DealDetective

RELAZIONE DI PROGETTO

Gruppo **The Stilton Assistants**
Jessica Theofilia • 894476
Cattaneo Francesco • 900411

Indice

Introduzione

L'obiettivo di questo progetto è creare un'applicazione mobile che centralizzi le offerte provenienti da tre supermercati (Carrefour, Esselunga e Tigros), offrendo agli utenti un'esperienza semplificata per confrontare i prezzi.

Adesso, per poter confrontare le offerte tra vari supermercati, un utente dovrebbe aver installato sul proprio telefono tutte e tre le applicazioni, separatamente, e continuamente cambiare da una all'altra.

Grazie a DealDetective, le si possono direttamente confrontare da un'unica pagina centralizzata. L'utente, inoltre, può aggiungere le offerte che desidera alla propria lista della spesa, o altrimenti filtrarle (per categoria, nome, prezzo, ecc...).

Abbiamo ritenuto fondamentale che l'app funzionasse anche senza una connessione internet attiva. Per questo motivo, non richiede la creazione di un account e conserva in memoria le offerte principali, garantendo l'accesso anche offline.

Il progetto è realizzato in Kotlin con Android Studio, IDE sviluppato da JetBrains.

Funzionalità

L'app offre numerose funzionalità per semplificare la ricerca delle migliori offerte nei supermercati. In tempo reale, permette di consultare promozioni dai tre supermercati menzionati prima, mostrando il prezzo originale, il prezzo scontato, la percentuale di sconto e la categoria del prodotto.

In ogni pagina, la barra di navigazione inferiore permette agli utenti di passare da una sezione dell'app all'altra; la barra superiore, invece, fornisce informazioni e/o funzionalità aggiuntive relative alla pagina in cui l'utente si trova.

Come detto nell'introduzione, l'applicazione funziona anche in modalità offline, mostrando un'icona per notificare l'utente. In questa modalità, tutte le funzionalità principali restano disponibili, come la visualizzazione delle offerte e l'aggiunta al carrello. Tuttavia, non è possibile accedere al proprio account se non si è già effettuato l'accesso, mentre gli utenti già loggati rimangono connessi. Inoltre, l'aggiunta di nuovi negozi è disabilitata, poiché il recupero dei prodotti richiede una connessione a internet.

Pagina iniziale

La schermata principale raccoglie tutte le offerte provenienti dai supermercati preferiti, selezionabili dall'utente.

Le offerte possono essere ordinate in modo **ascendente** o **discendente tenendo conto dello sconto** (attraverso il tasto a sinistra di Cerca), e possono essere cercate con la barra di ricerca.

Ogni offerta può essere aggiunta rapidamente al proprio carrello, accessibile direttamente tramite il pulsante dedicato in basso a destra.

È possibile anche ricaricare la pagina, attraverso il tasto in alto a sinistra.

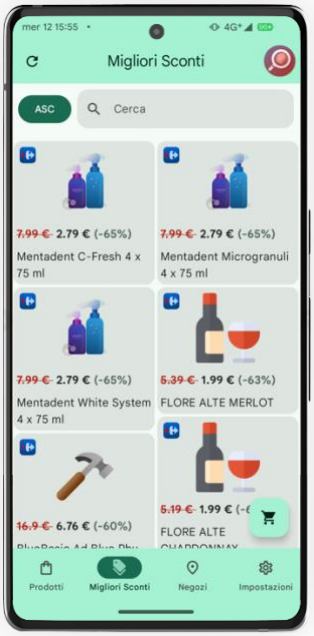


Fig. 1 · Schermata principale

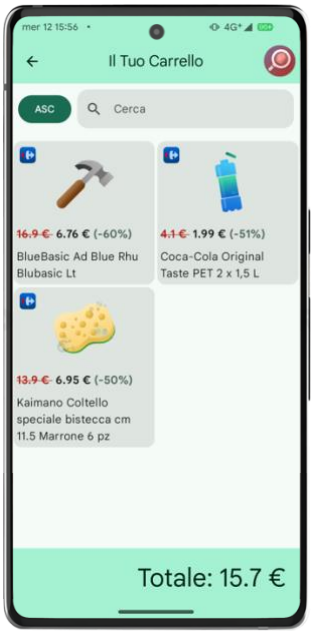


Fig. 2 · Carrello personale

Dettaglio delle offerte

Toccando la card di un'offerta, si accede a una schermata con maggiori dettagli.

