

Report progetto laboratorio di applicazioni mobili

Lorenzo Ramponi

Luglio 2021

Dati personali

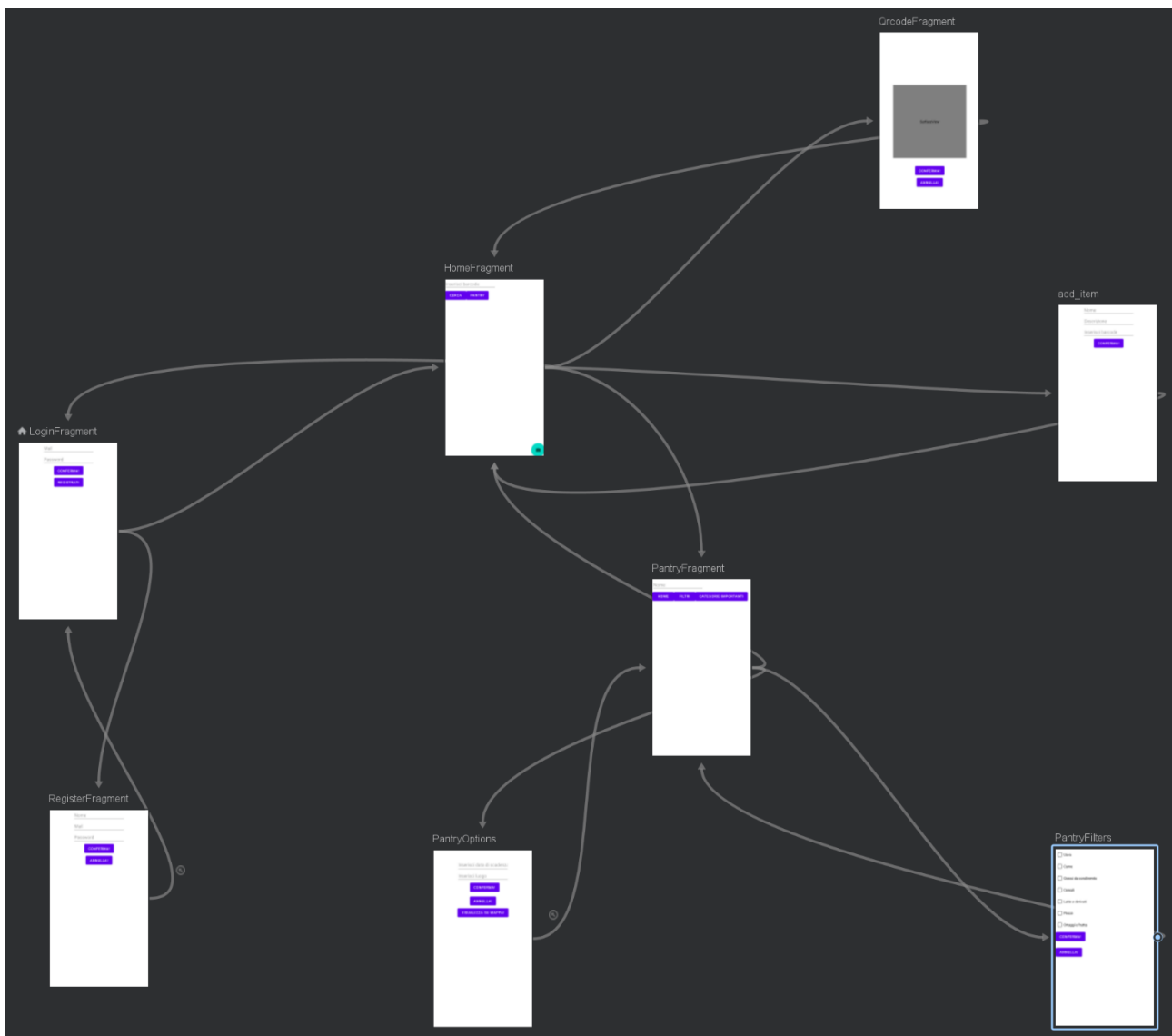
Nome: Lorenzo

Cognome: Ramponi

Email: lorenzo.ramponi@studio.unibo.it

Matricola: 0000873764

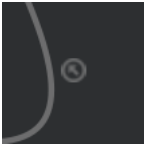
Grafo di navigazione



Il seguente grafo mostra tutte le possibili direzioni che l'utente può seguire.

