STATEMENT OF WORK

Rest Countries Challenge

Overview

Display a list of Countries, their language and currency with a swipeable list to delete each item.

Instruction sets:

1. Display a list of Countries
2. Create a Swipeable list to enable delete on swipe
    a. Add purple background to swipe
    b. Swipe item anchor point
    c. Delete on swipe passed anchor
    d. Stick when slow swipe passed anchor

Implementation

The project follows an MVVM adaptation using the newly released Android Architecture components. The project uses the Android **ViewModel** for saving state across configuration changes and **LiveData** for getting observable streams and reacting to data changes when the app is in a suitable life cycle for modifying the user interface.

Networking

The project features a **NetworkUtils.java** class which provides a method called **makeRequest** that takes in a Url and sends a GET request to the server which then returns an InputStream provided from the connection. This class is based of the **java.io** and **java.net** packages which are bundled with the android sdk.

Instruction set #1:

Inorder to display the list of countries, our Activity class (**MainActivity.java**) creates an instance of the Android viewmodel and makes a request to the countries endpoint with the **getCountryLiveData** method from our viewmodel. This method returns an observable which we attach to our views LifeCycle so if at any point our view is out of focus the data will not be delivered to the activity but will wait till the view is in a receivable state, then subsequently when or if the view is in a receivable state and data is retrieved from the endpoint, the observable interface method **onChanged** is called then we set the data to our RecyclerView adapter and call **notifyDataSetChanged** in order to tell the adapter to update our view.

Instruction set #2 A:

To add the swipeable ability to our list items with a background to our list we first of all, add an interface called **TouchHelpListener** from our custom **ItemTouchHelper** class called **RecyclerTouchHelper** which allows us to listen to swipe events on each list item and react accordingly with appropriate animation and behaviour.

As for the list item having a background, I used a parent framelayout with two views taking up the same height and width. I did this so when you swipe, the translation x is set according to the change in x(dx) on the top view *(i.e framelayout sets views z index based on view hierarchy so any view below another would translate to a higher index)* then this would show the other view with a purple background. The contextual icon is also set in place within our xml on the back view so when you swipe the icon is already displayed.

Instruction set #2 B:

A default anchor is already put in place for swipeable interfaces, which is **0.5f,** now for us to set operations on our anchor point, we use a value 1 off from the default anchor which would then be 0.4f. This is done, so as to give the user a natural feel to the swipe as implemented in other apps.

Instruction set #2 C:

When a user swipes quickly, passed the 0.5f anchor point, a custom interface method of **onSwiped** is called which is implemented in the activity. This method takes in the viewHolder and the direction of swipe. The viewholder of the currently swiped view is used to get the item position and pass that into the adapter method remove, which takes care of deleting the item from the adapters dataset and reanimating the views with the modified list.

Instruction set #3 D

Making the swiped item stick at the anchor point required a bit of modification to the swipe logic, this involved creating some variables which I used to determine when a user is slow swiping or not. These variables and implementation are:

1. **SwipeBack:** denotes that the currently swiped view should become stuck when the user releases their hand from swiping instead of it, deleting. I set this value to true when the user when the user releases their finger from the phone screen because the scroll events up and down where triggering swipe callbacks so it was necessary for us to be able to differentiate between these two events so we can act accordingly.
2. **Swipe:** This variable is false by default and is only set to true when a user is just about crossing the swipe threshold. If this is true, we proceed with the swipe and delete the item from the list but if the user maintains a slow swipe pass our anchor point, the view is given a translationx value of -200 which keeps it at a halfway stuck position showing the purple background with the delete hint symbol. Because of how swipe actions are handle by android, when a user slow swipes pass the anchor point and we set the translationx to -200 and the user continues their swipe, the **onSwipe** method will not be called again because the threshold for the swipe has moved relative to the start point of the swipe, hence giving the effect of slow swipes returning the item to the halfway point(-200).
3. **Scroll:** This variable is used to stop the view from swiping during scroll movements, and it is used with the **SwipeBack** variable. This variable set to true when a scroll event is detected by our ScrollListener callback and it's set to false on every **touchdown or touch up** event as a safeguard to assert that it only captures scrolls.