READING STATS
RELAZIONE DI PROGRAMMAZIONE MOBILE

Cristian Di Cintio - S1110150
Federico Di Giovannangelo - S0000000

Contents

1 Introduzione

L'applicazione sviluppata ha come obiettivo la memorizzazione delle statistiche di libri in lettura dell'utente. Tali statistiche riguardano la percentuale di completamento del libro e il tempo impiegato, memorizzato per ciascuno libro.

L'utente per usufruire delle funzionalità dell'app deve registrarsi con nome, cognome, email, username e password. Per lo username e l'email vengono effettuati dei controlli per verificare l'univocità e non permettere la registrazione di utenti con medesimo username o email.

In seguito l'utente può effettuare l'accesso con la propria email e password.

L'utente può ricercare i libri in un catalogo tramite la digitazione del titolo in una barra di ricerca o scansionando, con un pulsante apposito, il codice a barre di un libro fisico. In seguito può decidere di aggiungerlo in tre liste distinte, "da leggere", "in lettura" e "letti", per avere un resoconto delle pagine lette e del tempo impiegato nella lettura dei libri salvati. Il catalogo permette anche il filtraggio di libri tramite la selezione di categorie predefinite.

Al momento della lettura l'utente può avviare un timer per un libro in lettura, presente nell'omonima lista, e disattivarlo al termine della sessione.

L'utente può visualizzare e modificare i propri dati personali e interagire con altri utenti, visualizzando principalmente quali libri hanno salvato e quali stanno leggendo, ognuno con le loro rispettive metriche.

L'intera applicazione è stata sviluppata inizialmente in Kotlin per ambiente Android e in seguito in Flutter per un funzionamento multipiattaforma, in modo tale da permettere una maggiore compatibilità e una migliore interazione tra utenti con diversi dispositivi mobile.

Lo sviluppo in Kotlin è stato applicato con Jetpack Compose, per permettere uno sviluppo moderno e in linea con le attuali applicazioni sul mercato, con una struttura architetturale basata sul Model-View-ViewModel (MVVM) garantendo una composizione comprensibile per lo sviluppo dell'app. Le informazioni principali dei libri vengono visualizzate tramite l'utilizzo di API fornite da Google Books.

