

# Book Library App



## Relazione tecnica

**Corso:** Mobile Programming

**Team:** Landi Gennaro Francesco, Palmisano Elettra, Peluso Francesco

**Data:** 02 Giugno 2025

**Introduzione.....** ..... **3**

**Progettazione dell'Applicazione.....** ..... **4**

    1. Concettualizzazione..... ..... 4  
    2. Definizione..... ..... 4  
    3. Analisi dei Requisiti..... ..... 4  
    4. Design..... ..... 5

**Architettura & Navigazione tra le Schermate.....** ..... **6**

    Schermata Home: index.tsx..... ..... 8

        Dettaglio Libro: book-details.tsx..... ..... 9

    Schermata Aggiungi Libro: aggiungi.tsx..... ..... 9

        1. Aggiunta manuale: add-book.tsx..... ..... 10

        2. Scansione ISBN: scan.tsx..... ..... 11

    Schermata Profilo: profile.tsx..... ..... 11

        1. Schermata Statistiche di Lettura: statistics.tsx..... ..... 12

        2. Schermata Preferiti: favorites.tsx..... ..... 13

        3. Schermata Wishlist: wishlist.tsx..... ..... 13

        4. Settings: settings.tsx..... ..... 13

**Design Responsive & Adattamento Multi-Dispositivo.....** ..... **14**

**Persistenza Dati con SQLite.....** ..... **15**

**Stack tecnologico.....** ..... **19**

**Esperienza Utente e Progettazione dell'interfaccia.....** ..... **22**

# Introduzione

L'applicazione ReadIt consente all'utente di costruire e gestire la propria libreria personale, tenendo traccia di ogni libro in base allo stato di lettura ("da leggere", "in lettura", "completato"). Per ciascun libro è possibile:

- memorizzare informazioni bibliografiche (titolo, autore, copertina, anno di pubblicazione, ISBN);
- selezionare o creare generi e categorie personalizzate (es. "Giallo", "Fantasy", "Saggi");
- assegnare uno stato di lettura ("da leggere", "in lettura", "completato");
- aggiungere note personali (pensieri, citazioni, appunti);
- inserire una valutazione da 1 a 5 stelle e, opzionalmente, un commento;
- contrassegnare il libro come "preferito" o inserirlo in una "wishlist" se non è ancora in possesso.

I libri possono essere aggiunti manualmente con tutti i campi necessari oppure ricercati direttamente online per titolo, autore o ISBN. Inoltre ReadIt calcola delle raccomandazioni di nuovi titoli in base all'autore e ai generi già registrati nella propria libreria.

Sul profilo personale l'utente trova statistiche di lettura aggregate, fra cui:

- numero totale di libri "completati", "in lettura" o "da leggere";
- distribuzione dei titoli per genere (con grafici a torta);
- valutazione media dei libri già letti e valutazioni personali per ogni libro che ne possiede una;
- tempo totale e medio di lettura, calcolato tramite il tracciamento delle sessioni di lettura: l'utente può avviare, mettere in pausa e terminare una sessione, registrando in automatico la durata e opzionalmente il libro letto durante la sessione;
- streak giornaliero di lettura, volto ad incentivare la continuità.

Infine, ReadIt integra una funzionalità di ricerca avanzata che permette di filtrare i libri presenti nella libreria per stato di lettura, genere, rating o presenza in "wishlist"/"preferiti", ordinandoli secondo diversi criteri (es. data di aggiunta, valutazione). Grazie a un'interfaccia responsive, l'app si adatta automaticamente sia a smartphone di differenti dimensioni sia a tablet o ipad con orientamenti portrait/landscape, garantendo un'esperienza fluida e intuitiva.

## Tutorial iniziale (first-time walkthrough)

Alla prima apertura di ReadIt, viene mostrato all'utente un breve tutorial a schermo intero che spiega brevemente:

1. A cosa serve l'app;

2. Come avviare una sessione di lettura;

Al termine dell'introduzione, viene chiesto all'utente di indicare i propri **generi di lettura preferiti** (es. Giallo, Fantasy, Saggistica...) in modo che, in base ai generi selezionati, l'app calcola fin da subito alcuni consigli di lettura personalizzati, visibili nel carosello raccomandazioni.

Il tutorial è alternativamente **skippabile**:

- in seguito non verrà più mostrato;
- le raccomandazioni iniziali non saranno personalizzate, ma puramente casuali.

# Progettazione dell'Applicazione

La progettazione dell'applicazione ReadIt è stata condotta seguendo un approccio suddiviso in fasi logiche che hanno guidato la realizzazione dell'intero progetto.

## 1. Concettualizzazione

Partendo dall'idea di base che ci è stata fornita, ovvero di creare un'app per la gestione di una libreria personale, abbiamo identificato ulteriori funzionalità che potrebbero essere utili all'utente, come ad esempio:

- **Tracciamento delle sessioni di lettura**, con timer integrato, streak di letture e tempo totale impiegato.
- **Raccomandazioni dinamiche** di lettura basate su autore e genere.
- **Aggiunta semplificata dei libri** tramite scansione del codice ISBN.

## 2. Definizione

### Utenti Finali / End Users:

L'applicazione è realizzata in vista di lettori abituali, studenti universitari o utenti che desiderano tracciare i propri progressi di lettura e ricevere suggerimenti personalizzati.

## 3. Analisi dei Requisiti

### Requisiti Funzionali (RF)

RF-ID	Descrizione	Schermata
RF-01	Elenco libri per stato (In lettura / Da leggere / Completati)	Home
RF-02	Ricerca con filtri (titolo, autore, stato, preferiti, genere)	Home
RF-03	Aggiunta libro manuale o tramite scansione ISBN	Aggiungi libro
RF-04	Visualizzazione dettagli libro	BookDetails
RF-05	Visualizzazione statistiche di lettura	Statistiche (Profile)

RF-06	Gestione di una wishlist	Wishlist
RF-07	Visualizzazione dei libri preferiti	Favorites
RF-08	Raccomandazioni in base a libri simili	BookDetails

### Requisiti Non Funzionali (RNF)

RNF-ID	Descrizione
RNF-01	Accessibilità: contrasto sufficiente, supporto screen reader
RNF-02	Persistenza offline garantita tramite database locale SQLite
RNF-03	Scalabilità: codice strutturato per moduli, cartelle per feature
RNF-04	Supporto a schermi multipli e orientamento verticale/landscape responsive

## 4. Design

In questa fase abbiamo definito sia la **struttura dell'interfaccia utente**, sia l'**architettura dei dati** dell'applicazione.

