Mobile Programming 2024/25

Gruppo 11

Gaetano Carbone	0612707663
Mario Pellegrino Ambrosone	0612707417
Michele Cetraro	0612707419
Simone Grimaldi	0612707338
Pasquale Rodia	0612708053

Contents

L'IDEA

TomoTomo nasce come risposta alla traccia progettuale del corso di *Mobile Programming*, con l'obiettivo di sviluppare un'app mobile multipiattaforma dedicata alla gestione della propria libreria personale.
Il progetto è pensato per migliorare l'organizzazione e il tracciamento delle letture, rivolgendosi a lettori di ogni tipo che desiderano catalogare, consultare e valutare i propri libri in modo semplice, intuitivo e accessibile.

Il problema. Ricordare cosa si sta leggendo, cosa si è già letto, e cosa si vorrebbe leggere. Una difficoltà spesso sottovalutata, soprattutto quando la propria libreria cresce.

La soluzione. **TomoTomo**: la tua libreria, sempre con te. Digitale. Personalizzabile. Unica.

PROTOTIPAZIONE

Use Case Diagram

In Figure 1 è mostrato il diagramma dei principali casi d'uso; per brevità si evita la discussione dei flussi alternativi per ciascun caso d'uso e l'eccessiva formalizzazione di ognuno di quelli diagrammati.
Quanto segue è quindi un unico flusso organico di eventi che un utente potrebbe eseguire interagendo con l'app e che ha come fine il chiarimento del diagramma.
L'utente, a touch sull'icona **TomoTomo** dalla Home Page del proprio mobile device, visualizza la propria libreria.
Dalla schermata in cui si trova può visualizzare i dettagli di ognuno dei libri in libreria, modificarne i dettagli o eliminare il libro, dove per modifica si intende anche la sola aggiunta di una recensione o l'aggiunta del libro in modifica ai preferiti.
L'utente può inoltre visualizzare il proprio profilo, ricercare un libro o aggiungerne uno nuovo.

