DESIGN - GESTIONE DI UNA BIBLIOTECA UNIVERSITARIA

Gruppo n. 15

Francesco Grimaldi
Aniello Ercolano
Mario Dello Russo
Pierpaolo Corcione

INDICE

PREMESSE

Nella progettazione del sistema è stato adottato il pattern architetturale MVC (Model-View-Controller), che prevede la suddivisione dell'applicazione in tre componenti: il Model, che si occupa dei dati; la View, che permette di visualizzare i dati grazie all'interfaccia; il Controller, che riceve input e modifica gli altri due componenti di conseguenza.

La scelta di questo stile architetturale risulta particolarmente coerente con l'utilizzo di un linguaggio di programmazione orientato agli oggetti quale Java, che mette a disposizione lo strumento JavaFX per la cura della GUI.

INTRODUZIONE AGLI SCENARI

L'applicazione sarà suddivisa in varie scene che si presenteranno dinamicamente in base a come si interagisce con un menù. La prima scena in assoluto che viene mostrata è la *'**schermata di login**'* e, interagendo con essa sotto certe condizioni, è possibile passare alla *'**home page**'* dove finalmente si può iniziare a sfruttare tutte le operazioni messe a disposizione dal sistema.

La *schermata di login* si presenta come una normale pagina di accesso con username e password (oppure come semplice accesso senza credenziali). Se il login va a buon fine, verrà caricata la *home page*.

La *home page* invece si presenta sottoforma di split pane dove a sinistra verrà presentato un menù con tutte le operazioni a disposizione e il '***menù delle impostazioni***' in basso, mentre a destra si potrà effettivamente interagire con lo scenario desiderato. La parte sinistra che rappresenta il menù rimarrà fissa durante tutta l'esecuzione dell'applicazione mentre la parte destra dello split pane sarà la vera parte dinamica del sistema.
Il menù metterà a disposizione in totale 8 operazioni fondamentali suddivise in 4 macrocategorie. Si presenterà quindi come:

- **UTENTI**
  - ➔ Lista utenti
  - ➔ Aggiungi utente
- **CATALOGO**
  - ➔ Visualizza catalogo
  - ➔ Aggiungi libro
- **PRESTITI E RESTITUZIONI**
  - ➔ Prestiti attivi
  - ➔ Aggiungi prestito
  - ➔ Storico restituzioni
- **STATISTICHE**
  - ➔ Visualizza statistiche

DESCRIZIONE DETTAGLIATA DEGLI SCENARI

LISTA UTENTI

DESCRIZIONE GRAFICA

Alla pressione della voce *'Lista utenti'* verrà caricata nella parte destra della scena la **tabella relativa agli utenti**. Al di sopra della tabella sarà presente una **barra di ricerca** che il bibliotecario potrà usare per cercare un gruppo di utenti che hanno almeno uno dei campi compatibili alla stringa usata come pattern di ricerca. La tabella è ordinata di default per nome e cognome e, ovviamente, il bibliotecario può scegliere di ordinarla secondo l'ordine della colonna che preferisce.

Gli elementi della tabella sono di sola visualizzazione. Al doppio click su una riga si apre una **finestra pop-up** che mostra tutti i dati dettagliati dell'utente selezionato e ne permette la rimozione o la modifica tramite l'azione di **due pulsanti**.

Il pulsante *'elimina'* serve a visualizzare un **ulteriore pop-up** che chiede al bibliotecario la conferma dell'eliminazione. Se il bibliotecario procede all'eliminazione, l'utente verrà eliminato dinamicamente dalla tabella e il suo record verrà cancellato dal file degli studenti.

Invece se il bibliotecario preme il pulsante *'modifica'*, tutti i campi del pop-up informativo diventano modificabili. Vengono mostrati dei **nuovi pulsanti** *'salva'* e *'annulla'*. Se viene premuto il tasto "annulla" l'operazione viene abortita. Invece, se viene premuto il tasto *'salva'* viene controllata la validità dei campi e se questi sono corretti la modifica va a buon fine, sia graficamente (in modo dinamico) che all'interno del file degli utenti.

DESCRIZIONE IMPLEMENTATIVA

La tabella viene implementata con una **lista osservabile** ed ordinata per cognome e nome mentre la barra di ricerca viene implementata con un **textfield** che agisce sulla lista di utenti riempiendo una **lista filtrata** secondo la stringa inserita.

Quando si apre il pop-up informativo di un utente. Quando viene cliccato il pulsante *'modifica'* o *'elimina'*, la scena sarà bloccata fino a quando il pop-up non sarà chiuso tramite l'utilizzo dei pulsanti.

OSSERVAZIONI

Per implementare questo scenario abbiamo bisogno delle classi **Utente** e **GestoreUtenti**.

VISUALIZZA CATALOGO

DESCRIZIONE GRAFICA

Alla pressione della voce *'Visualizza catalogo'* verrà caricata nella parte destra della scena la **tabella relativa al catalogo dei libri**. Al di sopra della tabella sarà presente una **barra di ricerca** che il bibliotecario potrà usare per cercare un gruppo di libri che hanno almeno uno dei campi compatibili alla stringa usata come pattern di ricerca. La tabella è ordinata di default per titolo e, ovviamente, il bibliotecario può scegliere di ordinarla secondo l'ordine della colonna che preferisce.

Gli elementi della tabella sono di sola visualizzazione. Al doppio click su una riga si apre una **finestra pop-up** che mostra tutti i dati dettagliati del libro selezionato e ne permette la rimozione o la modifica tramite l'azione di **due pulsanti**.

Il pulsante *'elimina'* serve a visualizzare un **ulteriore pop-up** che chiede al bibliotecario la conferma dell'eliminazione. Se il bibliotecario procede all'eliminazione, il libro verrà eliminato dinamicamente dalla tabella e il suo record verrà cancellato dal file del catalogo.

Invece se il bibliotecario preme il pulsante *'modifica'*, tutti i campi del pop-up informativo diventano modificabili (tranne ISBN e data di pubblicazione). Vengono mostrati dei **nuovi pulsanti** *'salva'* e *'annulla'*: se viene premuto il tasto "annulla" l'operazione viene abortita invece, se viene premuto il tasto *'salva',* viene controllata la validità dei campi. Se questi sono corretti la modifica va a buon fine, sia graficamente (in modo dinamico) che all'interno del file del catalogo.

