

Software Design Document (SDD)

Software per la gestione di una biblioteca

Autori: Marino Francesco, Pepe Daniele, Orsini Giovanni, Palmieri Vito

Gruppo 1

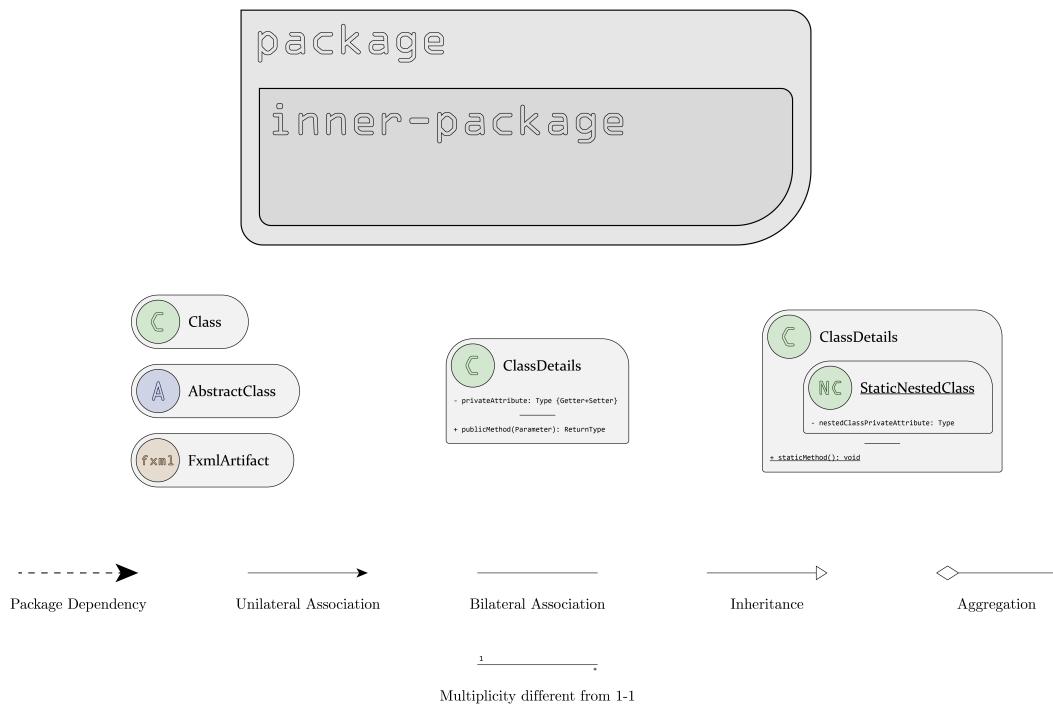
16 dicembre 2025

Indice

1 Introduzione	3
1.1 Convenzioni grafiche adottate nei diagrammi	3
2 Architettura del sistema	4
3 Modello statico	6
3.1 Diagramma delle classi correlate a Launcher	6
3.1.1 Classe Launcher	6
3.2 Diagramma delle classi di MVC	7
3.2.1 Classi di <code>mvc.model</code>	7
3.2.2 Classi di <code>mvc.controller</code>	7
3.2.3 Artifacts di <code>mvc.view</code>	8
3.2.4 Esempio delle interazioni di un Model-View-Controller specifico (User-Set)	9
3.2.5 Esempio delle interazioni di un modello e della sua aggregazione (Book e BookSet)	10
3.3 Diagramma delle classi di db e <code>db.omnisearch</code>	11
3.3.1 Classi di db	11
3.3.2 Classi di <code>db.omnisearch</code>	11
3.4 Diagramma delle classi di exceptions	12
3.4.1 Classi di <code>exceptions</code>	12
3.5 Scelte Progettuali	12
3.6 Principi di buona progettazione	13
4 Modello dinamico	14
4.1 Diagramma di sequenza: Aggiunta di un nuovo libro	14
4.2 Diagramma di sequenza: Prestito di un libro	14
4.3 Diagramma di Attività: Avvio dell'Applicazione e Caricamento Dati	15
5 Design dell'interfaccia utente	17
5.1 Schermate principali dell'applicazione	17
5.2 Finestre di visualizzazione	19
5.3 Finestre di modifica	20

1 Introduzione

1.1 Convenzioni grafiche adottate nei diagrammi



2 Architettura del sistema

L'architettura del sistema è organizzata secondo il pattern Model-View-Controller (MVC). I moduli principali sono divisi in package che riflettono questa struttura adattata alla complessità del progetto e alle esigenze specifiche dell'applicazione:

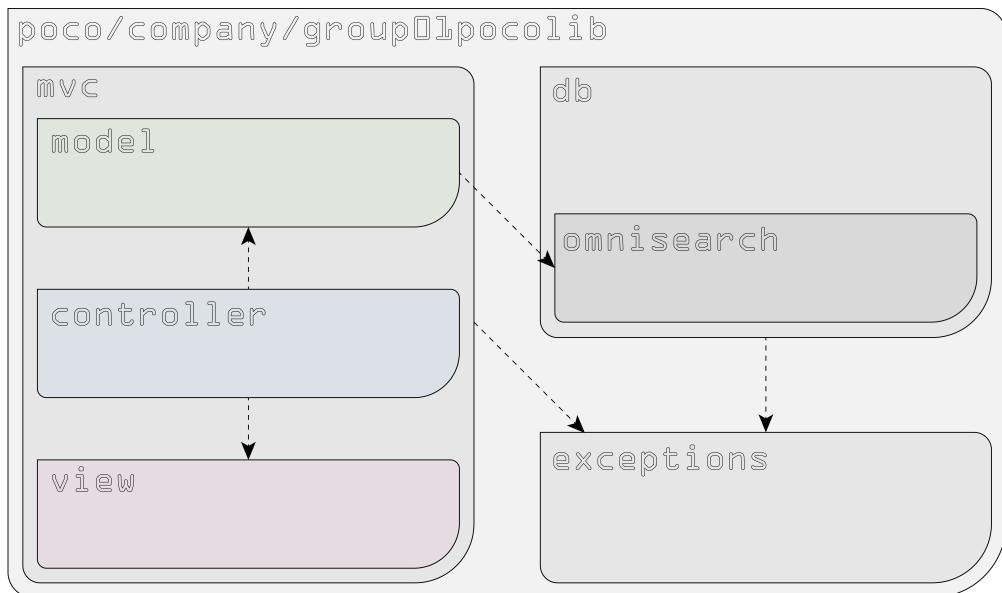


Figura 1: Diagramma dei Package

Modulo	Responsabilità	Dipendenze	Test
poco.company. group01pocolib	Contiene l'entry point dell'applicazione (Launcher) e coordina l'inizializzazione del sistema.	mvc.controller, mvc.model, JavaFX	-
mvc.controller	Gestisce la logica di presentazione e l'interazione utente. Include i controller per le proprietà degli elementi (BookPropController , ecc.) e per le tab. Il PocoLibController coordina i controller delle tab e gestisce la navigazione.	mvc.model, JavaFX FXML	-
mvc.model	Implementa la logica di business e mantiene lo stato dell'applicazione. Contiene le entità (Book , User , Lending) e le collezioni che le gestiscono. Fornisce metodi per operazioni CRUD e ricerca.	db, Java I/O	BookTest, UserTest, LendingTest (e relativi Set)
db	Responsabile della persistenza dei dati su file system (DB), gestione dei backup (Backup) e calcolo hash per l'integrità dei file (Hash). Serializza e deserializza le strutture dati.	mvc.model, Java I/O	DBTest, BackupTest, HashTest
db.omnisearch	Implementa la funzionalità di ricerca avanzata tramite indicizzazione trigram (Index) e algoritmi di ricerca fuzzy (Search). Supporta ricerche parziali e tolleranti agli errori di battitura.	mvc.model	SearchTest
exceptions	Contiene la gerarchia delle eccezioni personalizzate per la gestione degli errori specifici dell'applicazione (BookDataNotValidException , ecc.).	Nessuna dipendenza interna	-

3 Modello statico

Le classi del sistema sono state organizzate secondo il pattern Model-View-Controller (MVC) per separare la logica di business dalla logica di presentazione e dalla persistenza dei dati.

3.1 Diagramma delle classi correlate a Launcher

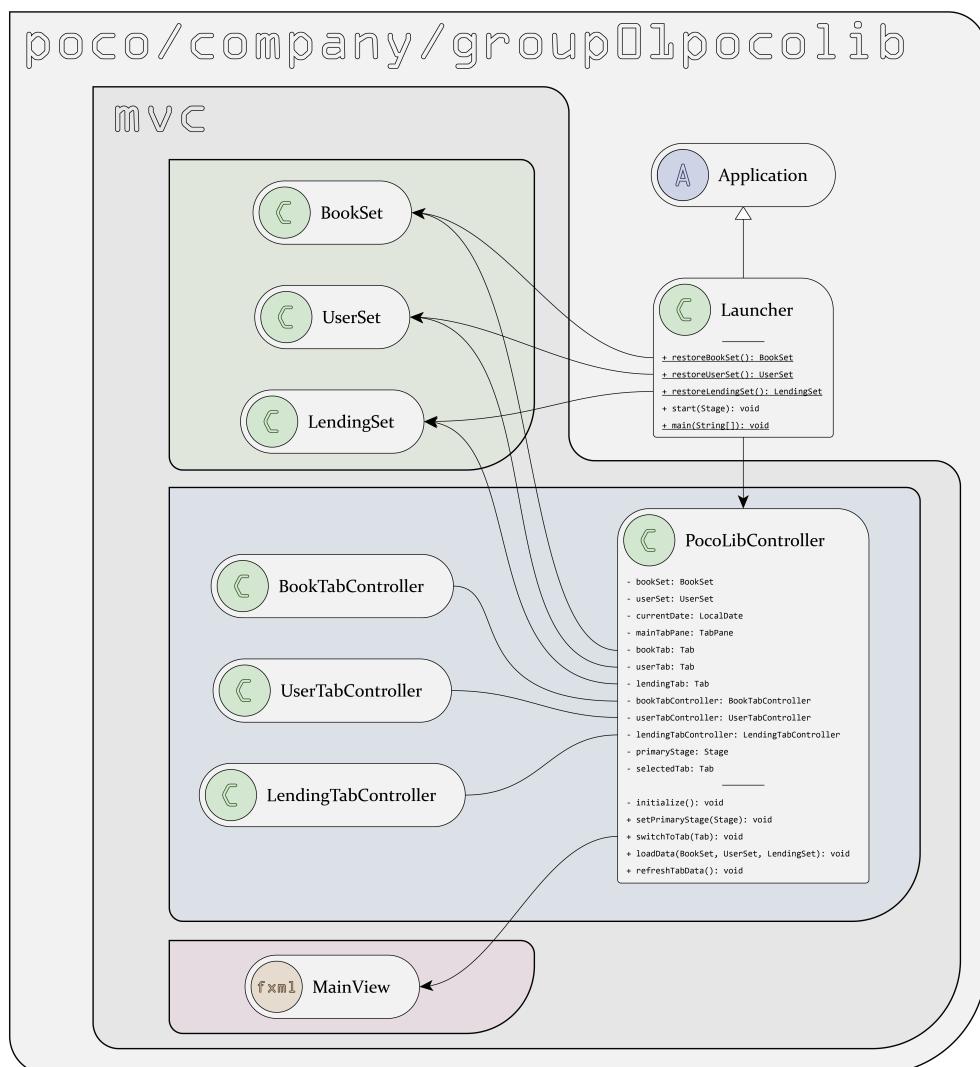


Figura 2: Diagramma delle classi correlate a Launcher

3.1.1 Classe Launcher

Questa classe contiene il metodo `main` e il metodo `start` per avviare l'app JavaFX. Contiene inoltre i metodi `restoreBookSet`, `restoreUserSet` e `restoreLendingSet` per caricare i dati.