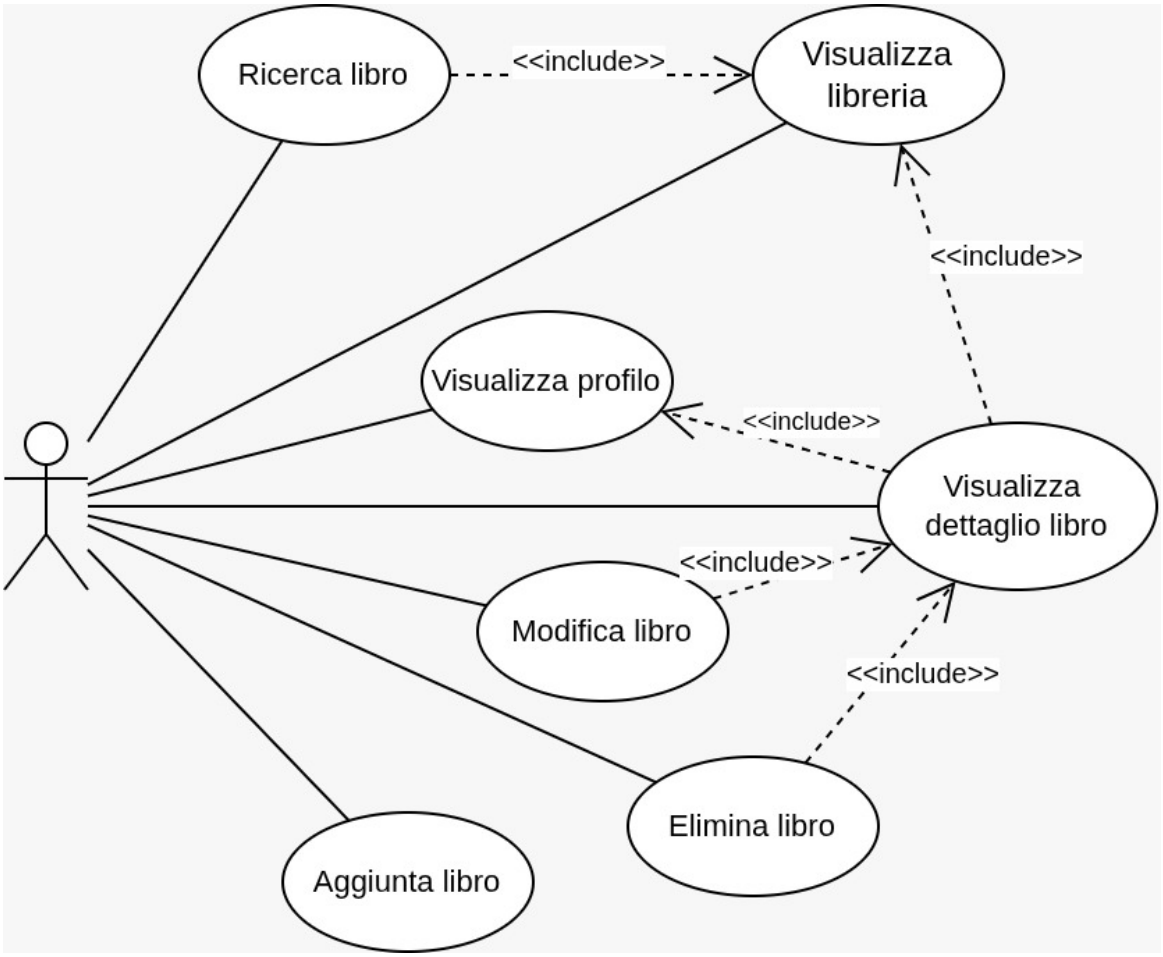

Figure 1: Diagramma dei principali casi d'uso

Wireframe e Wireflow

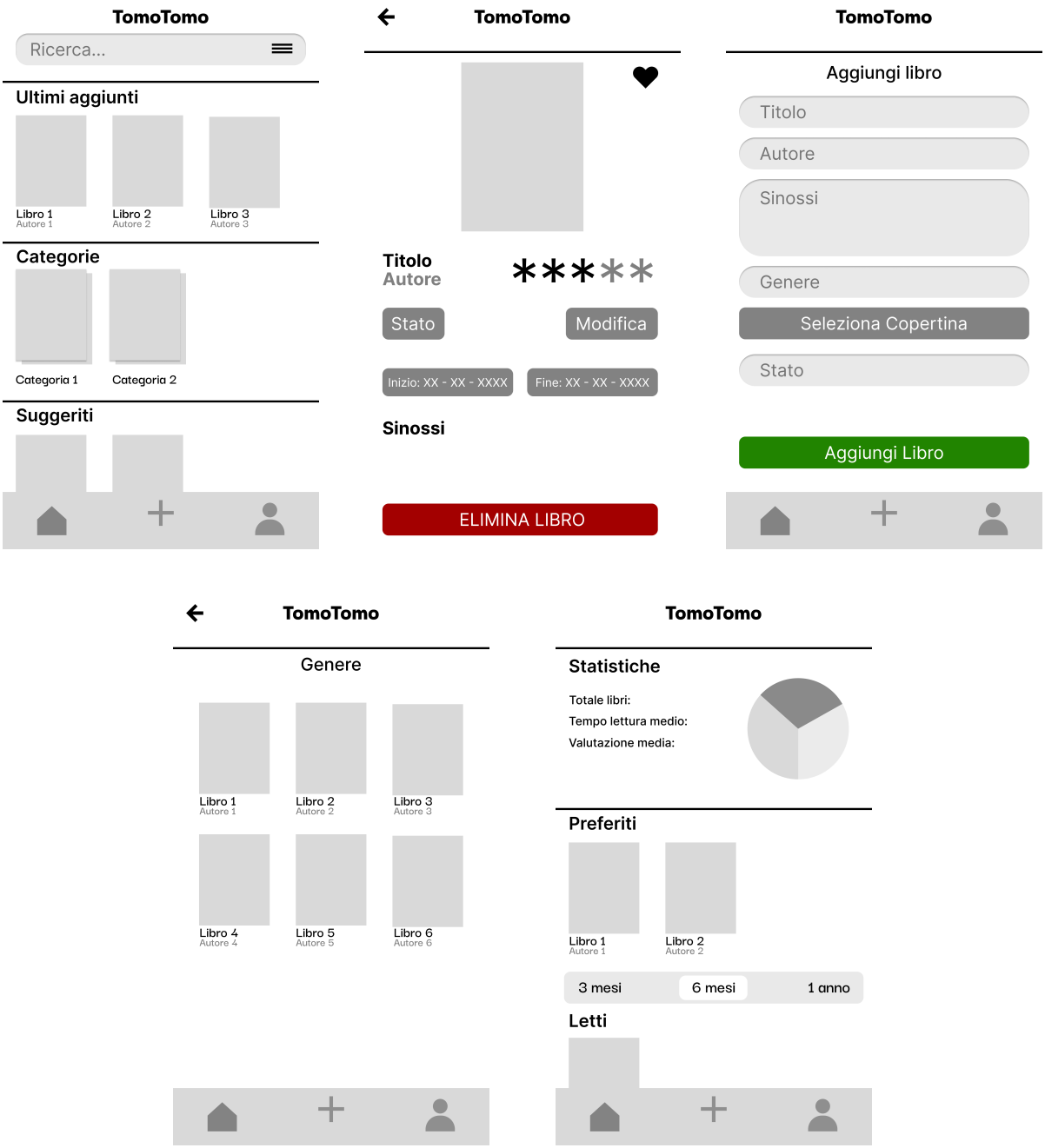

Figure 2: Wireframe di **TomoTomo**. In ordine, da sinistra a destra e poi in basso: *Home Screen*, *Detail Screen*, *Add Screen*, *Genre Screen* e *Profile Screen*