In alto, viene mostrata l'immagine del prodotto, se disponibile; in caso contrario, appare un'illustrazione della categoria di appartenenza.

Sono visibili il **nome** del prodotto, la sua **categoria**, il **prezzo originale**, il **prezzo scontato**, e la **percentuale di sconto**. Da questa schermata, è anche possibile aggiungere l'offerta al carrello.

Fig. 3 · Offerta senza immagine

Fig. 4 · Offerta con immagine

Pagina dei prodotti

Questa sezione mostra l'elenco completo delle offerte disponibili nei tre supermercati. L'utente può applicare **diversi filtri** per personalizzare la visualizzazione, scegliendo da quali supermercati vedere le offerte e in quali categorie. Per fare questo, è necessario schiacciare l'icona Filtro in alto a sinistra.

È possibile scegliere di visualizzare le offerte solo dal supermercato che si sceglie, ma è possibile scegliere **più di una categoria** all'interno del supermercato scelto.

Fig. 5 · Selezione delle categorie

Negozi

Nella pagina dei negozi, è possibile scegliere quali sono i negozi da cui visualizzare le offerte nella pagina principale.

Per l'Esselunga, è necessario inoltre scegliere da quale punto vendita si vogliono avere le offerte. Schiacciando sul pulsante di modifica, viene aperta una WebView in cui è possibile **scegliere dalla mappa** quale punto vendita si vuole. Una volta scelto, bisogna soltanto toccare "**Sfoglia il volantino**" per avere le offerte.

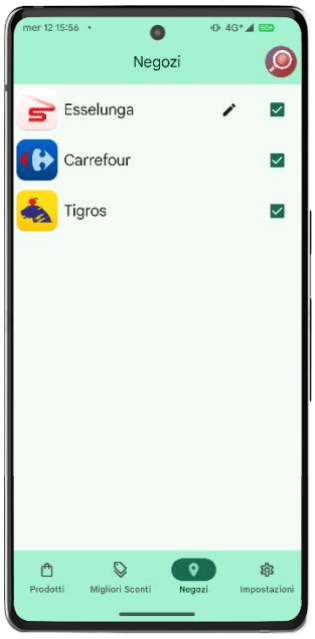


Fig. 6 · Selezione dei negozi

Fig. 7 · Mappa Esselunga

Impostazioni

Nella pagina delle impostazioni, l'utente ha la possibilità di personalizzare l'esperienza di utilizzo con tre opzioni:

- **Testo in grassetto** per migliorare la leggibilità delle offerte;
- **Dimensione delle immagini**, sempre per migliorare la visibilità di quest'ultime;
- La **frequenza delle notifiche toast**, che determina quali notifiche toast mostrare nell'app. Si può scegliere tra tutte, importanti, e solo errori.

> **Notifiche toast**
> *Brevi messaggi a comparsa che informano l'utente su eventi o azioni, senza interrompere l'uso dell'app. Sono usati, per esempio, per notificare l'utente dell'avvenuto caricamento dei dati dell'account sul cloud.*

Fig. 8 • Pagina delle impostazioni

Gestione Account

Dalle impostazioni, è possibile **registrarsi** o **accedere** a un account esistente. Dopo l'accesso, si possono gestire le impostazioni del proprio profilo, come *nome, foto e password*. Inoltre, è possibile **salvare e ripristinare i propri dati** (per il momento solo le impostazioni) **sul cloud**. L'app può essere utilizzata anche **senza un account**, ma in questo caso il salvataggio sul cloud non sarà disponibile.

Se si è dimenticati la propria password, è possibile resettarla attraverso la propria e-mail. In caso di creazione di un nuovo account, è possibile anche verificare la propria e-mail.

