Sistema di Gestione Biblioteca Universitaria

Documento di Progettazione

Corso d’ingegneria del software - Gruppo 28
Redatto il 08/12/2025
Aggiornato il 15/12/2025

Riferimento: Documento di specifica dei requisiti

Autori:
Luca Maresca
Angelo Palmieri
Deborah Tramontano
Alessandro Socci

Indice

1. Introduzione

Questo documento descrive nel dettaglio la progettazione software dell'applicazione per la gestione di una biblioteca universitaria, basata sui requisiti funzionali e non funzionali forniti nel documento di specifica dei requisiti. L'obiettivo è fornire tutte le informazioni necessarie per l'implementazione, i test e la consegna del sistema in Java con Maven, con memorizzazione basata su file.

2. Scopo del documento

Fornire una guida tecnica esaustiva che copra: architettura, modello dei dati, diagrammi UML principali (diagrammi di classi e di sequenza), specifiche delle classi Java, gestione della persistenza su file, regole di business (vincoli prestiti, controlli duplicati), scenari di errore, test plan e strategie di build/distribuzione.

3. Vincoli e assunzioni

3.1 Vincoli (derivati dal documento requisiti)

- Deve essere fornita una GUI.
- Codice identificativo del libro univoco (ISBN).
- Matricola utente univoca.
- Archivio (libri, utenti, prestiti) salvato su file (un file per categoria suggerita).
- Realizzazione in linguaggio Java con Maven.

3.2 Assunzioni di progetto

- Applicazione desktop Java (JavaFX).
- Numero massimo standard considerato nelle performance: fino a 10.000 libri.
- Un unico attore umano (bibliotecario) con login semplificato (password locale opzionale).
- Concorrenza limitata: ipotesi di singolo utente operativo alla volta; tuttavia si affrontano problemi minimi di consistenza file tramite locking a livello di processo.

4. Panoramica dell'architettura

4.1 Scelta architetturale

Per lo sviluppo del progetto abbiamo scelto un’architettura a tre layer logici, basata sul pattern MVC (Model View Controller):

- **Data Access Layer (Model)**: gestione della persistenza su file.
- **Presentation Layer (View)**: JavaFX, organizza tre sezioni: Libri, Utenti, Prestiti.
- **Application Layer (Controller)**: logica di business, validazioni, regole prestiti.

4.2 Diagramma componenti (testuale)

➔ GUI (JavaFX)

- ◆ Sezione Libri
- ◆ Sezione Utenti
- ◆ Sezione Prestiti

➔ Services

- ◆ BookService
- ◆ UserService
- ◆ LoanService

➔ Repositories

- ◆ BookRepository (file binario: books.bin)
- ◆ UserRepository (file binario: users.bin)
- ◆ LoanRepository (file binario: loans.bin)

➔ Utils

- ◆ DateUtils

4.3 Pattern architetturali adottati

- **Repository** per accesso ai dati.
- **Singleton** per **`BibliotecaManager`** (facilita accesso centralizzato ai servizi).
- **Factory** per creazione di oggetti da archivio.
- **Observer** semplice per aggiornare le viste quando cambiano i dati.

5. Design dei componenti

5.1 Servizi principali

- **`BookService`:** Gestisce le operazioni riguardanti i libri e si occupa delle funzioni di: inserimento, modifica, cancellazione, ricerca (per titolo, autore, ISBN) e ordinamento alfabetico.
- **`UserService`**: Gestisce le operazioni riguardanti gli utenti e si occupa delle funzioni di: inserimento, modifica, cancellazione, ricerca (cognome, matricola) e ordinamento.
- **`LoanService`:** Gestisce le operazioni riguardanti i libri e si occupa delle funzioni di: registrazione prestito, registrazione restituzione e visualizzazione elenco prestiti attivi ordinati per data di restituzione.

5.2 Repository

- Operano su file binari (una collezione per file). Forniscono operazioni atomiche che servono a salvare e leggere i dati sui/dai file.

5.3 GUI

L’interfaccia grafica (UI) sarà strutturata secondo i seguenti parametri e componenti:

- JavaFX con controller separati per ogni sezione (MVC-like): **`BookController`**, **`UserController`**, **`LoanController`.**
- Componenti UI: TableView per elenchi, Form per inserimento/modifica, DatePicker per data di restituzione, SearchBar globale.

6. Modello dei dati e formati di persistenza

6.1 Modello dati

Di seguito vengono elencate le principali entità facenti parte del sistema.

➔ **Author**

- nome: **`String`**
- cognome: **`String`**

➔ **Libro**

- id(codice univoco): **`String`** (ISBN)
- titolo: **`String`**
- autori: **`List<Author>`** (lista di author: oggetto contenente nome e cognome degli autori)
- annoPubblicazione: **`int`**
- numeroCopieTotali: **`int`**
- numeroCopieDisponibili: **`int`**

➔ **Utente**

- matricola(codice univoco): **`String`**
- nome: **`String`**
- cognome: **`String`**

- email: **`String`**
- prestitiAttivi: **`List<String>`**

➔ **Prestito**

- id: **`String`**
- codice: **`String`** (collegamento al libro)
- matricola: **`String`** (collegamento all'utente)
- dataPrestito: **`LocalDate`** (data registrazione)
- dataRestituzionePrevista: **`LocalDate`**
- dataRestituzioneEffettiva: **`LocalDate`**
- sanzioneApplicata: **`boolean`**

6.2 Formato file

I file su cui verranno salvati i dati saranno file binari con array di oggetti.

I singoli file saranno:

