotos siguientes.

Se calcula el inicio y fin de las nuevas páginas a cargar donde hay que tener en cuenta que cuando llego a la última página, igual no hay 10 fotos.

Paginación en un Recycler - State ó Model

```
fun scrollFotos(misDatos: Datos): Datos{
    var inicio: Int
    var fin: Int
    inicio = misDatos.paginaActual!!*10
    misDatos.paginaActual = misDatos.paginaActual!! + 1
    if(misDatos.paginaActual!! < misDatos.numPaginas!!){
        fin  = (misDatos.paginaActual!! *10 - 1)
    }else{
        fin  = (fotosPerrosCargado.message!!.size -1)
    }
    for (i in inicio ≤ .. ≤ fin){
        misDatos.message!!.add (fotosPerrosCargado.message!!.get(i))
    }
    return misDatos
}
```

Ojo al recorrer el array, hay que tener en cuenta las posiciones finales

Paginación en un Recycler - ViewModel

En el Viewmodel vamos a hacer el servicio de scroll, pero no hace falta que definamos tipos nuevos ya que seguiremos trabajando con actualizaciones de datos.

```
fun scrollFotos(misDatos: Datos){
    viewModelScope.launch { this: CoroutineScope
        _datos.value = myEstado.scrollFotos(misDatos)
    }
}
```

Paginación en un Recycler - View

En la actividad si hay que hacer algunos cambios más. Empezaremos por el más sencillo, y es que en el observador que tenemos ya actualmente sobre la variable datos voy a incluir la lógica para si estoy en la página 1 cargar el Recycler, pero si estoy en otra página, actualizarlo.

Recordemos: siempre que cargo fotos nuevas la página actual será la página 1.

He metido una variable llamada cargaInicial. Su objetivo es que la primera vez que entro en la pantalla, que en realidad aun no he buscado, no me salga el mensaje de que no hay datos. La inicializo a 0 y en cuanto hago una búsqueda la pongo a positivo. El mensaje de que no hay datos solo sale si estoy en positivo (no estoy en la actualización que hace al entrar en la pantalla). Seguramente hubiera sido mejor hacer un booleano aquí.

También me guardo los datos recuperados. Los necesitaré para hacer scroll.

Paginación en un Recycler - View

```
myViewModel.datos.observe( owner: this@MainActivity){ it: Datos!
    if(it.status=="success"){
        if(it.paginaActual==1){
            misDatos = it
            myAdapter = MyAdapter (it)
            //Y el adaptador se lo asignamos al recycler.
            rvPerros.adapter = myAdapter
        }else{
            myAdapter.notifyItemRangeInserted( positionStart: it.paginaActual!!*10 , it.message!!.size )
        }

    } else {
        if (cargaInicial!=0)
            Toast.makeText(applicationContext, text: "No hay fotos de esa raza", Toast.LENGTH_SHORT)
                .show()
    }
    if (cargaInicial==0) cargaInicial++
}
```

Cargo el Recycler siempre que la página es 1 porque estoy en la carga inicial de esa búsqueda

Guardo en un objeto los datos recuperados. Los necesitaremos para hacer scroll

Para siguientes búsquedas, simplemente aviso al Recycler que he insertado un rango nuevo.Le tengo que dar el inicio del rango nuevo y el fin del rango nuevo

Paginación en un Recycler - View

Por último para poder hacer scroll tenemos que capturar un evento nuevo: **addOnScrollListener**

Este evento se llama cada vez que haces scroll. Además en el objeto Layout que utilizamos para cargar el Recycler tenemos una función que me indica cual es la última posición que está visible, **findLastVisibleItemPosition.**

Vamos a codificar este evento para que si estamos en una posición cuya divisón entre 10 deje un resto mayor o igual que 9 (la última) y si NO estamos en la última página, sacaremos un snackbar avisando que hay más fotos disponibles.

Este snackbar tendrá un botón de acción para cargar más fotos y este botón lo único que tiene hacer es llamar a a la función scroll del ViewModel.

Paginación en un Recycler - View

```
rvPerros.addOnScrollListener(object : RecyclerView.OnScrollListener() {
    override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
        super.onScrolled(recyclerView, dx, dy)
        var finalSroll : Boolean = false ;
        if (mLayout.findLastVisibleItemPosition()%10 >=9&&
            mLayout.findLastVisibleItemPosition()/10==(misDatos.paginaActual!!-1)){
            finalSroll = true
        }
        if(finalSroll){
            Snackbar.make(main, text: "Si desea recuperar más fotos pulse: ", Snackbar.LENGTH_LONG)
                .setAction( text: "Cargar más fotos", View.OnClickListener { it: View!
                    myViewModel.scrollFotos(misDatos)
                } )
                .show()
        }

    }
})
```