Fig. 9 • Schermata di accesso

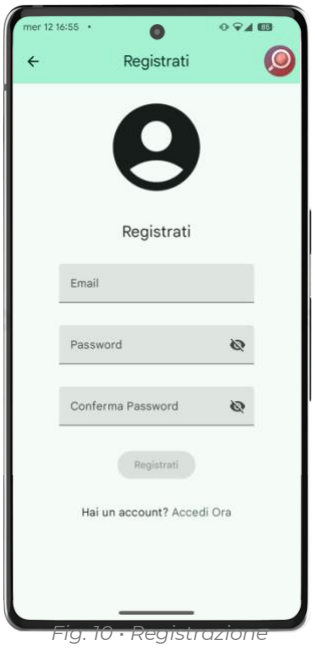


Fig. 10 • Registrazione

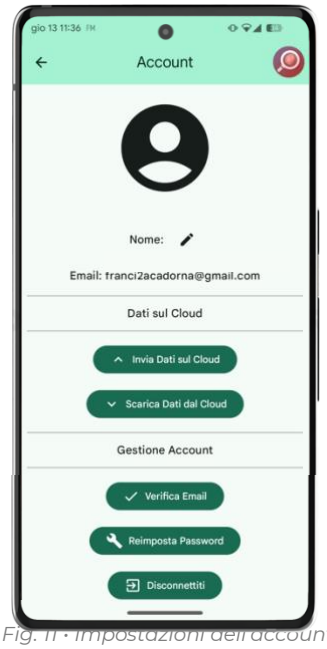


Fig. 11 • Impostazioni dell'account

Librerie utilizzate

UI

- *Jetpack Compose*

Storage

- *Room*
- *Protobuf*

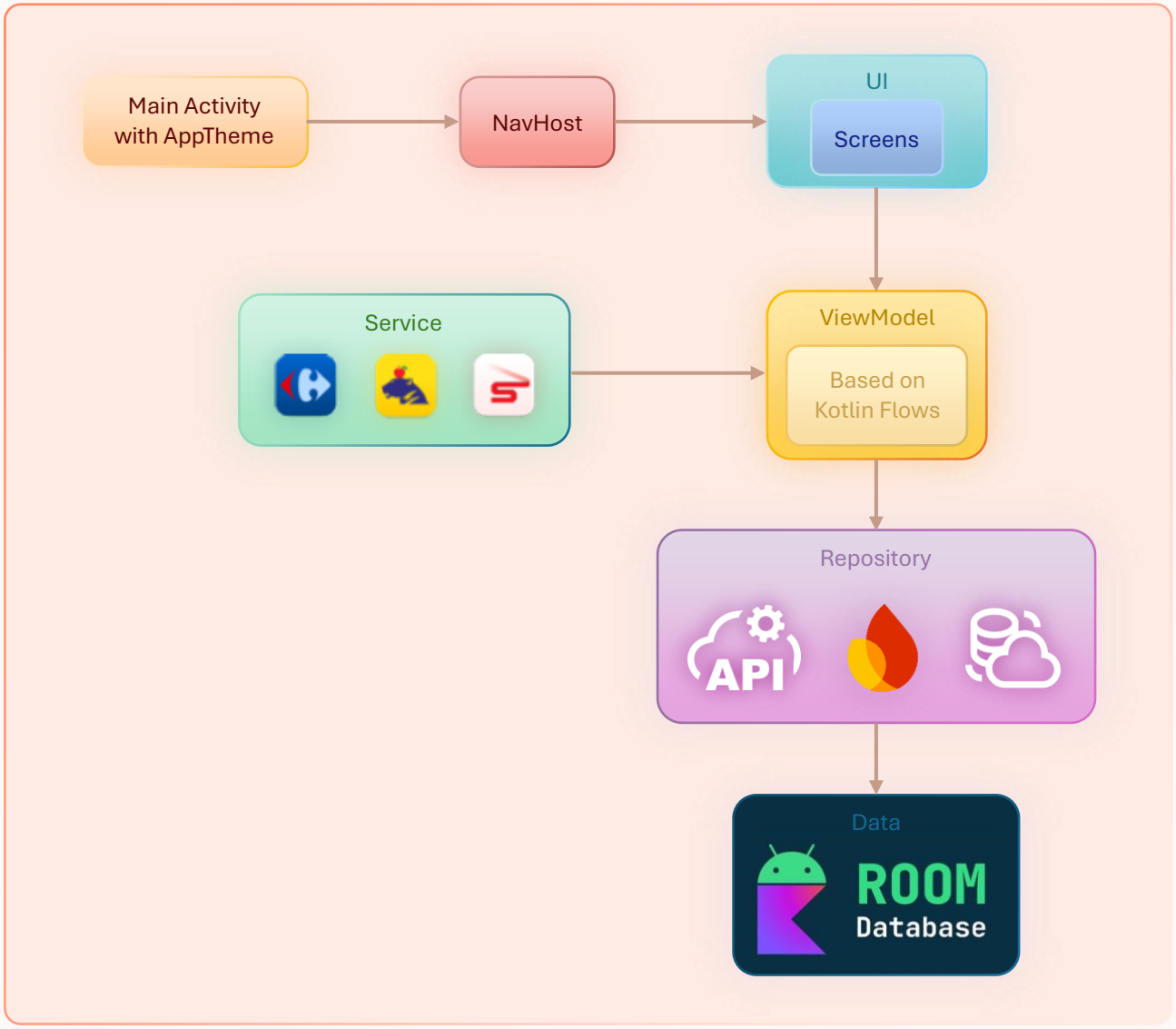
Librerie esterne

- *Ktor*
- *Ksoup*
- *Coil*

Librerie importanti

- *WorkManager*
- *LifeCycle*
- *Navigation Compose*
- *Firebase auth & storage*

Architettura



L'applicazione è costituita dai seguenti componenti principali, presentati nelle prossime pagine.

Main Activity

Questa rappresenta il nucleo dell'applicazione, sviluppato in modo unico utilizzando **Jetpack Compose** per garantire una UI moderna e reattiva.

NavHost

L'app utilizza **Jetpack Navigation** con **Compose** in un'architettura single-activity. Il controller principale di navigazione è gestito tramite **NavHost**, che definisce il punto di partenza dell'app e le rotte di navigazione. Le azioni di navigazione sono implementate con **navLambda** per la **navigazione standard**, e **navLambdaRemoveLast** per manipolare lo **stack di navigazione**, come nel caso di flussi specifici.

Le schermate principali includono *DealsScreen*, *ShoppingListScreen*, *ProductScreen*, *StoresScreen*, *AccountScreen*, *LoginScreen*, *RegisterScreen* e *SettingsScreen*.

Inoltre, l'app include componenti UI per la navigazione, come la **barra superiore** (*TopBar*) e la **barra inferiore** (*NavigationBottomBar*).

La struttura della navigazione offre numerosi vantaggi, tra cui una gestione centralizzata della navigazione, integrazione con i *ViewModel* per la gestione dello stato e coerenza visiva attraverso componenti di navigazione. L'implementazione dell'architettura consente di gestire scenari di navigazione semplici e complessi.