DESCRIZIONE IMPLEMENTATIVA

La tabella viene implementata con una **lista osservabile** ed ordinata per titolo mentre la barra di ricerca viene implementata con un **textfield** che agisce sulla lista dei libri riempiendo una **lista filtrata** secondo la stringa inserita.

Quando si apre il pop-up informativo di un libro, questo puoò essere chiuso solo mediante l'azione dei pulsanti del pop-up. Quando viene cliccato il pulsante *'modifica'* o *'elimina'*, la scena sarà bloccata fino a quando il pop-up non sarà chiuso tramite l'utilizzo dei pulsanti.

OSSERVAZIONI

Per implementare questo scenario abbiamo bisogno delle classi **Libro** e **GestoreLibri**.

AGGIUNGI UTENTE

DESCRIZIONE GRAFICA

Nella scena devono essere visualizzati una **Label** e un **TextField** per ogni attributo dell'utente da inserire. In fondo alla pagina sono presenti **due pulsanti**: *'Aggiungi'* e *'Annulla'*.

DESCRIZIONE IMPLEMENTATIVA

Le generalità che vengono raccolte in questa pagina verranno usate per creare un **oggetto Utente** ma prima di inserirlo verrà controllato se è già presente.

Inoltre, se almeno un campo viene compilato in modo errato (nome e cognome devono essere caratteri alfabetici, la matricola deve essere formata da 10 numeri e l'email deve avere come dominio "@studenti.unisa.it") il pulsante "aggiungi" sarà disattivato.

OSSERVAZIONI

Per implementare questo scenario abbiamo bisogno delle classi **Utente** e **GestoreUtenti**.

AGGIUNGI LIBRO

DESCRIZIONE GRAFICA

Nella scena devono essere visualizzati una **Label** e un **TextField** per ogni attributo del libro da inserire. In fondo alla pagina sono presenti **due pulsanti**: *'Aggiungi'* e *'Annulla'*.

DESCRIZIONE IMPLEMENTATIVA

Le informazioni che vengono raccolte in questa pagina verranno usate per creare un **oggetto Libro** e se si prova ad inserire un Libro già presente nel sistema (in TUTTE le sue caratteristiche), di quest'ultimo verranno incrementate solo il numero di copie. Questa situazione si verifica solo se tutti i campi inseriti sono identici, mentre se a coincidere è solo l'ISBN ma almeno uno degli altri campi è diverso l'inserimento verrà inibito.

Inoltre, se almeno un campo viene compilato in modo errato (nome e cognome devono essere caratteri alfabetici, la matricola deve essere formata da 10 numeri e l'email deve avere come dominio "@studenti.unisa.it") il pulsante "aggiungi" sarà disattivato.

OSSERVAZIONI

Per implementare questo scenario abbiamo bisogno delle classi **Libro** e **GestoreLibri**.

AGGIUNGI PRESTITO

DESCRIZIONE GRAFICA

Nella scena devono essere presenti **3 Label** e **3 TextField** che permetteranno l'inserimento della matricola dello studente, dell'ISBN del libro e della data prevista di restituzione. Inoltre, saranno visualizzati due pulsanti "Aggiungi" e "Annulla". Quando verrà premuto il pulsante "Annulla" tutti i campi saranno svuotati.

DESCRIZIONE IMPLEMENTATIVA

I campi matricola e ISBN possono essere implementati con un semplice **TextField**, mentre il campo data sarà implementato con una casella di tipo calendario (**DatePicker**).

La data prevista per la restituzione non deve essere precedente alla data odierna e non deve superare i 6 mesi dalla stessa. Una volta inserite la matricola e l'ISBN, si procede con la ricerca di questi dati nei rispettivi file. Quando questi dati saranno trovati verrà creato l'**oggetto Prestito**. Se il prestito va a buon fine, sarà decrementato il numero di copie disponibili per il libro prestato e il libro sarà aggiunto alla lista dei prestiti attivi dell'utente oltre all'aggiunta di un record nel file dei prestiti.

Se l'utente ha già tre prestiti attivi, l'utente specificato non è presente nel sistema, il libro specificato non è presente nel catalogo o non sono disponibili eventuali copie verrà inibito il prestito. La classe Prestito ha due riferimenti, uno alla classe Utente e uno alla classe Libro. Inoltre, ha due attributi di tipo LocalDate: uno per la data di inizio prestito e uno per la data prevista di restituzione. Nel file dei prestiti sarà creato un record che contiene la matricola dell'utente, l'ISBN del libro,, un ID univoco e le due date.

OSSERVAZIONI

Per implementare questo scenario abbiamo bisogno delle classi **Prestito**, **Utente**, **Libro**, **GestoreUtenti**, **GestoreLibri** e **GestorePrestiti**.

VISUALIZZA STATISTICHE

DESCRIZIONE GRAFICA

La scena sarà composta da varie **label**, **grafici** o **immagini** che raffigurano i risultati di alcuni calcoli statistici effettuati su tutta la biblioteca. Queste statistiche verranno aggiunte in base alla necessità manageriale del bibliotecario.

DESCRIZIONE IMPLEMENTATIVA

Per l'implementazione ci serviamo di tutti i dati che abbiamo a disposizione nelle **strutture dati** gli utenti, i libri, i prestiti e le restituzioni elaborandoli matematicamente per fini statistici.

OSSERVAZIONI

Per implementare questo scenario abbiamo bisogno delle classi **GestoreUtenti**, **GestoreLibri**, **GestorePrestiti** e **GestoreRestituzioni**.

PRESTITI ATTIVI

DESCRIZIONE GRAFICA

Nella scena sarà presente la **tabella relativa ai prestiti attivi**. Essa conterrà le informazioni più rilevanti come: nome dell'utente, cognome dell'utente, matricola, ISBN, data di inizio prestito, data prevista di restituzione e una colonna che indicherà di quanti giorni il prestito è in ritardo o in anticipo. Se il prestito è in ritardo, la casella che contiene il ritardo negativa, altrimenti sarà positiva. Inoltre in alto sarà presente anche una **barra di ricerca** come implementata già per gli utenti ed il catalogo. Se viene effettuato un doppio click su un prestito si aprirà un pop-up che mostrerà tutti gli attributi dell'utente, del libro e le date di inizio prestito e di prevista restituzione. Inoltre, in questo pop-up sarà presente un pulsante '*Termina'* che permetterà di terminare il prestito.