3.2 Diagramma delle classi di MVC

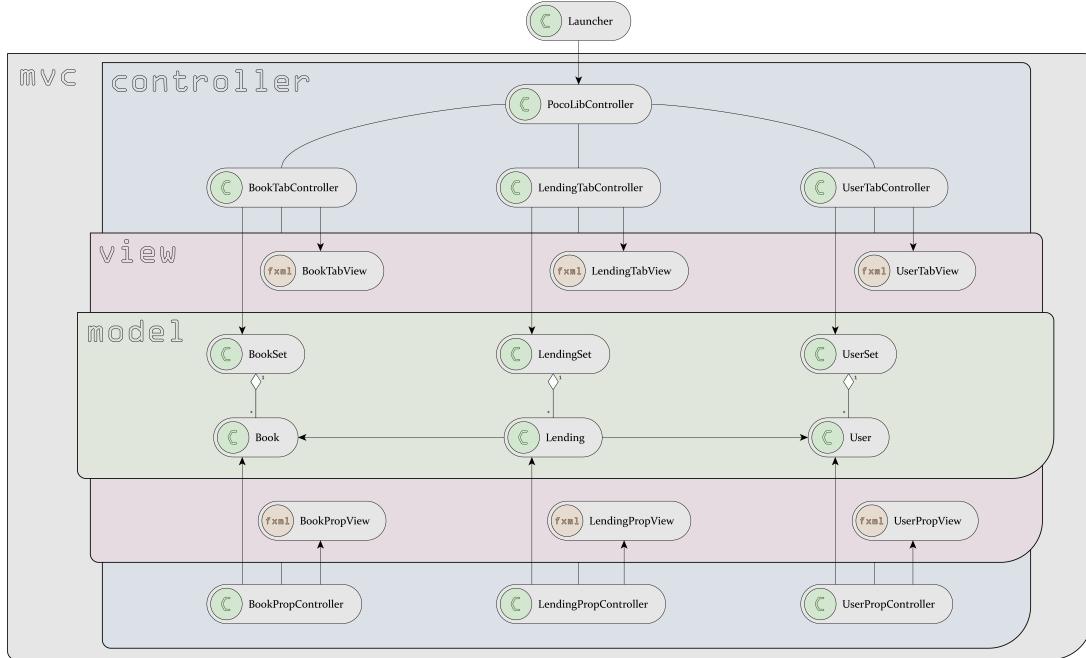


Figura 3: Class Diagram MVC

3.2.1 Classi di mvc.model

Il model contiene le classi che implementano la logica di business. Le principali classi sono:

- **Book:** È responsabile della rappresentazione di un oggetto libro con attributi come ISBN (che può contenere qualsiasi tipo di identificatore univoco, come discusso in SRS sez. 1.2.4), titolo, autori, anno di pubblicazione e numero di copie disponibili. Fornisce metodi per accedere e modificare questi attributi.
- **User:** È responsabile della rappresentazione di un utente con attributi come ID, nome, cognome, email e numero di telefono. Fornisce metodi per accedere e modificare questi attributi.
- **Lending:** È responsabile della rappresentazione di un prestito, collegando un **Book** a un **User** con attributi come data di prestito e data di restituzione prevista.
- **BookSet**, **UserSet**, **LendingSet**: Collezioni responsabili della gestione rispettivamente di libri, utenti e prestiti. Forniscono metodi per aggiungere, rimuovere e cercare elementi all'interno delle collezioni.

3.2.2 Classi di mvc.controller

Il controller è responsabile del collegamento tra il model (la logica di business) e la view (la logica di presentazione). Le principali classi sono:

- **BookPropController, UserPropController, LendingPropController:** Gestiscono le operazioni di visualizzazione e modifica delle proprietà. I metodi permettono di gestire azioni come visualizzazione dettagli, salvataggio e annullamento modifiche.
- **BookTabController, UserTabController, LendingTabController:** Gestiscono le operazioni sulle tabelle principali. I metodi permettono di gestire interazioni come reindirizzamento ai dettagli o aggiornamento dati.
- **PocoLibController:** È responsabile della gestione delle interazioni generali dell'applicazione, coordinando le operazioni tra i vari controller specifici di entità.

3.2.3 Artifacts di mvc.view

La view contiene i file FXML che implementano l'interfaccia grafica dell'app.

- **BookPropView, UserPropView, LendingPropView:** Finestre di visualizzazione e modifica delle proprietà.
- **BookTabView, UserTabView, LendingTabView:** Tab corrispondenti alla visualizzazione e interazione con le tabelle.

3.2.4 Esempio delle interazioni di un Model-View-Controller specifico (User-Set)

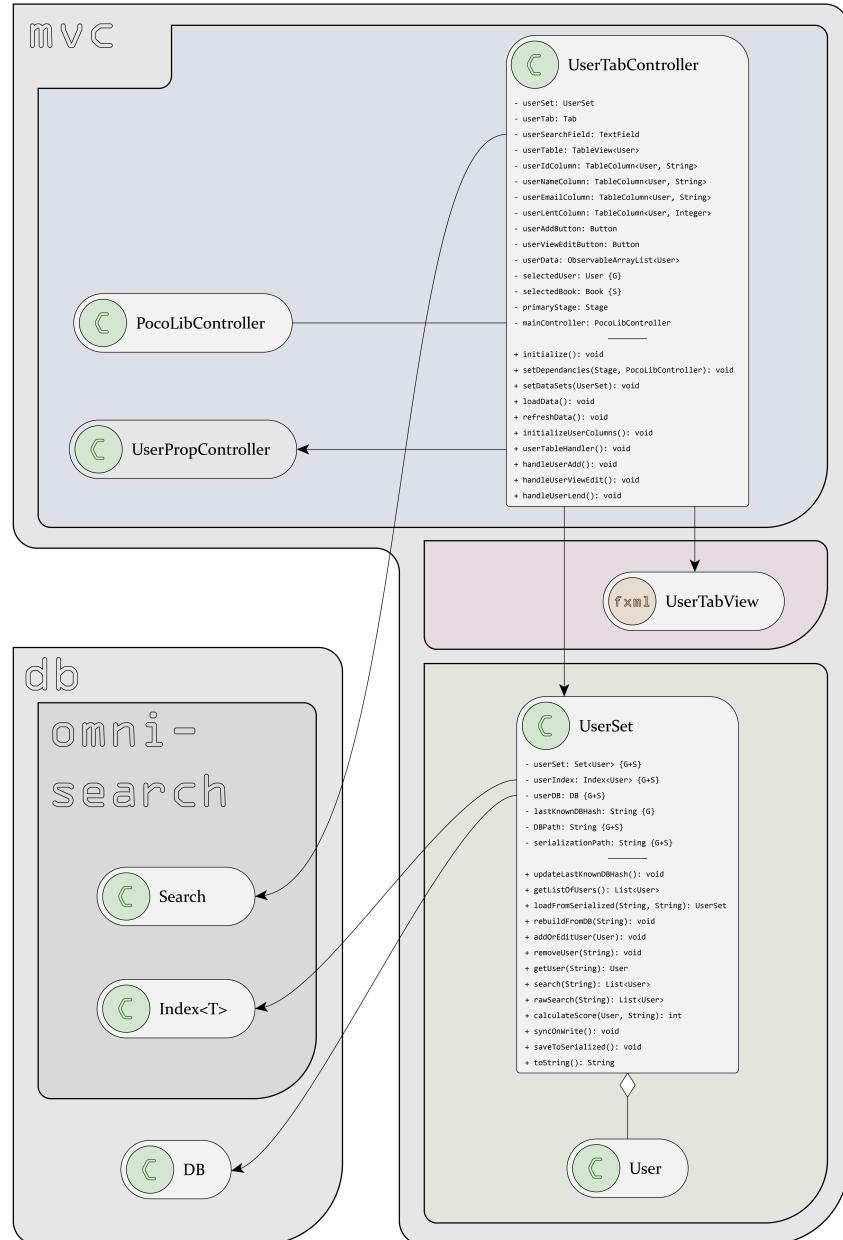


Figura 4: Esempio interazioni MVC specifico (UserSet)

