

Programmazione di Dispositivi mobili

- MoneyMind -

Bolzoni Michele

matricola 829834

Coldani Andrea

matricola 894684

Enobo Franck

matricola 899857

A.A. 2024/2025

12 febbraio 2025



Indice

1	Introduzione	3
2	Strumenti per lo sviluppo	3
2.1	Strumenti Tecnici	3
2.2	Strumenti di Progettazione della UI	3
3	API	4
3.1	Utilizzi Pratici dell'API	4
3.2	Disponibilità e Limitazioni	4
4	Architettura	5
4.1	UI Layer	6
4.1.1	Activity	6
4.1.2	Fragment e ViewModel	6
4.2	Data Layer	6
4.2.1	Repository	6
4.3	Sync	7
4.4	Pattern Utilizzati	7
5	Funzionalità	8
5.1	Login	8
5.2	Registrazione	9
5.3	Home	10
5.4	Movimenti	11
5.5	Budget	12
5.6	Obbiettivi	14
5.7	Impostazioni	16
6	Sviluppi Futuri	17

1 Introduzione

Il progetto si propone di sviluppare un'applicazione mobile per Android, progettata per fornire un supporto completo nella gestione delle **finanze personali**.

L'app offre una serie di funzionalità pensate per rispondere a tutte le esigenze di un'applicazione per dispositivi mobili tra cui il principio offline-first, l'adattabilità a più "forme" di dispositivo e flessibilità nel cambio lingua.

Tra le principali funzionalità offerte, l'utente potrà:

- Aggiungere **entrate** e **spese** in modo semplice e veloce, con la possibilità di categorizzare ogni transazione.
- Impostare un **budget** per ciascuna categoria, in modo da avere sempre sotto controllo le spese e rispettare i limiti di spesa prefissati.
- Definire **obiettivi** di risparmio per ogni categoria, aiutando l'utente a pianificare e raggiungere i propri traguardi finanziari.
- Cambiare **valuta** facilmente, consentendo all'utente di gestire le proprie finanze in diverse valute, rendendo l'app versatile per chi viaggia o gestisce entrate in valuta estera.

Inoltre, l'applicazione permette la creazione di un account personale, in modo che ogni utente possa salvare e sincronizzare i propri dati, accedere ai servizi personalizzati e tenere traccia delle proprie abitudini finanziarie in modo sicuro e protetto.

2 Strumenti per lo sviluppo

2.1 Strumenti Tecnici

- **Android Studio:** utilizzato come IDE
- **GitHub:** utilizzato per la gestione del progetto in modalità collaborativa. La nostra repository è strutturata come segue:
 - **Main branch:** il ramo principale in cui vengono effettuati i push delle versioni stabili e funzionanti dell'applicazione.
 - **Develop branch:** il ramo dedicato all'integrazione delle modifiche provenienti dai vari sviluppatori. È il punto di riferimento per le nuove funzionalità in fase di sviluppo. Questo nodo permette all'intero gruppo di rimanere allineato senza modificare i singoli branch.
 - **Michele/andrea/franck-develop branch:** un ramo separato per ciascun sviluppatore. Ogni volta che si implementava una nuova funzionalità, veniva eseguito il merge con il ramo michele/andrea/franck-develop.
- **Firebase/Firestore:** servizi di Google utilizzati in questa applicazione per gestire i login/registrazioni e come database remoto. Utili soprattutto per la sincronizzazione tra dispositivi.
- **Librerie varie:** per esempio **Retrofit** per interagire con l'API, **Room** per implementare il database locale.
- **ECB API:** servizio che fornisce i tassi di cambio tra valute. Viene sempre restituito il cambio con l'euro, poiché è la valuta predefinita. Maggiori dettagli nel paragrafo 3.

2.2 Strumenti di Progettazione della UI

Lo strumento utilizzato per progettare la parte grafica dell'applicazione è **Figma**. Questo strumento permette in maniera agile di aggiungere modificare elementi grafici facilmente replicabili secondo le linee guida **MaterialDesign3**.