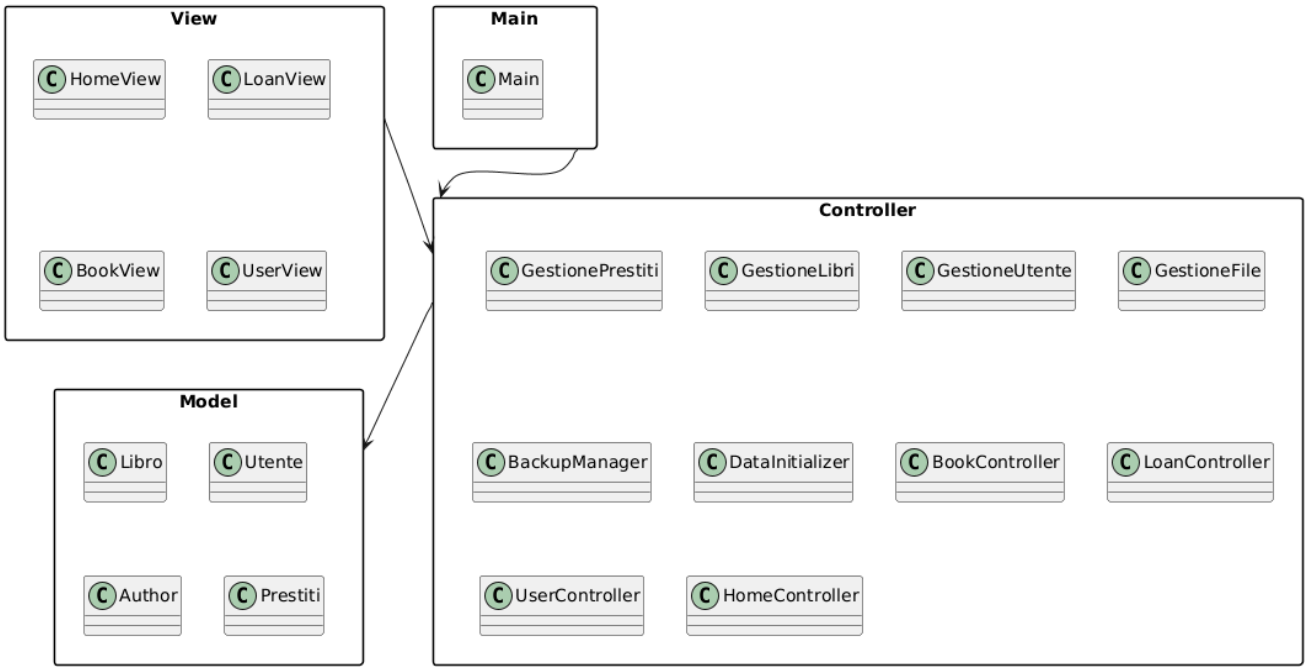
- **`books.bin`** — array di **`Libro`**
- **`users.bin`** — array di **`Utente`**
- **`loans.bin`** — array di **`Prestito`**

I file binari garantiscono sicurezza, in quanto non sono direttamente leggibili dall’utilizzatore del sistema, e velocità nelle operazioni, a discapito dello spazio di archiviazione utilizzato dal sistema per memorizzarli.

7. Diagrammi (package, classi, sequenza)

7.1 Diagramma dei package

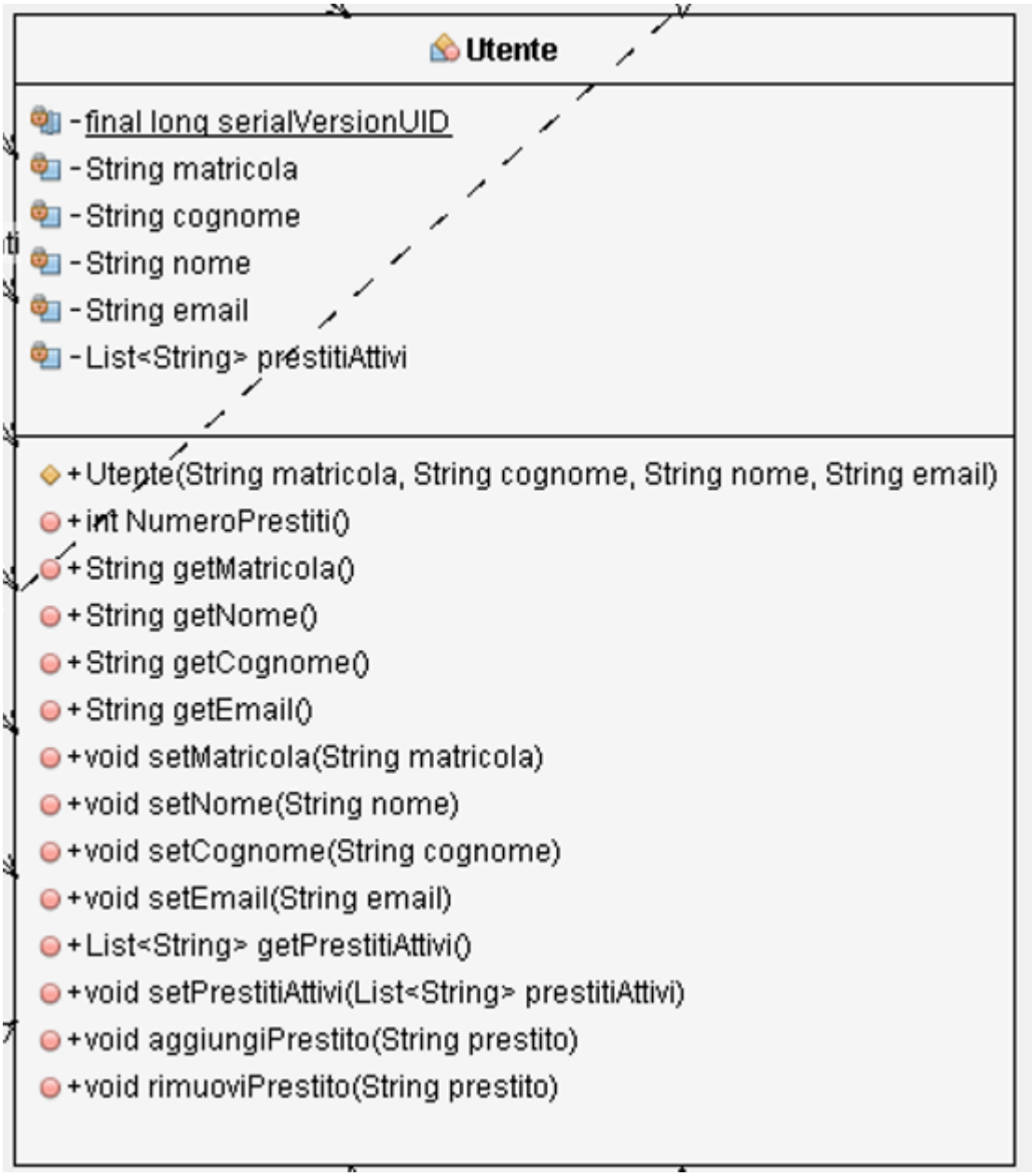
Il seguente diagramma descrive come verranno organizzate le classi in packages e come sarà organizzata, di conseguenza, l'architettura dell'applicazione



