DOCUMENTO DI PROGETTAZIONE
GRUPPO 14:
Elisa Lombardi, Mara Mariano, Bruno Oliva, Francesco Pepe

1.ARCHITETTURA DEL SISTEMA

Il sistema Biblioteca Universitaria è progettato per supportare una gestione completa, da parte del bibliotecario, di tutti i servizi disponibili, attraverso un'interfaccia semplice ed intuitiva e un'architettura software modulare e manutenibile. L'applicazione è sviluppata in JavaFX e adotta il pattern MVC(Model-View-Control) in modo da rendere chiara e precisa la separazione delle responsabilità e delle operazioni tra interfaccia grafica, logica di business e gestione dei dati.

L'architettura si divide in diversi moduli principali, ciascuno con compiti specifici:

- Autenticazione: gestisce il login e logout del bibliotecario
- Gestione Libreria: permette la consultazione e gestione dei libri
- Gestione Prestiti: si occupa della coordinazione delle operazione di salvataggio e aggiornamento dei prestiti e di tutte le informazioni correlate
- Gestione Utente: si occupa del salvataggio e aggiornamento dei dati degli utenti
- UI (JavaFX): cura l'interfaccia grafica attraverso layout e personalizzazione

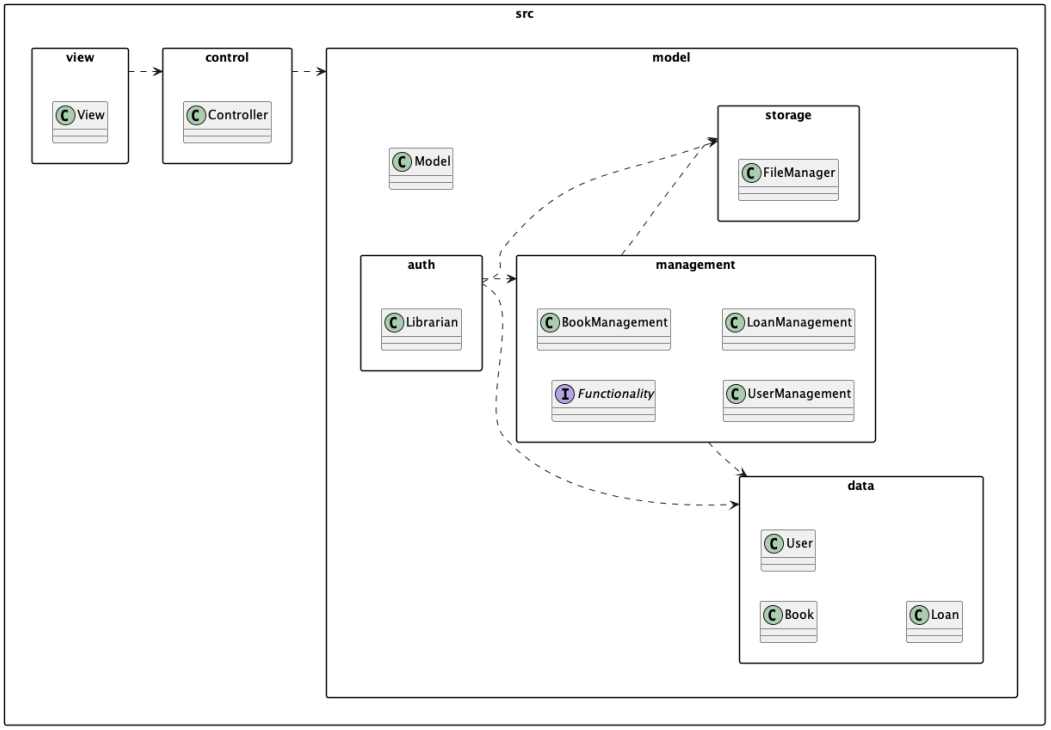
Ogni modulo interagisce con l'altro attraverso dipendenze specifiche, ad esempio: il modulo Gestione Prestiti collabora con i moduli Gestione Utente e Libreria per tenere traccia dei prestiti di ogni utente, di quali libri coinvolge e della loro disponibilità, mentre UI si appoggia a tutti i moduli citati per fornire i dati corretti in seguito alle interazioni dell'utente con l'interfaccia.

Il pattern MVC è stato adottato come scelta architetturale, per cui si definiscono tre package principali con precisi scopi funzionali:

- Model: ulteriormente suddiviso in data, auth,management e storage, definisce le principali funzioni e i dati attraverso la quale il bibliotecario può gestire il sistema
- Controller: si occupa del flusso di dati tra Model e View consentendo, in base alle interazioni dell'utente con la View, la corretta corrispondenza con i dati
- View: rappresenta il livello di presentazione del sistema e fornisce al bibliotecario un'interfaccia grafica facile ed intuitiva

La modularità rende il sistema facilmente manutenibile ed eventualmente estendibile ad ulteriori aggiornamenti.

1.1 DIAGRAMMA DEI PACKAGE



Il diagramma dei package evidenzia l'adozione del pattern MVC, con la suddivisione del sistema nei tre moduli principali.
View:

- Coesione: alta, responsabilità univoca su presentazione e interazione grafica con l'utente
- Accoppiamento: basso, ha interazioni solo con Controller

Controller:

- Coesione: alta, focalizzato sul flusso dati tra View e Model
- Accoppiamento: medio, funge da mediatore tra View e Model

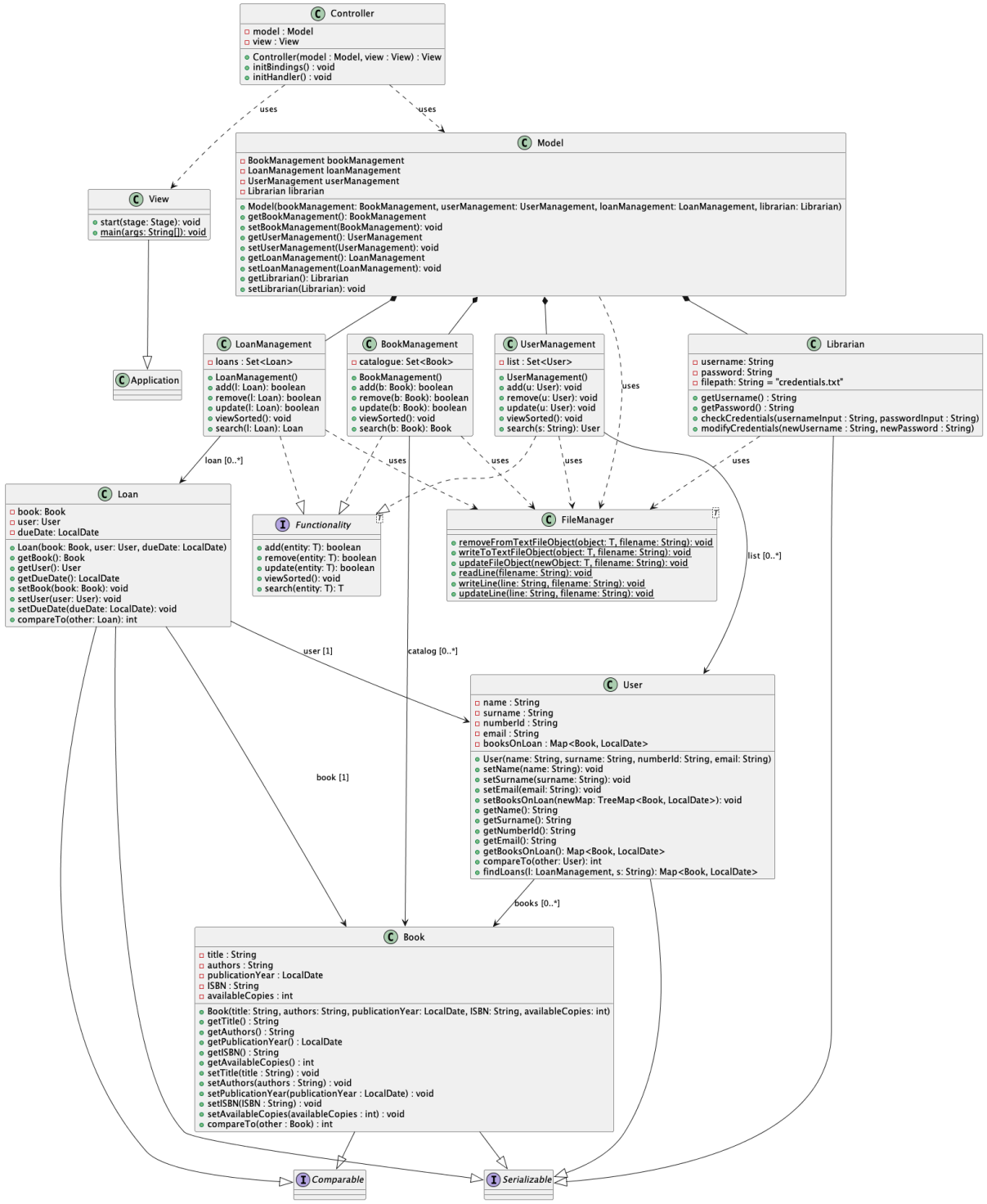
Model:

- Coesione: alta, distribuita in sottopackage con ruoli specifici:
  - data: definisce le entità fondamentali
  - management: gestisce le regole operative tra gli oggetti e l'interfaccia Functionality
  - auth: gestisce l'autenticazione
  - storage: gestisce il salvataggio e l'archiviazione dei dati
- Accoppiamento: relativamente basso, con un'adeguata separazione dei ruoli

I collegamenti evidenziano i legami tra i diversi package contenuti in model e dei tre principali package del pattern MVC.

2.MODELLO STATICO

2.1 DIAGRAMMA DELLE CLASSI



2.1.1 COMMENTO DIAGRAMMI DELLE CLASSI IN TERMINI DI COESIONE, ACCOPPIAMENTO E PRINCIPI DI BUONA PROGETTAZIONE

Nel nostro progetto abbiamo adottato una divisione delle classi mirata a garantire un alto livello di coesione e un debole accoppiamento, così da rispettare il più possibile i principi di buona progettazione del software. Per raggiungere questo obiettivo abbiamo sfruttato il pattern MVC che, da definizione, separa un'applicazione in 3 componenti principali per rendere il codice più organizzato, riutilizzabile e facilmente manutenibile.