Le frecce indicano il passaggio da un Fragment all'altro, mentre i rettangoli sono appunto i

Fragments.



Questo simbolo indica che il passaggio tra un Fragment all'altro avviene rimuovendolo dal backstack, in modo da gestire il tasto indietro (che l'utente potrebbe premere) precisamente, per un visual bug non viene però mostrato in tutte le action (ovvero i passaggi).

Navigation

L'applicazione è sviluppata utilizzando Navigation, nonostante non sia presente un menu oppure elementi condiviso da tutti i Fragment (oltre alla toolbar) ho ritenuto che usare navigation fosse un'ottima idea per gestire la struttura a grafo, oltre a garantirmi una soluzione semplice per gestire il backstack quando necessario.

Componenti dell'applicazione

MainActivity.java

E' la prima classe che viene invocata all'avvio dell'applicazione, oltre ad essere l'unica activity presente, mi serve per:

- richiedere i permessi necessari per eseguire determinate azioni (es: aprire fotocamera)
- impostare la Toolbar, in cui inserisco semplicemente il titolo
- impostare la View su activity_main, che include il primo Fragment dell'applicazione
- avviare il service che gestisce le notifiche una volta chiusa l'applicazione.

Login.java

TrackingMyPantry

Mail

Password

CONFERMA!

REGISTRATI

E' il primo Fragment invocato, e qui l'utente vede appunto la sua prima schermata, dove può inserire email e password ed effettuare il login premendo il pulsante conferma, oppure può entrare nel Fragment Register premendo il bottone registrati.

Nel caso in cui l'utente preme il bottone conferma, viene eseguita una richiesta post a /auth/login contenente email e password, se la richiesta ha esito positivo ottengo un JSONObject contenente una stringa di nome accessToken, eseguo infine l'action Login_Home, ovvero passo alla schermata Home, passando il Bundle b che contiene proprio l'accessToken.

Se la richiesta invece ha esito negativo, invio un Toast all'utente con scritto Credenziali errate, per semplicità invio lo stesso messaggio per qualsiasi tipo di errore.

Register.java

TrackingMyPantry

Nome

Mail

Password

CONFERMA!

ANNULLA!

E' il Fragment con cui gestisco il sistema di registrazione, contiene 3 campi testuali in cui l'utente può inserire mail, nome e password e 2 bottoni, il primo una volta cliccato esegue una richiesta post a /users contenente appunti mail, nome e password, in caso di esito positivo ritorno alla schermata Home, tramite l'action Register_Login, in caso contrario invio un Toast all'utente con scritto Errore.

Se l'utente clicca il secondo bottone invece torna semplicemente alla schermata Login tramite l'action Register_Login, senza eseguire nessuna azione complicata.

L'action Register_Login quando invocata svuota il backstack, in modo da non avere 2 schermate Login attive contemporaneamente.

Home.java

TrackingMyPantry

0

CERCA

PANTRY

AGGIUNGI

Nome: Caffe Lavazza

Id: ckpqmzlid153831dpb3d86elwk

Descrizione: Il caffè lavazza in formato capsule, gusto ginseng e gusto cioccolato bianco

Barcode: 0

Nome: Caffe Lavazza

Id: ckpwtt8hd270871dpb3t6snipa

Descrizione: Il caffè lavazza in formato capsule, gusto ginseng e gusto cioccolato bianco

Barcode: 0

Nome: Caffe Lavazza

Id: ckpwudr6x272311dpb6xwjfvlq

Descrizione: Il caffè lavazza in formato capsule, gusto ginseng e gusto cioccolato bianco

Barcode: 0

<

🔍

😊

GIF

📋

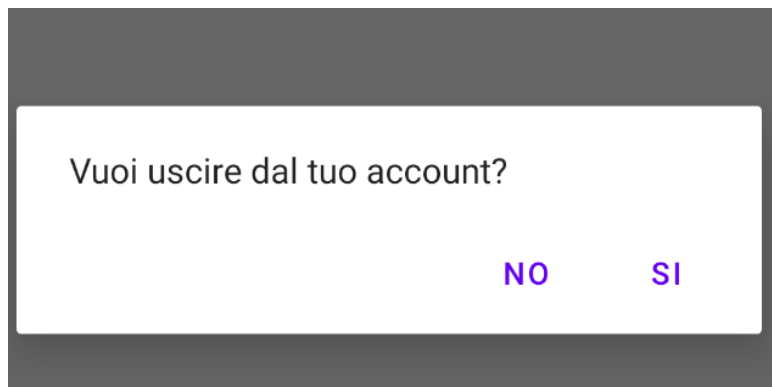
⋮

🎤

Il Fragment home contiene 1 campo testuale e 4 bottoni.

