

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIEM - Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica
Applicata



Corso di Laurea Triennale in Ingegneria Informatica

Progetto - Mobile Programming **Groceroo**

Gruppo 14:
Vincenzo Ragone - 0612708147

Gabriele Lupo - 0612707641

Mohamad Naim - 0612707348

Silvio Ventura - 0612707724

Gianmarco Vitolo - 0612708376

ANNO ACCADEMICO 2024/2025

Indice

1 Introduzione	2
1.1 Panoramica	2
2 Mockup e Design dell'Interfaccia	2
2.1 Wireframe principali	2
2.1.1 Schermata principale (Home)	3
2.1.2 Dettaglio lista della spesa	5
2.1.3 Aggiunta/modifica spesa	7
2.1.4 Prodotti recentemente acquistati	8
2.1.5 Liste della spesa	10
2.1.6 Analisi e statistiche	11
2.1.7 Gestione categorie	12
2.2 Flussi di navigazione	14
3 Architettura dell'applicazione	14
3.1 Librerie e dipendenze esterne	14
3.2 Pattern architettonico	14
3.3 Organizzazione del codice sorgente	15
3.4 Persistenza dei dati	15
3.4.1 Schema del database	15
3.4.2 Strategie di ottimizzazione	16
4 Caratteristiche principali	16
4.1 Gestione delle spese	16
4.1.1 Registrazione delle spese	16
4.1.2 Visualizzazione e modifica	17
4.2 Liste della spesa	17
4.2.1 Creazione e gestione	17
4.3 Categorie personalizzabili	17
4.4 Ricerca avanzata	17
4.5 Analisi e statistiche	17
4.5.1 Dashboard principale	18
4.5.2 Analisi temporale	18
5 Caratteristiche tecniche avanzate	18
5.1 Ottimizzazione delle prestazioni	18
6 Sfide affrontate e soluzioni	18
6.1 Sincronizzazione tra componenti dell'interfaccia	18
6.2 Ottimizzazione delle performance nella visualizzazione di liste	19

1 Introduzione

1.1 Panoramica

Groceroo è un'applicazione intelligente per la gestione quotidiana della spesa. Progettata per essere semplice, veloce e personalizzabile, l'app consente di:

- Registrare e monitorare ogni spesa con dettagli su prodotti, quantità, prezzo, data e categoria.
- Creare e gestire liste della spesa interattive, spuntando facilmente i prodotti acquistati.
- Personalizzare le categorie per una classificazione ordinata degli acquisti (alimentari, igiene, casa, ecc.).
- Trovare rapidamente ciò che serve grazie a una funzione di ricerca avanzata con filtri per nome, data, prezzo o categoria.
- Analizzare le abitudini di consumo con grafici intuitivi: andamento mensile, media settimanale, distribuzione per categoria e prodotti più acquistati.