- È stato progettato un **database relazionale locale** utilizzando **SQLite**, che garantisce la persistenza dei dati anche in modalità offline. La struttura del database è descritta nella sezione apposita del documento.
- Abbiamo realizzato **wireframe** delle schermate principali che ci hanno guidato nello sviluppo dell'interfaccia.

Alcuni dei wireframe sono riportati nelle slide di presentazione e sono affiancati agli screenshot delle schermate finali, per mostrare l'evoluzione dal design iniziale all'implementazione.

# Architettura & Navigazione tra le Schermate

Nel progetto è stato utilizzato **Expo Router** per gestire la navigazione tra le schermate. Ogni file presente nella cartella `app/` rappresenta automaticamente una schermata (rotta), secondo una logica file-based routing.

## Cartella principale app/

La cartella `app/` è il punto di ingresso per Expo Router. Al suo interno troviamo:

- `_layout.tsx`: definisce il layout di navigazione principale.
- Le due sottocartelle principali:
  - `app/(tabs)/` → contiene le schermate principali accessibili tramite la bottom tab bar.
  - `app/(screens)/` → contiene schermate secondarie (non visibili nella tab bar), ma raggiungibili tramite pulsanti o azioni (es. `router.push(...)`).

All'interno della cartella `app/` sono presenti due sottocartelle: `(tabs)` e `(screens)`.

### 1. Navigazione tra Tab: `app/(tabs)/`

Questa cartella contiene le tre **schermate principali**, secondo la seguente struttura:

`app/(tabs)/`

```
|── _layout.tsx      ← definisce il componente <Tabs> e la grafica della tab bar  
|── index.tsx       ← Schermata "Home"  
|── aggiungi.tsx    ← Schermata "Aggiungi Libro"  
└── profilo.tsx     ← Schermata "Profilo Personale"
```

Il file `_layout.tsx` in `(tabs)/` contiene la configurazione grafica delle tab (icone, colori, etichette, visibilità dell'header) tramite il componente `<Tabs>` di Expo Router.

### 2. Navigazione tra Schermate: `app/(screens)/`

Le **schermate che non fanno parte della bottom tab bar** ma che possono essere raggiunte tramite interazioni (es. tocco su un libro) sono collocate in `app/(screens)/`:

`app/(screens)/`

```
|── _layout.tsx  
|── book-details.tsx   ← Dettaglio di un libro selezionato  
|── add-book.tsx      ← Form per aggiungere/modificare un libro  
|── favorites.tsx     ← Lista dei libri preferiti
```

└─ wishlist.tsx	← Libri da acquistare
└─ scan.tsx	← Scanner ISBN
└─ statistics.tsx	← Statistiche di lettura
└─ settings.tsx	← Schermata Impostazioni

Ogni file è comunque accessibile tramite il metodo `router.push(...)`: Expo Router genera navigazione a stack in automatico per le route al di fuori delle Tab.

## Navigazione dinamica e hook useRouter

Per la navigazione è stato usata la funzione di hook `useRouter()` fornita da Expo Router:

- `router.push(path)` → inserisce una nuova schermata nella pila di navigazione.
- `router.back()` → rimuove la schermata corrente e torna a quella precedente.
- `useLocalSearchParams()` → consente di recuperare eventuali parametri passati

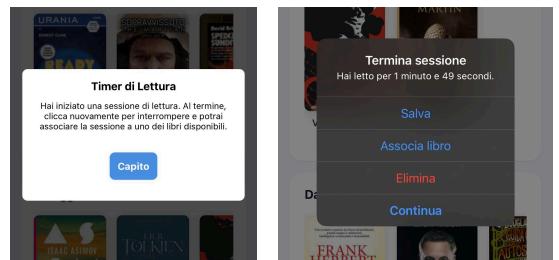
# Schermate dell'applicazione

## Schermata Home: *index.tsx*

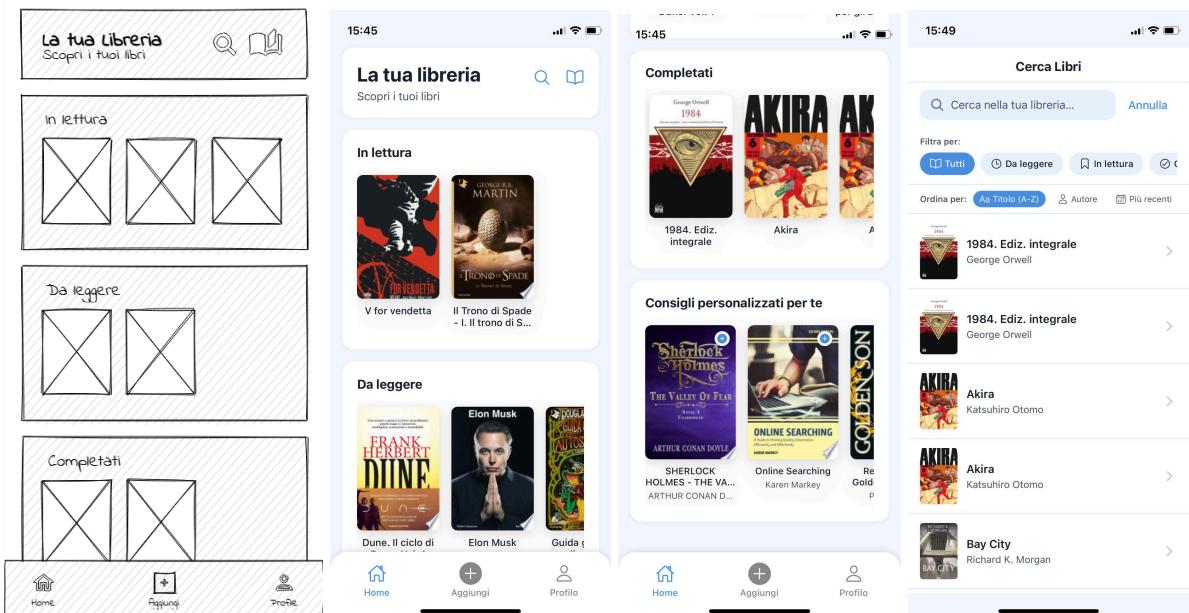
La **Home** dell'applicazione mostra la libreria personale dell'utente, con i libri in possesso dall'utente, visualizzati in tre sezioni distinte e scrollabili, una per ogni stato di lettura: *Da leggere*, *In lettura*, *Completati*. Ogni libro è cliccabile, e riporta alla schermata di *book-details.tsx* (solo visualizzazione) del libro selezionato.