7.2 Diagramma delle classi

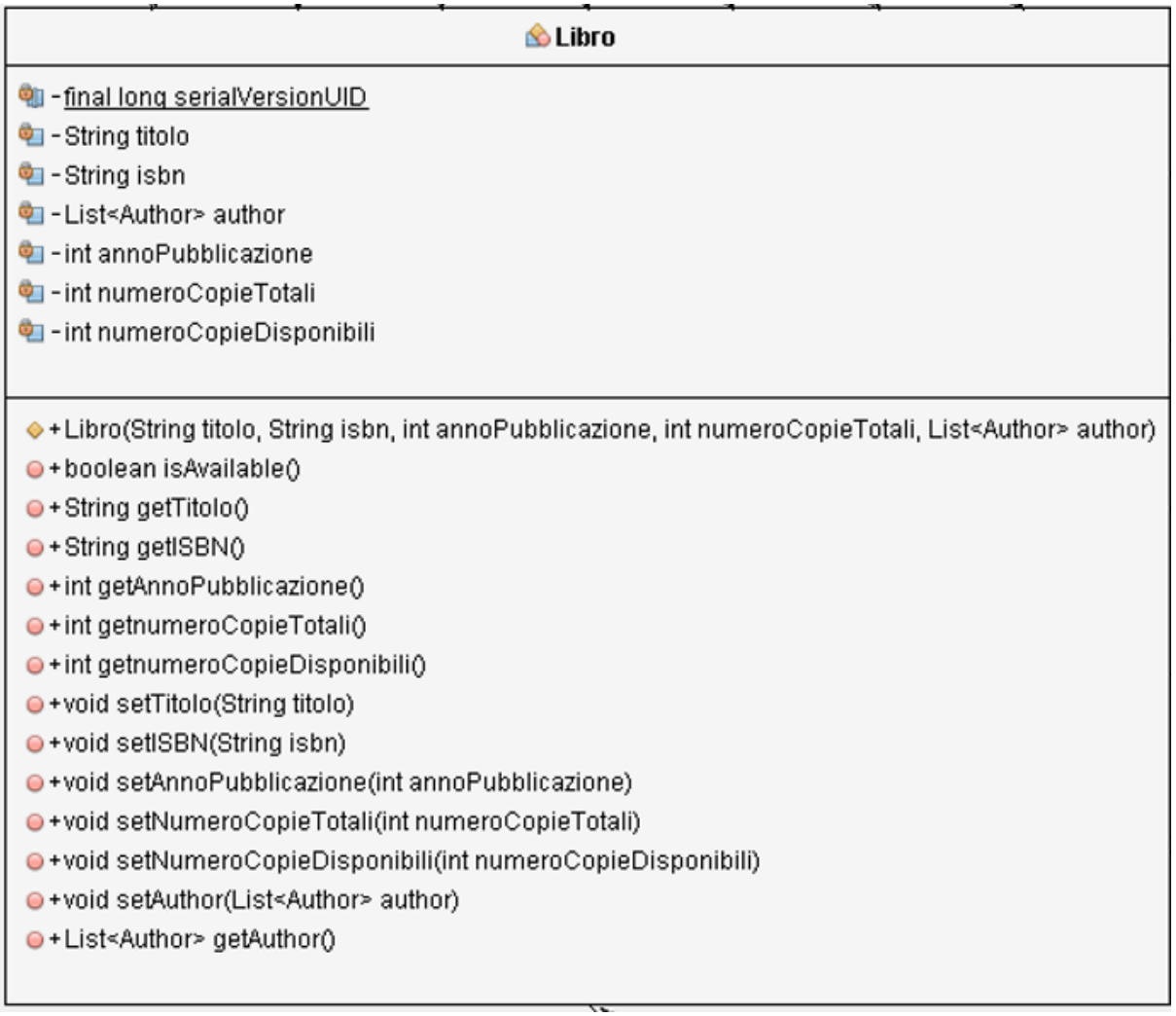
Il diagramma delle classi completo è stato caricato a parte, per via delle sue dimensioni. Per visualizzarlo vedere l'allegato "DiagrammaDelleClassi".
Si nota come le classi appartenenti allo stesso package abbiano similitudini nella loro struttura, di seguito sono spiegate le classi nel dettaglio e le rispettive funzionalità:

Classe utente:



La classe Utente descrive le persone che si registreranno al sistema della biblioteca, tramite il bibliotecario, per poter usufruire dei servizi che essa offre (come ad esempio il servizio di prenotazione libri), La classe si compone di attributi identificativi coerenti con quanto specificato nel SRS, e di metodi setter e getter che permettono l'implementazione delle funzionalità dell'applicazione. Inoltre è presente anche il metodo "numeroPrestiti()", che restituisce il numero di prestiti attualmente in corso di un determinato utente.

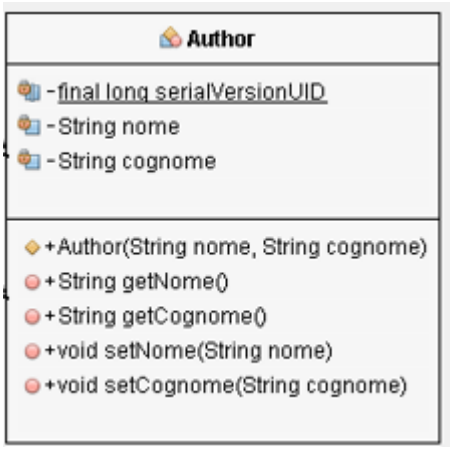
Classe libro:



La classe Libro descrive i libri che veranno gestiti dalla biblioteca e fornisce i metodi per salvare e recuperare tutti i dati relativi ad essi. Proprio come la classe Utente, è composta da metodi setter e getter standard e inoltre di un metodo particolare

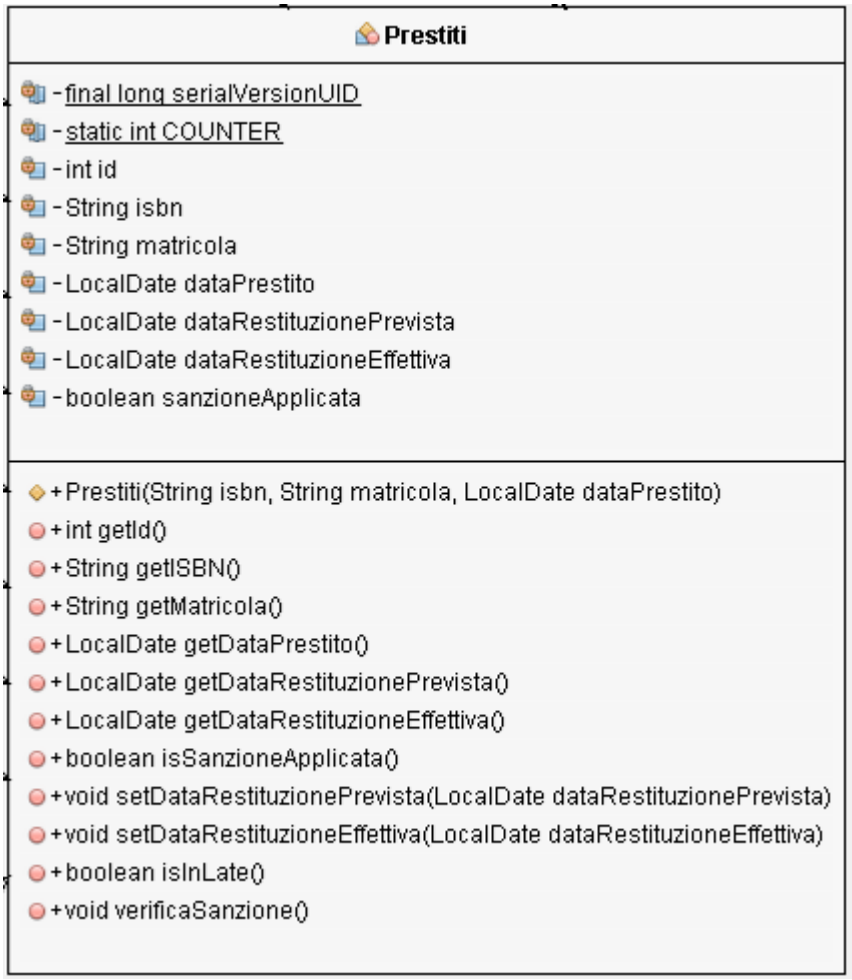
“IsAvaliable()” che restituisce come booleano la disponibilità di un libro, ovvero se è presente in biblioteca o meno.

