



**UNIVERSITA' POLITECNICA DELLE MARCHE
FACOLTA' DI INGEGNERIA**

Corso di Laurea in **INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE**

**Andrea Maranesi
Alessandro Marcolini**

A.A. 2020/2021

APP RECENSIONI

APP RECENSIONI

INTRODUZIONE

1. SVILUPPO IN KOTLIN

1.1 - REQUISITI E DESCRIZIONE DEI CASI D'USO

- 1) HOME
- 2) CODICE A BARRE
- 3) IMPLEMENTAZIONE RICERCA TESTUALE
- 3) RICERCA RECENSIONI E VISUALIZZAZIONE DEI RISULTATI
- 5) FILTRI
- 6) PRODOTTI CORRELATI
- 7) NOTIFICHE

1.2 - ARCHITETTURA E COMPONENTI AD ALTO LIVELLO

1.3 - MOCKUP E RAPPRESENTAZIONE GRAFICA DEI CASI D'USO

Ricerca Recensioni

Visualizzazione Recensioni Salvate

STILE DARK:

RAPPRESENTAZIONE GRAFICA DEI CASI D'USO

HOME

1. CONSENTE DI CERCARE TESTUALMENTE IL PRODOTTO
2. CONSENTE DI AVVIARE LA RICERCA TESTUALE
3. APRE L'ACTIVITY PER RILEVARE IL CODICE A BARRE
4. SEZIONE RICERCHE EFFETTUATE
5. SEZIONE RECENSIONI SALVATE
6. SEZIONE PRODOTTI SUGGERITI

RICERCA

1. RECENSIONE
 - 1.1 PULSANTE LISTA DESIDERI
 - 1.2 PULSANTE CONDIVIDI RECENSIONE
2. PULSANTE LISTA DESIDERI
3. ICONA PER FILTRARE PER TESTO
4. APRE IL FRAGMENT DEDICATO AI FILTRI

1.4 - SVILUPPO

SCELTE PROGETTUALI

DESIGN

LINGUA

RICERCA RECENSIONI

IMPORTANTE

MULTITHREADING

DESCRIZIONE COMPONENTI

UI

- MainActivity
ScannerActivity
FRAGMENT HOME
FRAGMENT RESULTS
FRAGMENT FILTERS
FilterModeActivity

MODELS

VIEW MODELS

REPOSITORIES

DATABASE

NETWORK

REVIEWS

SETTINGS

NOTIFICATIONS

ALtre DIRECTORIES

"BUG" APP

MALFUNZIONAMENTO CHE NON RIGUARDA LA VERSIONE CORRENTE DEL PROGETTO

LOGIN - REGISTRAZIONE - FIREBASE

5.0 - TESTING

JUnit

TEST UI

ReviewInstrumentTest

2. SVILUPPO IN FLUTTER

2.1 - REQUISITI E DESCRIZIONE DEI CASI D'USO

- 1) HOME
- 2) CODICE A BARRE
- 3) IMPLEMENTAZIONE RICERCA TESTUALE
- 3) RICERCA RECENSIONI E VISUALIZZAZIONE DEI RISULTATI
- 5) FILTRI

2.2 - ARCHITETTURA E COMPONENTI AD ALTO LIVELLO

2.3 - MOCKUP E RAPPRESENTAZIONE GRAFICA DEI CASI D'USO

Ricerca Recensioni

Visualizzazione Recensioni Salvate

STILE DARK:

RAPPRESENTAZIONE GRAFICA DEI CASI D'USO

HOME

1. CONSENTE DI CERCARE TESTUALMENTE IL PRODOTTO
2. CONSENTE DI AVVIARE LA RICERCA TESTUALE
3. APRE IL WIDGET PER RILEVARE IL CODICE A BARRE
4. APRE IL WIDGET CHE MOSTRA LE RECENSIONI SALVATE

RICERCA

1. RECENSIONE
- 1.1 PULSANTE LISTA DESIDERI
- 1.2 PULSANTE CONDIVIDI RECENSIONE
2. PULSANTE LISTA DESIDERI
3. ICONA PER FILTRARE PER TESTO

2.4 - SVILUPPO

SCELTE PROGETTUALI

DESIGN

LINGUA E STRINGHE

RICERCA RECENSIONI

DESCRIZIONE COMPONENTI

Home

QrCode

HomeReviewList

ReviewPage

WIDGETS

MODELS

DATABASE

NETWORK

DIRECTORY COMMON

"BUG" APP

INTRODUZIONE

L'app ha come obiettivo quello di fornire recensioni a partire dal codice a barre di un determinato prodotto, oppure da una ricerca testuale dello stesso, consentendo all'utente di filtrare poi i risultati. Come Marketplace di riferimento, per recuperare le varie recensioni, sono utilizzati Amazon e GoodReads.

Amazon è stato scelto per due ragioni:

- Grande quantità di articoli nei più svariati settori, dall'elettronica all'abbigliamento, e dunque scarsa probabilità di fallimento durante la ricerca di un articolo
- Elevato numero di recensioni per i prodotti

GoodReads è invece un marketplace dedicato al mondo dei libri.

Lo abbiamo scelto per l'elevato numero di recensioni per ogni libro e per la possibilità che fornisce, dato un certo ISBN, di restituire l'url del prodotto stesso.

1. SVILUPPO IN KOTLIN

1.1 - REQUISITI E DESCRIZIONE DEI CASI D'USO

L'app, partendo da una Home che funge da schermata primaria per accedere alle varie funzionalità (1), consente di cercare le recensioni su uno o più marketplace ([Amazon](#), [GoodReads..](#)) a partire da un codice a barre scansionato da un determinato prodotto (2), oppure attraverso una semplice ricerca testuale dell'articolo stesso (3).

Una volta trovate le recensioni per un certo articolo (4) l'utente può **filtrarle** (5), salvarle tra i preferiti e, previa selezione, condividerla con un contatto via Whatsapp, Telegram, Email ecc..

L'utente può anche vedere prodotti correlati a quelli ricercati (6) e, **periodicamente**, riceve una notifica che lo avvisa di uno dei potenziali prodotti correlati al quale potrebbe essere interessato a vederne le recensioni (7)

I dati vengono **esclusivamente** salvati nel database locale.

L'utente può perdere la connessione durante una ricerca, questa viene ripresa non appena l'utente ha campo sufficiente per ristabilire la connessione.

La ricerca delle recensioni avviene in modo **concorrenziale** tra i marketplace implementati, e ogni qualvolta viene letta una recensione, viene mostrata all'utente anche se il processo di ricerca non è ancora terminato.

1) HOME

L'**Home** dell'app è suddivisa in **3 sezioni**, riempite con un numero di contenuti **limitati**:

- Ricerche => ordinate dalla più recente

Premendo su una ricerca, verranno nuovamente trovate le recensioni dai vari marketplace.

- Prodotti che presentano una o più recensioni salvate => ordinati in base al prodotto cercato più recentemente

Premendo su un prodotto, viene aperta **l'Activity** che mostra tutte le recensioni salvate per quell'articolo.

- Prodotti suggeriti a partire dai prodotti cercati => ordinati in base al prodotto cercato più recentemente

Premendo su un prodotto correlato, vengono cercate le sue recensioni.

Per ogni **sezione**, cliccando su uno specifico pulsante **Vedi Tutto** in alto a destra, che **compare solo quando il numero di contenuti nella sezione raggiunge il limite imposto**, si apre un'**Activity** dedicata dove l'utente può visualizzare **Tutti** i contenuti associati:

- Activity Ricerche

- Ordine di visualizzazione: **dalla più recente**

Consente, tenendo premuto a lungo sul prodotto i-esimo, di selezionare gli elementi che l'utente vuole cancellare dal db locale. Se l'utente **conferma** la sua scelta, previa comparsa di un'apposita **dialogo**, viene cancellata la ricerca, **tutte le recensioni salvate associate a quel prodotto, ed eventuali prodotti suggeriti a partire dallo stesso**.

Cliccando sulla particolare ricerca, invece, l'utente può nuovamente ricercare le recensioni.

- Activity Recensioni Salvate Associate a uno Specifico Prodotto

- Ordine di visualizzazione: **Non Specifico**

Consente, tenendo premuto a lungo sulla recensione i-esima, di selezionare gli elementi che l'utente vuole cancellare dal db locale. Se l'utente **conferma** la sua scelta, previa comparsa di un apposito **dialogo**, vengono eliminate **definitivamente** le recensioni selezionate.

Cliccando sulla particolare recensione, invece, l'utente naviga verso l'**Activity** dedicata interamente alla visualizzazione del contenuto della recensione.

- Activity Prodotti Correlati
 - Ordine di visualizzazione: **Dal prodotto cercato più recentemente**

Mostra **Tutti** i prodotti suggeriti a partire dalle ricerche.

2) CODICE A BARRE

Il codice a barre può essere di qualunque tipo: ISBN, EAN-13, EAN-8, UPC... e viene letto attraverso l'uso della fotocamera posteriore del cellulare.

3) IMPLEMENTAZIONE RICERCA TESTUALE

La ricerca testuale viene gestita attraverso il **primo** prodotto che viene trovato sul marketplace **Amazon** al seguente url:

<https://amazon.it/s?k={stringa}>

Questa scelta è stata fatta per riuscire a **identificare univocamente** un prodotto ricercato, data l'ambiguità dei risultati che possono scaturire cercando un articolo in maniera generica, ad esempio indicando semplicemente il termine "cellulare" nella stringa di ricerca.

Dal codice articolo individuato dapprima tramite Amazon vengono cercate poi le recensioni **anche** dagli altri marketplace.

3) RICERCA RECENSIONI E VISUALIZZAZIONE DEI RISULTATI

Una volta ottenuto il codice a barre oppure la stringa di ricerca, l'app provvede alla ricerca delle recensioni sui Marketplace che sono stati implementati, **Amazon** e **GoodReads**.

Le recensioni trovate hanno tutte da 1 a 5 stelle.

La ricerca, che avviene totalmente in **background**, visita le varie pagine di recensioni fornite dai due marketplace e, come viene fornito il contenuto HTML, ne estrae i dati e aggiorna la lista corrente.

Le recensioni vengono ordinate, di **default**, da quella con voto più alto.

La lista mostra inizialmente un numero limitato di recensioni (massimo **150**), ma come l'utente scorre e arriva verso la fine, la ricerca **itera** il procedimento appena descritto.

Ogni recensione in questa ricerca mostra un numero limitato di caratteri.

Selezionando la recensione desiderata, però, l'effetto sarà quello di navigare verso l'**Activity** dedicata interamente alla visualizzazione del contenuto della recensione.

L'utente può, direttamente dalla lista dei risultati, salvare o eliminare la recensione dai preferiti.

La recensione **può** presentare contenuto **html**, questo viene interpretato graficamente durante la visualizzazione della stessa.

5) FILTRI

Mentre vengono trovate le varie recensioni, l'utente può impostare dei filtri, in logica **AND** da applicare alle stesse:

- Ordine di visualizzazione:
 - Dalla migliore alla peggiore
 - Dalla peggiore alla migliore
 - Dalla più recente
 - Dalla più vecchia
- Valutazione
- Parole più ricorrenti

In base alle parole più ricorrenti (le prime **10**) estrapolate dalle varie recensioni, l'utente potrà vederle e selezionare quelle alle quali è interessato. Verranno mostrate solo le recensioni che contengono al proprio interno quel termine. L'algoritmo può mostrare anche congiunzioni o verbi, non ci sono particolari controlli sul significato dei caratteri estratti.

- Marketplace

Fornisce la possibilità di escludere Amazon o GoodReads dai risultati di ricerca

Oltre ai filtri appena descritti, l'utente può filtrare anche cercando del contenuto all'interno della recensione stessa, attraverso un'apposita icona di ricerca che viene mostrata durante la visualizzazione dei risultati.

6) PRODOTTI CORRELATI

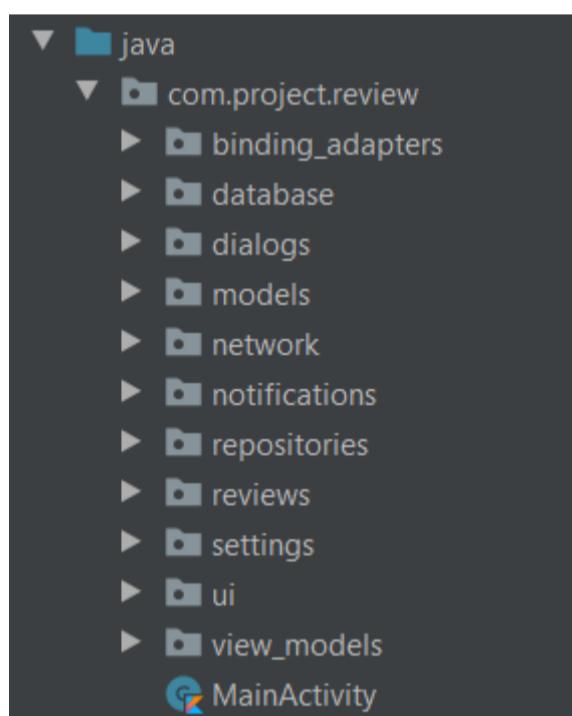
I prodotti correlati a uno cercato vengono **esclusivamente** estrapolati da **Amazon**.

7) NOTIFICHE

L'utente, ogni 48h, viene notificato se presenti prodotti correlati a quelli già cercati al quale potrebbe essere interessato.

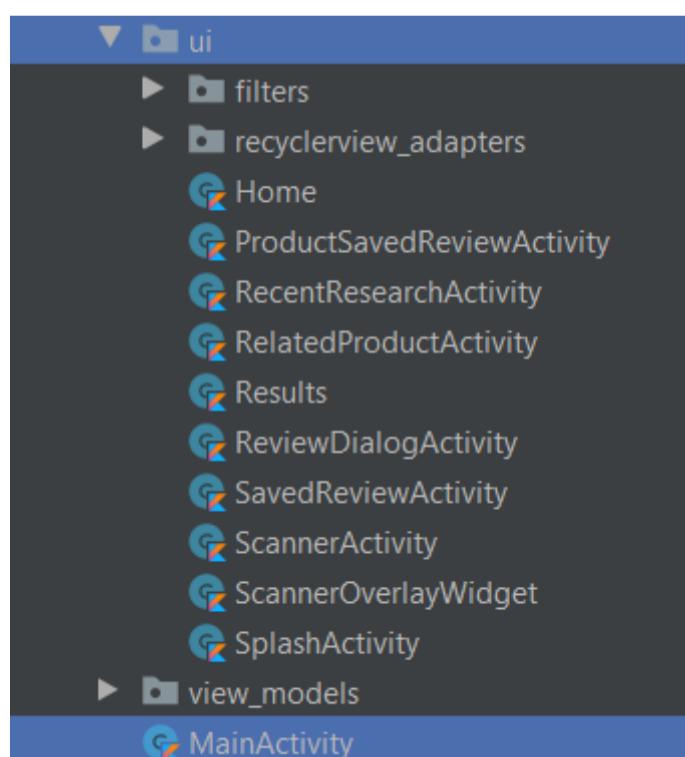
Se preme sulla notifica, vengono cercate le recensioni associate a quel determinato articolo.

1.2 - ARCHITETTURA E COMPONENTI AD ALTO LIVELLO



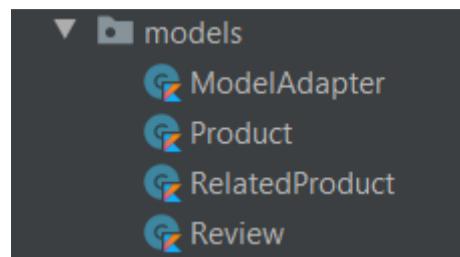
L'app è stata organizzata secondo il pattern **MVVM**.

Le viste sono organizzate dentro la directory **ui**:

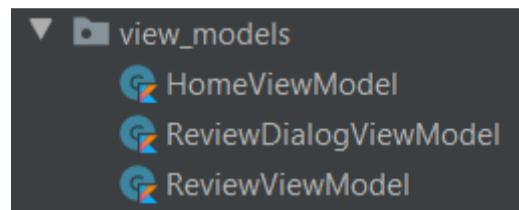


I **Fragment** sono denominati senza il suffisso **Activity**. Dentro **filters** ci sono le interfacce dedicate ai filtri delle recensioni.