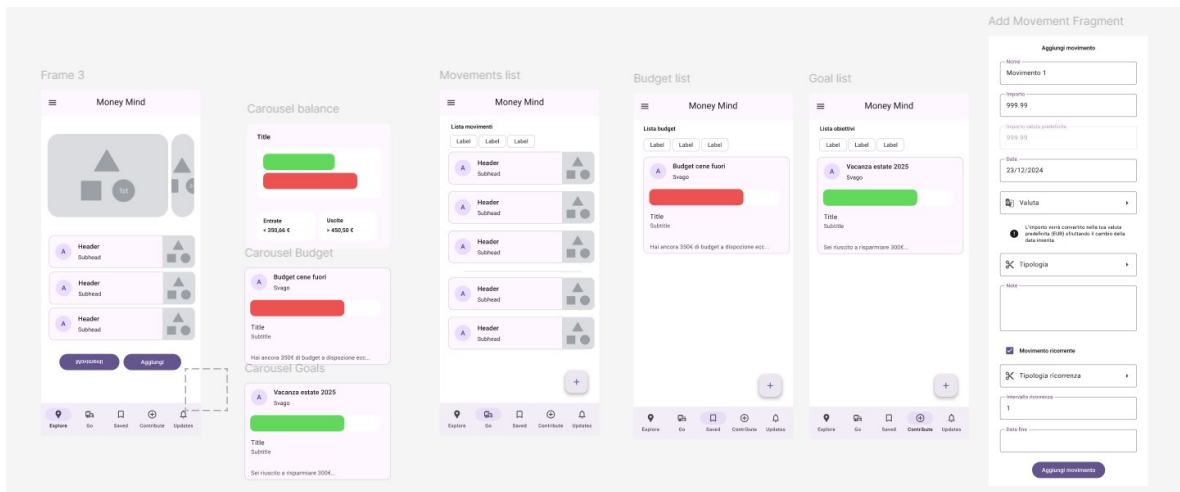


Figura 1: Bozza dell'UI in fase di progettazione

3 API

L'API utilizzata in questa applicazione ha lo scopo di recuperare i **tassi di cambio** delle valute rispetto all'euro, che rappresenta la valuta predefinita dell'applicazione.

L'endpoint utilizzato per poter ottenere i dati è il seguente:

"https://data-api.ecb.europa.eu/service/data/EXR/D..EUR.SP00.A"

Per poter specificare i giorni dei quali dobbiamo ottenere i cambi vengono usati i **parametri** di `startPeriod`, `endPeriod` e `format` da restituire:

"?startPeriod=YYYY-MM-dd&endPeriod=YYYY-MM-dd&format=jsonData".

Per poter **interagire** in modo efficiente e agile con l'API, è stata utilizzata la libreria **Retrofit** per la gestione delle comunicazioni HTTP.

Retrofit semplifica notevolmente il processo di invio di richieste e gestione delle risposte, rendendo il codice molto più **leggibile** e **manutenibile** rispetto all'utilizzo diretto delle classi di basso livello.

Dopo l'interrogazione l'API restituirà un **file JSON** che viene interpretato da una classe apposita.

3.1 Utilizzi Pratici dell'API

- L'utente può **registrare entrate** o **spese** in valute diverse dall'euro, anche con una data **anteriore** a quella odierna.
- L'API fornisce i tassi di cambio aggiornati per ben **32 valute** diverse. L'utente avrà la possibilità di selezionare facilmente la valuta desiderata tramite un menù a tendina.