In alto sono presenti:

- una **barra di ricerca personalizzabile**, che consente di filtrare i libri in base allo stato attuale, a quelli presenti nella lista dei preferiti, o associati ad un particolare genere.
- un'icona a forma di libro, tramite la quale l'utente può **avviare o interrompere una sessione di lettura**. Al termine della sessione, l'utente può salvare o eliminarla, cliccando nuovamente sull'icona, ed eventualmente associarla ad un libro della propria libreria e contrassegnarlo come completato. La prima volta che l'utente utilizza l'app, prima di iniziare la sessione di lettura verrà mostrato all'utente una spiegazione sul funzionamento dell'icona.



La schermata presenta inoltre **suggerimenti di lettura** basandosi sulle preferenze dell'utente (tramite le statistiche presenti nel profilo), volti a stimolare l'utente nell'esplorare nuove letture.



## Dettaglio Libro: `book-details.tsx`

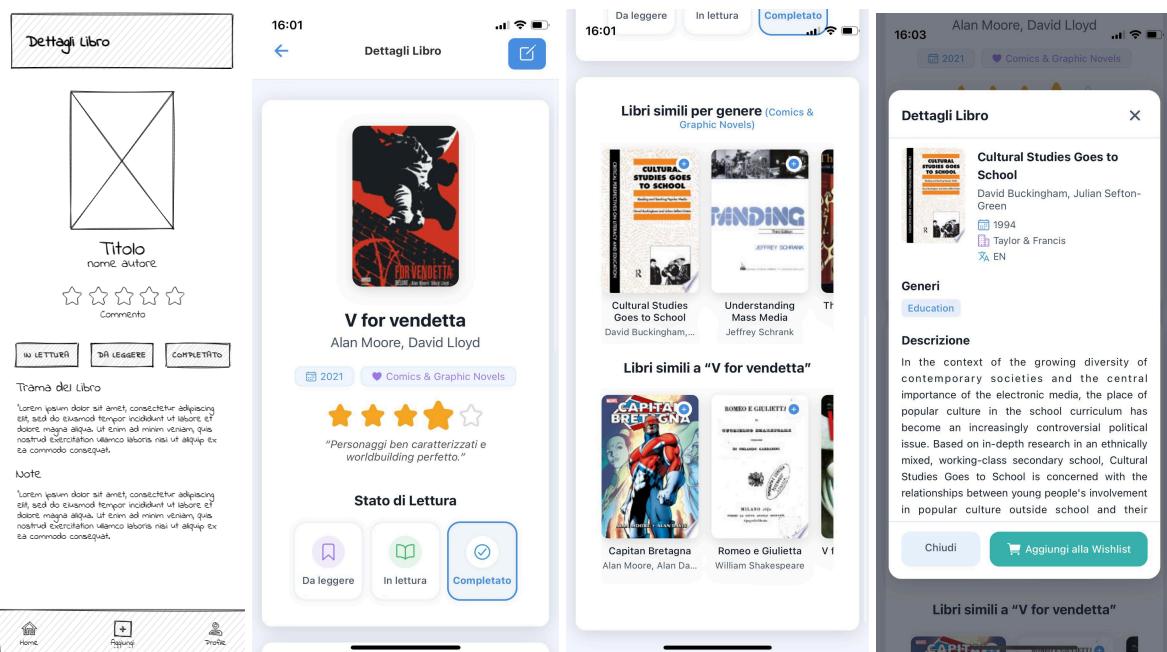
Questa schermata consente all'utente di **visualizzare in sola lettura** tutte le informazioni relative a un libro selezionato dalla propria libreria.

Sono mostrate le seguenti informazioni:

- **Titolo e autore**
- **Stato di lettura** attuale (Da leggere, In lettura, Completato)
- **Trama del libro**
- **Note personali e valutazione** se presenti
- **Un'icona di un cuore** visualizzabile solo se il libro è stato aggiunto ai preferiti dall'utente.

In fondo alla pagina sono proposte **raccomandazioni personalizzate** di lettura, basate sull'autore o sul genere del libro visualizzato.

Invece in alto a destra è presente un'**icona di modifica**, che reindirizza alla schermata `add-book.tsx`, permettendo di aggiornare le informazioni del libro.