I Modelli dai quali siamo partiti per poi sviluppare il database e le varie funzionalità dell'app sono 3, seguiti da **ModelAdapter** che è superclasse di **Product** e **Review**:

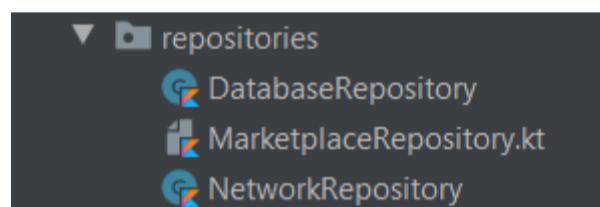


I modelli comunicano con 3 View Models:



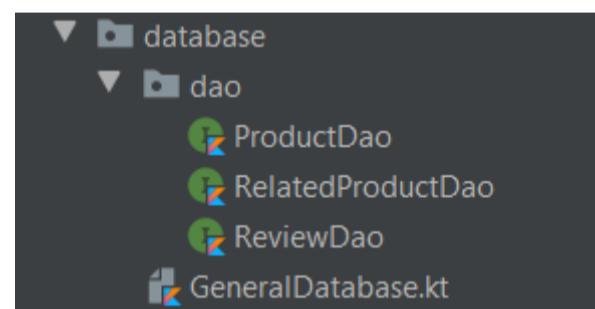
- **ReviewViewModel** gestisce il recupero delle recensioni con l'applicazione di eventuali filtri
- **ReviewDialogViewModel** gestisce funzionalità proprie dell'**Activity** dedicata alla visualizzazione del contenuto intero di una recensione
- **HomeViewModel** è un modello a supporto delle **3 sezioni** presenti nella **Home**

I View Models si interfacciano con 3 **repositories** differenti, separate in base alla logica delle funzionalità implementate:

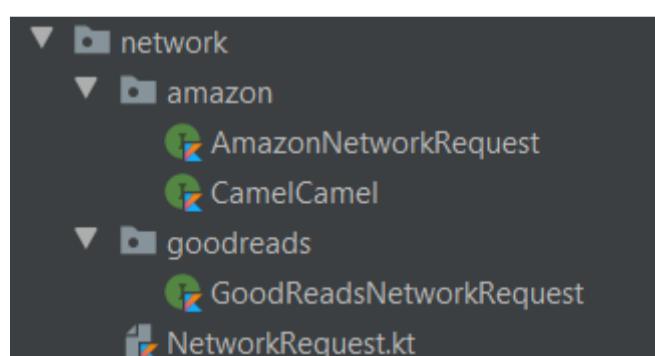


- **DatabaseRepository** si occupa del database
- **MarketplaceRepository** gestisce il recupero delle recensioni interfacciandosi con le classi della directory **reviews** e con **NetworkRepository**
- **NetworkRepository** comunica direttamente con i metodi per la connessione a internet

Il database, contenuto nell'omonima directory, contiene la classe **GeneralDatabase** per istanziarlo e altre, dentro **dao**, per eseguire le varie query:

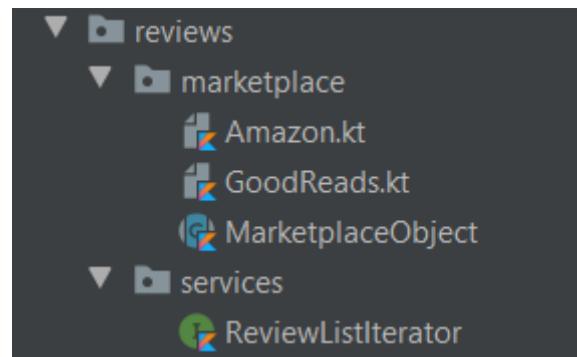


La connessione alla rete è gestita nella directory **network**:



La classe **NetworkRequest** istanzia le strutture per la connessione, dentro amazon e goodreads vengono gestite le varie richieste http.

La directory **reviews** contiene, per ogni marketplace implementato, una classe denominata **{Nome_Marketplace}** e un'altra, dentro lo stesso file **{Nome_Marketplace}.kt**, denominata **{Nome_Marketplace}ListIterator**. Le classi servono per recuperare le recensioni dai marketplace:

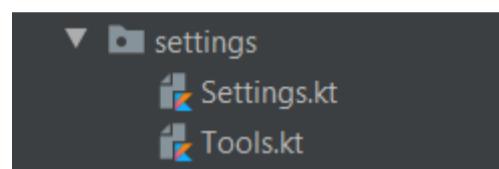


{Nome_Marketplace} stabilisce **come** deve essere estrapolato un prodotto dal rispettivo marketplace.

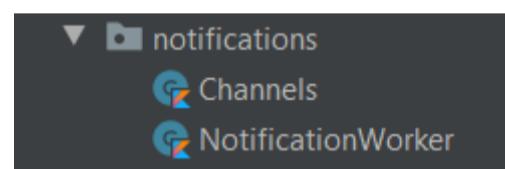
{Nome_Marketplace}ListIterator stabilisce **come** devono essere estrapolate le varie recensioni analizzando pagina per pagina i risultati fino al termine della ricerca.

Le altre classi presenti nella directory definiscono l'interfaccia che deve essere implementata da quest'ultime.

I valori di default come lo **User agent** usato per le richieste http, il numero minimo di recensioni da visualizzare durante una ricerca e così via, sono definiti dentro la classe **Settings**.



La directory **notifications** gestisce i canali per le notifiche e la schedulazione periodica delle stesse, realizzata attraverso un CoroutineWorker (**NotificationWorker**):

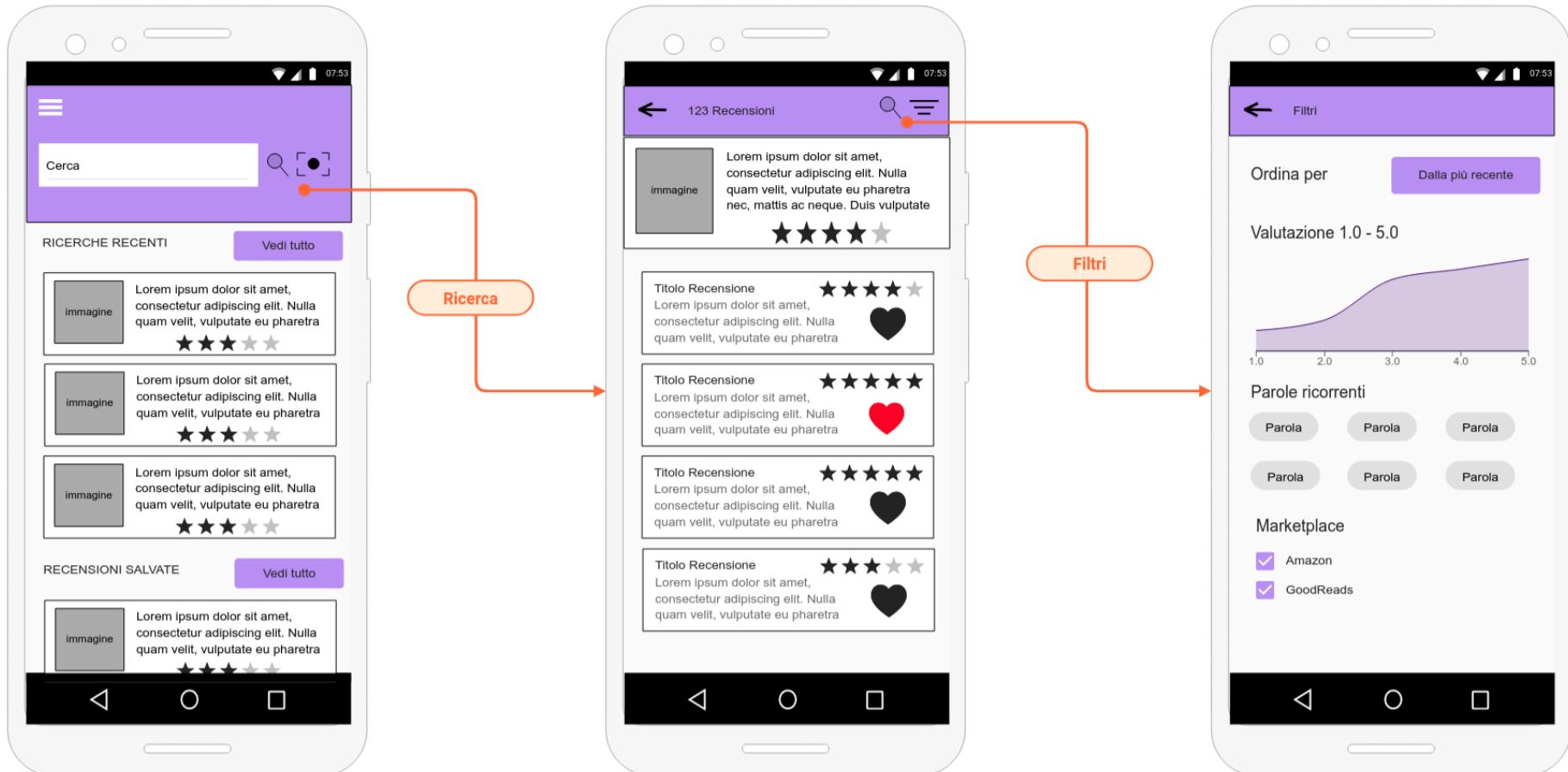


1.3 - MOCKUP E RAPPRESENTAZIONE GRAFICA DEI CASI D'USO

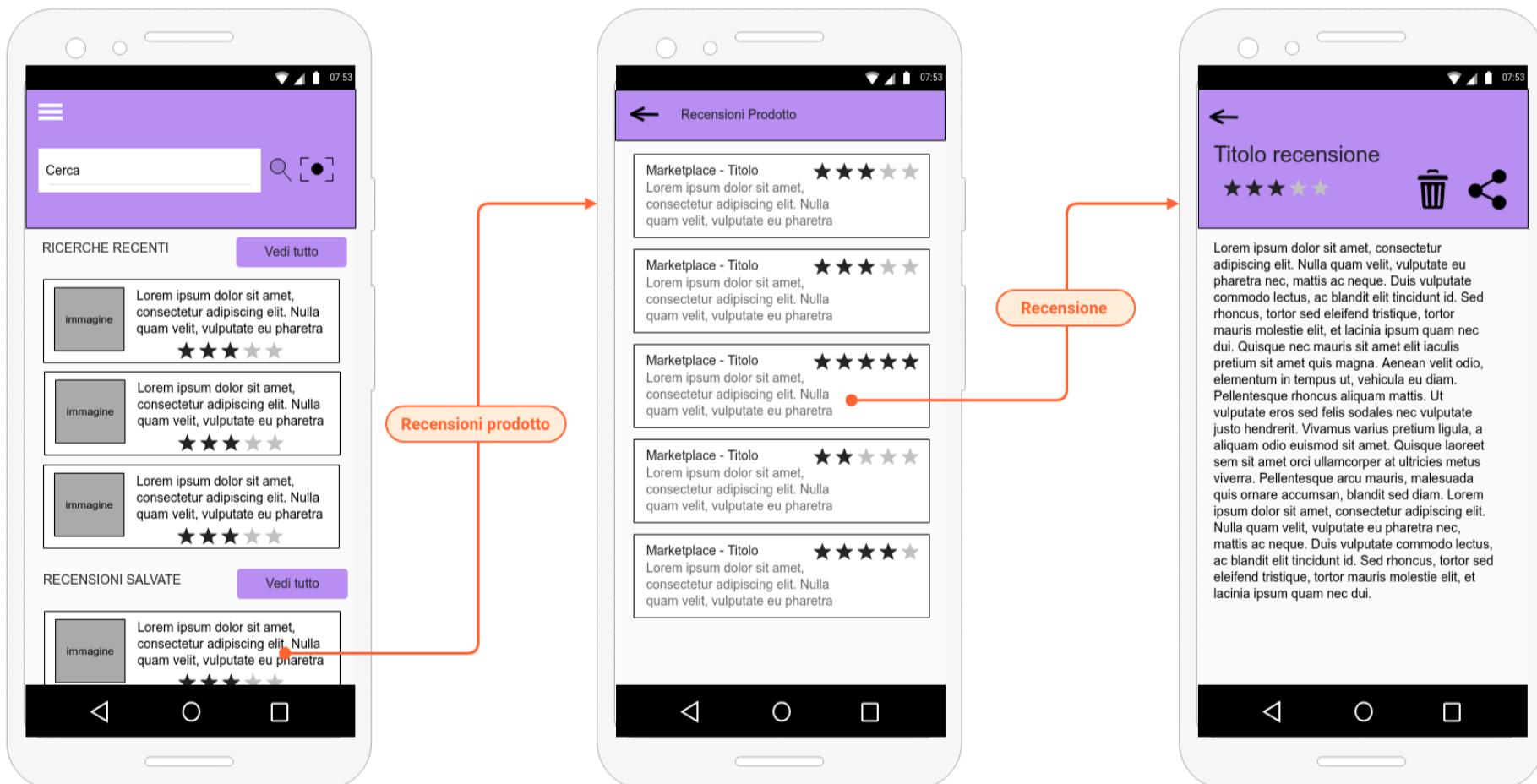
L'app è stata realizzata con due temi, uno light e uno dark.

Di seguito il mockup delle principali interfacce utente:

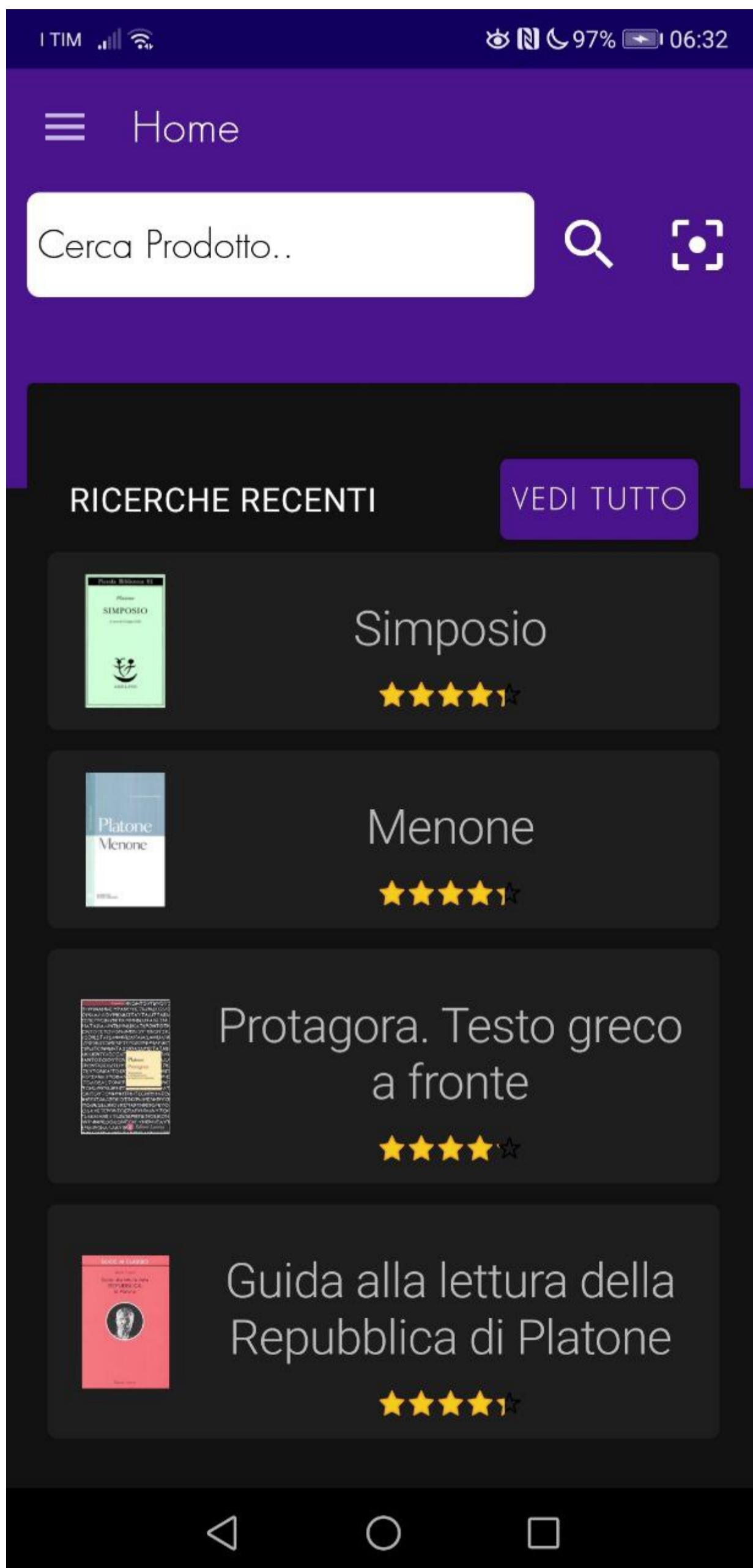
Ricerca Recensioni



Visualizzazione Recensioni Salvate



STILE DARK:



RAPPRESENTAZIONE GRAFICA DEI CASI D'USO

HOME

≡ Home

1

2

3

Cerca Prodotto..



4

RICERCHE RECENTI

VEDI TUTTO



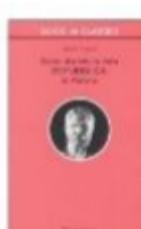
Simposio



Menone



Protagora. Testo greco
a fronte



Guida alla lettura della
Repubblica di Platone



5

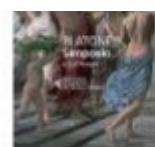
RECENSIONI SALVATE



Il simposio



Simposio



SIMPOSIO

★★★★★☆



Menone

★★★★★☆

6

PRODOTTI SUGGERITI

VEDI TUTTO



La Divina
Commedia:
Inferno,
Purgatorio
e Paradiso

★★★★★☆



Fedro

★★★★★☆



Simposio.
Testo
greco a
fronte

★★★★★☆



1. CONSENTE DI CERCARE TESTUALMENTE IL PRODOTTO

2. CONSENTE DI AVVIARE LA RICERCA TESTUALE

3. APRE L'ACTIVITY PER RILEVARE IL CODICE A BARRE



4. SEZIONE RICERCHE EFFETTUATE

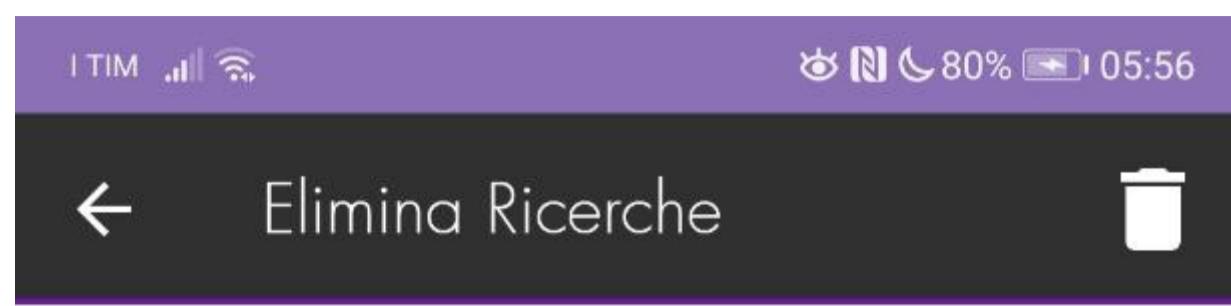
In questa sezione premendo sul pulsante **Vedi Tutto** si viene rimandati alla seguente Activity:

The screenshot shows a mobile application interface for searching books. At the top, there is a purple header bar with various icons: signal strength, battery level at 51%, and the time 07:01. Below this is a navigation bar with a back arrow and the text "Ricerche". The main content area displays five search results, each consisting of a small book cover thumbnail on the left and descriptive text on the right.