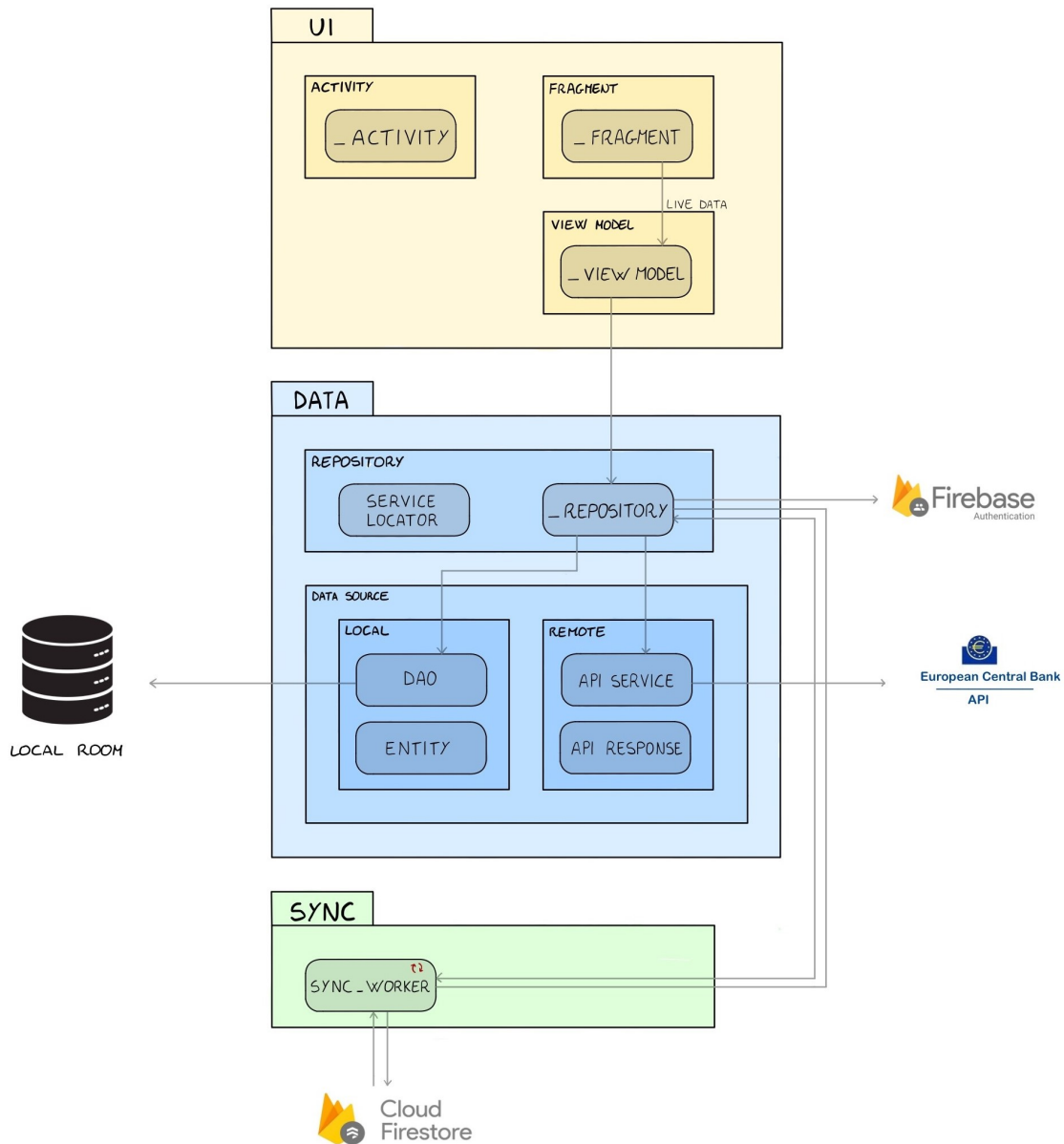
3.2 Disponibilità e Limitazioni

- I dati dei tassi di cambio sono disponibili a partire dal **1999** fino ad **oggi**.
- L'**aggiornamento** dei tassi avviene giornalmente alle **ore 16:00**.
- Poiché i dati relativi a **sabato**, **domenica** e **festivi** non sono disponibili, l'applicazione utilizza i tassi di cambio dell'ultimo giorno **feriale disponibile**.

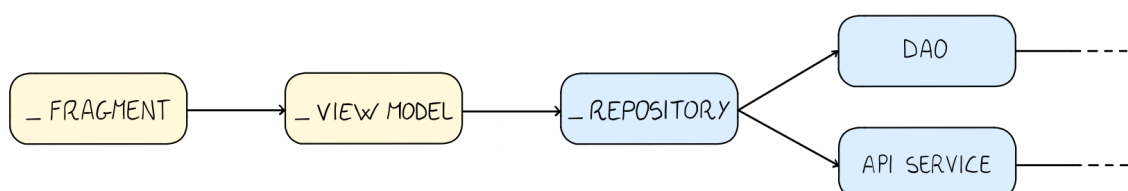
Questa gestione garantisce che l'utente possa ottenere dati accurati e aggiornati per registrare transazioni in diverse valute nel tempo.

4 Architettura

Il **pattern architetturale** al quale si riferisce l'applicazione è il MVVM (Model-View-ViewModel) che mira a separare la logica di **presentazione** (UI) dalla logica di **business**. L'**architettura** dell'applicazione è la seguente:



Il **flusso** generale per il quale i **dati** vengono ottenuti è:



4.1 UI Layer

Questo layer si occupa della gestione dei dati provenienti dal **Data Layer**.

Durante lo sviluppo dell'applicazione, è stata mantenuta una chiara separazione tra la **UI** e il **Data Layer**, garantendo che la UI **non** si occupi della **gestione** dei **dati**, ma si limiti esclusivamente alla loro **visualizzazione**. In particolare:

4.1.1 Activity

Sono 2 per garantire il "Single activity - multiple fragments": **MainActivity** che gestisce il login/registrazione e **MainNavigationActivity** che gestisce tutto il resto della UI.

4.1.2 Fragment e ViewModel

Comunicano principalmente tramite **LiveData**, che, utilizzando il pattern **Observer**, consente loro di scambiarsi dati in modo reattivo e senza **accoppiamento diretto**.

Ogni Fragment è associato a un ViewModel che gestisce i dati correlati, garantendo una separazione chiara tra la logica di gestione dei dati e la UI.

I Fragment e ViewModel utilizzati nell'applicazione sono:

- AddBudgetFragment e AddBudgetViewModel
- BudgetFragment e BudgetViewModel
- AddGoalFragment e AddGoalViewModel
- GoalFragment e GoalViewModel
- AddTransactionFragment e AddTransactionViewModel
- TransactionFragment e TransactionViewModel
- HomeFragment e HomeViewModel
- SettingFragment e SettingViewModel
- MainActivityViewModel: serve all'avvio dell'app per verificare se l'utente è già **loggato** in modo da **sincronizzare** i suoi dati e determinare quale **schermata** mostrare successivamente.

4.2 Data Layer

Innanzitutto, si distingue tra due tipi di classi: quelle responsabili di reperire i dati dalla fonte corretta, ovvero i **Repository**, e quelle che si interfacciano direttamente con le risorse dei dati, ossia le classi appartenenti al **Data Source**.

Il sottopackage Data Source contiene a sua volta:

- **Local:** contiene le classi per poter interagire con Room che modellano le entità e che definiscono le **Query**.
- **Remote:** contiene le classi necessarie per interagire con l'**API**.

4.2.1 Repository

I **Repository** sono responsabili di reperire i dati dalla corretta **Data Source** e sono i seguenti:

- BudgetRepository
- CategoryRepository
- GoalRepository

- ExchangeRepository
- TransactionRepository
- RecurringTransactionRepository
- UserRepository
- GenericRepository

Per poter favorire l'approccio **Offline-First** i Repository reperiscono i dati dal **Database locale** (Room). Successivamente questi dati verranno sincronizzati sul **Database remoto** (si veda paragrafo 4.3).

