

Report esame Corso LPSMT

Studenti: Federico Caporale (218813), Michele Coco (218306)

1. Introduzione

PawsPharma cerca di unire il mondo dei medici veterinari con i possessori di animali domestici, da allevamento, e non solo. I professionisti del settore medico veterinario si trovano spesso a dover consultare le direttive del Ministero della Salute riguardanti specifici farmaci. L'applicazione permetterà di accedere ai dati contenuti nella Banca dati dei Medicinali Veterinari pubblicata dal Ministero della Salute. Sarà possibile ricercare i farmaci per nome, principio attivo, azienda produttrice e codice a barre. Sarà inoltre possibile filtrare i risultati secondo vari criteri come la specie dell'animale, la sua età e la modalità di somministrazione del farmaco. Sarà possibile utilizzare l'applicazione in luoghi non coperti da una connessione internet, come ad esempio allevamenti in zone remote o pascoli. Quando il dispositivo si riconnetterà ad internet, l'app sincronizzerà i dati fra tutti i dispositivi collegati con lo stesso account. Ci saranno due tipologie di account: account base e veterinario. Questi ultimi potranno inserire i dettagli della loro attività, come la loro posizione, i loro contatti e gli orari lavorativi. All'interno dell'app sarà possibile cercare e contattare veterinari nelle vicinanze.

Sarà possibile salvare e condividere profilassi all'interno dell'app. Conterranno indicazioni come ad esempio gli animali che ne sono soggetti e quali farmaci somministrare. Sarà inoltre possibile attivare notifiche per ricordare la somministrazione dei farmaci.

Ogni utente potrà creare un "armadietto" per tenere traccia dei farmaci di cui è in possesso, delle loro date di scadenza, ed essere notificato quando i farmaci sono prossimi alla scadenza.

2. Identificazione del segmento utente

Il nostro target di utenza sarà composto dagli operatori del settore medico veterinario e da tutti i possessori di animali, in particolare gli allevatori. Queste categorie devono frequentemente somministrare farmaci, anche in zone non coperte da connessione internet.

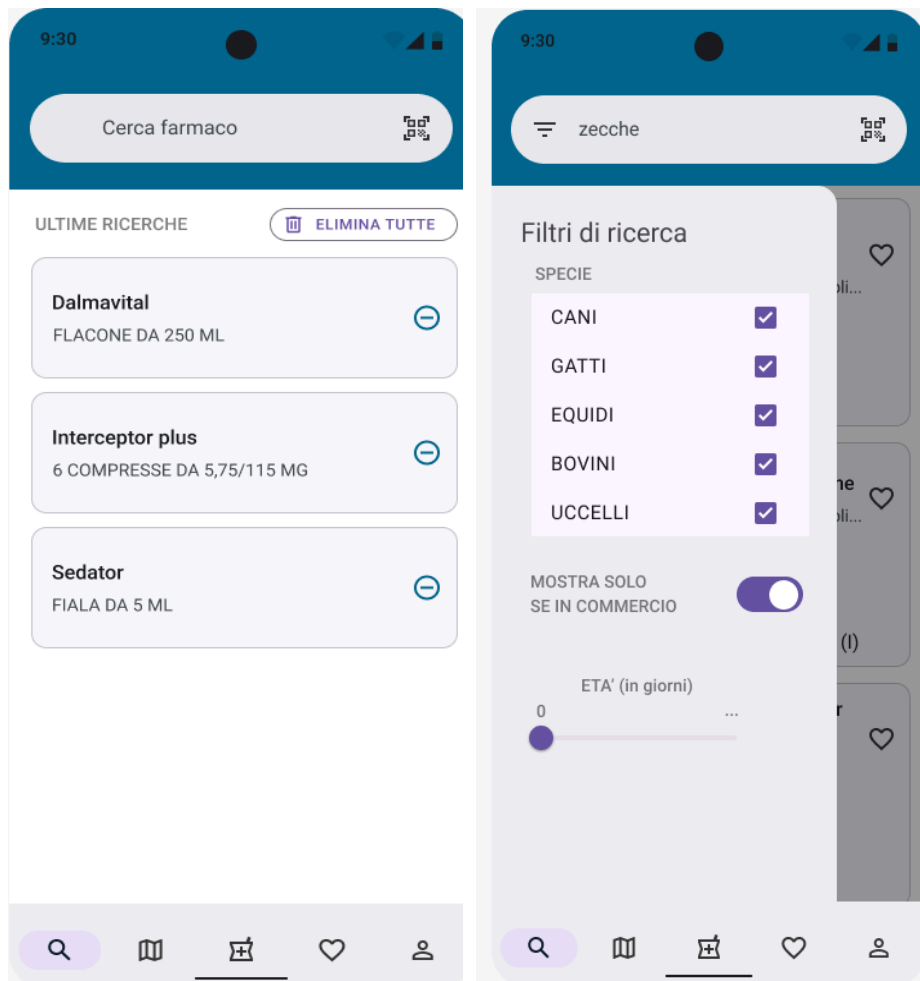
3. Stato dell'arte

Nello store sono presenti tre applicazioni che tentano di fornire delle funzionalità simili: "Animals&Veterinary Drugs", "VetEasy", e "VetDrugs". Hanno tutte un'interfaccia molto basilare, poco curata e di difficile utilizzo. La prima, disponibile solo per IOS a 0.99\$, non viene aggiornata dal 2017, permette solo di ricercare manualmente farmaci e ha pessime recensioni. "VetEasy" consente la ricerca di farmaci soltanto per cani e gatti, non ha nessuna valutazione e non viene aggiornata da più di 6 mesi. L'ultima applicazione simile è "VetDrugs". L'unica feature offerta è la ricerca manuale di farmaci veterinari, non ha alcuna recensione e non viene aggiornata da più di 6 mesi.

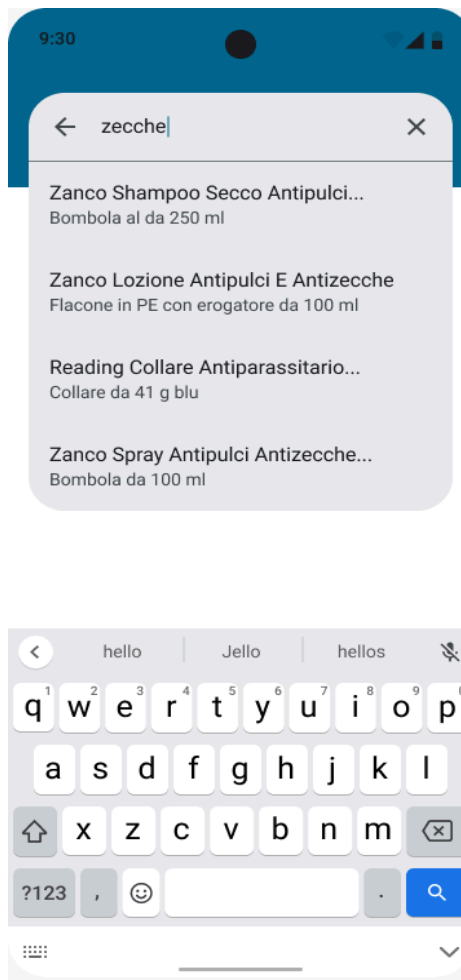
PawsPharma offrirà tutte le funzionalità offerte dalle altre applicazioni tramite un'interfaccia migliore e servizi extra.