## Schermata Aggiungi Libro: `aggiungi.tsx`

La schermata di Aggiunta di un Libro mostra all'utente **due opzioni**:

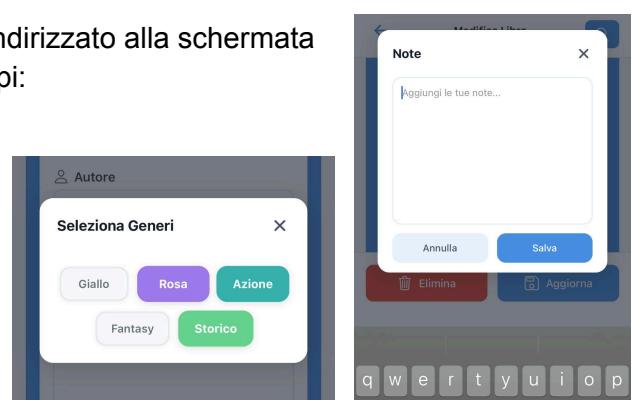
1. **Aggiungere un libro manualmente**
2. **Aggiungerlo tramite scansione di un codice a barre**



## 1. Aggiunta manuale: `add-book.tsx`

Se l'utente sceglie l'inserimento manuale, viene indirizzato alla schermata `add-book.tsx`, dove può compilare i seguenti campi:

- Titolo del libro e Autore
- Anno di pubblicazione
- ISBN
- Copertina (tramite URL o selezione in galleria)
- Trama
- Genere/i
- Stato di lettura (*da leggere, in lettura, completato*)

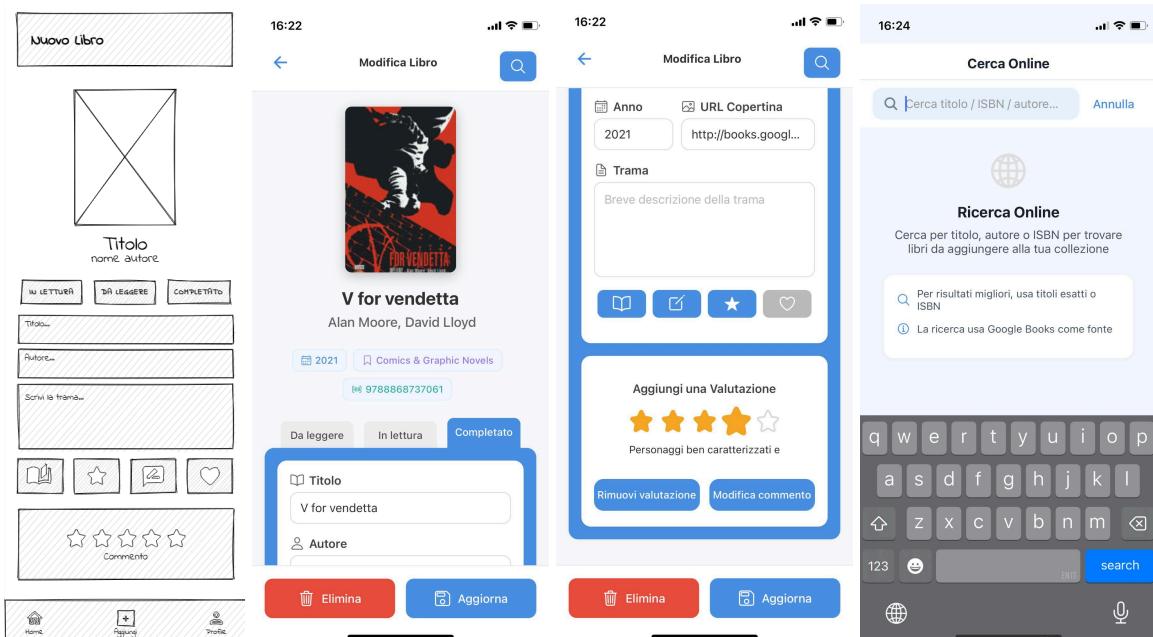


In alternativa, in alto a destra è presente un'**icona di ricerca**, che consente all'utente di ricercare un libro tramite titolo, autore o ISBN. Una volta selezionato il libro l'applicazione **compila automaticamente** i campi, dove possibile.

Ad ogni libro è possibile associare opzionalmente:

- Note personali (opzionali)
- Valutazione da 1 a 5 stelle e commento associato
- Aggiunta ai **preferiti**

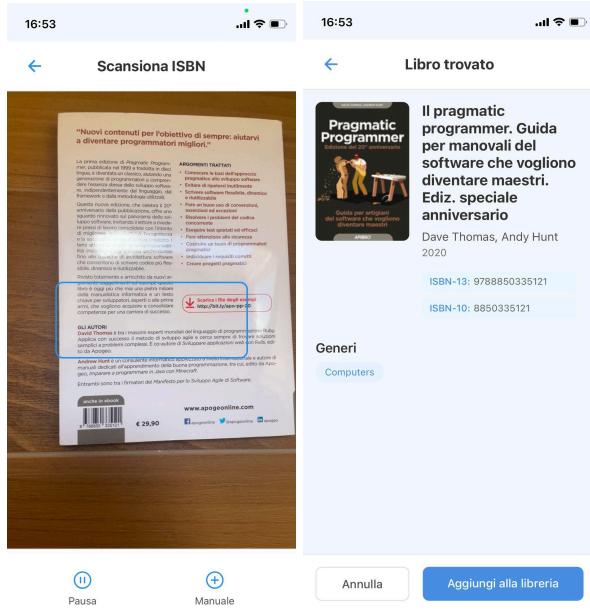
La schermata viene visualizzata sia per l'aggiunta di un nuovo libro, sia per la **modifica** di uno già esistente, tramite l'icona in alto a destra della pagina `book-details.tsx`.



## 2. Scansione ISBN: `scan.tsx`

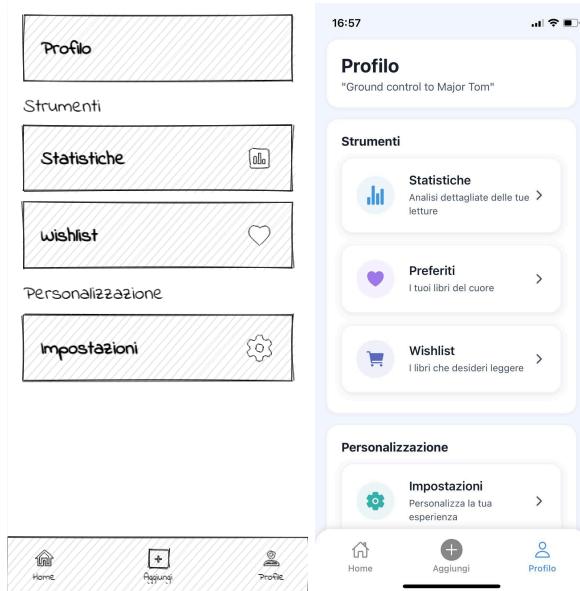
Se l'utente seleziona l'opzione di scansione, viene indirizzato alla schermata `scan.tsx`, che richiede i **permessi di accesso alla fotocamera**. Una volta concessi, viene attivato il lettore di codici a barre.