Se l'utente cerca di tornare indietro in questo Fragment, anziché tornare immediatamente alla schermata di Login gli mostro un Dialog in cui chiedo la conferma dell'azione che sta eseguendo, se conferma viene svuotato il backstack e l'utente torna appunto alla schermata di login; se annulla non succede niente.

Creo appunto una nuova variabile `OnBackPressedCallback` callback in cui gestisco il bottone "back" come descritto precedentemente.



Se l'utente preme il bottone 'Cerca' viene richiamata la funzione `getProducts(String barcode)` con il testo presente nel campo testuale.

Se l'utente preme il bottone 'Pantry' viene invocata l'action `Home_Pantry`, tramite la quale si passa alla schermata della propria dispensa personale, rimuovendo il callback personalizzato.

Se l'utente preme il bottone 'Aggiungi' viene invocata l'action `Home_Add`, tramite la quale si passa alla schermata per aggiungere un prodotto al server web, rimuovendo il callback personalizzato

Se l'utente preme il bottone con la foto della camera, viene invocata l'action `Home_Qrcode`, tramite la quale si passa alla schermata di scan del qrccode, rimuovendo il callback personalizzato.

`getProducts(String barcode)`

Questa funzione esegue una get a `/products?barcode=barcode`, che ha sempre esito positivo e restituisce un `JSONObject` contenente la stringa token e un json array contenente i prodotti salvati con il barcode cercato dall'utente.

Traduco il json array in una lista di tipo `<Product>` tramite Gson, se la lista è vuota apro un dialog in cui chiedo all'utente se vuole appunto inserire un prodotto con quel determinato barcode, in caso positivo invoco l'action `Home_Add`, per mostrare la schermata `add_item`, passando il Bundle b in cui ho inserito l'`accessToken`, il barcode ed il token ottenuto tramite questa get, rimuovo oltretutto il callback personalizzato; in caso negativo non succede niente.

Se la lista ha almeno un elemento, creo un `ArrayAdapter<Product>` e lo assegno alla `ListView productsview`, rispetto all'utilizzo di una `RecyclerView` è una soluzione molto semplice e non comporta particolari perdite di efficienza; chiamo oltretutto la funzione `setlistener()`;

`setlistener()`

Questa funzione aggiunge un `ClickListener` sugli elementi della mia lista, come da consegna

2. If the product is in the list, then the user selects it and notifies the WebServer about it. We will call this operation `POST PRODUCT PREFERENCE`.

Se un utente clicca su un elemento della lista, eseguo una Post a /votes, inviando il token ottenuto dalla get precedente, il rating, che imposto di default ad 1, e l'id del prodotto.

Siccome la web api non fornisce una get per ottenere il rating, ho supposto che questi dati vengano usati per analisi di mercato, ed ogni volta che viene eseguita la post sullo stesso prodotto il valore del rating viene aumentato automaticamente, nonostante venga restituito errore.

Oltre ad eseguire la post, una volta cliccato un elemento viene aperto un dialog in cui chiedo all'utente se vuole inserire quel determinato prodotto nella sua dispensa, in caso di esito positivo viene aggiunto il prodotto al database, mentre in caso di esito negativo non succede nulla.

Product.java

Questa è la classe in cui traduco tutti i json object ottenuti dalla get a /products?barcode=barcode.

```
{
  "id": "ckna2s1wc00302bn0465hkboy",
  "name": "Testing Object",
  "description": "Hi, if you read this you
    first object! Congrats!",
  "barcode": "00000000000000",
  "userId": "ckna223pm00002bn055hp3amj",
  "test": false,
  "createdAt": "2021-04-09T08:56:45.468Z",
  "updatedAt": "2021-04-09T08:56:45.469Z"
}
```

Di questi campi mi salvo appunto l'id, il nome, il barcode e l'userId, gli altri non mi sembravano utili da mostrare all'utente.

Per ogni variabile ho creato i setter ed i getter, ed il metodo toString() che verrà utilizzato per mostrare i campi del Prodotto nella ListView della Home.

```
@Override
public String toString() {
    return "Nome: " + name + '\n' +
        "Id: " + id + '\n' +
        "Descrizione: " + description + '\n' +
        "Barcode: " + barcode + '\n';
}
```

AddItem.java

E' il fragment che gestisce l'inserimento di nuovi prodotti attraverso la Web Api, la prima volta che ci si accede verrà già mostrato il barcode ottenuto tramite il bundle dalla Home.