- Il simposio**
Platone SIMPOSIO
- Protagora. Testo greco a fronte**
Platone Protagora
- Guida alla lettura della Repubblica di Platone**
Platone Guida alla lettura della Repubblica di Platone
- Fedro. Testo greco a fronte**
Platone Fedro
- Simposio: Testo greco a fronte**
Platone Simposio

At the bottom of the screen, there is a black navigation bar with three white icons: a triangle pointing left, a circle, and a square.

Premendo a lungo sul prodotto desiderato, comparirà la seguente schermata che consente di eliminare gli articoli selezionati:



AMAZON - C



Un classico senza tempo, curato da un autore.

Unico appunto: quando si acquista un libro,
segnalibro in omaggio sarebbe gradito.

AMAZON - A

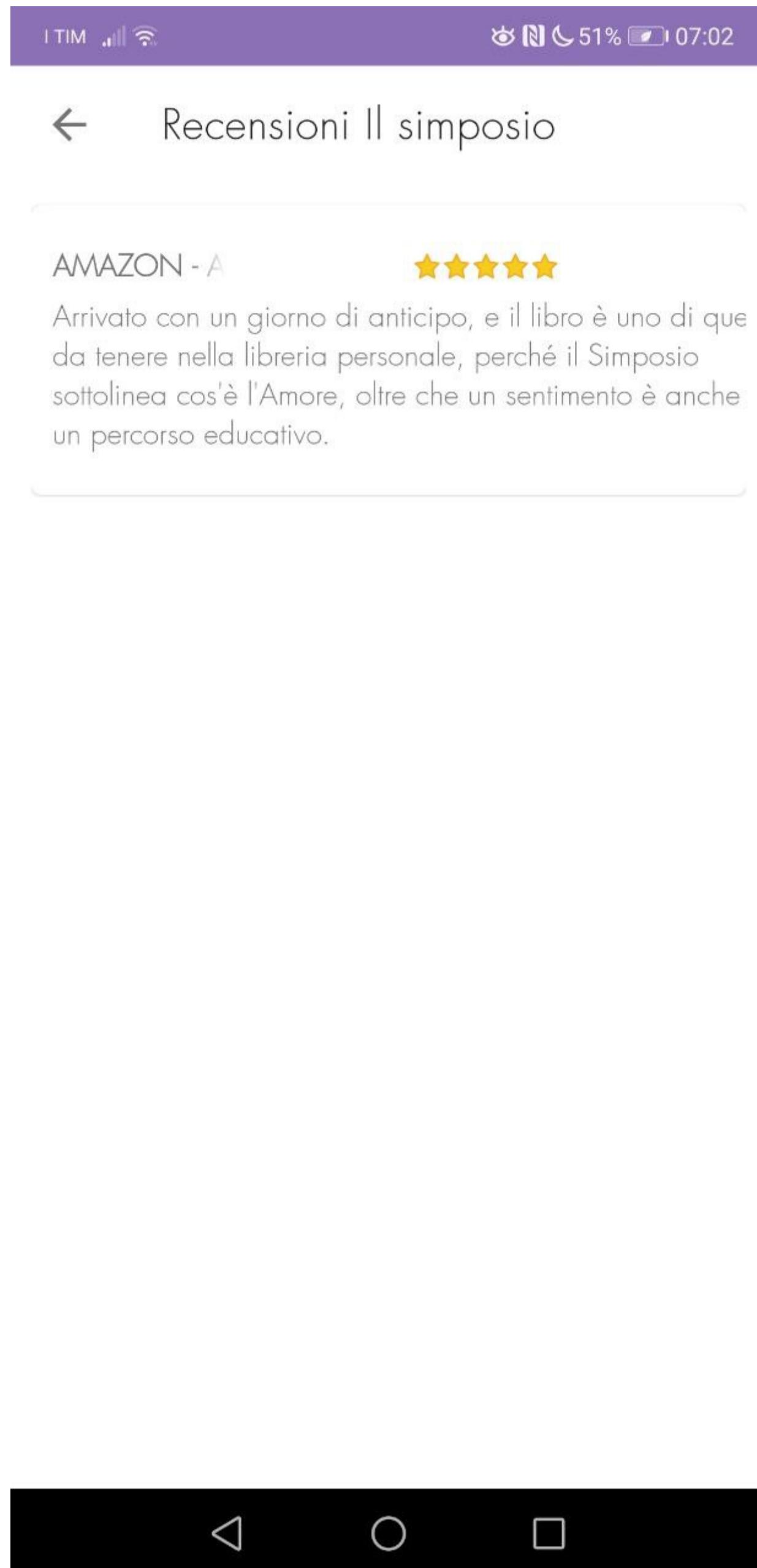


Arrivato con un giorno di anticipo, e il libro è
da tenere nella libreria personale, perché il S
sottolinea cos'è l'Amore, oltre che un sentime
un percorso educativo.



5. SEZIONE RECENSIONI SALVATE

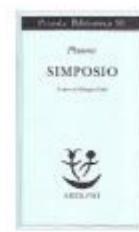
In questa sezione premendo sul pulsante **Vedi Tutto** si viene rimandati alla seguente Activity:



Premendo a lungo sulla recensione desiderata, comparirà la seguente schermata che consente di eliminare le recensioni selezionate:



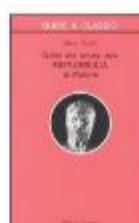
Elimina Ricerche



Il simposio



Menone

Protagora. Testo
greco a fronteGuida alla lettura
della Repubblica
di PlatoneFedro. Testo
greco a fronte

Simposio: Testo



Selezionando invece la recensione si apre l'Activity dedicata:



Un classico senza tempo, curato da un autore che amo. Unico appunto: quando si acquista un libro, ricevere un segnalibro in omaggio sarebbe gradito.

6. SEZIONE PRODOTTI SUGGERITI

In questa sezione premendo sul pulsante **Vedi Tutto** si viene rimandati alla seguente Activity

:

← Prodotti suggeriti



Da Il simposio



La Divina
Commedia:
Inferno,
Purgatorio e
Paradiso

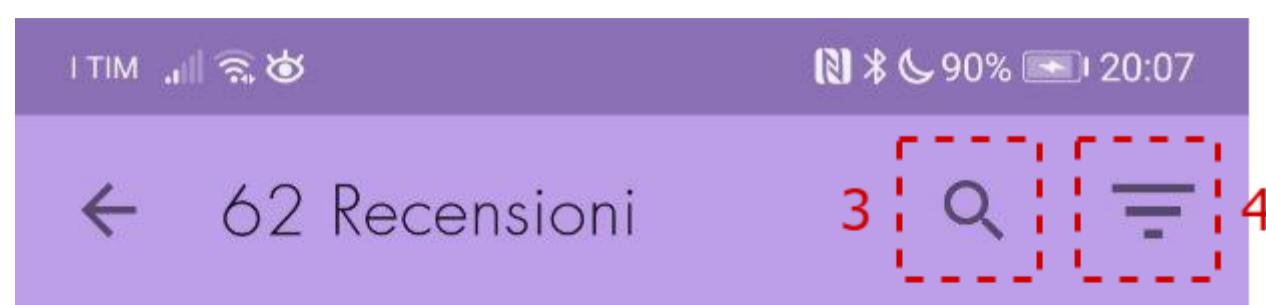


Fedro



Simposio.
Testo greco
a fronte





Simposio

4.33 

1

AMAZON - C





Questo saggio dell'antica tradizione ellenica mette al centro uno dei concetti in assoluto più complesso e inesauribile della vita di chiunque, quale è appunto l'amore. Più di tutte suscitano atten...

AMAZON - A





Arrivato con un giorno di anticipo, e il libro è uno di quelli da tenere nella libreria personale, perché il Simposio sottolinea cos'è l'Amore, oltre che un sentimento è anche un percorso educativo.

AMAZON - C





Un classico senza tempo, curato da un autore



1. RECENSIONE

Premendo sulla recensione desiderata, viene aperta l'Activity dedicata:

1 2

Questo saggio dell'antica tradizione ellenica mette al centro uno dei concetti in assoluto più complesso e inesauribile della vita di chiunque, quale è appunto l'amore. Più di tutte suscitano attenzione le declamazioni di Socrate che spiega di non riportare altro che gl'insegnamenti gli furono a sua volta esposti da una donna saggia quale era Diotima. Nonostante tutte le riduzioni, banalizzazioni e individualismi (che poi sono conseguenza della morte di dio) occorsi nell'età contemporanea, credo comunque l'ideale di amore romantico che siamo usi idealizzare e ricercare da un paio di secoli a questa parte, si rifaccia ancora all'amore cristiano, almeno idealmente. La ricerca dell'immortalità e la perpetuazione della bellezza nella discendenza del proprio sangue invece, credo sia un aspetto semplicemente mai preso neanche

1.1 PULSANTE LISTA DESIDERI

1.2 PULSANTE CONDIVIDI RECENSIONE

Avvia un intent implicito:



2. PULSANTE LISTA DESIDERI

3. ICONA PER FILTRARE PER TESTO

Consente di filtrare le recensioni che contengono, nel corpo, i termini ricercati:



4. APRE IL FRAGMENT DEDICATO AI FILTRI

1.4 - SVILUPPO

SCELTE PROGETTUALI

DESIGN

Abbiamo utilizzato come tema primario una specializzazione di **Theme.MaterialComponents.Light.DarkActionBar** per ciò che concerne il light theme, e di **Theme.MaterialComponents.DayNight** per quello dark.

```
<style name="Theme.Review" parent="Theme.MaterialComponents.Light.DarkActionBar">
```

Theme.Review è la base dalla quale abbiamo specializzato altri temi, come **Theme.Review.NoActionBar**, utile quando si vuole avere un'activity senza l'ActionBar di default

```
<style name="Theme.Review.NoActionBar">
    /....
</style>
```

LINGUA

L'app è disponibile sia in **italiano** che in **inglese**, MA le recensioni vengono sempre cercate in **italiano** nella versione corrente del progetto.

RICERCA RECENSIONI

Ci vogliamo soffermare su questo aspetto vista la particolare difficoltà che, sia Amazon che GoodReads, hanno comportato per estrarre le recensioni. Nessuno dei due marketplace possiede infatti delle API gratuite utili ai nostri fini. Abbiamo così deciso di usare un certo **User-agent** in ogni richiesta http, e di estrapolare i contenuti **direttamente dal dom della pagina**, usando a tal fine la libreria **Jsoup**.

Ogni volta l'utente comincia una nuova ricerca, discriminiamo se viene fornite il codice a barre oppure una stringa. Nel primo caso, la classe **Amazon** si appoggia al sito web **CamelCamelCamel** per ricavare dal codice il corrispondente **Asin**, ossia il codice identificativo dei prodotti su Amazon.

Questo lo si può fare in camelcamel attraverso la richiesta http get all'endpoint **/search?sq={codice}**, che porta o alla pagina **/product/{asin}** o a una che mostra più risultati. In quest'ultimo caso l'app seleziona l'asin del primo di questi.

Ottenuto l'asin, ogni prodotto si trova all'endpoint amazon **/dp/{asin}**, mentre le relative recensioni in **/product-review/{asin}**

GoodReads invece, attraverso l'url **https://www.goodreads.com/search?query={asin}** fa una **redirect** all'url del prodotto stesso, dove al proprio interno troviamo le recensioni, **non iterabili senza Javascript**. Dal momento che la libreria **retrofit2** , che abbiamo utilizzato per le richieste alla rete, non consente l'interpretazione js di una pagina, dopo un'ispezione del codice sorgente, abbiamo dedotto la possibilità di iterare le recensioni attraverso la richiesta http get a **{url_prodotto}?language_code={codice_lingua}&page={numero_pagina}**

Nel caso in cui, invece, la ricerca avviene tramite testo, l'app prende il primo risultato che trova all'url

<https://amazon.it/s?k={ricerca}>

e poi, tramite l'url <https://it.camelcamelcamel.com/product/{asin}>, ricava, **se possibile**, l'eventuale codice **ISBN, EAN o UPC**.

Product Details	
Gruppo prodotto	Libro
Categoria	Copertina flessibile
Produttore	Pearson Education
Conf. locali	IT
ISBN	1292100311
EAN	978129210031
Cod. prod.	1292100311
Scansione ultimo aggiornamento	2 mesi fa

[View at Amazon](#)

A questo punto, **tutti gli altri marketplace** vengono attivati per ricercare al proprio interno lo specifico prodotto

IMPORTANTE

CamelCamelCamel a volte fornisce codici **EAN** mancanti di una cifra finale **oppure**, in alcuni casi e in modo raro, rifiuta la **connessione dall'app**. In entrambe le situazioni **si potrebbe** avere come risposta in app **NESSUN RISULTATO**, anche se le recensioni ci sono sia su Amazon che su GoodReads.

Per risolvere il problema dell'**EAN** con la cifra finale mancante, se il prodotto non è un libro, memorizziamo come codice univoco l'**ASIN**. Questa soluzione viene adottata **solo** quando si cerca tramite testo. In questo modo, se l'utente seleziona la ricerca dalla **Home**, verranno ritrovate le recensioni a partire da **Amazon**.

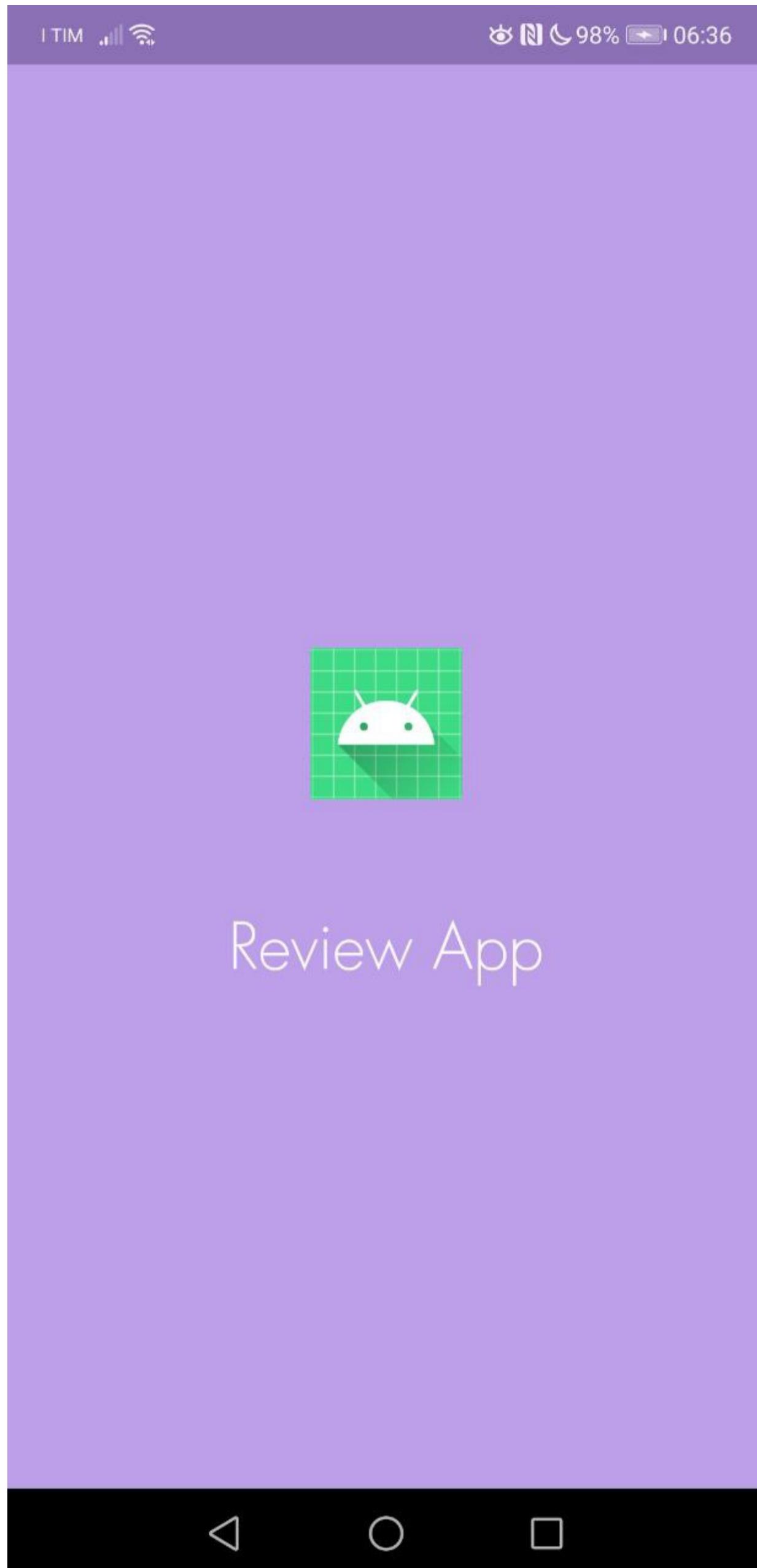
MULTITHREADING

Avendo come obiettivo quello di cercare le recensioni **contemporaneamente** su più marketplace, abbiamo pensato di costruire le classi, descritte nella sezione precedente, che su più **CoroutineScope** dedicate cercano prodotto e recensioni.