DESCRIZIONE IMPLEMENTATIVA

La tabella sarà implementata con una **lista osservabile** ed ordinata per data prevista di restituzione. Quando viene premuto il pulsante '*Termina'* , il prestito sarà eliminato dalla lista dei prestiti attivi dell'utente, dalla tabella dei prestiti attivi e dal file della lista dei prestiti. Inoltre, il numero di copie del libro sarà incrementato di 1 e il prestito attivo diventerà una restituzione (verrà aggiunta alla tabella e al file delle restituzioni).

OSSERVAZIONI

Per implementare questo scenario abbiamo bisogno delle classi **Prestito** e **GestorePrestiti**.

STORICO RESTITUZIONI

DESCRIZIONE GRAFICA

Nella scena verrà visualizzata una **tabella con lo storico delle restituzioni** ordinata in ordine di data effettiva di restituzione. Ogni record di questa tabella avrà: nome dell'utente, cognome dell'utente, matricola, nome del libro, ISBN, data effettiva di restituzione e penale applicata. Facendo doppio click su un record sarà possibile visualizzare un pop-up informativo con le caratteristiche dettagliate della restituzione. Questa scena non è dinamica visto che il bibliotecario non può interagire con nessun pulsante data la natura puramente informativa di questa sezione.

DESCRIZIONE IMPLEMENTATIVA

La tabella verrà implementata con una **lista osservabile** ordinata per inserimento (quindi data effettiva di restituzione). La penale verrà calcolata come il 5% del valore commerciale del libro per ogni giorno di ritardo alla consegna. Per ogni restituzione verrà registrato nel file un record con tutti i dati del prestito oltre alla data effettiva di restituzione.

OSSERVAZIONI

Per l'implementazione di questa sezione abbiamo bisogno della classe **Prestito**, **Restituzione** e **GestoreRestituzioni**.

SCELTE PROGETTUALI

- Si è scelto di implementare una classe di gestione per ogni entità fondamentale del sistema. Ogni classe 'manageriale' implementa l'interfaccia *'Manager<T>'* dove T rappresenta il tipo che si vuole aggregare e gestire con varie operazioni.

- Si è scelto di non gestire il caso in cui l'applicazione si arresta in modo anomalo mentre si trova in una fase di elaborazione dei dati (salvataggio di un nuovo utente, modifica di un libro, cessazione di un prestito). La scelta deriva dalla natura implementativa del sistema che, nel peggiore dei casi, non riesce a salvare l'ultima modifica che è in elaborazione ma lascia comunque una situazione coerente dell'archivio.

- L'intero archivio del sistema verrà organizzato nei seguenti file:
  - ➔ **prestiti.csv**
  - ➔ **utenti.csv**
  - ➔ **libri.csv**
  - ➔ **restituzioni.csv**
  - ➔ **credenziali.csv**

- Deve essere garantita la coerenza tra il contenuto dell'archivio e le strutture gestite dai *Manager*.

- Ogni *Manager* è implementato seguendo la definizione di *singleton,* garantendo così una singola istanza univoca di ogni classe di questo tipo.

- Ogni classe che viene gestita dai *manager* implementa l'interfaccia *searchable* così da ereditare il metodo `boolean containsString()` utile a capire se l'oggetto in questione è conforme a un pattern di ricerca (semplifica la logica di ricerca).

- Ogni *Manager* utilizza il metodo sopracitato per implementare il metodo `List<T> containsPattern()` che ritorna una lista di oggetti corrispondenti ad un certo pattern di ricerca.

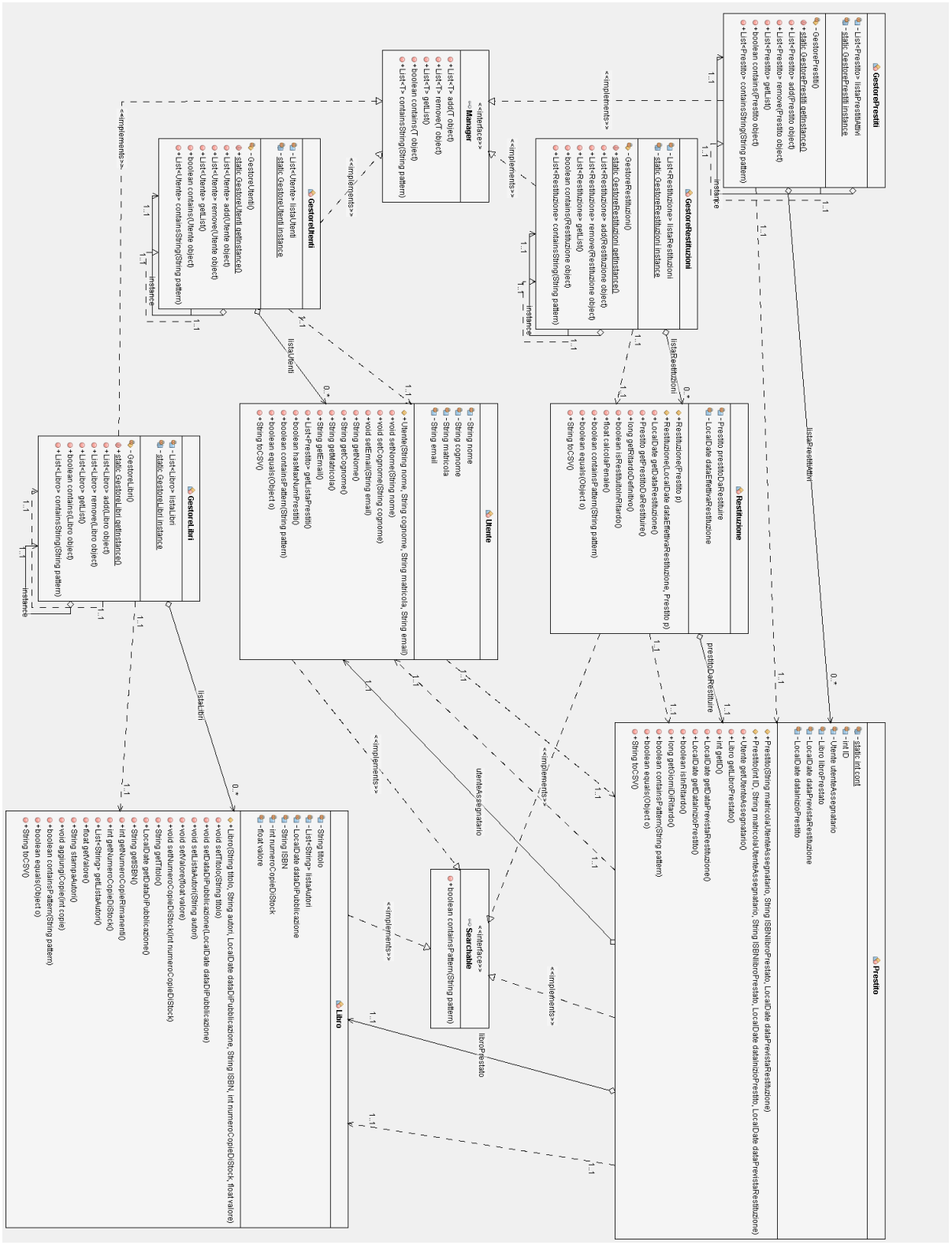
- La lista dei prestiti attivi di ogni utente non è un attributo della classe utente ma bensì qualcosa che può essere ricavato dinamicamente dalla lista dei prestiti attivi.

