
Moes
Documentation
Smart Sports Navigation & Tracking

INDICE DEI CONTENUTI

Il progetto Moes è un'applicazione di fitness tracker gamificata, scritta in Kotlin e Jetpack Compose, con una forte componente di geolocalizzazione (Mapbox) e sincronizzazione cloud (Firebase).

1. Introduzione e Concept

Moes è un'applicazione mobile nativa Android progettata per trasformare l'attività fisica all'aperto in un'esperienza coinvolgente, sociale e gratificante. Non si limita a essere un semplice fitness tracker: Moes combina la precisione del tracciamento GPS con meccaniche di gamification avanzate, incentivando l'utente a superare i propri limiti attraverso missioni, livelli e competizioni amichevoli.

1.1 Il Nome e l'Identità

Il nome "Moes" affonda le sue radici nella tradizione locale: è un termine del dialetto bergamasco che significa letteralmente "Muoviti". L'espressione viene spesso utilizzata anche in senso figurato come esortazione energica a "darsi una svegliata" o a reagire all'inerzia. Questa scelta non è casuale, ma racchiude la filosofia stessa del progetto: un invito diretto, quasi imperativo ma amichevole, ad abbandonare la sedentarietà e mettersi in azione.

L'identità visiva dell'applicazione riflette questa energia attraverso una palette cromatica vibrante, dominata da tonalità intense di arancione e giallo.

1.2 Visione e Obiettivi

L'obiettivo di Moes è abbattere la monotonia dell'allenamento solitario. Mentre le app tradizionali si concentrano esclusivamente sulle metriche (tempo, distanza, passo), Moes pone l'accento sulla **costanza** e sul **divertimento**:

- **Gamification:** L'attività fisica viene tradotta in progressi tangibili. Ogni chilometro percorso contribuisce al completamento di "Missioni" (es. Costanza, Macinatore di Km, Inarrestabile) che sbloccano nuovi livelli di prestigio per l'utente.
- **Socialità:** L'app permette di costruire una community, confrontare le proprie "Streak" (giorni consecutivi di attività) con quelle degli amici e inviare richieste di amicizia per stimolare una sana competizione.

1.3 Target di Riferimento

Moes è pensata per un pubblico eterogeneo di appassionati di attività outdoor:

- **Runner e Jogger:** Che cercano di migliorare il proprio passo e mantenere la motivazione alta.
- **Camminatori:** Che vogliono tenere traccia delle proprie escursioni e raggiungere obiettivi di salute quotidiani.
- **Ciclisti:** Interessati a mappare i propri percorsi e monitorare le distanze percorse.
- **Utenti Occasionali:** Che hanno bisogno di quello "stimolo in più" ("Moes!") per iniziare a muoversi con regolarità.

2. Funzionalità Principali

L'esperienza utente di Moes è organizzata in un flusso di navigazione intuitivo, progettato per accompagnare l'atleta dalla pianificazione dell'attività fino all'analisi dei risultati e alla condivisione sociale. Di seguito vengono descritte le schermate principali che compongono l'applicazione.

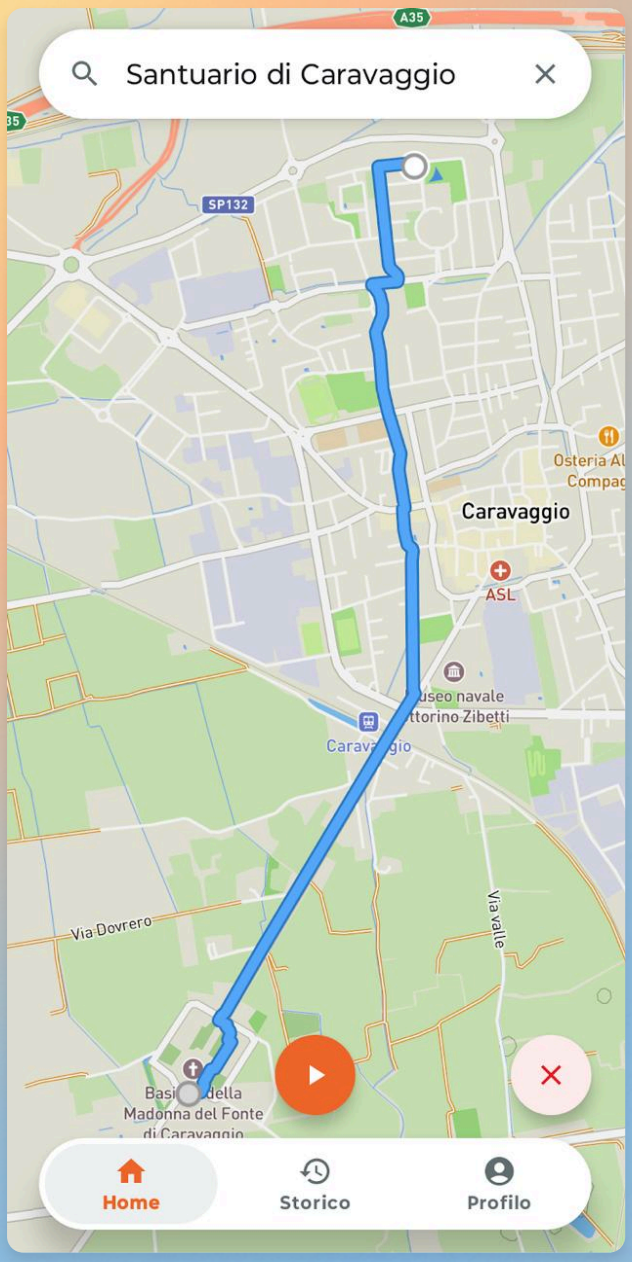
2.1 Home Screen

La Home è il centro operativo dell'applicazione. In questa schermata, l'utente visualizza la propria posizione in tempo reale su una mappa interattiva fornita da Mapbox e può scegliere tra due diverse modalità di allenamento.

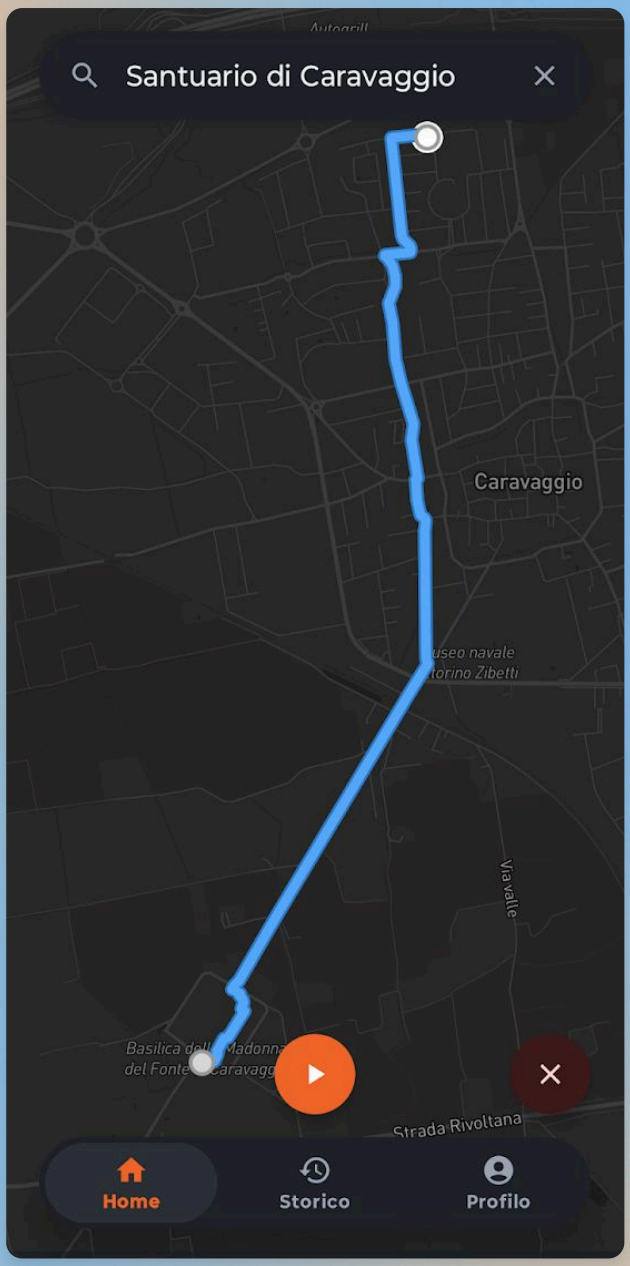
- **Modalità Allenamento Libero:** Pensata per chi vuole semplicemente correre o camminare senza una meta precisa. L'utente preme direttamente il pulsante "Start" e l'app inizia a tracciare il percorso, monitorando le statistiche in background senza fornire indicazioni di svolta.
- **Modalità con Navigazione (Ricerca):** Ideale per raggiungere una destinazione specifica o esplorare nuovi percorsi. La destinazione può essere impostata in due modi:
  - **Barra di Ricerca:** Digitando l'indirizzo o il nome del luogo nella SearchBar in alto.
  - **Long Press su Mappa:** Tenendo premuto un punto qualsiasi sulla mappa interattiva per impostarlo come destinazione.