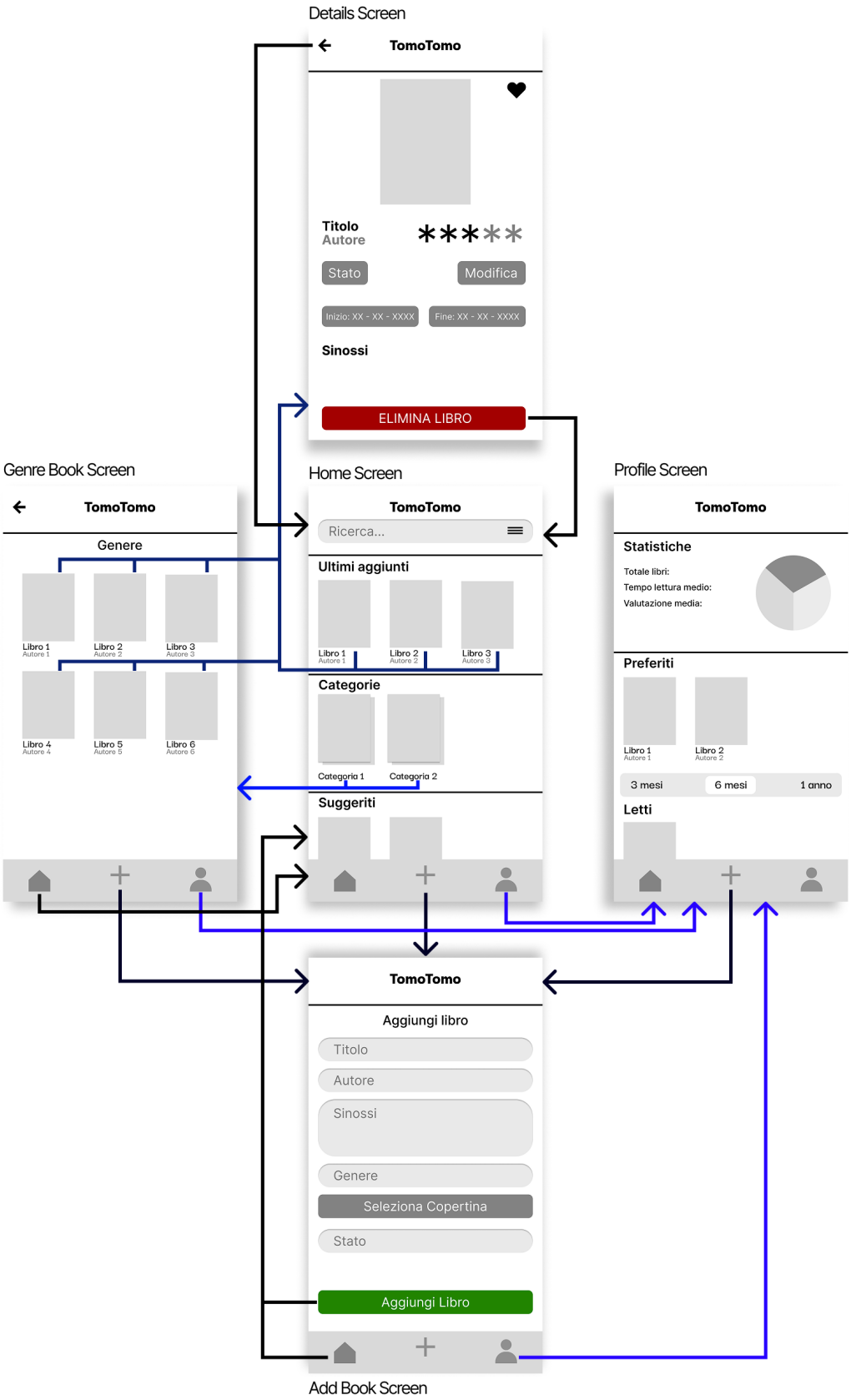

Figure 3: Wireflow di **TomoTomo**

REQUIREMENTS ANALYSIS: MUST HAVE

Home Screen

La *Home Screen* rappresenta il punto di ingresso principale dell'applicazione ed è accessibile da ogni schermata tramite il sistema di navigazione. La sua architettura interna è stata pensata per essere modulare, suddividendo quindi la schermata in più sezioni scrollabili che facilitano la fruizione da parte dell'utente.
Da un technical POV la schermata è implementata come componente funzionale React che sfrutta diversi hook per gestire lo stato locale e globale. Nello specifico, la lista complessiva dei libri è condivisa tramite il `BooksContext`, che centralizza e rende accessibile lo stato dati a tutti i componenti interessati.
Lo `Screen` si divide in:

- Section dei libri aggiunti più di recente; viene mostrata la foto copertina degli ultimi tre libri aggiunti.
- Section dei libri già aggiunti e divisi per categoria; viene mostrata la foto copertina dell'ultimo libro aggiunto per ogni fissata categoria. Al touch dell'icona si viene reindirizzati alla *Genre Screen* (vedi sez. - Le Nostre Features) rispetto alla specifica categoria cui appartiene il libro mostrato in icona.
  È importante chiarire che per categoria si intende il principale genere di apprtenenza del libro.
- Section dei suggerimenti casuali di letture.

Nell'header, la schermata integra inoltre una barra superiore contenente il logo e una inferiore con i controlli di navigazione, mentre il layout utilizza componenti di `SafeAreaView` per assicurare la corretta visualizzazione anche in presenza di notch o altre particolarità hardware.
Per quanto concerne lo stile poi, l'uso di `StyleSheet` permette di mantenere un'attenzione particolare al posizionamento degli elementi e alla gestione delle interazioni touch.
La *Home Screen* presenta inoltre una barra di ricerca ed un hamburger menu, posizionati nell'header della schermata, che permettono di effettuare ricerche tra i libri già aggiunti. Le funzionalità più complesse legate alla ricerca e al filtraggio dei libri saranno approfondite poi(vedi sez. - Le Nostre Features).