Nel package model abbiamo inserito i dati e la logica di business, infatti sono presenti le classi User, Book e Loan (raggruppate nel sottopackage model.data) che rappresentano gli oggetti fondamentali per lo stato dell'applicazione e non sanno nulla sulla View o Controller del sistema. Sono classi "basilari", puramente modellistiche che incapsulano semplicemente i dati degli oggetti rispettivi.
A favore di una separazione maggiore delle responsabilità abbiamo introdotto: il sottopackage model.auth in cui abbiamo definito la classe Librarian, la quale rispetta i vincoli del Model, dedicandola semplicemente a controlli sulle credenziali del bibliotecario; e il sottopackage model.management dove invece ci siamo concentrati sulle classi che (come ci fa intuire il nome) si occupano della gestione (in termini di liste ,aggiunte,modifiche ecc) delle classi basilari.
All'interno di model.management troviamo: BookManagement, UserManagement, LoanManagement e l'interfaccia Functionality, creata per motivi legati a organizzazione e flessibilità, per ottenere polimorfismo (caposaldo del linguaggio Java) e disaccoppiare il codice. Di fatti rappresenta un contratto comune per le operazioni di gestione, tutte le classi di questo package devono essenzialmente svolgere gli stessi compiti ma ovviamente in modi diversi, la scelta di un'interfaccia comune (che permette quindi di semplicemente definire le firme dei metodi e poi implementarli coerentemente nelle opportune classi) ci sembra la scelta più efficiente.
Infine, il sottopackage model.storage contiene la classe FileManager, responsabile del salvataggio del nostro archivio su un file. Per facilità, infatti, FileManager e le classi dei dati implementano l'interfaccia Serializable.

Il package control ospita una classe Controller, che ha il ruolo di collegare il Model con la View. La classe è stata progettata con due metodi principali: uno dedicato all'impostazione dei binding e l'altro alla gestione degli eventi generati dall'interazione dell'utente con l'interfaccia. In questo modo i compiti risultano più chiari e ben separati, anche visivamente.

Il package view contiene la classe View, che definisce tutto l'aspetto visivo. Essa contiene esclusivamente gli elementi necessari alla realizzazione dell'interfaccia utente.

Abbiamo cercato di fare in modo che ogni componente del nostro progetto avesse un compito specifico, ben definito e indipendente da altri, limitando al minimo le interferenze esterne e utilizzando solo le informazioni strettamente necessarie. Questo approccio si fonda sui principi di Single Responsability e Information Hiding, che favoriscono la manutenibilità, l'estensibilità e la robustezza complessiva del sistema.

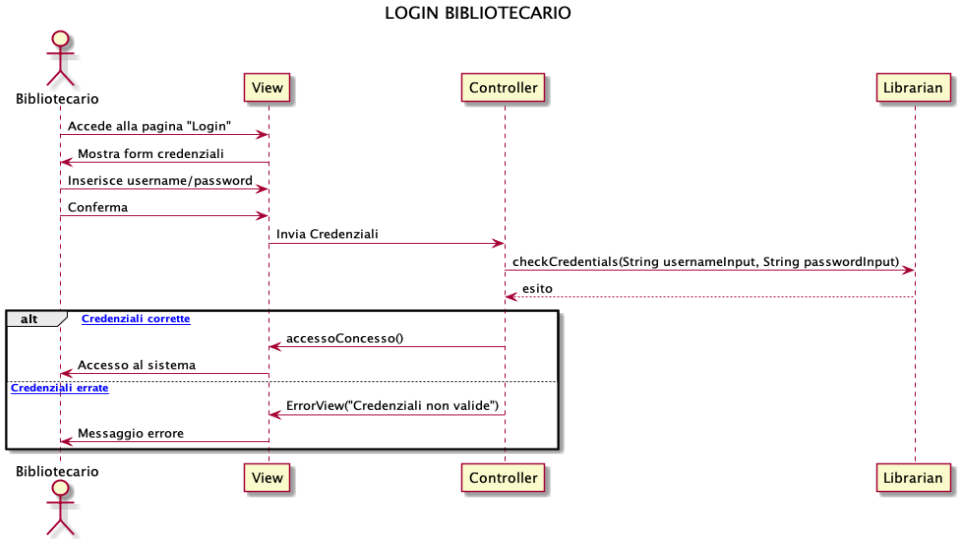
Il basso accoppiamento è giustificato dall'assenza di dipendenze circolare tra le classi: infatti, ognuna rimane indipendente e la comunicazione avviene tramite interfacce chiare e ben definite. Dunque, ciascun modulo conosce solo i dati indispensabili al proprio funzionamento.

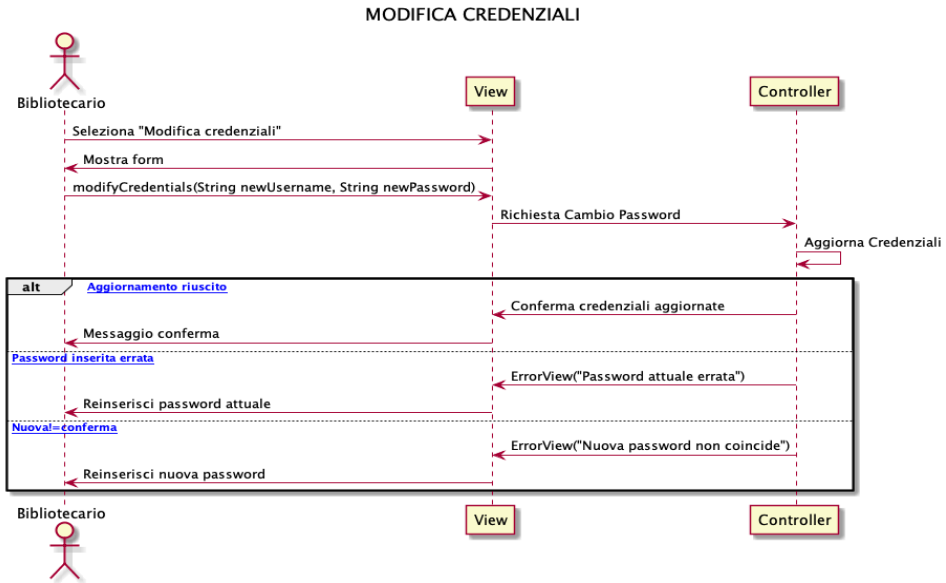
3. MODELLO DINAMICO

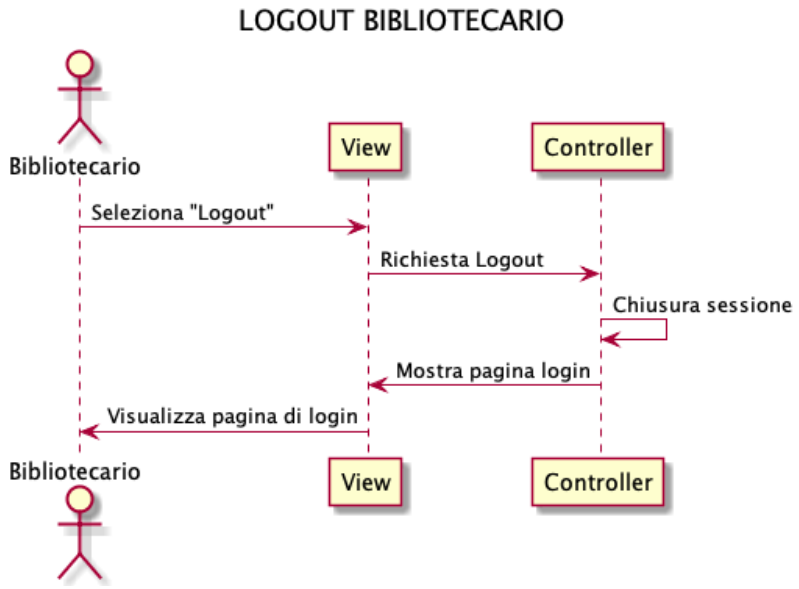
I diagrammi di sequenza UML sono strumenti utilizzati per rappresentare il comportamento dinamico di un sistema software. Essi mettono in evidenza lo scambio di messaggi tra oggetti o classi, sottolineando come i diversi componenti collaborano tra loro.
I diagrammi di sequenza hanno l'obiettivo di mostrare come le classi si coordinano per eseguire le funzionalità del sistema.

3.1 DIAGRAMMI DI SEQUENZA

3.1.2 DIAGRAMMI DI SEQUENZA BIBLIOTECARIO







3.1.2.1 COMMENTO IN TERMINI DI COESIONE, ACCOPPIAMENTO E PRINCIPI DI BUONA PROGETTAZIONE SUI DIAGRAMMI DI SEQUENZA BIBLIOTECARIO

Coesione

Il diagramma evidenzia un buon livello di coesione funzionale, grazie alla chiara suddivisione dei compiti tra i componenti coinvolti.

- View si occupa esclusivamente dell'interazione con l'utente, mostrando il form di login e i messaggi di conferma o errore.
- Controller gestisce il flusso tra interfaccia e logica applicativa, ricevendo le credenziali dalla View e inoltrandole al modulo di verifica.
- Librarian incapsula la logica di validazione delle credenziali, mantenendo la responsabilità della verifica.
- Bibliotecario rappresenta l'attore esterno che interagisce con il sistema.

Questa distribuzione evita sovrapposizioni funzionali e favorisce una progettazione modulare, in cui ogni componente è focalizzato su un singolo compito, garantendo un alto livello di coesione.

Accoppiamento

Il diagramma mostra un accoppiamento debole, prevalentemente riconducibile all'accoppiamento per dati:

- Le interazioni tra View e Controller avvengono tramite il passaggio di credenziali, senza dipendenze strutturali.
- Il Controller comunica con Librarian passando parametri specifici (username, password), mantenendo un buon grado di indipendenza.

Tuttavia, si osserva un accoppiamento per timbro tra Controller e Librarian, poiché vengono passate strutture dati necessarie per la verifica.
Infine, è presente un accoppiamento per controllo: il valore di ritorno da Librarian influenza il flusso del Controller, determinando se mostrare una conferma o un messaggio di errore.

