Relazione progetto applicazioni mobili

Tracking My Pantry

Presentata da: Lorenzo Tomesani

Email: lorenzo.tomesani2@studio.unibo.it

Dipartimento: Scienze Informatiche

Matricola: 0000881457

Anno Accademico 2020/2021

Sommario

Introduzione

Il progetto si basa sullo sviluppo di un'applicazione mobile, in questo caso android, per la gestione di una dispensa e degli articoli ad essa collegati.

Il progetto richiede la registrazione e il login dell'utente per l'esecuzione delle seguenti operazioni: rimozione, aggiunta, modifica di prodotti presenti nel proprio database locale/ nella propria dispensa. Per l'operazione di aggiunta viene fatta richiesta ad un server remoto dei possibili prodotti con un determinato barcode. Queste operazioni sono funzionalità base a cui ho aggiunto le seguenti:

- la possibilità di filtrare i prodotti in base alla tipologia, se inserita;
- impostare una data di scadenza dei prodotti e la ricezione della notifica che il prodotto è scaduto;
- modificare la quantità dell'articolo;
- inserire un prodotto tramite lo scan del suo barcode;
- ricercare tramite nome.

Struttura

a. Generale

Come struttura di partenza sono andato ad usare quella presentata in classe nell'esempio "Room – Word".
Sono quindi presenti le classi:

- ProductViewModel connette la View alla classe Repository tramite LiveData;
- Repository consente le azioni di aggiunta, eliminazione, modifica dei prodotti sia sul database locale che su quello remoto tramite API REST;
- ListAdapter fa il bind degli item nella lista con il ViewHolder e controlla se sono avvenute modifiche nella lista stessa;
- ViewHolder descrive l'item e la view connessa ad esso.

Per gli input dell'utente, come per esempio la modifica di un prodotto, ho usato classi che vanno ad estendere i DialogFragment. Queste classi sono Singleton, ovvero può esistere una sola istanza di esse. Per passare il risultato dal dialog all'attività, che sia esso un dato o un annullamento dell'operazione, sono andato a definire quattro interfacce le quali vengono implementate da una classe anonima che le estende e viene usata come "listener" di eventi.

b. Chiamate all'API REST E Database

è presente una classe denominata "ApiManager.java" che si occupa delle chiamate http al server. Per poter fare le chiamate sono andato ad usare la libreria "Volley". Ho creato un'interfaccia chiamata "VolleyCallback" che presenta due metodi: OnSuccess e onError. Essendo le chiamate di Volley asincrone, quest'interfaccia mi permette di chiamare il chiamante una volta che le richieste sono terminate.
Per il database ho usato Room grazie al quale tramite una semplice classe Java (Product.java) si riesce ad andare a creare una tabella senza la necessità di scrivere le Query di creazione di quest'ultima.

La tabella risulta essere così composta:

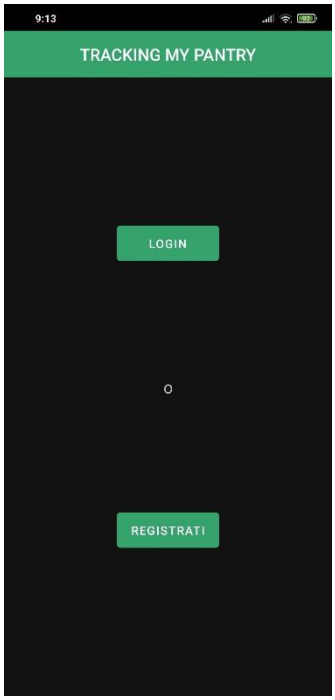
id	idProd	Name	Description	Barcode	idUser	Quantity	Date	Tag

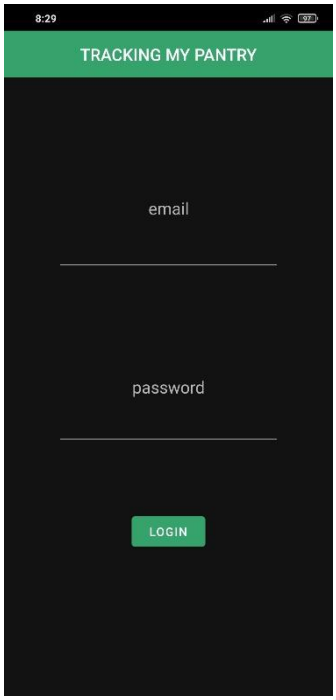
Per andare a descrivere le query di inserimento, cancellazione e aggiornamento delle tuple della tabella sono andato ad usare Dao (Data Access Objects).
La query per estrapolare i prodotti aggiunti da un utente risulta quindi essere:

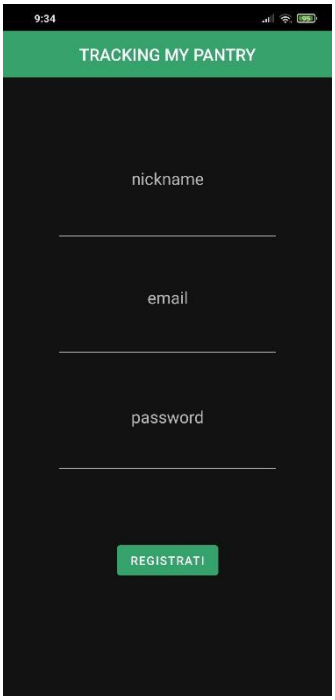
```
“Select * from products where idUser=:idUser”.
```

c. Attività Iniziale

L’interfaccia utente dell’applicazione è composta da due attività principali: l’attività iniziale e la dispensa. L’attività iniziale è quella che viene mostrata all’apertura dell’applicazione. Il layout è formato da un FragmentContainerView che occupa tutto lo schermo. I fragment che vengono passati in input al Container sono tre: il primo è composto da due bottoni su cui è presente un OnClickListener che al click del primo o del secondo esegue una transazione, usando la classe FragmentTransaction, verso il secondo o il terzo fragment; il secondo e il terzo invece sono usati per fare il login e la registrazione dell’utente. Per segnalare all’utente il successo o il fallimento della registrazione e del login appare un Toast il cui messaggio cambia a seconda dell’esito. Se il login avviene correttamente viene lanciata l’attività Pantry tramite un Intent. In questo caso le chiamate al server avvengono direttamente alla pressione del bottone anziché passare per il ViewModel, in quanto ho ritenuto che fosse eccessivo per le poche righe di codice scritto.

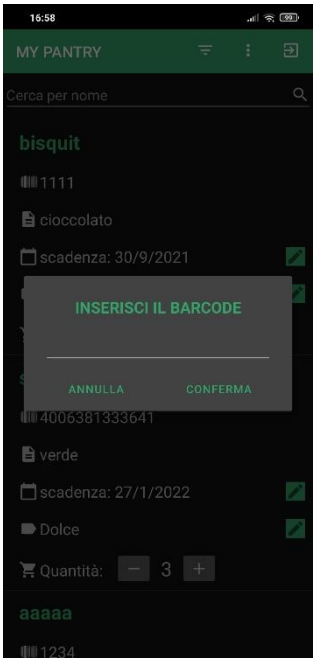


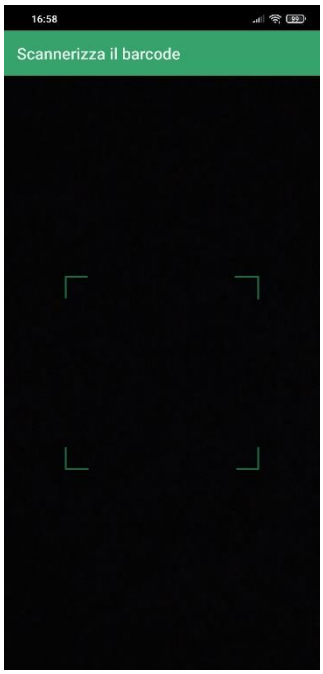




d. Dispensa

La seconda attività invece è l'attività principale che compone la dispensa vera e propria dell'utente. In essa troviamo una RecyclerView che è composta dai prodotti inseriti localmente, se presenti. La Toolbar è composta da 3 bottoni, uno per uscire, uno per andare ad aggiungere i prodotti e il terzo per applicare i filtri. Se non sono presenti prodotti un ViewText lo notifica all'utente. All'aggiunta di un nuovo prodotto il ViewText è reso invisibile, mentre appare la RecyclerView. Come si vede dallo screenshot affianco possiamo trovare in alto un EditText che permette di filtrare i prodotti tramite una ricerca per nome. In questo caso è presente nel ViewModel una variabile di tipo MutableLiveData che prende i valori della LiveData e filtra i prodotti che contengono lo stesso nome. Un observable nella View vede i cambiamenti e fa il submit della lista ricevuta passandola al ListAdapter. Il MutableLiveData viene anche usato per il filtro tramite tipologia.
È data la possibilità di aggiungere il prodotto sia inserendo direttamente il codice tramite input presente in un Dialog o scannerizzando un barcode. Per il barcode sono andato ad usare la libreria di google: "com.android.google.vision". Prima di aprire la fotocamera viene fatta richiesta all'utente del permesso di poter usare la fotocamera. Una volta che l'utente accetta i permessi, viene istanziato l'Intent di cambio dell'attività. I barcode disponibili sono EAN_13, EAN_8, UPC_A, UPC_E.