In entrambi i casi, l'app calcola il percorso ottimale (visualizzato come una linea sulla mappa) e, una volta avviato l'allenamento, fornisce indicazioni turn-by-turn tramite un banner di istruzioni.

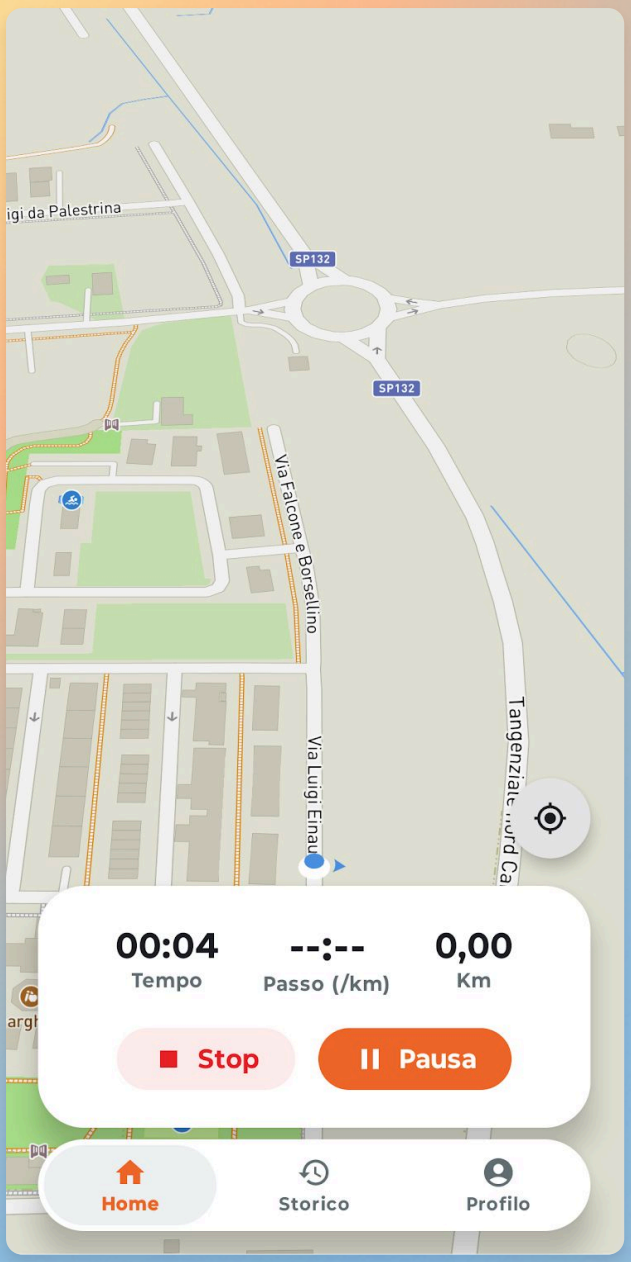
Fase di Tracking: Una volta avviata l'attività (in qualsiasi modalità), l'interfaccia cambia focus. Compare un overlay informativo che mostra le metriche essenziali in tempo reale (cronometro, distanza, passo). Da qui è possibile mettere in pausa, riprendere o terminare la sessione.



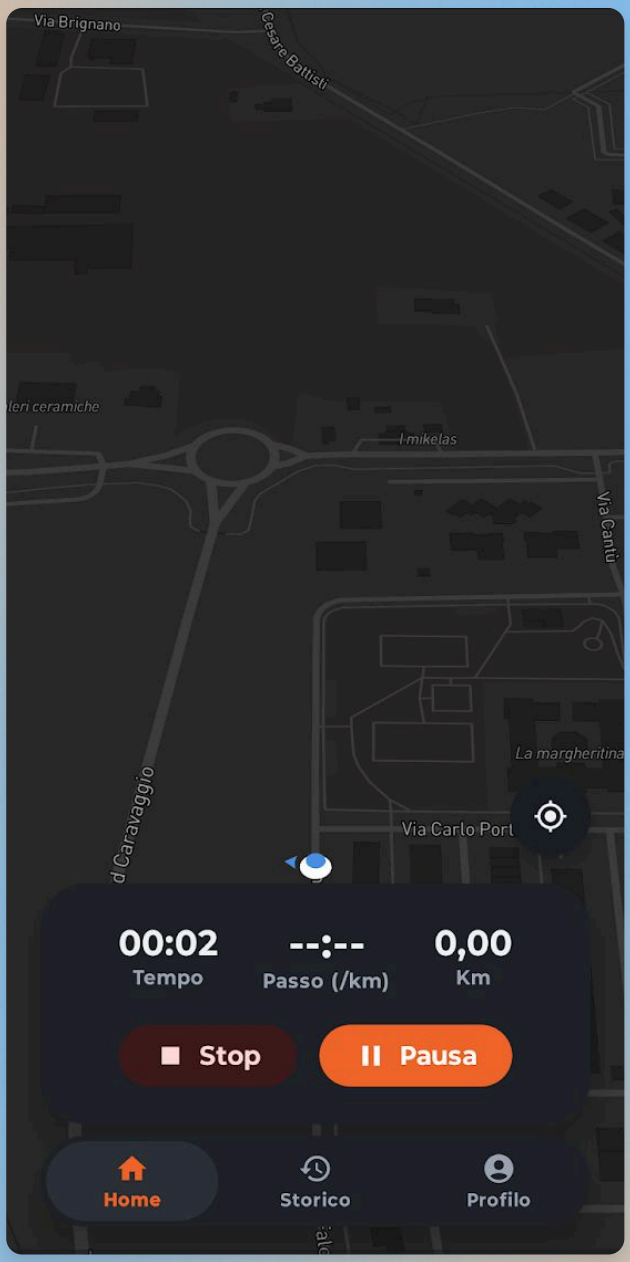
Navigazione (Light)



Navigazione (Dark)



Training Overlay (Light)

