

Relazione di Progetto
Programmazione di Dispositivi Mobili Anno Accademico 2023-2024



Gruppo Apperò
Riccardo Mattia 885964
Riccardo Savio 885955
Paolo Cislaghi 885899

Introduzione	2
Tecnologie e Linguaggi utilizzati	3
Organizzazione del progetto	3
Processo di Integrazione	3
Funzionalità	5
Diagramma dei Casi D'uso	5
Login	6
Registrazione	6
Home	7
Mappa	8
Profilo	8
Personalizzazione dei dati Utente	8
Creazione Viaggi	9
Aggiunta Membri	10
Aggiunta Tappe	10
Architettura dell'applicazione	11
1. UI Layer	11
2. ViewModels Layer	11
3. Repository Layer	11
4. DataSource Layer	12
5. Database e Servizi Remoti	12
Flusso dei Dati	12
Organizzazione Database	13
Introduzione a Firestore	13
Struttura del Database	13
Sviluppi Futuri	17

Introduzione

TravelHub è un'app di viaggio innovativa progettata per salvare e condividere le esperienze di viaggio degli utenti, fungendo da diario digitale interattivo. L'obiettivo principale di TravelHub è quello di offrire una piattaforma completa e intuitiva che consente agli utenti di documentare ogni dettaglio dei loro viaggi, dalle date alle tappe, dai partecipanti alle attività svolte.

TravelHub permette di rivivere le avventure passate attraverso mappe interattive che mostrano tutti i luoghi visitati. Inoltre, la funzione di condivisione facilita la comunicazione e la collaborazione con amici e familiari, permettendo loro di visualizzare e commentare i vostri viaggi.

Tecnologie e Linguaggi utilizzati

- **Android Studio:** utilizzato come IDE per sviluppare il progetto
- **Firebase:** è una piattaforma per la creazione di applicazioni per dispositivi mobili e web sviluppata da Google. TravelHub sfrutta **firebase Authentication** per registrare e loggare gli utenti, utilizza **Firestore Database** per memorizzare i nomi utenti e i viaggi intrapresi dai vari utenti e utilizza il servizio di **Storage** per memorizzare le immagini profilo degli utenti.
- [Google API Places](#): api di google sfruttata per trovare posti esistenti e le loro coordinate
- **Git e Github:** sfruttati come sistema di versionamento per il codice dell'applicazione, questo sistema ha velocizzato lo sviluppo in parallelo dell'applicazione

Organizzazione del progetto

Per organizzare lo sviluppo del progetto TravelHub, abbiamo adottato Git come sistema di controllo della versione e utilizzato diversi branch per suddividere i compiti tra i tre programmatore del team. Questa strategia ci ha permesso di lavorare in parallelo, mantenendo il codice organizzato e riducendo al minimo i conflitti.

Ecco come abbiamo strutturato il nostro workflow:

1. **Branch Principale (main):** Questo branch contiene la versione stabile e rilasciabile dell'app. Ogni modifica apportata a questo branch è attentamente controllata e testata.
2. **Branch di Sviluppo (dev):** I vari branch dev sono stati utilizzati per integrare delle funzionalità principali, poi unite tramite merge al branch main