Infine, l'accesso rapido all'aggiunta di nuovi libri, alla *Profile Screen*, e alla *Home Screen* stessa, sono feature contenute all'interno di un Tab Navigator nel footer di ogni schermata dell'app, tranne che nella *Detail Screen*.

Detail Screen

Al touch di una qualsiasi icona - copertina di un libro già aggiunto si viene sempre reindirizzati alla schermata *Detail Screen*; quest'ultima presenta il titolo del libro, la sua foto copertina, la trama (sinossi), lo stato di lettura (con associate date, se presenti), e infine, se il libro è in stato "Letto", un rating a stelle e le note personali.
La *Detail Screen* si fonde con la pagina di modifica, permettendo l'edit delle informazioni, relative al libro in visualizzazione, mediante un button.
La modifica non è "libera"; le policy che regolano i possibili valori dei vari dati associati ad ogni libro sono le stesse dell'aggiunta (vedi subsez. - *Add Screen*).

Dal punto di vista implementativo, il componente `BookDetailScreen` è sviluppato come componente funzionale React che sfrutta diversi hook per gestire lo stato e la sincronizzazione dei dati. Lo stato globale dei libri è condiviso tramite `BooksContext`, proprio come la *Home Screen*; la modifica è poi controllata tramite lo stato booleano `isEditing`, che se è alto rende i campi testo statici (titolo, autore, trama) dei `TextInput` controllati, il cui valore è legato a `editedBook`. Prima della conferma della modifica, il metodo `handleConfirmEdit` esegue controlli rigidi sulle date di lettura (`date_start`, `date_end`), sullo stato di lettura e sulla correttezza temporale, prevenendo inserimenti incoerenti tramite alert di errore. Questo riflette le regole di business condivise con la schermata di aggiunta libro, approfondite nella sezione di seguito. È da notare che, anche se la feature è fuori dai requisiti MH, nella barra superiore è incluso un button per segnare un libro come preferito, con toggle dello stato `favorite` aggiornato in modo asincrono.
Anche per questa schermata lo stile è gestito tramite `StyleSheet`.