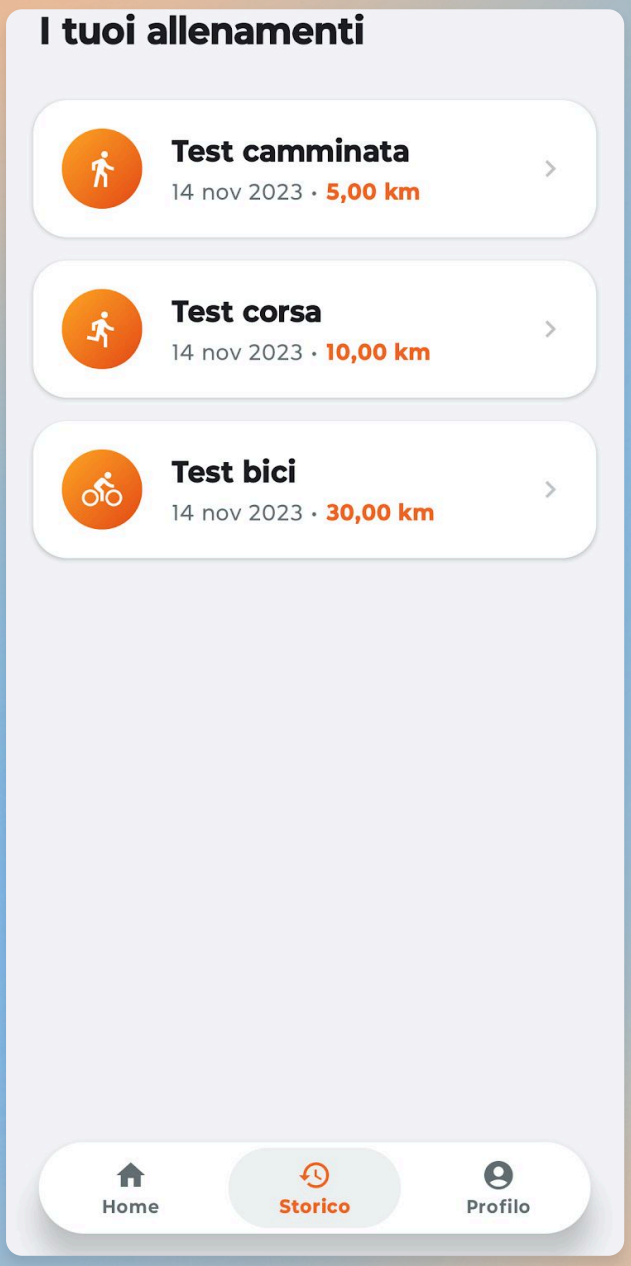


Training Overlay (Dark)

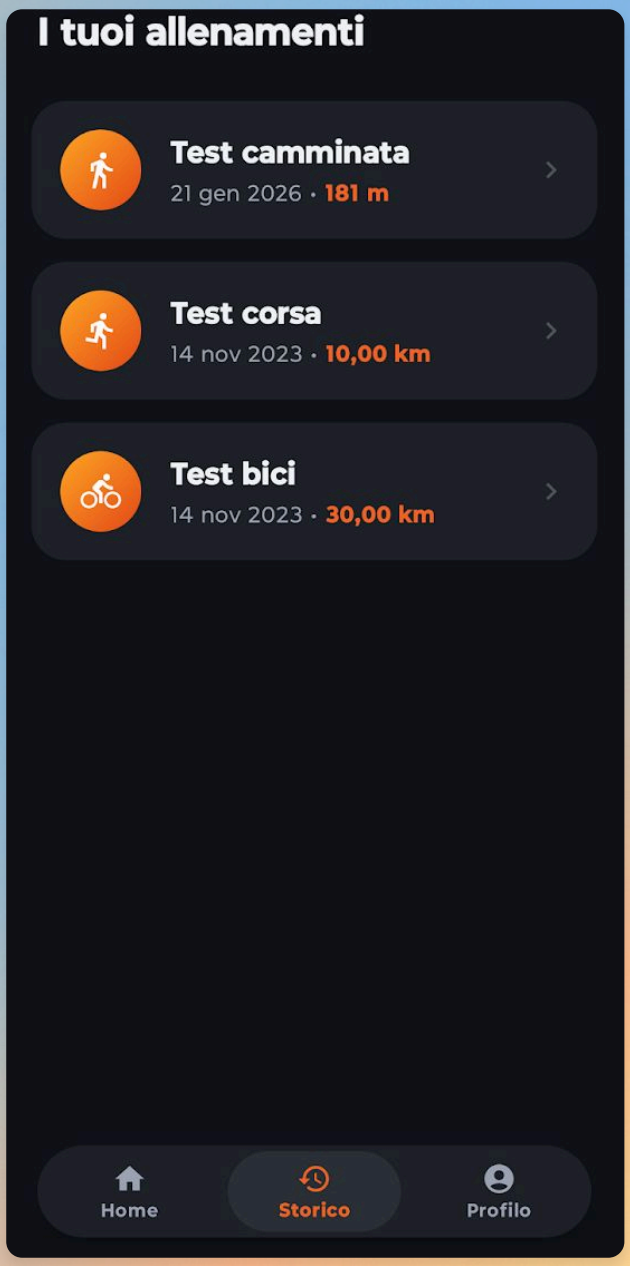
2.2 Schermata Sessioni

Questa sezione funge da registro storico di tutte le attività svolte. Le sessioni sono presentate in una lista a scorrimento verticale, ordinate dalla più recente alla meno recente.

Ogni elemento della lista è una scheda riassuntiva ("Pill Card") che offre un colpo d'occhio immediato sull'allenamento: titolo (o data se non modificato), distanza totale e un'icona identificativa che classifica automaticamente l'attività (camminata, corsa o ciclismo) in base alla velocità media registrata.



Sessioni (Light)

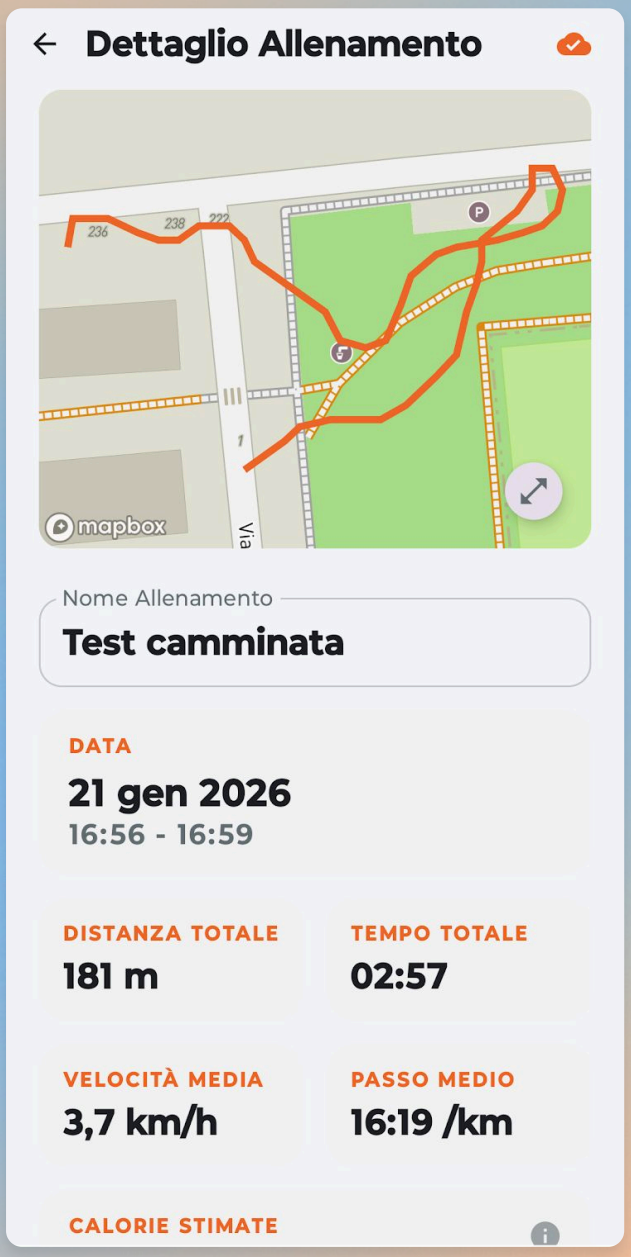


Sessioni (Dark)