Processo di Integrazione

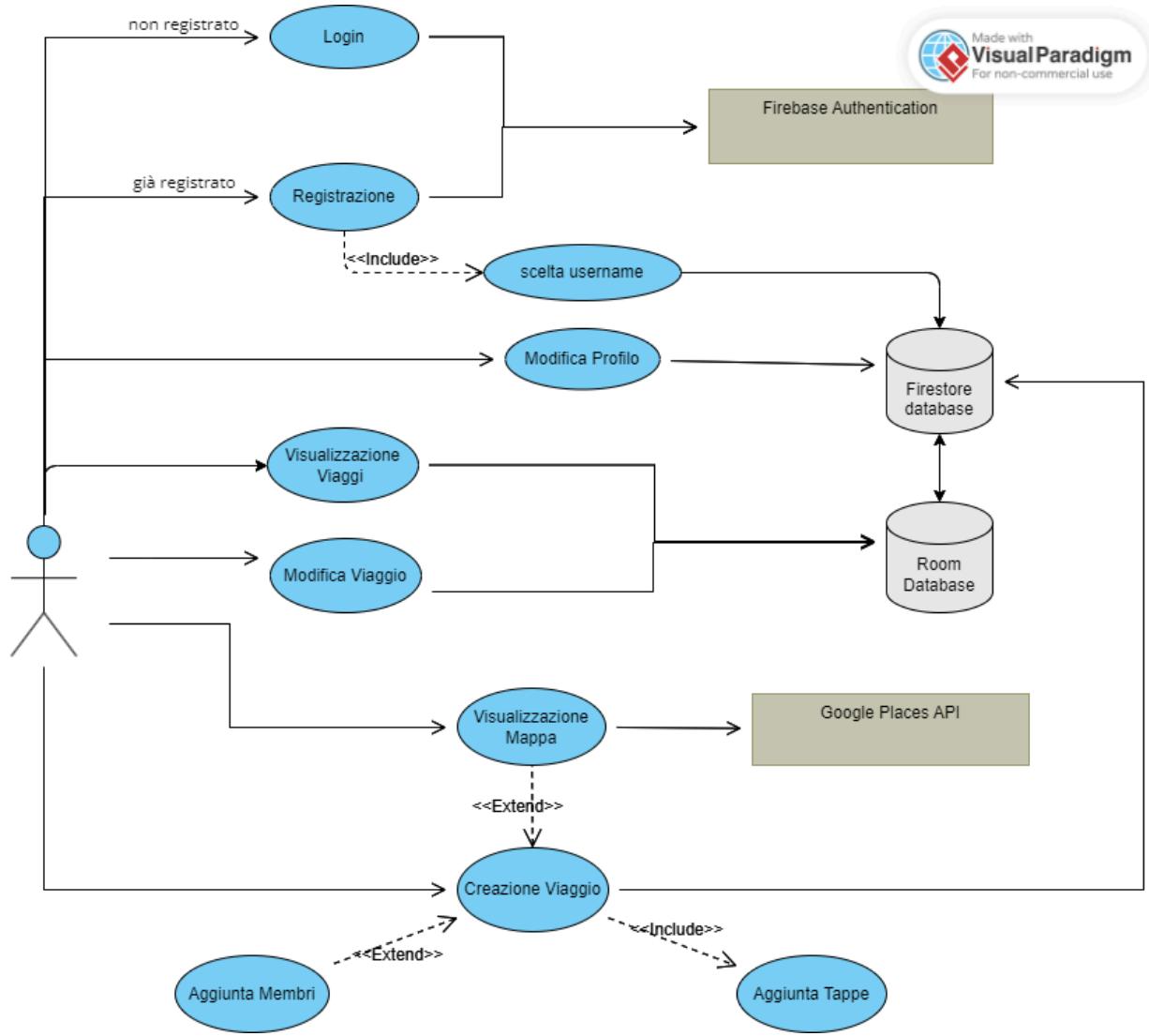
1. **Completamento delle Funzionalità:** Ogni programmatore sviluppa e testa la propria funzionalità nel rispettivo feature branch.
2. **Test:** Ogni programmatore verifica che la funzionalità implementata sia effettivamente corretta effettuando delle prove tramite l'applicazione

3. **Rilascio:** Quando tutte le funzionalità sono integrate e testate nel proprio ramo develop, il branch viene fuso in main per il rilascio della nuova versione dell'app.

Utilizzando questo approccio strutturato con Git e branch distinti, abbiamo garantito un flusso di lavoro efficiente e collaborativo, permettendo ai programmatori di lavorare simultaneamente su diverse funzionalità senza interferire con il lavoro degli altri.