Questa scelta ha portato inevitabilmente all'utilizzo, in alcune situazioni, di meccanismi di gestione della concorrenza. Nel nostro caso abbiamo sempre usato un'istanza di **ReentrantLock**, con ordinamento **FIFO**, per gestire le **sezioni critiche**.

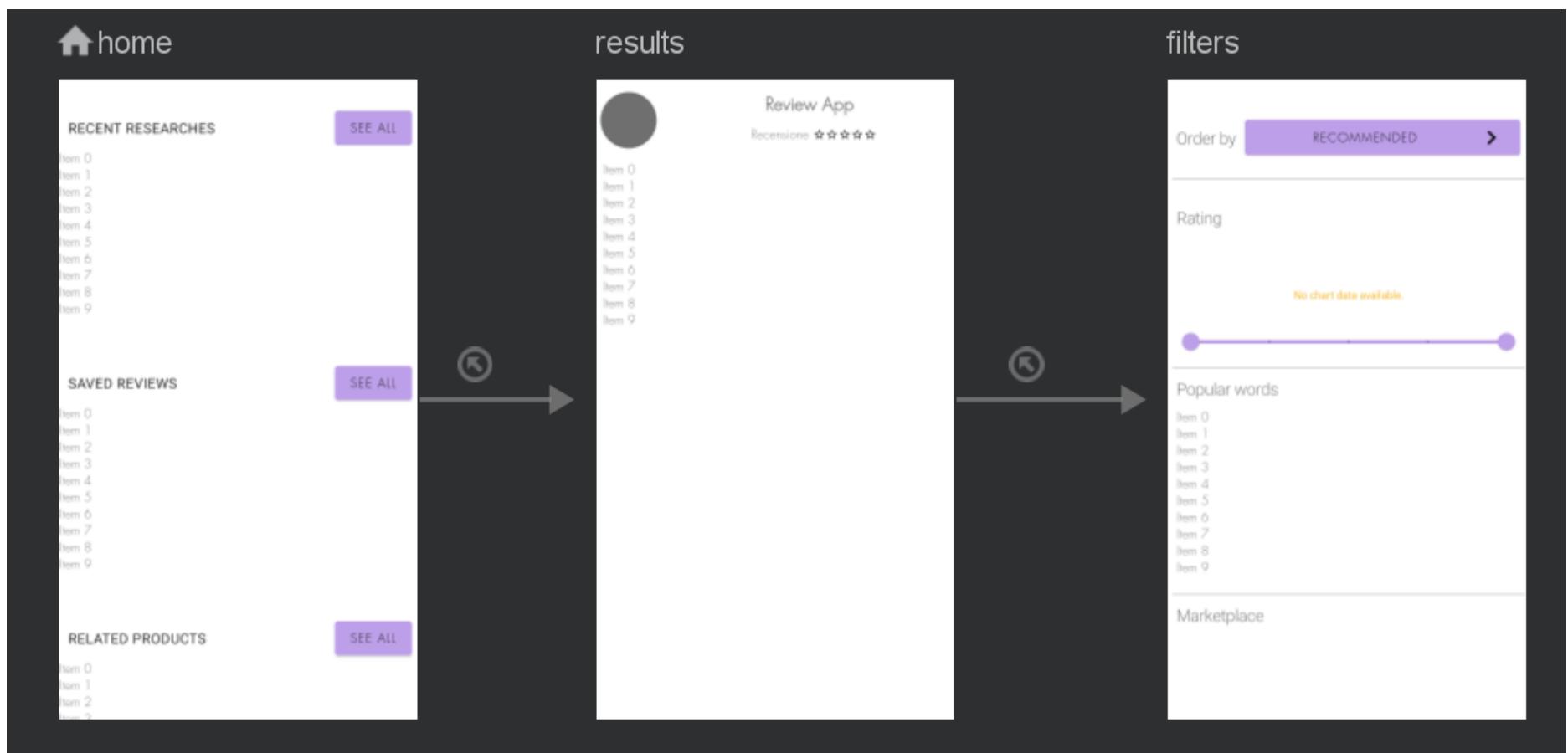
DESCRIZIONE COMPONENTI

L'App ha come entry point la classe **SplashActivity** che consente la comparsa di un'animazione iniziale:

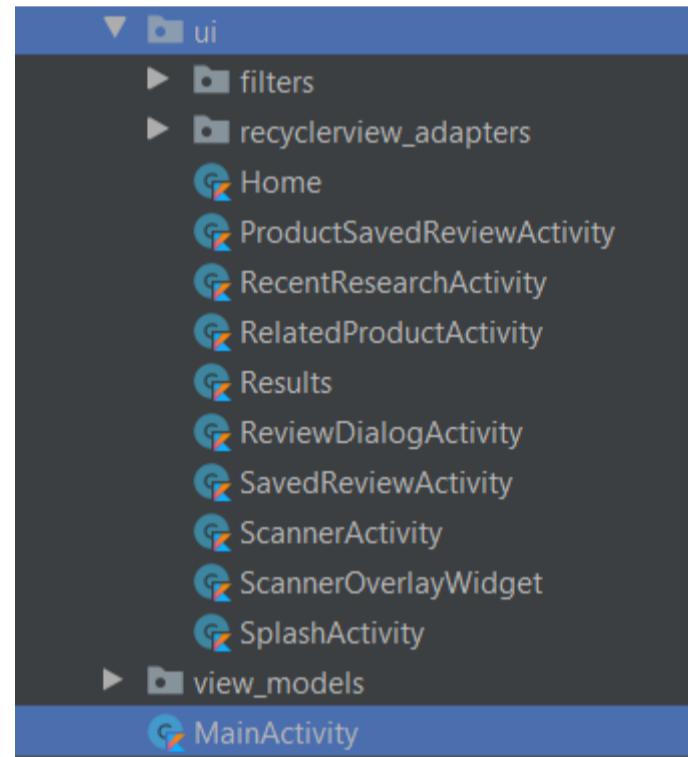


Per realizzare tale effetto abbiamo utilizzato la libreria [com.github.ViksaaSkool:AwesomeSplash](#)

Terminata l'animazione viene richiamata la classe **MainActivity** la quale contiene un **NavHostFragment**:



UI



MainActivity

MainActivity contiene tre fragment:

- **Home**

Include le 3 sezioni descritte nel paragrafo **1.1**. Ogni sezione con un numero di contenuti limitati è una **RecyclerView** che contiene all'interno della directory **recyclerview_adapters** il corrispondente adapter.

I dati che vengono iniettati nelle varie sezioni sono gestiti attraverso il View Model **HomeViewModel** che attraverso dei LiveData comunica direttamente con il database locale, passando attraverso la repository **DatabaseRepository**

- **Results**

Consiste principalmente di una RecyclerView dove al suo interno vengono iniettate le varie recensioni. Comunica con il View Model **ReviewViewModel** che fornisce al fragment, attraverso opportuni LiveData, la lista aggiornata e filtrata delle recensioni

- **Filters**

Il fragment si occupa di mostrare i filtri descritti nel paragrafo **1.1**.

Per realizzare il grafico che mostra il numero di recensioni per valutazione abbiamo utilizzato la libreria **com.github.PhilJay:MPAndroidChart**

Dall'AppBar è possibile cercare testualmente il prodotto o aprire l'Activity **ScannerActivity**.

La MainActivity è legata a **ReviewViewModel** e ogni **Fragment** fa riferimento **alla stessa istanza** del contenitore.

ScannerActivity

L'activity utilizza la libreria **com.google.android.gms.vision** per comunicare con i frames della fotocamera posteriore e rilevare eventuali qr codes. Non appena viene letto un qr code, questo viene comunicato alla MainActivity che, attraverso il **NavController**, naviga verso il fragment **Results**.

I frames della fotocamera sono gestiti attraverso l'utilizzo della componente grafica **SurfaceView**.

La classe **ScannerOverlayWidget** è figlia di **View**. All'interno della stessa abbiamo sovrascritto dei metodi per creare l'effetto opaco con il rettangolo trasparente al centro che si può notare avviando **ScannerActivity**.

Sempre all'interno di **ScannerActivity** abbiamo realizzato l'accesso al flash della fotocamera risalendo dall'oggetto **CameraSource**, tramite reflection, all'oggetto **android.hardware.Camera**, contenente i metodi necessari.

Il codice utilizzato per tale obiettivo è stato trovato su **Github**, ci siamo limitati a integrarlo e a comprendere la logica.

Abbiamo adottato questa soluzione vista l'impossibilità di accendere il flash tramite **Camersource**.

FRAGMENT HOME

Dalle 3 sezioni, premendo sul pulsante **Vedi Tutto** si naviga verso le Activities precedentemente descritte:

- **RecentResearchActivity**

Mostra **Tutti** i prodotti cercati a partire dal più recente.

Il meccanismo di cancellazione dei prodotti avviene attraverso un'**Action Mode**

- **ProductSavedReviewActivity**

Mostra i prodotti dei quali abbiamo salvato almeno una recensione. Selezionando un prodotto si passa a **SavedReviewActivity**

- **SavedReviewActivity**

Contiene la lista delle recensioni salvate associate al prodotto selezionato

Il meccanismo di cancellazione delle recensioni avviene attraverso un'**Action Mode**

Selezionando la recensione viene aperta l'Activity **ReviewDialogActivity**

- **RelatedProductActivity**

Contiene la lista dei prodotti suggeriti a partire dall'ultimo cercato

Dentro la directory **recyclerview_adapters** come accennato, troviamo gli adapters per gestire le varie RecyclerView:

- **RecentSearchAdapter**
- **ProductSavedReviewAdapter**
- **SavedReviewAdapter**
- **RelatedProductAdapter**

Tutte e 4 le classi sono figlie di **ListAdapter** per una gestione **più efficiente** della memoria

ReviewDialogActivity

L'activity comunica con il ViewModel **ReviewDialogViewModel** il quale gestisce il meccanismo di download dell'immagine del prodotto associato alla recensione, per la condivisione della stessa tramite Whatsapp ad esempio, e di cancellazione o salvataggio della recensione.

L'activity può essere richiamata da **SavedReviewActivity** o da **Results**. Nel primo caso l'utente può solo cancellare la recensione dai preferiti, nel secondo può sia salvarla che eliminarla dai preferiti.

Per visualizzare l'intero contenuto della recensione, in **ReviewDialogActivity** la stessa viene passata attraverso una stringa **JSON**, codificata a partire dal metodo `Json.encodeToString` della libreria **kotlinx.serialization** che, allo stesso modo, ricostruisce la **data class Review** attraverso il metodo `Json.decodeFromString`.

FRAGMENT RESULTS

Durante la ricezione delle recensioni, il fragment si occupa dell'aggiornamento della **RecyclerView**, della comunicazione diretta con **ReviewViewModel** per gestire situazioni come l'assenza di risultati.

La **RecyclerView** ha come adapter la classe **ReviewRecyclerViewAdapter**, che estende **ListAdapter** per ragioni di migliore efficienza.

Dentro **Results** ci sono i metodi necessari per capire quando l'utente sta per arrivare in fondo alla lista, in modo tale da poter comunicare al ViewModel la necessità di caricare altre recensioni.

Da Results si può navigare verso **Filters** solo quando cominciano a essere trovate delle recensioni

Durante il precaricamento delle recensioni e del prodotto viene utilizzato un particolare effetto grafico (**shimmer**) che abbiamo ottenuto includendo nel progetto la libreria [com.github.sharish:ShimmerRecyclerView](#)

Per limitare il numero di caratteri del titolo e del contenuto della recensione visualizzata nella lista, creando un effetto **a scomparsa**, abbiamo usato la libreria [com.github.bosphere.android-fadingedgelayout:fadingedgelayout](#)

Per la valutazione indicata con delle stelle, invece, abbiamo usato [com.github.ome450901:SimpleRatingBar](#)

FRAGMENT FILTERS

La classe è contenuta all'interno della directory **filters**.

Comunica con **ReviewViewModel** per ottenere la lista corrente delle recensioni e andare, in questo modo, ad aggiornare le varie componenti grafiche che si occupano di mostrare i possibili filtri.

Ad ogni nuova recensione ottenuta vengono aggiornati i seguenti filtri:

- Grafico del filtro per valutazione
- Lista delle parole ricorrenti
- Filtro per Marketplace

La lista delle **Parole Ricorrenti** viene gestita attraverso una **RecyclerView**, il cui adapter è la classe **FilterWordAdapter**, anch'essa figlia di **ListAdapter**.

Quando viene selezionato il filtro **Ordina Per** viene richiamato esplicitamente **FilterModeActivity**, che consente di selezionare il tipo di ordinamento che vogliamo nella lista dei risultati.

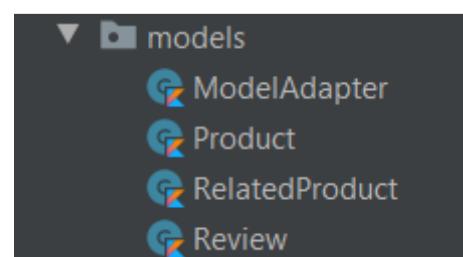
FilterModeActivity

Inietta delle componenti **RadioButton** all'interno di una **RadioGroup**, tante quanti sono gli elementi della enum class **OrderBy**, contenuta all'interno di **Filters.kt**.

Ogni qualvolta l'utente cambia l'ordinamento predefinito, questo verrà riproposto alla successiva ricerca.

Il salvataggio del dato avviene in **ReviewViewModel** utilizzando **android.content.SharedPreferences**.

MODELS



• Product

Data Class che descrive le proprietà generiche di un'entità Prodotto.

La proprietà `date` funge da timestamp.

• RelatedProduct

Data Class che descrive le proprietà generiche di un'entità Prodotto Correlato.

La proprietà **scheduledForNotification** serve a **NotificationWorker** per capire se quel prodotto era già stato suggerito all'utente tramite le notifiche periodiche.

• Review

Data Class che descrive le proprietà e i metodi generici di un'entità Recensione.

La proprietà **productCode** serve per identificare in modo univoco il prodotto alla quale è associata.

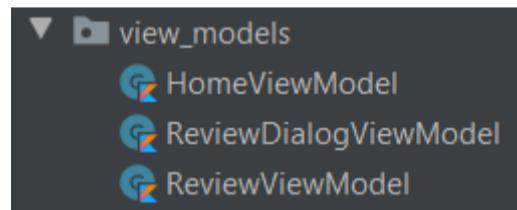
Il metodo **wasStored** verifica, fornita una lista di recensioni, se questa la contiene.

Il metodo statico **preloaderItem** serve per generare una recensione da usare con l'effetto grafico "**Shimmer**".

• ModelAdapter

Genitore di **Product** e **Review**, contiene l'unica proprietà **isSelected**, che consente di determinare durante la selezione per cancellare gli oggetti i-esimi, se questi erano stati scelti.

VIEW MODELS



• ReviewDialogViewModel

Contiene due metodi:

- `saveImage`

Comunicando con **NetworkRepository** attiva il metodo `saveLocally` che consente di salvare l'immagine associata al prodotto in una directory locale, in modo tale da poter condividere la recensione stessa su social come Instagram/Whatsapp.

- `storeReview`

Comunicando con **DatabaseRepository** consente di cancellare o salvare una recensione.

• HomeViewModel

Si occupa di comunicare con **DatabaseRepository** per fornire attraverso dei **LiveData**, alle varie activities/fragments che lo implementano, i dati necessari per aggiornare le rispettive liste.

I metodi `deleteResearches` e `deleteReviews` consentono inoltre di eliminare rispettivamente ricerche e recensioni.

• ReviewViewModel

Si occupa di istanziare la repository **MarketplaceRepository**, e all'interno di `init` istanzia la comunicazione con il metodo statico `Settings.isOnLine`, che va a controllare lo stato della connessione a internet aggiornando il **LiveData** `Settings.networkAvailable`. Sempre all'interno di `init` viene recuperato l'ordinamento predefinito.

Si occupa di inizializzare una nuova richiesta ai marketplace e di filtrare la lista di risultati che mano a mano viene aggiornata da **MarketplaceRepository**. Per fare questo abbiamo introdotto il **MediatorLiveData** `reviewListener`.

`reviewListener` :

ad ogni nuova recensione ricevuta verifica se l'oggetto fornito è `null`. Se `null` significa che `MarketplaceRepository` ha interrotto la ricerca. Se non è `null` significa la lista delle recensioni trovate è cambiata, e dunque viene aggiornata la lista delle recensioni `filtrate`, attraverso il metodo `applyFilters`.

`applyFilters` :

contiene una `ReentrantLock(true)` per evitare problemi di concorrenza durante l'accesso a dati condivisi nello stack.

I filtri vengono applicati in `background` su un thread parallelo, e di conseguenza la lista delle recensioni potrebbe essere aggiornata. A questo punto `reviewListener` può chiamare nuovamente `applyFilters`, e se un altro thread stava lavorando su variabili comuni, si andrebbe a generare un errore fatale.

`ReviewViewModel`, inoltre, contiene altri metodi come `scrollMore`, che consente di capire se sono disponibili altre recensioni da poter estrapolare, e `storeReview`, che consente direttamente dalla lista dei risultati di salvare o cancellare una recensione dai preferiti.

`onStart` :

inizializza una nuova ricerca oppure consente di riprenderla

REPOSITORIES

- **NetworkRepository**

Comunica con le interfacce della directory `network`.

Istanzia i vari servizi di rete attraverso il metodo `retrofit`.

Per ogni richiesta alla rete utilizza come **User-agent** il seguente:

Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0

Il metodo `get`, invece, gestisce la chiamata verso il server web, verificando che sia presente una connessione attiva, e che non si verifichino errori. Nel caso ci dovessero essere dei problemi legati alla connessione del dispositivo o il codice della risposta del server dovesse essere diverso da `200`, allora il Thread corrente viene temporaneamente sospeso, per poi riattivarsi periodicamente e ritentare la chiamata.