UI

L'architettura dell'interfaccia utente segue un modello moderno basato su Android, utilizzando **Jetpack Compose** per i componenti UI, il pattern **MVVM** (Model-View-ViewModel) e il **Navigation Component** per la gestione della navigazione tra le schermate. Le componenti comuni, come *TopBar*, *ProductCard* e *LoadingComponent*, sono riutilizzabili in tutta l'app.

La riusabilità e la consistenza nelle schermate sono assicurate attraverso componenti comuni condivisi e un design standardizzato, con un'attenzione alla responsività e alla gestione centralizzata degli stati operativi.

ViewModel

L'architettura dei **ViewModel** in Deal Detective segue un pattern ben definito, con una gerarchia che parte da una **classe base astratta**, *BaseViewModel*, che gestisce lo stato delle operazioni (*Idle, Loading, Success, Error*) e fornisce metodi come *startOperation()* e *resetOperation()*. Le **classi derivate**, come AProductsViewModel, offrono **funzionalità**

comuni come ordinamento e ricerca, mentre i ViewModel specifici come *AccountViewModel*, *ProductsViewModel* e *StoresViewModel* gestiscono aspetti particolari come autenticazione, prodotti e negozi, utilizzando tecniche come la gestione dello stato tramite *StateFlow* e il pattern repository per separare le operazioni di accesso ai dati.

L'architettura sfrutta anche **coroutines** per operazioni **asincrone** e **dependency injection** tramite il *factory pattern*, garantendo una struttura scalabile e mantenibile con una chiara separazione delle responsabilità. Lo stato dell'interfaccia è gestito attraverso delle sealed interface per una rappresentazione type-safe dello stato.