3.2.5 Esempio delle interazioni di un modello e della sua aggregazione (Book e BookSet)

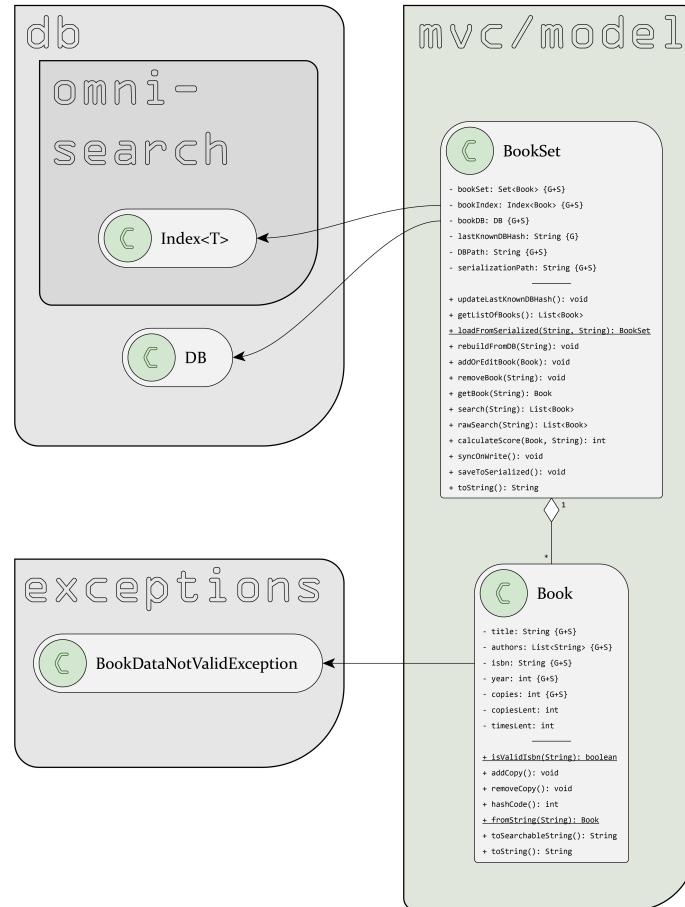


Figura 5: Esempio delle interazioni Book e BookSet

3.3 Diagramma delle classi di db e db.omnisearch

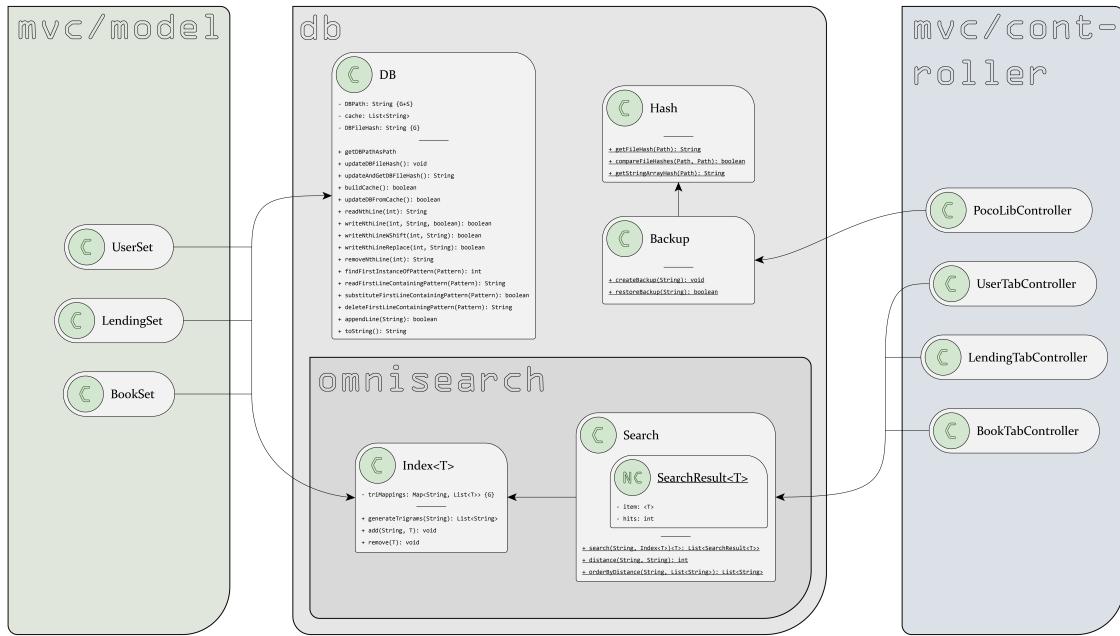


Figura 6: Class Diagram DB

3.3.1 Classi di db

Il package `db` è responsabile della persistenza dei dati e della gestione dei backup.

- **DB**: Responsabile della creazione, lettura, scrittura e aggiornamento del file di database. I metodi principali prevedono la costruzione e aggiornamento del file da una Cache.
- **Backup**: Responsabile della creazione effettiva dei backup del file di database.
- **Hash**: Responsabile del calcolo dell'hash del file di database per garantire l'integrità dei dati.

3.3.2 Classi di db.omnisearch

Il package `db.omnisearch` implementa la funzionalità di ricerca avanzata.

- **Index**: Responsabile dell'indicizzazione trigram per ricerche più efficienti.
- **Search**: Implementa algoritmi di ricerca fuzzy per trovare corrispondenze parziali, permettendo ricerche tolleranti agli errori di battitura.

3.4 Diagramma delle classi di exceptions

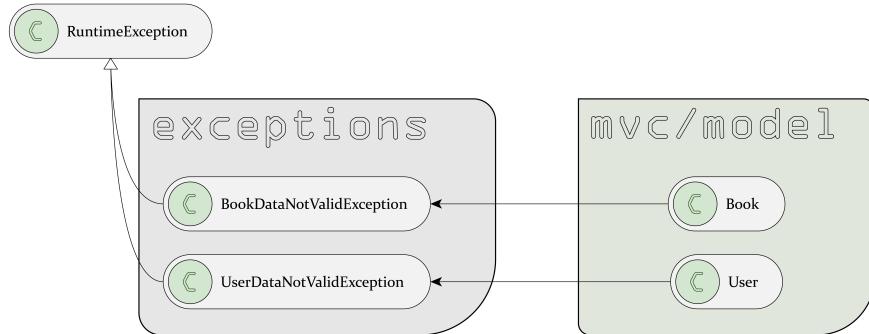


Figura 7: Class Diagram Exceptions

3.4.1 Classi di exceptions

Il package `exceptions` contiene la gerarchia delle eccezioni personalizzate.

- `BookDataNotValidException`: Sollevata quando i dati di un libro non sono validi.
- `UserDataNotValidException`: Sollevata quando i dati di un utente non sono validi.