- **DatabaseRepository**

La classe comunica con i metodi della directory `database`.

Il metodo `storeReview` è una **sezione critica**, e questo perché se l'utente clicca rapidamente sull'icona per salvare/rimuovere dai preferiti la recensione, potrebbero verificarsi dei conflitti all'interno del database, essendo l'operazione effettuata nel db contenuta all'interno di una `CoroutineScope`.

- **MarketplaceRepository**

Contiene i metodi fondamentali per gestire il multithreading per la ricerca sui marketplace implementati.

Contiene due variabili fondamentali:

```

private val mapMarketplaceToObjects: HashMap<Marketplace, KClass<MarketplaceObject>> =
    hashMapOf(
        Marketplace.AMAZON to Amazon::class as KClass<MarketplaceObject>,
        Marketplace.GOODREADS to GoodReads::class as KClass<MarketplaceObject>
    )

private val mapMarketplaceToIterator: HashMap<Marketplace, ReviewListIterator?> =
    hashMapOf(
        Marketplace.AMAZON to null,
        Marketplace.GOODREADS to null
    )

```

in `mapMarketplaceToObjects` ogni Marketplace viene mappato nella classe corrispondente, presente all'interno della directory **reviews**, specializzazione della classe astratta **MarketplaceObject** che definisce i metodi fondamentali che ogni Marketplace deve implementare.

`mapMarketplaceToIterator` contiene riferimenti alla classe `{Nome_Marketplace}ListIterator`, che contiene i metodi per estrapolare le ricerche.

KClass ci ha semplificato la scrittura di codice evitando di utilizzare meccanismi classici Java di referenziazione delle classi come, ad esempio, la sintassi `{classe}::class.java`

Per usare questo oggetto abbiamo importato la libreria `org.jetbrains.kotlin:kotlin-reflect`

Ogni qualvolta viene inizializzata o ripresa una ricerca, viene chiamato il metodo `onStart`

`onStart` :

nel caso di nuova ricerca popola la variabile `listOfObjects` con i marketplace che si devono istanziare e setta a **null** gli iterator, per poi chiamare il metodo `getReviews`. Nel caso si riprenda una ricerca interrotta si chiama il metodo `updateReviews`

`getReviews` :

per ogni Marketplace in `listOfObjects` chiama il metodo `initJob`

`updateReviews` :

per ogni Iterator in `listOfIterators` chiama il metodo `initUpdateJob`

`initJob` :

per ogni Marketplace passato ne istanzia la classe figlia di **MarketplaceObject** in un thread separato

```

val OBJ = reference.constructors.first().call(code, isSpecificCode, isAsin)
val init = OBJ.init()

```

Se il marketplace contiene il prodotto, oltre a chiamare `setCurrentProduct`, inizializza l'iteratore corrispondente e chiama il metodo `iterate` di `CoroutineScope`, che abbiamo implementato come **extension function**

```

val iterator = OBJ.getReviews().iterator()
listOfIterators[marketplace] = iterator
this.iterate(iterator, max = maxResults + beginning, context)

```

`setCurrentProduct` :

aggiorna il prodotto corrente nel database locale. Nel caso in cui due o più marketplace trovino il prodotto, nel db verrà aggiornata la colonna **feedback** attraverso una media aritmetica dei feedback trovati

`CoroutineScope.iterate` :

per ogni iteratore passato chiama il metodo `hasNext` e `next` per ottenere le varie recensioni, aggiornando così la lista dei risultati.

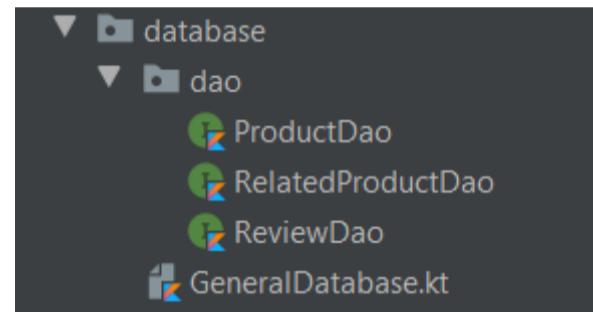
`initUpdateJob` :

per ogni oggetto in `listOfIterators` richiama il metodo `CoroutineScope.iterate`

MarketplaceRepository contiene anche un metodo chiamato `checkPopularWords`, che viene eseguito su un thread separato. Viene chiamato ad ogni nuova recensione ottenuta per aggiornare la lista delle parole ricorrenti, che viene poi usata da **Filters**.

Le parole trovate possono contenere termini che non sono effettivamente sostantivi.

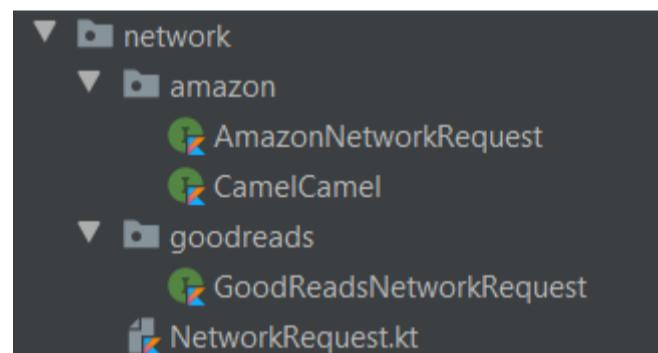
DATABASE



Nella directory **database** troviamo la classe **GeneralDatabase** che istanzia il database attraverso la libreria **Room**.

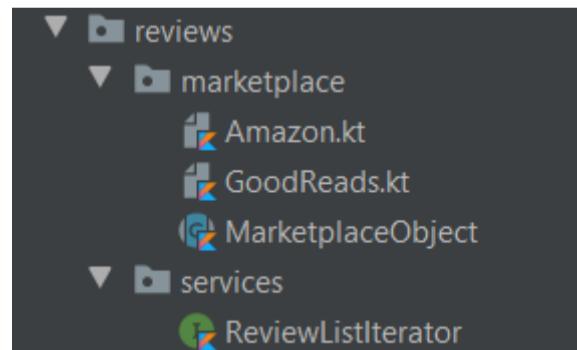
Dentro **dao** ci sono le varie query per comunicare con le entità del database locale.

NETWORK



Utilizza la libreria **retrofit2** per definire le interfacce di comunicazione con il web.

REVIEWS



Per ogni marketplace troviamo la classe {Nome_Marketplace}, dove all'interno dell'omonimo file troviamo la classe {Nome_Marketplace}ListIterator, che implementa l'interfaccia `ReviewListIterator`.

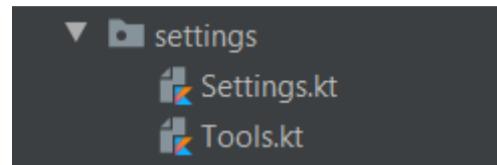
Le classi `Amazon` e `GoodReads`, figlie di `MarketplaceObject`, contengono i metodi necessari **alla ricerca del prodotto nel corrispondente marketplace**.

Le classi `AmazonListIterator` e `GoodReadsListIterator` contengono i metodi necessari **al recupero delle recensioni nel corrispondente marketplace**.

Particolari problematiche sono state riscontrate con il contenuto HTML dinamico restituito da **Amazon** per i diversi prodotti.

Per tale ragione abbiamo spesso usato **Regular Expressions** piuttosto che metodi propri di **Jsoup** per ricavare **direttamente l'elemento** del dom.

SETTINGS



Nella directory **settings** troviamo due classi:

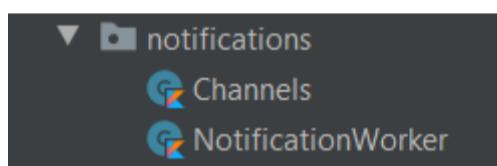
- **Settings**

Contiene costanti o metodi statici utili per modificare il comportamento dell'app senza dover intervenire su singole classi

- **Tools**

Contiene metodi statici utili a più classi, come `setImage`, che consente di specificare come va caricata all'interno di un' `ImageView` un url

NOTIFICATIONS



- **Channels**

Crea i canali per istanziare delle notifiche da parte dell'Android runtime.

Contiene il metodo statico `scheduleNotification` che consente di richiamare il metodo `enqueueUniquePeriodicWork` di `WorkRequest` per notificare all'utente, periodicamente, prodotti ai quali potrebbe essere interessato a vederne le recensioni.

La notifica si attiva solo se l'utente ha una connessione internet attiva.

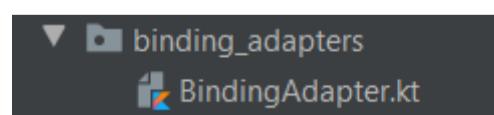
- **NotificationWorker**

La classe specializza **CoroutineWorker**. Consente di verificare nel database se ci sono prodotti suggeriti da poter mostrare all'utente. Se sì, dopo aver eventualmente scaricato l'immagine del prodotto, crea un oggetto `NotificationCompat.Builder` che notifica al **NotificationManager**.

Comunica sia con **DatabaseRepository** che con **NetworkRepository** pur non essendo un **ViewModel**. Questa scelta è stata fatta perché abbiamo ritenuto ridondante costruire metodi già presenti nelle due classi citate.

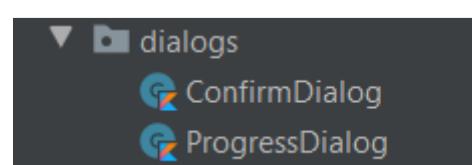
ALTRE DIRECTORIES

- **binding_adapters**



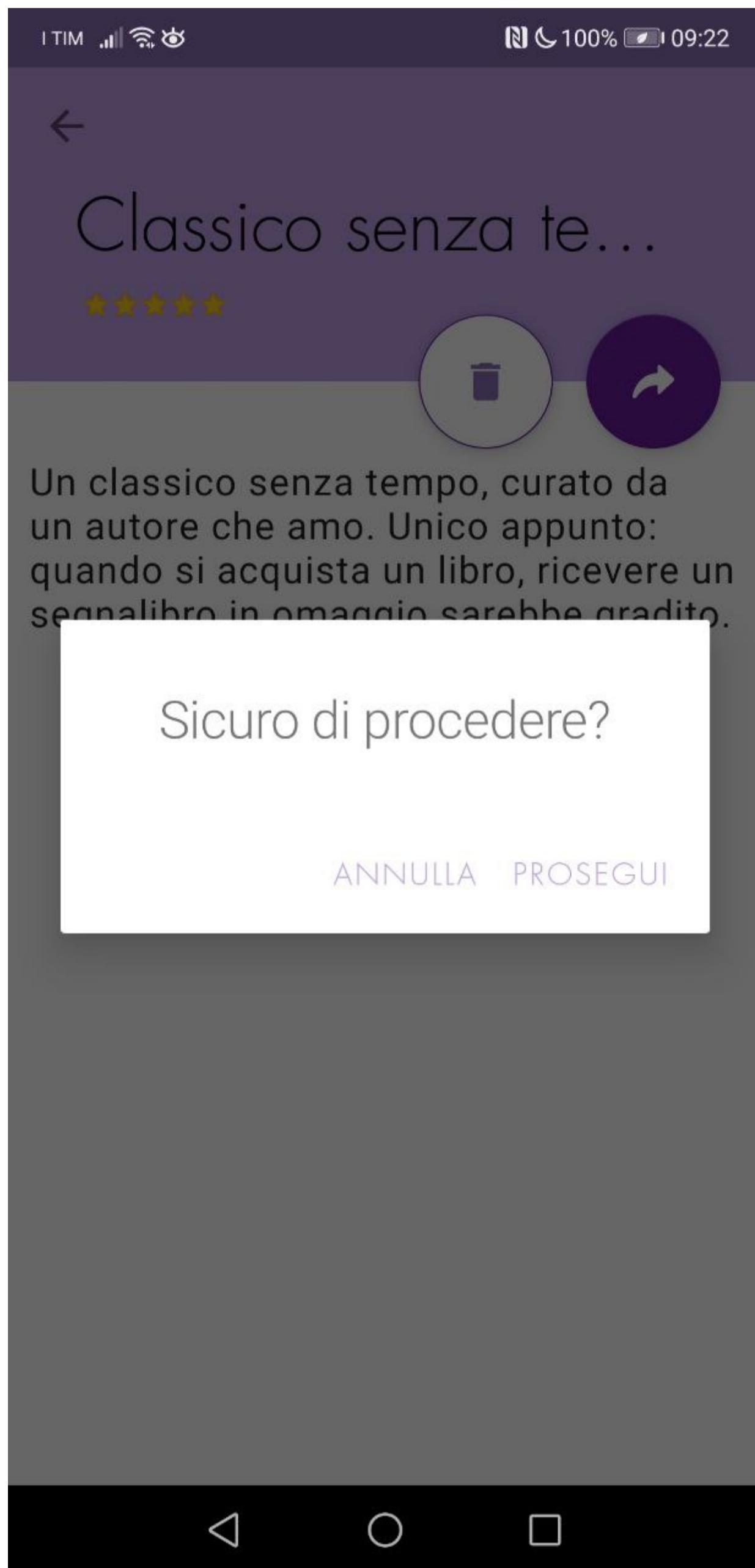
Nel file **BindingAdapter** ci sono i metodi per creare nuovi BindingAdapters usando l'annotation `@BindingAdapter`

- **dialogs**



- **ConfirmDialog**

Estende **AppCompatDialogFragment** e consente di creare un dialogo per far confermare o negare all'utente qualcosa:



- **ProgressDialog**

Estende **Dialog** e consente di creare un dialogo per mostrare all'utente una **ProgressBar** durante l'eliminazione di prodotti/recensioni dal database attraverso le activities **SavedReviewActivity** e **RecentResearchActivity**

"BUG" APP

Come descritto all'inizio di questa sezione, dal momento l'app si appoggia all'estrapolazione del dom di contenuti HTML, si potrebbero verificare problemi quando **Amazon** o **GoodReads**, in alcuni url, non rispecchiano la struttura di base che abbiamo analizzato per estrarre informazioni. In particolare **Amazon** cambia dinamicamente la struttura HTML della pagina, di conseguenza potrebbero verificarsi, seppur molto raramente, dei possibili malfunzionamenti. Inoltre, **CamelCamelCamel** e **GoodReads** potrebbero temporaneamente rifiutare la connessione e generare possibili risultati nulli. **Tuttavia**, non abbiamo **mai** sperimentato questo tipo di problema.

MALFUNZIONAMENTO CHE NON RIGUARDA LA VERSIONE CORRENTE DEL PROGETTO

L'altro principale problema è dato da **CamelCamelCamel** quando restituisce, a partire dall'**Asin**, un codice **EAN** privato della cifra finale. In quest'ultimo caso, che avviene durante una ricerca testuale, l'app non troverà le recensioni di quel particolare prodotto su altri marketplace. Nel nostro caso, usando solo **Amazon** e **GoodReads**, non avremo problemi. **GoodReads** infatti cerca tramite **ISBN** e, dunque, non è importante il problema della cifra finale **EAN** di CamelCamelCamel.

LOGIN - REGISTRAZIONE - FIREBASE

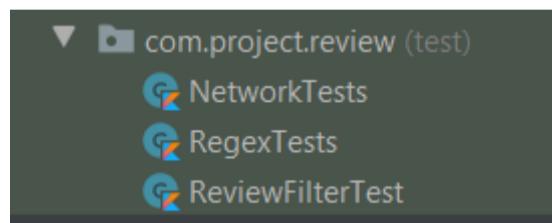
Dentro l'app abbiamo cominciato a realizzare dei layout per integrare, tramite Firebase, il meccanismo di login e registrazione.

Non abbiamo però proseguito con la definitiva implementazione.

5.0 - TESTING

JUnit

Con **JUnit** abbiamo realizzato 3 diverse classi per il test di alcune delle funzionalità dell'app:



- **NetworkTests:**

Include 2 test:

- `getAmazonProduct :`

Istanzia un oggetto **Amazon** e verifica che il nome del prodotto ottenuto sia effettivamente quello presente su amazon

- o `getGoodReadsProduct` :

Istanzia un oggetto **GoodReads**.

Verifica che il nome del prodotto ottenuto sia effettivamente quello presente su GoodReads e verifica che l'url corrisponda.

- **RegexTests:**

Include 2 test:

- o `testAmazonReviews` :

Partendo dalla stringa **x,y recensioni**, con **x,y** numeri naturali, verifica che il metodo `Amazon.filterStars` restituisca

un valore float valido

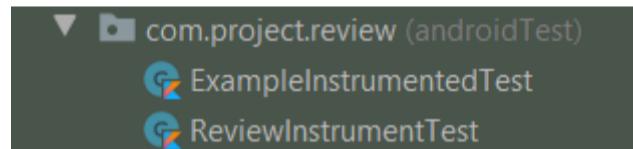
- o `testWords` :

Partendo da una descrizione, verifica che vengano estratte correttamente le **parole**, seppur, come già citato, non essendo preciso come algoritmo.

- **ReviewFilterTest:**

Istanzia 3 recensioni, le filtra secondo certi parametri, e verifica che la recensione in testa alla lista filtrata sia quella giusta.

TEST UI



ReviewInstrumentTest

Oltre alla libreria `androidx.test.espresso:espresso-core`, abbiamo anche incluso `androidx.test.espresso:espresso-contrib` che ci ha consentito di implementare metodi come `RecyclerViewActions.actionOnItemAtPosition`

Il test consiste nel cercare un prodotto partendo dal fragment **Home** e verificando che, salvando una recensione, questa poi sia nuovamente accessibile partendo dalla **Home**.

