RELAZIONE MOBILE PROGRAMMING

RecyclerView

Le RecyclerView sono uno strumento molto utile quando c’è bisogno di organizzare una lista di informazioni secondo uno schema preciso. Nell’applicazione, ogni Model di tipo Hero, Comics, Series, Events o Creators ha altre liste di Model a lui collegate (es i fumetti, le serie e gli eventi collegati ad un eroe) che vanno quindi mostrati all’utente.

Ogni RecyclerView è un contenitore di singole View, che possono avere uno schema più o meno complesso, in questo caso le View hanno una struttura molto semplice e sono composte da un ImageView ed una TextView. Le View devono essere fornite da un LayoutManager, che nel progetto è quasi sempre un LinearLayoutManager con opzione di scorrimento orizzontale.

Inoltre, ogni elemento della RecyclerView viene gestito da un ViewHolder, i cui metodi vengono chiamati proprio dal LayoutManager quando è necessario creare una nuova riga nella RecyclerView o bisogna aggiornarne il contenuto.

Le liste da visualizzare sono di dimensione variabile, andando da 20/30 elementi fino anche a più di 50. La RecyclerView mantiene tutti gli elementi che non possono essere mostrati per questioni di spazio nel display in una apposita cache, così da andare a caricare i nuovi quando viene eseguito uno scroll per visualizzare gli altri elementi, o ricaricare i vecchi quando si vuole tornare verso l’inizio della lista. In questo modo, la gestione della memoria è più efficiente e la lista è completamente gestita dalla RecyclerView.

Le RecyclerView sono state introdotte pressoché in ogni Activity, specialmente in quelle che mostravano il dettaglio di uno specifico Model, quindi la HeroDetailActivity, ComicsActivity, SeriesActivity, EventsActivity e CreatorsActivity.

Adapter

Ogni elemento della RecyclerView è rappresentato mediante una View indipendente, con un preciso layout che deve essere gestito mediante appositi ViewHolder.

Ogni ViewHolder è a sua volta gestito da un Adapter, che svolge la funzione di ponte fra la UI e i ViewHolder.

L’Adapter svolge due compiti fondamentali nella gestione dei ViewHolder: assegna ad ognuno di essi una specifica posizione nella RecyclerView e vi collega i dati che dovranno essere gestiti.

Ogni Adapter gestisce una lista di Model, i cui dati saranno poi inseriti nelle View gestite dai ViewHolder.

Abbiamo scelto di avere un’apposita classe Holder per ogni Adapter, che estende la classe Recycler.ViewHolder e gestisce quindi gli elementi del layout. In questo modo, il codice è molto più mantenibile e leggibile.

Ogni volta che la RecyclerView ha bisogno di mostrare un nuovo elemento, viene invocato il metodo onCreateViewHolder() dell’Adapter, che crea per l’appunto un nuovo oggetto di tipo Holder per la gestione della View.

Il metodo di onBindViewHolder() nell’Adapter viene invece chiamato quando c’è bisogno di aggiornare il contenuto di una View, tale metodo ha come parametro l’Holder di cui va aggiornata la View e la posizione di quest’ultimo nella RecyclerView.

Infine vi è un metodo getItemCount() che ritorna la lunghezza della lista di Model che viene gestita dall’Adapter.

Ognuno degli elementi nella RecyclerView è inoltre cliccabile, infatti ogni Adapter implementa il metodo di onClick(), andando a caricare la schermata di dettaglio per l’elemento selezionato dall’utente.

Menu contestuale (lo farà Adrian)

Per permettere all’utente di eliminare dalla lista dei preferiti un elemento, senza dover per forza aprire l’activity di dettaglio, è stato implementato un piccolo menu contestuale con la singola voce “elimina”, gestito dal FavouriteFragment.

Per far attivare il menu contestuale è stato usato il meccanismo dell’onLongClick(), che è stato implementato negli Holder gestiti dal FragmentAdapter. Ogni volta che viene effettuato un long press su uno dei preferiti, viene mostrato il menu contestuale e a questo punto l’utente può selezionare / deselezionare altri elementi con un semplice press ed eliminarli tramite la voce del menu, che si chiude subito dopo l’eliminazione.

Caching con Shared Preferences

Per poter mantenere sempre identici i suggeriti del giorno, è stato necessario aggiungere un meccanismo di salvataggio su file, per fare questo la scelta è ricaduta sulle Shared Preferences.

Dovendo salvare tre Model, ovvero un eroe, un fumetto ed una serie, costituiti da varie informazioni, i Model vengono convertiti in un Json tramite l’API Gson, per poi essere riconvertiti ogni volta che viene caricata la Home Page.

Abbiamo usato il metodo getPreferences() per ottenere il file su cui salvare i Json,in quanto è necessario un solo file per contenere tutto e tale file viene resettato ogni giorno con i nuovi suggeriti del giorno.

Per il salvataggio è necessario usare un oggetto di tipo SharedPreferences.Editor, il metodi di scrittura possibili sono due: commit(), che è un metodo sincrono con valore di ritorno di tipo booleano per indicare se la scrittura è andato a buon fine ed il metodo apply(), che viene invece eseguito su un thread e non riscontra al main se la scrittura è avvenuta con successo.

Abbiamo deciso di utilizzare il metodo di apply(), in quanto il controllo sulla presenza del dato nel file di SharedPreferences viene effettuato solo quando l’utente ritorna sulla Home Page in un secondo momento (così da verificare anche che i dati non sono stati cancellati dall’esterno), inoltre in questo modo il salvataggio su file non pesa sull’ Home Page Fragment, non essendo una chiamata sincrona.

Il file di SharedPreferences viene resettato ogni giorno tramite il metodo clear() e ripopolato con i nuovi suggeriti del giorno.