Tali eccezioni sono state create poiché in questo modo qualsiasi `model` istanziato conterrà sempre dati validi, nonostante si prevede che il bibliotecario non avrà possibilità di salvare un nuovo `model` se la validazione dei dati al momento dell'input risulterà errata. Il loro inserimento è correlato quindi all'idea che i `model` sono un componente funzionante autonomamente, poi interfacciato con la GUI.

3.5 Scelte Progettuali

Il sistema è stato progettato con l'obiettivo di garantire manutenibilità, modularità e facilità di estensione.

- **Adozione del pattern MVC:** Isola la logica di business, la logica di presentazione e la gestione degli eventi.
- **Progettazione orientata agli oggetti:** Segue principi come encapsulamento, ereditarietà e polimorfismo.
- **Implementazione di un sistema di ricerca avanzata:** L'uso di indici trigram e algoritmi di fuzzy search offre funzionalità di ricerca potenti.

- **Gestione della persistenza:** Classi dedicate (DB, Backup, Hash) isolano la logica di accesso ai dati.
- **Utilizzo di eccezioni personalizzate:** Facilitano la gestione degli errori e migliorano la robustezza.

3.6 Principi di buona progettazione

Il sistema rispetta diversi principi di buona progettazione software:

- **Single Responsibility Principle (SRP):** Ogni classe ha una singola responsabilità ben definita (es. DB serializzazione, Backup backup).
- **Don't Repeat Yourself (DRY):** La logica di validazione è centralizzata nel model; la ricerca fuzzy è implementata una sola volta in `db.omnisearch`.
- **Separation of Concerns:** Netta separazione tra View, Model e DB.
- **Information Hiding:** I dettagli implementativi delle strutture dati interne sono nascosti dietro interfacce pubbliche.

4 Modello dinamico

In questa sezione vengono presentati i diagrammi UML per i casi d'uso più significativi dell'applicazione.

4.1 Diagramma di sequenza: Aggiunta di un nuovo libro

Nel seguente diagramma viene illustrato il processo di aggiunta di un nuovo libro. I dati vengono elaborati dal controller, salvati nel model, e successivamente su file tramite la classe DB, creando anche un backup. In caso di errori, viene sollevata un'eccezione gestita dal controller.

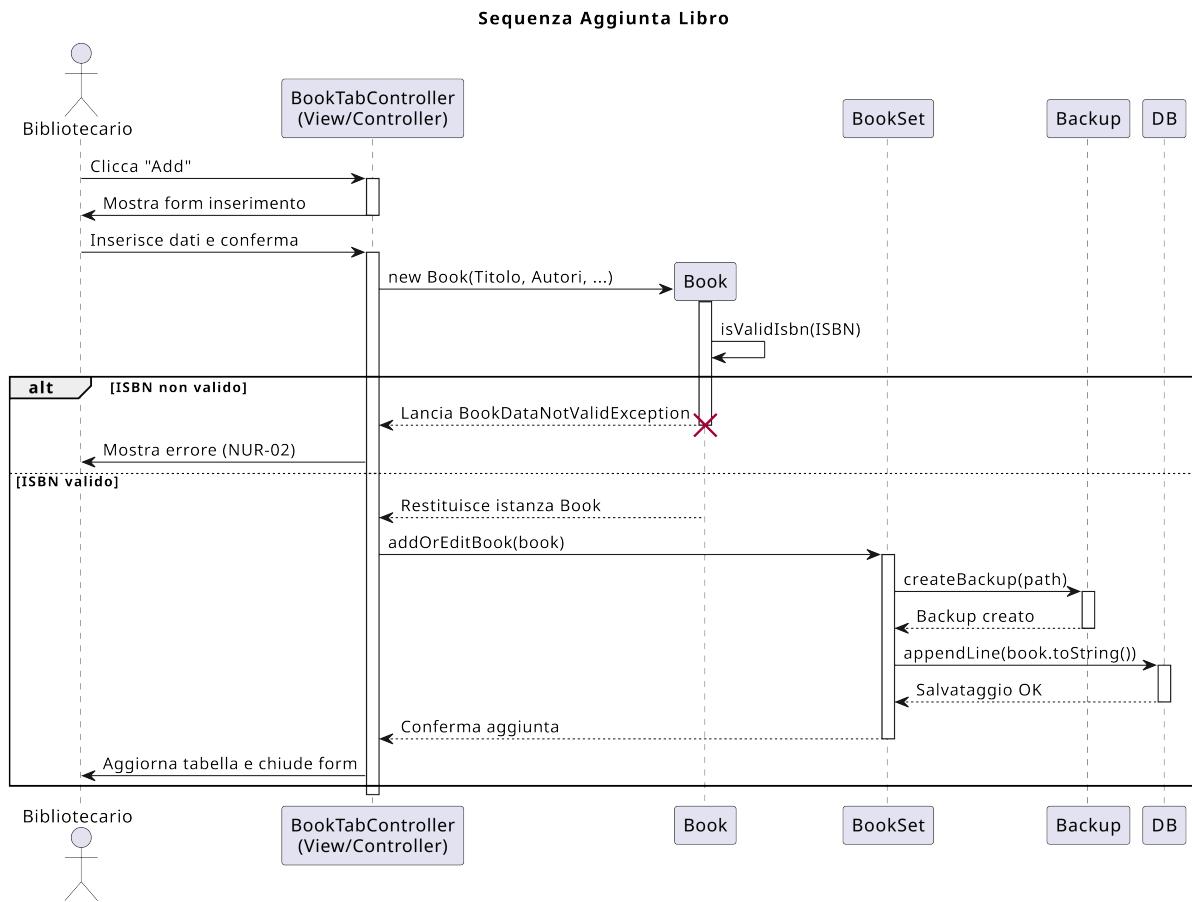


Figura 8: Sequence Diagram: Add Book

4.2 Diagramma di sequenza: Prestito di un libro

Il bibliotecario seleziona utente e libro. Il controller verifica le condizioni (limite prestiti, disponibilità libro). Se soddisfatte, viene creato il prestito, aggiornato lo stato del libro e salvato tutto su file (con backup). In caso contrario, viene sollevata un'eccezione.