Dopo aver inizializzato la ricerca, si attende che vengano mostrati i primi risultati e, a quel punto, viene premuta la prima recensione sulla **RecyclerView**. Si apre la **ReviewDialogActivity** e il test procede cliccando sul pulsante per salvare la recensione. L'app torna indietro fino alla **Home**, dove cerca la seconda sezione. A questo punto seleziona il primo prodotto della **RecyclerView**, e all'interno dell'activity **SavedReviewActivity** cerca se presente quella salvata. In caso positivo il test viene superato.

Per riuscire a posizionarci, sulla **Home**, sulla seconda RecyclerView, abbiamo dapprima provato il metodo di espresso `perform(scrollTo())`, ma quando la lista diventa completamente nascosta durante la prima costruzione del fragment, poiché in prima posizione troviamo le ricerche recenti, veniva restituita la seguente eccezione:

```
java.lang.RuntimeException: Action will not be performed because the target view does not match one or more of the
following constraints:
(view has effective visibility=VISIBLE and is descendant of a: (is assignable from class: class
android.widget ScrollView or is assignable from class: class android.widget.HorizontalScrollView or is assignable
from class: class android.widget.ListView))
```

Nel nostro caso la **RecyclerView** è situata all'interno di una **NestedScrollView**.

Per risolvere, abbiamo usato il metodo `scrollTo(id)` della libreria `com.schibsted.spain:barista`.

2. SVILUPPO IN FLUTTER

2.1 - REQUISITI E DESCRIZIONE DEI CASI D'USO

L'app, partendo da una Home che funge da schermata primaria per accedere alle varie funzionalità (1), consente di cercare le recensioni su **Amazon** a partire da un codice a barre scansionato da un determinato prodotto (2), oppure attraverso una semplice ricerca testuale dell'articolo stesso (3).

Una volta trovate le recensioni per un certo articolo (4) l'utente può **filtrarle** (5), salvarle tra i preferiti e, previa selezione, condividerla con un contatto via Whatsapp, Telegram, Email ecc..

I dati vengono **esclusivamente** salvati nel database locale.

L'utente può perdere la connessione durante una ricerca, questa viene ripresa non appena l'utente ha campo sufficiente per ristabilire la connessione.

1) HOME

L'**Home** dell'app mostra i **Prodotti di cui abbiamo salvato almeno una recensione**, ordinati da quelli cercati più recentemente.

Premendo su un articolo verranno visualizzate le recensioni salvate, che possono essere cancellate o condivise.

2) CODICE A BARRE

Il codice a barre può essere di qualunque tipo: ISBN, EAN-13, EAN-8, UPC... e viene letto attraverso l'uso della fotocamera posteriore del cellulare.

3) IMPLEMENTAZIONE RICERCA TESTUALE

La ricerca testuale viene gestita attraverso il **primo** prodotto che viene trovato sul marketplace **Amazon** al seguente url:

<https://amazon.it/s?k={stringa}>

Questa scelta è stata fatta per riuscire a **identificare univocamente** un prodotto ricercato, data l'ambiguità dei risultati che possono scaturire cercando un articolo in maniera generica, ad esempio indicando semplicemente il termine "cellulare" nella stringa di ricerca.

3) RICERCA RECENSIONI E VISUALIZZAZIONE DEI RISULTATI

Una volta ottenuto il codice a barre oppure la stringa di ricerca, l'app provvede alla ricerca delle recensioni su **Amazon**

Le recensioni trovate hanno tutte da 1 a 5 stelle.

La ricerca, che avviene in **background**, visita le varie pagine di recensioni fornite dal marketplace e, come viene fornito il contenuto HTML, ne estrae i dati e aggiorna la lista corrente.

Le recensioni vengono ordinate, di **default**, a seconda di come le fornisce sequenzialmente Amazon.

La lista mostra **tutte** le recensioni mano a mano che vengono trovate.

Ogni recensione in questa ricerca mostra un numero limitato di caratteri.

Selezionando la recensione desiderata, però, l'effetto sarà quello di navigare verso il **Widget** dedicato interamente alla visualizzazione del contenuto della recensione.

L'utente può, direttamente dalla lista dei risultati, salvare o eliminare la recensione dai preferiti

La recensione **può** presentare, nel suo corpo, contenuto **html**, questo viene interpretato graficamente durante la visualizzazione della stessa.

esempio:



- COMODISSIME -
FUNZIONALI AL 100% - ...



Bluetooth: - - - - 5.0

Colore:- - - - - Rosso

Sistema: - - - - - Bluetooth 5.0 TWS

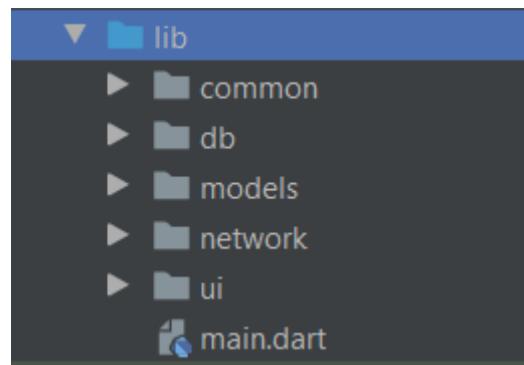
Impermeabili:- - - Certificazione
IPX7



5) FILTRI

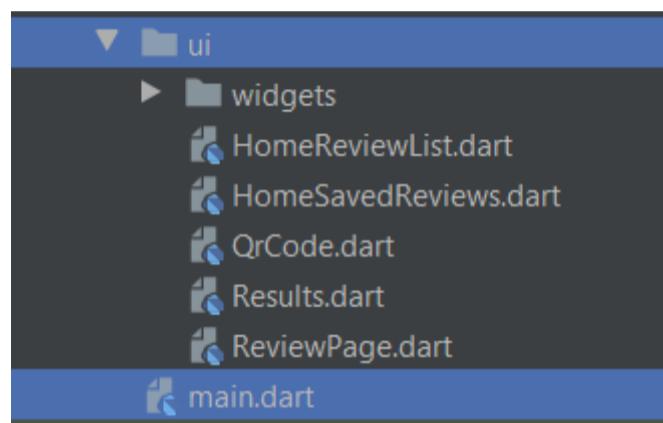
L'utente può filtrare i risultati cercando del contenuto all'interno del **corpo** della recensione stessa, attraverso il click su un'apposita icona di ricerca che viene mostrata durante la visualizzazione dei risultati.

2.2 - ARCHITETTURA E COMPONENTI AD ALTO LIVELLO

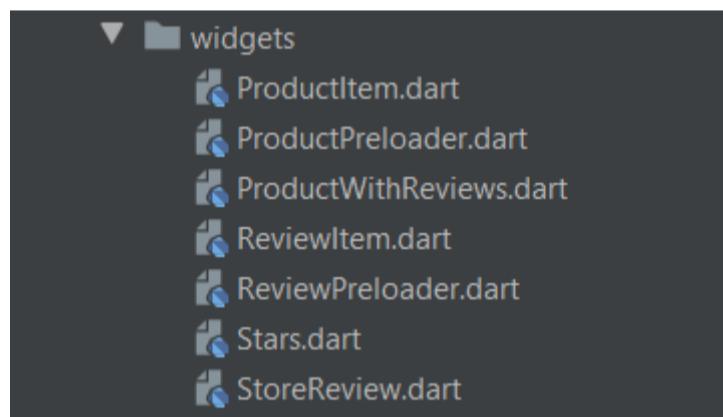


L'app è stata organizzata secondo il pattern **MVC**.

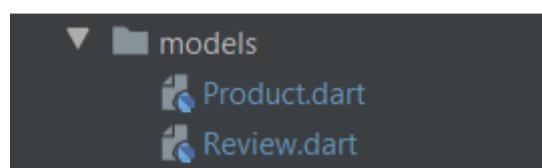
Le viste primarie sono organizzate dentro la directory **ui**:



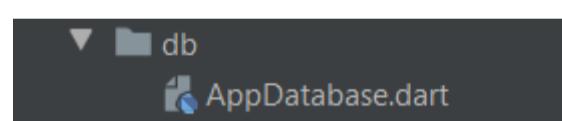
Le varie componenti grafiche, come il widget per visualizzare le recensioni trovate, sono organizzate dentro **widgets**:



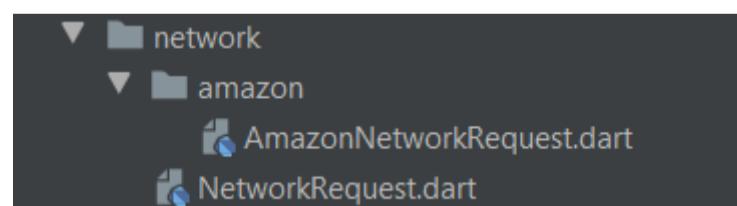
I Modelli dai quali siamo partiti per poi sviluppare il database e le varie funzionalità dell'app sono 2: **Product** e **Review**



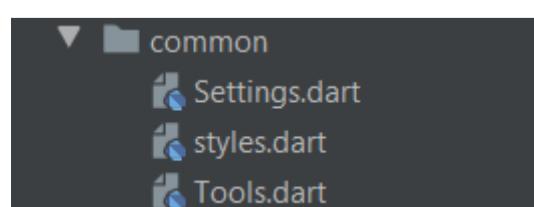
Nella classe **AppDatabase** abbiamo inserito l'inizializzazione del database e i metodi necessari per comunicare con le entità.



Nella directory **network** ci sono i metodi per comunicare con i server web, in questo caso **Amazon** e **CamelCamelCamel**



in **common** abbiamo messo le classi che servono per definire proprietà e metodi generali dell'app, come i temi in **styles.dart**

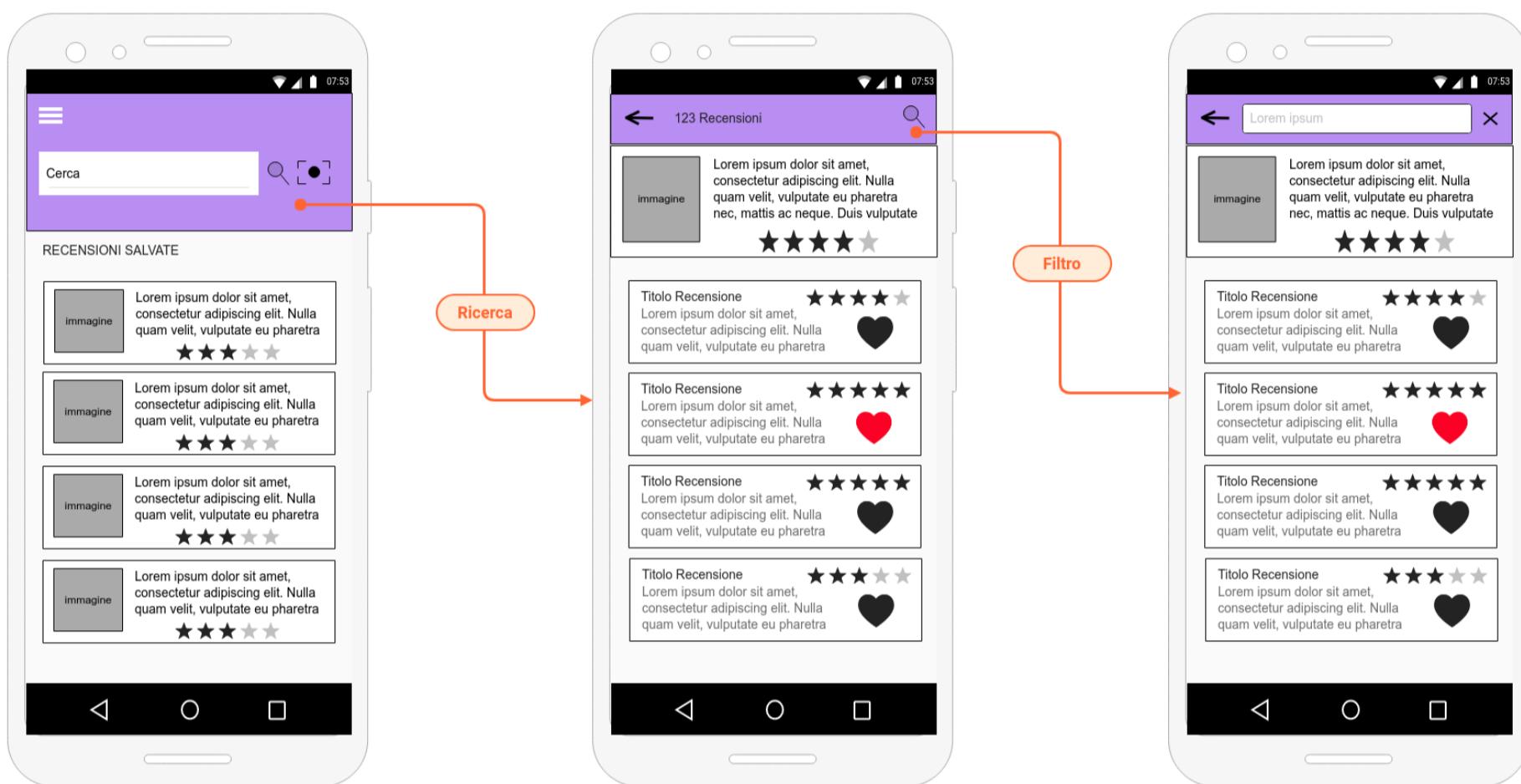


2.3 - MOCKUP E RAPPRESENTAZIONE GRAFICA DEI CASI D'USO

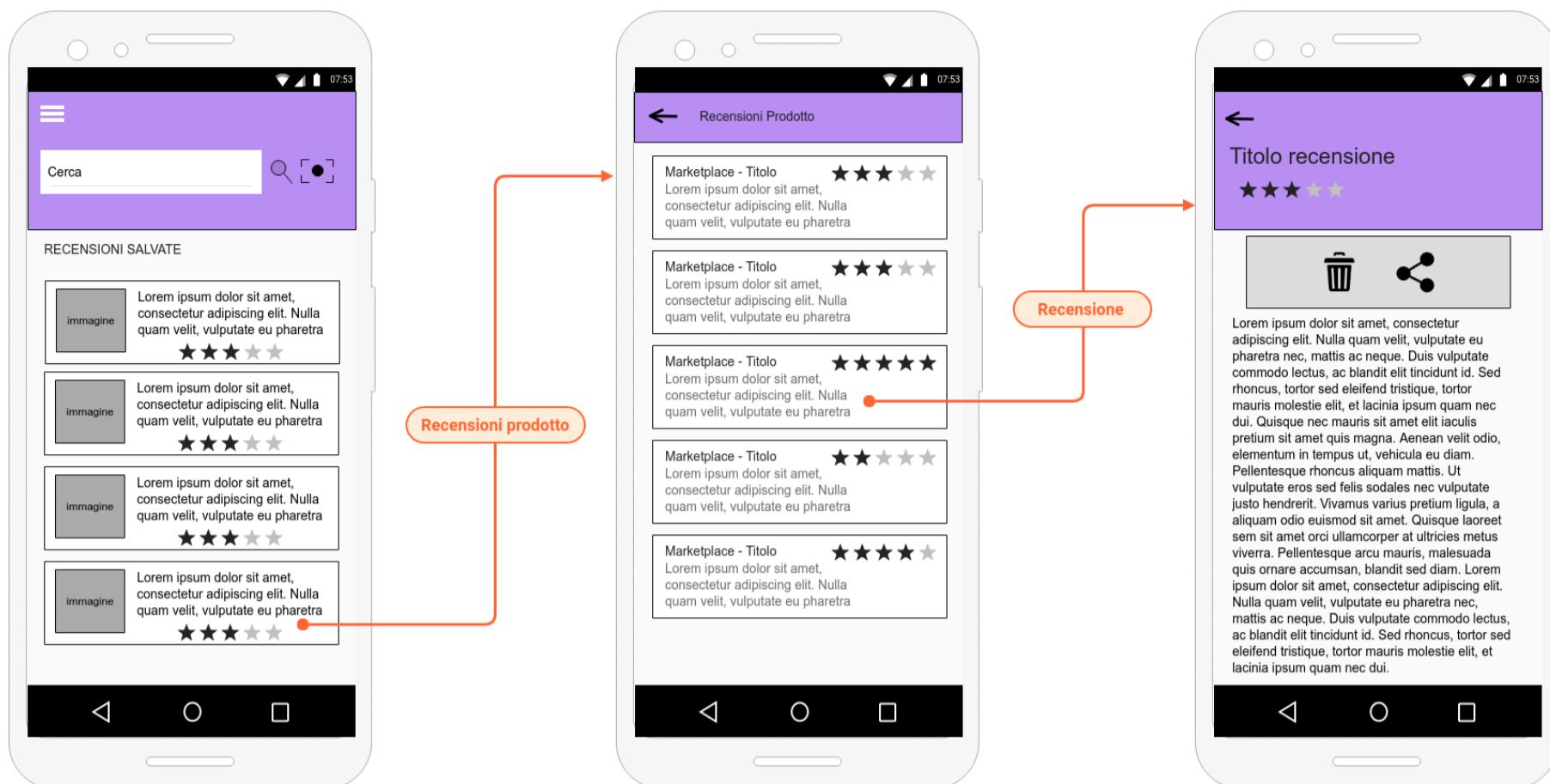
L'app è stata realizzata con due temi, uno light e uno dark.