2 Progettazione e sviluppo Android

2.1 Requisiti

Si riportano di seguito i requisiti *funzionali* e *non funzionali*

2.1.1 Requisiti funzionali

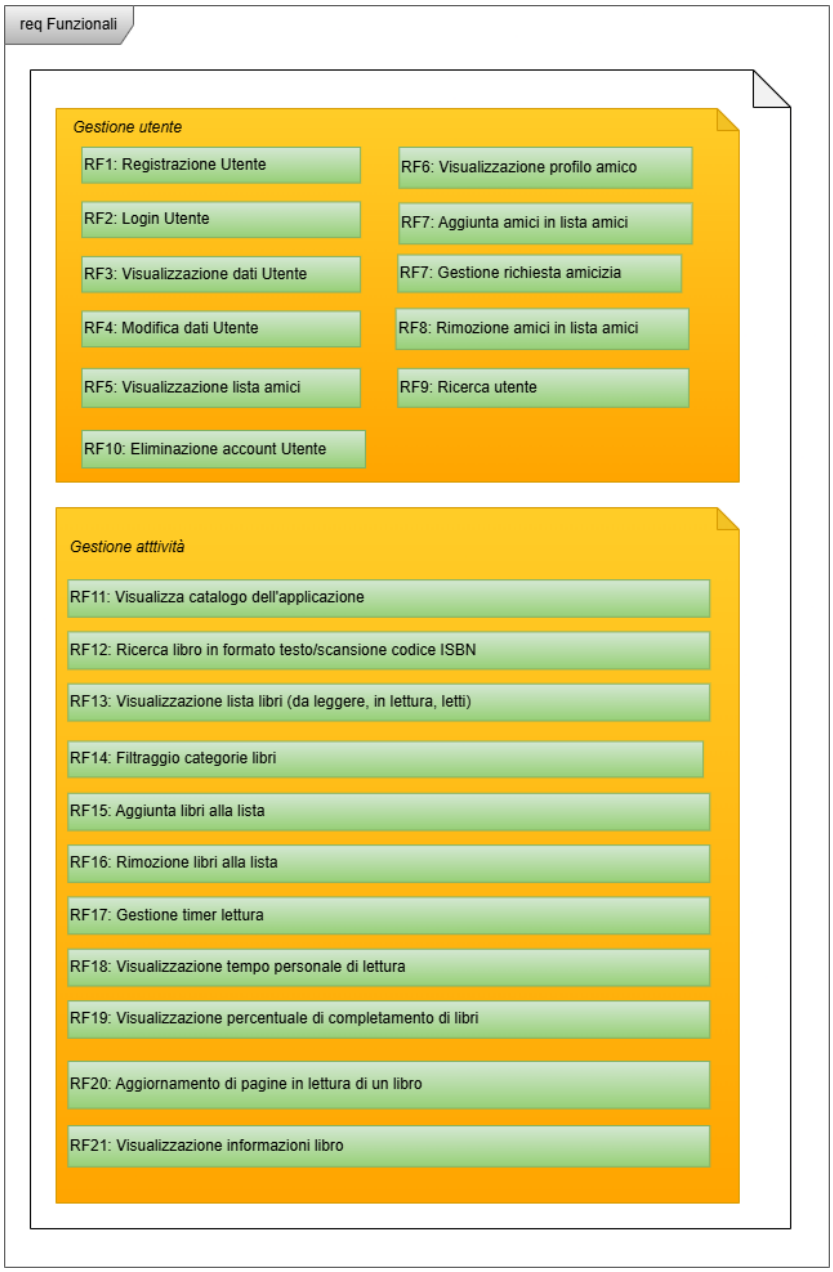

Figure 1: Requisiti funzionali

- *Gestione utente*
  - *RF1: Registrazione utente* - L'applicazione permette di registarsi con le proprie informazioni personali
  - *RF2: Login utente* - L'applicazione permette di autenticare l'utente registrato con le proprie informazioni personali
  - *RF3: Visualizzazione dati utente* - L'applicazione permette di visualizzare i propri dati personali
  - *RF4: Modifica dati utente* - L'applicazione permette di modificare i propri dati personali
  - *RF5: Visualizzazione lista amici* - L'applicazione permette di visualizzare la lista degli amici che hanno accettato la sua richiesta d'amicizia
  - *RF6: Visualizzazione profilo amico* - L'applicazione permette di visualizzare il profilo di un suo amico nella lista
  - *RF7: Aggiunta amici in lista amici* - L'applicazione permette di aggiungere un amico nella sua lista inviando una richiesta
  - *RF8: Gestione richiesta di amicizia* - L'utente può decidere se accettare o rifiutare una richesta di amicizia
  - *RF9: Rimozione amici da lista amici* - L'applicazione permette di rimuovere un amico dalla lista amici
  - *RF10: Ricerca utente* - L'applicazione permette di ricercare un utente digitando il suo username o nome completo
  - *RF11: Eliminazione account utente* - L'applicazione permette all'utente di cancellare il proprio profilo
- *Gestione attività*
  - *RF12: Visualizza catalogo dell'applicazione* - L'applicazione permette di visualizzare il catalogo dei libri distinti in categorie
  - *RF13: Ricerca libro in formato testo o tramite scansione codice a barre* - L'applicazione permette la ricerca del libro digitando testualmente il titolo in una casella di ricerca apposita oppure scansionando il codice a barre di un libro fisico
  - *RF14: Visualizzazione lista libri (da leggere, in lettura, letti)* - L'applicazione permette di visualizzare il contenuto delle liste principali formate dai libri salvati dall'utente
  - *RF15: Filtraggio categorie libri* - L'applicazione permette all'utente di filtrare le categorie dei libri da visualizzare nel catalogo
  - *RF16: Aggiunta libri alla lista* - L'applicazione permette all'utente di aggiungere i libri in una delle apposite liste fornite dall'app
  - *RF17: Rimozione libri dalla lista* - L'applicazione permette la rimozione di libri dalle liste

– *RF18: Gestione timer lettura* - L'applicazione permette di tenere sotto controllo il tempo di lettura di un libro con l'utilizzo di un timer dedicato attivabile e disattivabile dall'utente

– *RF19: Visualizzazione tempo personale di lettura* - L'applicazione permette all'utente di visualizzare per ogni singolo libro il tempo di lettura impiegato con l'utilizzo del timer

– *RF20: Visualizzazione percentuale completamento di libri* - L'applicazione permette di visualizzare la percentuale di completamento di ogni singolo libro, calcolata tramite il numero di pagine lette dall'utente rispetto al numero di pagine totali di un determinato libro