2 Mockup e Design dell'Interfaccia

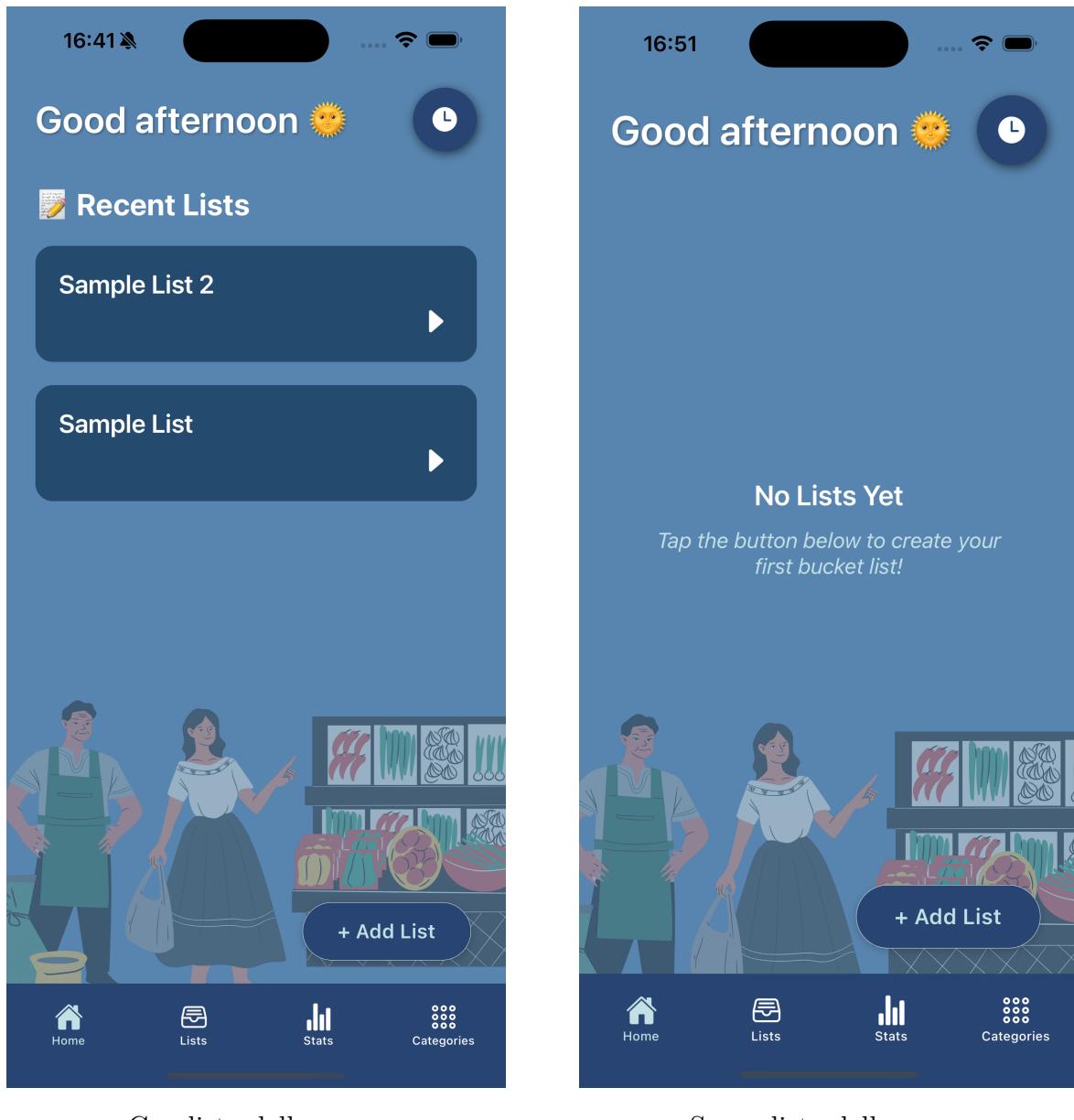
2.1 Wireframe principali



2.1.1 Schermata principale (Home)

La schermata principale è stata progettata per offrire un accesso rapido alle funzionalità più utilizzate e si adatta dinamicamente in base al contenuto disponibile. In particolare, l'interfaccia cambia a seconda che siano presenti o meno delle liste della spesa salvate:

- **Quando sono presenti liste della spesa:** vengono mostrate le ultime spese effettuate con visualizzazione compatta, e l'utente può accedere rapidamente alle liste attive.
- **Quando non sono presenti liste:** la schermata evidenzia un invito all'azione per creare una nuova lista, presentando un'interfaccia più semplice e focalizzata sull'aggiunta iniziale.
- Pulsante FAB (Floating Action Button) per aggiungere rapidamente una nuova lista della spesa.
- Barra di navigazione inferiore per accedere alle altre sezioni dell'app.



Con lista della spesa

Senza lista della spesa

Figura 1: Confronto dell'app con e senza liste

2.1.2 Dettaglio lista della spesa

La schermata di dettaglio di una lista della spesa consente all'utente di visualizzare e gestire in modo completo gli elementi presenti nella lista selezionata. L'interfaccia si adatta dinamicamente in base al contenuto della lista:

- **Quando la lista contiene prodotti:** viene mostrato l'elenco completo degli articoli aggiunti, ciascuno con quantità, eventuale categoria e stato (acquistato o meno). In più , viene aggiornato in tempo reale l'ammontare della spesa totale contestualmente a quella lista man mano che si spuntano i prodotti come acquistati
- **Quando la lista è vuota:** viene visualizzato un messaggio che invita l'utente ad aggiungere nuovi prodotti, accompagnato da un FAB animato , migliorando l'UX.
- Intestazione con il nome della lista e la data di creazione o ultima modifica.
- Pulsante per aggiungere rapidamente nuovi prodotti alla lista.
- Funzionalità per modificare, duplicare o eliminare la lista.