Di seguito il mockup delle interfacce utente:

Ricerca Recensioni



Visualizzazione Recensioni Salvate

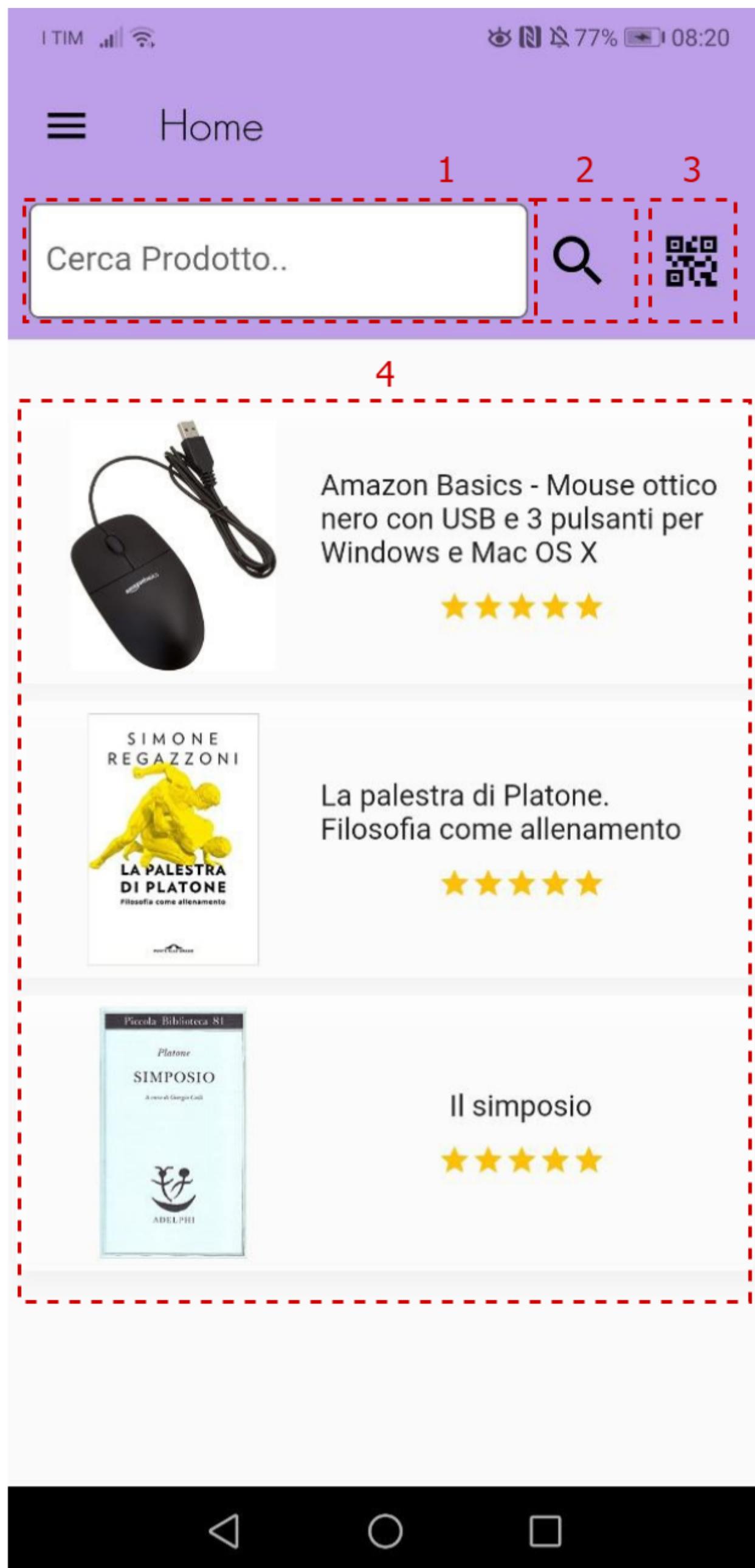


STILE DARK:



RAPPRESENTAZIONE GRAFICA DEI CASI D'USO

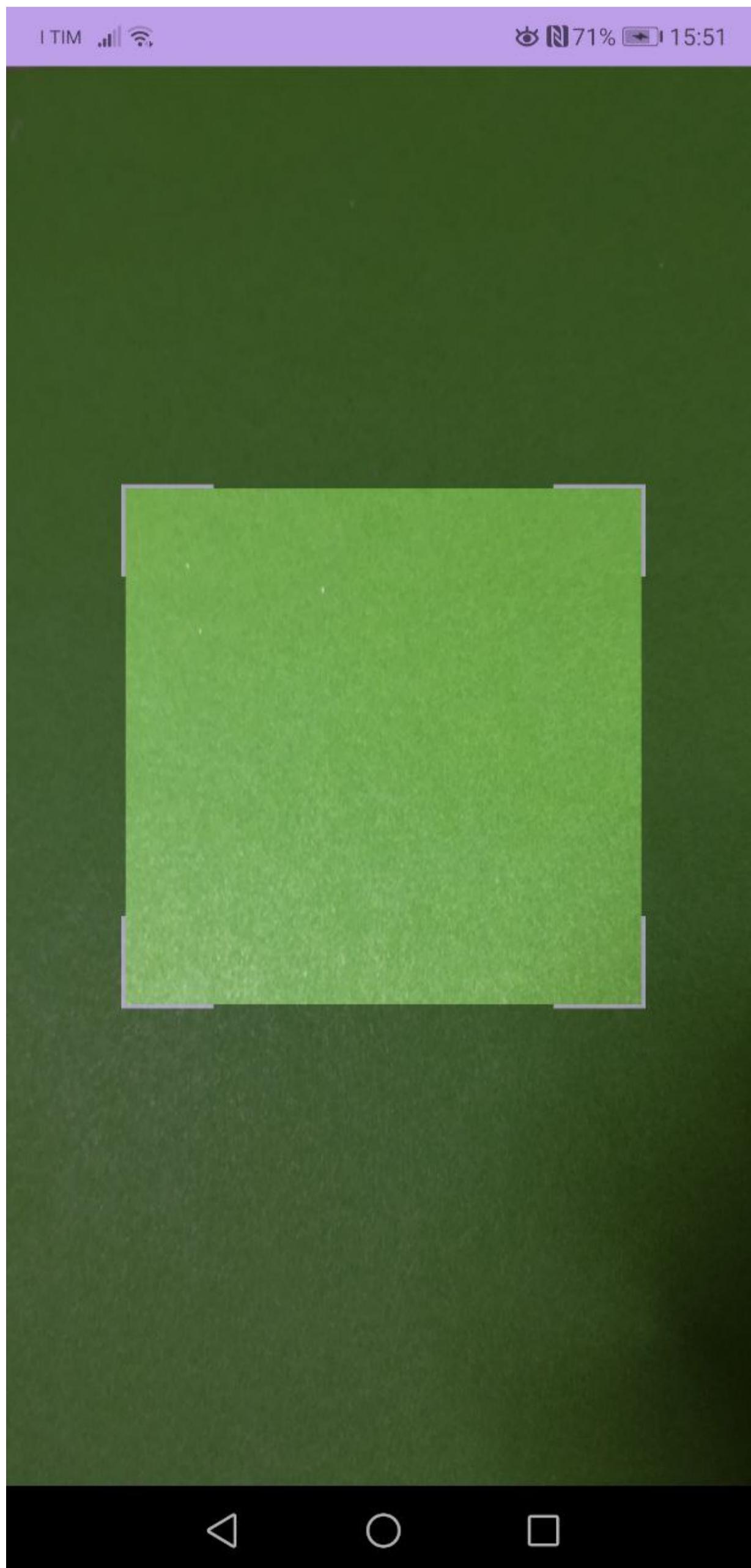
HOME



1. CONSENTE DI CERCARE TESTUALMENTE IL PRODOTTO

2. CONSENTE DI AVVIARE LA RICERCA TESTUALE

3. APRE IL WIDGET PER RILEVARE IL CODICE A BARRE



4. APRE IL WIDGET CHE MOSTRA LE RECENSIONI SALVATE



Tenendo premuto a lungo sulla recensione, si ha la possibilità di selezionare quelle da cancellare



Cancella Recensioni



Classico senza tempo.



Un classico senza
tempo, curato da un
autore che amo. Unico
appunto: qu...



Considerazioni alternative...



Questo saggio dell'antica
tradizione ellenica mette
al centro uno dei c...



Il Simposio



Libro letto per la prima
volta a 14 anni... andato
perduto... e ricompr...



Selezionando semplicemente la recensione, invece, compare il Widget dedicato:



Classico senza tempo.



Un classico senza tempo, curato da un autore che amo. Unico appunto: quando si acquista un libro, ricevere un segnalibro in omaggio sarebbe gradito.



ITIM 100% 11:26

← 62 Recensioni Trovate 3

Il simposio

★★★★★

1 Considerazioni alternative rispett...
Questo saggio dell'antica tradizione ellenica mette al centro uno dei c...

★★★★★ 2

Assolutamente da Leggere.
Arrivato con un giorno di anticipo, e il libro è uno di quelli da tener...

★★★★★

Classico senza tempo.
Un classico senza tempo, curato da un autore che amo.
Unico appunto: qu...

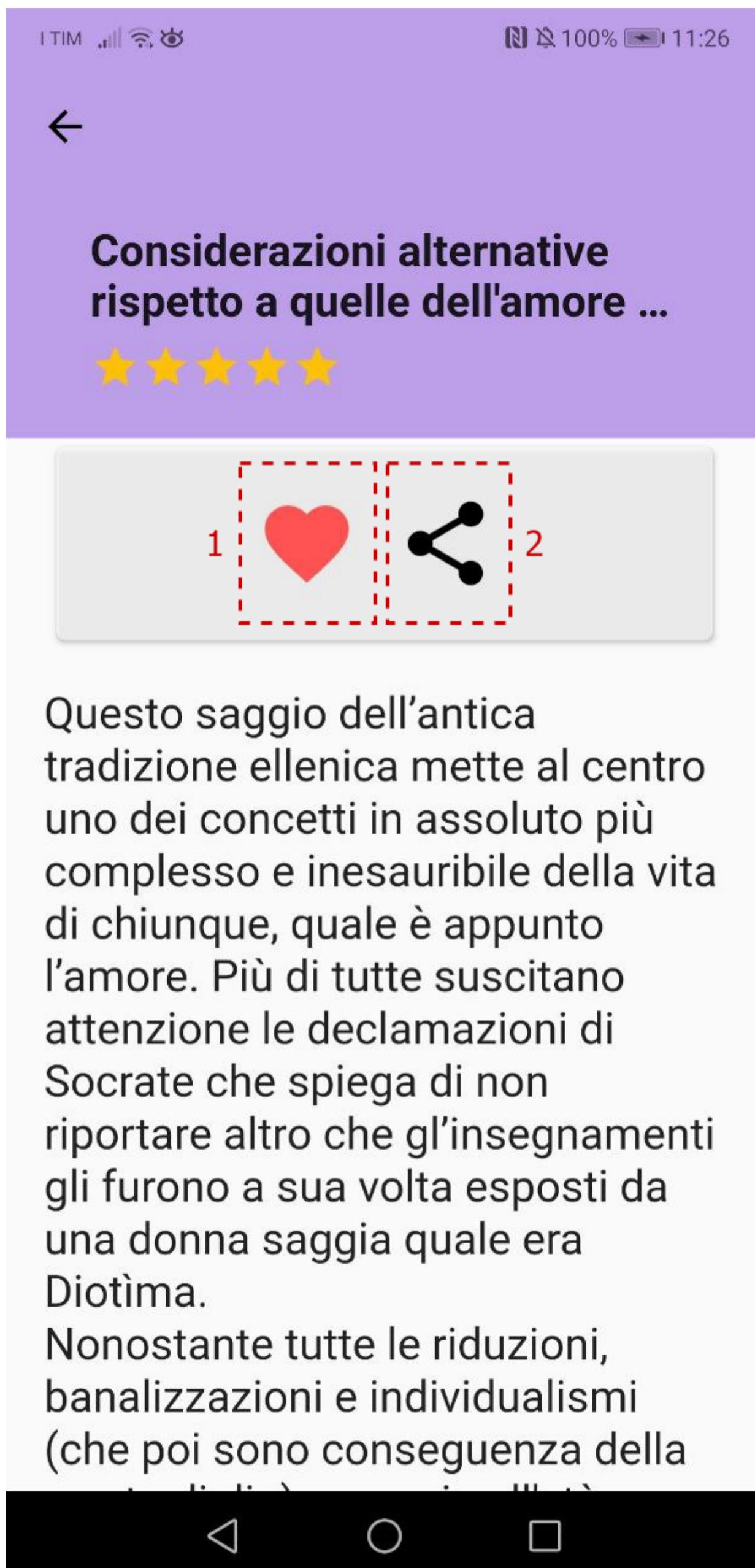
★★★★★

Il Simposio
Libro letto per la prima volta a 14 anni... andato perduto... e ricompr...

★★★★★

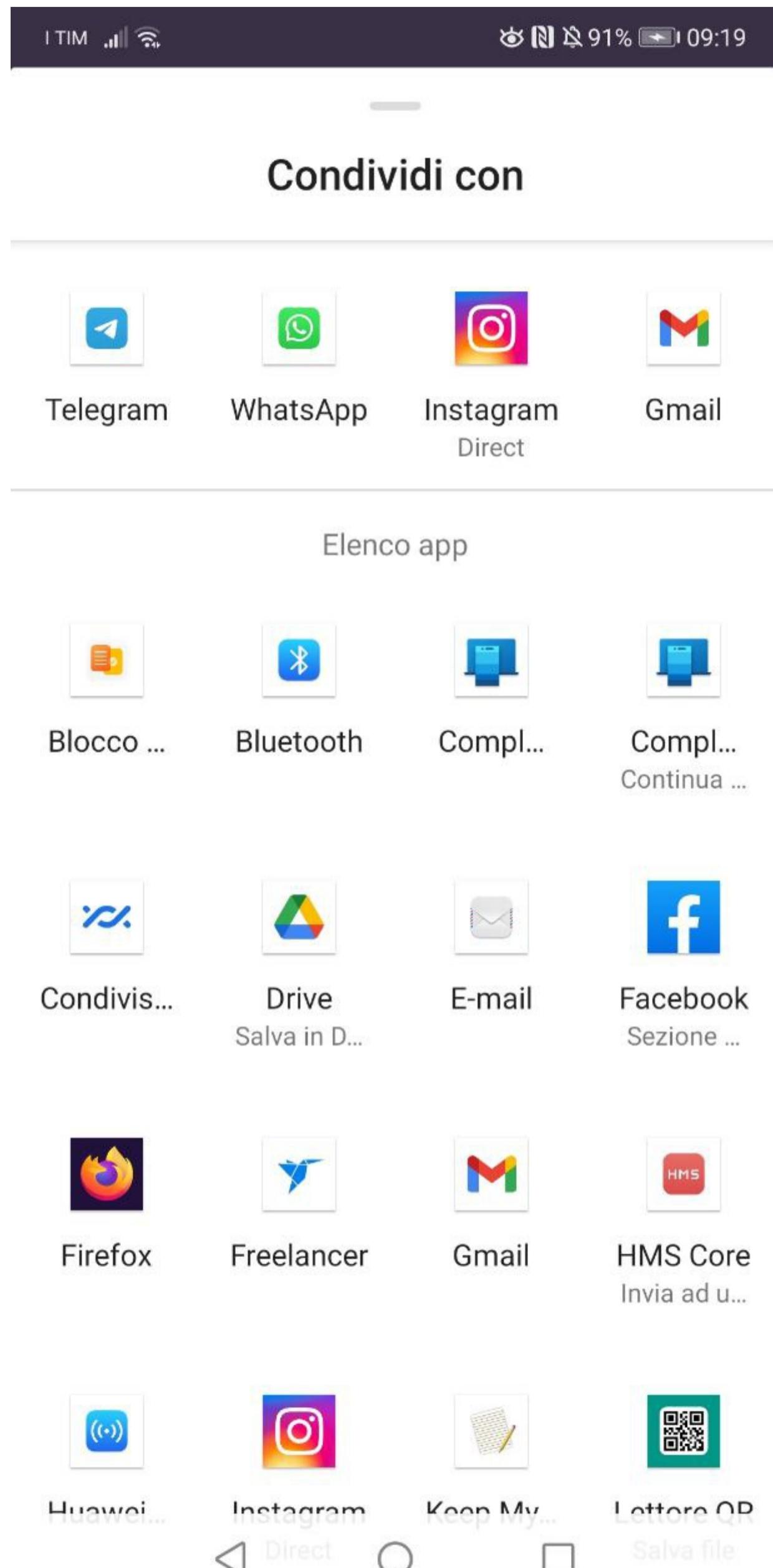
1. RECENSIONE

Premendo sulla recensione desiderata, viene aperto il Widget dedicato:



1.1 PULSANTE LISTA DESIDERI

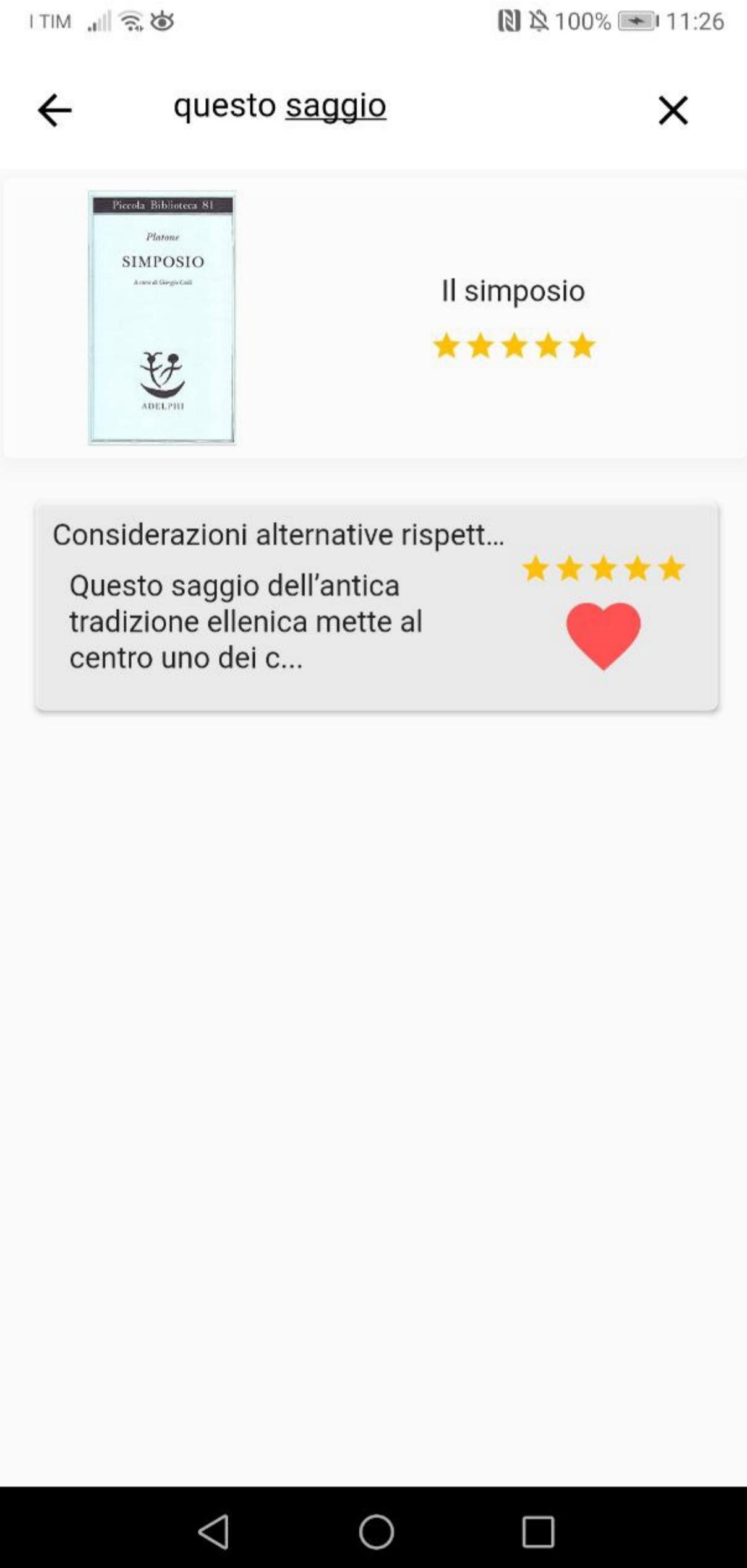
1.2 PULSANTE CONDIVIDI RECENSIONE



2. PULSANTE LISTA DESIDERI

3. ICONA PER FILTRARE PER TESTO

Consente di filtrare le recensioni che contengono i termini ricercati:



2.4 - SVILUPPO

SCELTE PROGETTUALI

DESIGN

Abbiamo utilizzato come tema primario una specializzazione di `ThemeData.light()` per ciò che concerne il light theme, e di `ThemeData.dark()` per quello dark.

```
//light  
final ThemeData base = ThemeData.light();  
  
//dark  
final ThemeData base = ThemeData.dark();
```

LINGUA E STRINGHE

L'app è disponibile solo in **italiano**. Nel file `assets/strings.json` abbiamo messo le varie stringhe, accessibili ovunque tramite il metodo

```
Strings.get
```

RICERCA RECENSIONI

Come descritto nella sezione Android, Amazon non fornisce delle API gratuite utili ai nostri fini. Abbiamo così deciso di usare un certo **User-agent** in ogni richiesta http, e di estrapolare i contenuti **direttamente dal dom della pagina**, usando a tal fine il package [html](#).

Ogni volta l'utente comincia una nuova ricerca, discriminiamo se viene fornita il codice a barre oppure una stringa. Nel primo caso, la classe **Amazon** si appoggia al sito web [CamelCamelCamel](#) per ricavare dal codice il corrispondente **Asin**, ossia il codice identificativo dei prodotti su Amazon.

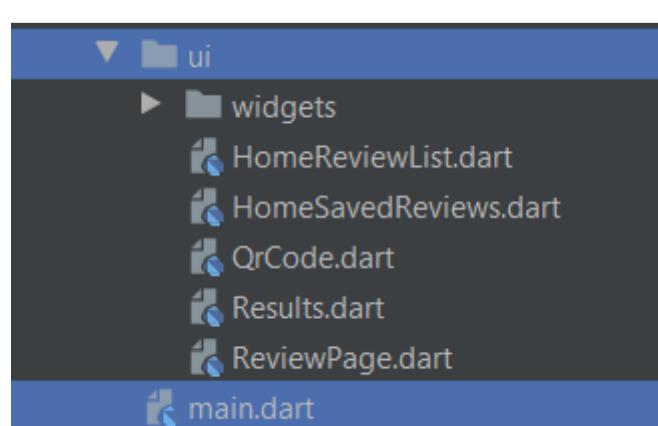
Questo lo si può fare in camelcamel attraverso la richiesta http get all'endpoint `/search?sq={codice}`, che porta o alla pagina `/product/{asin}` o a una che mostra più risultati. In quest'ultimo caso l'app seleziona l'asin del primo di questi.

Ottenuto l'asin, ogni prodotto si trova all'endpoint amazon `/dp/{asin}`, mentre le relative recensioni in `/product-review/{asin}`

Nel caso in cui, invece, la ricerca avviene tramite testo, l'app prende il primo risultato che trova all'url

<https://amazon.it/s?k={ricerca}>

DESCRIZIONE COMPONENTI



Home

In `main.dart` troviamo la classe `Home`, che contiene due Widgets all'interno della componente `NestedScrollView`:

- **HomeSavedReviews**

Restituisce una `ListView` con all'interno tutti i prodotti di cui l'utente ha salvato almeno una recensione.

Premendo sul prodotto si naviga verso `HomeReviewList`.

- **Results**

Consiste principalmente di una `ListView` dove al suo interno vengono iniettate le varie recensioni.

Al click sulla recensione, si naviga verso `ReviewPage`.

Dall'AppBar è possibile cercare testualmente il prodotto o aprire il Widget `QrCode`.

Per gestire quando l'utente torna nella schermata precedente, visto non abbiamo un `NavController` come in Android, abbiamo deciso di intercettare l'evento tramite il widget `WillPopScope`. In questo modo l'utente, da dentro il widget `Results` può tornare a visualizzare `HomeSavedReviews`.

QrCode

Il widget si appoggia al package `qr_code_scanner` per comunicare con la fotocamera posteriore e rilevare eventuali qr codes. Non appena viene letto un qr code, questo viene comunicato alla vista che lo ha chiamato.

HomeReviewList

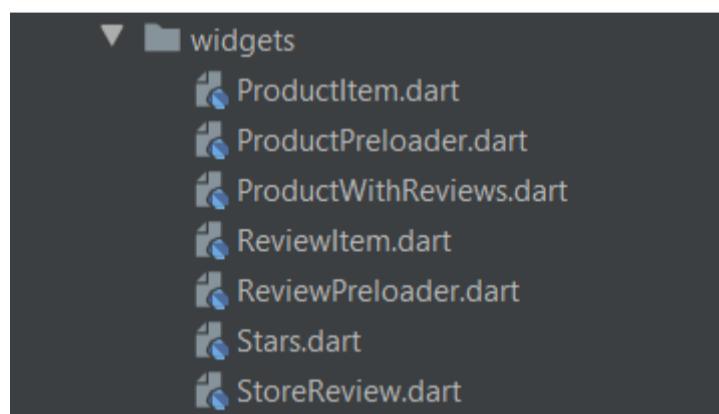
Mostra le recensioni associate a uno specifico prodotto.

ReviewPage

Mostra l'intero contenuto della recensione selezionata. La vista consente di salvare, cancellare, o condividere la stessa.

Per poter condividere la recensione abbiamo usato il package `share`.

WIDGETS



- **ProductItem**

Restituisce una `Card` con le info sul prodotto.

- **ProductPreloader**

Restituisce `ProductItem` con l'effetto shimmer. Abbiamo utilizzato, per quest'ultimo, il package `shimmer`

- **ProductWithReviews**

Restituisce un'istanza di `ProductItem`. Gestisce la navigazione verso `HomeReviewList`

- **ReviewItem**

Restituisce una `Card` con le info sulla recensione. Premendo su questo oggetto si naviga sempre verso `ReviewPage`

- **ReviewPreloader**

Restituisce `ReviewItem` con l'effetto shimmer.

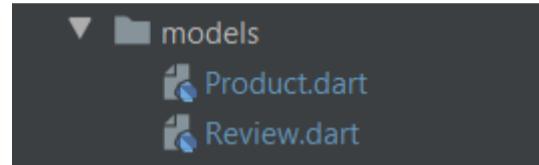
- **Stars**

Costruisce le stelle che corrispondono al voto della recensione. Abbiamo utilizzato il package `flutter_rating_bar`

- **StoreReview**

Costruisce il pulsante per salvare o cancellare dai preferiti la recensione.

MODELS



- **Product**

Describe le proprietà tipiche dell'entità e contiene i metodi necessari per interfacciarsi con il database locale.

Per riuscire a ordinare in base alla ricerca più recente, abbiamo introdotto la proprietà `date`, che funge da timestamp.

Review

Describe le proprietà tipiche dell'entità e contiene i metodi necessari per interfacciarsi con il database locale.

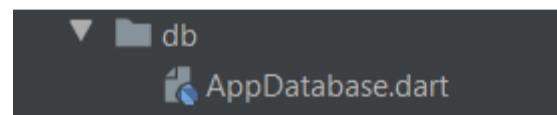
Sovrascrive l'operatore `==`, in tal modo è possibile confrontare due istanze diverse di **Review** semplicemente confrontando la proprietà `id`.

```
bool operator ==(other) => other is Review && (other.id == id);
```

Questa scelta è stata adottata perché spesso è necessario, nell'app, confrontare due istanze diverse di **Review**.

Essendo **Review** identificata univocamente dal suo codice e visto, in Flutter, cerchiamo recensioni solo su Amazon, confrontiamo il campo `id` per restituire `true` o `false` durante la comparazione.

DATABASE

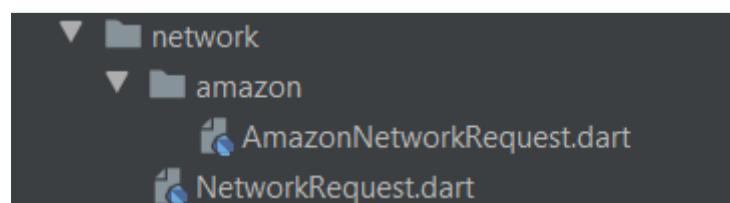


La classe **AppDatabase** istanzia il database in modalità **singleton**. Ogni qualvolta si richiede una lettura/scrittura su db, allora viene chiamato il metodo `database`:

```
if (_database != null) return _database!;  
_database = await _initDB('review.db');
```

Contiene tutti i metodi necessari per ottenere istanze di **Product** e **Review**.

NETWORK



• NetworkRequest

`NetworkRequest.get` gestisce un oggetto `http.Client()` e la risposta del server web.

Per riuscire a tentare nuovamente la chiamata in caso di fallimento, abbiamo racchiuso `http.Client()` all'interno di `RetryClient`:

```
final client = RetryClient(UserAgentClient(http.Client()));
```

e, infine, abbiamo racchiuso `client` all'interno di `try-catch`, in caso di perdita di connessione da parte dell'utente.

Come **User-agent** abbiamo utilizzato il seguente:

Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0

• AmazonNetworkRequest

Gestisce la ricerca del prodotto su Amazon, tramite il metodo `getProd`, e l'estrapolazione delle recensioni attraverso il metodo `getReviews`.

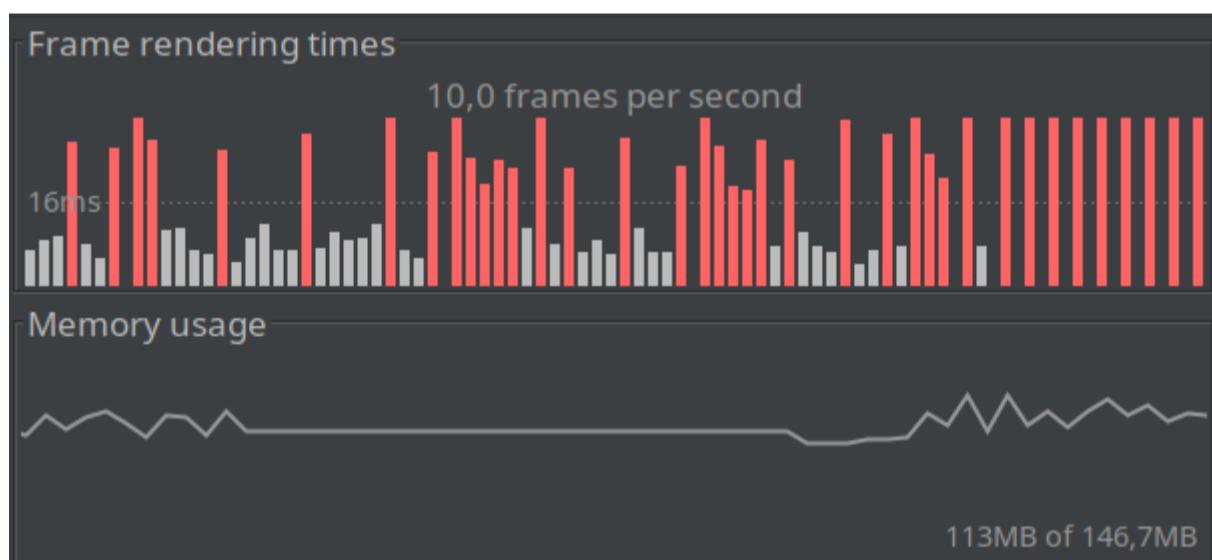
• `getReviews`:

restituisce uno `Stream<Review>`. Questa scelta è stata adottata per riuscire, in background, a ottenere **tutte** le recensioni da Amazon e visualizzarle progressivamente su una `ListView`, senza però degradare troppo le prestazioni dell'app.

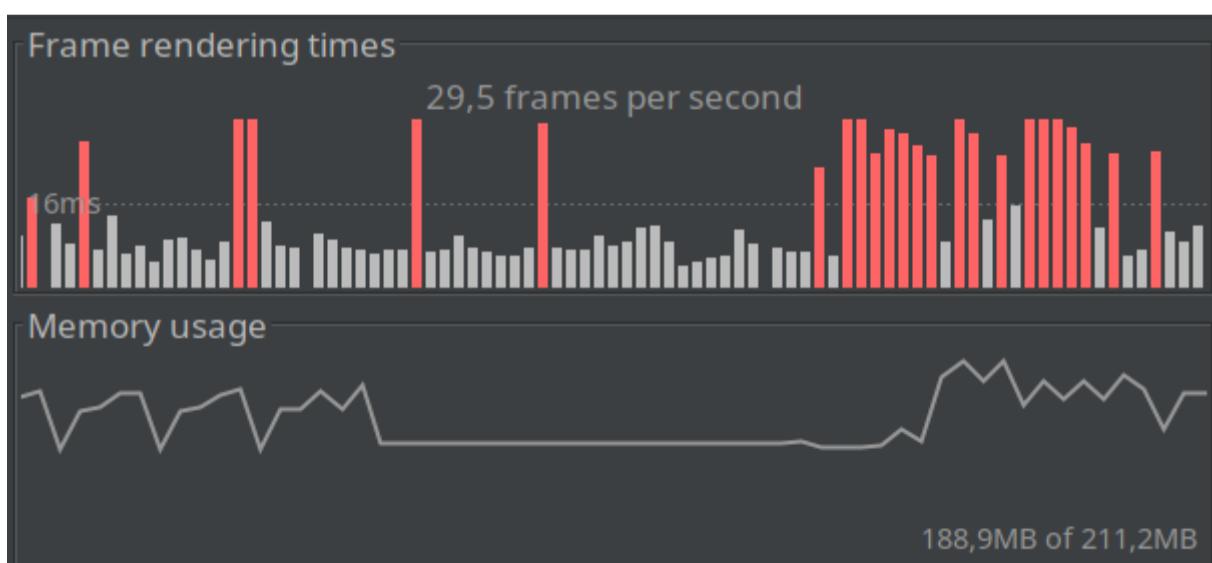
Abbiamo usato, sempre in `getReviews`, un approccio basato sul metodo `compute`, che ci ha consentito di migliorare le performance.

Il seguente test è stato eseguito durante lo scorrimento della lista delle Recensioni:

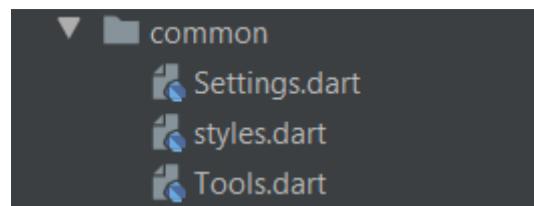
senza compute:



con compute:



DIRECTORY COMMON



- **Tools**

Contiene metodi statici utili a più componenti.

Nella funzione `getImage` abbiamo usato il package `extended_image`.

- **Settings**

La classe contiene proprietà generiche dell'app, come l'url di Amazon.

- **styles**

Il file contiene i `ThemeData`, `lightTheme` e `darkTheme`.

"BUG" APP

Come descritto all'inizio di questa sezione, dal momento l'app si appoggia all'estrapolazione del dom di contenuti HTML, si potrebbero verificare problemi quando **Amazon**, in alcuni url, non rispecchia la struttura di base che abbiamo analizzato per estrarre informazioni. **Amazon**, infatti, cambia dinamicamente la struttura HTML della pagina, di conseguenza potrebbero verificarsi, seppur molto raramente, dei possibili malfunzionamenti. Inoltre, **CamelCamelCamel** potrebbe temporaneamente rifiutare la connessione e generare possibili risultati nulli durante una ricerca tramite **codice a barre**. **Tuttavia**, non abbiamo **mai** sperimentato questo tipo di problema.