Funzionalità

Diagramma dei Casi D'uso



Login

All'utente che apre l'app per la prima volta o successivamente a un logout verrà richiesto di effettuare il login. Questo può avvenire in due modalità:

1. **Tramite email e password:** L'utente dovrà inserire le proprie credenziali definite in fase di registrazione.
2. **Tramite Google:**
 - Se l'utente ha già effettuato il login con Google o utilizzato l'email associata all'account Google, verrà reindirizzato direttamente alla home dell'applicazione.
 - Se l'utente è nuovo e tenta di accedere per la prima volta con Google, verrà reindirizzato a una pagina in cui dovrà scegliere un nome utente. Dopo averlo scelto l'applicazione ne verificherà l'unicità, se fosse un username disponibile l'utente verrà reindirizzato alla home dell'applicazione.

Il login verrà richiesto una sola volta; successivamente, l'accesso sarà effettuato automaticamente utilizzando le credenziali salvate sul dispositivo.

Registrazione

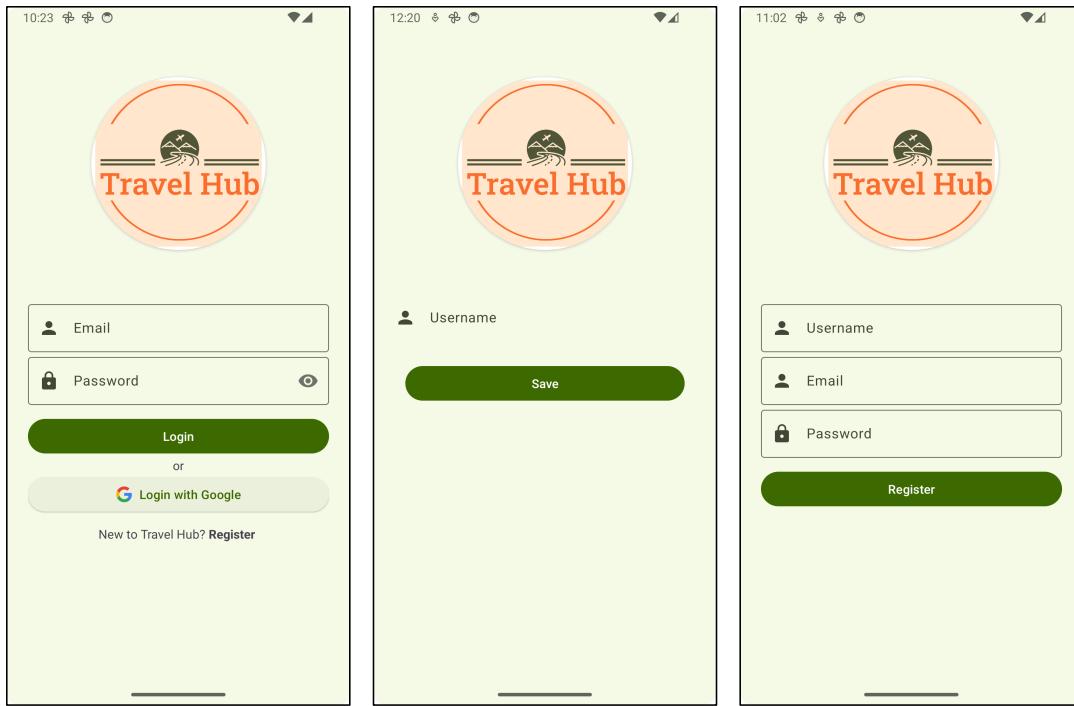
Se l'utente non è registrato, può farlo nella schermata di registrazione. I campi necessari per la registrazione sono:

- Nome utente
- Email
- Password

In questa fase, sono stati implementati dei meccanismi di validazione. L'utente non verrà registrato se:

- Inserisce un nome utente già esistente
- Inserisce una email già esistente
- Inserisce nel campo email una stringa che non corrisponde a un'email valida
- Inserisce una password troppo debole

L'utente riceverà un feedback a schermo su cosa correggere e, una volta effettuate le correzioni necessarie, potrà completare la registrazione e verrà reindirizzato alla home dell'applicazione.