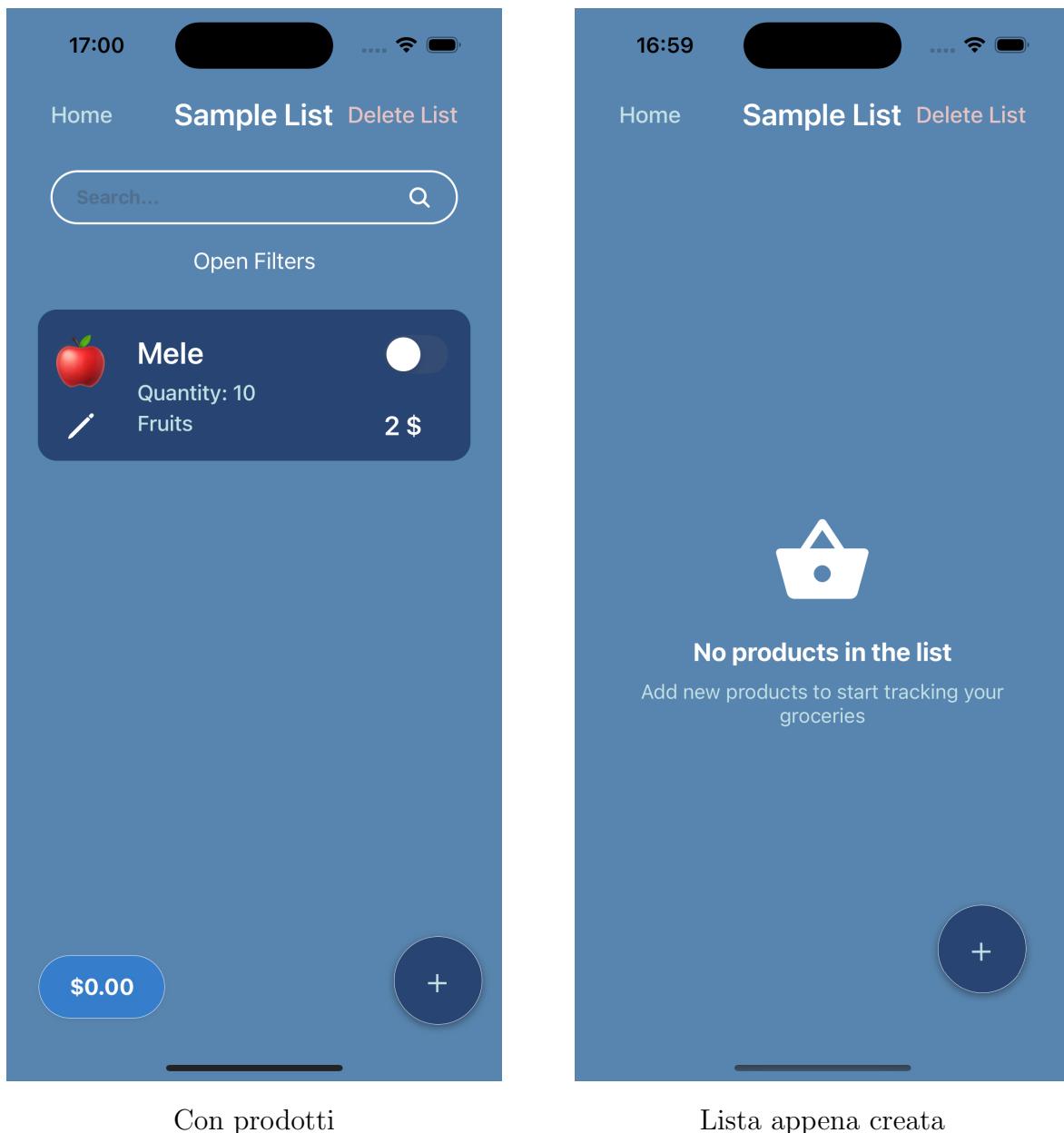