– *RF21: Aggiornamento percentuale di pagine in lettura di un libro* - L'applicazione permette di aggiornare il numero di pagine lette di un libro per calcolare la percentuale di completamento

– *RF22: Visualizzazione informazioni libro* - L'applicazione permette la visualizzazione delle info di un determinato libro

2.1.2 Requisiti non funzionali

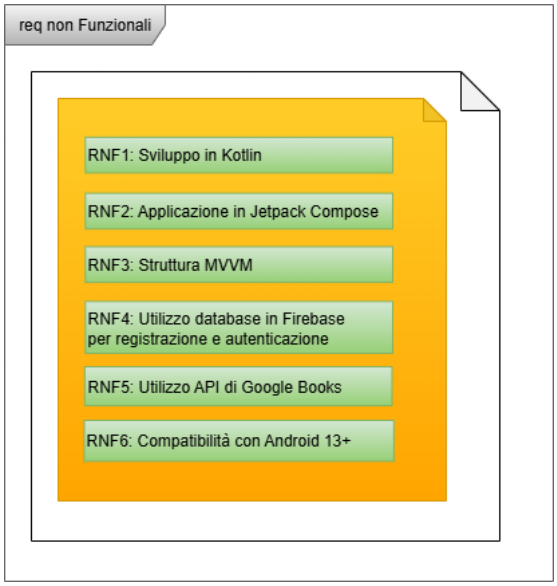

Figure 2: Requisiti non funzionali

– *RNF1: Sviluppo in Kotlin* - L'applicazione è stata sviluppata principalmente con il linguaggio di programmazione Kotlin

– *RNF2: Applicazione in Jetpack Compose* - L'applicazione è stata realizzata con Jetpack Compose per realizzare le interfacce grafiche

- *RNF3: Struttura MVVM* - L'applicazione ha una struttura architetturale basata su Model-View-ViewModel per agevolare la programmazione con sezioni dedicate
- *RNF4: Firebase Firestore per memorizzazione e accesso/autenticazione utente* - L'applicazione fa utilizzo per la memorizzazione di dati e la gestione della registrazione e l'accesso dell'utente di un database tramite la piattaforma Google Firebase
- *RNF5: API di Google Books* - L'applicazione si sincronizza e fa utilizzo delle API Key di Google Books per mostrare i libri nel catalogo e nella ricerca
- *RNF6: compatibilità con Android 13+* - L'applicazione ha una compatibilità con il livello di API 33, permettendone l'utilizzo su android 13 e oltre

2.1.3 Casi d'uso

Si riportano gli attori e i tre casi d'uso principali che descrivono le azioni principali che si possono svolgere nell'applicazione a partire dai requisiti elencati in precedenza.