- Le credenziali sono custodite in modo non sicuro per semplicità di implementazione.

- Per altrettanta semplicità, non è gestito il caso di *recupero delle credenziali* che potranno essere recuperate solo aprendo esplicitamente il file CSV. (estremamente poco sicuro e rudimentale, ma in linea con gli obiettivi del progetto)

- I test sono stati implementati solo per la parte *Model* del sistema per scelta tecnica. Inoltre non sono stati testati i metodi setter e getter vista la loro banale implementazione.

DIAGRAMMA DELLE CLASSI



ANALISI DELLA COESIONE E DELL'ACCOPPIAMENTO

1. Architettura e Separazione delle Preoccupazioni (SoC)

In conformità con il principio di **Separazione delle Preoccupazioni (Separation of Concerns)**, il diagramma delle classi evidenzia una netta distinzione tra la rappresentazione dei dati e la logica di gestione.

- **Model (Entità):** Le classi `Libro`, `Utente`, `Prestito` e `Restituzione` sono progettate come "Plain Old Java Objects" (POJO) o entità di dominio. La loro responsabilità primaria è mantenere lo stato e garantire l'invarianza dei dati (es. tramite costruttori e setter), aderendo al principio di **Singola Responsabilità (SRP)**.
- **Logic (Gestori):** La logica applicativa, inclusa la gestione delle collezioni e la ricerca, è delegata alle classi "Gestore" (`GestoreLibri`, `GestoreUtenti`, ecc.). Questo evita che le entità diventino oggetti con troppe responsabilità.

2. Astrazione e Principio DRY (Don't Repeat Yourself)

Per evitare duplicazioni di codice e standardizzare le operazioni Create, Read, Update, Delete, è stata introdotta l'interfaccia generica `Manager<T>`. Come visibile nel diagramma, tutte le classi gestore (`GestoreLibri`, `GestorePrestiti`, etc.) implementano questa interfaccia. Questa scelta progettuale soddisfa il principio **DRY (Don't Repeat Yourself)**, poiché definisce un contratto unico per le operazioni comuni (`add`, `remove`, `search`, `contains`, `containsString`), permettendo al sistema di essere esteso in futuro con nuovi tipi di entità senza dover ridefinire le firme dei metodi di gestione base.

3. Gestione delle Relazioni e Accoppiamento

Le relazioni tra le classi sono state modellate privilegiando l'associazione rispetto all'ereditarietà, riducendo l'accoppiamento rigido come suggerito dai principi di buona progettazione.

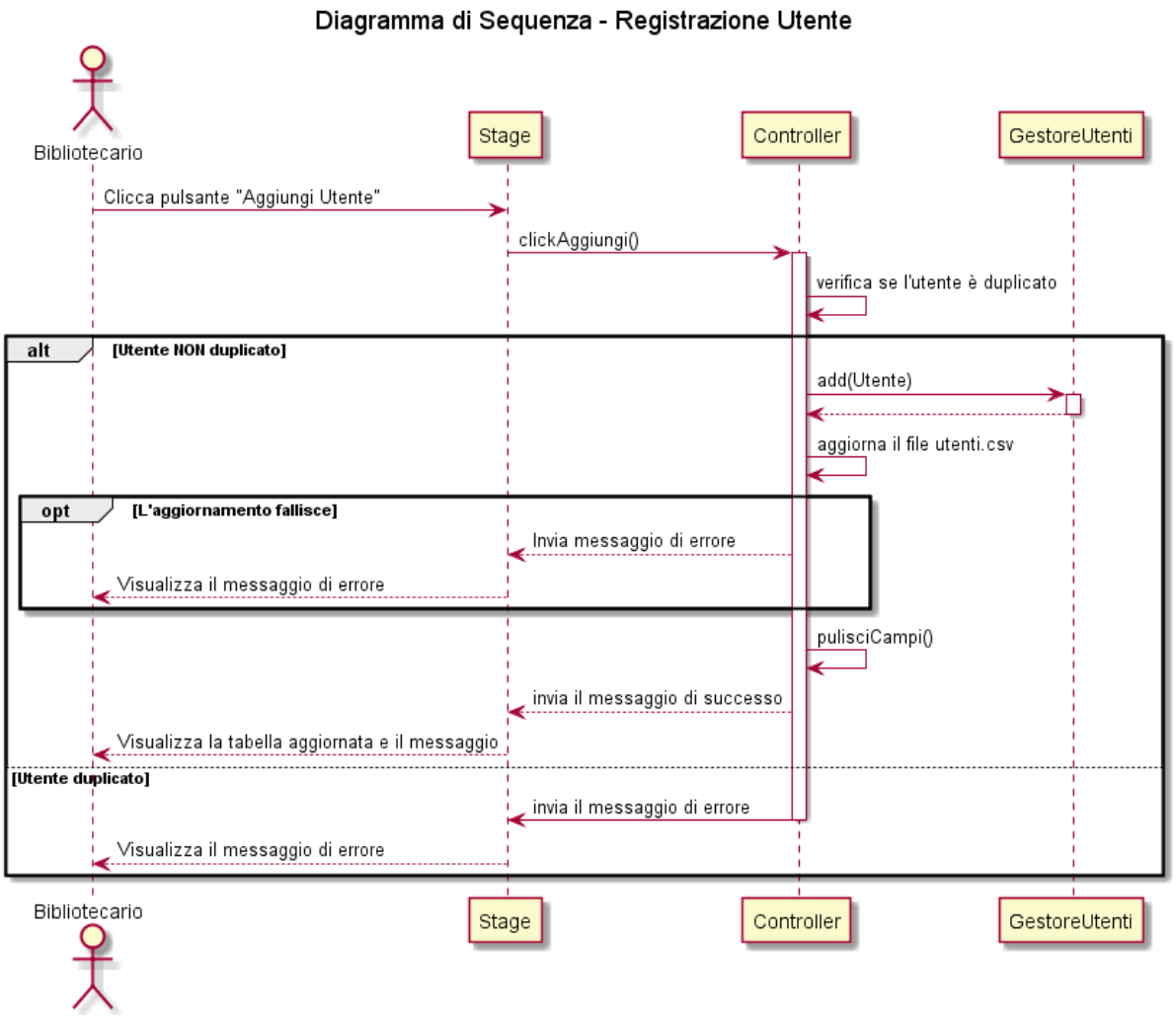
- **Associazioni "Has-A":** La classe `Prestito` mantiene associazioni dirette con `Utente` e `Libro`. Questo riflette la natura del dominio: un prestito "ha un" utente assegnatario e "ha un" libro oggetto del prestito.
- **Navigabilità:** Le relazioni sono state progettate per garantire la navigabilità necessaria senza creare cicli di dipendenza eccessivi. Ad esempio, `Utente` possiede una lista di prestiti ricavata dinamicamente con un metodo (`getListaPrestiti`), permettendo di risalire facilmente allo storico dei prestiti attivi dell'utente.