Unica **eccezione** viene fatta per il **Login/Registrazione**: il repository in questo caso comunica direttamente con **Firebase Authentication** sia **online** che **offline**, a condizione che l'utente abbia effettuato l'accesso almeno una volta.

4.3 Sync

Questo componente è essenziale per garantire la **persistenza** dei dati dell'account dell'utente, indipendentemente dal dispositivo utilizzato. In particolare, il **Sync_Worker** si occupa della sincronizzazione dei dati tra il **database locale** (Room) e **Firestore**, trasferendo dati in entrambe le direzioni.

Di default, questo processo viene eseguito ogni **30 minuti**, ad **ogni avvio** dell'app o ad ogni **login**, anche se Android potrebbe modificarne le tempistiche in base alla gestione delle risorse di sistema.

Ogni Repository memorizza nelle **Shared Preferences** un valore **TIMESTAMP**, che indica l'ultima sincronizzazione effettuata.

In particolare il funzionamento avviene come segue:

1. **Sincronizzazione** delle risorse locali su **Firestore**: viene fatta una query dove **SYNC = false** e successivamente i risultati vengono aggiunti a **Firestore**.
2. **Download** dei dati da **Firestore** non ancora sincronizzati: avviene scaricando i dati e determinando la data massima tra le transazioni già sincronizzate su **Firestore**. Successivamente, confrontiamo questa data con il **timestamp** attuale. Utilizziamo la data più recente tra le due e recuperiamo da **Firestore** tutti i record con un valore di **last_sync** maggiore della data più recente, garantendo così che vengano sincronizzati solo i dati aggiornati.
3. **Aggiornamento del timestamp**: una volta completata la sincronizzazione, il valore **timestamp** delle **shared preferences** viene aggiornato.

4.4 Pattern Utilizzati

A livello strutturale, per garantire un codice pulito ed efficiente e perseguire il **Low Coupling** tra le classi, sono stati adottati principalmente i seguenti **Pattern**:

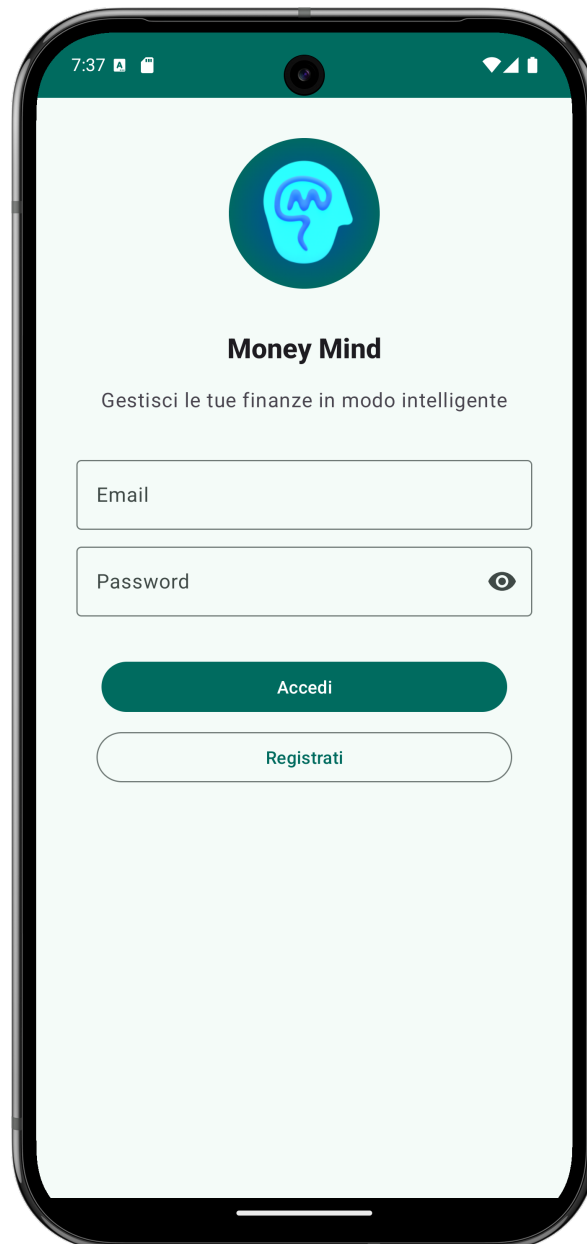
- **Observer**: per permettere ai fragment di osservare i **LiveData**
- **Service Locator**: permette di delegare la **responsabilità** dell'istanza dei **Repository** a un componente centrale, evitando che siano le singole classi a doverla gestire
- **Adapter**: permette di **adattare interfacce incompatibili** tra loro in modo che possano lavorare insieme. È stata utilizzata per esempio nel **carousel** della home e nei **menù a tendina**.
- **Singleton**: garantisce che una classe abbia **una sola istanza** e fornisca un punto di accesso globale a quella istanza. Per esempio è stato utilizzato per istanziare il **Retrofit Client**
- **Factory**: utilizzato per istanziare i **ViewModel** che richiedevano un **Adapter** specifico. In questo modo, la Factory centralizza l'istanza del **ViewModel**, **gestendo** la **dipendenza** dell'adapter e migliorando la modularità e la manutenibilità del codice
- **Helper**: classi che contengono solo metodi **statici** e servono a fornire funzionalità di **supporto** per un dominio specifico senza la necessità di creare istanze. Tali classi sono state inserite nel package **"Utils"**