L'utente è comunque libero di cambiare questo campo a suo piacimento.

Oltre al campo per inserire il barcode, sono presenti i campi per inserire il nome e la descrizione del prodotto, ed un bottone per confermare l'operazione.

Se l'utente preme il bottone "Conferma" viene mandata una richiesta POST a /products contenente il token ottenuto dal bundle della home, il nome, la descrizione, il barcode e il booleano test settato a false in quanto non sto eseguendo la richiesta come prova.

La richiesta va sempre a buon fine e viene restituito un JSONObject con i dati del prodotto salvati dal server web

```
JSONObject newbody = new JSONObject();
try {
    newbody.put("id", response.getString("id"));
    newbody.put("name", namestring);
    newbody.put("description", descstring);
    newbody.put("barcode", barcodestring);
    insertedproducts.add(newbody.toString());
}
```

E creo un AlertDialog per chiedere all'utente se vuole aggiungere altri prodotti, in caso affermativo resetto i campi nome, descrizione e barcode in attesa che l'utente li reinserisca e si ripete questo ciclo.

In caso negativo aggiungo al bundle l'ArrayList con tutti i prodotti aggiunti dall'utente, e tramite l'action Add_Home lo rimando sulla schermata Home, che noterà che il parametro "products" del Bundle non è più null ed inserirà automaticamente questi prodotti nella ListView, per mostrare all'utente un riscontro immediato sulle sue azioni.

Questo ArrayList è necessario in quanto l'utente potrebbe aggiungere prodotti con barcode diversi, e non potrei mostrargli in altro modo il riscontro immediato.

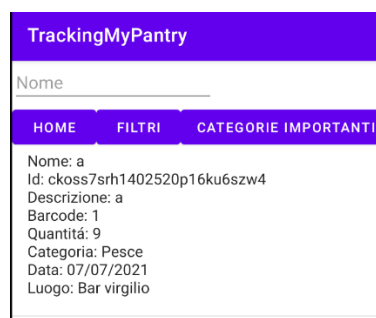
PantryProduct.java

E' una classe che estende Product, mi serve per mostrare più parametri nella Dispensa locale che non sono appunto presenti nella classe Product.

Questi parametri sono:

- Categoria
- Quantità
- Data
- Luogo

Pantry.java



The screenshot shows the 'TrackingMyPantry' app interface. At the top is a purple header with the title 'TrackingMyPantry'. Below the header is a text input field labeled 'Nome'. Underneath the input field is a purple navigation bar with three buttons: 'HOME', 'FILTRI', and 'CATEGORIE IMPORTANTI'. Below the navigation bar, the app displays the details of a product entry: 'Nome: a', 'Id: ckoss7srh1402520p16ku6szw4', 'Descrizione: a', 'Barcode: 1', 'Quantità: 9', 'Categoria: Pesce', 'Data: 07/07/2021', and 'Luogo: Bar virgilio'.

E' la classe con cui gestisco la dispensa locale, i prodotti qui mostrati sono salvati su un Database creato nella classe DBHelper.java

Una volta che l'utente entra in questa schermata gli vengono già mostrati i prodotti che si è salvato in precedenza.

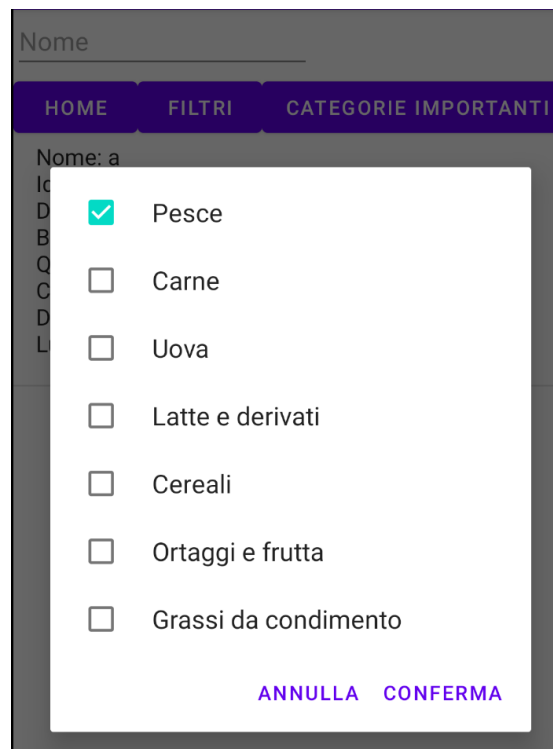
La schermata presenta un campo di testo e 3 bottoni.

