Android MVI architecture with Jetpack & Coroutines/Flow

Kotlin Coroutines & Flow

Structured concurrency and Kotlin Flow that can essentially replace the RxJava way we used to work, in order to transform the data between our architecture's layers, to a stream. That could be either a bi-directional or a uni-directional fashion, depending on the needs of each project.

Creating a base RecyclerView adapter with ViewBinding

In order to create a base adapter for all of our RecyclerViews in the app, we decided to make use of the newly released ViewBinding as well and use ListAdapter's functionality to implement diffing quite easily.

Creating a base adapter item

By doing this we will be able to have a single interface to implement and use as common configuration for our adapter's items.

```
interface ViewBindingAdapterItem {
    val itemViewType: Int
}
```

Creating a base ViewHolder

Our base ViewHolder will be extended from each ViewHolder in each of our adapters.

```
import androidx.recyclerview.widget.RecyclerView
import androidx.viewbinding.ViewBinding

abstract class ViewBindingViewHolder<Item : ViewBindingAdapterItem, out VB : ViewBinding>(
    protected val binding: VB
) : RecyclerView.ViewHolder(binding.root) {
    abstract fun bind(item: Item)
    open fun bind(item: Item, payloads: List<Any>) {
        if (payloads.isEmpty()) {
            bind(item = item)
        }
    }
}
```

Creating a base adapter

Our base adapter will extend from ListAdapter since we want to use DiffUtil for diffing in our RecyclerViews.

```
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.ListAdapter
import androidx.viewbinding.ViewBinding

abstract class ViewBindingAdapter<Item : ViewBindingAdapterItem, VB : ViewBinding>(
    diffCallback: ViewBindingDiffUtilCallback<Item>
) : ListAdapter<Item, ViewBindingViewHolder<Item, VB>>(diffCallback) {
    override fun onBindViewHolder(holder: ViewBindingViewHolder<Item, VB>, position: Int) =
        holder.bind(item = getItem(position))

    override fun onBindViewHolder(
        holder: ViewBindingViewHolder<Item, VB>,
        position: Int,
        payloads: MutableList<Any>
    ) {
        holder.bind(item = getItem(position), payloads = payloads)
    }

    override fun getItemViewType(position: Int): Int = getItem(position).itemViewType

    protected val ViewGroup.layoutInflater: LayoutInflater
        get() = LayoutInflater.from(this.context)
}
```

Creating a base DiffUtil callback

Our base DiffUtil callback will extend from **`DiffUtil.ItemCallback`** and use **`ViewBindingAdapterItem`** as a generic input.

```
abstract class ViewBindingDiffUtilCallback<Item : ViewBindingAdapterItem> :
    DiffUtil.ItemCallback<Item>()
```

We can now create as many adapters as we want with minimal effort by just extending the already created base components :)