4. Pragmatismo e Principio KISS (Keep It Simple, Stupid)

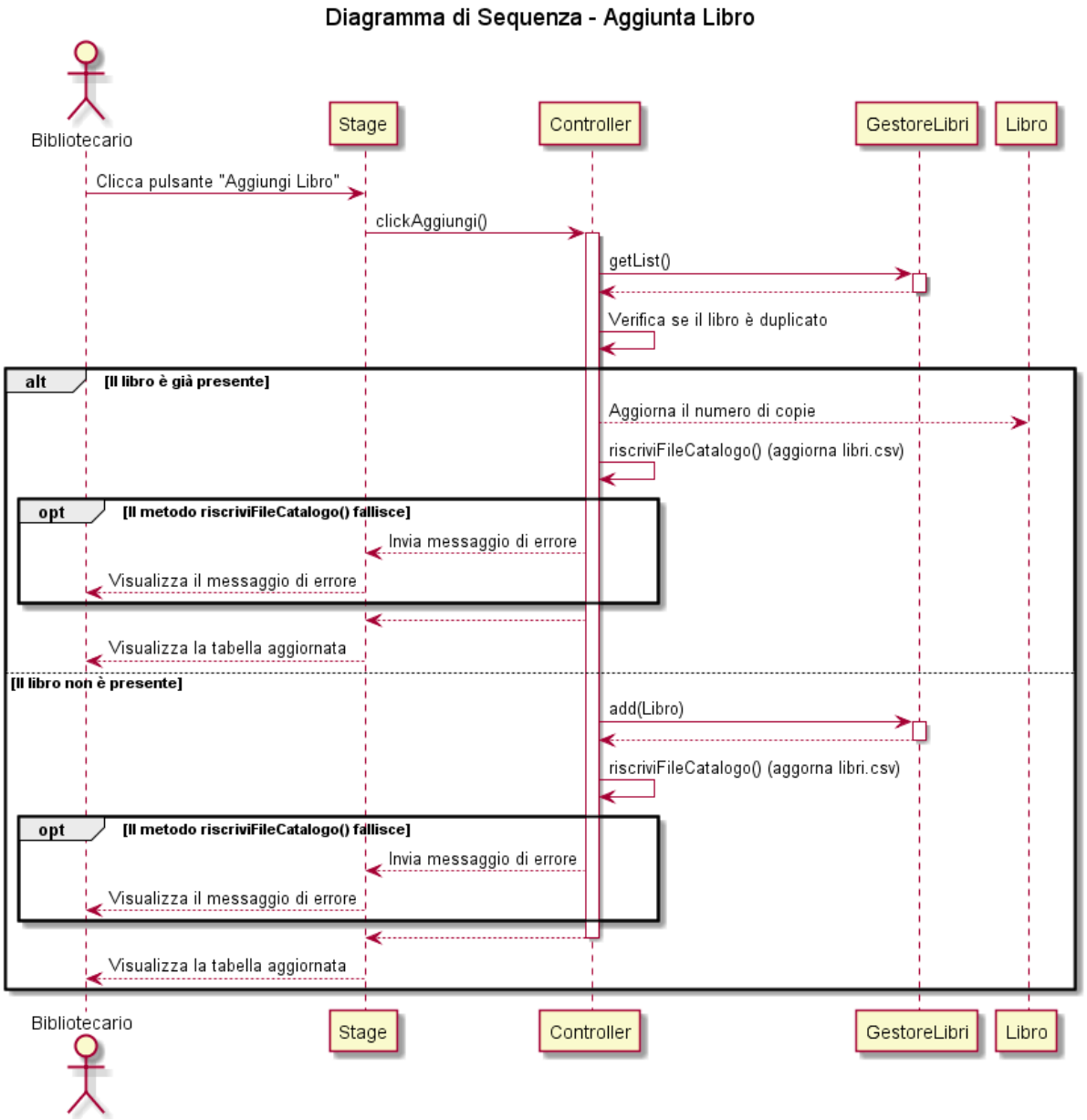
Nelle scelte implementative, come la gestione della persistenza tramite i metodi `toCSV()` presenti nelle entità, si è scelto di seguire il principio **KISS**. Sebbene si stia “infrangendo” il **Principio Di Singola Responsabilità** dato che una rigorosa architettura enterprise potrebbe richiedere un layer separato per la gestione dei dati, per rispettare il principio **YAGNI** (You Aren't Going to Need It) data la scala di questo progetto si è preferito mantenere la logica di serializzazione vicina ai dati. Questa scelta rappresenta un compromesso consapevole (un debito tecnico "prudente") per ridurre la complessità infrastrutturale e focalizzarsi sulla logica di dominio.