4. Wireframe e Navigazione

In questa sezione descriviamo le schermate principali di PawsPharma e la navigazione fra di esse. Tutte hanno in comune una barra in basso che permette di accedere facilmente alle cinque funzionalità primarie, ossia: ricerca dei farmaci, visualizzazione dei veterinari nelle vicinanze, gestione dell'armadietto farmaci, lista dei farmaci preferiti, e impostazioni account.



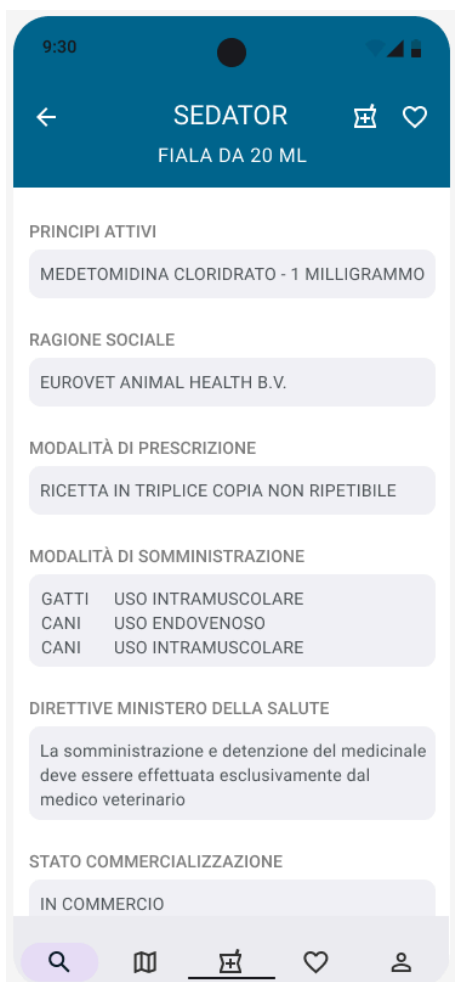
Questa è la schermata principale che viene mostrata appena l'utente apre l'applicazione. Permette di **ricercare farmaci manualmente** o tramite **scansione**, inoltre si potranno applicare dei **filtri** per facilitare la ricerca di essi.



La schemata di destra mostra i risultati della ricerca effettuata a sinistra.

Questa pagina viene aperta quando si seleziona il risultato di una ricerca. Dato che i farmaci possono essere venduti in formati diversi, da questa pagina è possibile selezionare il formato per visualizzarne i dettagli.





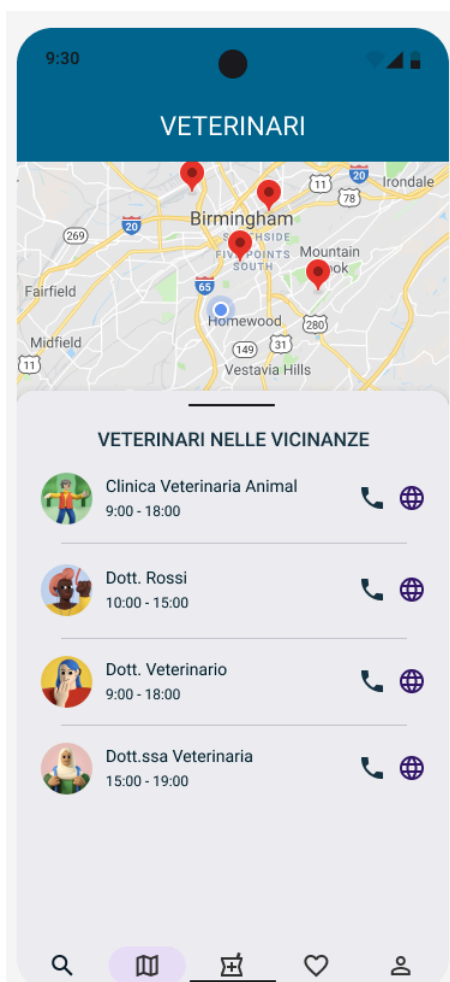
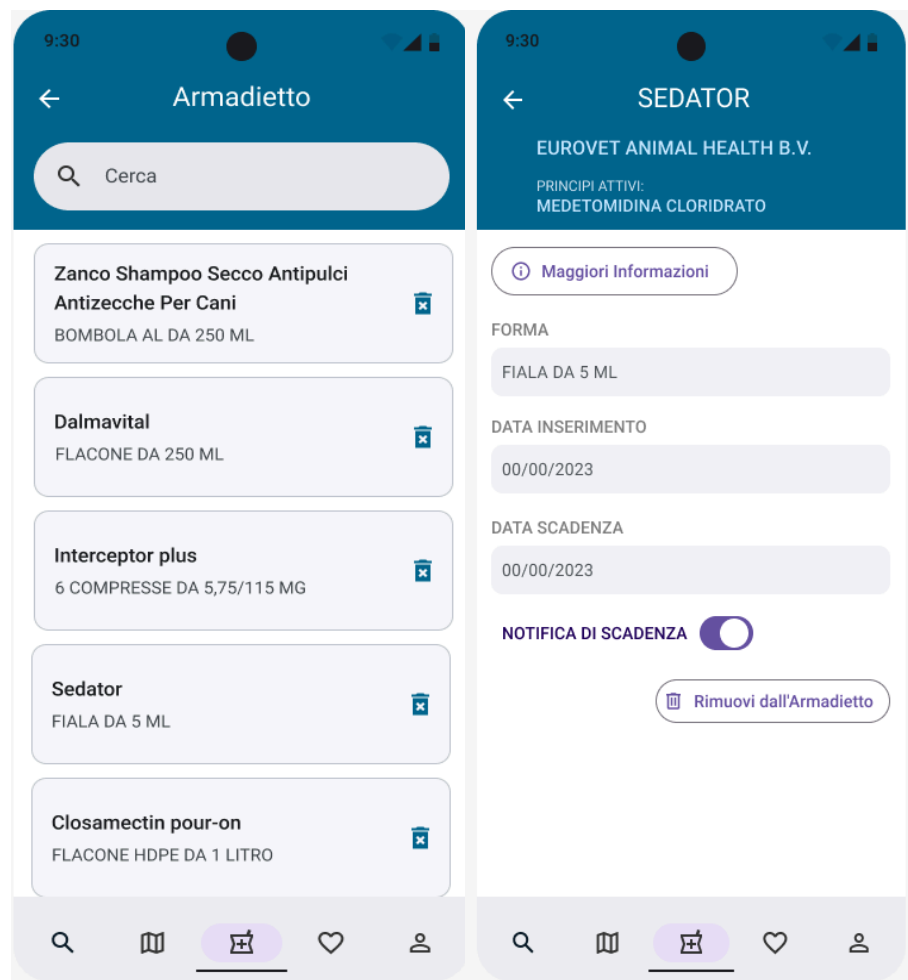
Una volta selezionato un formato di farmaco, viene aperta questa pagina, la quale mostra i **dettagli**.