Classe Author:



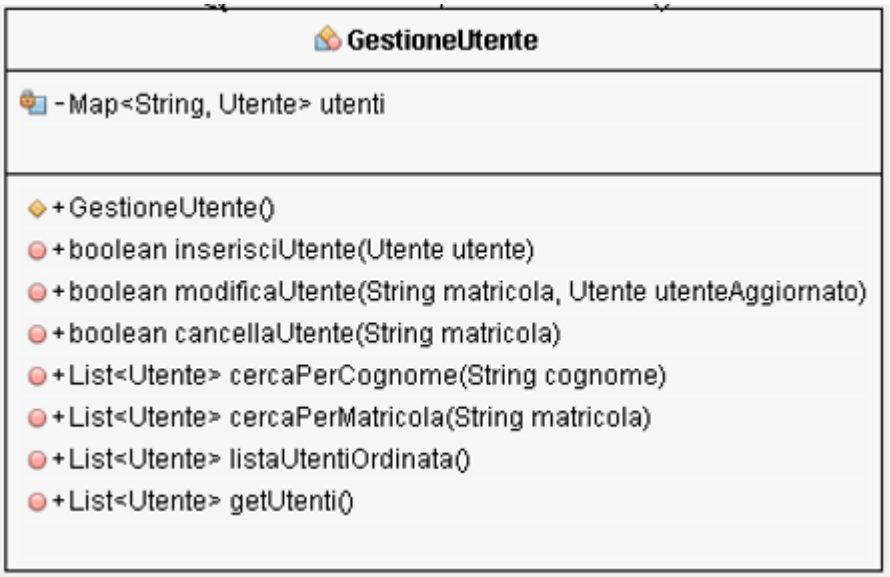
La classe Author serve a contenere le informazioni riguardanti gli autori dei libri, siccome per ogni libro potrebbero esserci più autori, o degli stessi autori potrebbero aver scritto più libri. Si compone di due semplici attributi, che servono ad identificare nome e cognome di ogni autore, e dei metodi getter e setter standard.

Classe Prestiti:



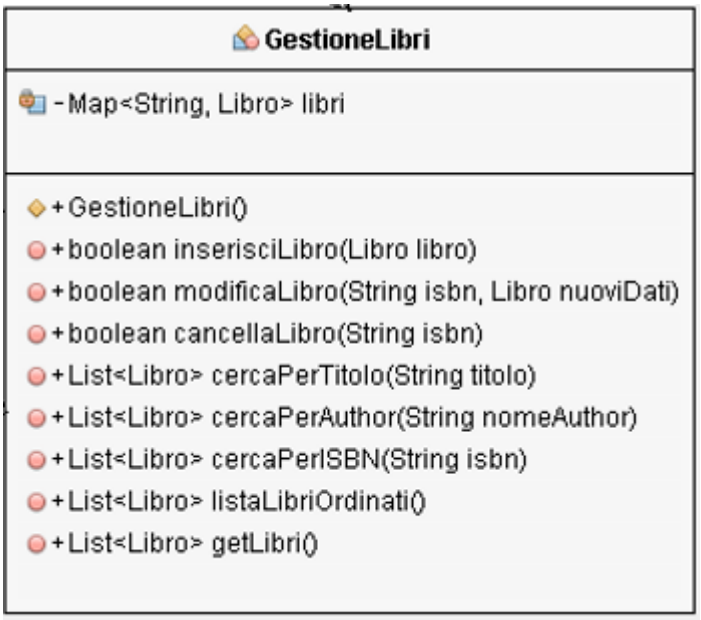
La classe Prestiti descrive i prestiti di libri che verranno effettuati dalla biblioteca, verso gli utenti. Si occupa dunque di tenere traccia di quali libri vengono prestati a quali utenti, della data di restituzione prevista e delle sanzioni applicate in caso che questa non venga rispettata. Per tanto è composta dai relativi attributi autodescrittivi e da metodi setter e getter appositi per ogni attributo. Inoltre è presente un metodo aggiuntivo “isInLate()” che ci dice se la data di restituzione prevista sia già passata o meno (ritornando un valore booleano), facendoci capire se applicare le sanzioni all’avvenuta restituzione.

Classe GestioneUtente:



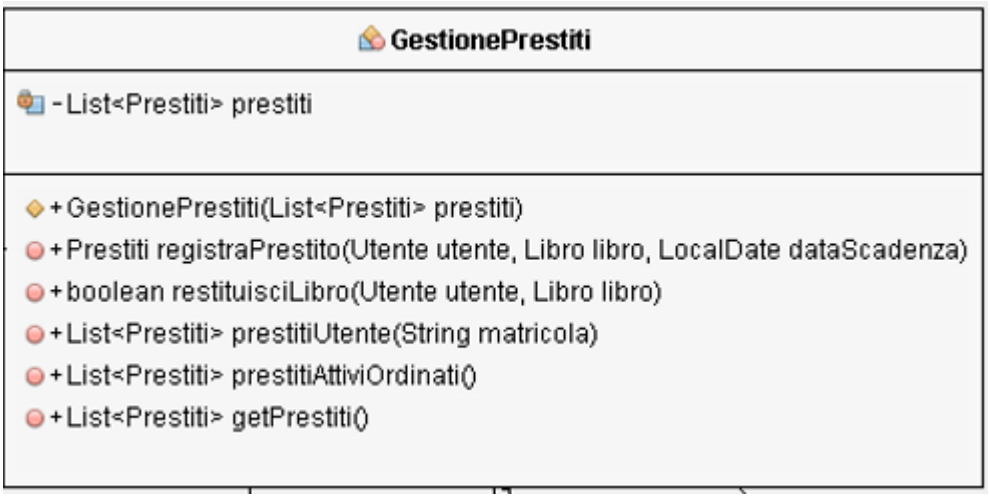
La classe GestioneUtente si occupa di gestire tutti gli oggetti di tipo Utente, fornendo le funzionalità per poter immagazzinare i dati riguardanti essi. Si compone di un unico attributo, la lista completa di utenti registrati, e di 6 metodi autoesplicativi che garantiscono le funzionalità descritte nel SRS.

Classe GestioneLibri:



La classe GestioneLibri, proprio come la classe GestioneUtente, si occupa di gestire gli oggetti di tipo Libro, garantendone le funzionalità per gestire i dati. Si compone di metodi molto simili alla classe Utente, con l'unica differenza che ci sono tre diversi metodi per poter effettuare la ricerca di un libro, specificando parametri diversi.