Repository

L'architettura di DealDetective adotta il pattern repository in modo ben strutturato, con repository principali come *ProductsRepository*, *SettingsRepository* e *StoresSettingsRepository*, e repository specifici per **Firebase**, inclusi *FirebaseAuthRepository* per l'autenticazione e *FirebaseDatabaseRepository* per la gestione dei dati utente remoti.

Utilizza un **container** per la dependency injection tramite **AppContainer**, garantendo una separazione chiara delle responsabilità tra i vari componenti.

Ogni repository **implementa un'interfaccia**, facilitando il testing, e viene impiegato un tipo *Result* per gestire successi e errori. L'uso estensivo delle coroutines per le operazioni asincrone, insieme alla *Clean Architecture* che separa dati, dominio e view, assicura un'app testabile, mantenibile e facilmente scalabile.

Service

L'architettura del layer di servizio di DealDetective si basa su una struttura ben organizzata, in cui il principale componente, il *StoresServiceHandler*, coordina l'inizializzazione degli *scraper* e le operazioni di recupero dei prodotti, interagendo con i vari repository. I vari scraper per i negozi, come *EsselungaScraperService* e *CarrefourScraperService*, implementano interfacce specifiche per gestire la raccolta dei dati relativi agli sconti, separando le operazioni comuni e quelle più specifiche.

L'architettura utilizza pattern come *dependency injection* e la segregazione delle interfacce per garantire una gestione flessibile e scalabile dei servizi. Inoltre, viene impiegato il *Result Pattern* per la gestione degli errori e della risposta dei servizi e lo stato delle operazioni viene monitorato tramite StateFlow. La responsabilità del layer di servizio include la gestione dello scraping dei dati, la sincronizzazione delle operazioni tra diversi scraper e il tracciamento dello stato delle operazioni, con una chiara separazione delle preoccupazioni e la possibilità di estendere facilmente l'architettura con nuovi scraper.

Data

DealDetective adotta un'architettura che integra **Room** e **Protobuf** per la gestione dei dati in modo efficiente. **Room** è utilizzato come database locale per memorizzare informazioni sui prodotti, gestendo operazioni *CRUD* tramite un'interfaccia *DAO* e supportando flussi reattivi con *Kotlin Flows*.

Per le **impostazioni** dell'app e le configurazioni dei negozi, si impiega **Protobuf** insieme a **DataStore**, garantendo una serializzazione compatta ed efficiente. I dati vengono gestiti in modo sicuro grazie all'uso di *Protobuf* per la memorizzazione dei settings e dei parametri di configurazione.

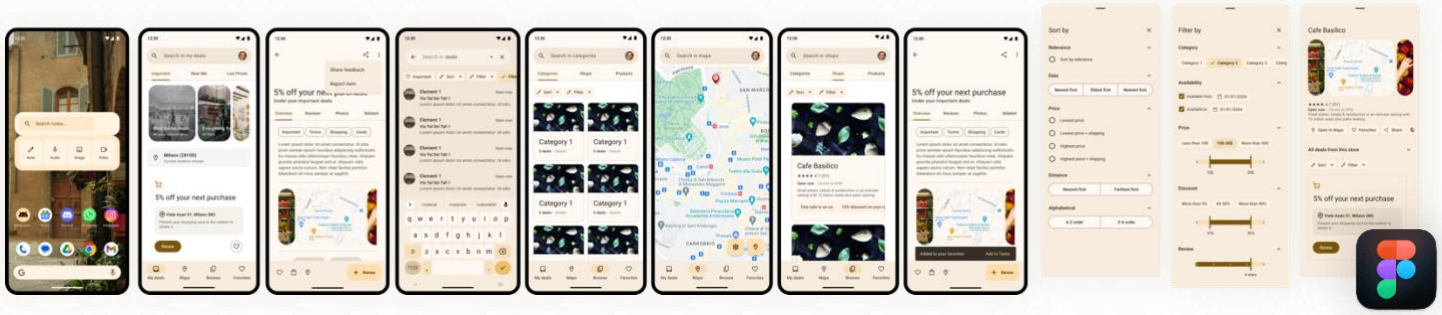
L'architettura segue il pattern del repository, che astragga le fonti di dati per offrire API pulite ai *ViewModel*, separando chiaramente la logica di accesso ai dati dalla logica di presentazione e migliorando la gestione e il recupero dei dati in modo reattivo e sicuro.