Questa schemata mostra i farmaci segnati come **preferiti** dall'utente. Ogni farmaco può essere inserito in questa lista selezionando l'icona del cuore, presente in ogni pagina riguardante un farmaco.

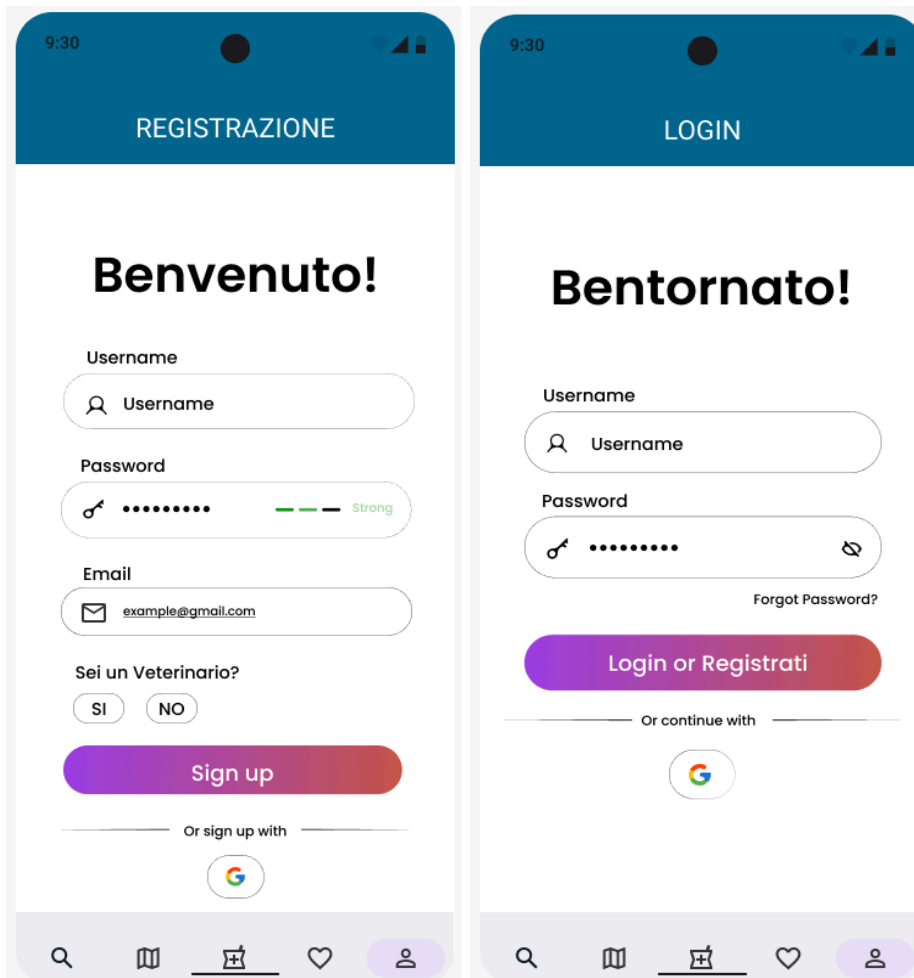


L'**armadietto** consente di tenere traccia dei farmaci che si possiedono e ricevere notifiche alla loro scadenza.

La schermata specifica del farmaco (destra) viene aperta tramite la selezione di uno di essi.



Questa schermata permette di **cercare veterinari** nelle vicinanze e visualizzare i loro contatti e orari di apertura.



È possibile effettuare la **registrazione** e successivamente il **login** di un account. Questi passaggi sono opzionali e gli utenti che decideranno di farli, potranno sincronizzare i dati dell'applicazione fra tutti i loro dispositivi.

5. Architettura

PawsPharma è basata su un'architettura a tre livelli, è organizzata in tre componenti principali o livelli distinti. Abbiamo scelto questo approccio in quanto promuove la modularità, la manutenibilità e la scalabilità del software, consentendoci di lavorare in modo più efficiente e facilitandoci l'evoluzione dell'applicazione nel tempo. Ecco una panoramica di ciascun livello:

1. **Presentation Layer:** Questo è il livello con cui l'utente interagisce direttamente. Si occupa dell'interfaccia utente (UI) dell'applicazione. In alcuni casi include anche la logica di presentazione, evitano la logica di business. Risponde agli input dell'utente e li passa al livello successivo per l'elaborazione.
1. **Logic Layer:** Questo layer contiene la logica di business e si occupa dell'elaborazione dei dati. Accetta richieste provenienti dal livello di presentazione, le elabora e genera risposte adeguate.
2. **Data Layer:** Questo livello si occupa dello storage e della gestione dei dati. Includere database locali e remoti. Fornisce un'interfaccia per accedere e manipolare i dati

necessari alla logica di business. È responsabile della memorizzazione e del recupero dei dati in modo efficiente e sicuro.

I farmaci mostrati all'interno dell'app sono estratti da un .csv fornito dal Ministero Della Salute. Questo file è contenuto all'interno dell'app. Al primo avvio, l'app effettua un parsing del .csv e inserisce i dati in un database SQLite locale. Abbiamo fatto questa scelta per ridurre i necessari per la ricerca all'interno del dataset.

Il sistema di registrazione e autenticazione si basa su Firebase Authenticator. Gli utenti possono registrarsi tramite un'email e password.

I dati relativi agli utenti vengono salvati su Firebase Cloud Firestore. Il database contiene un'unica collection all'interno della quale sono contenuti un documento per ogni utente.

PawsPharma utilizza le API di OpenStreetMap per mostrare una mappa all'interno dell'app e per ricavare la latitudine e longitudine da un indirizzo.

6. Implementazione

In questa sezione descriviamo l'implementazione dell'applicazione attraverso l'uso di screenshot effettuate all'interno di essa.

1. Pagina di ricerca

In questa schermata (fig. 1) è possibile osservare come vengano visualizzate le Card per i singoli farmaci con una semplice descrizione contenente Nome, Formato e Principi attivi di esso; inoltre la Card presenta un'icona per l'aggiunta ai preferiti. Tutto questo è stato realizzato tramite la classe DrugCard con codice visibile nella fig. 2. Questa classe richiede l'oggetto Drug che contiene tutte le informazioni riferite ad uno specifico farmaco (fig. 3) e si appoggia sulla classe Card fornita dalla libreria standard di Dart.