DIAGRAMMI DI SEQUENZA

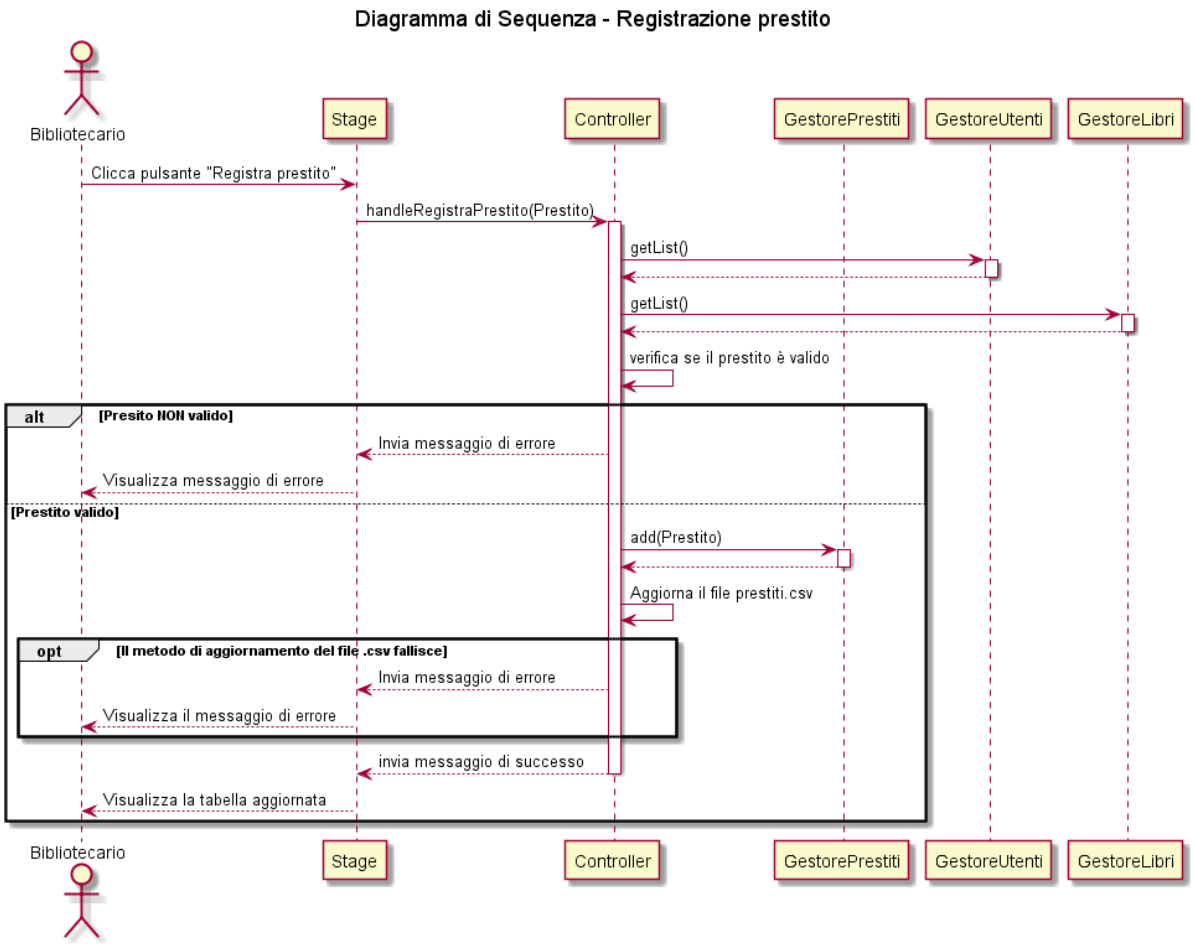
REGISTRAZIONE UTENTE



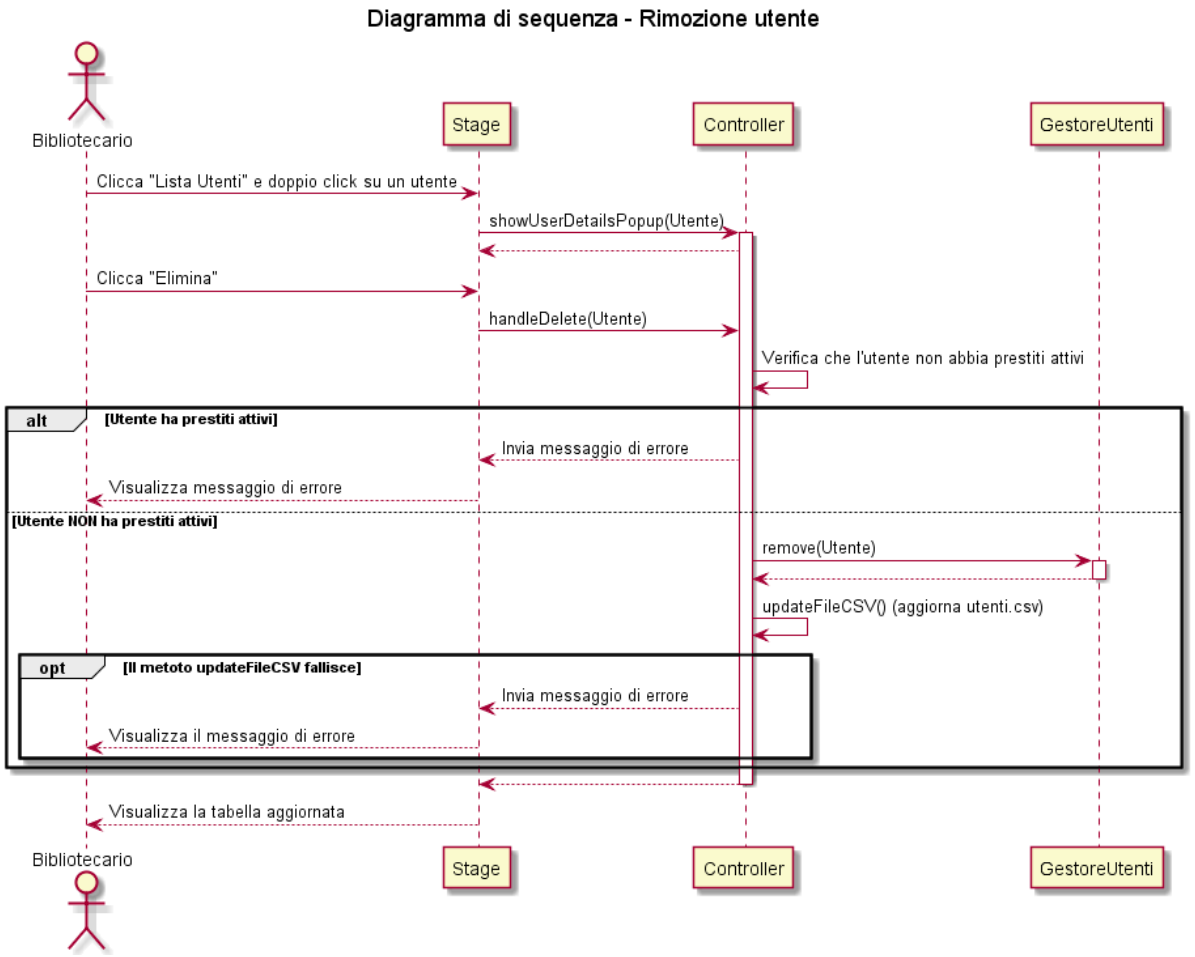