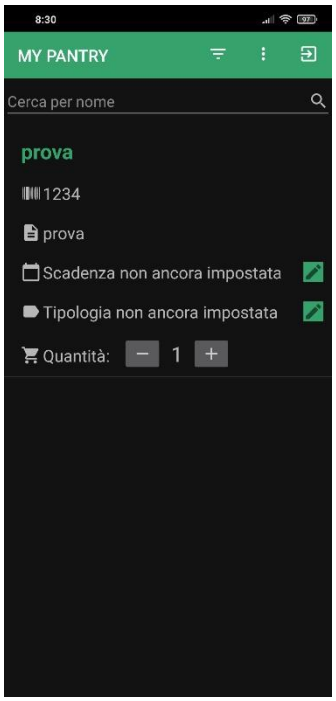


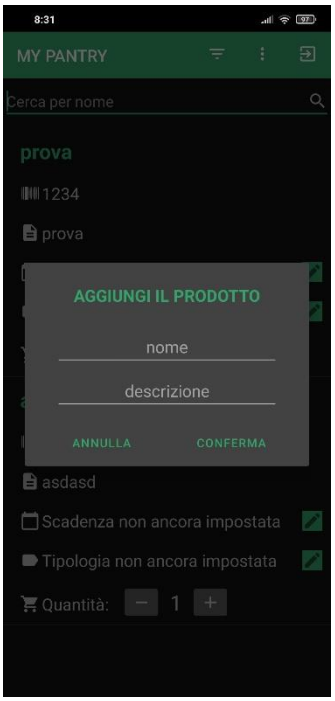


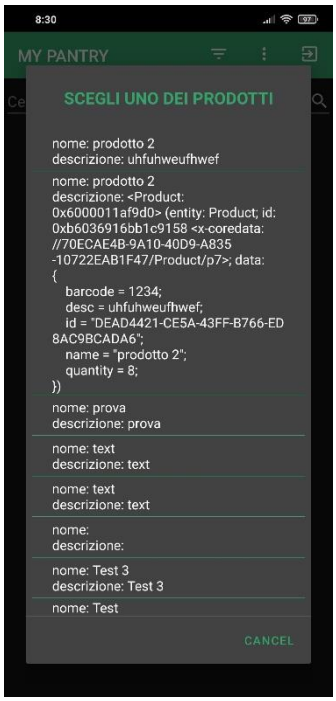
Nella schermata per scannerizzare il barcode è presente oltre che una SurfaceView anche un frameLayout che permette la visualizzazione del riquadro verde. Una volta inquadrato il barcode, ci vorrà pochissimo per riconoscerlo.
L'attività che scannerrizza il barcode viene lanciata usando l'ActivityResultLauncher la quale registra una callback che viene chiamata al termine dell'attività per poter restituire dei valori.

Nel caso in cui l'utente cercasse di aggiungere un prodotto già presente localmente non viene eseguita nessuna azione, ma viene inviato un avvertimento tramite Toast. Al contrario, se il prodotto non è presente possono presentarsi le seguenti casistiche:

1) Un solo prodotto è stato trovato sul server con quel barcode. Viene inserito nel database locale e avvertito l'utente tramite un Toast;
2) il prodotto non è stato trovato sul server, appare un Dialog in cui andare a scrivere nome e descrizione. Viene inserito sia localmente che sul server;
3) Più prodotti sono stati trovati sul server con lo stesso barcode, appare un Dialog con una lista da cui scegliere.

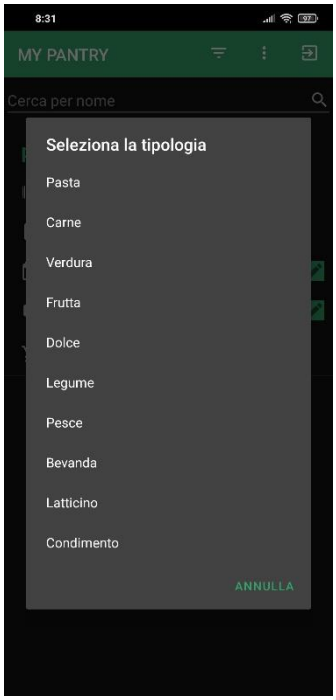






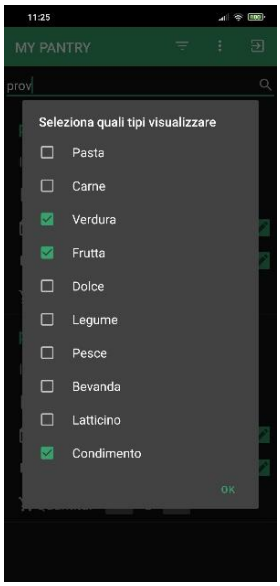
Le due icone verdi al lato destro non sono bottoni, ma DrawableRight con un onTouchListener che va a controllare che tipo di evento è accaduto (in questo caso ACTION_DOWN) e se il click è all'interno del riquadro.Quest'ultimo controllo è stato trovato su StackOverflow in quanto non ero sicuro di come controllare che il click fosse effettivamente all'interno.