La classe DrugCard è una StatefulWidget che accetta diversi parametri:

- drug: Il farmaco da visualizzare nella card.
- isFavoriteInitial: Un flag per inizializzare lo stato preferito.
- onFavorite: Una callback per gestire l'azione di aggiunta/rimozione dai preferiti.
- setStateOnFavorite: Un flag per decidere se eseguire setState quando l'azione preferita viene eseguita.

Nella classe _DrugCardState, il metodo initState inizializza la variabile isFavorite con il valore iniziale specificato.

Nel metodo build, viene restituita una "Card" contenente un "InkWell", che fornisce feedback visivo quando l'utente tocca la scheda.

La "DrugCard" contiene un "ListTile" che mostra il nome e la descrizione del farmaco, nonché un'icona del cuore (IconButton) che rappresenta lo stato preferito. L'azione del cuore viene gestita da onFavorite, e se setStateOnFavorite è vero, viene eseguito setState per aggiornare l'aspetto dell'icona del cuore.

Per concludere, nella parte inferiore della scheda, vengono visualizzati i principi attivi del farmaco. Inoltre è stata implementata anche la ricerca con i filtri.

	<pre>class DrugCard extends StatefulWidget { final Drug drug; final bool isFavoriteInitial; final VoidCallback onFavorite; final bool setStateOnFavorite; @override Widget build(BuildContext context) { return Card(child: InkWell(onTap: () { Navigator.of(context) .push(MaterialPageRoute(builder: (context) => DrugDetailsPage(drug: widget.drug, isFavoriteInitial: isFavorite,), // MaterialPageRoute),); },),); } }</pre>	<pre>class Drug { final String name; final int aicCode; final List<int> gtinCodes; final String description; final String activePrinciple; final String species; final String atcVet; final String manufacturer; final String prescriptionMode; final DateTime? soldSince; final DateTime? soldUntil; final String? extraInfo;</pre>
fig. 1	fig. 2	fig. 3

2. Specifiche di ogni farmaco

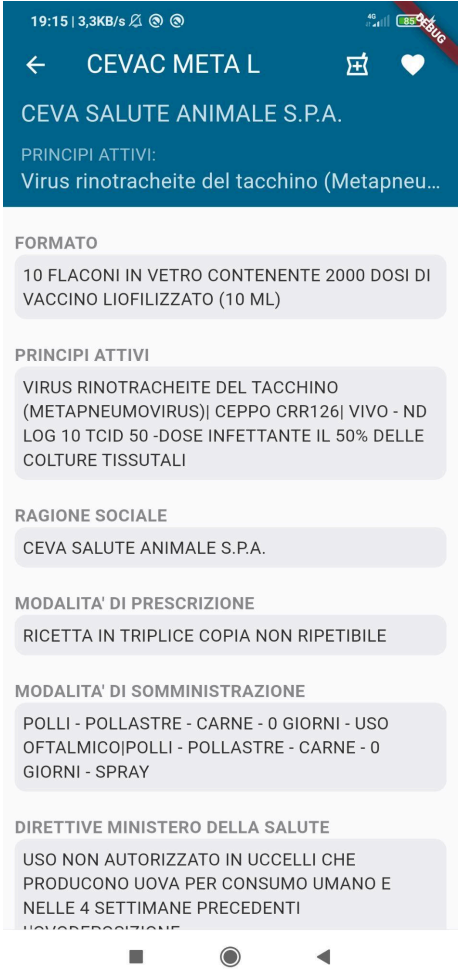
Quando l'utente tocca la "DrugCard", viene aperta una nuova pagina dei dettagli (fig. 4) del farmaco utilizzando `Navigator.of(context).push`. La pagina dei dettagli viene creata utilizzando `DrugDetailsPage` e passa il farmaco e il valore `isFavorite` corrente.

La classe `DrugDetailsPage` (fig. 5) è una `StatefulWidget` che rappresenta la pagina dei dettagli del farmaco. Ha tre parametri obbligatori: `repository`, `drug`, e `isFavoriteInitial`. Il costruttore inizializza la classe e imposta il repository su `FirestoreRepository`.

La classe `_DrugDetailsPageState` è la classe di stato associata a `DrugDetailsPage`. Il metodo `initState` viene utilizzato per inizializzare lo stato, in particolare il valore di `isFavorite` basato su `isFavoriteInitial`.

Il metodo `build` è il principale per la creazione dell'interfaccia utente della pagina dei dettagli del farmaco. Utilizza un `WillPopScope` per gestire il comportamento quando l'utente preme il pulsante di "back". La pagina è suddivisa in una barra superiore (`AppBar`) con il nome del farmaco e azioni come l'aggiunta ai preferiti e l'aggiunta a un armadietto virtuale, e un corpo con ulteriori dettagli del farmaco.

Le funzioni `openAddToCabinetDialogue` e `submitDialogue` vengono utilizzate per mostrare una finestra di dialogo che consente agli utenti di selezionare un armadietto virtuale esistente in cui aggiungere il farmaco e per gestire la conferma della selezione.

	<pre> class DrugDetailsPage extends StatefulWidget { final IRepository repository; final Drug drug; final bool isFavoriteInitial; DrugDetailsPage({super.key, required this.drug, required this.isFavoriteInitial}) : repository = FirestoreRepository(); @override State<DrugDetailsPage> createState() => _DrugDetailsPageState(); } class _DrugDetailsPageState extends State<DrugDetailsPage> { late bool isFavorite; @override void initState() { super.initState(); isFavorite = widget.isFavoriteInitial; } </pre>
fig. 4	fig. 5

3. Armadietto farmaci

Nella pagina degli armadietti troviamo la lista di tutti gli armadietti disponibili per l'utente, con il nome, il numero di farmaci al suo interno e un tasto per eliminarlo (fig. 6).

Il tutto è gestito dalla classe `CabinetPage` (fig. 7) è una `StatefulWidget` che accetta due parametri:

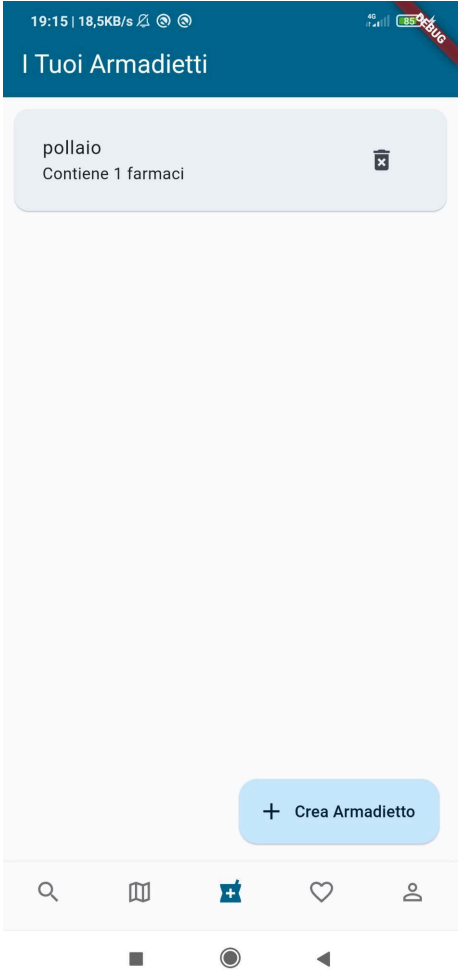
- `repository`: Il repository utilizzato per gestire i dati.
- `cabinet`: L'armadietto di cui visualizzare il contenuto.