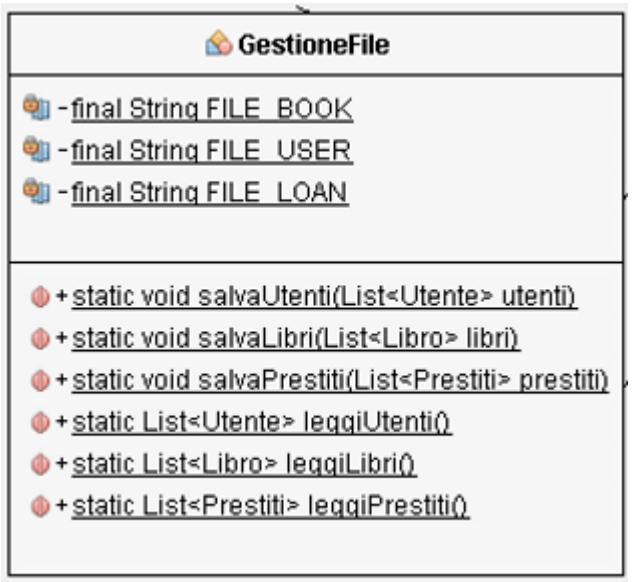
Classe GestionePrestiti:



Proprio come le due precendti, si occupa di gestire oggetti di tipo Prestiti, garantendone le funzionalità. Specifichiamo di seguito la funzione di alcuni metodi che potrebbero risultare ambigui:

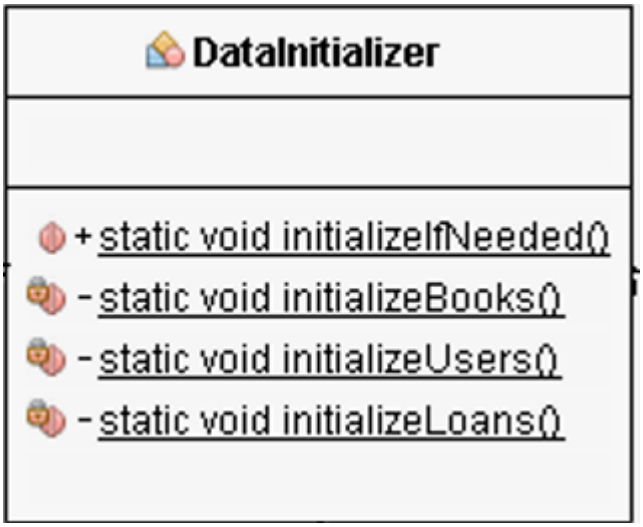
- Il metodo “RestituisciLibro(...)” si occupa di effettuare la registrazione della restituzione di un libro, ripristinando la sua disponibilità in biblioteca e liberando spazio nella lista dei prestiti attivi dell’utente.
- Il metodo “PrestitiUtente(...)” ci restituisce una lista contenente tutti i prestiti attualmente attivi di un determinato utente, il quale è passato come parametro; serve dunque a vedere quali prestiti sono attivi per ogni utente, e in caso di nessun prestito attivo, ritorna una lista vuota.
- Il metodo “PrestitiAttiviOrdinati()” invece restituisce una lista ordinata alfabeticamente di tutti i prestiti, effettuati da tutti gli utenti, attualmente attivi; proprio come per l’altro metodo, in caso di mancanza di prestiti attivi, restituisce una lista vuota.

Classe GestioneFile:



La classe GestioneFile si occupa della lettura e della scrittura dei dati su/da file binari, è composta da metodi autoesplicativi che garantiscono il corretto funzionamento della lettura e scrittura da file.

Classe DataInitializer:

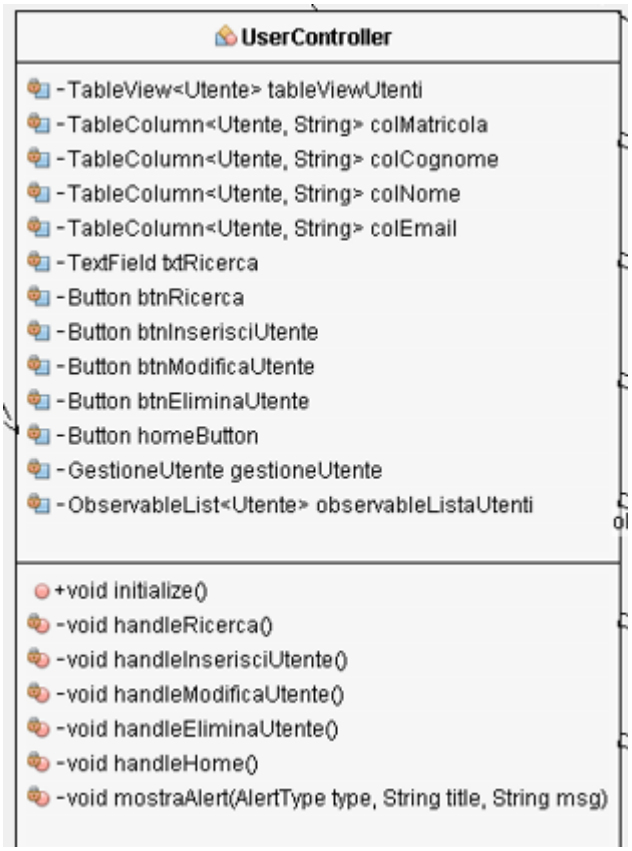


La classe DataInitializer è il centro di tutte le classi appartenenti al package Controller, si occupa infatti di interagire con il Main, e di demaniare tutti i compiti alle corrette classi Controller. Perciò si compone di soli metodi, che garantiscono il corretto coordinamento di tutti i componenti dell'applicazione, secondo il pattern MVC.

Classi View:

Le classi appartenenti al package View, nominativamente “HomeView”, “LoanView”, “UserView” e “BookView” si occupano della visualizzazione a schermo dei dati di ogni rispettiva categoria, garantendone una formattazione adeguata e comprensibile.

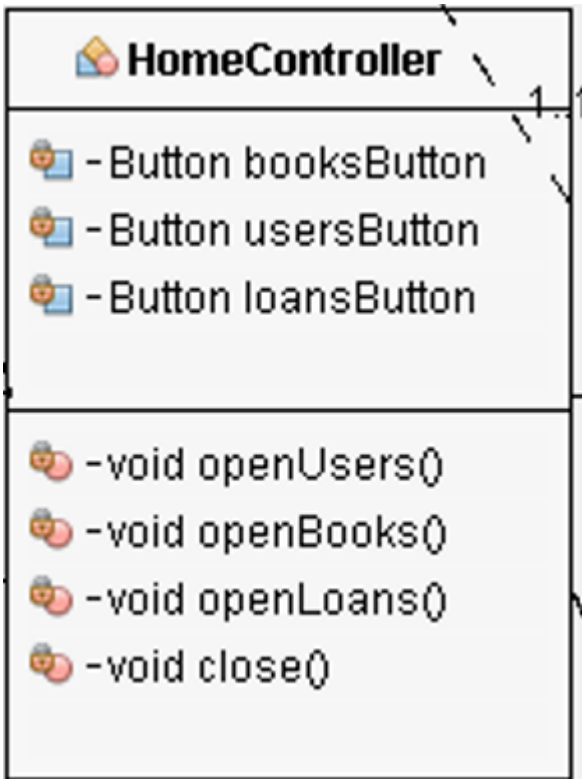
Classi Controller:



UserController gestisce il flusso di dati relativo agli utenti della biblioteca. Riceve i dati inseriti nella UserView, li valida e li inserisci nella GestioneUtenti.