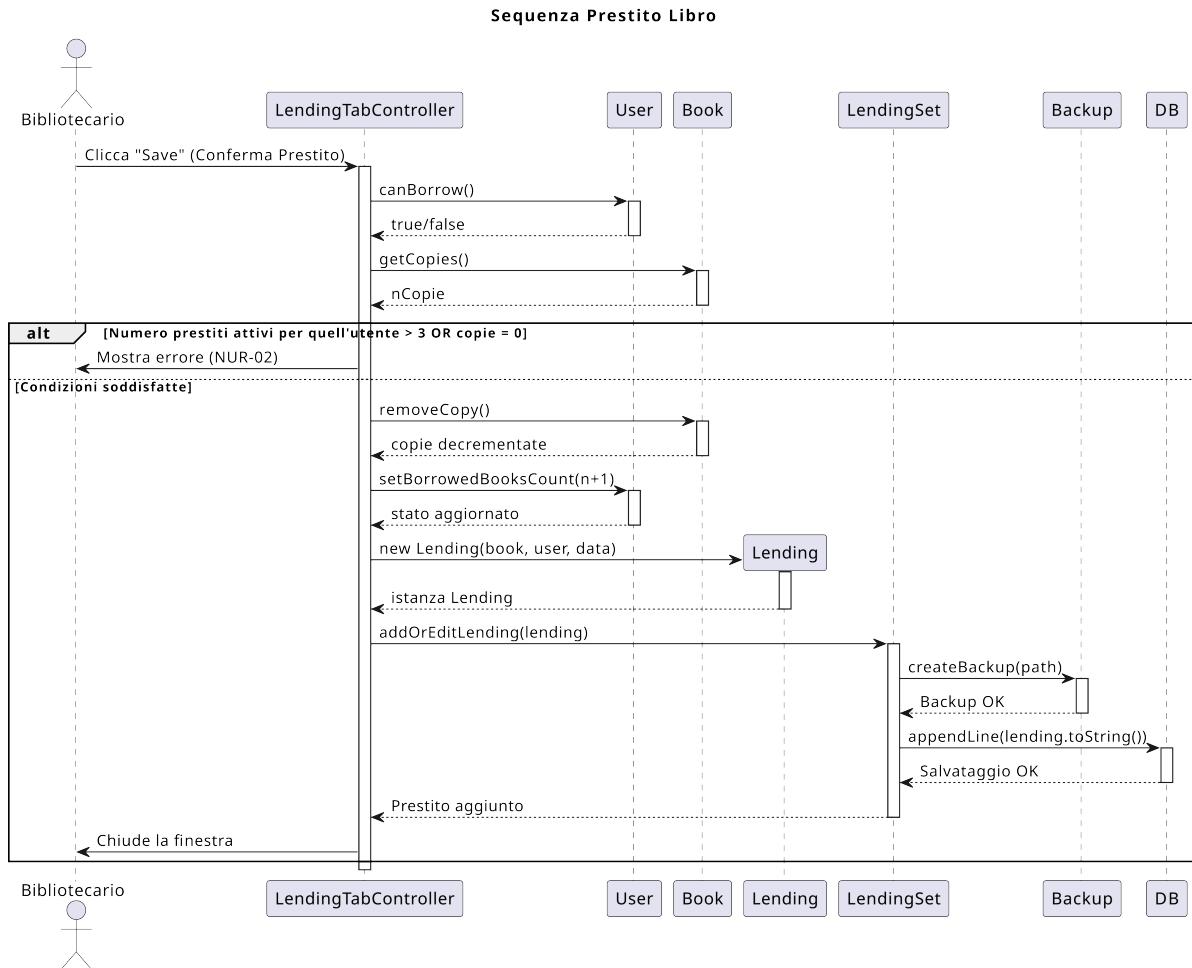


Figura 9: Sequence Diagram: Lend Book

4.3 Diagramma di Attività: Avvio dell’Applicazione e Caricamento Dati

All'avvio, l'app verifica l'esistenza del file di salvataggio. Se non esiste, ne crea uno vuoto. Se esiste, tenta il caricamento. In caso di fallimento del caricamento, tenta il ripristino da backup. Se anche il ripristino fallisce, restituisce errore critico e inizializza un nuovo file vuoto.

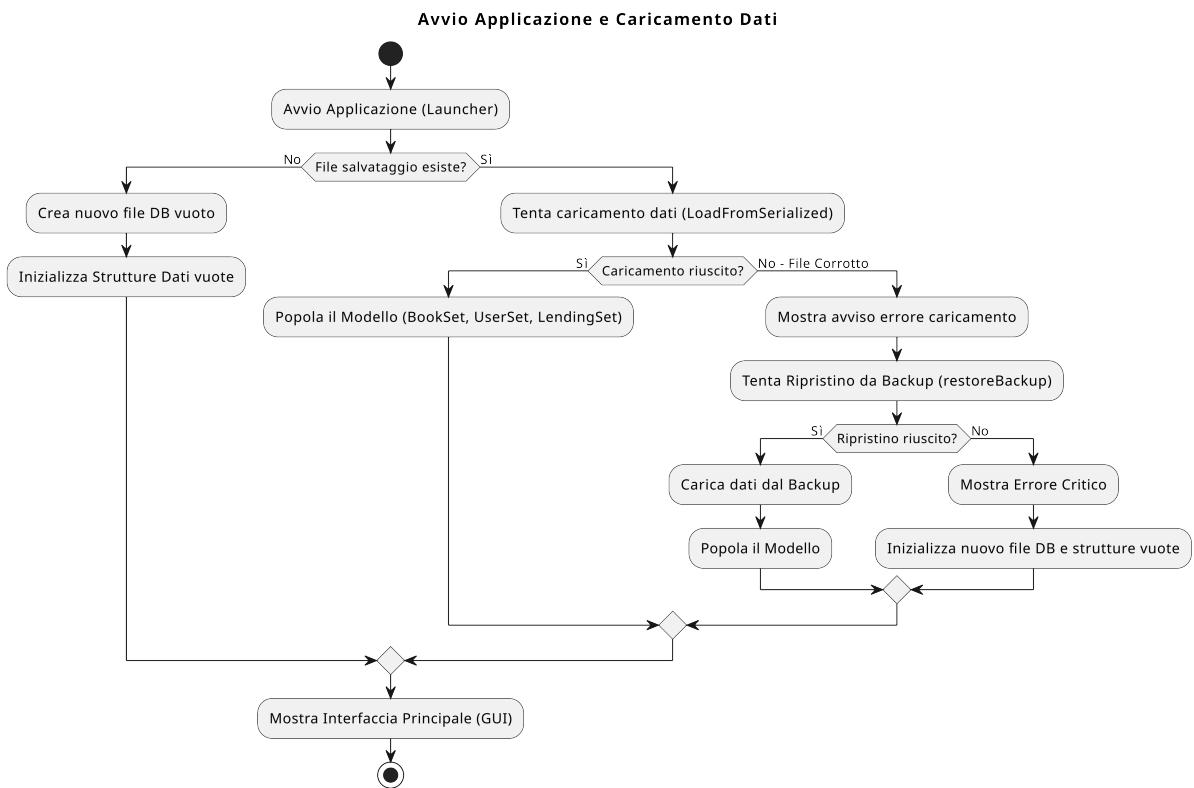


Figura 10: Activity Diagram: Startup

5 Design dell'interfaccia utente

L'interfaccia utente è progettata per essere intuitiva e user-friendly, con un layout chiaro e funzionale.

