Relazione progetto LAM

Lorenzo Borelli 0000789622

Settembre 2019

Contents

1 Scopo e funzionalità

Scheduler è un'applicazione per l'organizzazione e la pianificazione di task personali. Attraverso l'uso di un calendario come principale *entry point* dell'app, permette all'utente di ottenere fin da subito un feedback visivo sui task che ha registrato, potendo eventualmente modificarli o cancellarli a sua discrezione.
L'app fornisce la possibilità di visionare una lista di tutti i task registrati fino a quel momento, e di filtrarli attraverso una barra di ricerca. Esistono, inoltre, tre grafici che forniscono informazioni specifiche su attributi dei task dell'utente, in particolare:

- **Grafico a torta**: mostra la percentuale di task divisi per categoria
- **Istogramma**: mostra il numero di task per priorità
- **Grafico a linee**: mostra il numero di task completati nell'anno corrente

Gli attributi di un task sono nome, data e ora, priorità (*Urgent*, *Important*,*Secondary*), Tipo/Classe (customizzabile dall'utente) e stato (*Pending*, *Ongoing*, *Completed*)

L'app fa uso di un sistema di notifiche che avvisa l'utente ogniqualvolta è tempo di eseguire un task; a questo punto, l'utente può scegliere di posporlo, o marcarlo come *Ongoing*, implicando così che abbia intenzione di iniziarlo. Contemporaneamente, diventa possibile per l'utente, da quel momento in poi,

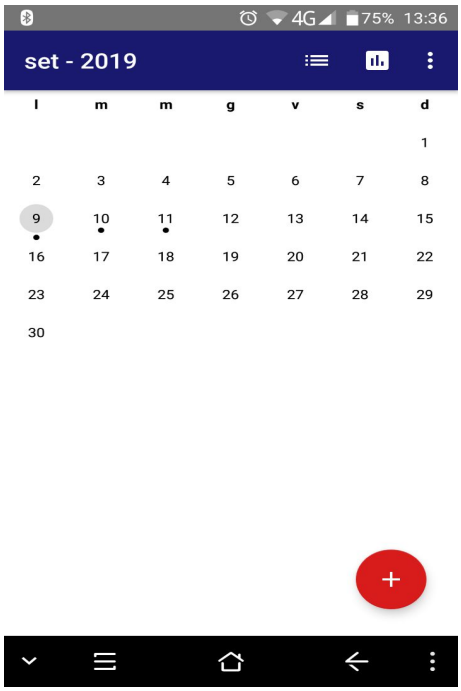

Figure 1: Calendar View

marcare un task come *Completed* anche senza passare dallo stato precedente: si lascia qui libertà all'utente di decidere come meglio agire.
Infine, le impostazioni dell'applicazione permettono di disabilitare le notifiche, che di default sono attive, con diversa granularità rispetto alla priorità delle stesse: disabilitare le notifiche per task di priorità n porta il sistema a disabilitare automaticamente anche notifiche per task a priorità minore. L'utente può anche personalizzare minimamente il calendario scegliendo di mostrare in una stessa schermata solo i giorni di un mese, o riempirla mostrando anche giorni dei mesi contigui.

1.1 Requisiti

L'app è sviluppata per sistemi android >= 6.0 Marshmallow, quindi con API level 23. Android Studio è stato scelto come ambiente di sviluppo.

