Room

Per poter salvare in locale i dati dell’applicazione, abbiamo utilizzato un database implementato utilizzando la libreria Room.

Il primo passo è definire una classe astratta che estenda RoomDatabase, da noi chiamata AppDatabase. Questa classe utilizzerà il pattern singleton per consentire un unico punto di accesso al database, istanziato mediante il metodo Room.databaseBuilder.

Questa classe verrà utilizzata successivamente per chiamare i diversi Dao che abbiamo realizzato, qui dichiarati in maniera astratta.

Il secondo passo è definire le entità. Per ogni entità necessaria creiamo una classe Entity (ossia model), come ad esempio HeroEntity. In questa classe definiamo il mapping tra i campi del database e gli attributi java mediante l’annotazione @ColumnInfo da aggiungere in ogni attributo. Definiamo inoltre il mapping tra nome della classe e nome della tabella con @Entity, e la chiave primaria con @PrimaryKey. Non è necessario nel nostro caso avere una chiave di tipo AutoIncrement, per cui non aggiungiamo ulteriori annotazioni.

Fatto ciò, abbiamo definito i Dao per le entità, che ci consentono di accedere alle tabelle mediante interrogazioni. I Dao sono delle interfacce annotate da @Dao, e contengono solamente la firma del metodo. Sarà compito di Room implementare queste operazioni con opportuni metodi. Ogni operazione deve avere una annotazione che può essere @Insert, @Update e @Delete in caso di inserimenti, aggiornamenti e cancellazioni, o @Query (seguite dal testo della query) in caso di query personalizzate.

Infine, il codice per eseguire una query sarà del seguente tipo, nel caso di un inserimento:

AppDatabase.getInstance(getActivity().getApplicationContext()).heroDao().insertHero(hero);

Fragments

Per permettere all’applicazione di essere dotata di un menu di navigazione, è stato necessario implementare i fragment.

Per ciascun fragment abbiamo dichiarato il relativo layout (es. fragment\_search.xml) e la classe java di riferimento (es SearchFragment.java).

Per assegnare alla classe java il relativo layout utilizziamo il metodo Inflater.inflate()

Questi layout vengono iniettati nell’activity\_mail.xml attraverso il FrameLayout avente come id fragment\_container.

A questo punto, nella MainActivity gestiamo il listener relativo all’elemento BottomNavigationView, e mediante un costrutto switch-case andiamo ad attivare il fragment desiderato.

Per attivare il fragment bisogna iniziare una FragmentTransaction, e con il metodo fragmentTransaction.replace() andiamo ad impostare nella transazione di sostituire il fragmento attuale contenuto nella fragment\_container con quello selezionato. Con un commit confermiamo l’operazione.

Animazioni

Le animazioni sono dichiarate in file xml e sono contenute nella cartella res/anim.

Il tag apertura di una animazione è <set>, contenente l’attributo duration (in millisecondi), e al suo interno può contenere tag relative ai possibili attributi di un elemento per cui possiamo modificarne il valore. Alcuni esempi sono translate (posizione nello spazio), alpha (trasparenza), scale (dimensioni). Ogni tag contiene elementi che definiscono il valore iniziale, il valore finale e il tipo di interpolazione.

Un esempio pratico di animazione è quello di anim\_zoom\_in.xml, in cui animiamo l’effetto con cui i risultati della ricerca sono presentati all’utente.

In questa animazione combiniamo gli effetti dati da translate, alpha e scale, ottenendo la seguente animazione: l’elemento, rispetto al suo stato finale, comincia la sua animazione con una posizione del 20% più in basso lungo l’asse Y, con una trasparenza impostata a 0.5 e una dimensione del 2% piu piccola. Successivamente, attraverso una interpolazione di tipo decelerate, completa la sua animazione in 200 millisecondi posizionando il suo offset lungo y a 0, la sua opacità come completa e la sua dimensione al 100%.

Il codice java utilizzato per riprodurre l’animazione è

holder.itemView.setAnimation(AnimationUtils.*loadAnimation*(appContext,R.anim.*anim\_zoom\_in*));

Presa la view desiderata, ne settiamo l’animazione con il metodo AnimationUtils.loadAnimation, e viene riprodotta esattamente in quel momento.