Add Screen

La *Add Screen*, accessibile tramite il Tab Navigator presente nel footer dell'app, è stata progettata come un form completo e dinamico che permette all'utente di inserire un nuovo libro nel proprio catalogo personale.
Dal punto di vista implementativo, il componente funzionale React utilizza una serie di hook `useState` per mantenere i valori di ogni campo in modo controllato e reattivo: titolo, autore, trama(sinossi), genere, immagine di copertina, stato di lettura, valutazione a stelle, note personali e date associate alla lettura.
Tra le caratteristiche principali è da notare l'integrazione con il modulo `expo-image-picker` per consentire la selezione di un'immagine di copertina dalla galleria del dispositivo. La funzione `pickImage` gestisce i permessi necessari, l'apertura della libreria e la copia del file selezionato in una cartella dedicata all'interno della document directory, generando un nome sicuro e univoco per il file (l'univocità è garantita dal metodo `sanitizeFileName`, il quale concatena al filename la stringa timestamp del momento in cui il metodo viene invocato).
Il campo `stato` poi è gestito tramite un picker con tre opzioni principali: "Da leggere", "In lettura" e "Letto". A seconda del valore selezionato, il form adatta dinamicamente la visualizzazione degli input per le date di inizio e fine lettura, implementate attraverso componenti `DatePickerEdit`. In particolare:

- Se lo stato è "In lettura" è possibile e obbligatorio inserire solo la data di inizio lettura.

- Se lo stato è "Letto" è possibile e obbligatorio inserire sia la data di inzio che di fine lettura.

- Se lo stato è "Da leggere" non è possibile inserire date.

Altra policy da chiarire è quella sul management delle note personali e dei rating a stelle, e quindi, più in generale, sui possibili valori dei campi di compilazione del form. I vincoli, citati anche nella *Detail Screen*, sono riassumibili come segue:

- Titolo, autore, trama, stato, genere e immagine sono parametri obbligatori.

- Stato è di default a valore "Da leggere", ma può essere modificato a scelta tra: "Letto" o "In lettura".

- Rating a stelle e Note personali sono consentiti solo se lo stato è ”Letto”.

Prima della conferma dell'inserimento quindi, la funzione `handleSubmit` esegue i controlli sui campi obbligatori e sulle date, assicurandosi che le policy siano rispettate. In caso di errori, vengono mostrati alert informativi all'utente, mentre in caso di successo il libro viene aggiunto tramite il metodo `addBook` fornito dal `BooksContext` e l'utente viene riportato automaticamente alla schermata principale.
Il layout invece si sostanzia di: un'area scrollabile contenente input testuali, picker e pulsanti, integrata con un `KeyboardAvoidingView` per assicurare la visibilità dei campi anche con tastiera attiva.

LE NOSTRE FEATURES: I NICE TO HAVE

Genre Screen

L'accesso alla schermata *Genre Screen* è possibile esclusivamente dalla seconda sezione della *Home Screen*, denominata ”Playlist per genere”. Qui, l'utente può selezionare una delle icone rappresentanti i generi disponibili, attivando la navigazione verso questa schermata che si adatta dinamicamente al genere selezionato.
Dal punto di vista tecnico, la schermata è implementata come componente funzionale React Native che utilizza il hook `useRoute` per recuperare il parametro `genreName` passato dalla schermata precedente. Questo parametro determina il filtro applicato sulla lista globale di libri.
Il filtraggio è realizzato tramite la funzione `Array.filter`, che estrae solo i libri il cui campo `genre` corrisponde esattamente al genere selezionato, garantendo così un sottoinsieme coerente e rilevante di libri da visualizzare. Da un punto di vista strettamente pragmatico, la schermata è la stessa che appare quando vengono filtrate le ricerche.
L'interfaccia utente si compone di un `FlatList` configurato con una griglia a tre colonne in cui ogni elemento è un `TouchableOpacity` che mostra l'immagine di copertina (se presente, altrimenti un'immagine di default), il titolo e l'autore del libro, e consente la navigazione verso la schermata di dettaglio specifica per quel libro passando il suo `id` come parametro. Anche in questo `Screen` c'è un pulsante per il ritorno alla pagina precedente del `Navigator`.
Nel caso in cui non siano presenti libri per il genere selezionato, viene mostrato un messaggio esplicito di empty state, gestito tramite un semplice controllo condizionale, migliorando l'usabilità e comunicando chiaramente lo stato dell'app all'utente.