Scansionando il **codice ISBN** di un libro, l'app restituisce la visualizzazione del libro, se trovato.



## Schermata Profilo: `profile.tsx`

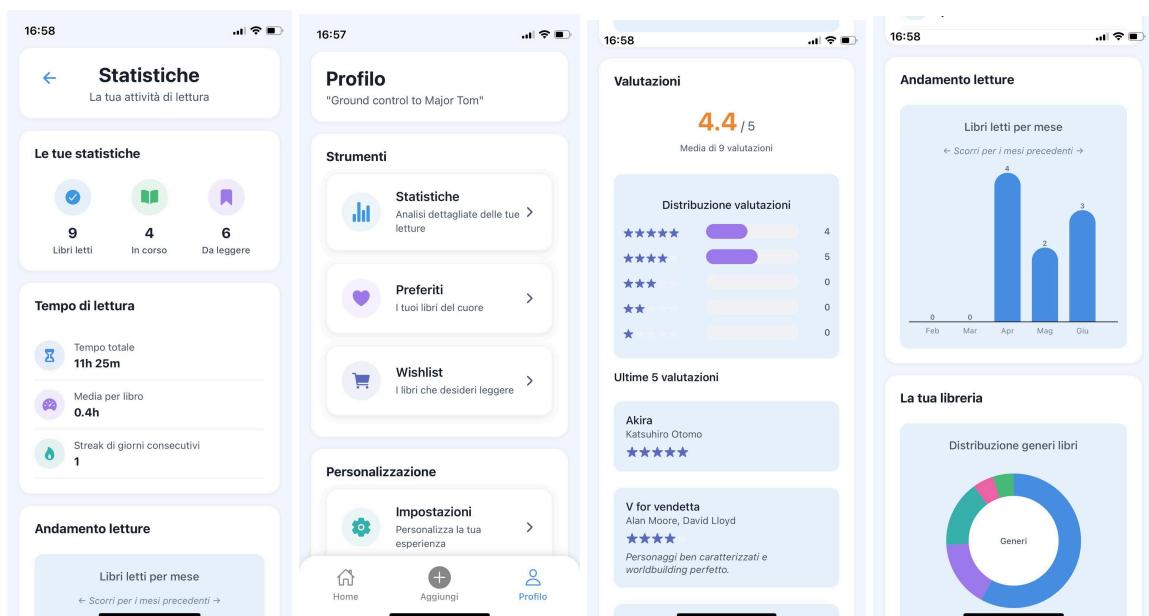
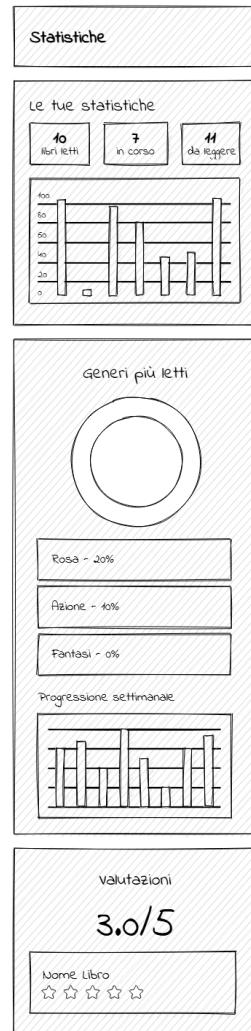
La schermata **Profilo** permette all'utente di accedere rispettivamente alle seguenti schermate:



## 1. Schermata Statistiche di Lettura: *statistics.tsx*

La schermata dedicata alle **statistiche di lettura** consente all'utente di monitorare in modo visuale e interattivo il proprio andamento nella lettura dei libri. Essa è suddivisa in diverse sezioni informative:

- **Stato dei Libri:** Mostra il numero totale dei libri in base allo stato di lettura.
- **Tempo di Lettura:** Vengono visualizzati:
  - Il **tempo totale di lettura** cumulativo (espresso in ore/minuti),
  - Il **tempo medio di lettura per libro**, se disponibile,
  - Lo **streak di lettura** giornaliero, ovvero il numero di giorni consecutivi in cui l'utente ha letto almeno una volta.
- **Statistiche Visive:** La sezione include:
  - Un **grafico a barre** che mostra il numero di libri completati per mese;
  - Un **grafico a torta** che mostra la distribuzione dei libri per genere, calcolata in base ai libri effettivamente letti;
  - Un **grafico a linee** che mostra l'attività di lettura settimanale.
- **Valutazioni:** Infine, viene mostrata una sezione dedicata alla **valutazione media** calcolata tra i libri che la possiedono e all'insieme di valutazioni associate a ciascun libro.



## 2. Schermata Preferiti: `favorites.tsx`

All'interno di questa sezione viene visualizzata la **lista dei libri che l'utente ha contrassegnato come preferiti**. Per ogni libro è mostrata la **data di aggiunta ai preferiti** e ogni elemento della lista è **cliccabile**, portando l'utente alla **pagina di dettaglio del libro** selezionato.