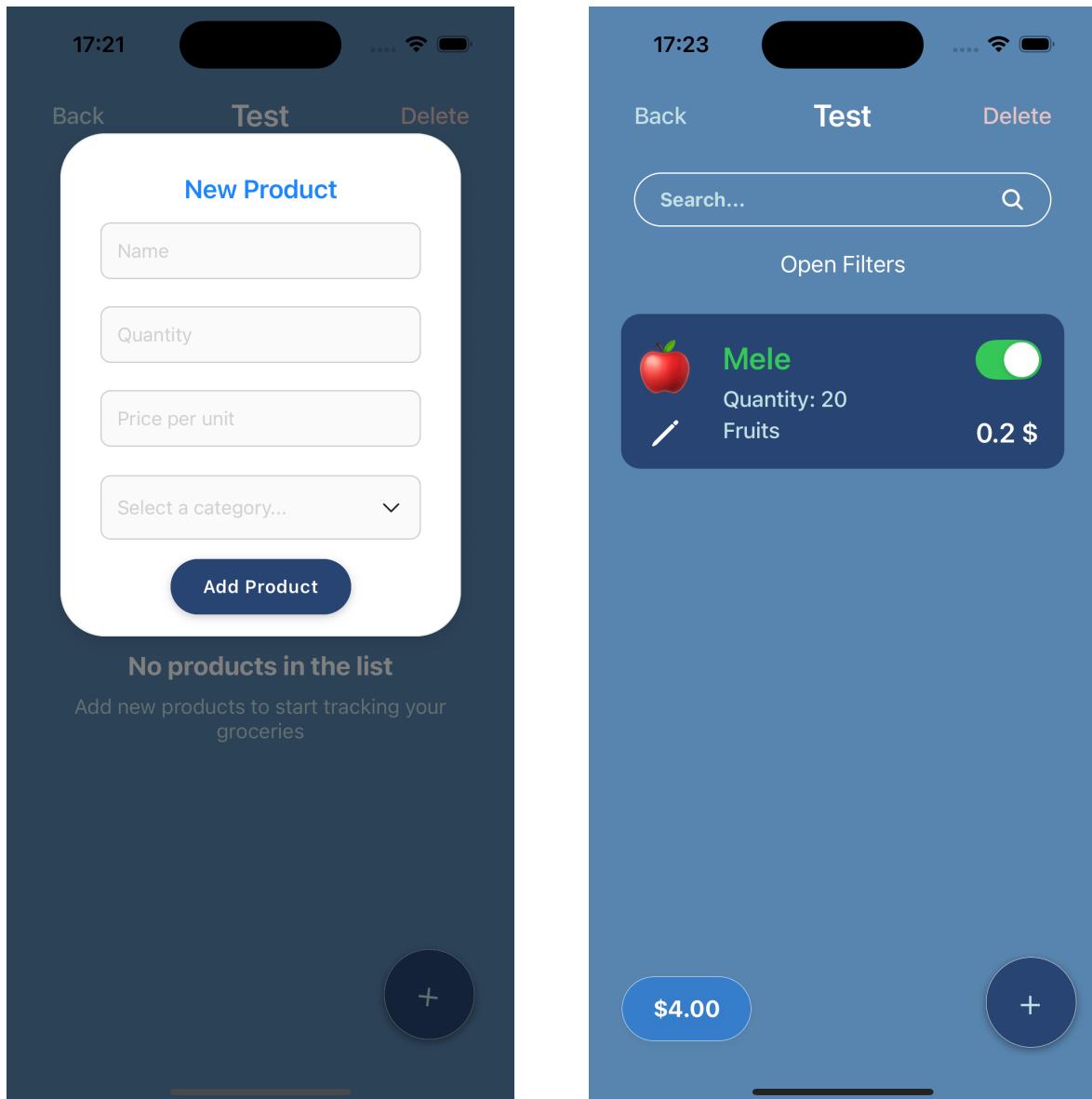