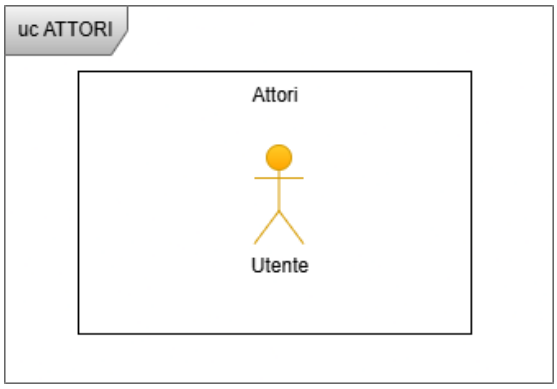

Figure 3: Attori principali

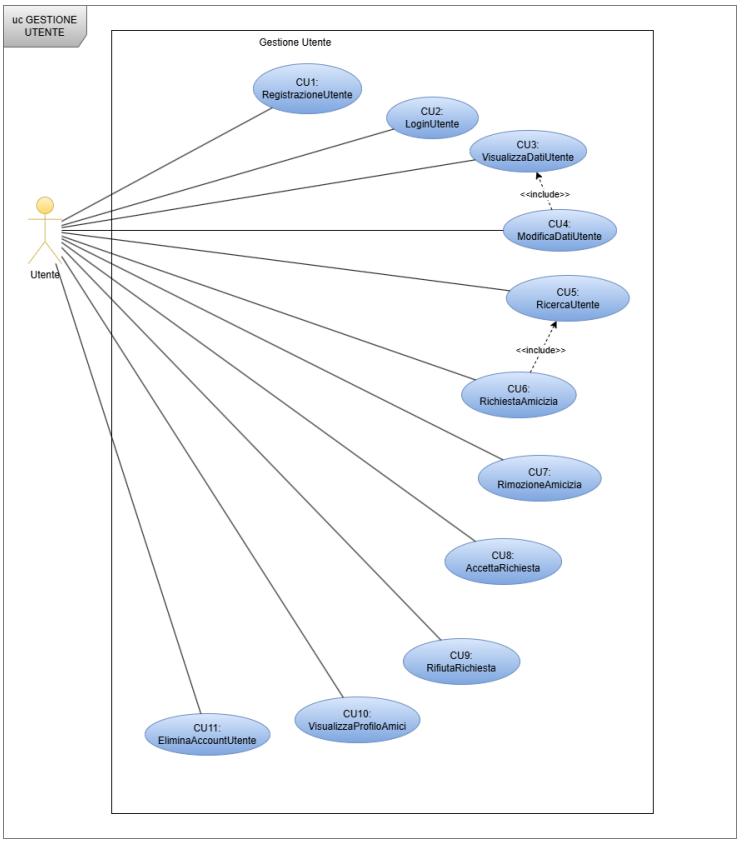

Figure 4: Gestione utente

- *CU1: RegistrazioneUtente* - L'utente, per usufruire dei servizi offerti dall'applicazione, ha bissogno di registarsi con le proprie informazioni personali
- *CU2: LoginUtente* - L'utente può accedere ai servizi forniti inserendo le proprie credenziali (email e password) precedentemente utilizzate per registrarsi
- *CU3: VisualizzaDatiUtente* - L'utente può visualizzare le proprie informazioni personali in una sezione dedicata
- *CU4: ModificaDatiUtente* - L'utente può modificare i propri dati personali
- *CU5: RicercaUtente* - L'utente può ricercare un utente digitando il suo nome utente (usernam) o nome completo (nome e/o cognome)
- *CU6: RichiestaAmicizia* - L'utente può aggiungere amici nella propria lista inviando richieste ad altri utenti

– *CU7: RimozioneAmicizia* - L'utente può rimuovere gli amici dalla propria lista

– *CU8: AccettaRichiesta* - L'utente può accettare la richiesta di amicizia da un altro utente che l'ha inviata

– *CU9: RifiutaRichiesta* - L'utente può rifiutare la richiesta di amicizia da un altro utente che l'ha inviata

– *CU10: VisualizzaProfiloAmici* - L'utente può visualizzare il profilo dei propri amici con le loro statistiche di lettura

– *CU11: EliminaAccountUtente* - L'utente può eliminare il proprio account utente

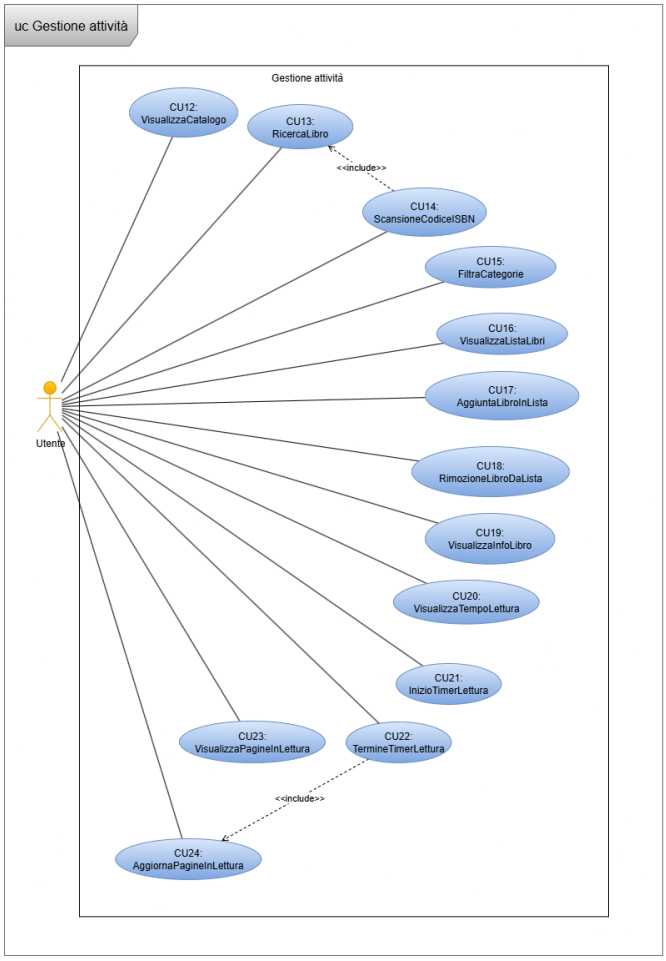

Figure 5: Gestione attività

– *CU12: VisualizzaCatalogo* - L'utente può visualizzare il catalogo di libri separato in categorie