2 Progettazione e scelte implementative

L'app è stata progettata seguendo, per la maggior parte, il pattern **MVVM**

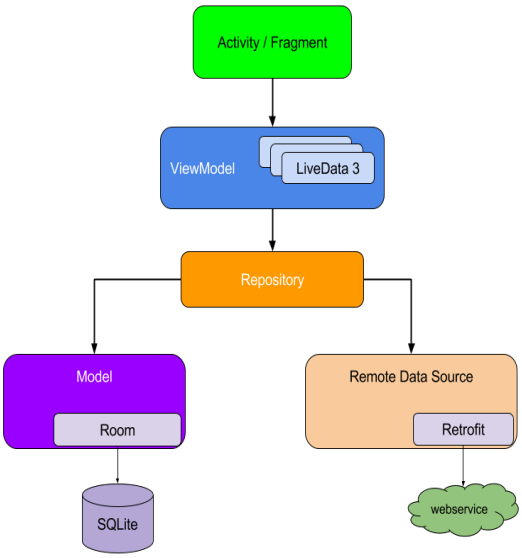

Figure 2: Model-View-ViewModel

2.1 Model

In questo caso, si è utilizzato un database Room con un'unica tabella *Tasks* che contiene tutti gli attributi di un task di un utente. Il nome e la data e ora (memorizzate come un'unica stringa per problemi di compatibilità con SQLLite e l'API android) sono state impostate come **primary key**. Si sono usate in più occasioni anche diverse **SharedPreference**, per le impostazioni utente, per esempio, o per memorizzare i nuovi tipi di task inseriti dagli utenti. Il database è stato implementato tramite *Singleton*, evitando così di avere più connessioni al DB da diversi oggetti. Si è implementata poi una classe **Repository** che si occupi del dialogo con la parte **Model** dell'architettura, centralizzando così tutte le chiamate al DAO così come le definizioni delle *AsyncTask* necessarie per eseguire le operazioni CRUD. Ogni ViewModel invoca i servizi del database tramite la repository.

2.2 View e ViewModel

Per quanto riguarda l'UI dell'applicazione, sono state implementate diverse *Activity*

- **MainActivity**: l'entry point dell'app, che contiene il calendario *CompactCalendarView* e un *FloatingActionButton* per la creazione di nuovi task. Si occupa principalmente di due compiti: osservare (tramite *LiveData*, un

observable di Android) cambiamenti nel database, aggiornando il calendario di conseguenza e aggiungendo le nuove date come eventi, e rispondere a click sui giorni che corrispondono agli eventi suddetti. Per controllare che il giorno del calendario cliccato sia effettivamente uno di quelli registrati, lo si compara con l'array di *Event* che appartengono alla *CompactCalendarView*. Cliccare su quei giorni porta a far apparire un *PopupMenu* che permette di editare, cancellare o segnare, se possibile, come completato il task di quella data, invocando lo stesso Fragment ma passandogli un Bundle diverso a seconda dell'azione richiesta. Poiché è possibile avere più di un'task per giorno, prima di completare l'operazione richiesta dall'utente, l'app mostra un *DialogFragment*, più precisamente **SelectTaskFragment**, che non fa altro che creare una *ListView* contenente i nomi di tutti i task di quel giorno, in modo che l'utente possa scegliere esattamente su quale eseguire l'operazione desiderata: aggiornare lo stato del db con lo stato *completed* se la data corrente lo permette, cancellare l'entry nel db, o, se è richiesto l'editing, fare una query al db per l'entry corrispondente al nome e data cliccati, entry che poi passerà con un Intent a *TaskActivity*(in questo modo l'activity presenterà i campi già riempiti con i valori correnti); poiché la query al db non viene fatta su *LiveData*, c'è bisogno di un *AsyncTask* che esegua la query in background e che recuperi poi i risultati con una callback: l'AsyncTask definisce un'interfaccia(la cui istanza viene inizializzata nel suo costruttore) con un metodo che sarà di fatto la callback che prende i risultati della query quando viene invocato nell'*onPostExecute* dell'AsyncTask, poiché il Fragment stesso implementa quella stessa interfaccia, può definire lo stesso metodo callback e ricevere i risultati della query. La *ListView* non conosce il proprio contenuto, perciò ne fa richiesta a un adapter, in questo caso *TasksAdapter* che si occupa di renderizzare (*inflate*) la view e settarne il contenuto testuale. Inoltre, quest'*Activity* contiene un menu, su un'apposita toolbar, che permette all'utente di usufruire della altre funzioni dell'app cliccando sul *MenuItem* corrispondente, in particolare: visualizzare la lista di tutti i task registrati, grafici che raccolgono statistiche su questi ultimi, e le impostazioni.