- **Static Utility Class:** classe con metodi **statici** che fornisce funzionalità di **supporto generale** all'applicazione, rendendo disponibili strumenti **riutilizzabili** in più parti del codice. Include operazioni comuni, come la **conversione** di **date** o la creazione di **SnackBar**, evitando la duplicazione del codice

5 Funzionalità

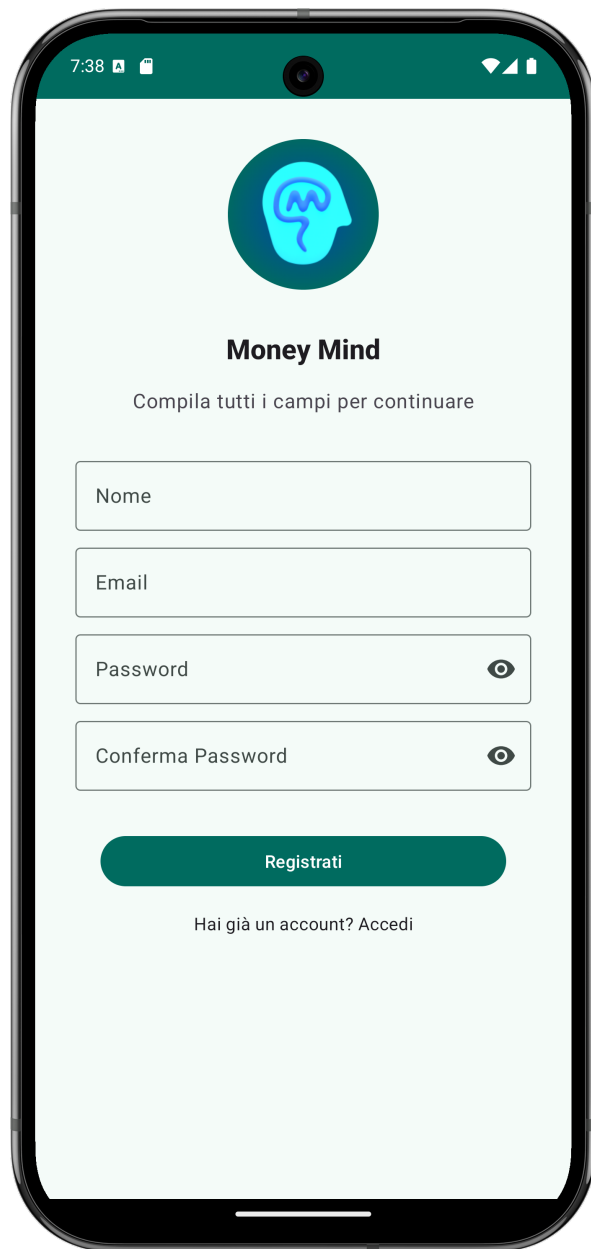
5.1 Login

L'utente può accedere al proprio account inserendo **email** e **password**. Se non è ancora registrato, ha la possibilità di passare alla **schermata di registrazione** per creare un nuovo account. Di seguito, la schermata di **login**:



5.2 Registrazione

L'applicazione gestisce la registrazione degli utenti richiedendo un **nome**, un'**email** e una **password**. L'**email** deve essere univoca, e questo viene garantito da **Firebase**, che impedisce la creazione di un nuovo account se l'indirizzo è già associato a un altro utente. Inoltre, la **password** deve avere una lunghezza minima di 6 caratteri per garantire un livello base di sicurezza. Di seguito, la schermata di **registrazione**:



The image shows a smartphone screen with a registration form for an app called "Money Mind". The status bar at the top shows the time 7:38 and various icons. The app's logo, a blue circle with a white brain icon, is at the top center. Below the logo is the app name "Money Mind" in bold. A subtitle "Compila tutti i campi per continuare" (Fill out all fields to continue) is centered. The form consists of four input fields: "Nome" (Name), "Email", "Password", and "Conferma Password" (Confirm Password). The "Password" and "Conferma Password" fields have an eye icon to toggle visibility. Below the fields is a green "Registrati" (Register) button. At the bottom, there is a link "Hai già un account? Accedi" (Already have an account? Log in).

