## Flexible UI with Airbnb Epoxy Library

To create a flexible UI usually we use Recycler view, which allow us to load multiple contents at the same view, even adding a scroll view if the number of contents exceed the screen size.

However, for this app we will be using the Airbnb Epoxy library, which is a library built on top of Recycler view, making it easier to use and understand, reducing boilerplates and adding more useful functionalities.

To start we had to import the library to our app gradle file.

def epoxyVersion = "4.6.4"  
implementation "com.airbnb.android:epoxy:$epoxyVersion"  
kapt ("com.airbnb.android:epoxy-processor:$epoxyVersion")

Their readme is really detailed on how to install all the necessary components: <https://github.com/airbnb/epoxy>

Also, we added to our project their ViewBindingKotlinModel class found at “epoxy/kotlinsample/src/main/java/com/airbnb/epoxy/kotlinsample/helpers/ViewBindingKotlinModel.kt”

This class is a helper available from Airbnb to use Epoxy with ViewBinding, which we will be using soon.

**Activity\_main.xml**

To make our flexible UI, we will separate each part of the screen in sections and using Epoxy we will be able to build the screen calling each separated part as many times as needed. For that, we took our activity\_main.xml and broke it in 4 pieces: The poster, the movie title, the movie overview, and the movie details.

For that, we just moved our code to different xml, and on the activity\_main.xml we created a EpoxyRecyclerView, which will be the controller for our flexible UI.

<com.airbnb.epoxy.EpoxyRecyclerView  
 android:id="@+id/epoxyRecyclerView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:paddingBottom="24dp"  
 app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"  
 />

**MovieDetailsEpoxyController**

This class as the name says, is a class created to control the EpoxyRecyclerView and populate the UI accordingly.

This class extends EpoxyController(), which implements a function to buildModels.

We also created 2 variables, one to verify if the content is loading, and other to hold the movie response.

The function buildModels verify the situation of the screen and updates it. Therefore, it checks if it’s loading, and if it is, it adds the model\_loading to the controller, loading that into the screen.

The model\_loading is a simple ContentLoadingProgressBar:

<androidx.core.widget.ContentLoadingProgressBar  
 style="?android:attr/progressBarStyleLarge"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:indeterminateOnly="true"  
 android:indeterminateTint="?attr/colorSecondary"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />

In case it’s not loading and it has a movie response, then all the models in reference to the movie is added to the controller.

For each model, a data class is created which defines what are the parameters that will be passed and what will be modified on the xml. For example, the Overview:

data class OverviewEpoxyModel(  
 val overview: String  
) : ViewBindingKotlinModel<ModelMovieOverviewBinding>(R.layout.*model\_movie\_overview*) {  
 override fun ModelMovieOverviewBinding.bind() {  
 overviewTextView.*text* = overview  
 }  
  
}

Then in the buildModels function, we get the OverviewEpoxyModel created above and add it to the controller, passing the movie overview loaded:

OverviewEpoxyModel(movieResponse!!.overview!!).id("overview").addTo(this)

The order in which each model is added to the controller, will define the order that it will be displayed on the screen.

Also, in case many models are added, the Epoxy automatically adds a scroll to the screen.

**MainActivity**

Now that the MovieDetailsEpoxyController is taking care of the data and the display, we update the MainActivity and remove all that commands to deal with the data observed.

Instead, every time the live data observes a change, it accesses the epoxyController and updates the movie response. After that the controller will be able to deal with all the UI.

Then to get it all working, we have to access the epoxyRecyclerView and set the controller instantiated.

val epoxyRecyclerView = findViewById<EpoxyRecyclerView>(R.id.*epoxyRecyclerView*)  
epoxyRecyclerView.setControllerAndBuildModels(epoxyController)

With all that, we still have the same screen, but now we can easily change it and adapt it.