Principi di Buona Progettazione

1. SINGLE RESPONSIBILITY PRINCIPLE

Ogni componente svolge un compito specifico:

- View: interazione con l'utente.
- Controller: coordinamento tra interfaccia e logica.
- Librarian: verifica delle credenziali.

Questa separazione rispetta il principio di responsabilità singola, migliorando la manutenibilità.

2. OPEN-CLOSED PRINCIPLE

Le classi sono progettate per essere aperte all'estensione e chiuse alla modifica:

- Le credenziali vengono gestite tramite metodi dedicati, senza esporre direttamente gli attributi.
- È possibile estendere la logica di verifica senza alterare il comportamento esistente.

3. LISKOV SUBSTITUTION PRINCIPLE

La logica di verifica può essere specializzata in sottoclassi di Librarian senza compromettere il funzionamento del Controller, garantendo la sostituibilità.

4. INTERFACE SEGREGATION PRINCIPLE

Ogni modulo espone solo le operazioni necessarie:

Controller non gestisce direttamente i dati, ma si limita a coordinare.

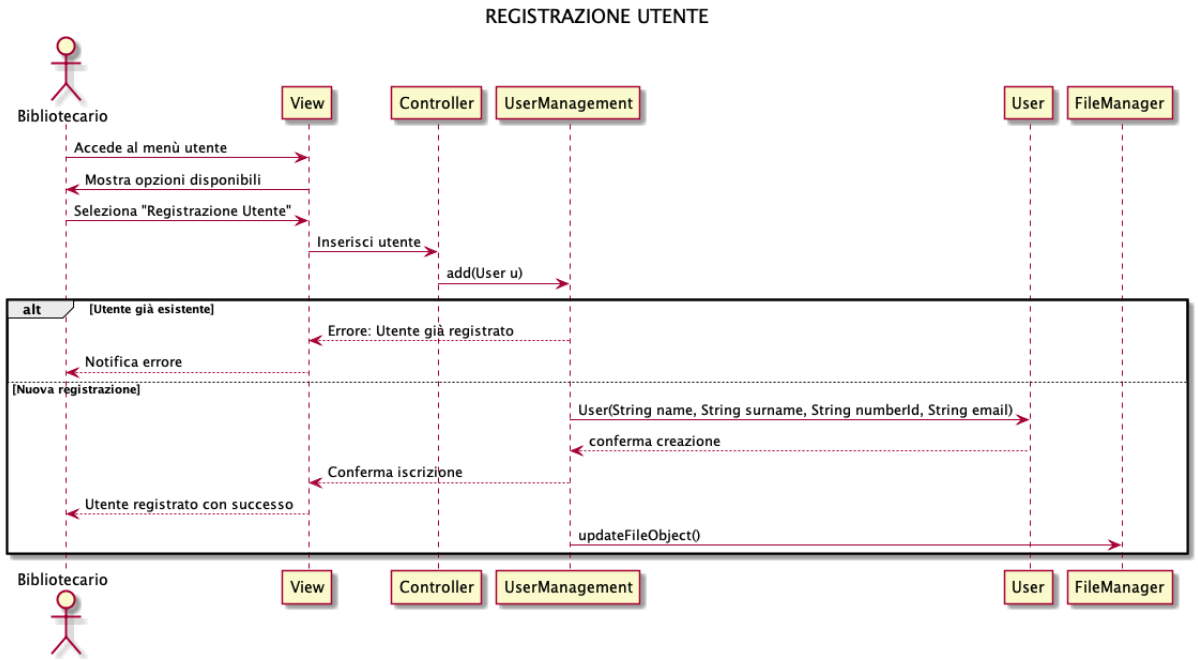
Librarian si occupa esclusivamente della logica di verifica.
Questo evita interfacce generiche e mantiene i componenti specifici e indipendenti.

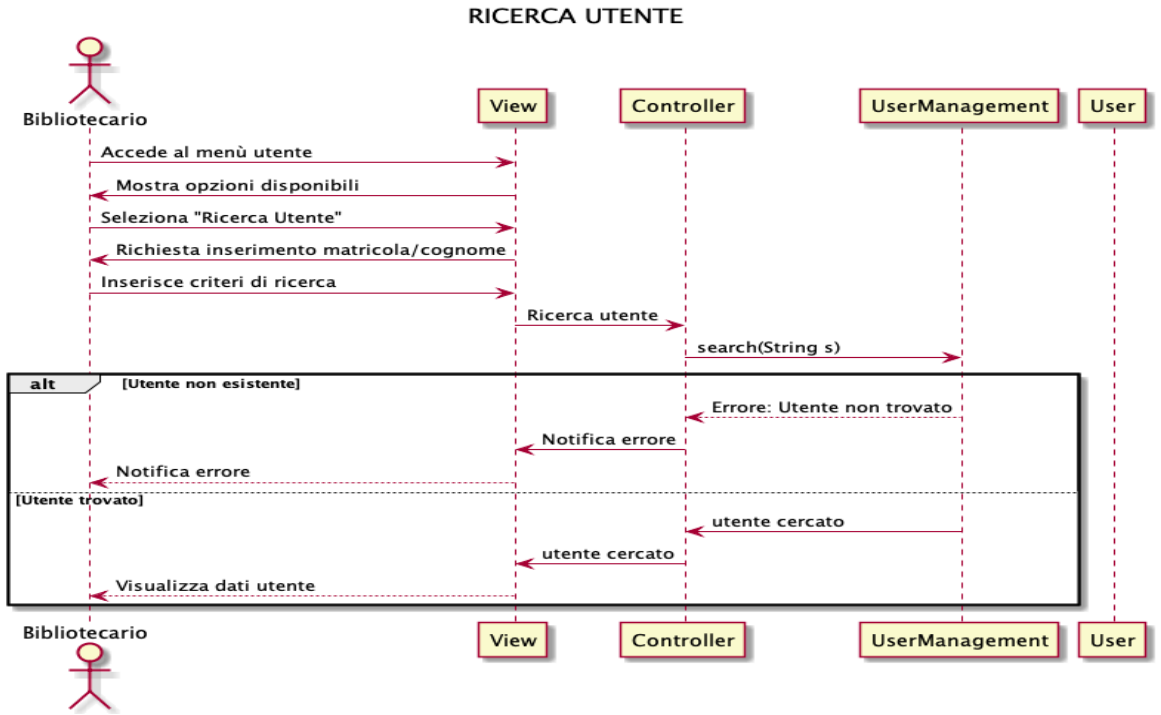
5. DEPENDENCY INVERSION PRINCIPLE

Il principio è parzialmente rispettato in quanto il Controller dipende direttamente dalla classe Librarian, senza un'interfaccia astratta.

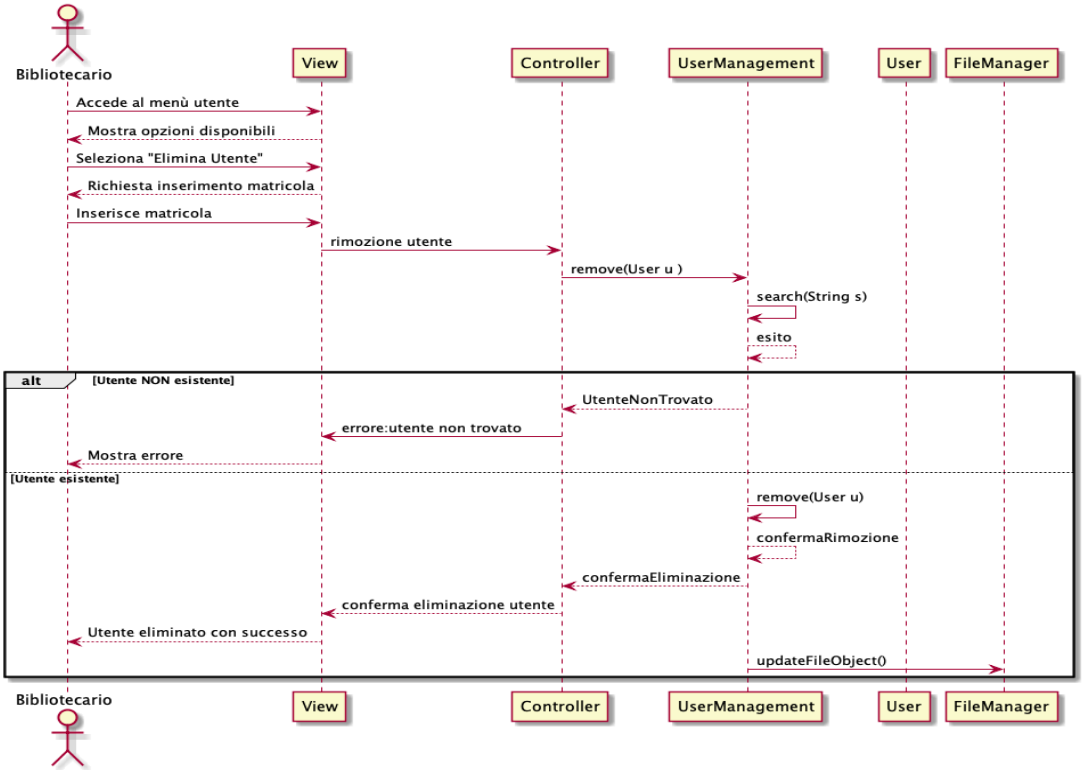
Per migliorare, si potrebbe introdurre un'interfaccia CredentialChecker, da cui Librarian deriva, riducendo la dipendenza da implementazioni concrete.