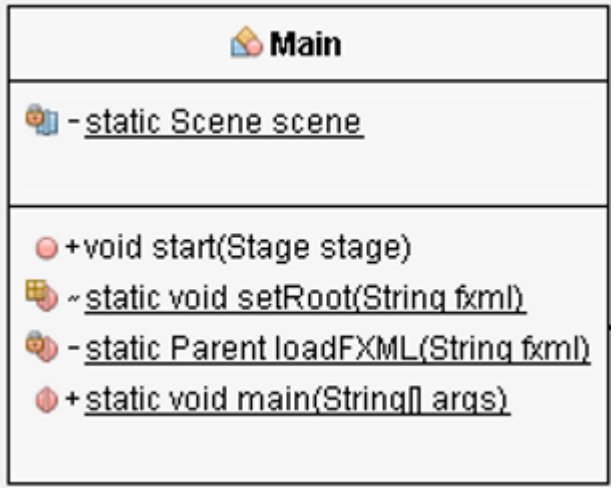
BookController gestisce il flusso di dati relativo ai libri della biblioteca. Riceve i dati inseriti nella BookView, li valida e li inserisci nella GestioneLibri.

LoanController gestisce il flusso di dati relativo ai libri della biblioteca. Riceve i dati inseriti nella LoanView, li valida e li inserisci nella GestionePrestiti.



HomeController gestisce gli eventi provenienti da HomeView,se il bibliotecario clicca uno dei pulsanti(“Prestiti”, “Libri” o “Utenti”), HomeController nasconde la vista attuale e cede il comando al controller richiesto.

Classe Main:



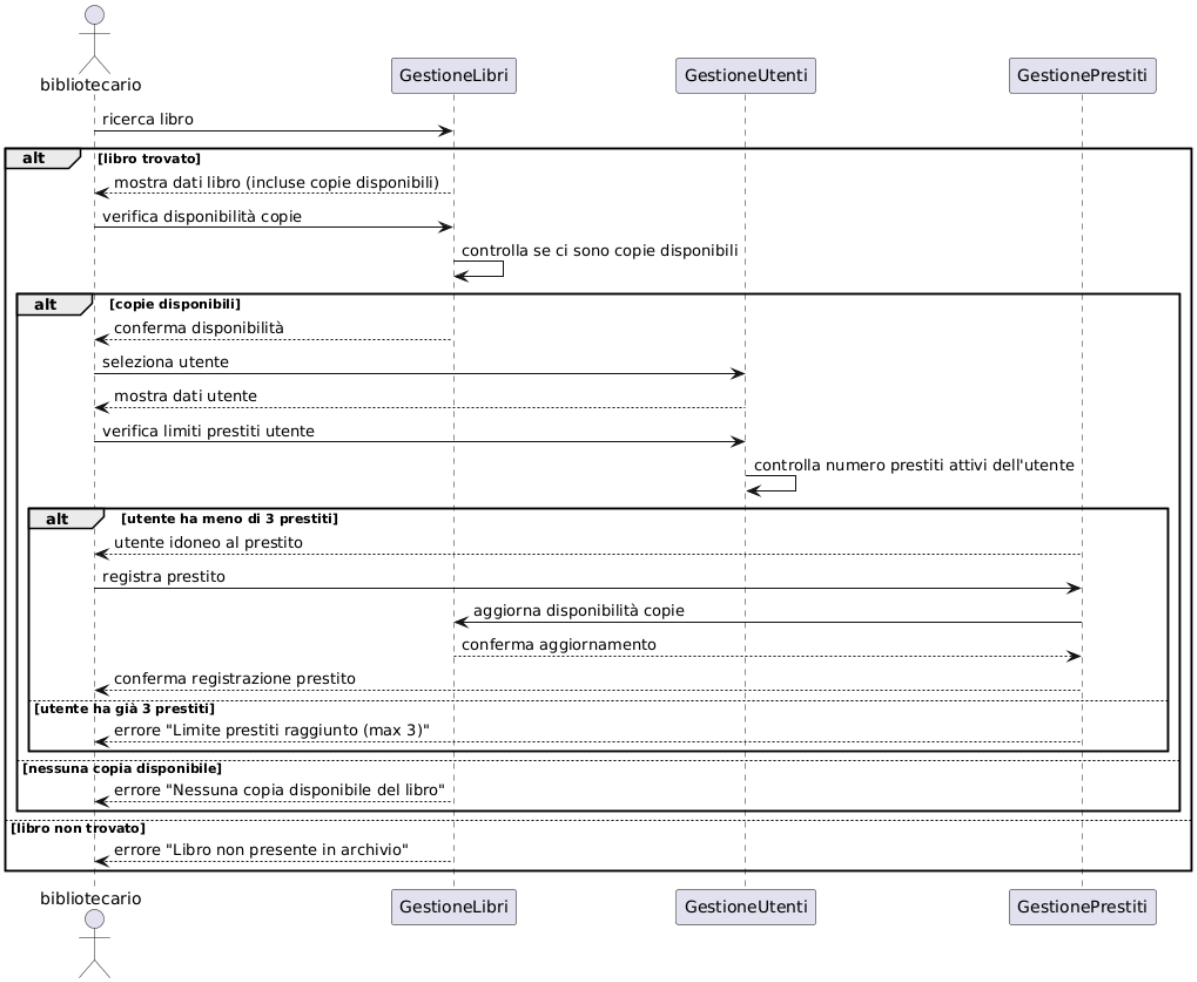
La classe “Main” è la classe principale del progetto, viene eseguita all’avvio dell’applicazione e si occupa di interagire col Controller principale (classe Gestore) e di gestire le eventuali situazioni di errore che possono accadere.

Tutte le classi sono state realizzate tenendo a mente l’implementazione secondo pattern MVC e cercando di ottenere il massimo livello di coesione funzionale (ovvero che ogni funzione faccia svolga solo un’unica atomica responsabilità) ed il minimo livello di accoppiamento, ove possibile, con lo scopo di rendere il codice più facilmente leggibile e manutenibile.