Design

L'app è stata progettata seguendo le linee guida di design di Android, per garantire una perfetta integrazione con la piattaforma e un'esperienza utente ottimale.

Il design si ispira ai principi di **Material Design 3**, il sistema introdotto con Android 12, che permette una personalizzazione visiva dinamica e adattiva. Grazie al supporto dei temi, l'app si adatta automaticamente alle preferenze estetiche dell'utente, offrendo un'interfaccia moderna, coerente e facilmente navigabile.

Per la progettazione del design dell'applicazione, è stato inizialmente creato un mockup su Figma. Questo ci ha permesso di definire l'interfaccia ideale per l'app, fornendo una base visiva su cui fare riferimento durante lo sviluppo finale dell'app su Android Studio.



L'icona

L'icona dell'app è stata progettata per comunicare immediatamente la funzione dell'applicazione: la lente di ricerca presenta all'interno dei tag che richiamano le offerte e gli sconti, rendendo chiaro il suo scopo principale. Abbiamo cercato di garantire che l'icona fosse facilmente riconoscibile anche a dimensioni ridotte.

I colori scelti per l'icona sono distintivi dell'app, sebbene non siano gli stessi utilizzati nell'interfaccia, la quale segue il sistema di temi di Android, per garantire coerenza con il resto del sistema e Material You. La scelta dei colori dell'icona viene spiegata nella sezione *Color palette*.

L'icona è stata realizzata in Figma, usando un icon template, e ha subito diverse evoluzioni:

Versione finale dell'icona, con e senza effetti grafici (icona no. 5 e 6)

Le illustrazioni

Per le offerte prive di immagini, utilizziamo un'illustrazione rappresentativa della loro categoria.

Queste illustrazioni sono state realizzate in Figma per garantire **coerenza visiva**, **facilità di riconoscimento**, ed **ottima visibilità** sia su sfondo chiaro che scuro, anche a piccole dimensioni.

Di seguito, sono presenti alcuni esempi:

Illustrazioni su sfondo chiaro (a sinistra), e su sfondo scuro (a destra)

Color palette

La scelta della color palette per l'icona dell'app è stata fatta in base a colori che trasmettessero un senso di tranquillità e familiarità. L'obiettivo era creare un'icona visivamente amichevole e facilmente riconoscibile, con un'atmosfera accogliente che invogli l'utente a interagire con l'app. Sono stati scelti quindi i seguenti due colori per creare la sfumatura nell'icona:

Dusky purple	Pale carmine
#8A617C	#A63B3B

L'app non ha una color palette, poiché, aderendo ai principi di Material Design 3, si adatta automaticamente al tema di sistema.

Material Design 3

L’app segue i principi di Material Design 3, lo stile grafico introdotto da Google con Android 12.

Si adatta automaticamente al tema di sistema, come mostrato nelle seguenti screenshot – indipendentemente dal tema scelto, l’app garantisce una lettura ottimale, e un aspetto visivamente gradevole, in ogni suo componente.

Pattern utilizzati

Model-View-ViewModel (MVVM) pattern

MVVM è un pattern architetturale che separa la logica di presentazione dalla logica di business.

- Il **Model** gestisce i dati, la validazione e le regole di business senza dipendere dall'UI.
- La **View** è responsabile dell'interfaccia utente, osserva il ViewModel per aggiornamenti e inoltra le azioni dell'utente senza contenere logica di business.
- Il **ViewModel** funge da ponte tra Model e View, gestendo lo stato dell'UI e la logica di presentazione senza riferimenti diretti alla View.

Questo approccio offre diversi vantaggi, tra cui **separazione delle responsabilità**, testabilità del codice senza UI, **manutenibilità** grazie a un'architettura modulare, **riusabilità** dei componenti e una **gestione centralizzata** dello stato dell'interfaccia utente.

Dependency Injection

La **Dependency Injection** (DI) è un pattern di progettazione in cui le dipendenze (servizi o oggetti) vengono "iniettate" in una classe invece di essere create al suo interno. Questo approccio favorisce un basso accoppiamento tra i componenti, migliora la testabilità, rende il codice più mantenibile e promuove una chiara separazione delle responsabilità.

Nel progetto utilizziamo due container:

1. **AppContainer** per dipendenze relative all'intera applicazione;
2. **ActivityContainer** per dipendenze relative all'interfaccia.