Figura 2: Confronto della schermata con e senza prodotti inseriti nella lista della spesa

2.1.3 Aggiunta/modifica spesa

Form strutturato per l'inserimento o la modifica di una spesa:

- Aggiunta prodotti con campi per nome, quantità, prezzo unitario e categoria
- Calcolo automatico del totale



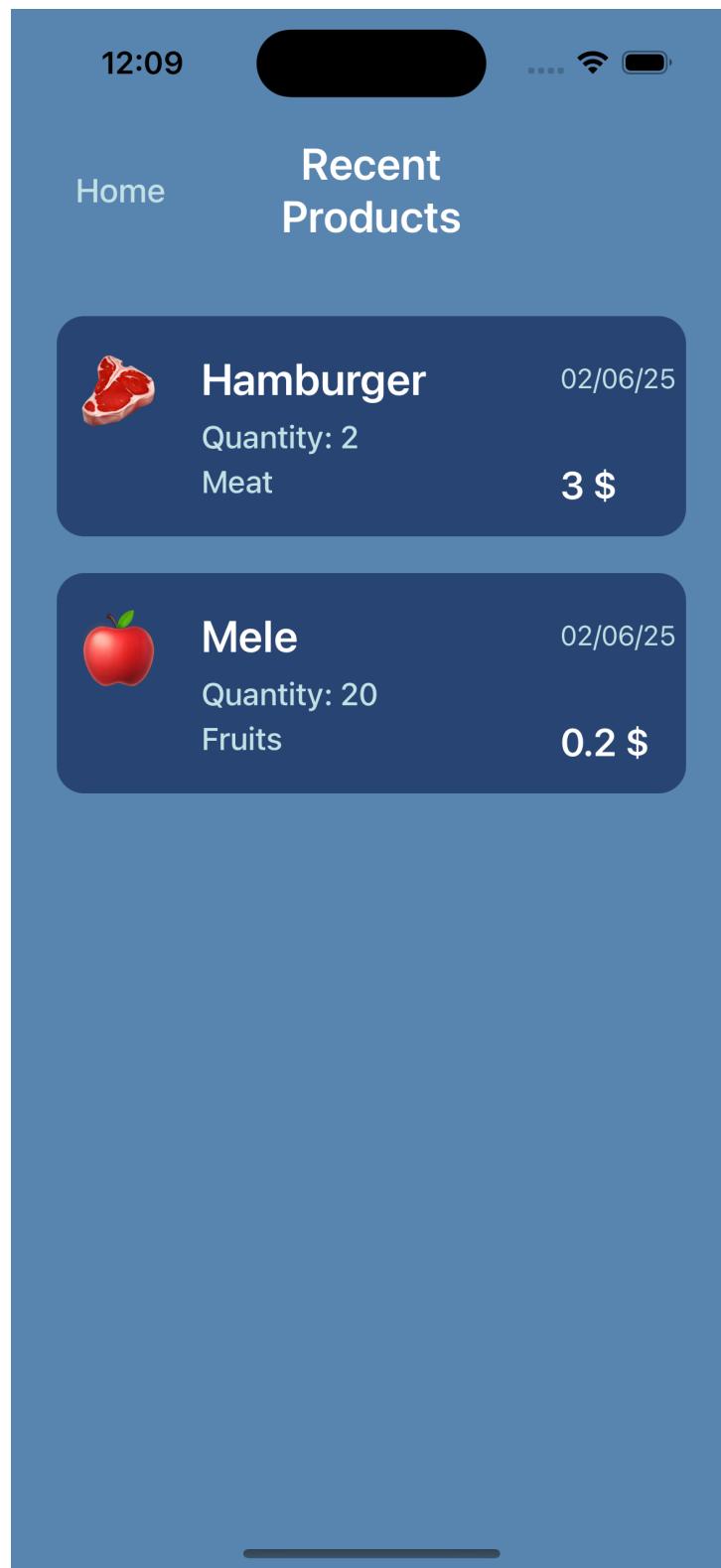
Form di aggiunta di una nuova spesa

Prodotto marcato come acquistato ,
con report del totale speso in basso a
sinistra

2.1.4 Prodotti recentemente acquistati

Interfaccia dedicata alla consultazione dei prodotti acquistati di recente, utile per riutilizzare articoli frequenti e tenere traccia della spesa abituale:

- Elenco dei prodotti acquistati con visualizzazione di nome, quantità, prezzo e data

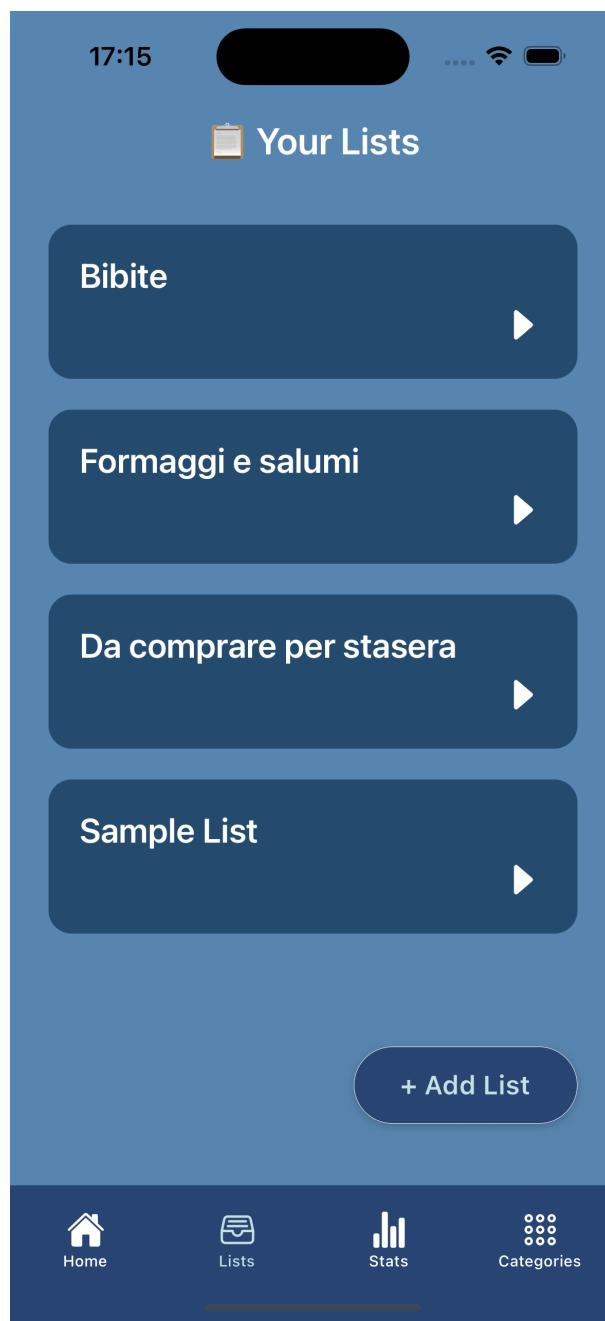


Schermata dei prodotti recentemente acquistati con opzioni di riaggiunta

2.1.5 Liste della spesa

Visualizzazione delle liste della spesa con:

- Elenco delle liste con titolo
- FBA per l'aggiunta di una nuova lista direttamente nella pagina stessa



2.1.6 Analisi e statistiche

Dashboard con diverse visualizzazioni per analizzare le abitudini di spesa:

- Grafico a torta per la distribuzione delle spese per categoria
- Grafico a linee per l'andamento della spesa nel tempo