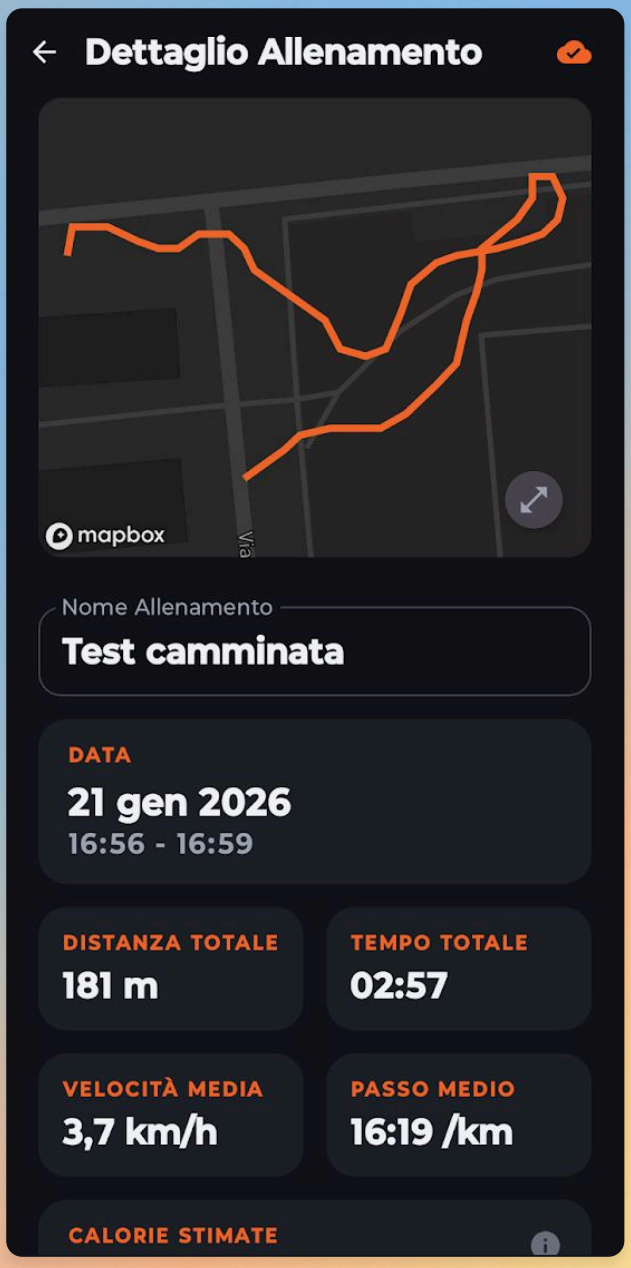
2.3 Dettaglio Sessione

Toccando una sessione dallo storico, si accede alla schermata di dettaglio, dove i dati vengono analizzati in profondità.

- **Mappa del Percorso:** Visualizzazione della polyline esatta del tragitto compiuto.
- **Statistiche Avanzate:** Oltre a durata e distanza, vengono mostrati la velocità media, il passo al km e il calcolo delle calorie bruciate, stimato incrociando i dati dell'attività con i parametri biometrici dell'utente (peso, altezza, età).
- **Gestione:** L'utente può rinominare l'allenamento per renderlo più riconoscibile o eliminarlo definitivamente. Un indicatore visivo segnala se la sessione è stata sincronizzata correttamente con il cloud.



Dettaglio (Light)



Dettaglio (Dark)

2.4 Profilo e Account

La schermata **Account** è il fulcro della gamification e della gestione personale.

- **Dati Utente:** Permette di inserire e modificare le informazioni biometriche (peso, altezza, data di nascita) fondamentali per l'accuratezza dei calcoli energetici.
- **Missioni e Livelli:** Una sezione dedicata mostra i progressi dell'utente verso obiettivi a lungo termine (es. "Macinatore di Km", "Costanza"). Ogni missione ha una barra di progresso e livelli sbloccabili (Bronzo, Argento, Oro, ecc.) che incentivano la continuità.
- **Streak:** Viene evidenziata la "striscia" di giorni consecutivi di allenamento, premiando la regolarità con emoji dedicate (es. 🔥 o 💯).

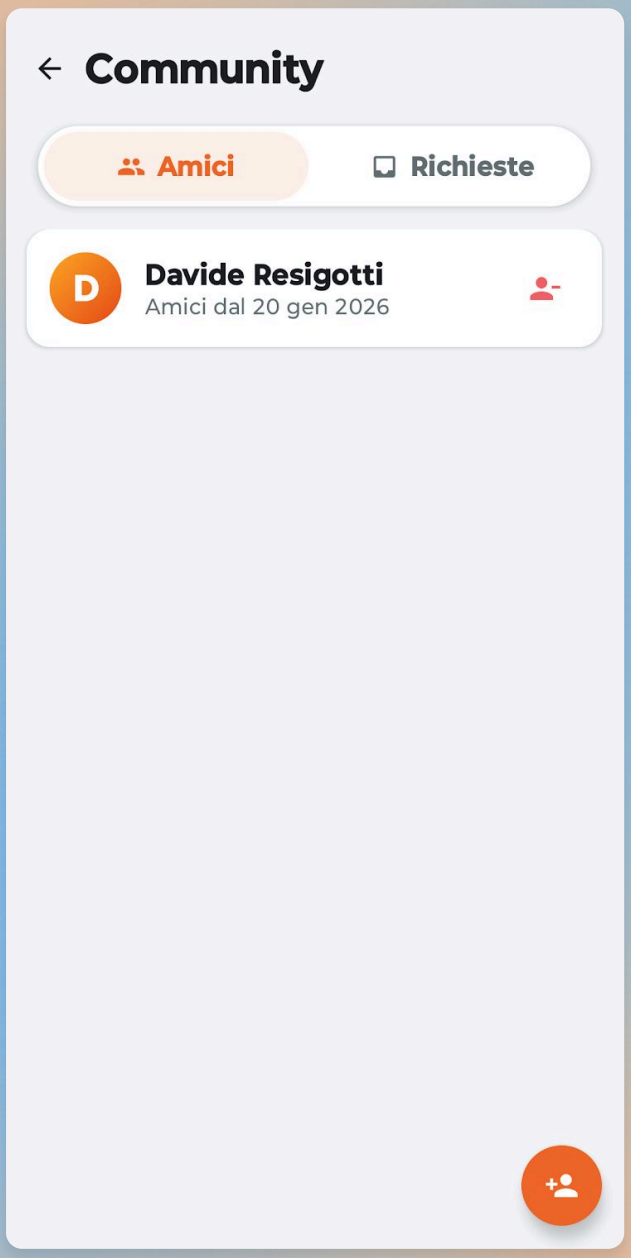
Profilo (Light)

Profilo (Dark)

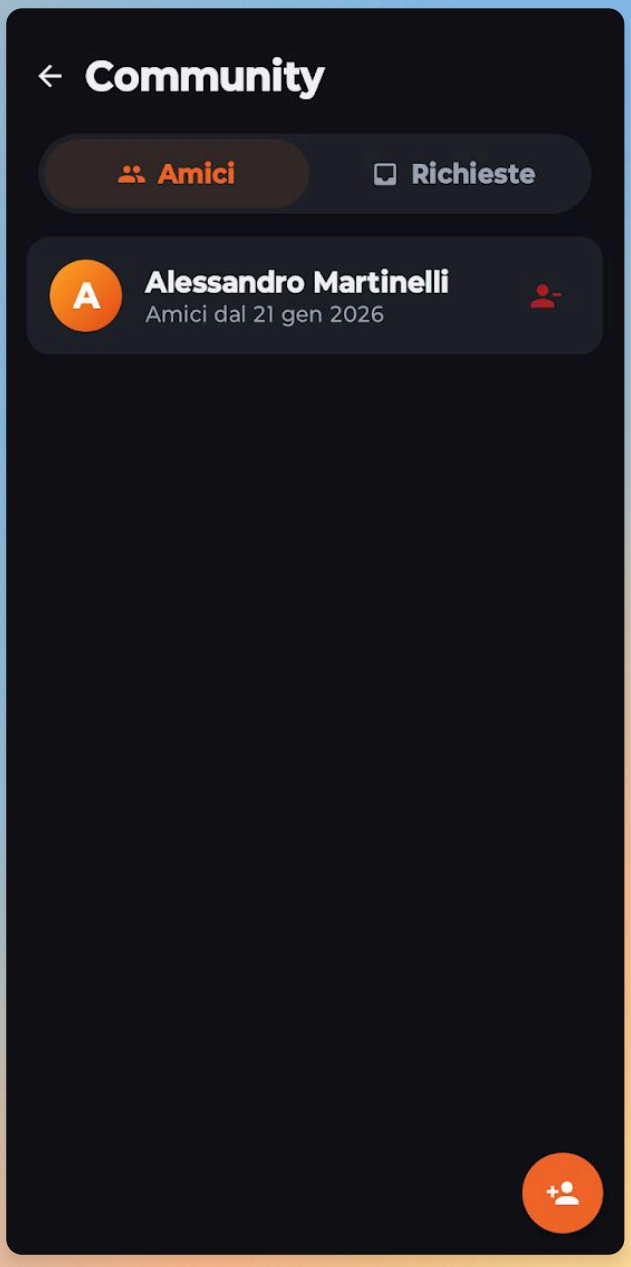
2.5 Social e Community

La componente sociale di Moes permette di connettersi con altri atleti per mantenere alta la motivazione.