Repository Pattern

Il **Repository Pattern** è un pattern architetturale che astrae la logica di accesso ai dati dai *ViewModel* o da altri componenti dell'applicazione. Questo permette una chiara separazione tra le sorgenti di dati (database, API, file, ecc.) e la logica di business, rendendo il codice più modulare, testabile e manutenibile.

Result Pattern

Il **Result Pattern** è un approccio per la gestione degli errori che utilizza un tipo *Result* personalizzato invece di eccezioni per gestire i casi di errore previsti. Questo migliora la chiarezza del codice, evita il costo delle eccezioni e rende più esplicito il flusso dei dati, facilitando la gestione degli esiti positivi e degli errori in modo strutturato.

Factory Pattern

Il **Factory Pattern** è un pattern di creazione utilizzato per istanziare oggetti, come i *ViewModel*, gestendo le loro dipendenze in modo centralizzato.

In Android, viene implementato tramite *ViewModelProvider.Factory*, permettendo di fornire parametri personalizzati ai *ViewModel* senza violare il principio di inversione delle dipendenze.

Observer

L'**Observer Pattern** è un pattern di progettazione che consente a più componenti di osservare e reagire ai cambiamenti di stato in modo automatico.

In Kotlin, viene implementato tramite *Kotlin Flows*, come *StateFlow* e *Flow*, permettendo una gestione reattiva dei dati e aggiornamenti efficienti dell'interfaccia utente.

Builder

Il **Builder Pattern** è un pattern di progettazione utilizzato per costruire oggetti complessi passo dopo passo, separando la creazione dell'oggetto dalla sua rappresentazione.

Questo pattern è utile quando un oggetto ha molte proprietà opzionali o configurazioni, evitando costruttori con numerosi parametri. Un esempio di utilizzo è il Proto *DataStore* in Android.

Service

Il **Service Pattern** è un pattern di progettazione utilizzato per separare la logica di business e le operazioni di lunga durata (come chiamate API, operazioni su database o altre attività di backend) in un'astrazione chiamata "servizio".

Il *Service Pattern* si concentra sull'orchestrazione di logiche complesse e operazioni di lunga durata che potrebbero includere chiamate al database, ma non sono limitate ad esse. Mentre il *Repository* di solito non contiene logiche di business, il *Service Pattern* le incapsula.

I servizi sono generalmente progettati per essere riutilizzabili, testabili e modulari, e sono spesso utilizzati per centralizzare la gestione di attività comuni, come l'autenticazione, la gestione dei dati o la comunicazione con altri sistemi. In pratica, i service possono essere invocati da altre parti dell'applicazione per eseguire operazioni senza compromettere la logica di presentazione o la gestione dello stato.

Sviluppi futuri

In futuro, prevediamo di ampliare le funzionalità dell'app in diversi ambiti:

- **Sicurezza**:
  - Consentire l'accesso all'app tramite biometria;
  - Consentire l'autenticazione al proprio account tramite *passkey* e/o gestore di password di sistema;
- **Usabilità**:
  - Ottimizzare l'interfaccia per gli schermi orizzontali (tablet e dispositivi pieghevoli inclusi);
  - Implementare la funzionalità "pull to refresh" nelle schermate;
  - Abilitare la navigazione tramite gesture tra le pagine;
  - Migliorare l'usabilità per l'uso con una sola mano (es. spostare i controlli del filtraggio in basso);
- **Connettività**:
  - Integrare il carrello con Google Keep, per poter tenere traccia delle offerte anche nelle proprie note;
  - Consentire l'esportazione e l'importazione del carrello;
  - Aggiungere la possibilità di condividere offerte e/o il proprio carrello;
- **Offerte**:
  - Per migliorare la visibilità delle offerte, eseguire una ricerca online per i prodotti che non hanno immagini;
  - Integrare più supermercati che hanno volantini online;
  - Mostrare una notifica che suggerisce di vedere le offerte relative a un supermercato, quando ci si trova vicino ad uno;
- **Material You**:
  - Implementazione di widget per mostrare rapidamente offerte nella schermata home;
  - Icona dell'app che segue il tema di sistema;
- **Account**:
  - Aumentare la tipologia di dati salvabili sul cloud, come ad esempio l'immagine di profilo, il nome, etc.