5.1 Schermate principali dell'applicazione

La schermata principale presenta una tabella che elenca i contenuti in base alla sezione selezionata (Prestiti, Libri, Utenti) tramite barra di navigazione. Include funzionalità di ricerca testuale (anche parziale/fuzzy) e bottoni per azioni CRUD.

Pagina dei Prestiti

Permette di visualizzare: Lending ID, Return Date, ISBN, Title, User ID, User. Include bottoni per segnare come restituito un libro e per visualizzare/modificare un prestito.

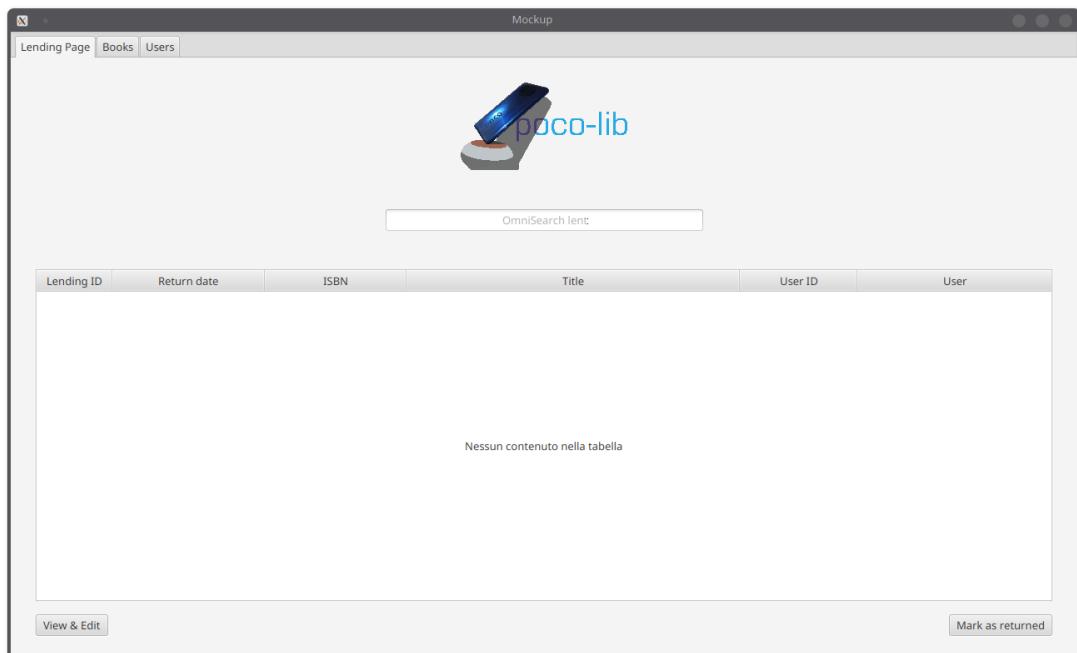


Figura 11: Main Page: Lending

Pagina dei Libri

Permette di visualizzare: ISBN, Title, Authors, Year, Available, Lent. Include bottoni per aggiungere, visualizzare/modificare e prestare libri.

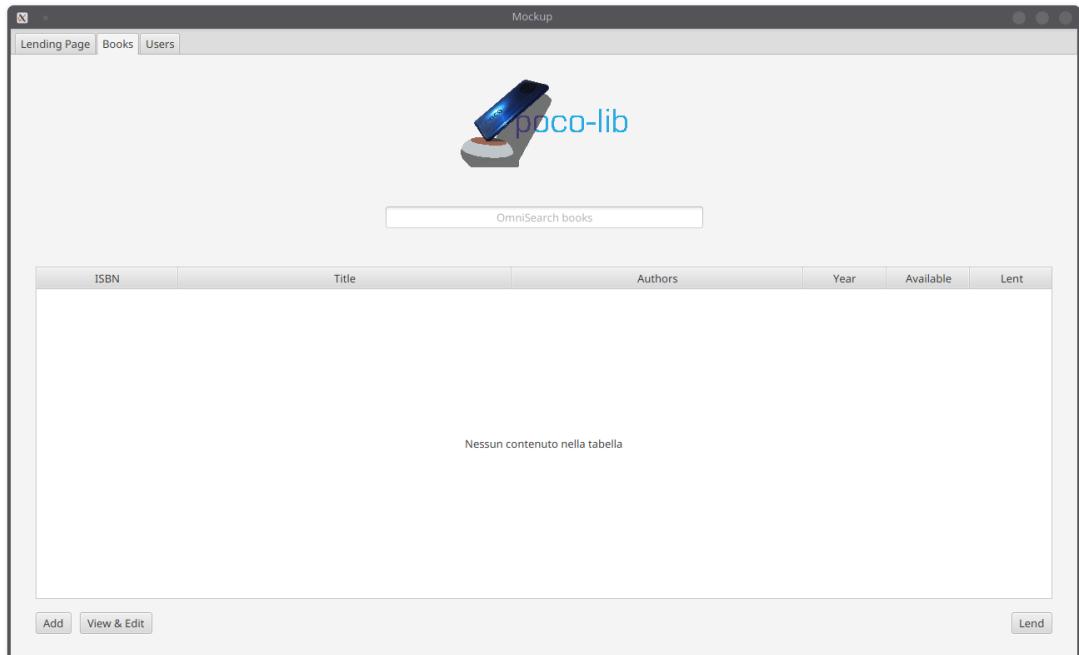


Figura 12: Main Page: Books

Pagina degli Utenti

Permette di visualizzare: ID, Name, Surname, Email, Lent (numero libri in prestito). Include bottoni per aggiungere, visualizzare/modificare e avviare prestiti per l'utente.