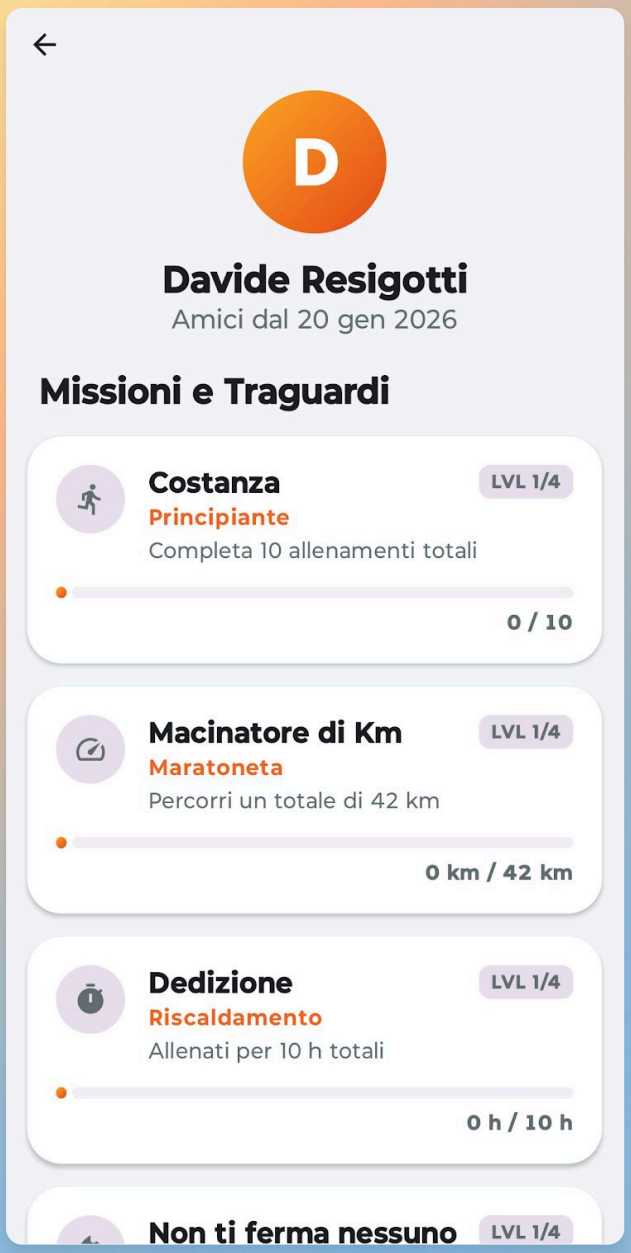
- **Gestione Amici:** È possibile cercare altri utenti tramite indirizzo email e inviare richieste di amicizia. Una gestione a schede separa la lista degli amici attuali dalle richieste in sospeso (inviate e ricevute).
- **Profilo Amico:** Visitando il profilo di un amico, è possibile visualizzare le sue statistiche principali, la sua streakattuale e le missioni che ha completato, stimolando una sana competizione.



Community (Light)



Community (Dark)



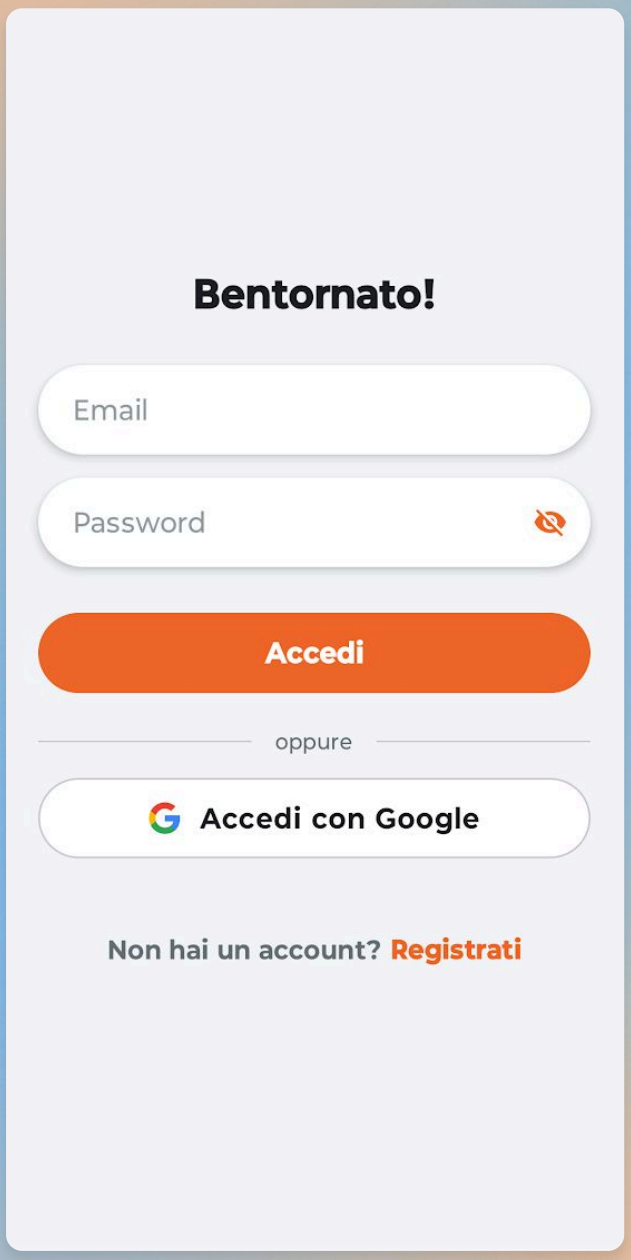
Profilo Amico (Light)

Profilo Amico (Dark)

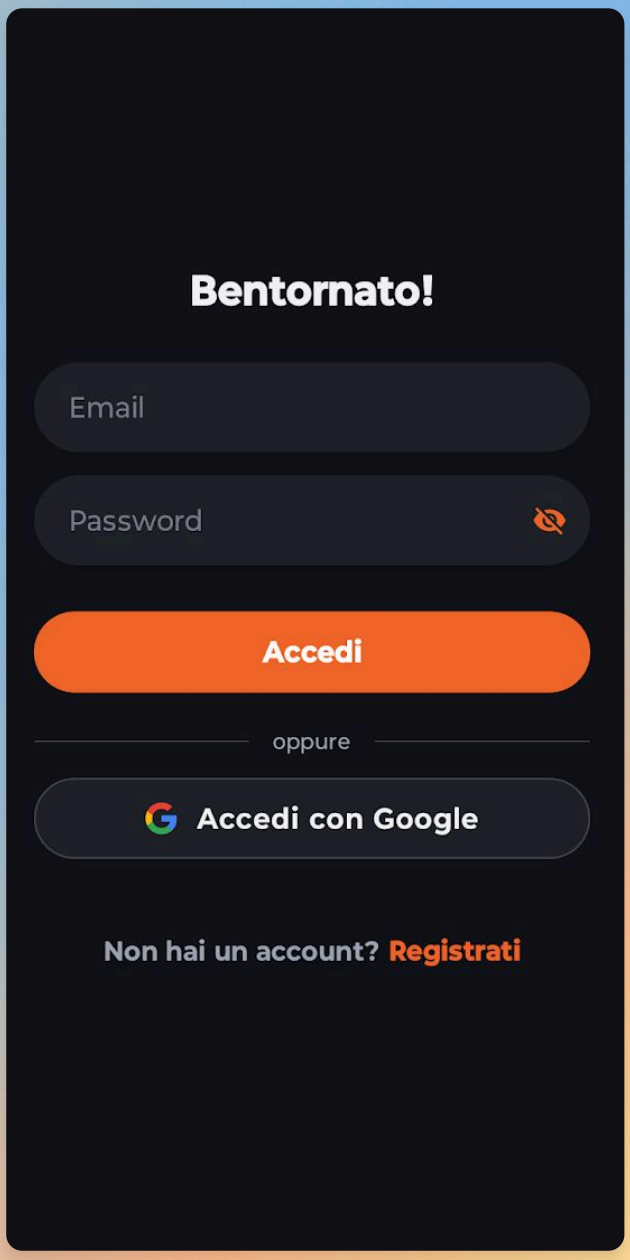
2.6 Autenticazione

L'accesso all'app è gestito in modo flessibile per abbattere le barriere all'ingresso.