Ricerca

Agendo secondo i medesimi principi alla base della *Genre Screen*, la *Home Screen* offre all'utente la possibilità di filtrare e visualizzare libri in base a criteri multipli, combinando una barra di ricerca testuale e un menu hamburger per la selezione dei filtri.
La barra di ricerca consente di effettuare ricerche dinamiche su due campi, ovvero il titolo e l'autore del libro tramite un filtro applicato sull'array globale dei libri secondo gli stessi canoni già esplicitati nel paragrafo precedente.
Parallelamente, l'hamburger menu permette all'utente di attivare filtri, gestiti attraverso uno stato locale che mantiene informazioni dettagliate sulle preferenze attive, relativi allo stato di lettura (da leggere, in lettura, letto) e alla valutazione a stelle. Lo stato è

quindi aggiornato in modo atomico tramite le funzioni `toggleFilter` e `resetFilters`.
L'integrazione tra ricerca testuale e filtri viene infine orchestrata tramite l'hook `useMemo`, che memorizza il risultato delle operazioni di filtraggio mentre la schermata rende dinamicamente visibile la lista dei libri risultanti.

Profile Screen

La *Profile Screen*, accessibile esclusivamente tramite il Tab Navigator, attivabile con un semplice tocco sull'icona dedicata, è progettata per offrire all'utente una panoramica dettagliata della propria attività di lettura e delle statistiche correlate alla propria libreria digitale.
L'interfaccia si articola in due sezioni principali: *Statistiche* e *La tua libreria*. La sezione *Statistiche* raccoglie indicatori aggregati quali il totale dei libri presenti nella libreria, la media dei giorni di lettura, la valutazione media dei libri completati, il numero di libri attualmente in lettura e un diagramma a torta che rappresenta la distribuzione dei generi letterari all'interno dei libri già aggiunti.
Tali dati vengono calcolati dinamicamente utilizzando metodi di aggregazione sui dati disponibili nel contesto globale dei libri (`BooksContext`). In particolare:

- Il filtraggio dei libri in base allo stato di lettura è realizzato con il già discusso filtro sull'array globale, estraendo rispettivamente i libri con stato "Letto", "In lettura" e "Da leggere".
- La media delle valutazioni è calcolata come la somma dei rating divisa per il numero di libri letti, mentre la media dei giorni di lettura viene ottenuta sottraendo le date di inizio e fine lettura per ciascun libro e calcolando una media ponderata.
- La composizione del diagramma a torta per i generi è generata aggregando i libri letti per categoria, e associando ad ogni categoria un colore generato dinamicamente tramite la funzione `hsl`, assicurando un effetto visivo distintivo e coerente.

La seconda sezione, *La tua libreria*, è suddivisa in sottosezioni che presentano i libri raggruppati per stato di lettura: *Libri letti*, *Da leggere* e *In lettura*. Ogni sottosezione utilizza il componente `BookCarousel`, che permette una visualizzazione orizzontale a carosello delle copertine dei libri, con la possibilità di accedere al dettaglio di ciascun libro tramite touch, navigando alla schermata *Detail Screen*.
Qualora siano presenti, viene inoltre mostrata una lista dedicata ai libri contrassegnati come preferiti, rafforzando l'esperienza utente attraverso la personalizzazione e facilitando l'accesso rapido ai contenuti di maggior interesse.
Dal punto di vista tecnico, la schermata gestisce il layout tramite `ScrollView` contenente tutte le sezioni, assicurando una navigazione fluida e reattiva nel render. In sintesi, la *Profile Screen* rappresenta un cruscotto completo per il monitoraggio dell'attività di lettura personale, combinando dati aggregati, visualizzazioni grafiche e accesso rapido ai contenuti.

LA TOMOTOMO ARCHITECTURE

Il progetto **TomoTomo** poggia su di un'architettura progettata per garantire modularità, manutenibilità e facilità di estensione. In accordo ai principi di buon design fondamentali (i.e. SRP - Single Responsability Principle e KIS - Keep It Simple), il design è stato guidato da una forte attenzione allo sviluppo dei requisiti fondamentali, con poche features aggiuntive e con un grande interesse nello sviluppo di un product quanto più user-oriented possibile.

La User Experience e il Design

L'interfaccia utente di **TomoTomo** è stata progettata al fine di minimizzare la frustrazione dell'utente, garantendo un'esperienza intuitiva e accessibile.
Le schermate principali dell'app includono la *Home Screen*, la *Detail Screen*, dalla quale è possibile modificarne o cancellarne i dati, e un modulo *Add Screen* dedicato all'aggiunta dei libri.
La navigazione all'interno dell'app è stata implementata utilizzando React Navigation con un Navigator annidato, il quale permette un passaggio fluido tra le schermate e una gestione efficiente dello storico delle visualizzazioni. La scelta di un Navigator innestato, seppur l'utente interagisca sempre e solo con il Tab Navigator, è a vantaggio di una maggiore navigabilità nelle schermate più complesse, quali Home e Profile; quest'ultime sono infatti a loro volta degli Stack Navigator (`HomeStackNavigator` e `ProfileStackNavigator`) utili nella gestione dei propri set di schermate.
L'interattività è assicurata dall'uso di componenti touchabili, input dinamici, selettori e un sistema di valutazione a stelle, che insieme contribuiscono a rendere l'utilizzo naturale e moderno.
Particolare attenzione è stata dedicata alla responsività dell'interfaccia, che si adatta automaticamente a diverse dimensioni di schermo, garantendo un'ottima usabilità, in sola portrait mode, sia su smartphone che su tablet.
Infine, lo stile grafico è caratterizzato da un design pulito e minimalista, con cura nell'uso delle spaziature e nella gerarchia visiva degli elementi, nonché un caricamento efficiente delle immagini delle copertine, per ottimizzare la fruizione da parte dell'utente.