Il costruttore inizializza la classe e imposta il repository su `FirestoreRepository`.

La pagina mostra l'elenco dei farmaci contenuti nell'armadietto. Se l'armadietto è vuoto, viene visualizzato un messaggio informativo.

La lista dei farmaci viene creata utilizzando `ListView.builder`. Ogni farmaco è rappresentato da un `CabinetEntryCard` che mostra le informazioni dell'elemento di armadietto, inclusi il nome del farmaco e altre informazioni.

Per ciascun elemento di armadietto, viene fornita la possibilità di rimuoverlo dall'armadietto. Questa azione viene gestita tramite la funzione `openConfirmDeletionDialogue`, che mostra una finestra di dialogo di conferma per l'eliminazione. Quando l'utente conferma l'eliminazione, l'elemento viene rimosso dall'armadietto e l'armadietto viene aggiornato nel repository.

	<pre>class CabinetPage extends StatefulWidget { final IRepository repository; final Cabinet cabinet; CabinetPage({super.key, required this.cabinet}) : repository = FirestoreRepository(); @override State<CabinetPage> createState() => _CabinetPageState(); }</pre>
fig. 6	fig. 7

La classe `CabinetEntryPage` (fig. 8) è una `StatefulWidget` che accetta diversi parametri:

- `repository`: Il repository utilizzato per gestire i dati.
- `cabinet`: L'armadietto associato all'elemento.
- `cabinetEntry`: L'elemento di armadietto di cui visualizzare i dettagli.

Il costruttore inizializza la classe e imposta il repository su `FirestoreRepository`.

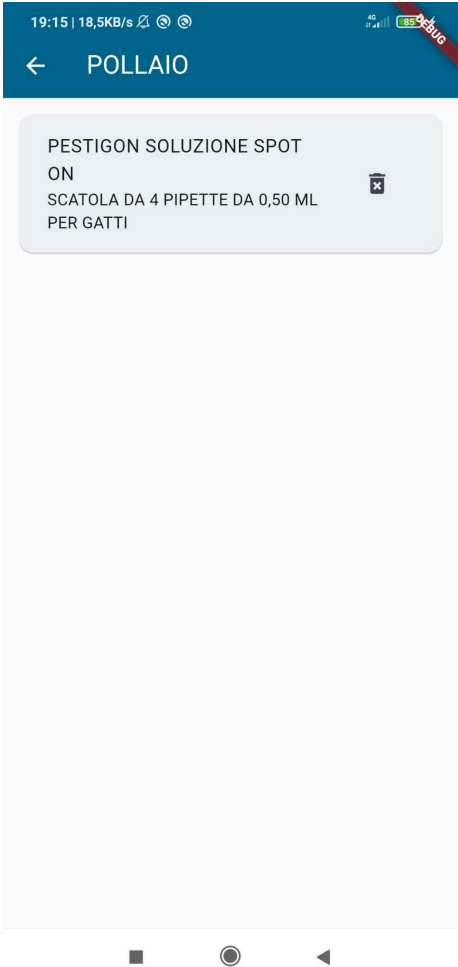
La pagina mostra i dettagli dell'elemento di armadietto, inclusi il nome del farmaco, il produttore, i principi attivi, la data di inserimento e la data di scadenza (se presente).

L'icona "Maggiori Informazioni" consente agli utenti di accedere ai dettagli completi del farmaco tramite `DrugDetailsPage`. Viene controllato se il farmaco è contrassegnato come preferito utilizzando `widget.repository.isFavorite`, e quindi viene aperta la pagina dei dettagli del farmaco.

Gli utenti possono selezionare una nuova data di scadenza utilizzando il pulsante "Seleziona la data di scadenza". Viene utilizzato `showDatePicker` per consentire all'utente di selezionare una data.

Un'interruttore "NOTIFICA SCADENZA" permette all'utente di attivare o disattivare la notifica di scadenza per l'elemento di armadietto. Il valore viene aggiornato nel repository tramite `widget.repository.cabinetUpdate`.

Un pulsante "Rimuovi dall'Armadietto" consente agli utenti di rimuovere l'elemento di armadietto. Viene visualizzata una finestra di dialogo di conferma per confermare l'eliminazione. L'elemento viene rimosso dall'armadietto tramite `widget.cabinet.removeDrug`, e l'armadietto viene aggiornato nel repository.

 <p>The screenshot shows a mobile application interface. At the top, there's a status bar with the time 19:15, signal strength, and battery level. Below it, a blue header bar contains a back arrow and the text 'POLLAIO'. The main content area features a light blue card with the text 'PESTIGON SOLUZIONE SPOT ON' and 'SCATOLA DA 4 PIPETTE DA 0,50 ML PER GATTI'. To the right of the text is a small trash icon. The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps buttons.</p>	<pre> class CabinetEntryPage extends StatefulWidget { final IRepository repository; final Cabinet cabinet; final CabinetEntry cabinetEntry; CabinetEntryPage({super.key, required this.cabinet, required this.cabinetEntry}) : repository = FirestoreRepository(); @override State<CabinetEntryPage> createState() => _CabinetEntryPageState(); } </pre>
fig. 8	fig. 9