REGISTRAZIONE LIBRO



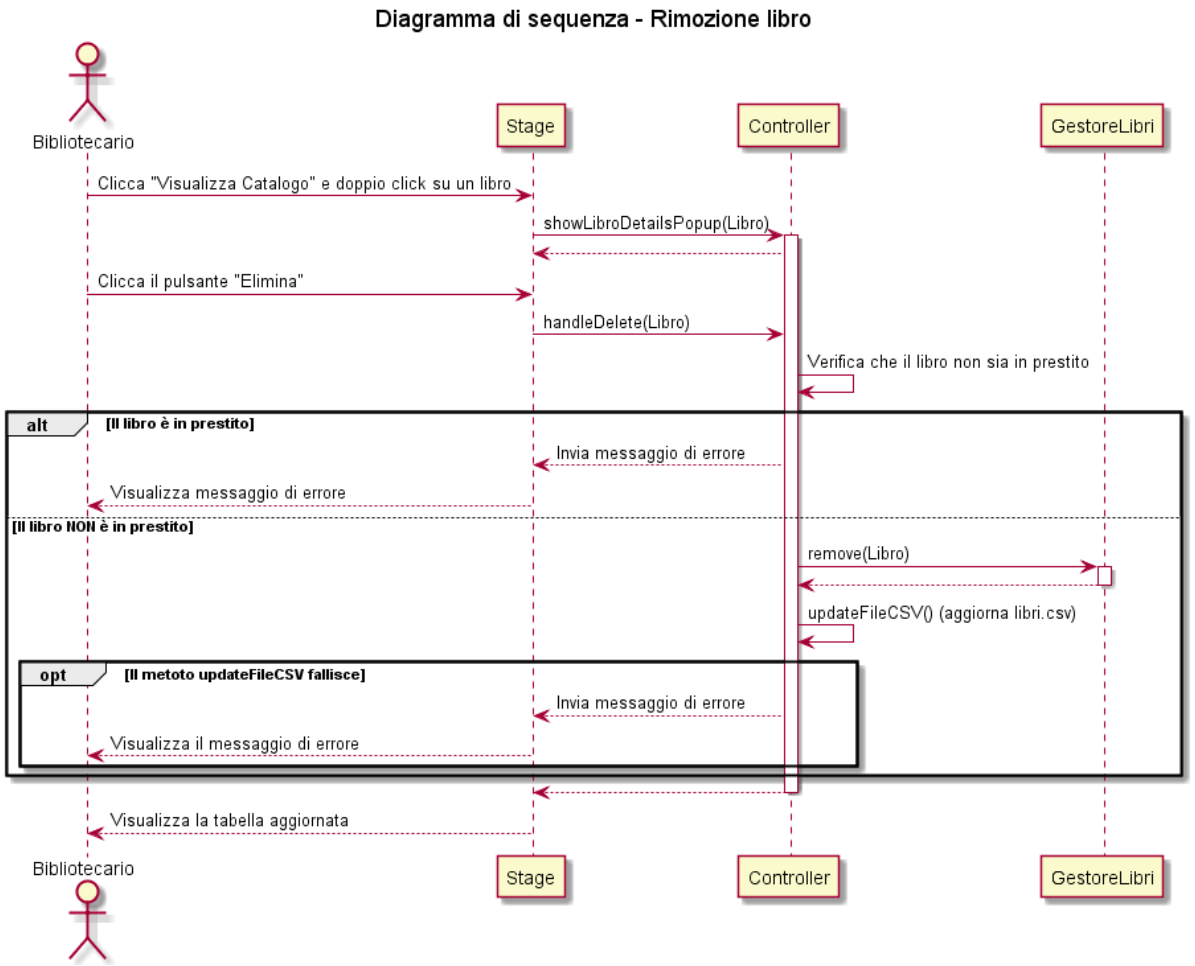
REGISTRAZIONE PRESTITO



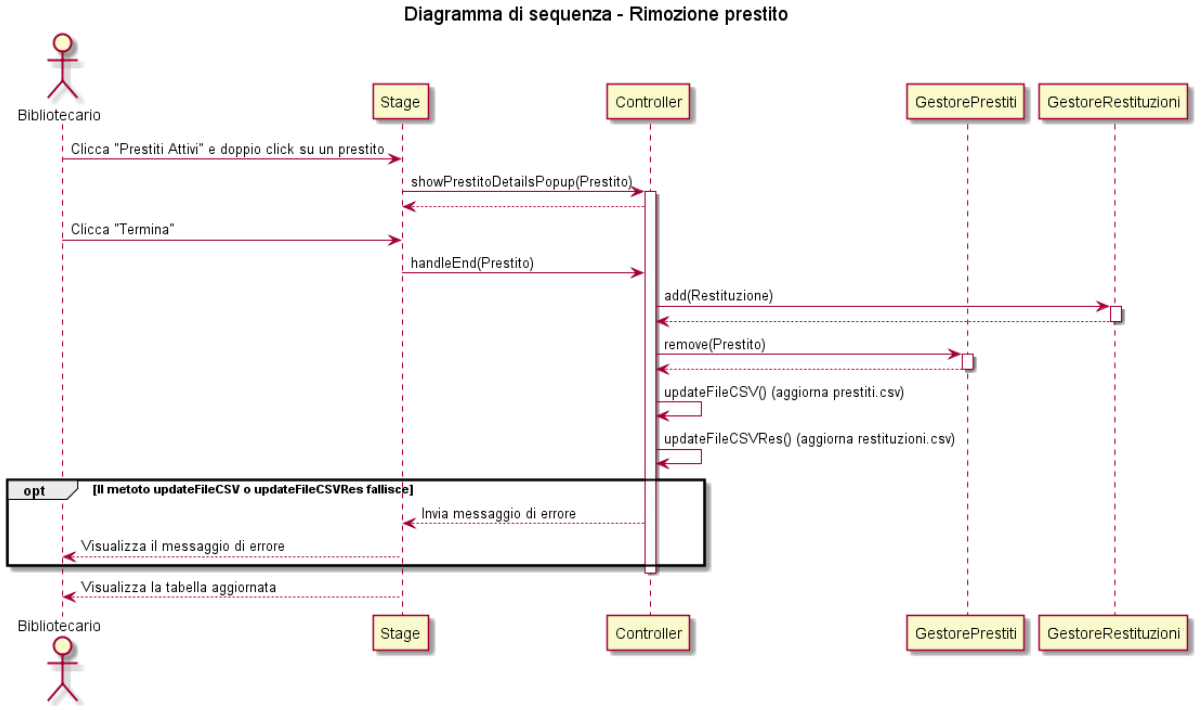