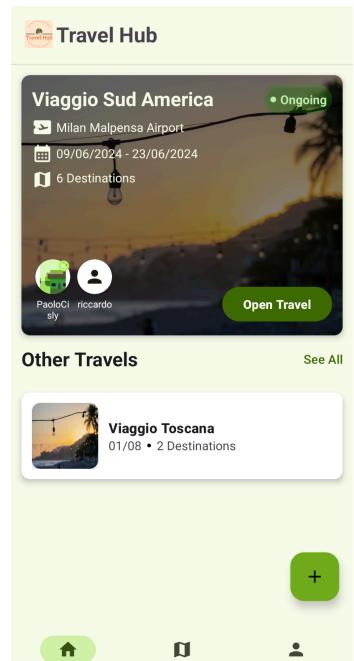
In ordine schermate di Login, Scelta Username, Registrazione

Home

Nella schermata principale vengono visualizzati i viaggi che l'utente ha creato o ai quali è stato aggiunto come membro da altri utenti.

Nella home vengono mostrati al massimo due viaggi, ordinati in base alle date del viaggio nel seguente ordine: prima i viaggi in corso, successivamente quelli futuri, e infine, in assenza di questi, i viaggi passati.

Dalla home è possibile accedere rapidamente, tramite il menu inferiore, alla mappa dei luoghi visitati e alla sezione dedicata al profilo. Inoltre, tramite un pulsante elevato (elevated button), è



possibile accedere al fragment che guiderà l'utente nella [creazione di un nuovo viaggio](#).

Mappa



Questa schermata mostrerà all'utente tutti i luoghi visitati o nei quali ha fatto tappa, visualizzati su una cartina interattiva. L'obiettivo di questa schermata è migliorare l'esperienza utente, offrendo una visualizzazione immediata e piacevole di tutti i luoghi visitati.

Profilo

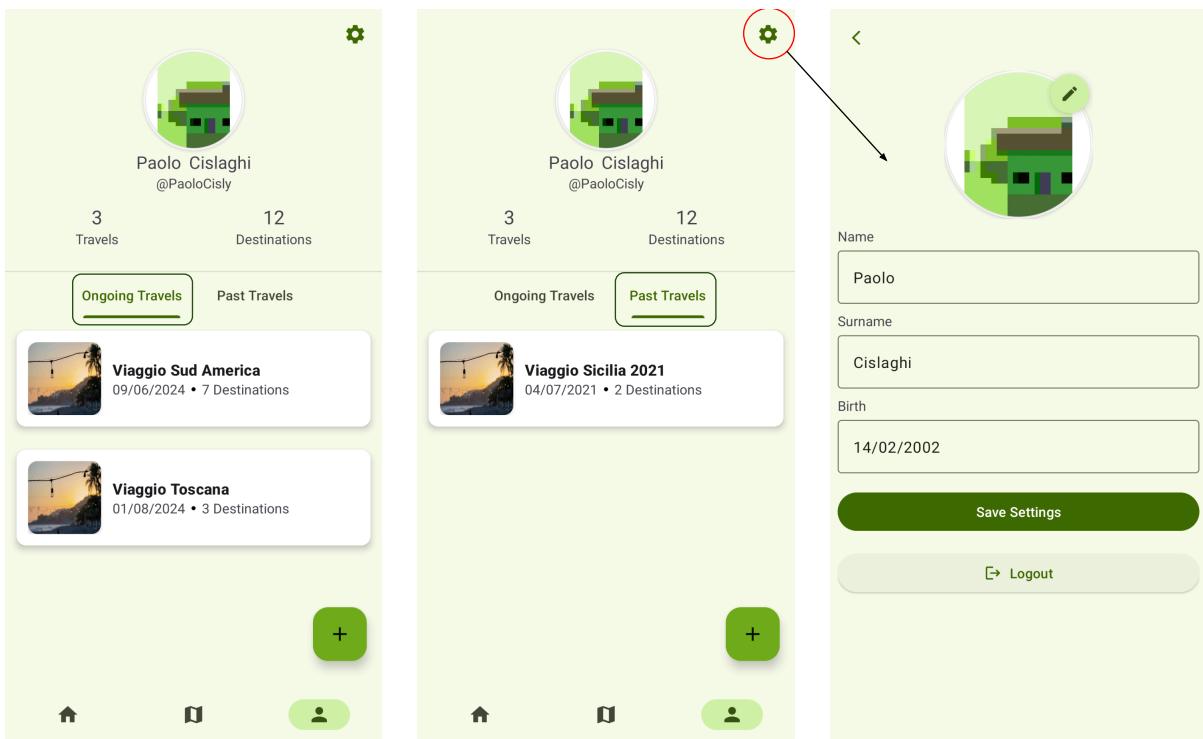
Questa schermata presenta la foto profilo inserita dall'utente, il numero di avventure che ha intrapreso e un elenco di tutti i suoi viaggi futuri e passati. Inoltre da questa schermata si può raggiungere la schermata di [personalizzazione dei dati utente](#).