## 3. Schermata Wishlist: `wishlist.tsx`

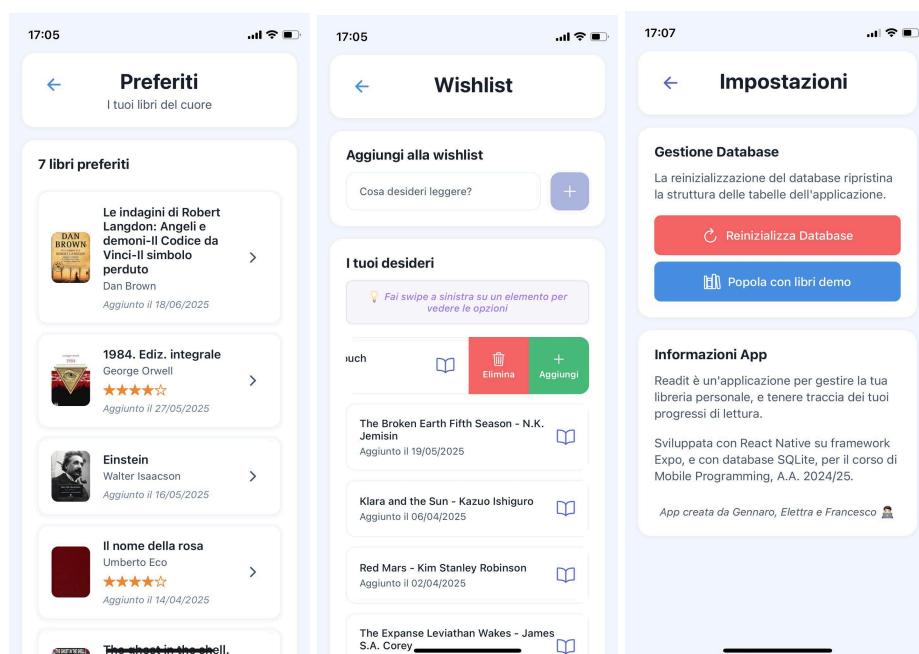
Questa sezione mostra la **lista dei libri che l'utente desidera leggere o acquistare in futuro**. Per ogni libro è indicata la **data di aggiunta alla wishlist**. L'utente può:

- **Scorrere lateralmente (swipe)** ciascun elemento per:
  - Aggiungere il libro alla propria librerie personale;
  - Rimuoverlo dalla wishlist.
- **Aggiungere manualmente un nuovo libro alla wishlist** utilizzando l'**input di testo** posizionato nella parte superiore della schermata.

## 4. Settings: `settings.tsx`

All'interno della schermata **Impostazioni** l'utente ha accesso a funzionalità di gestione dell'app, tra cui:

- **Reinizializzazione del database**, per cancellare completamente tutti i libri e ripristinare l'app allo stato iniziale;
- **Popolamento con dati demo**, utile per simulare rapidamente il funzionamento dell'app con una libreria di esempio, creata grazie al file `demoInitialize.ts`.
- **Visualizzazione delle informazioni sull'app**.

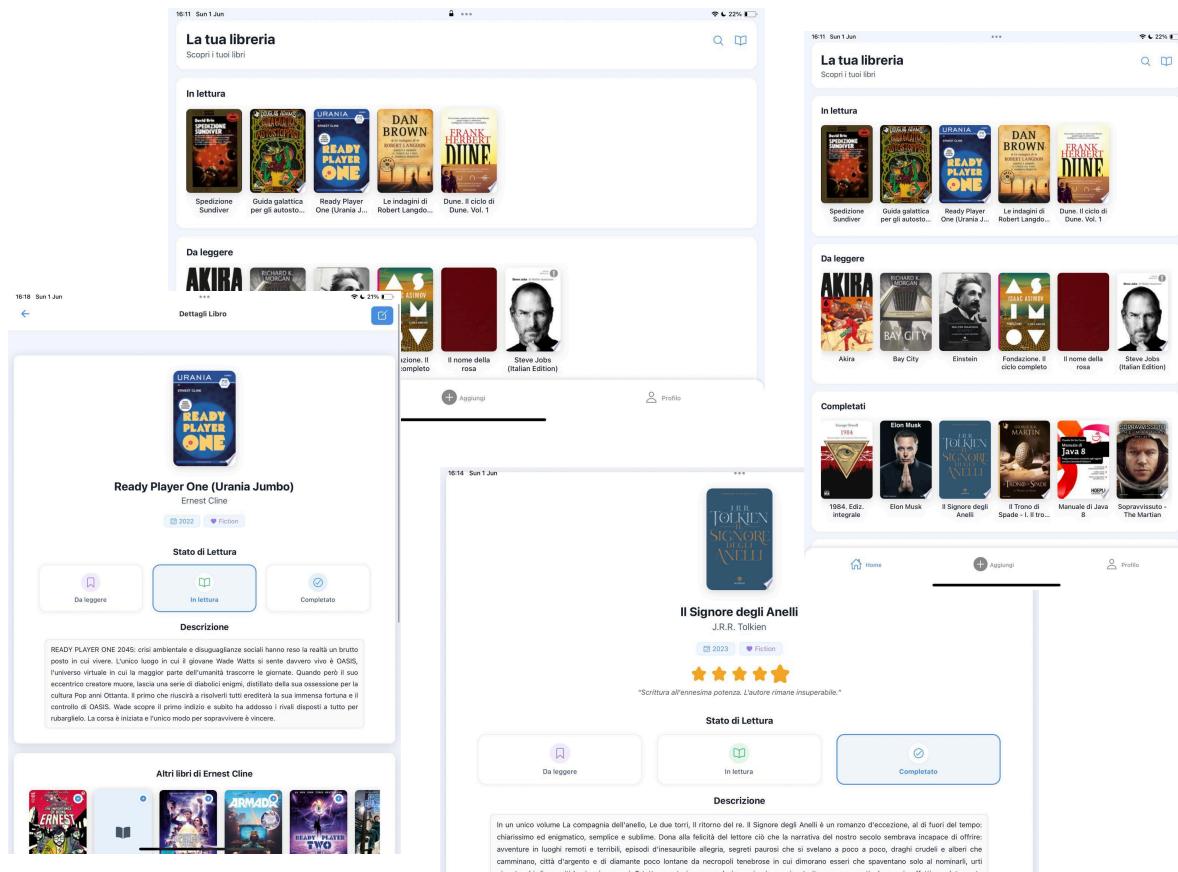


# Design Responsive & Adattamento Multi-Dispositivo

L'interfaccia dell'app **ReadIt** è stata progettata con attenzione alla **responsività**, per garantire un'esperienza ottimale su dispositivi con dimensioni differenti.