4. Login e Pagina Utente

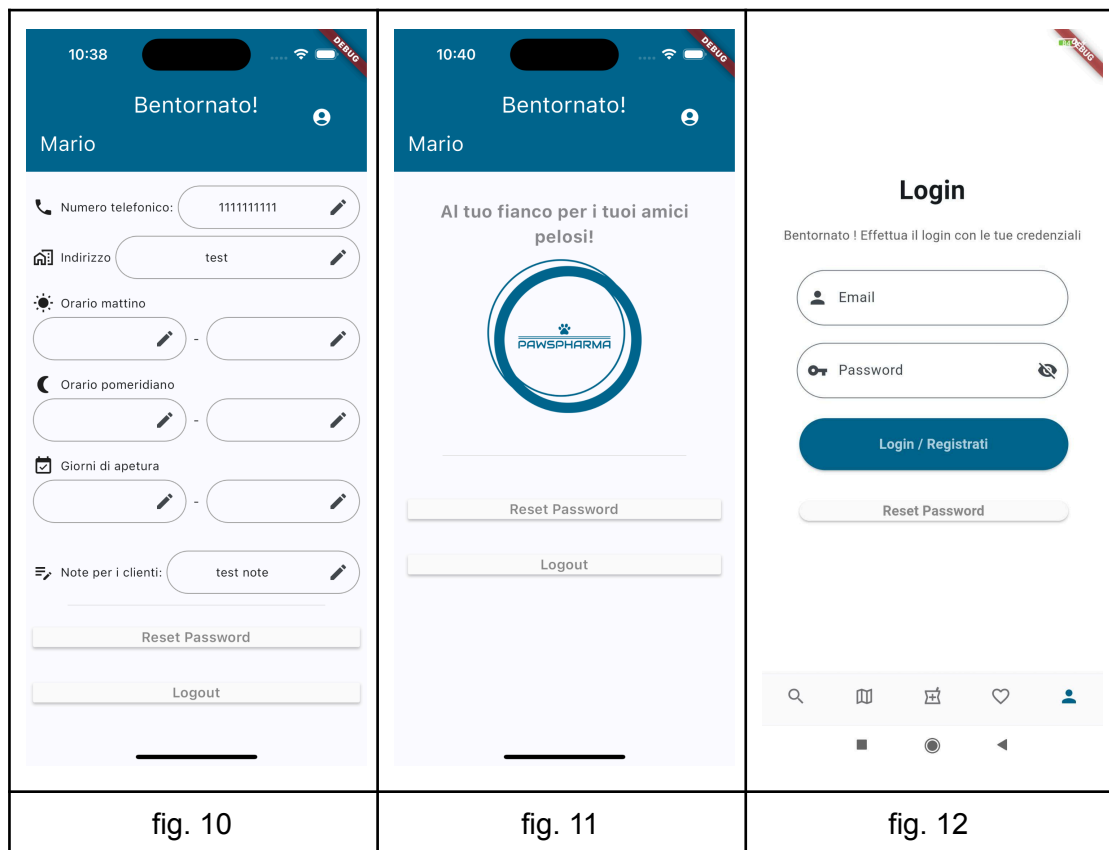
La classe `ProfileLandingPage` è una `StatefulWidget` che accetta un parametro `repository`, che rappresenta il repository utilizzato per gestire i dati. Il costruttore inizializza la classe e imposta il repository su `FirestoreRepository`.

La pagina controlla se l'utente è già autenticato utilizzando `widget.repository.isUserSignedIn()`. Se l'utente è autenticato, la pagina utilizza un `FutureBuilder` per ottenere il tipo di utente (veterinario o utente comune) utilizzando `widget.repository.getUserType()`.

Durante l'attesa del caricamento dei dati, viene visualizzata una schermata di caricamento con un messaggio di testo e un indicatore di progresso.

Se il caricamento dei dati ha successo e viene restituito il tipo di utente, la pagina reindirizza l'utente alla pagina del profilo corrispondente in base al tipo di utente. Ci sono tre possibilità: `ProfileVetPage` (fig. 10) per i veterinari, `ProfileUserPage` (fig. 11)

per gli utenti comuni o, in caso di tipo di utente non riconosciuto viene reindirizzato alla pagina di accesso LoginPage (fig. 12).



La classe ProfileUserPage è una StatefulWidget che accetta un parametro repository, che rappresenta il repository utilizzato per gestire i dati. Il costruttore inizializza la classe e imposta il repository su FirestoreRepository. La pagina ottiene l'indirizzo email dell'utente utilizzando `_userEmail = widget.repository.getUserEmail()`; e mostra l'indirizzo email dell'utente nella parte superiore della pagina.

La pagina contiene un pulsante "Reset Password" che consente agli utenti di richiedere il reset della password. Quando l'utente fa clic su questo pulsante, viene chiamata la funzione `_performReset`, che invoca il reset della password attraverso `widget.repository.resetPassword()`. Successivamente, viene mostrato un messaggio di conferma o un messaggio di errore in base all'esito dell'operazione.

Se l'utente è autenticato e l'indirizzo email è stato recuperato con successo, la pagina mostra il profilo utente. Se l'indirizzo email non è stato recuperato o se si verifica un errore durante il recupero dei dati, vengono mostrati messaggi di errore appropriati.

La classe ProfileVetPage è una StatefulWidget che accetta un parametro repository, che rappresenta il repository utilizzato per gestire i dati. Il costruttore inizializza la classe e imposta il repository su FirestoreRepository.

La pagina ottiene i dati del veterinario utilizzando `_veterinarian = widget.repository.getVeterinarian()`; e mostra le informazioni del veterinario, inclusa l'email, se disponibili.

La pagina contiene un pulsante "Reset Password" che consente ai veterinari di richiedere il reset della password. Quando il veterinario fa clic su questo pulsante, viene chiamata la funzione `_performReset`, che invoca il reset della password attraverso `widget.repository.resetPassword()`. Successivamente, viene mostrato un messaggio di conferma o un messaggio di errore in base all'esito dell'operazione.

La pagina consente ai veterinari di modificare diversi campi, come il numero di telefono, l'indirizzo, gli orari di apertura, i giorni di apertura e le note per i clienti. La classe `EditableTextField` è utilizzata per consentire l'editing dei campi.

Se i dati del veterinario sono disponibili e sono stati recuperati con successo, la pagina mostra il profilo del veterinario, consentendo l'editing dei campi. Se i dati non sono stati recuperati o se si verifica un errore durante il recupero dei dati, vengono mostrati messaggi di errore appropriati.

All'interno delle due schermate utenti troviamo anche un bottone per eseguire il logout dal proprio profilo.