3.1.3 DIAGRAMMA DI SEQUENZA UTENTE

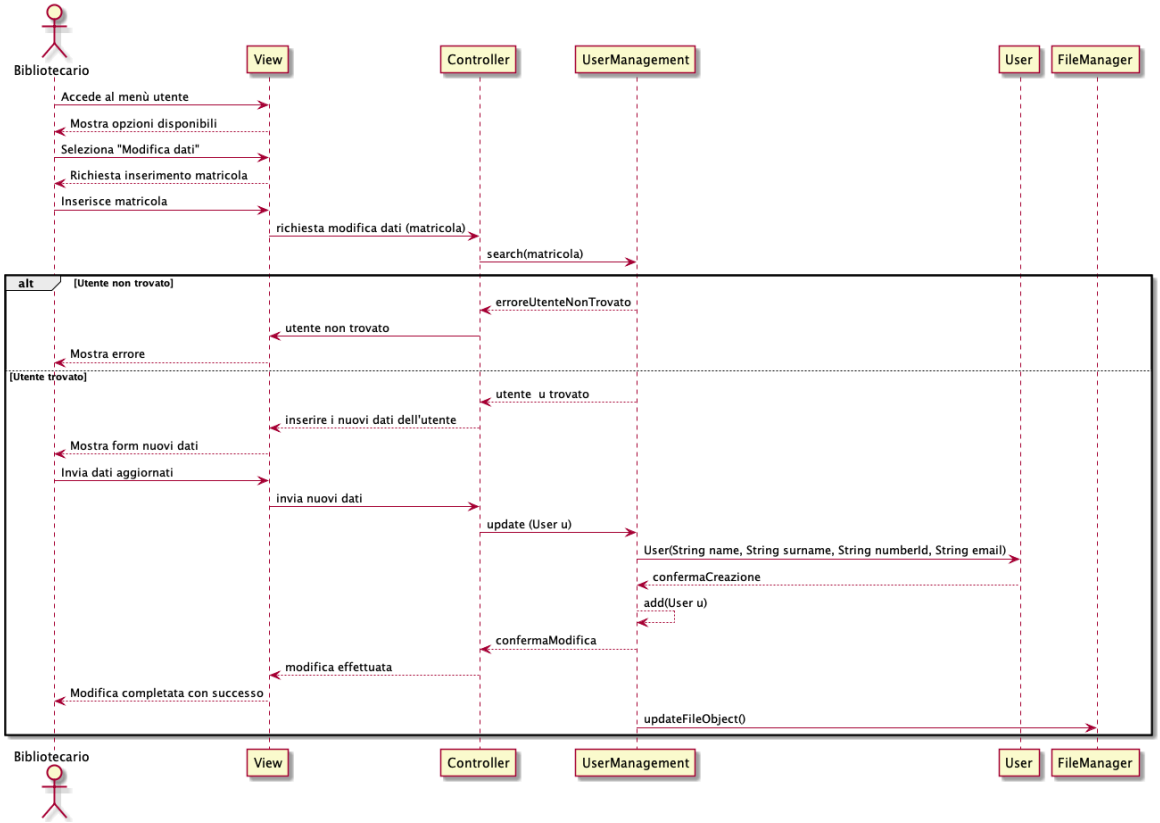


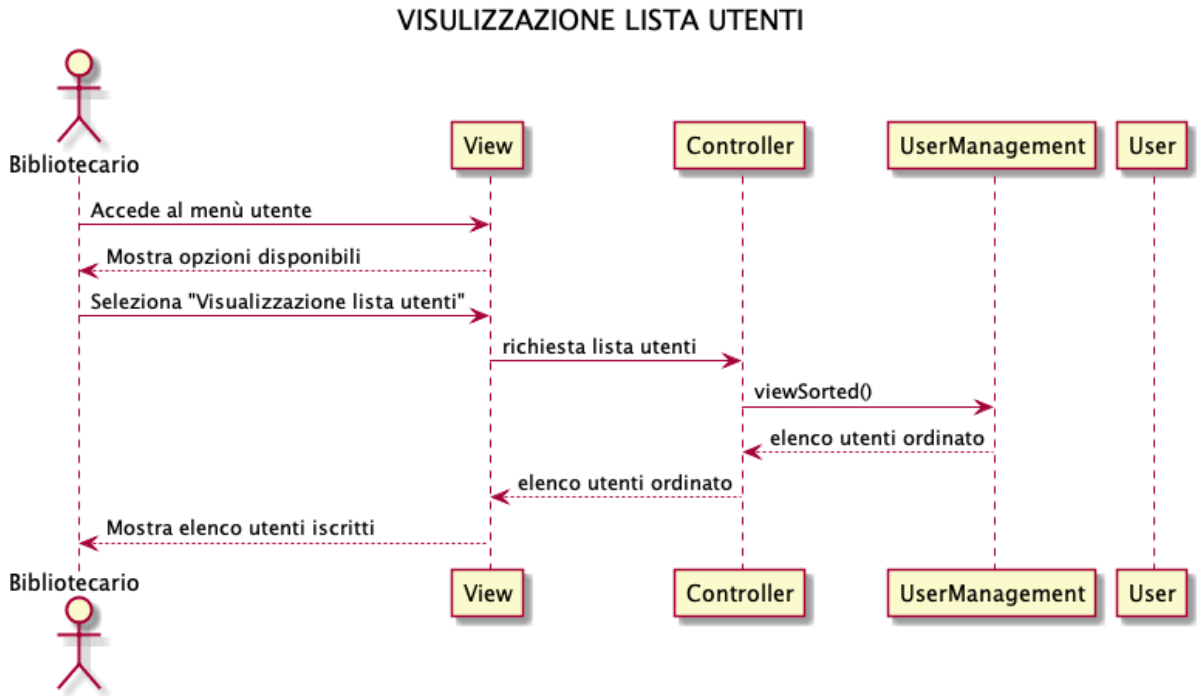


ELIMINAZIONE UTENTE



MODIFICA DATI UTENTE





3.1.3.1 COMMENTO IN TERMINI DI COESIONE, ACCOPPIAMENTO E PRINCIPI DI BUONA PROGETTAZIONE SUI DIAGRAMMI DI SEQUENZA UTENTE

COESIONE

UserManagement gestisce tutta la logica di ricerca, aggiornamento e aggiunta dell'elenco utenti mantenendo coerenza.

View essendo una user interface si occupa semplicemente della visualizzazione ,quindi gestisce dati in input e di output.

User non interviene sempre(è stato comunque inserito nei diagrammi per chiarezza) ed è pensato solo come semplice entità dati.

Controller coordina le richieste dalla View al model,quindi è solo un mediatore

FileManager si occupa esclusivamente del salvataggio su un file in seguito a modifiche dell'archivio.

In conclusione ogni elemento svolge un compito specifico,senza contaminazione e dipendenze inutili tra classi.

ACCOPPIAMENTO

L'interfaccia comunica solo con Bibliotecario e Controller e mai direttamente con User o UserManagement ,dato che funge da strato di presentazione è necessario l'accoppiamento verso il controller.

FileManager è chiamato solo da UserManagement e non conosce nulla sul resto del sistema quindi ha un accoppiamento molto basso.

Controller comunica coerentemente con View e con le classi del Model stando al pattern MVC

UserManagement è il perno centrale del nostro sistema quindi va da sè che abbia contatti con più componenti,tuttavia l'accoppiamento risulta moderato

User ha accoppiamento bassissimo,come richiesto da una classe modello

l'accoppiamento è basso/moderato, perfettamente in linea con MVC, perché le classi interagiscono solo attraverso interfacce chiare.

PRINCIPI DI BUONA PROGETTAZIONE

1. Open/Closed Principle (OCP)

- UserManagement potrebbe essere esteso (es. nuovi metodi)
- FileManager potrebbe cambiare implementazione senza modificare UserManagement

2. SRP (Single Responsibility Principle)

ogni classe ha compiti chiari,singoli,così da avere una buona separazione tra i ruoli ed evitare classi troppo grandi o difficili