7:38

Money Mind

Compila tutti i campi per continuare

Nome

Email

Password

Conferma Password

Registrati

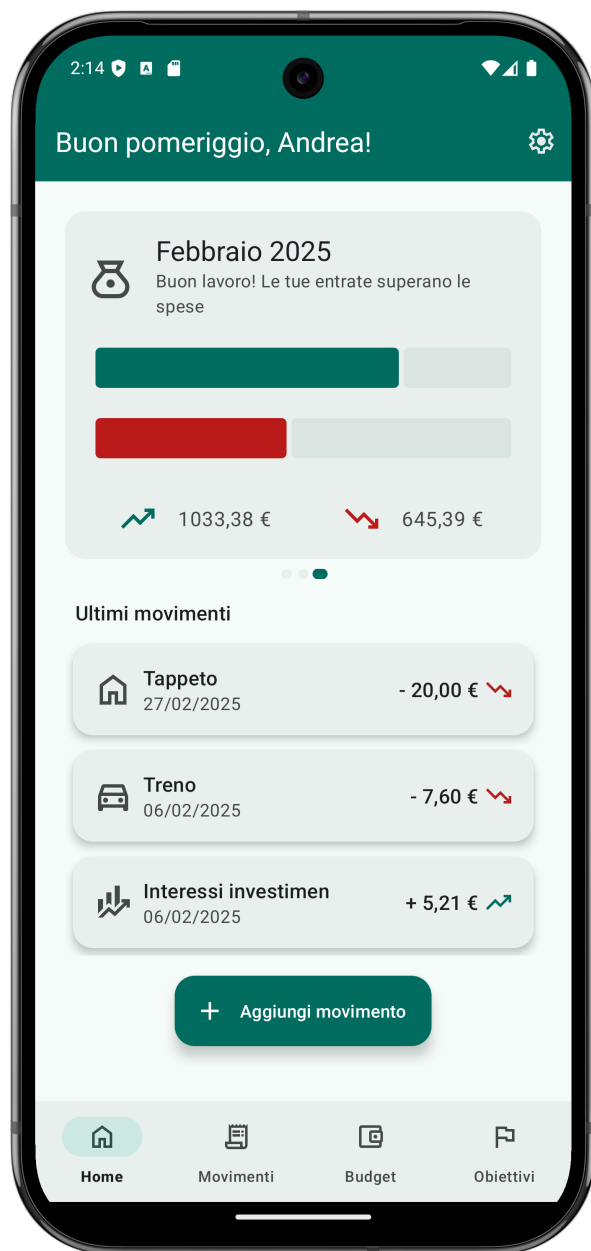
Hai già un account? Accedi

5.3 Home

La Home offre una panoramica delle **spese** e delle **entrate** del mese corrente, accompagnata da una rappresentazione **grafica**. Inoltre, un **carosello** interattivo consente di scorrere e visualizzare anche i bilanci dei tre mesi precedenti.

Più sotto, la schermata mostra gli **ultimi tre movimenti** effettuati, con la possibilità di aggiungerne uno nuovo tramite un apposito bottone.

Una **naigation bar** permette di navigare tra le varie schermate. Di seguito, la schermata **home**:



5.4 Movimenti

Questa schermata consente di visualizzare tutti i **movimenti** effettuati dall'utente, suddivisi in due sezioni: "Movimenti" e "Movimenti ricorrenti". I movimenti ricorrenti possono essere inseriti dall'utente per spese con **cadenza fissa** (giornaliera, settimanale, mensile o annuale) e vengono verificati ed eventualmente contabilizzati da un **worker** ogni 4 ore, all'apertura dell'app o quando ne viene registrato uno nuovo. Anche in questo caso viene effettuata l'eventuale **conversione** di valuta.

Inoltre, da questa schermata è possibile accedere facilmente alla schermata di **aggiunta movimento**. Nella schermata di **aggiungi movimento** è possibile **convertire** movimenti e visualizzarne il valore nella valuta predefinita (Euro). Di seguito, le schermate **Movimenti** e **Aggiungi movimento**:

2:44

← Aggiungi Movimento SALVA

Nome

Importo

Importo Convertito
0.00

Data
12/02/2025

Valuta
EUR - Euro

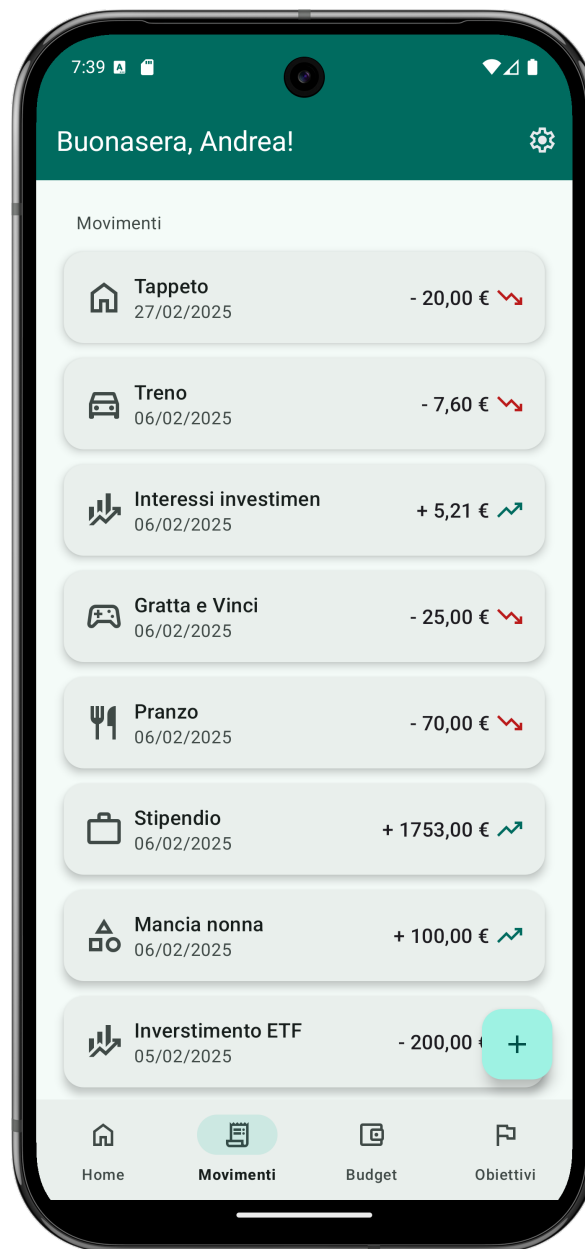
i L'importo verrà convertito nella tua valuta predefinita (EUR) sfruttando il cambio alla data inserita