7.3 Diagrammi di sequenza

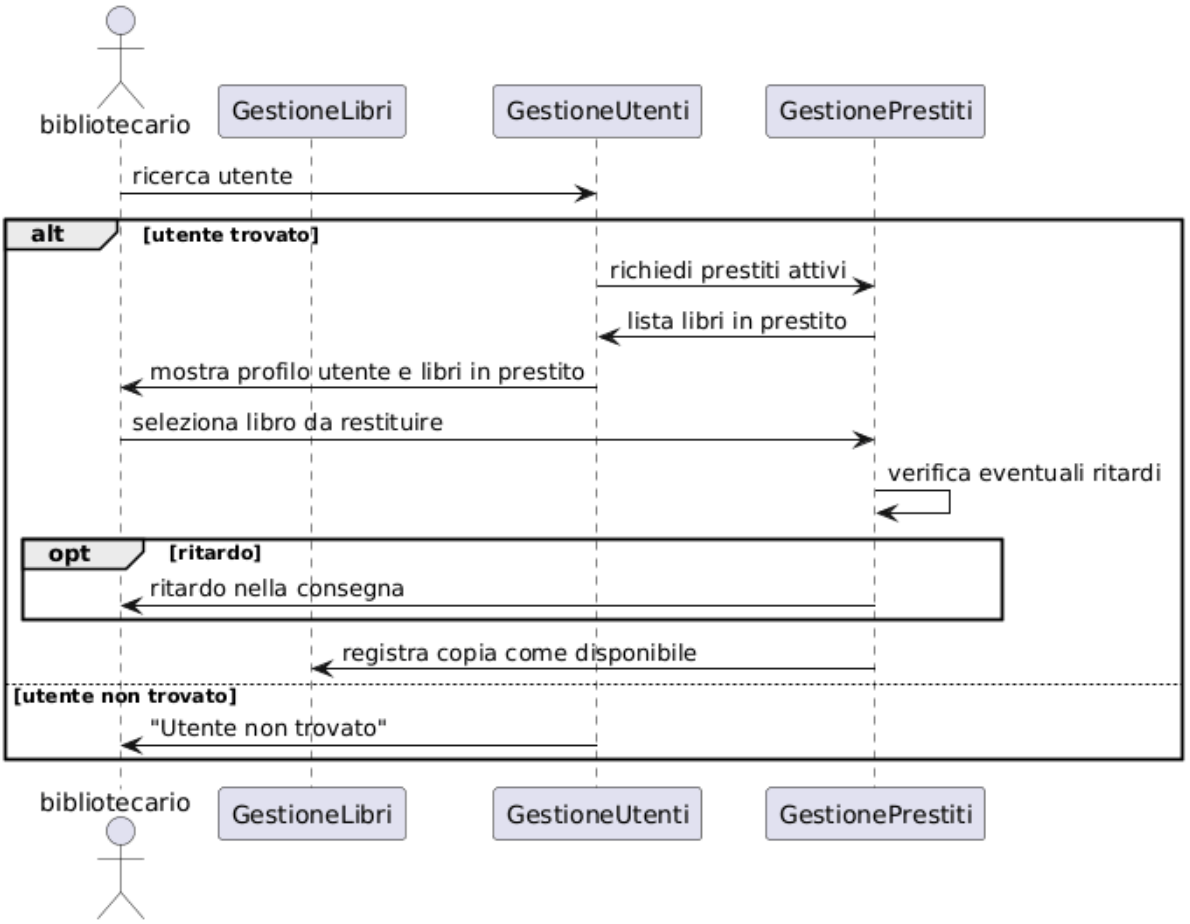
7.3.1 Registrazione Prestito (riassunto)

1. UI -> GestionePrestiti: registraPrestito (isbn, matricola, dataRestituzione)
2. GestionePrestiti -> Prestiti: **`Prestiti(...)`**
3. GestionePrestiti -> Utente: **`getPrestitiAttivi()`** (controllo prestiti)
4. GestionePrestiti ->Libri: **`isAvailable()`** (controllo copieDisponibili)
5. GestionePrestiti crea **`Prestito`**, aggiorna il numero di **`copieDisponibili`** e la lista dei **`prestitiAttivi`** dell’utente
6. Gestore -> GestioneFile: **`salvaPrestiti(...)`**
7. Risposta a UI con esito



7.3.2 Restituzione Libro (riassunto)

1. UI -> GestionePrestiti: **`restituisciLibro(user, loanId)`**
2. GestionePrestiti -> Prestiti
3. Prestiti setta **`dataRestituzioneEffettiva`**, calcola ritardo, setta eventuale sanzione
4. GestionePrestiti aggiorna il numero di **`copieDisponibili`** e **`prestitiAttivi`** dell'utente
5. I cambiamenti vengono effettuati anche nei file



8. Interfaccia utente (wireframes e flussi)

8.1 Struttura generale

L'interfaccia utente si compone di una finestra principale con barra superiore e tre tab:

- Tab Libri: TableView (colonne: Titolo, Autori, Anno, Codice, Copie Disponibili), pulsanti **`Nuovo`, `Modifica`, `Cancella`, `Ricerca`.**
- Tab Utenti: TableView (Nome, Cognome, Matricola, Email, #Prestiti), pulsanti Nuovo, **`Modifica`, `Cancella`, `Ricerca`.**
- Tab Prestiti: TableView (ID, Titolo, Matricola, DataPrestito, DataRestituzionePrevista, Stato), pulsanti **`Registra Prestito`, `Registra Restituzione`.**

8.2 Componenti chiave

- **Barra di ricerca global**: campo testo + dropdown per tipo (Libri/Utenti)
- **DatePicker**: per scelta data di restituzione
- **Highlight ritardi**: ciascuna riga prestito con ritardo deve essere evidenziata