- **Modalità Ospite:** L'utente può iniziare a usare l'app immediatamente senza registrarsi, con un profilo locale e un nome generato casualmente (es. "Bepi Polenta").
- **Login e Registrazione:** È possibile creare un account stabile tramite email/password o **Google Sign-In**. Una caratteristica fondamentale è la **migrazione dei dati**: se un utente ospite decide di registrarsi, tutto lo storico degli allenamenti e le statistiche accumulate vengono trasferiti automaticamente sul nuovo account cloud, senza perdita di dati.



Login (Light)



Login (Dark)

3. Architettura Software

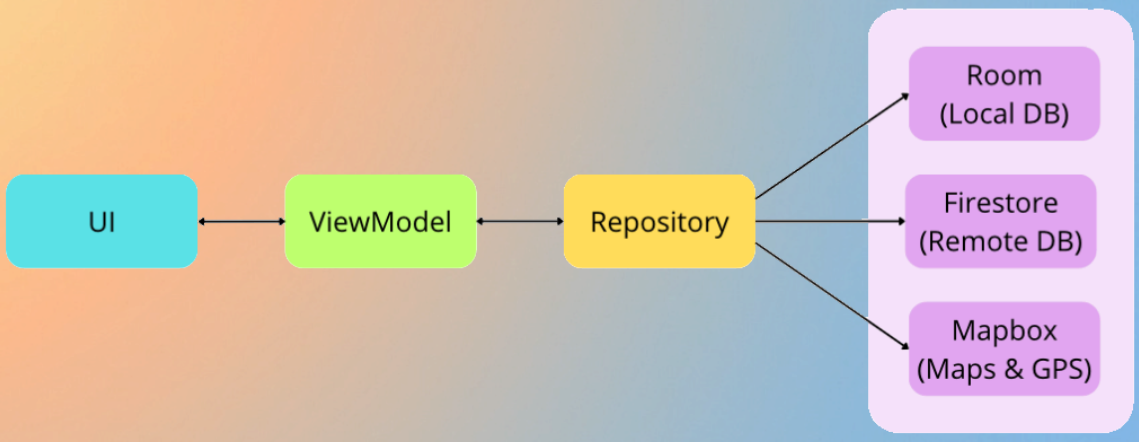
L'ingegneria del software alla base di **Moes** è stata guidata dai principi della Clean Architecture e dalla Separation of Concerns (SoC). L'obiettivo primario era creare un sistema modulare, scalabile e facilmente testabile, capace di gestire la complessità derivante dalla natura ibrida dei dati (sincronizzazione Locale/Cloud) e dalle interazioni in tempo reale con i sensori (GPS).

L'applicazione adotta il pattern architetturale MVVM (Model-View-ViewModel), standard de facto per lo sviluppo Android moderno, implementato attraverso un approccio strettamente reattivo.

3.1 Pattern Architetturale: MVVM e Unidirectional Data Flow (UDF)

L'interazione tra i componenti segue il paradigma del Flusso Dati Unidirezionale (UDF): lo stato fluisce verso il basso (dal ViewModel alla UI), mentre gli eventi fluiscono verso l'alto (dalla UI al ViewModel).

1. **View (UI Layer):** Realizzata interamente con Jetpack Compose, l'interfaccia è puramente dichiarativa. Le schermate (es. `SocialScreen`) non possiedono uno stato interno complesso, ma si limitano a osservare lo stato immutabile esposto dai ViewModel. Quando lo stato cambia, Compose ridisegna automaticamente solo le parti dell'interfaccia che necessitano di aggiornamento (Recomposition).
2. **ViewModel (Presentation Layer):** I ViewModel (es. `SocialViewModel`) fungono da "gestori di stato". Trasformano i dati grezzi provenienti dai Repository in oggetti `UiState` (es. `SocialUiState`) pronti per essere consumati dalla UI. Utilizzano `StateFlow` per mantenere l'ultimo stato valido e garantire che la UI sia sempre sincronizzata, anche dopo una rotazione dello schermo.
3. **Model (Data Layer):** I Repository (es. `TrainingRepository`) astraggono completamente le fonti dati. Il ViewModel ignora se i dati provengano dalla cache locale (Room), dal cloud (Firestore) o dai sensori; è il Repository a orchestrare queste fonti secondo la logica di business definita.



Schema del pattern MVVM e flussi dati in Moes

3.2 Stack Tecnologico

Lo stack tecnologico è stato selezionato per massimizzare le performance e la manutenibilità:

- **Linguaggio:** Kotlin (100%), sfruttando funzionalità avanzate come Extension Functions e Sealed Classes per la gestione degli stati.
- **Interfaccia Utente:** Jetpack Compose, che elimina la complessità dei file XML e del View Binding.
- **Persistenza Locale:** Room Database, un layer di astrazione su SQLite che offre verifica delle query a tempo di compilazione e integrazione nativa con i flussi asincroni (Flow).
- **Backend & Cloud:**
  - **Firebase Authentication:** Per la gestione sicura delle sessioni utente.
  - **Firebase Firestore:** Database NoSQL scalabile per la sincronizzazione e i dati social.
- **Mappe:** Mapbox SDK v10, scelto per le performance superiori nel rendering vettoriale e le capacità di routingoffline rispetto alle soluzioni native.

3.3 Navigazione "Single Activity"

L'applicazione segue l'architettura Single Activity: esiste un unico punto di ingresso (`MainActivity`) che ospita un grafo di navigazione gestito dal componente `MoesNavHost`. La navigazione tra le schermate non avviene lanciando nuove Activity (operazione costosa in termini di risorse), ma sostituendo i composable all'interno dello stesso container. Le rotte sono definite in modo type-safe nell'oggetto `Routes` (es. `Routes.SESSION_DETAIL`), prevenendo errori di digitazione e facilitando il passaggio di argomenti (es. ID sessione) tra le schermate.

3.4 Gestione della Concorrenza e Reactive Streams

La natura asincrona dell'app (GPS, Network, Database) è gestita tramite Kotlin Coroutines e Flow APIs.

- **Structured Concurrency:** Ogni operazione asincrona è legata a uno scope specifico (es. `viewModelScope`), garantendo che le risorse vengano liberate automaticamente quando l'utente lascia una schermata, prevenendo memory leaks.
- **Hot vs Cold Streams:** I Repository espongono spesso "Cold Flows" (che si attivano solo se c'è un osservatore). I ViewModel convertono questi flussi in "Hot Flows" (StateFlow) utilizzando l'operatore `stateIn(SharingStarted.WhileSubscribed(5000))`. Questa tecnica avanzata permette di mantenere i dati vivi per breve tempo (5 secondi) anche se la UI va in background (es. rotazione), ottimizzando le risorse.

3.5 Dependency Injection e Service Locator

Per la gestione delle dipendenze, si è optato per un approccio pragmatico basato sul pattern Service Locator, implementato manualmente nella classe `MoesApplication`. All'avvio, l'applicazione istanzia i singleton fondamentali (Database, Repository, Auth). Questi vengono poi iniettati nei ViewModel tramite una ViewModelFactory personalizzata. Questa scelta offre i benefici dell'Inversione del Controllo (IoC) — come la facilità di testare i componenti isolati — mantenendo però l'architettura snella e priva della complessità di framework come Hilt/Dagger, che avrebbero introdotto un overhead non necessario per le dimensioni attuali del progetto.

4. Dettaglio Componenti Chiave

In questo capitolo vengono analizzati i componenti tecnici fondamentali che abilitano le funzionalità core di Moes, con particolare attenzione alle scelte implementative per la gestione della posizione, l'integrazione delle mappe e la complessa architettura di sincronizzazione dei dati.