Tipologia
Entrata

Categoria



Note

☐ Movimento ricorrente



5.5 Budget

Analogamente alla schermata **movimenti** vengono mostrati i **budget** aggiunti dall'utente dando la possibilità, tramite un'altra schermata associata, di **aggiungerne** di nuovi. Lo stato di avanzamento di ciascun budget è accompagnato da una **rappresentazione grafica** che ne facilita la visualizzazione. Di seguito, le schermate **Budget** e **Aggiungi budget**:

7:40   

← Aggiungi Budget SALVA

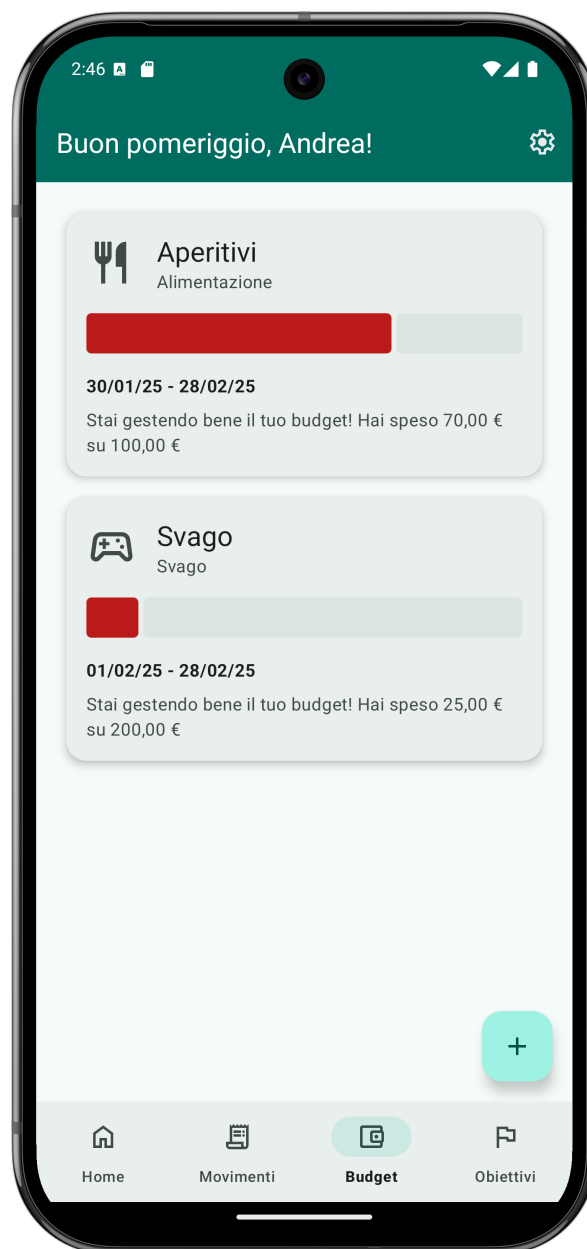
Nome Budget

Importo

Categoria ▼

Data di Inizio

Data di Fine



5.6 Obiettivi

Esattamente come nelle altre due sezioni precedenti vi sono due schermate associate dedicate agli **obiettivi** aggiunti dall'utente. Di seguito, le schermate **Obiettivi** e **Aggiungi obiettivo**:

7:40   

← Aggiungi Obiettivo SALVA

Nome Goal

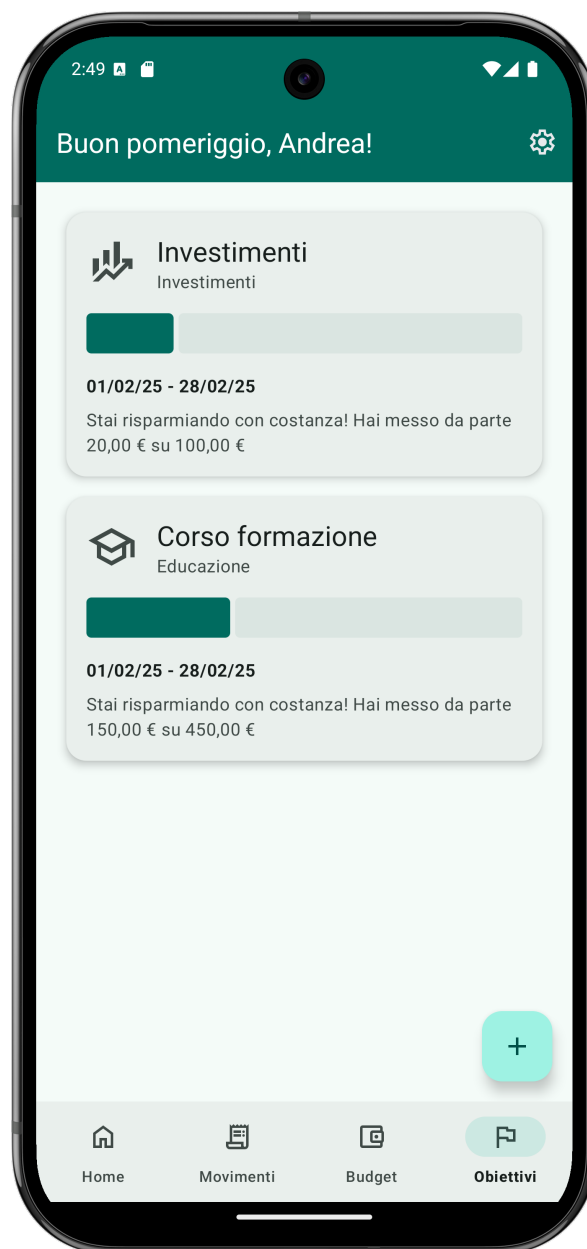
Importo Obiettivo

Importo Risparmiato

Categoria ▼

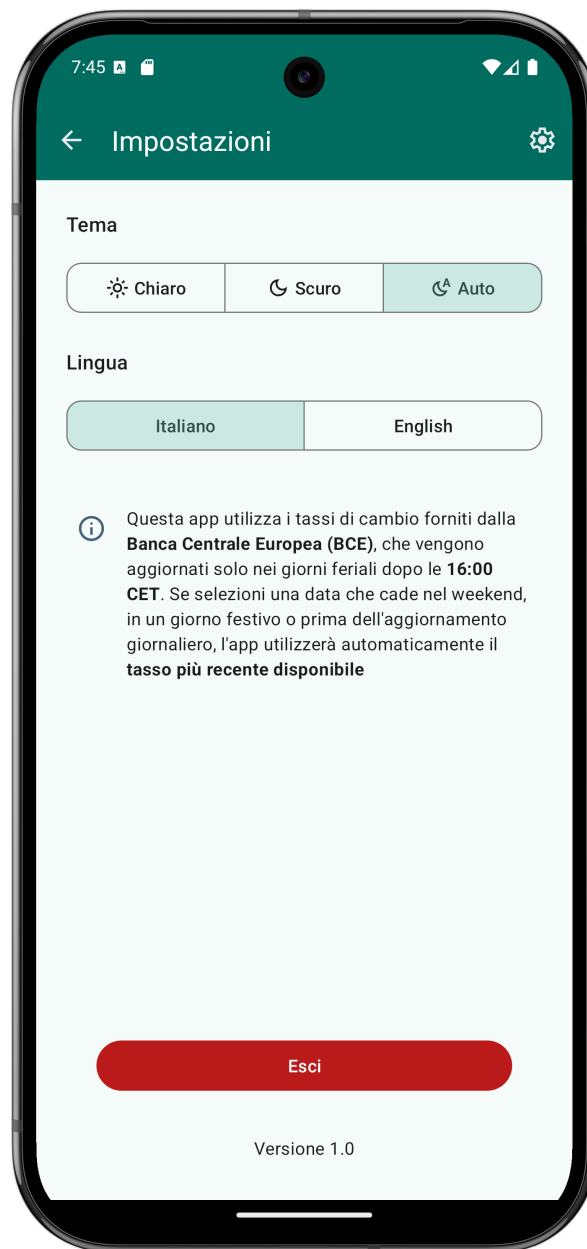
Data di Inizio

Data di Fine



5.7 Impostazioni

Nella sezione **impostazioni** accessibile tramite la home è possibile settare il **tema** (Chiaro/Scuro) e le **lingue** disponibili (Italiano/Inglese). Inoltre è possibile effettuare il **Logout**. Di seguito, la schermata **Impostazioni**:



6 Sviluppi Futuri

Durante lo sviluppo dell'applicazione, abbiamo dovuto fare delle scelte strategiche per rispettare le tempistiche del progetto assicurando al contempo una versione **stabile** e **funzionante**.

Tuttavia, alcune funzionalità aggiuntive che avrebbero **migliorato** ulteriormente l'esperienza utente non sono state implementate a causa della loro **complessità tecnica** e quindi del tempo limitato per svilupparle.

Ecco le principali caratteristiche che avremmo voluto aggiungere:

- **Sistema di Notifiche:** avvisi personalizzati per soglie di spesa superate o promemoria per risparmi programmati.
- **Login con Account Google:** dare la possibilità ad un utente non ancora registrato di poter effettuare il login con le sue credenziali di Google

- **Recupero Password:** Un sistema di reset della password tramite email, con l'invio di un link di ripristino.
- **Possibilità di Cambiare la Valuta Predefinita:** Attualmente, la valuta di base è fissa, ma sarebbe stato utile permettere agli utenti di selezionare una valuta predefinita.
- **Pagina di Conversione Valuta:** un'interfaccia dedicata per convertire rapidamente importi tra diverse valute utilizzando l'API della Banca Centrale Europea.
- **Grafici e Analisi Dettagliate:** implementazione di report visivi per monitorare l'andamento delle spese nel tempo.
- **Lingue applicazione:** traduzione dell'applicazione in altre lingue oltre all'italiano ed all'inglese