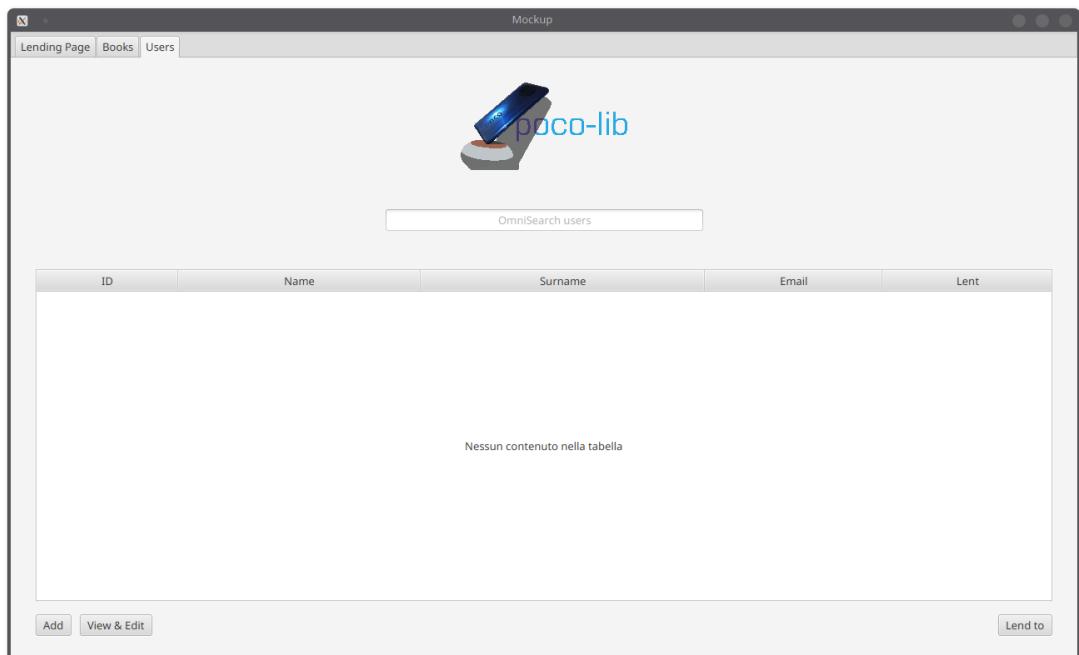


Figura 13: Main Page: Users

5.2 Finestre di visualizzazione

Finestre aperte tramite “View and Edit”. Mostrano dettagli e offrono collegamenti rapidi agli elementi collegati.

- **Lending View:** Mostra dettagli prestito. Bottoni per modifica, eliminazione (se restituito), restituzione libro.
- **Book View:** Mostra dettagli libro. Bottoni per modifica, eliminazione (se non in prestito), avvio prestito.
- **User View:** Mostra dettagli utente. Bottoni per modifica, eliminazione (se senza prestiti), avvio prestito.

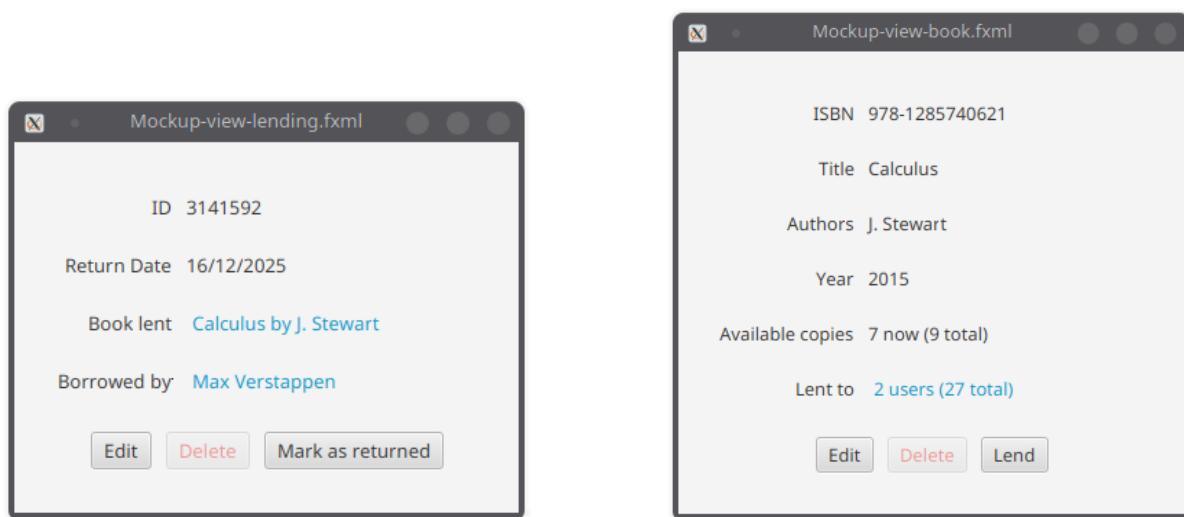


Figura 14: Finestre di visualizzazione: Prestito (sx) e Libro (dx)

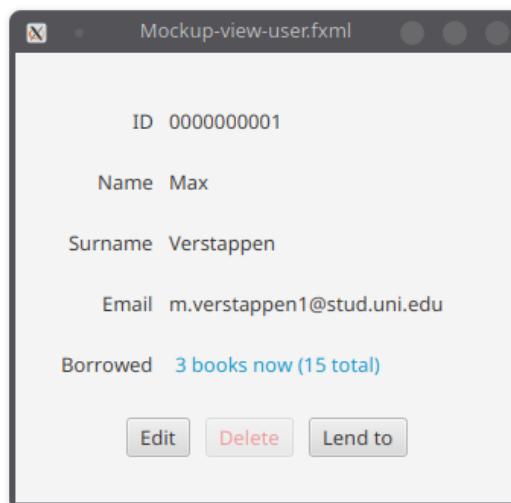


Figura 15: Finestra di visualizzazione: Utente

5.3 Finestre di modifica

Finestre aperte tramite bottone “Edit” nelle schermate di visualizzazione. Permettono di salvare le modifiche ai dati.

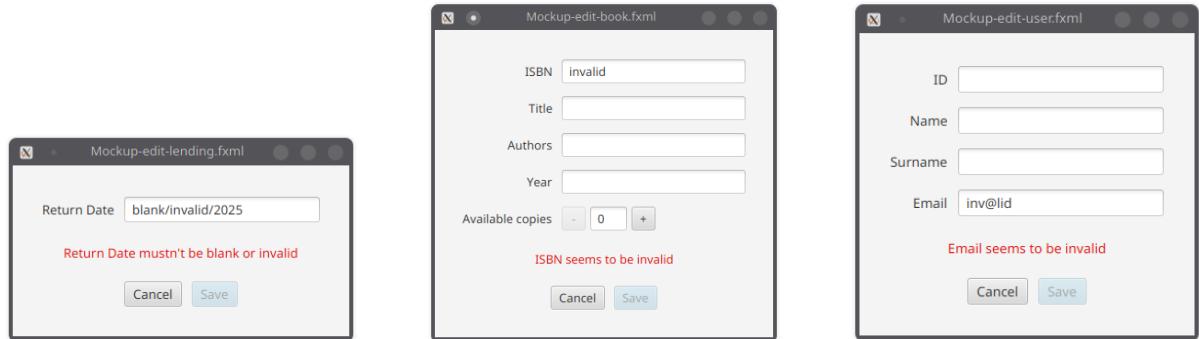


Figura 16: Finestre di modifica: Prestito, Libro, Utente