Mobile Programming

# Glide

Glide è una libreria open-source che si occupa della gestione dei contenuti multimediali, ed in particolare è un framework per il caricamento delle immagini in ambiente Android.

Il suo ruolo principale è quello di rendere trasparente al programmatore il decoding dei media, il caching su disco e memoria principale, e la gestione delle risorse attraverso l’uso di un’interfaccia semplice da utilizzare.

Nella nostra applicazione avevamo bisogno di un meccanismo semplice di caricamento e caching delle immagini offerte per gli eroi, i fumetti, gli autori e le storie dei personaggi, e la nostra scelta è ricaduta su questo particolare framework per la sua semplicità d’uso.

String urlThumbnail = comicOfTheDay

.getThumbnail().getPath().replaceFirst("http", "https")

+ "." +

comicOfTheDay.getThumbnail().getExtension();

Glide.*with*(getActivity())

.setDefaultRequestOptions(requestOptions)

.load(urlThumbnail)

.diskCacheStrategy(DiskCacheStrategy.*ALL*)

.into(ivComic);

Si può osservare dall’esempio in alto che in pochi passi è possibile implementare il prelievo delle risorse multimediali ed il caricamento nei relativi contenitori attraverso poche righe di codice, passando come parametri il *context* dell’*Activity*, il *URL* della risorsa, le modalità di *caching* da utilizzare, il riferimento al contenitore, ed un oggetto con le opzioni che possono essere condivise tra più richieste, che riguardano il ritaglio, lo zoom, e la possibilità di inserire dei *placeholder* nel contenitore prima del completamento della richiesta.

# Menù contestuale

Per permettere all’utente di eliminare dalla lista dei preferiti uno o più eroi salvati, senza dover aprire l’*Activity* di dettaglio, è stato implementato un menù contestuale con la singola voce “elimina”, gestito all’interno del FavouriteFragment.

Per l’attivazione del menù contestuale è stato utilizzato il meccanismo dell’*onLongClick* sugli elementi contenitori *view* degli eroi preferiti, implementato negli *Holder* che sono gestiti dal *FragmentAdapter*.

Ogni volta che viene effettuato un *long press* su uno degli eroi preferiti, appare il menù contestuale dove l’utente può selezionare o deselezionare altri elementi con un semplice tocco ed eliminarli tramite la voce del menù, che si chiude subito dopo l’eliminazione.

L’implementazione si basa sul meccanismo di *callback*, infatti, ogni elemento *view*, associato ad un eroe salvato nei preferiti, fa una chiamata tramite il riferimento *smListener* per iniziare la *ActionMode* ed attivare il menù contestuale. Osserviamo infatti che nella *onLongClick* che viene impostata su tutti i contenitori *view*, abbiamo:

if (smListener != null) {  
 smListener.onSelect(selectedHeroesList.size()); //callback verso l'Activity

Il parametro passato rappresenta il numero di elementi che sono stati selezionati, così da permettere la disattivazione del menù, quando questo numero arriva a 0.

L’attributo *smListener* viene impostato durante l’istanziazione di *FavouriteAdapter*, poiché viene passata l’istanza creata nella *FavouriteFragment*, dove la classe viene definita come:

public class SelectModeListener implements HeroSelectMode {  
 @Override  
 public void onSelect(int size) {  
 if (mActionMode != null) {  
 if (size == 0) {  
 mActionMode.finish();  
 mActionMode = null;  
 }  
 } else  
 mActionMode = ((AppCompatActivity)

Objects.*requireNonNull*(getActivity()))

.startSupportActionMode(hActionModeCallback);  
 }  
}

Si osserva da qui che il ciclo di vita del menù contestuale inizia e termina attraverso le chiamate di *callback* sul metodo *onSelect*. Tuttavia, può terminare anche nel caso in cui venga premuto il pulsante di *back*.

In tal caso, viene direttamente effettuata una chiamata al metodo *onDestroyActionMode* dell’attributo della classe *FavouriteFragment* che ha il nome di *hActionModeCallback*.

Quest’ultimo rappresenta l’istanza della classe astratta *ActionMode.Callback*, e la cui implementazione viene definita sempre all’interno della classe *FavouriteFragment*, che permette di definire le operazioni di creazione e disattivazione della barra in alto e l’eliminazione degli eroi dalla sezione dei Preferiti.

# Sectioned recycler view

La libreria SectionedRecyclerView permette di dividere facilmente gli elementi nella *RecyclerView*, la classe a cui vengono estese le funzionalità, in sezioni espandibili comprese del relativo *header*.

L’obiettivo era organizzare gli eroi preferiti in modo da facilitare la ricerca del personaggio filtrando per la prima lettera del nome.

I componenti principali offerti dalla libreria sono la classe *SectionedRecyclerViewAdapter*, ed i metodi astratti offerti da essa, che sono *onPlaceSubheaderBetweenItems*, *onBindItemViewHolder* e *onBindSubheaderViewHolder*.

Il primo metodo permette di determinare quando due sezioni vadano separate, e quindi quando è necessario inserire un *subheader* tra due elementi, nel nostro caso bastava confrontare le prime lettere dell’eroe dell’elemento corrente e del successivo e verificare se fossero diverse.

public boolean onPlaceSubheaderBetweenItems(int position) {  
 final char heroNameFirstCharacter = heroes.get(position).getName().charAt(0);  
 final char nextHeroNameFirstCharacter =

heroes.get(position + 1).getName().charAt(0);

return heroNameFirstCharacter != nextHeroNameFirstCharacter;  
}

Il secondo metodo permette di impostare i valori contenuti nell’elemento *view* dell’eroe, mentre il terzo riguarda ciò che deve essere visualizzato all’interno del *subheader*.