- **TaskActivity**: l'*Activity* che viene lanciata con un *Intent* per creare un nuovo task, e che propone all'utente un form per definirne gli attributi. In particolare, utilizza degli *Spinner* per decidere priorità e tipo, i quali settano il proprio contenuto con degli adapter; mentre le priorità dei task sono definite staticamente nelle risorse, il tipo è memorizzato in una SharedPreference che viene aggiornata ogni volta che si clicca sul bottone *New Type*, il quale usa un *AlertDialog* per permettere all'utente di inserire il nuovo tipo di task. La data e l'ora vengono impostati attraverso **SelectdateFragment** e **TimePickerFragment**, che rispettivamente implementano dei listener di *DatePickerDialog* e *TimePickerDialog*. Nel caso l'utente non definisca data, ora e nome, appare un messaggio d'errore e il task non viene salvato (perché essendo primary key almeno quei campi devono essere definiti). Questa stessa activity viene anche invocata quando l'utente

vuole aggiornarne uno: in questo caso, i campi sono già riempiti con i valori correnti, grazie al meccanismo del *SelectTaskFragment* spiegato prima. Quest'activity ha anche il compito di settare l'allarme per le notifiche. Una volta creato un task, oltre a ad aggiornare il suo ViewModel e inserire i nuovi campi nel db (o aggiornarli), deve settare il PendingIntent che verrà lasciato all'handler dell'allarme(la descrizione di un intent specificamente creata per essere lasciata a un altro oggetto da utilizzare a tempo debito). Il payload del PendingIntent sono gli attributi del task, e ha un id incrementale che viene memorizzato in un altra variabile SharedPreference. La scadenza dell'allarme è chiaramente data dalla data e ora del task e segna l'apparizione della notifica.

- **UsageGraphActivity**: mostra i tre tipi di grafico descritti prima, attraverso dei *tab* che permettono di navigare i tre rispettivi *Fragment*. La navigazione funziona grazie a un layout manager chiamato *ViewPager*, il quale si serve di un *PagerAdapter* per generare i contenuti dei tab, in questo caso *ChartPagerAdapter*, che tiene conto del numero di tab e genera quello corrispondente alla posizione attuale dell'utente. *TabPie* mostra un grafico a torta che divide tutti i task per tipo, usando un Observer per contare tutte le coppie (nome,tipo) e creando una mappa che associa ad ogni tipo il numero di task. Questo numero viene aggiunto in percentuale al grafico, generando per ogni nuovo tipo un colore casuale (in maniera dinamica, visto che i tipi possono aumentare). *TabLineChart* crea un grafico a linee in cui in ascissa abbiamo i mesi di un anno e in ordinata il numero di task completati in quell'anno. *TabHistogram* è un istogramma che mostra la distribuzione dei task per priorità. Tutti utilizzano degli Observer per prendere i dati.

- **ListTaskActivity**: una ListView di tutti i task registrati finora aggiornata tramite un Observer. Presenta una searchbar nel menù per facilitare il filtro dei task tramite i parametri che interessano all'utente, tramite l'adapter della ListView invocato da un listener delle query dell'utente.

- **SettingsActivity** istanzia **SettingsFragment**, che estende *PreferenceFragmentCompat*, la classe dell'API Android per gestire le impostazioni utente. Il Fragment usa delle *SwitchPreference* per le notifiche e la customizzazione del calendario.

La gestione delle notifiche passa attraverso il meccanismo del PendingIntent settato nella TaskActivity. Una volta scaduto il tempo, il broadcast receiver *AlarmReceiver* si occupa del setup della notifica, usando il Builder di *NotificationCompat*. Prepara gli intent per le diverse azioni che l'utente può intraprendere con la notifica: marcare il task come Ongoing (cosa che viene fatta in background tramite un Service inizializzato tramite Intent), posporlo, il che riporta all'activity *TaskActivity* o semplicemente toccare la notifica che riporta al main entry point.

3 Conclusioni

L'app permette un'organizzazione elaborata delle proprie attività, e permette anche una certa personalizzazione. Tuttavia, si potrebbe rendere più interattiva estendendo, per esempio, le notifiche, e facendo in modo che vengano inviate anche come promemoria, permettendo all'utente di ricordare gli impegni presi con un certo anticipo. Potrebbe essere inoltre utile un grafico che mostri il numero di task cancellati.