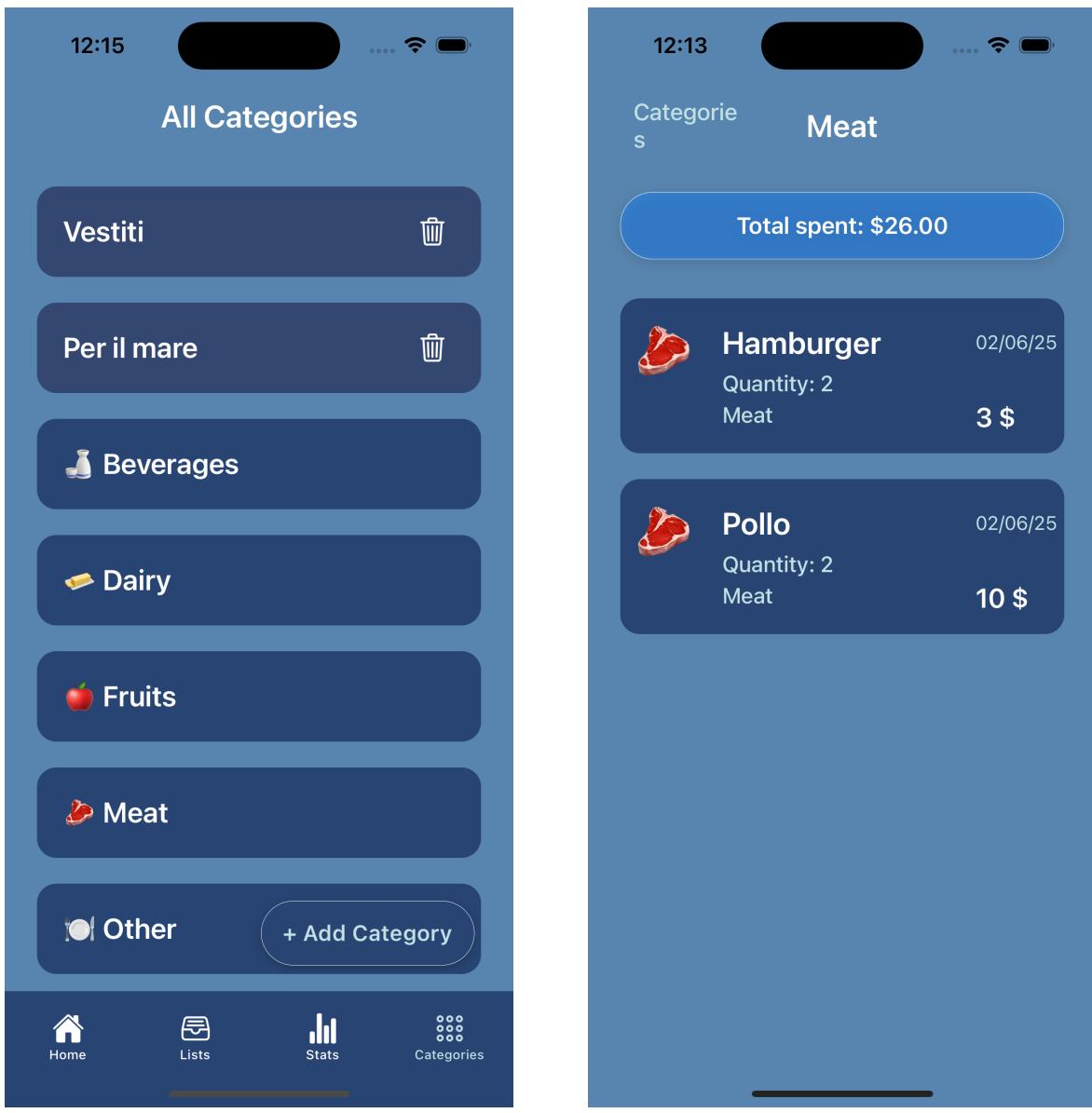
Figura 3: Componenti per le statistiche

2.1.7 Gestione categorie

Schermata dedicata alla visualizzazione e modifica delle categorie, utile per organizzare le spese in modo personalizzato:

- Visualizzazione dell'elenco categorie esistenti
- Possibilità di aggiungere una nuova categoria tramite FAB (+)
- Modal di inserimento nome della categoria
- Possibilità di vedere il totale speso per una determinata categoria

2 MOCKUP E DESIGN DELL'INTERFACCIA



Tutte le categorie

Visualizzazione dei dettagli della singola categoria

Figura 4: Tutto sulle categorie

2.2 Flussi di navigazione

I principali flussi di navigazione dell'app sono stati progettati per minimizzare il numero di interazioni necessarie per completare le operazioni più frequenti:

- **Registrazione nuova spesa:** Home → Lista Specifica → FAB (+) → Modal di aggiunta prodotto → Conferma
- **Creazione lista della spesa:** Home → FAB(+ Add List) → Modal di aggiunta lista → Conferma
- **Consultazione statistiche:** Home → Tab Analisi → Selezione periodo → Visualizzazione grafici
- **Ricerca prodotto:** Home → Lista Specifica → Inserimento query → Filtri → Risultati