Personalizzazione dei dati Utente

In questa schermata l'utente potrà inserire la sua foto profilo, il suo nome e cognome e la sua data di nascita. Il pulsante di salvataggio scriverà le modifiche ai suoi dati su Firestore Database mentre se si esce da questa schermata senza salvare si perderanno tutte le modifiche ai dati inseriti in quel momento.

Inoltre da questa schermata l'utente potrà accedere al pulsante di logout, il logout cancellerà il file di memoria criptato che contiene le credenziali e il file non criptato che

contiene le preferenze utente. Inoltre dopo il Logout l'utente dovrà procedere nuovamente ad effettuare il [login](#) con il suo o un'altro account per utilizzare l'applicazione.



Creazione Viaggi

Da questa schermata è possibile creare un nuovo viaggio riempiendo i campi necessari contrassegnati con *.
Sono implementati controlli per i campi vuoti, controlli di congruenza delle date per le [tappe](#) e controlli sull'esistenza dei membri da aggiungere al viaggio.
Una volta creato un viaggio sarà permesso all'utente di salvarlo tramite l'apposito pulsante, questo pulsante rimarrà bloccato fino a che non sono stati riempiti tutti i campi necessari alla creazione di un viaggio.

The screenshot shows the 'Add Member' screen for a trip titled 'Viaggio Venezia'. It includes fields for 'From' (16/06/2024) and 'To' (22/06/2024), a 'Departure' field set to 'Milan', and a 'Description' section. Below these, the 'Participants' section lists 'PaoloCisly' with an 'Add' button. An arrow points from the 'Add' button to a 'Username' input field and a 'Aggiungi Partecipante' button.

Title* - Viaggio Venezia

From* - 16/06/2024 To* - 22/06/2024

Departure* - Milan Select Location

Description

Participants

Add PaoloCisly

Aggiungi Partecipante

Username

+ Aggiungi Partecipante

Aggiunta Membri

per aggiungere un membro al viaggio basterà cliccare sull'apposito pulsante, a quel punto verrà chiesto all'utente di inserire l'username del membro.

A questo punto, se l'username viene trovato nel database, l'applicazione aggiungerà al viaggio il nuovo membro, altrimenti verrà mostrato a schermo un messaggio di errore.

The screenshot shows the 'Add Destination' screen for a trip. It includes fields for 'Destination', 'From' (16/06/2024), and 'To'. Below these, there is a 'Description' section and a large green 'Destination' button at the bottom.

Add destination

Destination Select Location

From - 16/06/2024 To

Description

Destination

Aggiunta Tappe

Si raggiunge questa schermata soltanto durante la creazione di un viaggio. Secondo il nostro modello dei dati un viaggio è composto da una partenza e una o molte destinazioni. per ogni destinazione desiderata si aprirà una di queste schermate, ancora una volta l'utente sarà guidato dai dall'interfaccia e dai feedback dell'applicazione per aggiungere le tappe desiderate.