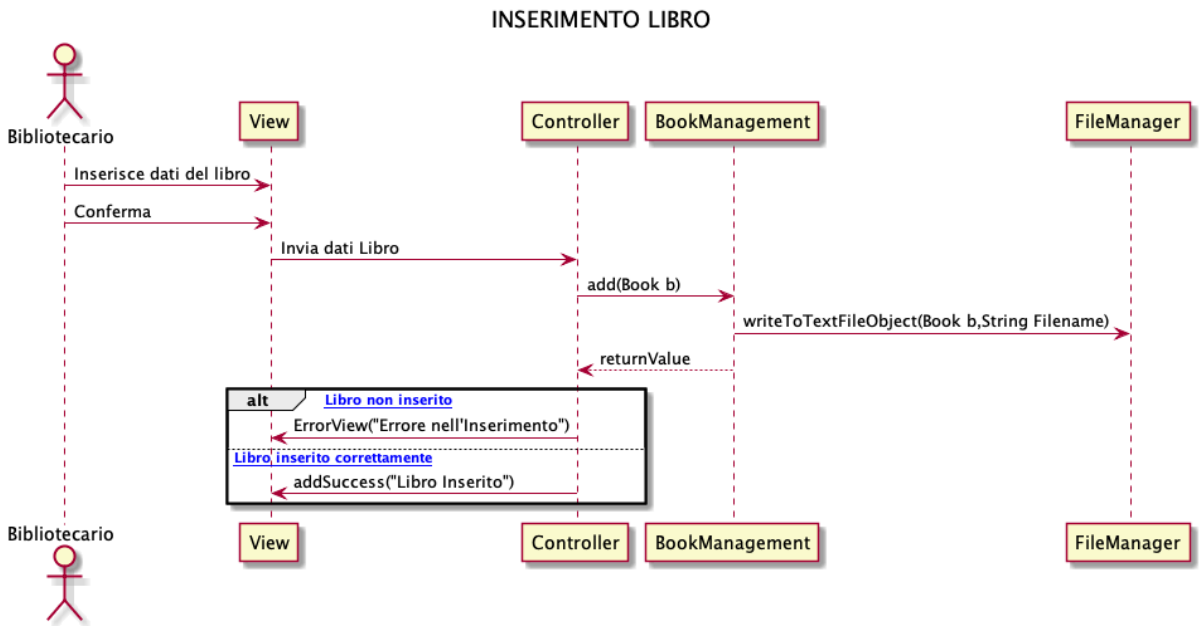
3. INCAPSULAMENTO

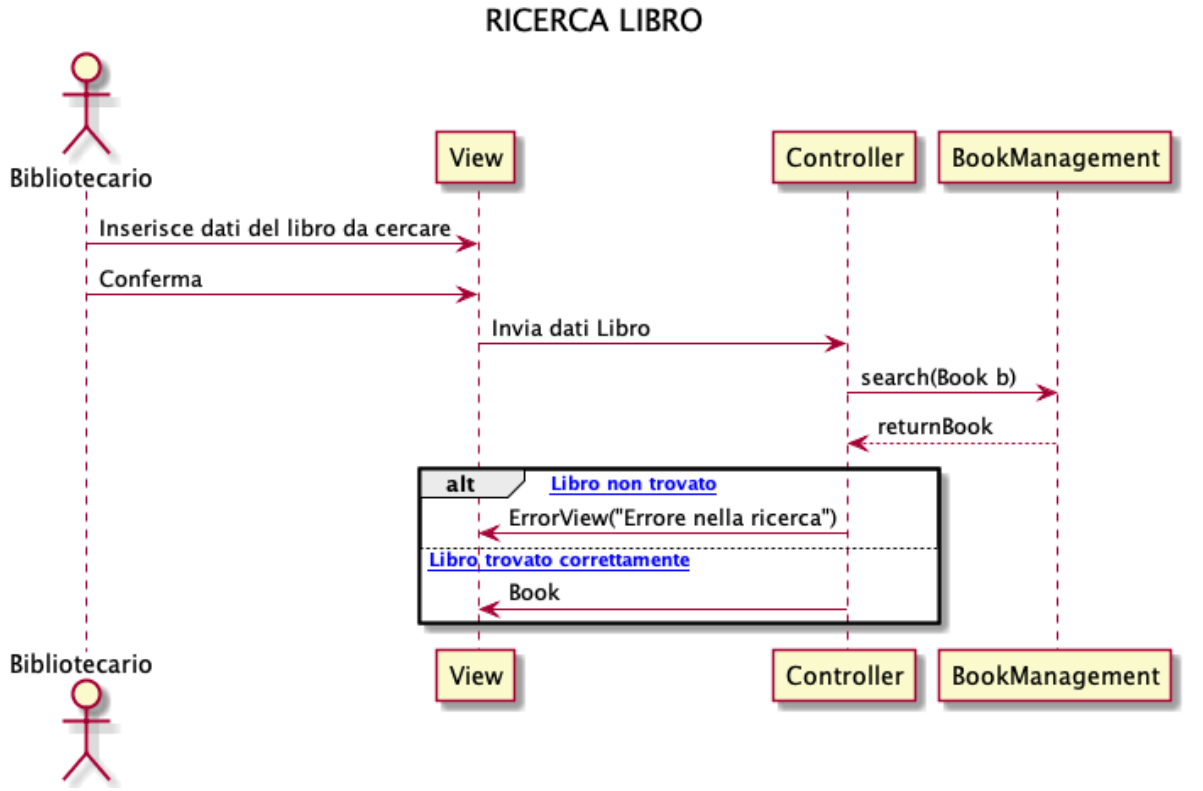
la creazione di User avviene all'interno della logica di gestione, non nell'interfaccia.

4. BASSO ACCOPPIAMENTO-ALTA COESIONE : rispettati.

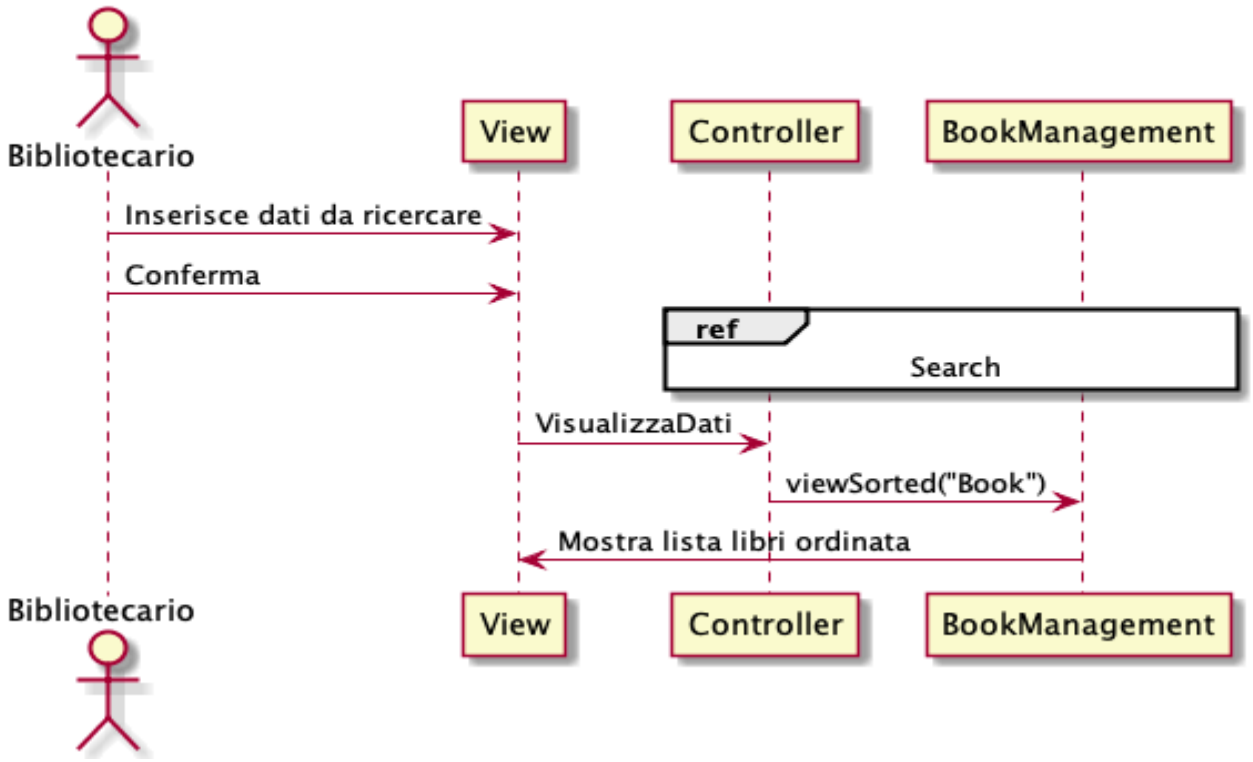
INTERFACE SEGREGATION PRINCIPLE
Riprendendo le descrizioni dei compiti delle classi esplicate nella sottosezione COESIONE ,notiamo che nessuna classe è costretta a implementare metodi inutili,nessuna classe è obbligata a conoscere operazioni che non le competono e ogni elemento usa solo ciò che gli serve davvero rispecchiando a pieno lo spirito dell'ISP