3 Architettura dell'applicazione

3.1 Librerie e dipendenze esterne

Per implementare le funzionalità richieste, abbiamo integrato diverse librerie:

Libreria	Utilizzo	Motivazione
React Navigation	Sistema di navigazione tra schermate	Gestione efficiente della navigazione e transizioni fluide
SQLite	Database locale per la persistenza dei dati	Storage strutturato con supporto per query complesse
Victory Native	Visualizzazione di grafici e statistiche	Libreria di data visualization completa e personalizzabile
React Native Paper	Componenti UI	Design system coerente basato su Material Design
React Native Vector Icons	Icone	Ampia collezione di icone vettoriali per migliorare l'UI
React Native Reanimated	Animazioni fluide	Animazioni ottimizzate per migliorare l'esperienza utente

3.2 Pattern architettonico

Per Groceroo abbiamo adottato un'architettura ispirata ai principi della Clean Architecture, che si riflette nella struttura organizzativa del progetto. L'applicazione è suddivisa in livelli logici, ognuno con una responsabilità chiara:

- **Livello di presentazione:** È rappresentato principalmente dalle cartelle `screens`, `components` e `navigation`. Include i componenti React, le schermate

e la logica di navigazione, gestendo l'interazione con l'utente e la visualizzazione dei dati.

- **Livello di gestione dello stato:** Lo stato dell'applicazione è gestito attraverso gli hook di React (`useState`, `useEffect`), distribuiti all'interno dei componenti e delle schermate, senza l'utilizzo di librerie esterne come Redux.
- **Livello di business logic:** Alcuni file all'interno di `components` (come `btntsObj.js`, `modalObj.js`, ecc.) contengono funzioni e strutture che incapsulano la logica applicativa e di supporto, mantenendo separata l'elaborazione dei dati dalla UI.
- **Livello di accesso ai dati:** La cartella `data` e il file `db.tsx` gestiscono l'interazione con il database locale SQLite, occupandosi della persistenza e del recupero dei dati.

3.3 Organizzazione del codice sorgente

Il progetto è organizzato secondo una struttura modulare che riflette l'architettura descritta:

```
/src
  /assets                      # Risorse statiche (immagini, font, ecc.)
  /components                   # Componenti UI riutilizzabili
    /graphs                     # Componenti per le statistiche spesa(PieChart, ecc)
    /navigation                 # Configurazione della navigazione (BottomBar e MainStack)
    /screens                    # Schermate dell'applicazione
    /data                       # Configurazione database e query di accesso ai dati
    App.js                      # Punto di ingresso dell'applicazione
```

3.4 Persistenza dei dati

3.4.1 Schema del database

Per la persistenza dei dati abbiamo implementato un database SQLite con il seguente schema:

Tabella	Campi	Descrizione
Lists	id , name , inserted-at	Liste dei prodotti personalizzabili dall'utente
Items	id, name , quantity , price , category , inserted-at , list-id , comprato , data-compera	Prodotti della spesa che l'utente può inserire e marcare come acquistati
Categories	name	Categorie personalizzabili dall'utente

3.4.2 Strategie di ottimizzazione

Per migliorare la coerenza e la manutenzione del database, abbiamo definito specifiche policy di reazione all'eliminazione (ON DELETE CASCADE) per i vincoli di foreign key:

- **items(list_id) → lists(id):** In caso di eliminazione di una lista, tutti gli elementi (items) associati vengono automaticamente rimossi. Questo comportamento garantisce che non rimangano record orfani nella tabella `items`, migliorando l'integrità referenziale e riducendo la necessità di interventi manuali o logica applicativa aggiuntiva.
- **items(category) → categories(name):** Analogamente, l'eliminazione di una categoria comporta la rimozione di tutti gli item che vi fanno riferimento. Questa scelta è stata motivata dalla volontà di mantenere il database pulito ed evitare inconsistenze, specialmente in contesti in cui le categorie possono essere modificate o rimosse dinamicamente dall'utente.

4 Caratteristiche principali

4.1 Gestione delle spese

La funzionalità di gestione delle spese rappresenta il cuore dell'applicazione.

4.1.1 Registrazione delle spese

Il processo di registrazione di una nuova spesa è stato ottimizzato per minimizzare il tempo necessario:

- **Calcolo automatico:** Totale della spesa calcolato dinamicamente durante l'inserimento dei prodotti , dopo averli marcati come acquistati.
- **Gestione delle categorie personalizzate:** Possibilità per l'utente di creare, modificare o eliminare categorie, mantenendo il sistema flessibile e personalizzabile.
- **Storico degli acquisti:** Archivio accessibile delle spese precedenti, utile per statistiche e analisi delle abitudini di consumo.
- **Validazione dei dati in input:** Controlli per prevenire l'inserimento di valori non validi (es. prezzi negativi, campi obbligatori mancanti).