Architettura dell'applicazione

Questo schema architettonico rappresenta l'architettura di un'applicazione Android che segue il pattern MVVM (Model-View-ViewModel) e utilizza sia un datasource locale che uno remoto. Ecco una descrizione dettagliata dei componenti e delle loro interazioni:

1. UI Layer

- **Activity**
- **Fragment**

Questi componenti costituiscono il livello dell'interfaccia utente (UI). L'Activity è l'elemento principale dell'interfaccia utente che può ospitare uno o più Fragment, i quali rappresentano porzioni dell'interfaccia utente.

2. ViewModels Layer

- **User ViewModel**
- **Travel ViewModel**

Questi componenti gestiscono i dati per la UI e fungono da intermediari tra l'UI e il livello dei repository. Contengono la logica di presentazione e osservano i dati esposti dai repository per aggiornare la UI di conseguenza.

3. Repository Layer

- **User Repository**
- **Travel Repository**

I repository fungono da unica fonte di verità per i dati, gestendo la logica per recuperare e salvare i dati. Decidono se ottenere i dati dal datasource locale o remoto. In questo schema, i repository sono responsabili di ottenere i dati dai datasource appropriati. Nella nostra soluzione i viaggi vengono scaricati dal database remoto e inseriti nel database locale. ci sarà un periodo di tempo in cui alla richiesta di nuovi viaggi il repository chiederà le informazioni al datasource locale, finito questo periodo di tempo i nuovi viaggi verranno scaricati dal database remoto e inseriti in quello locale

4. DataSource Layer

- **Local DataSource**
- **Remote DataSource**

Questi componenti gestiscono la logica per l'accesso ai dati. Nella nostra applicazione il Travel Repository ha a disposizione 2 datasource uno locale e uno remoto, mentre lo user repository ha solo il datasource remoto.

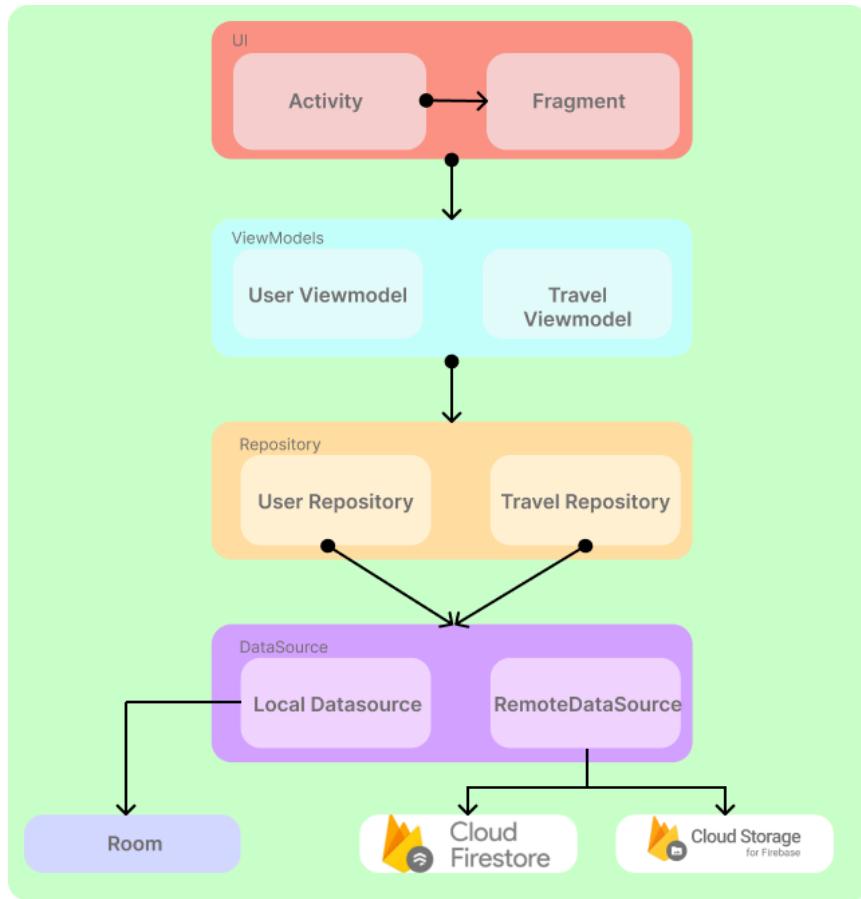
- **Local DataSource:** Interagisce con il database locale, in questo caso, **Room**.
- **Remote DataSource:** Interagisce con servizi cloud, in questo caso, **Cloud Firestore** e **Cloud Storage for Firebase**.