3.1.4 DIAGRAMMA DI SEQUENZA LIBRO

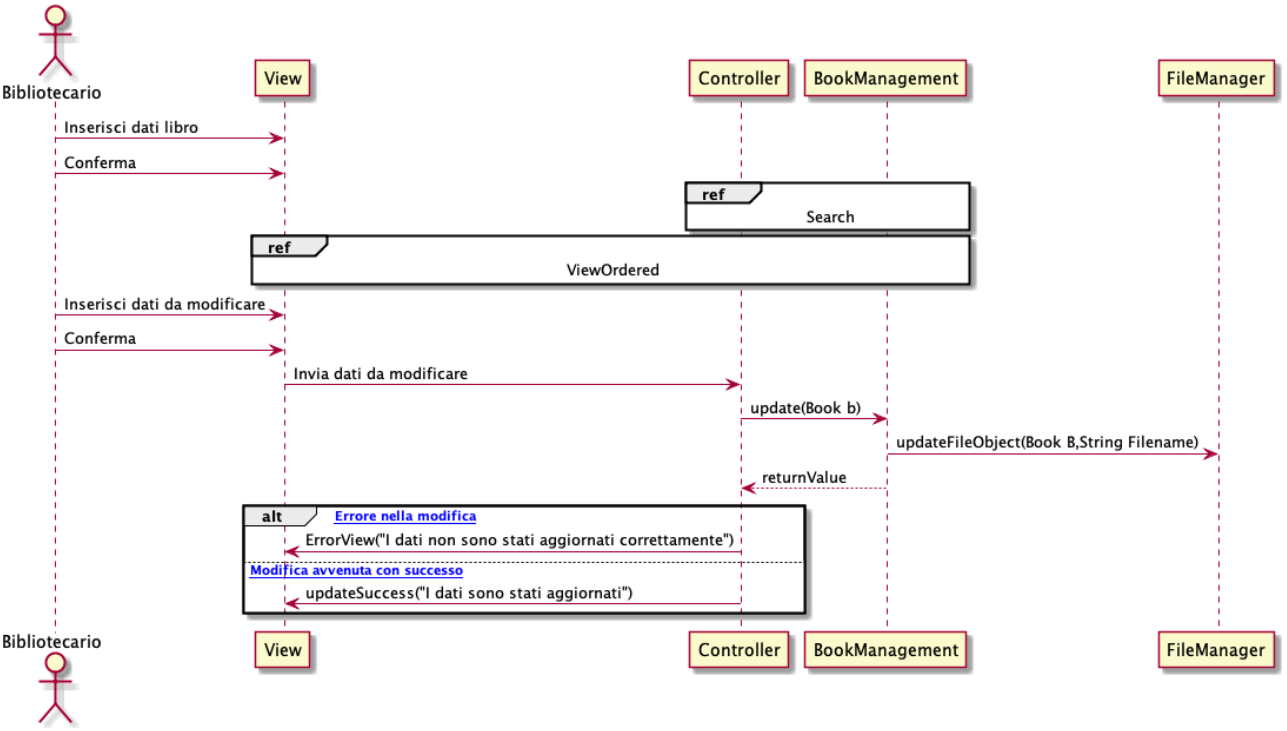


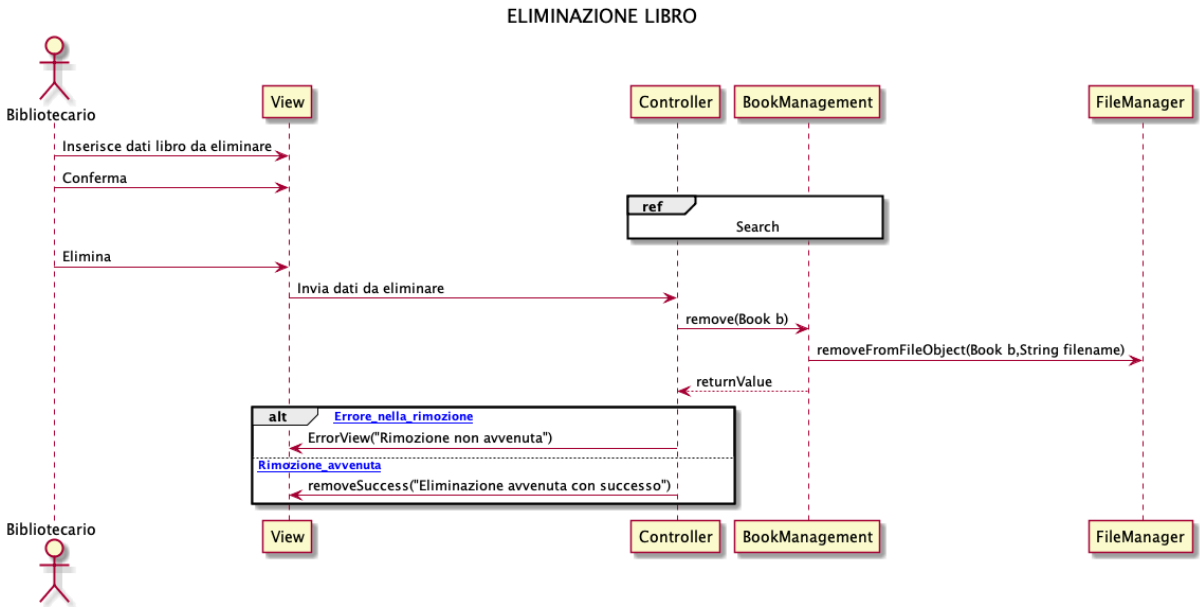


VISUALIZZAZIONE LISTA LIBRI



MODIFICA DATI LIBRO





3.1.4.1 COMMENTO IN TERMINI DI COESIONE, ACCOPPIAMENTO E PRINCIPI DI BUONA PROGETTAZIONE SUI DIAGRAMMI DI SEQUENZA UTENTE

Coesione

Il diagramma evidenzia un buon livello di coesione, riconducibile alla Coesione Funzionale:

- BookManagement si occupa unicamente della gestione dei dati dei libri accentrando tutte le operazioni che riguardano eventi sui dati
- FileMenager si occupa delle operazione su un file di testo esterno
- View, gestisce l'interazione con l'utente mostrando eventuali messaggi di conferma o di errore
- Controller gestisce l'interazione tra la View e la BookManagement, coordinando il flusso tra di essi

Visti i ruoli ben definiti sopra si evitano sovrapposti funzionali, garantendo un alto livello di coesione.

Accoppiamento

Il diagramma mostra in generale un basso accoppiamento, riconducibile principalmente all'Accoppiamento per dati. I diversi componenti interagiscono scambiandosi informazioni strettamente necessarie al fine di concludere il compito assegnatogli garantendo così un buon grado di indipendenza, questo avviene soprattutto tra View>->Controller Ad eccezione del BookManagement<->FileManger e Controller<->BookManagement, il quale livello di accoppiamento risulta più forte, riconducibile all'accoppiamento per timbro, in quanto vengono passate strutture dati necessarie per eseguire le singole operazioni. Infine è presente un tipo di accoppiamento riconducibile all'accoppiamento per controllo, a partire dal BookManagement, per quanto riguarda il valore di ritorno che influenza il flusso in base al suo valore.

Principi di Buona Progettazione

1. SINGLE RESPONSIBILITY PRINCIPLE

Si evidenzia come ogni componente svolga un compito preciso che riguarda un aspetto differente del sistema, in accordo con il principio di buona progettazione sopra citato.
in particolare:

- View : si occupa dell'interazione con l'utente
- Controller : coordina il flusso tra l'interfaccia e la logica interna
- BookManagement : gestisce la logica applicativa dei libri
- File Manager : si occupa della gestione delle informazioni sul file

2. OPEN-CLOSED PRINCIPLE

Le differenti classi seguono l'open-closed principle, in quanto gli attributi non sono direttamente modificabili e accessibili dall'utente impedendo l'alterazione improprie e rispettando la chiusura alla modifica, però le classi mettono a disposizione metodi che consentono di modificare i dati senza dover intervenire sul codice, questo permette di ampliare i metodi delle classi senza alterarne il codice e le funzionalità in accordo con l'apertura all'estensione.

3. LISKOV SUBSTITUTION PRINCIPLE

Grazie alla specializzazione delle singole funzioni relative alla gestioni dei dati, è possibile, in accordo al pricipio citato, sostituire la classe BookManagement con delle sottoclassi specializzate senza alterare il comportamento atteso.

4. INTERFACE SEGREGATION PRINCIPLE

Grazie al rispetto del single responsibility principle, ogni modulo mette a disposizione solo le operazioni necessarie a svolgere la funzione designata.
Classi come:

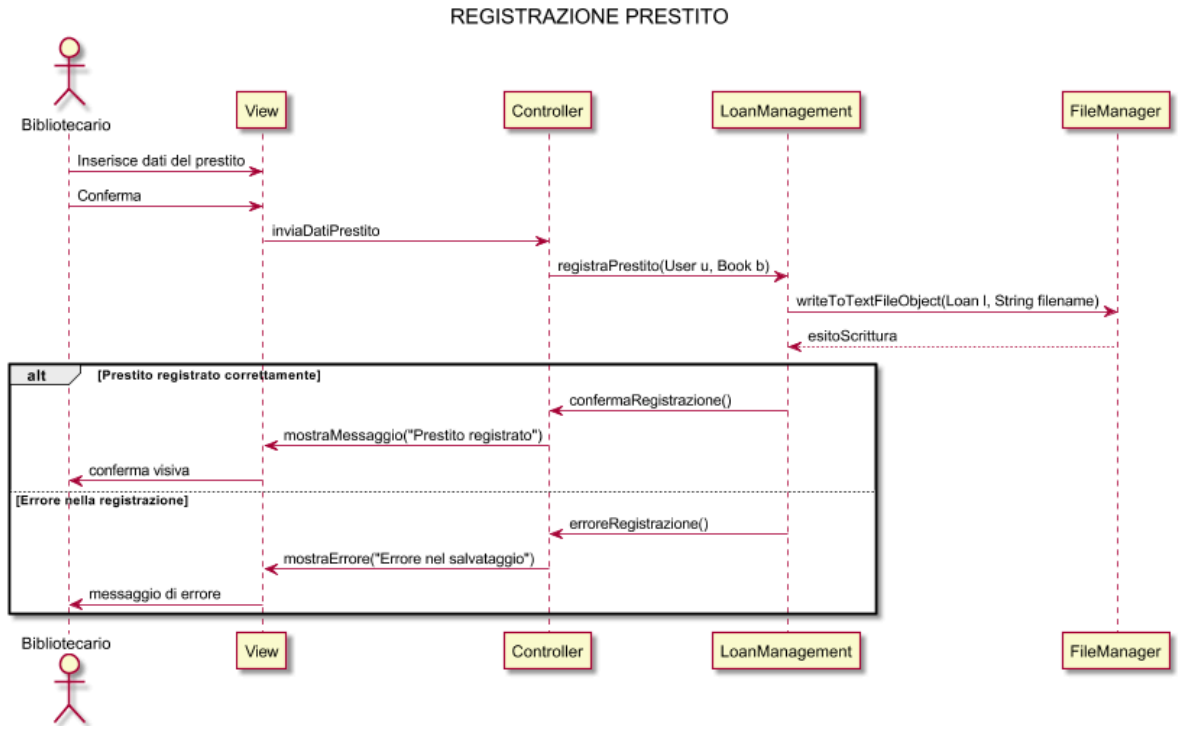
- Controller, si occupa unicamente del flusso dei dati senza metodi di gestione dati
- BookManagement che al contrario di controller, presenta metodi di gestione senza interferire con l'interfaccia

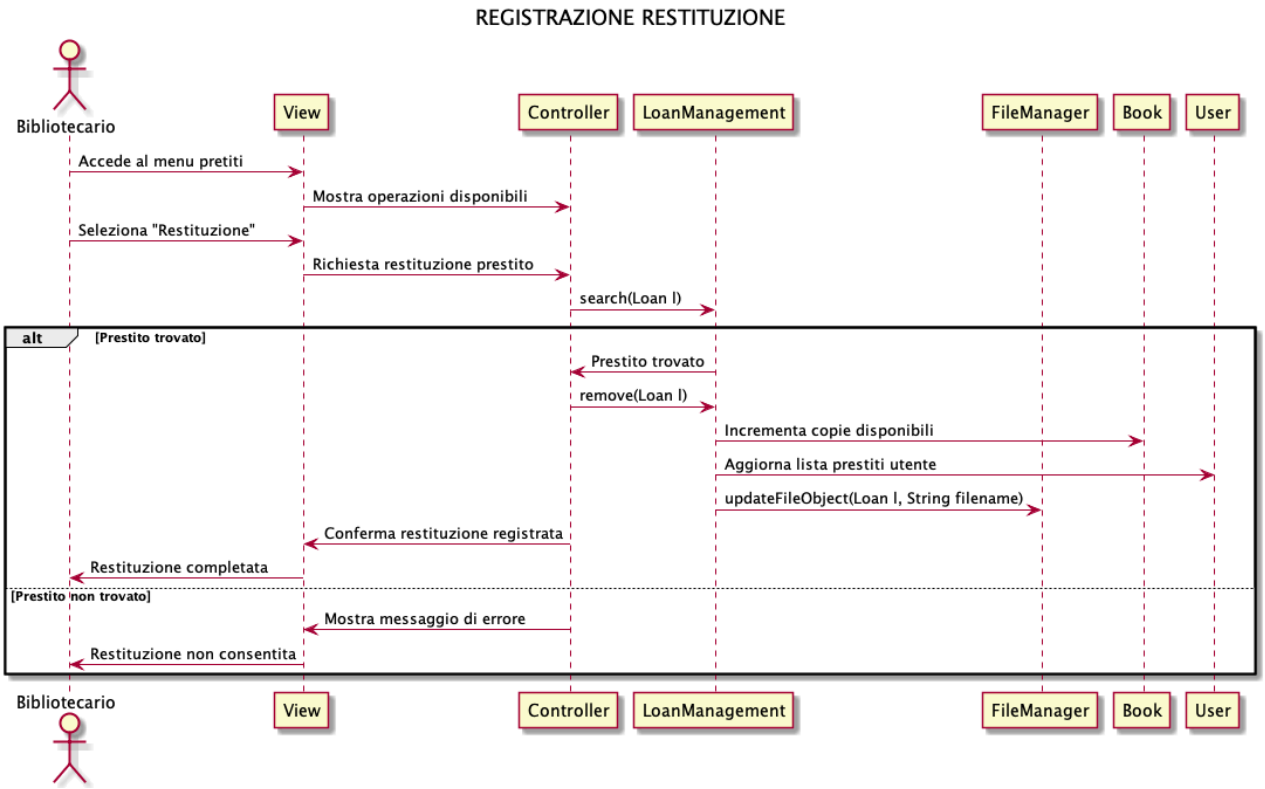
Questa evita interfacce generiche e mantiene i moduli indipendenti e specifici.