4.1.2 Visualizzazione e modifica

Le spese registrate possono essere:

- **Visualizzate in dettaglio:** Accesso a tutte le informazioni su prodotti, prezzi, categorie e note.
- **Modificate:** Correzione di errori o aggiunta di prodotti dimenticati.
- **Eliminate:** Rimozione completa con conferma per evitare cancellazioni accidentali.

4.2 Liste della spesa

Le liste della spesa in Groceroo sono progettate per essere strumenti pratici e interattivi.

4.2.1 Creazione e gestione

- **UX diretta:** Possibilità di creare una nuova lista o eliminarla direttamente dalla HomeScreen, tramite l'uso di componenti swipeable che rendono l'interazione rapida e intuitiva.
- **Modifica del titolo:** È possibile rinominare una lista della spesa in qualsiasi momento, consentendo una gestione più ordinata e personalizzata.

4.3 Categorie personalizzabili

Il sistema di categorizzazione è stato progettato per essere flessibile e adattabile alle esigenze individuali:

- **Categorie predefinite:** L'app include un set iniziale di categorie comuni , che l'utente non può modificare.
- **Personalizzazione completa:** Possibilità di creare ed eliminare categorie.

4.4 Ricerca avanzata

La funzionalità di ricerca è stata progettata per consentire agli utenti di trovare rapidamente informazioni specifiche:

- **Filtri multipli:** Combinazione di diversi criteri di ricerca (data, categoria, fascia di prezzo).
- **Ricerca contestuale:** Adattamento dell'interfaccia di ricerca in base al prodotto cercato .

4.5 Analisi e statistiche

Il modulo di analisi rappresenta uno degli elementi distintivi di Groceroo, offrendo visualizzazioni intuitive e informative:

4.5.1 Dashboard principale

La dashboard di analisi include:

- **Spesa totale:** Visualizzazione dell'importo totale speso nel periodo selezionato.
- **Media giornaliera:** Calcolo della spesa media giornaliera.
- **Distribuzione per categoria:** Grafico a torta che mostra la percentuale di spesa per ciascuna categoria.

4.5.2 Analisi temporale

- **Andamento spese:** Grafico a linee che mostra l'evoluzione della spesa nel tempo.

5 Caratteristiche tecniche avanzate

5.1 Ottimizzazione delle prestazioni

Per garantire un'esperienza utente fluida anche con grandi quantità di dati, abbiamo implementato diverse strategie di ottimizzazione:

- **Virtualizzazione liste:** Utilizzo di `FlatList` con renderizzazione lazy per gestire efficacemente lunghi elenchi di prodotti o spese.
- **Query ottimizzate:** Progettazione attenta delle query SQL per minimizzare i tempi di risposta del database.

6 Sfide affrontate e soluzioni

Durante lo sviluppo di Groceroo, il team ha dovuto affrontare diverse sfide tecniche e implementative:

6.1 Sincronizzazione tra componenti dell'interfaccia

Problema: Mantenere sincronizzati diversi componenti dell'interfaccia che visualizzano gli stessi dati (ad esempio, un prodotto modificato doveva aggiornarsi in tutte le viste).

Soluzione: Il problema è stato risolto utilizzando `useFocusEffect` di React Navigation, che consente di eseguire codice ogni volta che una schermata torna in primo piano. Questo approccio permette di ricaricare i dati quando necessario, assicurando che le informazioni visualizzate siano sempre aggiornate e coerenti tra le diverse viste, senza dover ricorrere a uno state manager globale.

In fase iniziale è stato valutato anche l'uso di Redux. Tuttavia, la soluzione con `useFocusEffect` si è dimostrata più adatta al contesto del progetto, semplificando l'implementazione e riducendo la complessità del codice.

Questa scelta ha migliorato l'esperienza utente e la manutenibilità del codice, pur presentando dei limiti in scenari con aggiornamenti in tempo reale.

6.2 Ottimizzazione delle performance nella visualizzazione di liste

Problema: Durante lo sviluppo della schermata con la lista dei prodotti, è emerso un problema di performance: lo scroll risultava poco fluido, soprattutto quando il numero di elementi aumentava. Questo causava una percezione negativa dell'interfaccia da parte dell'utente.

Soluzione: Per risolvere il problema, è stata utilizzata la componente `FlatList` di React Native, che supporta nativamente la renderizzazione lazy degli elementi. È stata ottimizzata la proprietà `initialNumToRender`, ed è stato definito un `keyExtractor` stabile per migliorare il tracciamento degli elementi.

Risultato: Lo scroll è diventato molto più fluido, anche con dataset di dimensioni medio-grandi, e il tempo di caricamento iniziale della schermata si è ridotto sensibilmente. Questa ottimizzazione ha migliorato significativamente l'esperienza utente.