RIMOZIONE UTENTE



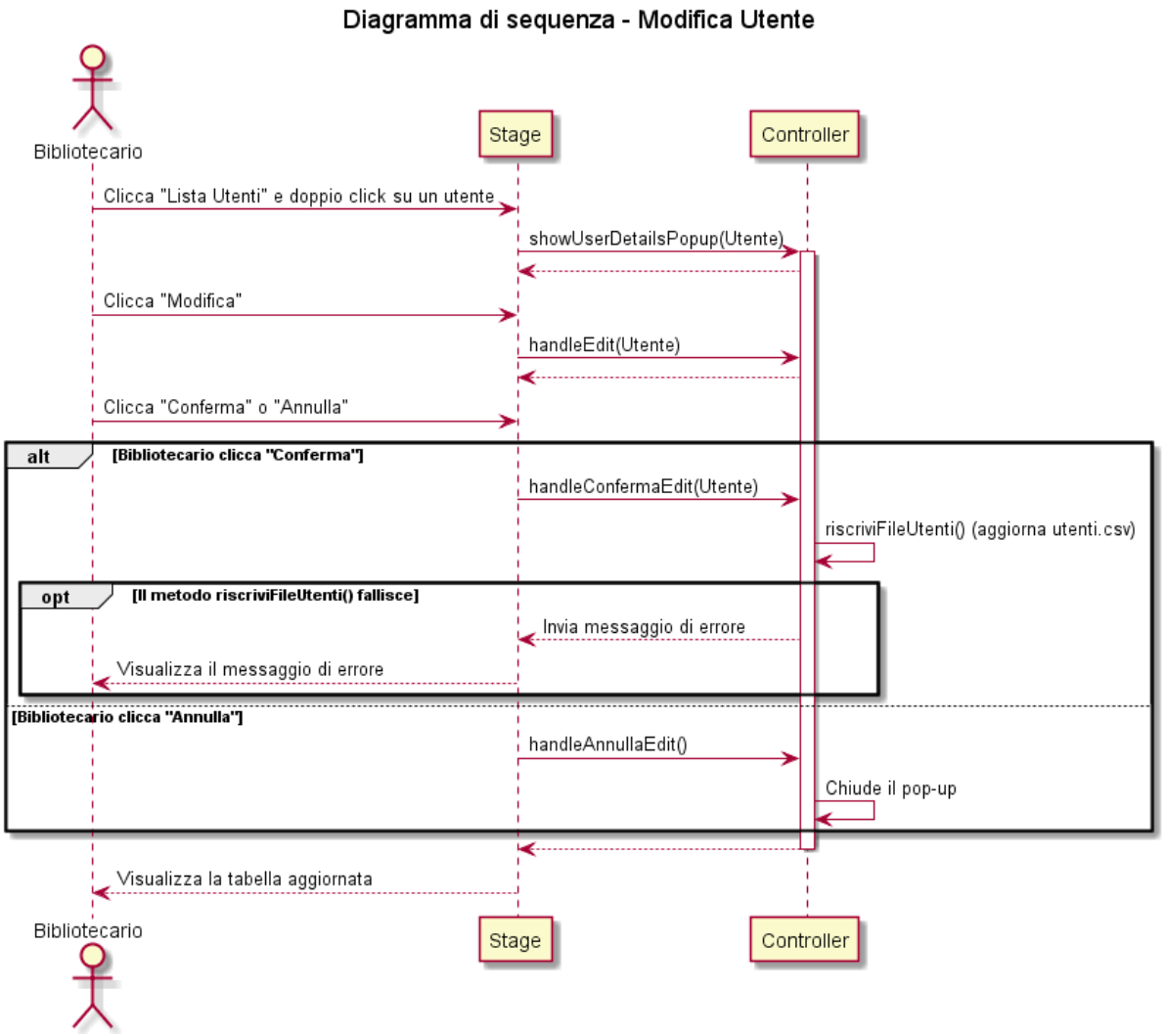
RIMOZIONE LIBRO



RIMOZIONE PRESTITO



MODIFICA UTENTE



MODIFICA LIBRO