5. **Registrazione:**

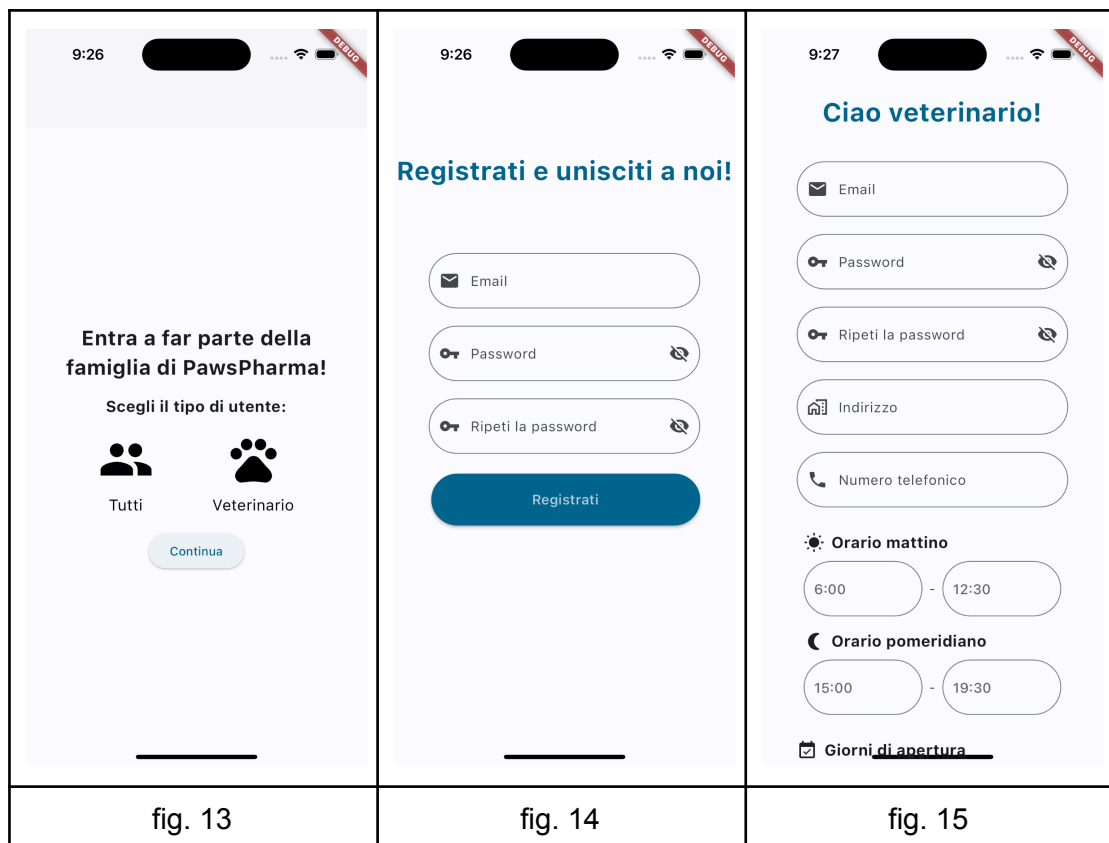
`RegisterLandingPage` (fig. 13) è uno `StatefulWidget` che serve come pagina di accesso per gli utenti che desiderano registrarsi come utenti o veterinari.

La pagina contiene una barra delle app senza un'icona di ritorno automatico.

Gli utenti possono scegliere il tipo di utente tra "Tutti" e "Veterinario". La scelta del tipo di utente viene memorizzata nella variabile `selectedUserType`.

Ci sono due icone (gruppo e animali domestici) e testo corrispondente per ciascun tipo di utente. Toccando una di queste opzioni, la variabile `selectedUserType` viene aggiornata per riflettere la selezione dell'utente. Se un utente tocca "Tutti", la variabile `selectedUserType` sarà impostata su "Tutti", e lo stesso per "Veterinario".

Alla fine della pagina, c'è un pulsante "Continua". Quando viene premuto, l'applicazione decide quale pagina di registrazione visualizzare in base alla selezione dell'utente.



Sia RegisterUserPage che RegisterUserPage sono StatefulWidget che consente agli utenti comuni di registrarsi nell'applicazione.

La pagina contiene tre campi di testo: uno per l'indirizzo email (emailController), uno per la password (passwordController), e uno per la conferma della password (passwordController2).

C'è anche un'icona "mostra/nascondi password" che permette agli utenti di mostrare o nascondere la password mentre la digitano.

La funzione `_doPasswordsMatch()` verifica se le due password inserite corrispondono tra loro. Questa verifica viene utilizzata per mostrare un messaggio di errore se le password non corrispondono.

La funzione `_togglePasswordVisibility()` cambia lo stato della variabile `_isPasswordVisible` per mostrare o nascondere la password quando l'utente tocca l'icona "mostra/nascondi password".

La funzione `_performRegistration()` viene chiamata quando l'utente preme il pulsante "Registrati". Questa funzione ottiene l'indirizzo email e la password dai controller dei campi di testo, quindi chiama il metodo `registerCommonUser` del repository per registrare l'utente. Se la registrazione ha successo, l'app reindirizza l'utente alla pagina ProfileLandingPage. In caso di errore durante la registrazione, viene visualizzato un messaggio di errore appropriato mediante un dialogo.

6. Mappe:

Le mappe (fig. 16) vengono implementate grazie alla libreria Open_Street_Map che offre un'implementazione per la visualizzazione delle mappe utilizzando

OpenStreetMap (OSM) come provider di mappe. OpenStreetMap è una fonte di dati geografici open source che consente di accedere a mappe dettagliate e dati di geolocalizzazione. Il widget che le gestisce è il MapWidget e queste sono le sue principale funzionalità:

Init State: Nel metodo initState, la posizione corrente dell'utente viene ottenuta utilizzando il pacchetto location.

`_getUserCustomMarker()`: Questo metodo ottiene dati dai documenti di una collezione Firebase Firestore chiamata "users". Se un documento contiene un campo "indirizzo", viene creato un oggetto CustomMarker con informazioni come nome, orari di apertura, numero di telefono e posizione (convertita da un indirizzo tramite una richiesta API) e quindi aggiunto all'elenco dei marcatori `_markers`.

`_convertAddressToLatLng()`: Questa funzione prende un indirizzo e lo converte in una posizione geografica LatLng utilizzando una richiesta HTTP a un servizio di geocodifica di terze parti (in questo caso, Nominatim di OpenStreetMap).

`_addMarker()`: Aggiunge un marcatore all'elenco dei marcatori `_markers` con informazioni specifiche come nome, orari di apertura, numero di telefono e posizione.

`_searchMarkers()`: Questo metodo cerca i marcatori all'interno di una distanza massima (impostata come `thresholdDistance`) dalla posizione corrente dell'utente. I marcatori che soddisfano il criterio vengono aggiunti all'elenco `_filteredMarkers` e la mappa viene centrata su di essi.

`_addMarkerOnMap()`: Questa funzione viene chiamata quando un utente tocca la mappa e apre un dialog per aggiungere un nuovo marcatore personalizzato sulla mappa. Le informazioni come nome, orari di apertura e numero di telefono del nuovo marcatore vengono inserite dall'utente.

`_showMarkerInfo()`: Questo metodo mostra un dialog con le informazioni del marcatore quando un utente fa clic su un marcatore sulla mappa.

`_getCurrentLocation()`: Questo metodo ottiene la posizione corrente dell'utente utilizzando il pacchetto location. Verifica se il servizio di localizzazione è abilitato e se l'app ha ottenuto l'autorizzazione. Se tutto è in ordine, viene aggiornata la posizione corrente dell'utente e viene aggiunto un nuovo marcatore "Posizione attuale" sulla mappa.