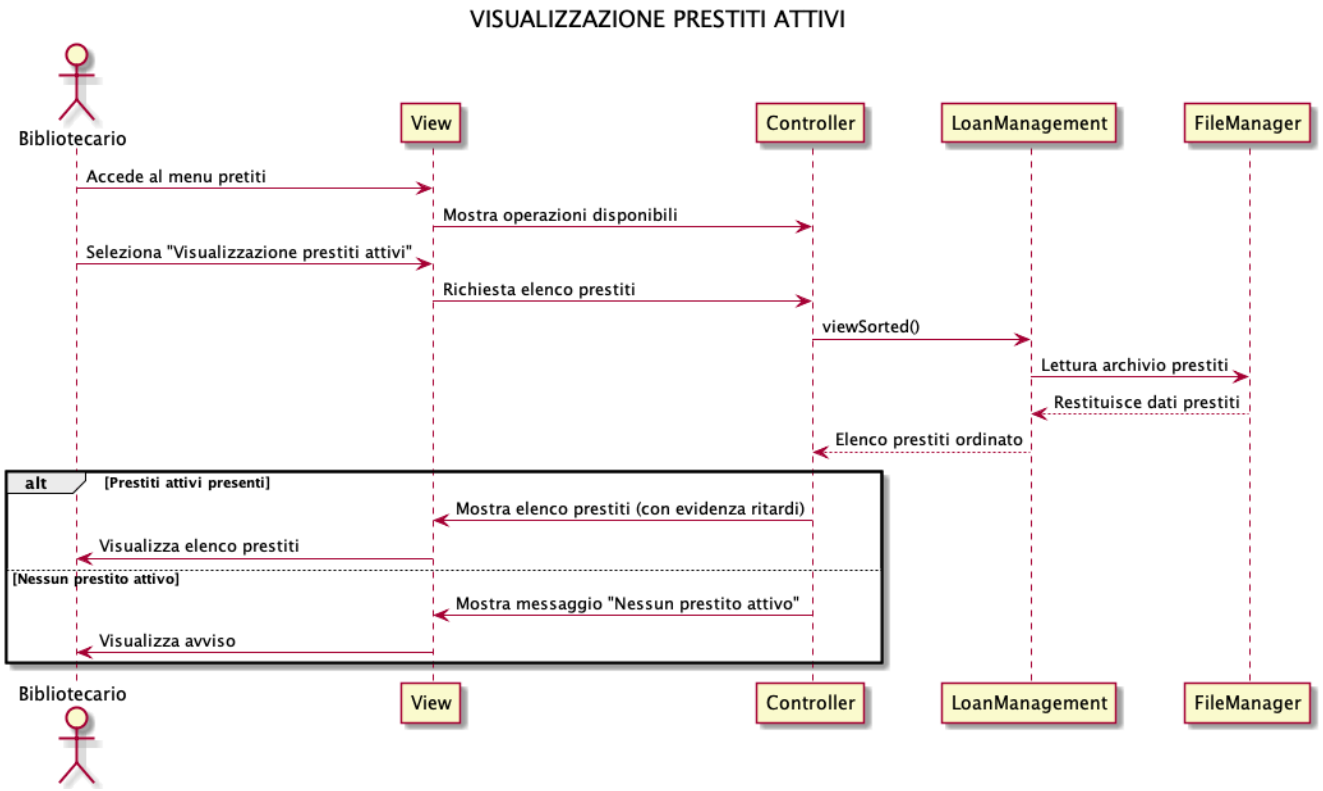
5. DEPENDENCY INVERSION PRINCIPLE

Il sistema mostra una parziale coerenza con il principio sopra citato in quanto la classe BookManagement dipende direttamente da File Menager per la scrittura dei dati sul file.

3.1.5 DIAGRAMMA DI SEQUENZA PRESTITI







3.1.5.1 COMMENTO IN TERMINI DI COESIONE, ACCOPPIAMENTO E PRINCIPI DI BUONA PROGETTAZIONE SUI DIAGRAMMI DI SEQUENZA PRESTITO

COESIONE
Il modello evidenzia un buon livello di coesione funzionale, grazie alla chiara distribuzione dei compiti tra i componenti:

- LoanManagement: gestisce la logica di ricerca, registrazione e restituzione dei prestiti
- View: si occupa della visualizzazione della raccolta dei dati in input dall'utente
- User e Book: sono le classi degli oggetti di user e book e servono per aggiornare i dati relativi alle operazioni coinvolte
- Controller: coordina le richieste dalla View e le inoltra al Model
- FileManager: gestisce l'aggiornamento dei file esterni per la registrazione delle informazioni del sistema

Questa suddivisione favorisce una progettazione modulare, con alta coesione, perché ciascun componente svolge un compito specifico e ben definito.

ACCOPPIAMENTO
Il sistema mostra un accoppiamento basso, riconducibile all'accoppiamento per dati: infatti, View comunica solo con Librarian e Controller, mai in modo diretto con LoanManagement e FileManger.
L'accoppiamento tra FileManager e LoanManagement risulta più alto, riconducibile all'accoppiamento per timbro poiché non vengono passate semplici informazioni ma oggetti utili all'aggiornamento dei file esterni; un discorso analogo si può fare tra Controller e LoadManagement.

PRINCIPI DI BUONA PROGETTAZIONE

1. Open/CLosed Principle (OCP**)**

LoanManagement può essere esteso (gestione ritardi, ecc) e FileManager può cambiare implementazione senza modifcare LoanManagement. Inoltre gli attributi non sono direttamente accessibili dall'utente ma possono essere modificati attraverso opportuni metodi.

2. Single Resposinbility Principle (SRP**)**

I compiti di ogni classe sono chiari e ben delineati, questo permette una maggiore modularità e di conseguenza una grande manutenibilità.

3. Incapsulamento

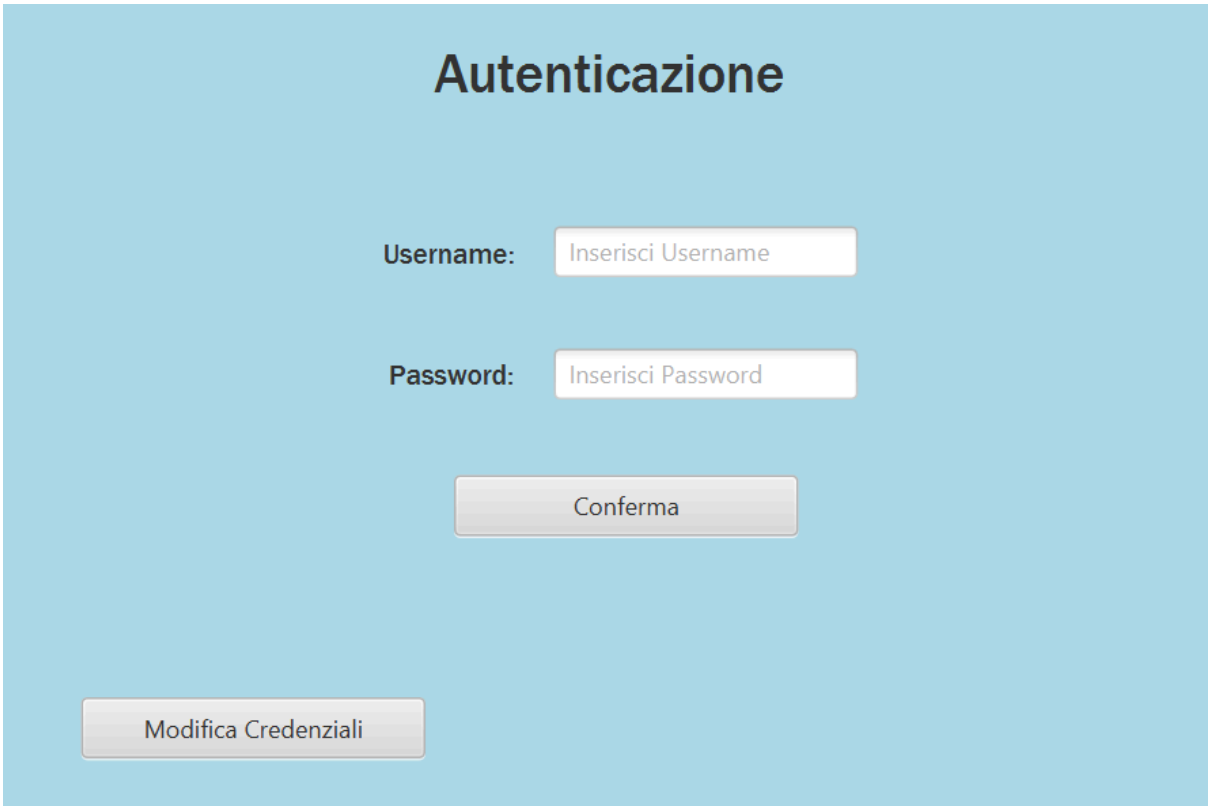
La creazione e gestione dei prestiti avviene all'interno di LoanManagement, evitando che la View interferisca con la logica applicativa. Questo permette di rispettare un basso accoppiamento e un altra coesione.