- **Layout flessibili:** le principali schermate utilizzano componenti come `ScrollView`, `FlatList` e `flexbox` (`flexDirection`, `justifyContent`, `alignItems`) per adattare automaticamente i contenuti allo spazio disponibile.
- **Stili proporzionali:** margini, padding e dimensioni dei componenti sono stati definiti in modo relativo (ad es. usando `%`, unità di spacing centralizzate o `Dimensions.get('window')`) così da essere scalabili in base alla dimensione della finestra del dispositivo.
- **Orientamento Portrait vs Landscape:** l'app è ottimizzata per l'uso in modalità verticale (Portrait), ma è visualizzabile senza problemi anche in modalità Landscape (ad esempio su Ipad o tablet), offrendo in ogni modo un utilizzo ottimale.
- **Comportamento adattivo delle tab:** la Bottom Tab Bar è stata stilizzata per adattarsi correttamente su schermi piccoli e grandi, con etichette leggibili e icone che mantengono una disposizione bilanciata.

Inoltre è da tenere presente che l'uso di Expo Router e React Native garantiscono nativamente un supporto multi-piattaforma (iOS/Android), semplificando la gestione della UI nei diversi ambienti.



# Persistenza Dati con SQLite

L'applicazione **ReadIt** utilizza **SQLite** come database locale per garantire **persistenza dei dati offline**, rapidità di accesso e affidabilità.

## Vantaggi di Utilizzo:

- **Integrato nativamente** in React Native tramite import di librerie come `expo-sqlite`, evitando dipendenze server-side e configurazioni complesse.
- Non richiede una connessione a internet → supporto completo anche **offline**.
- **Dati persistenti** tra sessioni dell'utente.
- Ottimo per applicazioni mobile con strutture dati leggere ma relazionali.
- Buona performance per query locali e supporto ad operazioni **CRUD (Create, Read, Update, Delete)** richieste dal front-end dell'applicazione.

## Struttura del Database

Il database viene inizializzato all'avvio dell'app (se non già esistente) grazie a una funzione centralizzata in `database.ts`. Lo schema include le varie tabelle necessarie per la memorizzazione dei dati. I **vincoli di integrità referenziale** e **trigger automatici** (come la creazione automatica dello stato di lettura per nuovi libri) mantengono la coerenza dei dati a livello database.



Alcune delle **tabelle** create sono:

1. **books** → libri in possesso dall'utente:

```
SQL
CREATE TABLE IF NOT EXISTS books (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    description TEXT,
    cover_url TEXT,
    editor TEXT,
    publication INTEGER,
    language TEXT,
    isbn10 TEXT CHECK(length(isbn10) = 10 OR isbn10 IS NULL),
    isbn13 TEXT CHECK(length(isbn13) = 13 OR isbn13 IS NULL),
    created_at TEXT NOT NULL DEFAULT (datetime('now'))
);
```

2. **genres** → gestione delle categorie disponibili (generi):

```
SQL
CREATE TABLE IF NOT EXISTS genres (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    description TEXT,
    created_at TEXT NOT NULL DEFAULT (datetime('now'))
);
```

3. **book\_genres** → relazione molti a molti tra i libri e i generi:

```
SQL
CREATE TABLE IF NOT EXISTS book_genres (
    book_id INTEGER NOT NULL,
```

```
genre_id INTEGER NOT NULL,  
PRIMARY KEY (book_id, genre_id),  
FOREIGN KEY(book_id) REFERENCES books(id)  
    ON DELETE CASCADE ON UPDATE CASCADE,  
FOREIGN KEY(genre_id) REFERENCES genres(id)  
    ON DELETE CASCADE ON UPDATE CASCADE  
);
```

**4. notes** → collega le note ai rispettivi libri:

```
SQL  
CREATE TABLE IF NOT EXISTS genres (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    name TEXT NOT NULL UNIQUE,  
    description TEXT,  
    created_at TEXT NOT NULL DEFAULT (datetime('now'))  
);
```

**5. ratings** → Permette di associare una valutazione (1–5 stelle) e un commento opzionale:

```
SQL  
CREATE TABLE IF NOT EXISTS ratings (  
    book_id INTEGER PRIMARY KEY,  
    rating INTEGER NOT NULL  
        CHECK(rating BETWEEN 1 AND 5),  
    comment TEXT,  
    rated_at TEXT NOT NULL DEFAULT (datetime('now')),  
    FOREIGN KEY(book_id) REFERENCES books(id)  
        ON DELETE CASCADE ON UPDATE CASCADE  
);
```