Il campo di testo serve per ricercare un determinato prodotto/categoria/nome/... e la ricerca è eseguita automaticamente senza dover premere nessun bottone.

Il bottone Home invoca l'action Pantry_Home, che riporta l'utente sulla schermata Home pulendo il backstack.

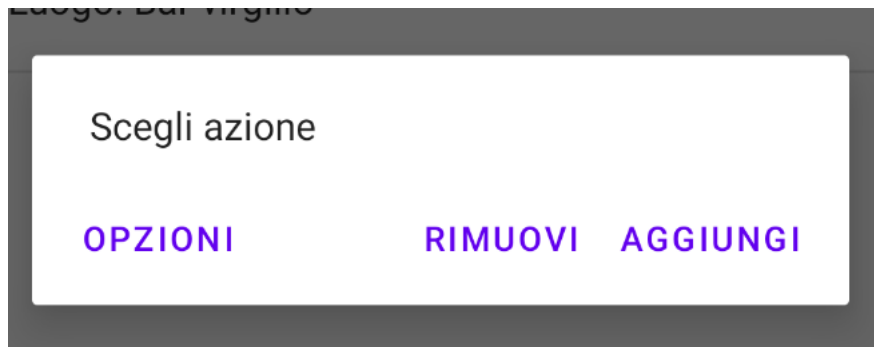
Il bottone Filtri invoca l'action Pantry_PantryFilters, che porta l'utente sulla schermata per aggiungere i filtri alla Lista della dispensa.

Il bottone Categorie importanti crea un AlertDialog che gestisce le categorie per cui l'utente vuole ricevere una notifica una volta finite e/o se stanno finendo.



Se l'utente clicca su un elemento della lista compare un AlertDialog con 3 elementi:

- Aggiungi, che aumenta la quantità del prodotto
- Opzioni, che aggiunge il ProductId al bundle ed invoca l'action Pantry_PantryOptions, aggiungendo il bundle.
- Rimuovi, che diminuisce di 1 la quantità di quel prodotto, nel caso la quantità sia 1 invece lo rimuove dalla Lista.



PantryFilters.java

Questa schermata serve a filtrare la lista di prodotti nella dispensa per categoria.

Sono presenti 7 checkbox, ognuno indica una categoria e premendo il bottone conferma si salva nel database ogni categoria che è stata selezionata, oltre ad invocare l'action PantryFilters_Pantry che ritorna alla schermata della Dispensa, dove la lista ottenuta tramite db.getFilters() non sarà più vuota, ed aggiungo alla ListView solamente i prodotti che hanno categoria uguale a uno presente nella lista.

Se premo il bottone annulla, non viene salvata nessuna modifica e si torna semplicemente alla Dispensa invocando l'action PantryFilters_Pantry.

PantryProductOptions.java

E' la schermata in cui si possono aggiungere informazioni aggiuntive ai prodotti salvati nella dispensa, come la data di scadenza, il luogo e la categoria.

Premendo il bottone conferma si salvano nel database le informazioni inserite, controllando prima

che siano state digitate correttamente, invocando infine l'action `PantryOptions_Pantry` per tornare alla dispensa.

Premendo il bottone visualizza su mappa si esegue un intent che apre l'applicazione Maps, con query il luogo inserito dall'utente.

Premendo il bottone annulla si ritorna semplicemente alla dispensa invocando l'action `PantryOptions_Pantry`.

`DBHelper.java`

È la classe con cui gestisco la connessione al database, creato con SQLite, la prima volta che viene invocato crea 3 tabelle:

- `Filters`, dove salvo i filtri per la dispensa, la colonna è di tipo TEXT
- `Products`, dove salvo i prodotti della dispensa, tutti le colonne sono di tipo TEXT tranne la quantità che è di tipo INT
- `Categories`, dove salvo le categorie importanti per l'utente, è essenzialmente uguale a `Filters` ma il suo utilizzo è volto alle notifiche.

`addProduct(Product p)`

Aggiunge un prodotto alla tabella `Products` se non è presente, se è già presente aumenta la quantità di 1.