Testing e Deployment

Durante lo sviluppo sono stati effettuati test manuali su emulatori Android e iOS, nonché su dispositivi reali tramite Expo Go. La scelta di Expo ha facilitato il testing continuo grazie agli aggiornamenti live e ha semplificato il deployment iniziale.
Nello specifico, per il testing e il deployment, sono state seguite le seguenti strategie e tecnologie:

- Expo CLI: per sviluppo, testing e build, offrendo strumenti integrati per il debug e la distribuzione.

- Test funzionali manuali: verifica delle funzionalità CRUD (Create, Read, Update, Delete) sui libri, gestione delle categorie e ricerca.

L'app può essere quindi avviata facilmente tramite il comando `npx expo start`, permettendo l'accesso su device reali tramite QR code, come esplicitato nel README.

TomoTomo è inoltre cross-platform e modulare, grazie all'utilizzo di:

- React Native: framework principale che consente di scrivere codice JavaScript/TypeScript per interfacce native su Android e iOS.

- Expo: piattaforma per la gestione semplificata dell'ambiente di sviluppo, testing e build.

- AsyncStorage: soluzione per la persistenza locale dei dati, utilizzata per salvare informazioni sui libri senza dipendere da un backend esterno.

- React Navigation: gestione della navigazione tramite tab e stack, con passaggio di parametri e gestione dello storico.

- Componenti custom e librerie di terze parti: per elementi UI come rating a stelle, picker e gestione immagini.

Struttura del Progetto

Da un punto di vista architetturale, **TomoTomo** è suddiviso in:

- `assets/`: contiene tutte le risorse statiche del progetto, come icone favicon , file .png associati all'identificazione degli screen da Tab Navigator e immagini di deafult per rollback.

- `components/`: raccoglie i componenti React riutilizzabili e indipendenti, come ad esempio la `SearchBar.js`, pulsanti personalizzati, widget di rating a stelle, `BottomBar.js`. Questi componenti favoriscono la coerenza stilistica e funzionale in tutta l'app.

- `screens/`: include le schermate principali dell'applicazione, ognuna rappresentata da un componente React dedicato. Ad esempio, troviamo la schermata *Home Screen*, *Detail Screen*, *Add Screen*, ecc...
Questa separazione facilita la navigazione e l'organizzazione logica del flusso dell'app.
Si nota che la schermata `SplashScreen.js` (vedi Figure 4) contenuta in questa directory è la schermata utilizzata per l'animazione a start dell'app.

- `navigation/`: contiene la configurazione della navigazione tra schermate, implementata con la libreria React Navigation. Qui si definiscono gli stack navigator, le tab bar, e le relative rotte, permettendo un passaggio fluido e strutturato tra le diverse parti dell'app.

- `context/`: raccoglie i provider e i context React per la gestione globale dello stato condiviso, permettendo una comunicazione efficace tra componenti lontani nella gerarchia e facilitando il mantenimento dello stato applicativo centralizzato.

- `services/`: contiene moduli dedicati alla logica di accesso ai dati in un unico file `Storage.js`.

Segue la schematizzazione completa, riportata in Figure 4.