– *CU13: RicercaLibro* - L'utente può ricercare il libro digitando il titolo nella barra di ricerca presente nel catalogo

– *CU14: ScansioneCodiceISBN* - L'utente può effettuare la scansione del codice ISBN di un libro fisico tramite la fotocamera del telefono

– *CU15: FiltroCategorie* - L'utente può filtrare le categorie del catalogo selezionandole da un lista con checkbox

– *CU16: VisualizzaListaLibri* - L'utente può visualizzare una delle tre liste principali (da leggere, in lettura, letti) per memorizzare i libri salvati

– *CU17: AggiuntaLibroInLista* - L'utente può aggiungere un libro selezionato in una delle tre liste principali (da leggere, in lettura, letti)

– *CU18: RimozioneLibroDaLista* - L'utente può rimuovere un libro presente in una delle tre liste principali (da leggere, in lettura, letti)

– *CU19: VisualizzaInfoLibro* - L'utente può visualizzare le informazioni principali che rappresentano il libro selezionandolo dal catalogo o dalle liste

– *CU20: VisualizzaTempoLettura* - L'utente può visualizzare il tempo di lettura di un proprio libro

– *CU21: InizioTimerLettura* - L'utente può iniziare la sessione di lettura di un libro tramite l'inizio di un timer dedicato

– *CU22: TermineTimerLettura* - L'utente può terminare la sessione di lettura di un libro permettendo di fermare il timer

– *CU23: VisualizzaPagineInLettura* - L'utente può visualizzare le pagine in lettura di un libro salvato nell'omonima lista

– *CU24: AggiornaPagineInLettura* - L'utente può aggiornare le pagine

2.2 Architettura applicazione

2.2.1 MVVM

Per la struttura dell'applicazione è stato scelto il pattern architetturale Model-View-ViewModel (MVVM) per ottenere una separazione netta delle responsabilità tale da rendere il codice scalabile e facilmente aggiornabile. Le sue principali componenti sono:

- **Model:** Gestisce i dati dell'applicazione e della loro persistenza, principalmente in remoto con Firebase nel caso specifico di questa applicazione, permettendo operazioni di accesso, modifica e salvataggio dei dati per gli *utenti* che si registrano e dei *libri* che salvano in determinate liste.

- **View:** Rappresenta le interfacce dell'applicazione per permettere all'utente di interagire con la logica dell'applicazione. Vengono definite tramite funzioni con la notazione iniziale *@Composable* e possono essere richiamate nel file contenente il MainActivity o in altre funzioni con la stessa notazione.
- **ViewModel:** Funge da intermediario tra View e Model e permette il cambiamento dello stato della UI con le interazioni dell'utente tramite *LiveData*

2.2.2 Diagramma delle componenti

Il seguente diagramma di componenti rappresenta le principali istanze e le loro interazioni, in particolare tra l'utente e un singolo libro scelto, la lista dei libri e la richiesta di amicizia verso un altro utente. Per il libro l'utente può decidere se inserirlo in una lista o rimuoverlo, mentre per la richiesta l'utente può inviarne una e visualizzarne una ricevuta, per poi decidere se accettarla o rifiutarla.