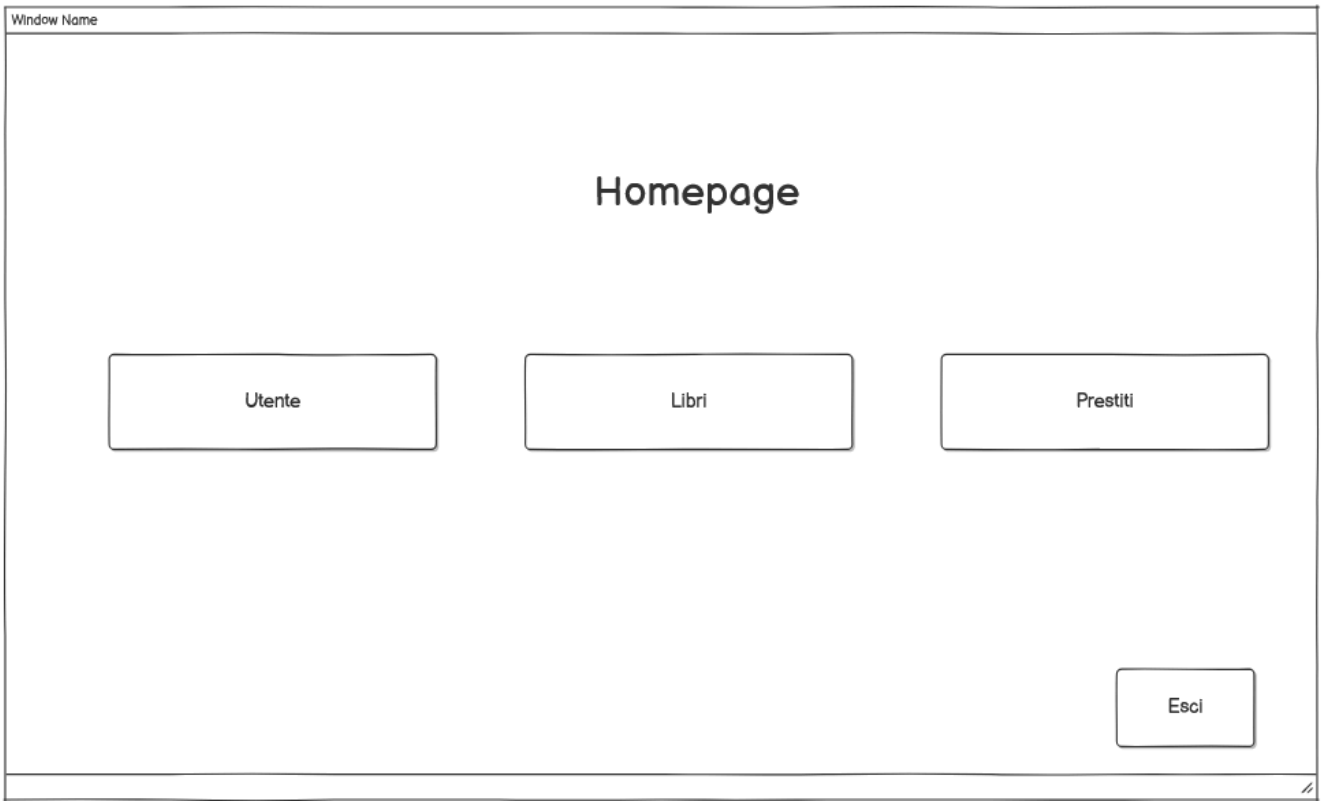
8.3 Messaggi e validazioni

- Messaggi di successo/errore tramite dialog modal.
- Validazioni: codice e matricola univoci, campi obbligatori, formato email.

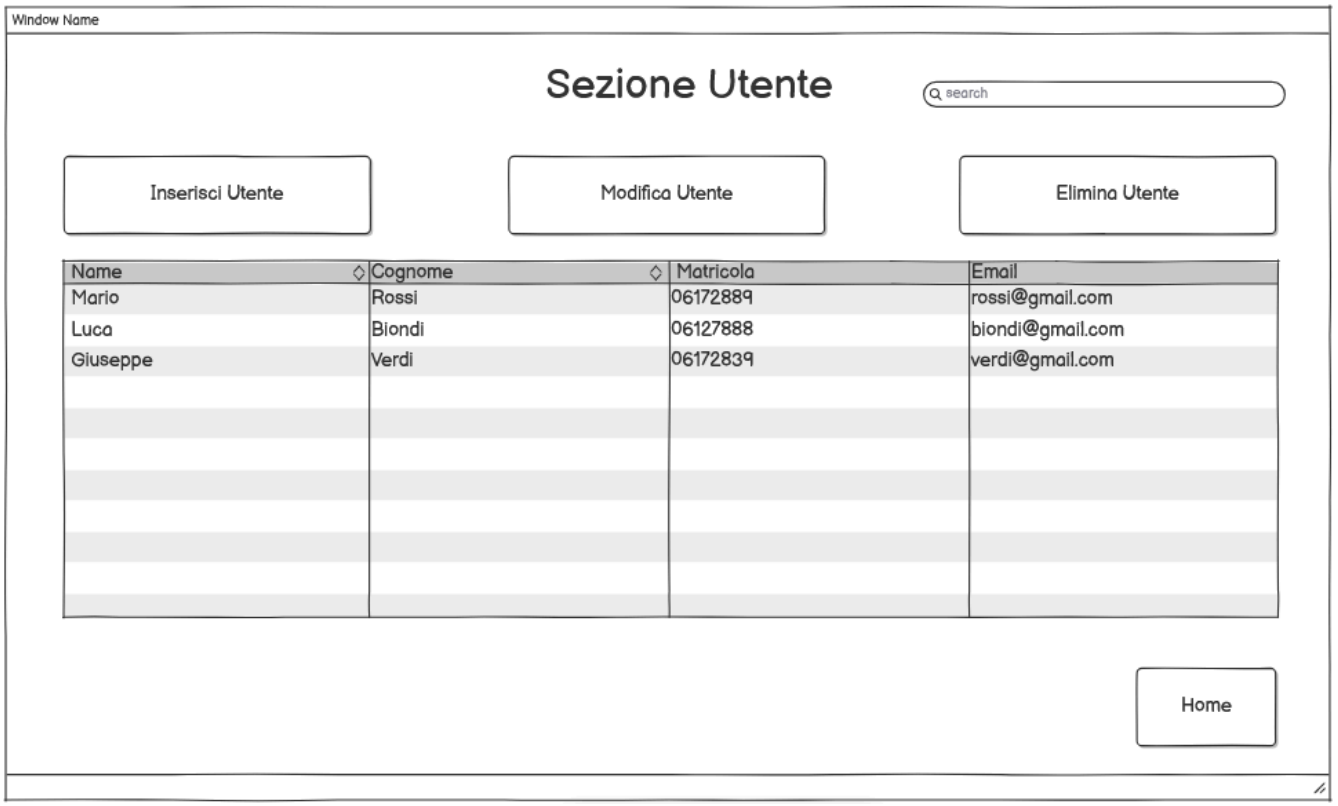
8.4 Wireframe

Di seguito vengono riportate dei Wireframe di come dovrebbe sembrare l'interfaccia utente, è possibile che durante la progettazione subiscano dei cambiamenti.

Homepage



Sezione Utente



Sezione Libri

Sezione Prestiti

9. Gestione della persistenza e I/O su file

9.1 Strategia di salvataggio

- All'avvio: carica file in memoria (oggetti in **`ConcurrentHashMap` o `HashMap`**).
- Ogni modifica (create/update/delete): repository aggiorna struttura in memoria e chiama funzione per persistere su file.
- Per efficienza: si può utilizzare un buffer di persistenza differita (commit periodico).

9.2 Backup e versioning

- Prima di ogni salvataggio completo, creare copia di backup del file su cui si sta operando, in caso in cui l'operazione non vada a buon fine ed il file si corrompa nel processo.

10. Strategie di test

10.1 Tipi di test

- **Unit tests**: JUnit 5 per servizi, repository, util.
- **Integration tests**: test su repository con file temporanei (directory `target/test-data`).
- **UI tests (opzionali)**: TestFX per interazione GUI.

10.2 Casi di test essenziali

- Inserimento libro valido / duplicato codice
- Modifica libro con cambiamento ISBN in conflitto
- Cancellazione libro in prestito (dev'essere gestito: impedire cancellazione o richiedere conferma)
- Registrazione prestito: utente con 2 prestiti (OK), con 3 prestiti (rifiuta)
- Registrazione prestito: zero copie disponibili
- Restituzione con ritardo -> calcolo sanzione