4.1 Il Service di Tracking: LiveTrainingService

Il monitoraggio dell'attività fisica è gestito da un **Android Service** dedicato, il `LiveTrainingService`. La scelta di utilizzare un servizio, specificamente un Foreground Service, è dettata dalla necessità di garantire la continuità del tracciamento anche quando l'applicazione viene messa in background o lo schermo viene spento per risparmiare batteria.

Logica di Acquisizione e Filtraggio: Il servizio utilizza il `FusedLocationProviderClient` per ottenere aggiornamenti sulla posizione. Per evitare "rumore" nei dati (es. punti GPS imprecisi che falserebbero il calcolo della distanza), è stato implementato un doppio filtro software all'interno del callback `onLocationResult`:

1. **Filtro di Accuratezza:** Vengono scartati automaticamente i punti con un raggio di accuratezza superiore a 20 metri.
2. **Filtro di Distanza (Jitter Filter):** Viene calcolata la distanza tra il nuovo punto e l'ultimo registrato usando la formula dell'Haversine. Se lo spostamento è inferiore a 3 metri, il punto viene ignorato. Questo previene il fenomeno per cui il GPS sembra "muoversi" anche quando l'utente è fermo.

4.2 Integrazione Mapbox

La gestione cartografica è affidata all'SDK di Mapbox, che viene utilizzato sia per la navigazione attiva che per la visualizzazione statica dei percorsi.

- **Repository di Navigazione:** Il `MapboxNavigationRepository` astrae la complessità dell'SDK. Espone la funzione `fetchRoute`, che accetta una lista di coordinate (origine e destinazione) e richiede al servizio di routing un percorso ottimizzato per il profilo "walking".
- **Visualizzazione Percorsi (SessionRouteMap):** Per mostrare il riassunto di un allenamento, è stato creato il componente `SessionRouteMap`. Questo non si limita a disegnare dei punti, ma decodifica la stringa compressa della geometria del percorso (utilizzando `PolylineUtils`) e la renderizza sulla mappa. Il componente gestisce anche una modalità Fullscreen tramite un Dialog per permettere all'utente di analizzare il tracciato nel dettaglio.

4.3 Gestione dei Dati e Sincronizzazione

La robustezza dei dati è garantita dal `DatabaseRepository`, che orchestra le operazioni tra il database locale Room e il backend Firestore, agendo come Single Source of Truth.

Logica di Base (Sync Strategy): Il sistema adotta un approccio ottimistico con fallback:

1. **Persistenza Immediata:** Al termine di un allenamento, la sessione viene salvata in locale con il flag `isSynced = false`.
2. **Tentativo di Upload:** Viene immediatamente tentato l'upload su Firestore. Se ha successo, il flag locale viene aggiornato a `true`.

Sincronizzazione Offline e Risoluzione Conflitti: La gestione della connettività intermittente è critica per un'applicazione outdoor. Moes implementa una strategia di sincronizzazione "Reactive & Event-Based" che assicura l'integrità dei dati senza richiedere l'intervento dell'utente.

- **Gestione dei Fallimenti:** Se l'upload immediato fallisce per mancanza di rete, la sessione rimane salvata localmente nel database Room con il flag `isSynced = false`. Questo stato persistente garantisce che nessun dato vada perso, anche se l'app viene chiusa.
- **Meccanismo di Retry (Reactive Sync):** Non affidandosi a un polling continuo che consumerebbe batteria, l'app utilizza un approccio reattivo. Il componente `NetworkMonitor` osserva lo stato della connessione in tempo reale. Non appena viene rilevato il ripristino della connettività (`isOnline = true`), l'interfaccia invoca automaticamente `syncPendingSessions()`. Questa funzione interroga il DB locale per tutte le sessioni non sincronizzate e ritenta l'upload in batch, marcandole come sincronizzate solo al successo dell'operazione.
- **Risoluzione dei Conflitti:** Per prevenire sovrascritture accidentali, il sistema applica la politica "Last Write Wins" basata sul timestamp per il profilo utente: viene mantenuta sempre la versione con il campo `lastEdited` più recente.

Migrazione Dati Ospite-Utente: Quando un utente decide di creare un account, viene invocata la funzione `migrateGuestData` che esegue tre operazioni atomiche:

1. Aggiorna l'ID utente (`ownerId`) di tutte le sessioni locali.
2. Fonde le statistiche accumulate come ospite con quelle sul cloud.
3. Elimina i dati temporanei dell'ospite.

5. Classi Utils

La logica di business di Moes si avvale di una serie di classi di utilità (Object singleton in Kotlin) che incapsulano algoritmi specifici per il calcolo delle prestazioni, la compressione dei dati e la generazione di contenuti.

5.1 Calcolo delle Calorie: CaloriesCalculator

Per fornire una stima realistica del dispendio energetico, l'applicazione implementa un algoritmo fisiologico che incrocia i dati dell'attività con i parametri biometrici dell'utente.

Il processo di calcolo avviene in quattro fasi:

1. **Recupero Dati Biometrici:** Vengono estratti peso, altezza, età e genere dal profilo utente. In caso di dati mancanti, vengono utilizzati valori di fallback standard.
2. **Calcolo BMR:** Viene calcolato il metabolismo basale utilizzando la Formula di Harris-Benedict, che calcola il consumo calorico del corpo a riposo in 24 ore.
3. **Stima del MET:** Viene determinato il coefficiente MET basandosi sulla velocità media della sessione (da 2.3 per camminata lenta a 16.0 per sprint).
4. **Calcolo Finale:** Il consumo totale è ottenuto moltiplicando il BMR orario per il valore MET e per la durata dell'attività.

5.2 Compressione Percorsi: PolylineUtils

Un allenamento di lunga durata può generare migliaia di punti GPS. Per risolvere questo problema, è stata implementata l'utility `PolylineUtils` che utilizza l'algoritmo `Encoded Polyline Algorithm` (standard utilizzato da Google e Mapbox).

- **Encoding:** Converte una lista di oggetti `Coordinate` in un'unica stringa ASCII compressa. Questo riduce drasticamente la dimensione dei dati da salvare nel campo routeGeometry della TrainingSession.
- **Decoding:** Quando l'utente visualizza il dettaglio di una sessione, la stringa viene decodificata per ricostruire la lista di punti e disegnare il tracciato sulla mappa.

5.3 Calcolo Statistiche e Streak: StatisticsUtils

La gamification si basa sul concetto di "Streak" (giorni consecutivi di attività). L'algoritmo di calcolo, contenuto in `StatisticsUtils`, gestisce l'aggiornamento incrementale delle statistiche.

Ad ogni nuovo allenamento completato:

1. **Calcolo Giorni:** Viene calcolata la differenza in giorni tra la data dell'ultimo allenamento e quella attuale.
2. **Logica Streak:**
   - Se la differenza è 0 (stesso giorno): La streak non cambia.
   - Se la differenza è 1 (giorno consecutivo): La streak viene incrementata.
   - Se la differenza è > 1 (giorno saltato): La streak viene resettata a 1.
3. **Aggiornamento Totali:** Vengono sommati distanza, durata e numero di sessioni ai totali esistenti, aggiornando se necessario il record di `longestStreakDays`

Inoltre, l'utility gestisce il Merge delle Statistiche per unire i dati di un utente ospite con quelli di un account cloud esistente al momento del login.

5.4 Generazione Nomi Ospite: BerghemNameGenerator

Per rendere l'esperienza utente simpatica fin dal primo avvio (senza login), è stato creato un generatore casuale di nomi "alla bergamasca". L'utility `BerghemNameGenerator` combina una lista di nomi tipici (es. "Bepi", "Giopì") con cognomi folcloristici (es. "Polenta", "Casoncello", "MolaMia"), assegnando un'identità temporanea unica a ogni nuovo utente ospite.

6. Design e UI/UX

L'interfaccia di Moes è stata realizzata interamente con il toolkit dichiarativo **Jetpack Compose**, adottando i principi del Material Design 3 ma personalizzandoli per riflettere l'energia e il dinamismo del brand. Il design privilegia la leggibilità in movimento (essenziale per un'app di fitness) e un feedback visivo immediato.