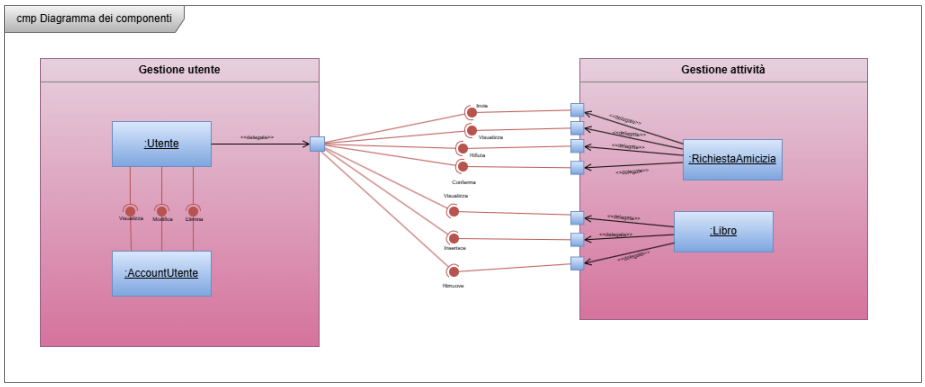

Figure 6: Diagramma delle componenti

2.3 Gestione Database - Firebase

In presenza di un sistema di autenticazione e registrazione a un servizio online, con la possibilità per l'utente di interagire con altri utenti, e permettere il salvataggio personale di libri nel proprio profilo, l'infrastruttura migliore per manipolare questi dati risulta essere il database gestionale Firestore di Google Firebase.

Firestore è un DB NoSQL strutturato a documenti, il quale permette il salvataggio di dati in documenti raggrupati in collezioni e può contenere sottocollezioni e campi annidati. Tali documenti e collezioni vengono generati con la scrittura di dati tramite codice.

Il suo utilizzo si concentra su due componenti della sezione *Creazione*:

- *Authentication:* permette di gestire il metodo di accesso degli utenti tramite diversi servizi. Nel caso di questo progetto abbiamo implementato esclusivamente l'autenticazione semplice tramite email e password.

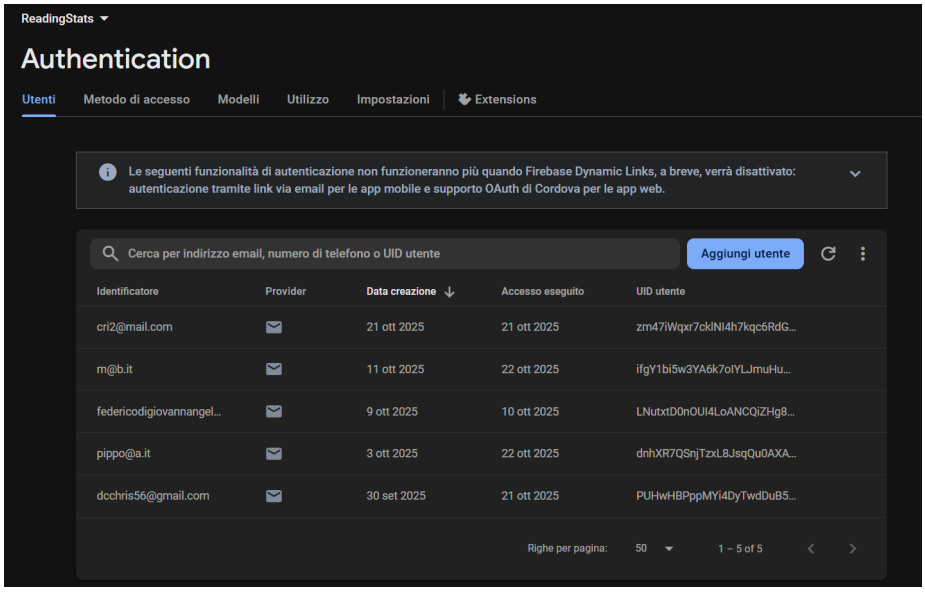

Figure 7: Diagramma delle componenti

- *Firestore Database:* permette la memorizzazione tramite documenti e collezioni descritti in precedenza dei dati dell'utente, delle richieste di amicizia inviate tra utenti e dei libri salvati con ognuno la relativa sessione. Sono state implementate delle regole specifiche per permettere un corretto e coerente accesso ai dati e la loro relativa modifica. Di seguito i documenti e le collezioni generate:

  INSERIRE SCREEN PER OGNI SEZIONE

  - **Salvataggio utenti:** Utenti registrati e salvati nel servizio tramite il form di registrazione, memorizzando nome, cognome, username e mail;
  - **Salvataggio usernames:** Username salvati al momento della registrazione per permettere un controllo di univocità;
  - **Salvataggio di amici tramite richieste:** Lista di utenti che si aggiorna per ogni singolo utente quando accetta una richiesta di amicizia da un altro utente;
  - **Salvataggio di libri dal catalogo:** Raccolta di libri salvati per ogni utente, ognuno con le sue informazioni e lo stato per visualizzarlo in una determinata lista tra quelle fornite;