4. Interface Segretation Principle (ISP)

Nessuna classe implementa metodi superflui; ogni elemento si serve solo di ciò che gli è strettamente utile. Per esempio Controller presenta unicamente funzioni che gestiscono il flusso dei dati mentre LoanManagement presenta funzioni utili alle operazioni possibili al sistema.

4. DESIGN DELL'INTERFACCIA DELL'UTENTE

4.1 INTERFACCIA DI AUTENTICAZIONE



1. **Descrizione funzionale**

Questo primo mock-up rappresenta l'interfaccia iniziale all'avvio dell'applicazione. L'utente, il bibliotecario, può inserire le credenziali necessarie per accedere al sistema oppure modificarle.

2. **Elementi tecnici**

È presente un Pane principale che contiene i diversi elementi:

- Label:
  - Autenticazione
  - Username
  - Label nascosto, utilizzato per visualizzare eventuali messaggi di errore
- TextField per l'inserimento di username e password
- Button:
  - Conferma: confronta le informazioni presenti, se corrette, porta alla pagina principale
  - Modifica Credenziali: permette di modificare le credenziali di base

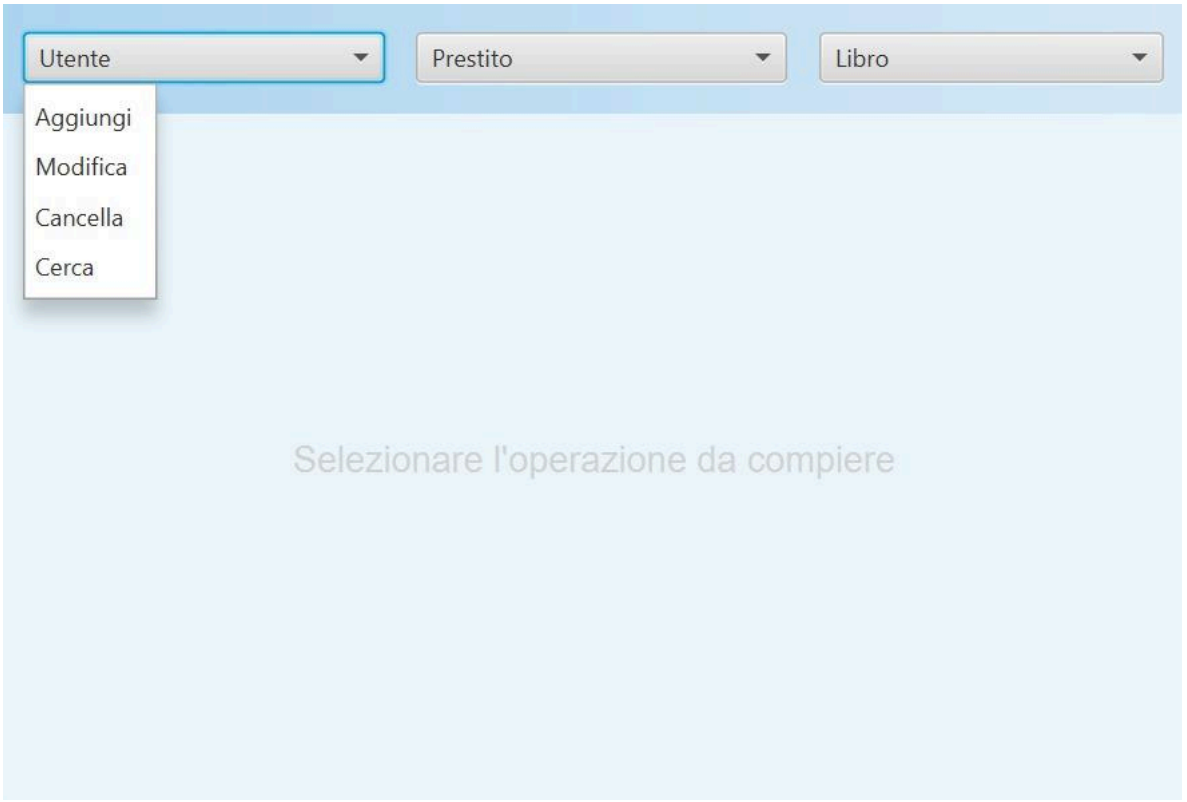
3. **Interazioni previste**

L'utente inserisce username e password e clicca su Conferma:

- Se i dati sono corretti accede alla pagina principale
- Se i dati sono errati il label nascosto diventa visibile e mostra un messaggio di errore

Cliccando su Modifica Credenziali viene aperto una nuova pagina che permette la modifca e l'aggiornamento delle credenziali

4.2 INTERFACCIA PRINCIPALE



1. **Descrizione funzionale**

Questo mock-up rappresenta la pagina principale mostrata all'utente in seguito all'autenticazione.

Il bibliotecario attraverso questa pagina riesce a gestire tutti i dati relativi alla Biblioteca.

Le operazioni sono divise in moduli in base ai dati di riferimento; il menù mostra questa divisione e per ogni ambito sono presenti le scelte mostrate.

2. **Elementi tecnici**

È presente un Pane principale che contiene i diversi elementi:

Label centrale che indica all'utente di scegliere le operazioni da eseguire

MenuBar che contiene i moduli attraverso la quale l'utente può interagire:

- Utente
- Prestito

- Libro

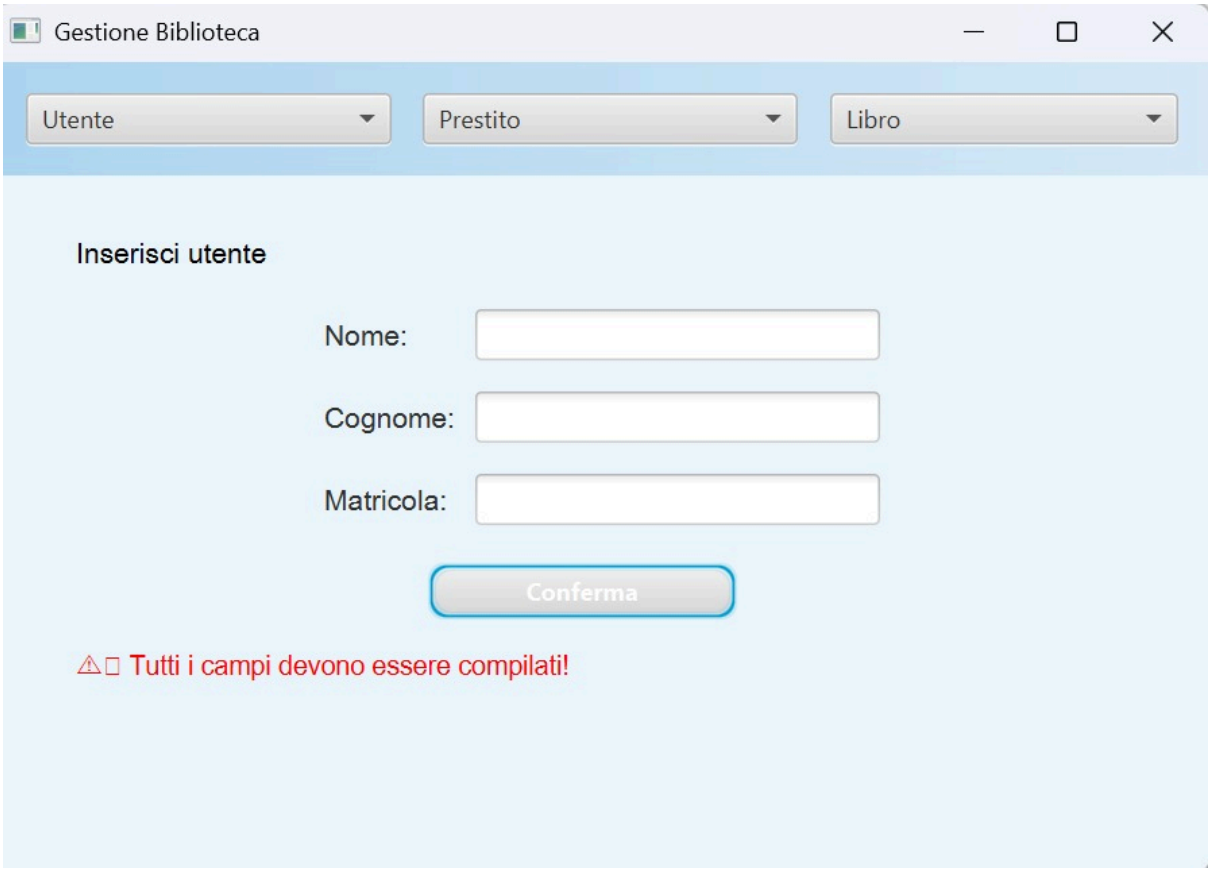
MenuItem, ogni modulo del MenuBar contiene i seguenti elementi specializzati per i diversi moduli:

- Aggiungi
- Modifica
- Visualizza
- Cerca
- Elimina

3. Interazioni previste

L'utente clicca su uno dei moduli previsti dal Menù e inseguito scegliere l'operazione da compiere.

4.3 INTERFACCIA GENERALE SVOLGIMENTO DI UNA OPERAZIONE



1. **Descrizione funzionale**

Questo mock-up, infine, mostra la selezione di una delle opzioni, in particolare quello dell'inserimento dell'utente.
Il bibliotecario attraverso questa interfaccia riesce, in questo particolare caso, a registrare un nuovo utente nel sistema, inserendo nome cognome e matricola.

2. **Elementi tecnici**

Un panel contiene i principali elementi:

- MenuBar attraverso il quale si può selezionare uno dei moduli presenti
- Label
    - "Inserisci Utente"
    - "Nome:"
    - "Cognome:"
    - Label che compare, come nell'esempio, in caso di warning
- TextField:attraverso i quali il bibliotecario inserisce i dati dell'utente delineati dalle Label
- Button di conferma per inviare i dati affinchè il sistema li salvi

3. **Interazioni principali**

Il bibliotecario attraverso questa interfaccia può navigare all'interno del sistema attraverso il Menu selezionando uno dei moduli oppure può concludere l'operazione che ha selezionato inserendo i dati specifici e confermando l'inserimento.