

ReciclerView

Recicler View

1.- A on vaja el recycler afegirem el Widget **androidx.recyclerview.widget.RecyclerView**
🔥 **Nota:** No hem de oblidar posar-li el **id**

2.- Definirem un POJO o DTO (data class) per a modelitzar les dades **MyModel**
🔥 **Nota:** Si es un mock al hem de fer-ho al **init**

3.- Definirem una classe Controller o Provider que em torne un **listOf** o **mutableListOf** amb les dades a omplir al recycler.

4.- Definirem un **RecyclerView.ViewHolder** encarregat de dibuixar cada cella del meu recycler.

4.1.- Definirem un recurs **layout** que contindrà les dades de un objecte de mi model.
Podem anomenar-ho **item_myModel_layout.xml**
🔥 **Nota:** No oblidar **layout_height="wrap_content"**

4.2.- Definirem una classe:
class MyModelViewHolder(v : View) : RecyclerView.ViewHolder(v)
🔥 **Nota:** El holder també podrà rebre els 'callbacks' que se executaran al produir-se els events dels widgets de la vista del holder.
class MyModelViewHolder(..., private val onClickWidget: (String) -> Unit)

4.3.-Definirem propietats privades per que al **init {}** inicialitzarem amb referencies als objectes que representen el Widgets a establir a la nostra vista.
🔥 **Nota:** No serà necessari, si activem **viewBinding** podem invocar al mètode **bind** la associació amb **ItemMyModel_layoutBinding.bind(v)**

4.4.-Dins de **MyModelViewHolder** definirem un mètode public anomenat:
fun bind(data: MyModel) encarregar de associar le propietats del model al widgets a la meua vista que hem definit a **item_myModel_layout.xml** i m'arriba pel constructor.
🔥 **Nota:** Per això, usarem les referencies establides al pas 4.3 o el binding en el seu cas.

4.4.-Definirem els menejadors, si fos necessary per als Widgets cridant al callback corresponent (**onClickWidget**) rebut al constructor.

5.1.- Definirem una classe:
class MyModelAdapter(propietats) : RecyclerView.Adapter<MyModelHolder>()
Les **propietats** típiques seran:
• Dades per a omplir el Recycler → **private val datos: List<MyModel>**,
• Context (opcional) → **private val context: Context**,
• Callback per al 'holder' (opcional) → **private val onClickWidget: (DadesWidget) -> Unit**,
• Callback per a pulsació 'item' (opcional) → **private val onClickItem: (MyModel) -> Unit)**

5.2.- Invalidem el mètodes per heretar de **RecyclerView.Adapter<..>()**
• **getItemCount()**: retornarà la grandària de la llista de dades.
• **onCreateViewHolder()**: Unflarà la vista i retornarà una instància de **MyModelViewHolder**.
• **onBindViewHolder()**: Obtindrà el objecte corresponent al item actual de la llista i cridarà al mètode bind del holder. A més, amb...
- **holder.itemView** podrem accedir a la vista del holder i establir el callbacks als seus events.
- **holder.adapterPosition** podrem accedir a posició en el recyclerView avanç de generar cap notificació.

5.- Definirem un **RecyclerView.Adapter** encarregat d'omplir i actualitzar el recycler amb el nostre dades.

6.- A la **MainActivity**

6.1- Obtindrem una instància del **DAO** (objecte ams les operacions de CRUD) o al **ViewModel** si fós necessari.
🔥 **Nota:** En cas de ser tindre un **ViewModel** el serà el encarregat de tindre una instància del DAO i gestionar-ho.

6.2- Obtindrem una referencia al objecte **RecyclerView (rv)**

6.3- Definirem ael callbacks a passar al **MyModelAdapter**, manejant operacions amb les dades i notificant de canvis al Recycler. **rv.adapter?.notifyItemRemoved(recyclerView.getChildAdapterPosition(vistalItem)).**

6.4- Assignarem una instància de **MyModelAdapter** a **rv.adapter = MyModelAdapter(...)**

6.5- Assignarem una instància de gestor de layout (Linear, Grid, ...) al **rv.layoutManager = layoutManager(this, LinearLayoutManager.VERTICAL, false)**
• Establirà com s'organitzen el items dins del **RecyclerView**
• A més amb **scrollToPosition...** em permetrà mourem a una posició dins del layout.

Multi-selecció d'items