– **Salvataggio sessione di lettura:** Per ogni libro nello stato "In lettura", viene memorizzata una sessione di lettura contenente il tempo memorizzato in secondi e convertito a schermo in ore:minuti:secondi;
– **Regole di accesso ai documenti definiti:** Le seguenti regole scritte nell'omonima sotto-sezione permettono la definizione di un accesso;

2.4 API utilizzate - Google Books

Le API di Google Books sono state utilizzata per ottenere le informazioni principali dei libri, in modo da permettere all'utente la loro visualizzazione e il conseguente salvataggio. L'API viene fornita tramite un link per le ricerche e una chiave generata da Google Cloud per permetterne l'abilitazione e la limitazione del tipo di dispositivo che la utilizzerà.

Link: https://www.googleapis.com/books/v1/volumes?q=search-terms.

"q" è il parametro da inserire per effettuare la ricerca tramite titolo o autore. Sono presenti altri parametri, i cui principali per l'applicazione sono stati:

- *maxResults*, per mostrare la quantità di libri da visualizzare, utilizzato in particolare nel catalogo e nella ricerca.
- *orderBy*, il quale può essere relevance o newest per mostrare libri di maggiore rilievo tramite parametriche di Google o nuove uscite.
- *langRestrict*, ossia la restrizione dei libri da visualizzare e ricercare con la una lingua specifica.

2.5 Sviluppo

Come anticipato in precedenza, l'applicazione è interamente sviluppata in **Jetpack Compose**, un toolkit dichiarativo di Android che permette, al variare del proprio stato, di ridisegnare le parti necessarie al cambiamento di tale stato.

Permette un approccio dichiarativo tramite la definizione di funzioni con notazione *@Composable*, le quali permettono di definire parti di interfaccia utente esclusivamente con codice Kotlin. In queste funzioni si controlla lo stato per permettere l'aggiornamento di determinate parti coinvolte.

Sono disponibili una serie di layout predefiniti (Column, Row, Box) per strutturare al meglio una componente dell'interfaccia e le funzioni accettano un parametro Modifier che permette di modificarne l'aspetto estetico.

Per la costruzione di un'interfaccia tradizionale è stato utilizzata la funzione *Scaffold* con parametri per definire componenti predefinite come TopAppBar, BottomAppBar, FloatingActionButton e altri ancora.

Si possono distinguere due tipi di funzioni Composable:

- **Stateful**, fa utilizzo della keyword *remember* che permette di memorizzare dati tra diverse recomposition
- **Steless**, ossia un Composable orivo di stato

Le liste scorrevoli che visualizzano molti elementi vengono gestite da componenti *Lazy*, che forniscono un blocco per descrivere i singoli *item* da visualizzare. Ne esistono diverse versioni:

- **LazyRow** per lo scorrimento orizzontale,
- **LazyColumn** per lo scorrimento verticale,
- **LazyVerticalGrid e LazyHorizontalGrid** per lo scorrimento di una griglia rispettivamente verticale e orizzontale

La navigazione viene principalmente gestita tramite un file dedicato, denominato **AppNavHost.kt**, nel quale viene creato un *NavController* e un *NavHost*, il quale fa uso della funzione *composable* che permette di definire le rotte di destinazione all'interno dell'applicazione. Tali rotte vengono definite in formato di stringa in un altro file denominato **Routes.kt**

Le funzioni con notazione *@Composable* possono essere richiamate da un Activity o Fragment, da altre funzioni *@Composable*, all'interno di una destinazione definita in Navigation-Compose e in liste Lazy.

2.5.1 Struttura package

Il progetto, strutturato con architettura MVVM, è stato suddiviso in macro-package al cui interno sono presenti file o altri package caratteristici.

La struttura delle macro-cartelle risulta la seguente:

- **core:** contiene i package con i file di parti di interfaccia *@Composable* riutilizzabili nel codice e i file utilizzati per la personalizzazione globale dell'app.
- **di:** moduli di *Dependency Injection* (Hilt) che descrivono come creare e fornire le dipendenze dell'app. Le classi ricevono gli oggetti necessari tramite iniezione, senza istanziarli al proprio interno. Contiene i file usati a livello globale per integrare le Google Books API e configurare lo stack di rete.
- **features:** è presente il codice principale dell'applicazione scritto in diversi file distinti e gestito in package annidati per la loro funzionalità nell'applicazione.
- **navigation:** contiene i file di gestione della navigazione e di creazione delle route