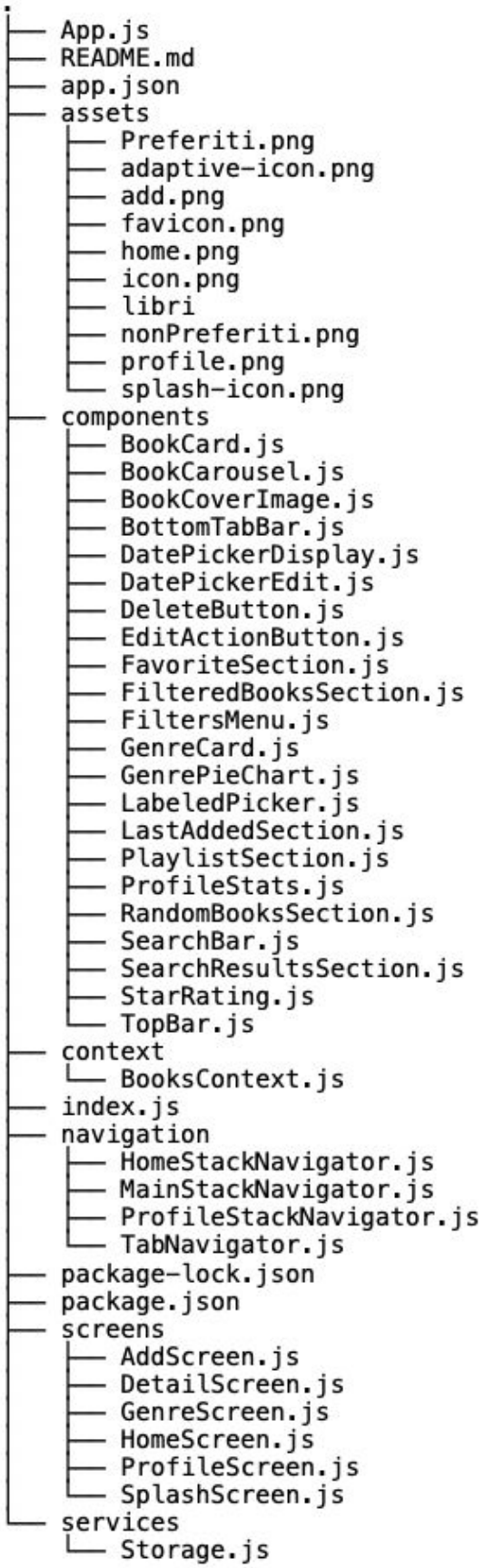

Figure 4: Directory Tree di **TomoTomo**

CONCLUSIONI E SVILUPPI FUTURI

Quality Analysis

La **TomoTomo** Architecture ha molteplici punti di forza che ne garantiscono la validità come base di un progetto potenzialmente molto più vasto.
La struttura modulare, l'intuitività della User Interface e le tecnologie utilizzate rendono **TomoTomo** semplice nella manutenzione e anche nell'estensione. Lato provider, la separazione delle responsabilità dei componenti software favorisce una buona organizzazione del codice; lato client poi, la UX solida e la compatibilità multipiattaforma suggeriscono un possibile discreto posizionamento nel mercato delle Mobile app, soprattutto grazie all'estetica friendly e minimal dell'interfaccia e ad una persistente brand identity negli elementi di design in app. Nella sua forma attuale, il progetto **TomoTomo** si configura come un MVP (Minimum Viable Product) funzionalmente completo, in grado di coprire le operazioni fondamentali per la gestione di una semplice libreria personale idonea solo all'utilizzo individuale e in modalità offline.
Chiaramente, nell'interpretazione del progetto come un long-term project, è necessario valutare alcune criticità, come ad esempio:

- *Assenza di backend*: attualmente l'app funziona esclusivamente in modalità offline; non è prevista sincronizzazione cloud o accesso da più dispositivi.

- *Persistenza scalabile limitata*: la tecnologia `AsyncStorage`, seppur a vantaggio di una facile gestione delle sessioni utente, non sembra adatta a grandi volumi di dati.

- *Assenza di automatic testing practices*: non sono implementati test unitari, di integrazione o end-to-end.

W.I.P.

La base tecnica di cui **TomoTomo** attualmente dispone può essere considerata solida per una prima release, ma per garantire la scalabilità, la resilienza e la manutenibilità del progetto sarebbe necessario pianificare interventi mirati sul rafforzamento dell'infrastruttura, sull'introduzione di uno stato globale condiviso e sul miglioramento della qualità del codice.
Alcune funzionalità per eventuali rel., restando nell'ipotesi di un progetto a lungo termine e che mira a costruire una Mobile App scalabile e distribuita, potrebbero essere:

- Esperienza utente avanzata: introduzione di backup locale esportabile/importabile e supporto a notifiche push per promemoria di lettura.

- Cloud e multi-device: sincronizzazione multi-device della libreria utente.

- Social e Intelligenza: sistema avanzato di tagging, note, e filtri personalizzati, possibilità di condividere la libreria con altri utenti o gruppi e suggerimenti intelligenti basati sulle abitudini di lettura.

- Migrare da `AsyncStorage` a SQLite o a un backend cloud.

- Separare ulteriormente il codice in moduli orientati alle funzionalità (*feature-based structure*).

- Implementare una suite di test per garantire la stabilità delle future release.