`removeProduct(PantryProduct p)`

Rimuove un prodotto dalla tabella `Products` se la quantità è 1, altrimenti diminuisce la sua quantità di 1.

`getProducts()`

restituisce una `List<PantryProduct>` in cui sono presenti tutti i prodotti della tabella `Products`.

`getFilters()`

Restituisce una `List<String>` contenente i filtri inseriti dall'utente per la `ListView` della dispensa.

`getCategories()`

Restituisce una `List<String>` contenente le categorie importanti inserite dall'utente per le notifiche.

`addCategories(List<String> categories)`

Aggiunge le nuove categorie importanti inserite dall'utente per le notifiche, eliminando le precedenti.

`getProduct(String productId)`

Restituisce un `PantryProduct` dal database il cui id è uguale a `productId`

`addPlace(String id, String place)`

Aggiorna il luogo per un determinato prodotto

`addCategory(String id, String category)`

Aggiorna la categoria per un determinato prodotto

addDate(String id, String date)

Aggiorna la data per un determinato prodotto

MyService.java

È il servizio con cui gestisco il sistema di notifiche, viene invocato sull'onStop() di MainActivity.java. Ogni 24 ore, l'app invierà una notifica nel caso in cui i prodotti siano scaduti o stanno per scadere (mancano 7 giorni alla data di scadenza), e nel caso in cui la quantità di prodotti per ogni categoria importante è minore di 10.

Ho creato infatti un Timer che schedula una TimerTask ogni $60 \cdot 1000 \cdot 60 \cdot 24$ millisecondi (ovvero 24 ore).

La TimerTask ottiene la lista di PantryProduct con db.getProducts() e la lista di categorie importanti con db.getCategories().

Per ogni PantryProduct viene quindi visto se è scaduto, e per semplicità se un prodotto è scaduto o vicino alla data di scadenza, interrompo il ciclo inviando un messaggio generale "Alcuni tuoi prodotti sono scaduti"/"Alcuni tuoi prodotti stanno per scadere", in questo modo mi basta che un solo prodotto sia scaduto o vicino alla scadenza per inviare la notifica.

Successivamente gestisco la notifica riguardante le categorie importanti, se la lista di categorie non è 0, per ogni categoria importante inserisco chiave in una HashMap<String,Integer> utilizzando la categoria come chiave, per ogni prodotto, la quantità di quel prodotto se la chiave non è presente nella hashmap, e in caso contrario sommo la quantità al valore che è già presente.

Dopodiché controllo che la quantità per ogni categoria sia almeno 10, in caso contrario interrompo il ciclo e invio una notifica generale "Alcuni prodotti nelle tue categorie importanti stanno finendo", se invece non è presente nessun prodotto di quella categoria invio "Alcuni prodotti nelle tue categorie importanti sono finiti"

QRCode.java

È il Fragment con cui gestisco l'inserimento del Barcode tramite QRCode.

Se non sono stati forniti i permessi non succede nulla, in caso contrario è presente una SurfaceView in cui mostro la camera.

Se la camera inquadra un QrCode, il telefono vibra e viene mostrato a schermo il testo trovato; a questo punto l'utente può premere il bottone Conferma, che aggiunge al Bundle il qrCode, invoca l'action Qrcode_Home e passa alla schermata Home, che esegue la funzione getProducts(String barcode) con il codice mostrato a schermo nella schermata QRCode.

Se l'utente preme invece il pulsante Annulla, viene invocato l'action QRCode_Home e si passa alla schermata Home senza che succeda nulla di particolare.

Volley

Tutte le richieste http vengono eseguite con Volley, in quanto mi permette di gestire il body in modo più che semplice:

```
@Override
public byte[] getBody() {
    return requestBody.getBytes(StandardCharsets.UTF_8);
}
```

E il sistema di autorizzazione con accessToken:

```
@Override
public Map<String, String> getHeaders() throws AuthFailureError {
    Map<String, String> params = new HashMap<>();
    params.put("Authorization", "Bearer " + accessToken);
    return params;
}
```

Nel caso in cui la richiesta non sia corretta, gestisco il sistema di errori inviando all'utente un Toast:

```
@Override
public void onErrorResponse(VolleyError error) {
    Log.d( tag: "ErrorRegister", error.toString());
    Toast.makeText(context, text: "Account già esistente",
        Toast.LENGTH_SHORT).show();
}
```