6.1 Palette Cromatica e Temi

L'identità visiva è definita da colori caldi e vibranti che stimolano l'azione.

- **Colori Primari:** Il colore principale è il `Brand Primary` (un arancione intenso), affiancato dal `Brand Secondary` (giallo). Questi vengono spesso combinati in gradienti lineari (`LogoGradientStart` → `LogoGradientEnd`) per dare profondità a elementi chiave come i pulsanti di azione e le barre di progresso delle missioni.
- **Colori Funzionali:** Per le indicazioni di errore o stop è utilizzato un rosso deciso (`ErrorRed`), mentre per i testi secondari si impiegano diverse tonalità di grigio (`TextGray`, `TextGrayDark`) per garantire il giusto contrasto.
- **Gestione Tema Chiaro/Scuro:** L'applicazione supporta nativamente la modalità notte. Il `MoesTheme` rileva le impostazioni di sistema (`isSystemInDarkTheme()`) e adatta automaticamente la palette:
  - **Light Mode:** Utilizza sfondi chiari e moderni (`BackgroundModern`) con superfici bianche.
  - **Dark Mode:** Passa a sfondi scuri profondi (`BackgroundDark`) con superfici in grigio scuro (`SurfaceDark`) per ridurre l'affaticamento visivo e il consumo di batteria sugli schermi AMOLED.

6.2 Componenti Custom (Composables)

Per soddisfare requisiti specifici non coperti dai componenti standard, sono stati sviluppati diversi Widget personalizzati:

- **Training Overlay:** È il pannello di controllo flottante visualizzato durante l'attività. Progettato con angoli fortemente arrotondati e un'elevazione marcata (`shadowElevation`), raggruppa le metriche vitali (tempo, passo, distanza) e i controlli di riproduzione (Pausa/Stop) in un'interfaccia compatta che non ostruisce la mappa sottostante.
- **Instruction Banner:** Utilizzato nella modalità navigazione, questo componente mostra le indicazioni di svolta ("Turn-by-Turn"). Include un'icona direzionale dinamica che ruota in base al tipo di manovra e il conteggio della distanza rimanente alla prossima svolta.
- **Mission Card:** Per la gamification, è stata creata una scheda complessa che visualizza lo stato di una missione. Utilizza gradienti per la barra di avanzamento e cambia aspetto visivo (colori e icone) quando la missione viene completata, offrendo una gratificazione immediata all'utente.

6.3 Feedback Visivo e Iconografia

L'interfaccia comunica lo stato dei dati non solo tramite testo, ma anche attraverso icone intelligenti:

- **Activity Icon:** Nella lista delle sessioni, l'icona dell'attività non è statica ma viene determinata algoritmicamente. In base alla velocità media registrata, il sistema assegna automaticamente l'icona appropriata: "Camminata" (< 6.5 km/h), "Corsa" (< 20 km/h) o "Ciclismo" (> 20 km/h).
- **Indicatori di Sync:** Icone specifiche (`CloudDone` vs `CloudOff`) informano l'utente a colpo d'occhio se i dati di una sessione o del profilo sono stati sincronizzati con il cloud o sono in attesa di connessione.

7. Organizzazione del Progetto

7.1 Struttura del Codice (Package Structure)

Il codice è organizzato seguendo il pattern MVVM. La gerarchia è la seguente: MVVM

- **ui (Presentation Layer):** Contiene tutto ciò che riguarda l'interfaccia utente.
  - `screens`: Le schermate principali dell'app (es. `HomeScreen`, `SessionsScreen`, `SocialScreen`).
  - `viewmodels`: I componenti che gestiscono la logica di presentazione e lo stato della UI (es. `HomeViewModel`, `SocialViewModel`).
  - `composables`: Elementi UI riutilizzabili divisi per contesto (es. profile, sessions, social, utils), come le `MissionCard` o i `TrainingOverlay`.
  - `navigation`: Gestione del grafo di navigazione e delle rotte (`MoesNavHost`).
  - `theme`: Definizioni di colori, tipografia e temi chiaro/scuro.

- **data (Data Layer):** Gestione dei dati e dei modelli.
  - `local`: Configurazione del database Room e dei DAO (`AppDatabase`, `TrainingDao`, `UserDao`).
  - `remote`: Gestione delle chiamate a Firestore (`FirestoreDataSource`).
  - `models`: Le Data Class che rappresentano le entità del dominio, come `TrainingSession`, `UserProfile`, `LiveTrainingSession` (per i dati in tempo reale) e `SocialModels` (per amici e richieste).
  - `missions`: Definizioni statiche delle missioni e logica di progressione.

- **repositories (Domain Layer):**
  - Contiene le classi che astraggono l'accesso ai dati, agendo da unica fonte di verità per i ViewModel. Include `TrainingRepository`, `SocialRepository`, `AuthRepository` e i repository specifici per Mapbox.

- **services:**
  - Contiene i componenti Android che devono sopravvivere al ciclo di vita della UI, in particolare il `LiveTrainingService` per il tracciamento GPS in background.

- **utils:**
  - Classi di utilità e algoritmi puri, come `CaloriesCalculator` (calcolo metabolico), `PolylineUtils` (codifica percorsi), `FormatUtils` (formattazione date/distanze) e `BerghemNameGenerator`.

7.2 Gestione delle Risorse

Oltre al codice Kotlin, il progetto fa ampio uso delle risorse Android standard:

- **Drawable:** Icone vettoriali (SVG/XML) per garantire nitidezza su ogni densità di schermo.
- **Strings:** Tutte le stringhe visibili sono estratte nel file `strings.xml` (anche se non mostrato nel codice caricato, è standard in Android) per facilitare eventuali traduzioni future.
- **Gradle:** La gestione delle dipendenze (librerie esterne come Mapbox, Firebase, Room) è centralizzata nei file di build Gradle.

8. Conclusioni e Sviluppi Futuri

Il progetto Moes ha raggiunto l'obiettivo di creare un'applicazione di fitness tracking completa, moderna e coinvolgente. Partendo dalla necessità di combattere la sedentarietà, è stato sviluppato un ecosistema software che non si limita a registrare dati, ma li trasforma in motivazione attraverso la gamification e la socialità.

Dal punto di vista tecnico, l'applicazione dimostra la robustezza dell'architettura MVVM e l'efficacia dello stack tecnologico basato su Kotlin e Jetpack Compose. La gestione ibrida dei dati (Locale + Cloud) garantisce un'esperienza utente fluida in ogni condizione di rete, mentre l'integrazione profonda con i servizi di geolocalizzazione e mappatura offre precisione e affidabilità.

8.1 Sviluppi Futuri

Nonostante il set di funzionalità sia completo per un utilizzo quotidiano, l'architettura modulare del progetto si presta a diverse estensioni future:

- **Companion App per Wear OS:** L'attuale logica di tracciamento incapsulata nel `LiveTrainingService` potrebbe essere portata su smartwatch Android. Questo permetterebbe agli utenti di registrare le sessioni lasciando il telefono a casa, sincronizzando i dati con l'app principale al rientro.
- **Social Sharing Avanzato:** Attualmente i percorsi sono visibili solo in-app. Una funzionalità chiave da implementare è la generazione di immagini "condivisibili" (es. per Instagram Stories) che sovrappongono la mappa del percorso (`SessionRouteMap`) con le statistiche chiave della sessione, sfruttando le API di rendering di Mapbox.
- **Sfide in Tempo Reale:** Sfruttando la natura Realtime di Firestore, si potrebbe introdurre una modalità "Gara": due amici potrebbero correre contemporaneamente in luoghi diversi, vedendo il progresso dell'avversario aggiornarsi in tempo reale sul proprio overlay di allenamento.
- **Supporto Multi-Profilo di Navigazione:** Attualmente il calcolo delle rotte è ottimizzato per i pedoni (mapbox/walking). Un'evoluzione naturale sarebbe permettere all'utente di scegliere profili specifici per il ciclismo, ottenendo percorsi che privilegiano le piste ciclabili e le strade asfaltate.