5. Database e Servizi Remoti

- **Room:** È una libreria di persistenza per Android che fornisce un livello di astrazione sul database SQLite.
- **Cloud Firestore:** È un database flessibile e scalabile per lo sviluppo mobile, web e server da Firebase e Google Cloud Platform.
- **Cloud Storage for Firebase:** Utilizzato per memorizzare e recuperare file come immagini, video e altri contenuti generati dagli utenti.

Flusso dei Dati

1. **UI Layer** (Activity/Fragment) interagisce con i **ViewModel** per ottenere i dati.
2. I **ViewModel** richiedono i dati dai **Repository** (User Repository/Travel Repository).
3. I **Repository** decidono se recuperare i dati dal **Local DataSource** o dal **Remote DataSource** in base alla disponibilità e alla logica di caching.
4. Il **Local DataSource** interagisce con **Room** per ottenere o memorizzare i dati localmente.
5. Il **Remote DataSource** interagisce con **Cloud Firestore** o **Cloud Storage for Firebase** per ottenere o memorizzare i dati nei servizi cloud.



Organizzazione Database

Introduzione a Firestore

Firestore è un database NoSQL offerto da Google come parte della piattaforma Firebase. È progettato per la sincronizzazione in tempo reale e supporta la scalabilità automatica. In Firestore, i dati sono organizzati in **collezioni** e **documenti**.

- **Collezioni:** Una collezione è un contenitore di documenti. Ogni collezione può contenere un numero illimitato di documenti.
- **Documenti:** Un documento è un contenitore di coppie chiave-valore, dove ogni chiave è un campo e ogni valore può essere un tipo di dato semplice (come stringhe, numeri) o più complesso (come array, oggetti nidificati).

Struttura del Database

Nel nostro database Firestore, abbiamo organizzato i dati in tre collezioni principali per memorizzare i dati: `travels`, `usernames`, e `users`.

1. Collezione `travels`

La collezione `travels` contiene i documenti dei viaggi. Ogni documento rappresenta un viaggio e il nome del documento è l'ID del viaggio. Ogni documento ha i seguenti campi:

- `title`: il titolo del viaggio (stringa)
- `description`: la descrizione del viaggio (stringa)
- `startDate`: la data di inizio del viaggio (data)
- `members`: un array di oggetti che rappresentano i membri del viaggio, dove ogni oggetto contiene:
 - `userId`: l'ID dell'utente (stringa)
 - `role`: il ruolo dell'utente nel viaggio, che può essere `MEMBER` o `CREATOR` (stringa)
- `destinations`: un array di destinazioni (oggetti, la struttura specifica può variare), dove ogni oggetto contiene:
 - `location`: Luogo di destinazione (stringa)
 - `dateFrom`: data di partenza per quella destinazione (data)
 - `dateTo`: data di arrivo per quella destinazione (data)
 - `description`: descrizione della tappa (stringa)
 - `lat`: rappresenta la latitudine della posizione (float)
 - `lon`: rappresenta la longitudine della posizione (float)

Ecco un esempio della collezione `travels`:

JavaScript

```
"travels": {  
  "-1267757977": {  
    "title": "Vacanza Estiva",  
    "description": "Una splendida vacanza al mare",  
    "startDate": "2024-07-04",  
    "endDate": "2024-07-12",  
    "members": [  
      {  
        "userId": "user123",  
        "role": "CREATOR"  
      },
```

```
{
  "userId": "user456",
  "role": "MEMBER"
}
],
"destinations": [
  {
    "location": "Milan Linate Airport",
    "dateFrom": null,
    "dateTo": null,
    "description": null,
    "lat": 45.4507,
    "lon": 9.2729
  },
  {
    "location": "Catania",
    "dateFrom": "2024-07-04",
    "dateTo": "2024-07-12",
    "description": null,
    "lat": 37.5078836,
    "lon": 15.0830032
  }
]
}
```

2. Collezione `usernames`

La collezione `usernames` mappa gli username degli utenti ai loro ID. Ogni documento ha come nome l'username e contiene i seguenti campi:

- `id`: l'ID dell'utente (stringa)

Esempio di collezione `usernames`:

JavaScript

```
"usernames": {
```

```
"PaoloCisly":  
{  
  "id" : "Eq5wig0ZyyC9EFU10XG0sx0eyFe2"  
}  
}
```

3. Collezione users

La collezione **users** contiene i dettagli degli utenti. Ogni documento ha come nome l'ID dell'utente e contiene i seguenti campi:

- **birthdate**: la data di nascita dell'utente (data)
- **idToken**: il token di autenticazione dell'utente (stringa)
- **email**: l'email dell'utente (stringa)
- **name**: il nome dell'utente (stringa)
- **surname**: il cognome dell'utente (stringa)
- **photoUrl**: l'URL della foto del profilo dell'utente (stringa)
- **travels**: un array di ID dei viaggi a cui l'utente partecipa (array di stringhe)

Esempio di collezione **users**:

```
JavaScript  
"users": {  
  "abcdef123456": {  
    "birthdate": "1990-01-01T00:00:00Z",  
    "idToken": "abcdef123456",  
    "email": "utente@example.com",  
    "name": "Mario",  
    "surname": "Rossi",  
    "photoUrl": "https://example.com/photo.jpg",  
    "travels": ["travel123", "travel456"]  
  }  
}
```

Sviluppi Futuri

Abbiamo pensato a delle possibili funzionalità da aggiungere in un eventuale aggiornamento dell'app:

1. **Condivisione delle esperienze di viaggio:** Implementare una sezione dedicata dove gli utenti possono caricare foto, video e racconti dei loro viaggi. Questa funzione potrebbe includere la possibilità di lasciare recensioni, aggiungere tag geografici e creare album tematici. Gli utenti potrebbero anche seguire altri viaggiatori per trarre ispirazione dalle loro esperienze.
2. **Social Network:** Per rendere l'app più simile a un social network, si potrebbe introdurre un feed di attività, simile a quello di Facebook o Instagram, dove gli utenti possono vedere aggiornamenti dai loro amici e dagli influencer che seguono. Funzionalità come i "like", i commenti e la possibilità di condividere post aumenterebbero l'interazione. Inoltre, si potrebbero creare gruppi di interesse, eventi e chat per permettere agli utenti di connettersi e organizzare viaggi insieme.
3. **Funzionalità avanzate nella mappa:** La mappa potrebbe diventare uno strumento centrale nell'app, offrendo non solo la navigazione, ma anche una serie di funzionalità interattive. Gli utenti potrebbero lasciare segnalazioni sui luoghi visitati, aggiungere consigli e suggerimenti, e visualizzare le recensioni direttamente sulla mappa. Inoltre, la mappa potrebbe mostrare itinerari personalizzati, punti di interesse vicini, eventi locali in corso e la possibilità di salvare i luoghi preferiti. Un'altra utile funzionalità potrebbe essere la mappa offline, per permettere agli utenti di accedere alle informazioni anche senza connessione internet.

Questi sviluppi non solo arricchirebbero l'app di viaggi, ma creerebbero una comunità attiva e partecipativa, migliorando l'esperienza complessiva degli utenti e rendendo l'app un compagno di viaggio indispensabile.