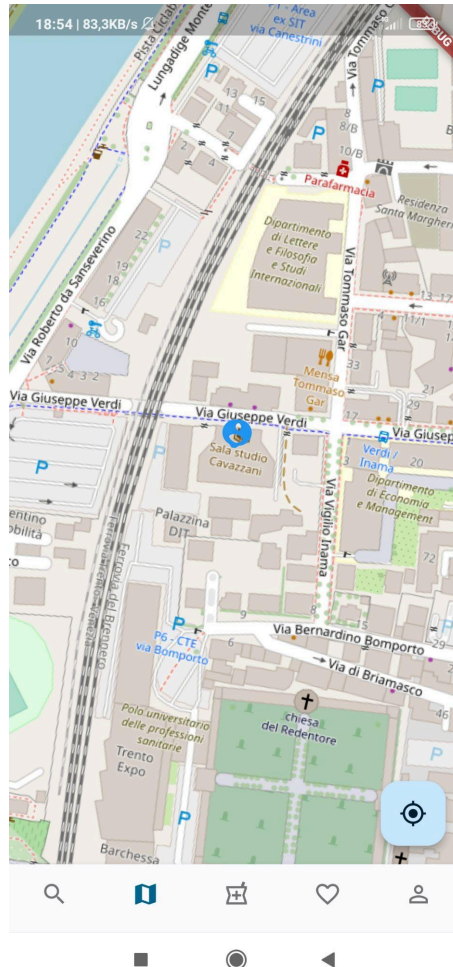


fig. 16

7. Preferiti:

Questa schermata (fig. 17) consente agli utenti di visualizzare i loro farmaci preferiti e di cercare tra di essi. Gli utenti possono anche rimuovere i farmaci dai preferiti direttamente dalla schermata. La classe che si occupa di questo è la Favorites, all'interno di essa troviamo:

- Un'appBar personalizzato con un'icona a forma di cuore che rappresenta i preferiti.
- Un campo di ricerca per cercare i farmaci tra i preferiti.
- Una lista di farmaci preferiti visualizzati sotto il campo di ricerca.

Quando l'app viene avviata, i preferiti vengono caricati utilizzando la funzione `_favorites` dal repository (FirestoreRepository). Questa funzione sembra restituire una lista di oggetti Drug.

Il campo di ricerca (`_searchController`) ha un listener (`queryListener`) che ascolta le modifiche e chiama la funzione `query` ogni volta che il testo di ricerca cambia.

La funzione query filtra i farmaci preferiti in base al testo di ricerca inserito dall'utente. Se il testo di ricerca non è vuoto, verranno mostrati solo i farmaci che contengono il testo di ricerca nel loro nome o descrizione.

La schermata mostra i risultati della query in una ListView.builder. Per ogni farmaco nei risultati, viene creato un widget DrugCard personalizzato.

All'interno del widget DrugCard, l'utente può rimuovere un farmaco dai preferiti facendo clic su un'icona a forma di cuore. Viene quindi chiamata la funzione widget.repository.favoriteRemove per rimuovere il farmaco dai preferiti.

Se la lista dei preferiti diventa vuota, verrà visualizzato un messaggio che indica che non ci sono preferiti.

Durante il caricamento dei dati, viene visualizzato un indicatore di progresso.

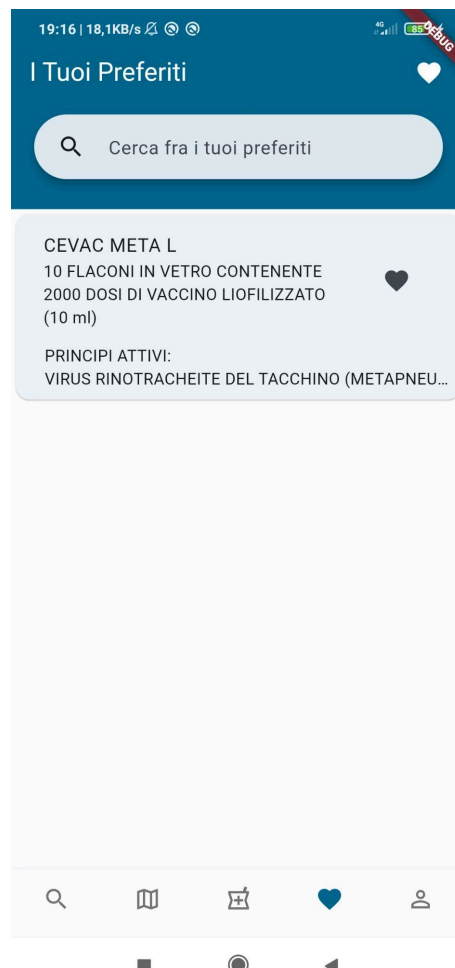


fig. 17

7. Valutazione

“L'applicazione PawsPharma mi piace, mi è molto utile nella cura della mia gatta. E' molto semplice da usare. Mi permette di inserire i farmaci che utilizzo abitualmente in un armadietto virtuale personale oppure semplicemente tra i farmaci preferiti, e, in caso di bisogno, di cercare i veterinari più vicini a me. L'unica cosa che mi infastidisce è il continuo spuntare fuori della barra di loading mentre faccio una ricerca.”

“Questa applicazione è bella e utile per cercare e salvare i farmaci che utilizzo per curare tutti i miei animali creando vari armadietti per ogni tipologia di animale. Mi fa ricordare i farmaci che ho già oppure no. Peccato che non mi vadano le mappe per vedere i veterinari nelle vicinanze e l'applicazione non permette di effettuare una ricerca semplice, quindi se non vedo i veterinari nelle mappe l'applicazione per i veterinari è completamente inutile.”

8. Analisi critica dei limiti dell'applicazione

Criticità da sottolineare nell'uso di PawsPharma:

- **Uso di Google Maps:** Una delle limitazioni principali è l'uso di Google Maps come servizio di mappatura. Sebbene sia ampiamente utilizzato e ricco di funzionalità, potrebbe comportare costi aggiuntivi per l'utente o per lo sviluppatore dell'app. Inoltre, potrebbe non essere la scelta ideale per utenti preoccupati per la privacy.
- **Prestazioni** su dispositivi meno recenti: L'applicazione sembra avere problemi di lag su dispositivi meno recenti. Questo può limitare la base di utenti, poiché non tutti dispongono di dispositivi di fascia alta. Migliorare le prestazioni su una gamma più ampia di dispositivi potrebbe essere cruciale per il successo dell'app.
- **Manca l'opzione di registrazione e login tramite servizi terzi come Google:** L'assenza di opzioni di registrazione e login tramite servizi terzi come Google o Facebook potrebbe rendere meno comoda l'esperienza dell'utente. Queste opzioni consentono agli utenti di accedere rapidamente e senza dover ricordare ulteriori credenziali. La loro assenza potrebbe disincentivare alcuni utenti dall'utilizzare l'app.
- **Assenza di supporto per lingue multiple:** Se l'app è destinata a un pubblico internazionale, l'assenza di supporto per lingue diverse potrebbe essere un ostacolo. Inoltre anche il dataset dei farmaci è fornito dal Ministero della Salute Italiano, quindi andrebbe eseguito un refactor di molte sezioni del codice.