Per l'aggiornamento dei prodotti sono andato a creare un'interfaccia "OnItemClickListener" in cui è presente l'intestazione di un solo metodo: void onItemClick(Product p, String op). Viene poi creata nella View una classe anonima che la implementa. La Stringa "op" permette di identificare quale azione eseguire su quale prodotto. Questa classe è passata in input al ProductListAdapter che a sua volta la passa nell' onBindViewHolder al ViewHolder. Nel ViewHolder viene richiamato il metodo quando uno dei due bottoni o uno dei DrawableRight presenti nell'interfaccia viene premuto.



Ai metodi che creano i due Dialog è passato in input una copia del prodotto. In questo modo il prodotto viene aggiornato prima nel database e dopo anche nella View tramite LiveData. Questo procedimento avviene anche per l'update delle quantità.

Grazie a DiffUtil.ItemCallBack che viene richiamato dal submitList del ListAdapter vengono riconosciuti i cambiamenti e aggiornati solo gli item coinvolti.



Premendo il bottone per filtrare i prodotti compare un dialog con la lista delle tipologie. È possibile filtrare i prodotti per tipologia e per nome contemporaneamente grazie ai due metodi, getProductsByName e filterTag, presenti nella ProductViewModel che anziché usare tutti i prodotti del database vanno ad applicare i filtri sulla lista già precedentemente filtrata. Una volta tolti i filtri o eliminato il nome inserito nella TextView, viene reimpostata la lista presa dal database.

e. Notifiche

Per le notifiche delle scadenze dei prodotti sono andato a creare una classe NotifyThread che dopo aver preso in input una lista di prodotti controlla se la data di scadenza è passata o è quella attuale e crea le notifiche.
NotifyThread viene istanziata nell'attività Pantry.java e viene fatto lo scheduling di essa tramite l'Executor. Viene eseguita ogni 60 minuti e ha un delay di 1 minuto. Ho pensato di andare a creare semplicemente un thread anziché un servizio in quanto i prodotti cambiano a seconda dell'utente che ha fatto login.
Una volta che l'utente ha fatto logout tramite il bottone presente in alto a destra non riceverà le notifiche in quanto viene fatto lo shutdown del ScheduledExecutorService.

f. Modalità Landscape e Portrait

È possibile visualizzare l'applicazione nelle modalità Landscape e Portrait in quanto sono stati creati due file xml per i layout più importanti. Inoltre è possibile ruotare lo schermo anche mentre è aperto un Dialog. Quando il metodo onStop viene richiamato, viene cercato nel SupportFragmentManager se il dialogFragment è presente e nel caso lo fosse viene effettuata la rimozione di esso dallo stack. Per fare il commit sono andato ad usare il metodo commitAllowingStateLoss() in quanto non è possibile fare commit dopo savedInstanceState e devo essere sicuro che l'attività sia realmente distrutta. Quando l'attività viene ricreata, vado a vedere nel metodo.
I filtri applicati vengono salvati nel bundle e applicati nuovamente una volta che l'attività viene ripristinata.

Conclusione

Con questo progetto mi è stata data la possibilità di imparare le basi per lo sviluppo di applicazioni android. Applicandomi nella risoluzione dei bug sono riuscito a comprendere meglio il concetto di lifecycle e i vari stati in cui le attività, i fragments e i componenti si trovano nei vari momenti di vita dell'applicazione.
Per quanto presenti diverse funzionalità essa può essere ampliata e perfezionata con l'aggiunta, per esempio, della possiblità di vedere le immagini dei prodotti o anche scaricarne la lista in pdf. Inoltre può essere migliorata a livello estetico, come per esempio andando a definire uno stile migliore per i vari dialog o per la prima attività. Per quanto sia stato interessante lo sviluppo dell'applicazione, ho trovato difficoltà nell'usare Android Studio con l'annesso emulatore. Difficoltà soprattutto dovute ai non pochi frequenti freeze e crash dell'emulatore che mi hanno fatto arrendere ad usare il mio dispositivo mobile in modalità debug usb. Ho comunque trovato il corso e gli argomenti trattati molto attuali e sicuramente mi aiuteranno in futuro.