**6. reading\_sessions** → Tiene traccia del tempo di lettura per ciascun libro:

```
SQL
CREATE TABLE IF NOT EXISTS reading_sessions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    book_id INTEGER,
    start_time TEXT NOT NULL,
    end_time TEXT,
    duration INTEGER GENERATED ALWAYS AS (
        strftime('%s', end_time) - strftime('%s', start_time)
    ) STORED,
    FOREIGN KEY(book_id) REFERENCES books(id)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

## Integrazione nel Codice: *bookApi.ts*

Il file *bookApi.ts* funge da **interfaccia** per interagire con il database locale SQLite, gestendo le operazioni di creazione, lettura, aggiornamento e cancellazione (**CRUD**) relative ai libri e alle loro informazioni associate.

### Funzionalità principali di *bookApi.ts*

#### 1. Aggiunta di un nuovo libro

La funzione *addBook(book: Book)* consente di inserire un nuovo libro nel database, tramite un'istruzione *INSERT INTO* nella tabella *books*.

#### 2. Recupero di tutti i libri

La funzione *getAllBooks()* esegue una query *SELECT \* FROM books* per recuperare tutti i libri presenti nel database.

#### 3. Aggiornamento delle informazioni di un libro

La funzione *updateBook(book: Book)* permette di aggiornare i dettagli di un libro esistente nel database, tramite un'istruzione *UPDATE* sulla tabella *books*, modificando i campi con i nuovi valori forniti.

#### 4. Eliminazione di un libro

La funzione `deleteBook(bookId: number)` consente di rimuovere un libro dal database utilizzando il suo `id`, con l'istruzione `DELETE FROM books WHERE id = ?`.

#### 5. Gestione delle valutazioni

La funzione `saveRating(bookId: number, rating: number, comment?: string)` permette di inserire o aggiornare una valutazione per un libro, all'interno della tabella `ratings` associata al libro specificato.

#### 6. Gestione delle note

La funzione `saveNotes(bookId: number, note: string)` consente di aggiungere note personali a un libro, inserendo i dati nella tabella `notes` con riferimento al `book_id`.

#### 7. Gestione dei generi

Le funzioni `getGenres()`, `addGenre(genre: string)` e `toggleGenreForBook(bookId: number, genreId: number)` gestiscono i generi dei libri. Permettono di recuperare la lista dei generi, aggiungerne di nuovi e associare o dissociare un genere da un libro specifico.

# Stack tecnologico

L'applicazione **ReadIt** è stata sviluppata utilizzando il framework **React Native 0.79.2** con **Expo SDK 53.0.9**, interamente in **TypeScript 5.8.3**, per garantire type-safety.

## Perché è stato scelto TypeScript

La scelta di TypeScript rispetto a JavaScript è motivata dall'esigenza di garantire maggiore **robustezza, leggibilità e prevenzione degli errori** a tempo di compilazione.

- Il **sistema di tipi di TypeScript** ci permette di individuare errori potenziali già durante la fase di sviluppo, molto prima che l'applicazione raggiunga gli utenti finali. Questo **approccio proattivo** nella prevenzione degli errori si traduce in un'esperienza utente più stabile e affidabile.
- I tipi fungono da “**documentazione vivente**” del codice: utilizzando nel codice una funzione TypeScript, è immediatamente chiaro quali parametri accetta, cosa restituisce e come dovrebbe essere utilizzata, senza dover consultare documentazione esterna o analizzare l'implementazione.
- L'ecosistema di **React Native** è inoltre **nativamente compatibile** con TypeScript.

## Perché è stato scelto React Native (con Expo) invece di Flutter

La scelta di utilizzo di **React Native + Expo** rispetto a Flutter è stata motivata dalla familiarità del team con JavaScript/TypeScript e l'ecosistema React, permettendo di riutilizzare conoscenze esistenti del web development. Inoltre, la sintassi JSX e la community online hanno accelerato significativamente lo sviluppo.

Expo semplifica inoltre la **gestione delle build** (tramite EAS Build) e consente test immediati su dispositivi fisici attraverso Expo Go, velocizzando significativamente il ciclo di sviluppo.

## Librerie utilizzate

Di seguito sono elencate le principali librerie e dipendenze utilizzate all'interno del progetto (per una lista completa si rimanda al file `package.json`):

### Framework Core

- *React Native 0.79.2*
- *Expo SDK 53.0.9*
- *TypeScript 5.8.3*

### Navigazione

- *Expo Router 5.0.7* → per una navigazione file-based semplificata e integrata con lo stack Expo.

## Database Locale

- `expo-sqlite 15.2.10` → per la persistenza offline dei dati tramite database relazionale locale.

## UI e Componenti Essenziali

- `react-native-gesture-handler 2.24.0` → gesture-based interaction.
- `react-native-safe-area-context 5.4.0` → gestione automatica delle aree sicure.
- `react-native-gifted-charts 1.4.61` → visualizzazione di grafici a torta e istogrammi.
- `@expo/vector-icons 14.1.0` → icone cross-platform.

## Animazioni

- `Moti 0.30.0` → animazioni dichiarative e fluide basate su Reanimated.
- `react-native-reanimated 3.17.4` → motore per animazioni performanti.

## Funzionalità Native

- `expo-camera 16.1.6` → utilizzo della fotocamera per lo scanner ISBN.
- `expo-haptics 14.1.4` → feedback tattile nativo per migliorare l'interazione utente.

## Notifiche, Toast & Modal

- `react-native-toast-message 2.3.0` → notifiche toast personalizzate.
- `react-native-modal 14.0.0` → modali animati e configurabili.

In molte situazioni è stata preferita l'adozione di librerie offerte da **Expo** rispetto a quelle native di React Native. Questo approccio è stato scelto con l'obiettivo di:

- evitare configurazioni native complesse,
- migliorare la compatibilità cross-platform (iOS e Android),
- accelerare il prototyping e lo sviluppo,
- semplificare l'accesso a feature native tramite interfacce JavaScript.

# Esperienza Utente e Progettazione dell'interfaccia

ReadIt è stata progettata per garantire un'esperienza d'uso intuitiva, accessibile e coerente, curando sia aspetti funzionali che visivi dell'interfaccia utente.

### **Design e Colori**

- La paletta di colori utilizzata risulta coerente all'interno di tutta l'applicazione;
  - I pulsanti “attivi” e “non attivi” sono visivamente distinti, per guidare l'utente;
  - Lo sfondo è neutro per garantire leggibilità e non sovraccaricare la visualizzazione;
  - Sono utilizzate tonalità chiare ma leggibili per testi e label secondari;
  - I colori sono contrastanti per garantire una chiara leggibilità sia in condizioni normali che per utenti con disabilità visive leggere;
  - Le icone utilizzate (*Ionicons* e *AntDesign*) sono intuitive ed associate ad azioni comuni: cuore per preferiti, carrello per wishlist, stella per rating.



## Feedback aptico

L'app fornisce un **feedback** (vibrazione breve) nelle interazioni principali, rendendo più piacevole la navigazione e più intuitivo l'avvio e la chiusura delle sessioni di lettura.

## Form e Inserimento Dati

- Gli input form presentano **etichette chiare e placeholder**;
  - Supporto per **autocomplete** nella ricerca dei libri;
  - Pulsanti disabilitati se i campi obbligatori non sono compilati, per evitare errori da parte dell'utente.

## Ricerca e Filtri

- **Ricerca intelligente integrata** nella Home, con filtri combinabili.
  - Ordinamento personalizzabile (es. per data di aggiunta, titolo).

## Gestione Sicura dei Dati

- Azioni potenzialmente distruttive (es. elimina libro, reset DB) sono protette da **conferme o pulsanti secondari**, per evitare tocchi accidentali.

## Link utili

- [Slide della Presentazione](#)
- [Repository GitHub](#)