

# Progettazione

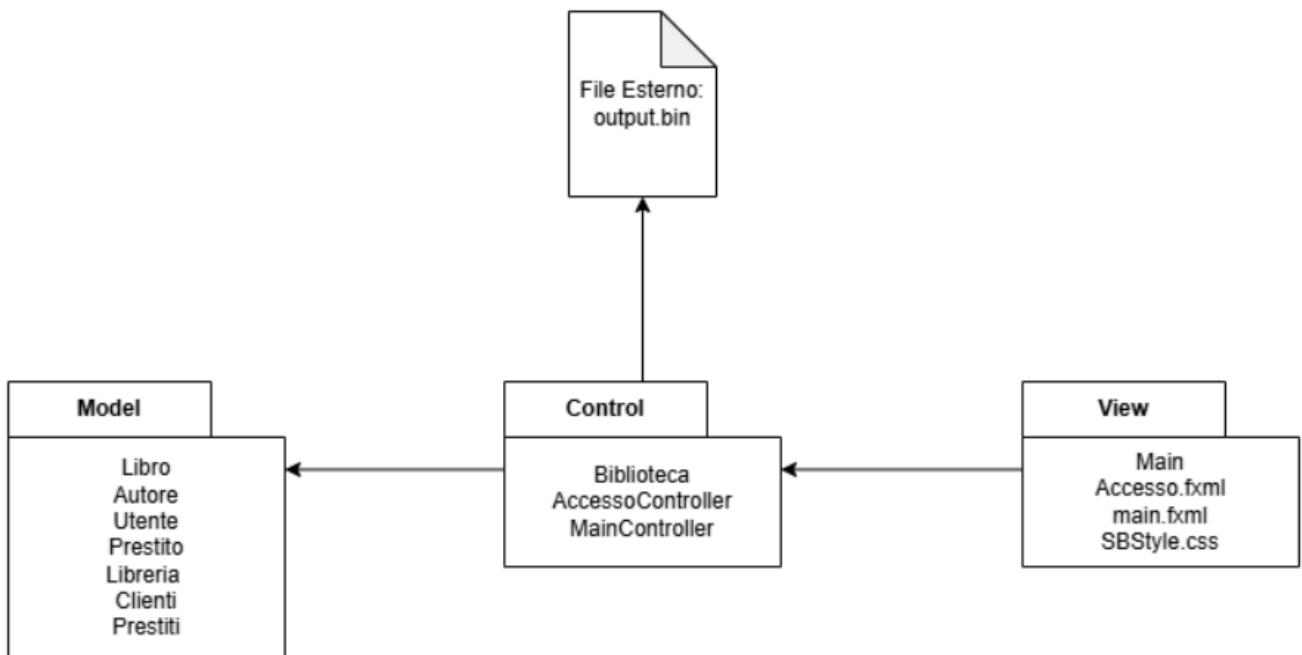
## Moduli principali

I moduli principali che compongono l'applicazione sono:

- **Libreria:** Il seguente modulo si occupa della gestione generale dei libri comprendendo le informazioni dei singoli componenti, i controlli di validità e le azioni disponibili.
  - **Autore:** È l'insieme delle informazioni relative agli autori (nome, cognome) di un singolo libro.
  - **Libro:** Rappresenta tutte le informazioni relative al libro (Titolo, Lista degli Autori, Anno di pubblicazione, Codice ISBN, Numero di copie disponibili).
  - **Libreria:** Contiene l'insieme dei libri presenti in archivio e fornisce le operazioni per aggiungere, modificare, eliminare e ricercare i libri.
- **Clienti:** Il seguente modulo si occupa della gestione di tutti gli utenti registrati della biblioteca comprendendo le informazioni di ogni utente, i controlli di validità delle informazioni e le azioni disponibili.
  - **Utente:** Comprende le informazioni relative ad ogni utente (Nome, Cognome, Matricola, E-mail istituzionale)
  - **Clienti:** È il contenitore che raggruppa tutti gli utenti registrati della biblioteca e fornisce i metodi per l'inserimento, la modifica, l'eliminazione e la ricerca degli utenti.
- **Prestiti:** Il seguente modulo si occupa della gestione dei prestiti che ogni utente può richiedere per ogni libro disponibile al momento della richiesta. Comprende l'associazione utente-libro, la data di restituzione prevista, il controllo per la scadenza ed il numero massimo di prestiti per singolo utente, oltre le azioni disponibili.
  - **Prestito:** Rappresenta le informazioni relative all'associazione utente-libro per un determinato intervallo di tempo e la data di restituzione prevista.
  - **Prestiti:** Raccoglie l'insieme dei prestiti ancora attivi e fornisce le operazioni per aggiungere, rimuovere e controllare i prestiti.

- **Interfaccia utente:** È il modulo che gestisce l'interfaccia con cui l'utente interagirà in prima persona.
  - **Main:** È il file main che verrà eseguito per avviare l'applicazione.
  - **MainController:** Gestisce l'interfaccia realizzata mediante FXML.
  - **AccessoController:** Gestisce l'accesso nell'interfaccia.
  - **main FXML:** È la rappresentazione dell'interfaccia grafica.
  - **accesso FXML:** È la rappresentazione dell'accesso nell'interfaccia grafica.
- **Biblioteca:** Il compito di questo modulo è quello di raccogliere e coordinare il funzionamento degli altri moduli, facendo interagire tra di loro la parte relativa alla tipologia di dati da salvare, il contenitore effettivo (archivio) dei dati salvati e l'interfaccia utente. Si occupa di: ricevere le richieste generate dalla Interfaccia Utente, utilizzare i moduli Libreria, Clienti, Prestiti, caricare i dati (all'avvio dell'applicazione) e del salvarli sul file esterno alla fine di ogni operazione.
- **DataBase:** È il modulo che comprende il file esterno che funzionerà da archivio permanente dei dati dell'applicazione (libri, utenti, prestiti, restituzioni) dal quale il modulo "Biblioteca" caricherà e salverà informazioni durante l'utilizzo dell'applicazione.

## Diagramma dei package



Per rappresentare l'architettura del Sistema Gestione Biblioteca Universitaria, abbiamo scelto di usare il pattern architetturale "MVC : Model Control View". Il diagramma infatti rappresenta l'organizzazione in package dei componenti del sistema suddivisi in base ai loro compiti.

### 1. Model

Questo package contiene i moduli "Libreria" (per la gestione dei Libri), "Clienti" (per la gestione degli utenti della Biblioteca) e "Prestiti" (per la gestione dei prestiti attivi e delle restituzioni), che insieme rappresentano il nucleo del sistema, in quanto contengono tutti i dati e tutta la logica del programma.

### 2. Control

Questo package contiene il modulo "Biblioteca", che funge il ruolo di "Controller" ovvero una sorta di "Coordinatore" che si occupa di gestire la comunicazione tra i package "Model" e "View". Andrà, infatti, a ricevere le richieste inviate dall'utente (tramite l'interfaccia utente, gestita da AccessoController e MainController) e

utilizzerà i moduli del “Model” per eseguire la logica richiesta. Infine, oltre a caricare i dati all’avvio dell’applicazione, salverà anche le informazioni su un File Esterno che funge da DataBase dopo ogni operazione.

### **3. View**

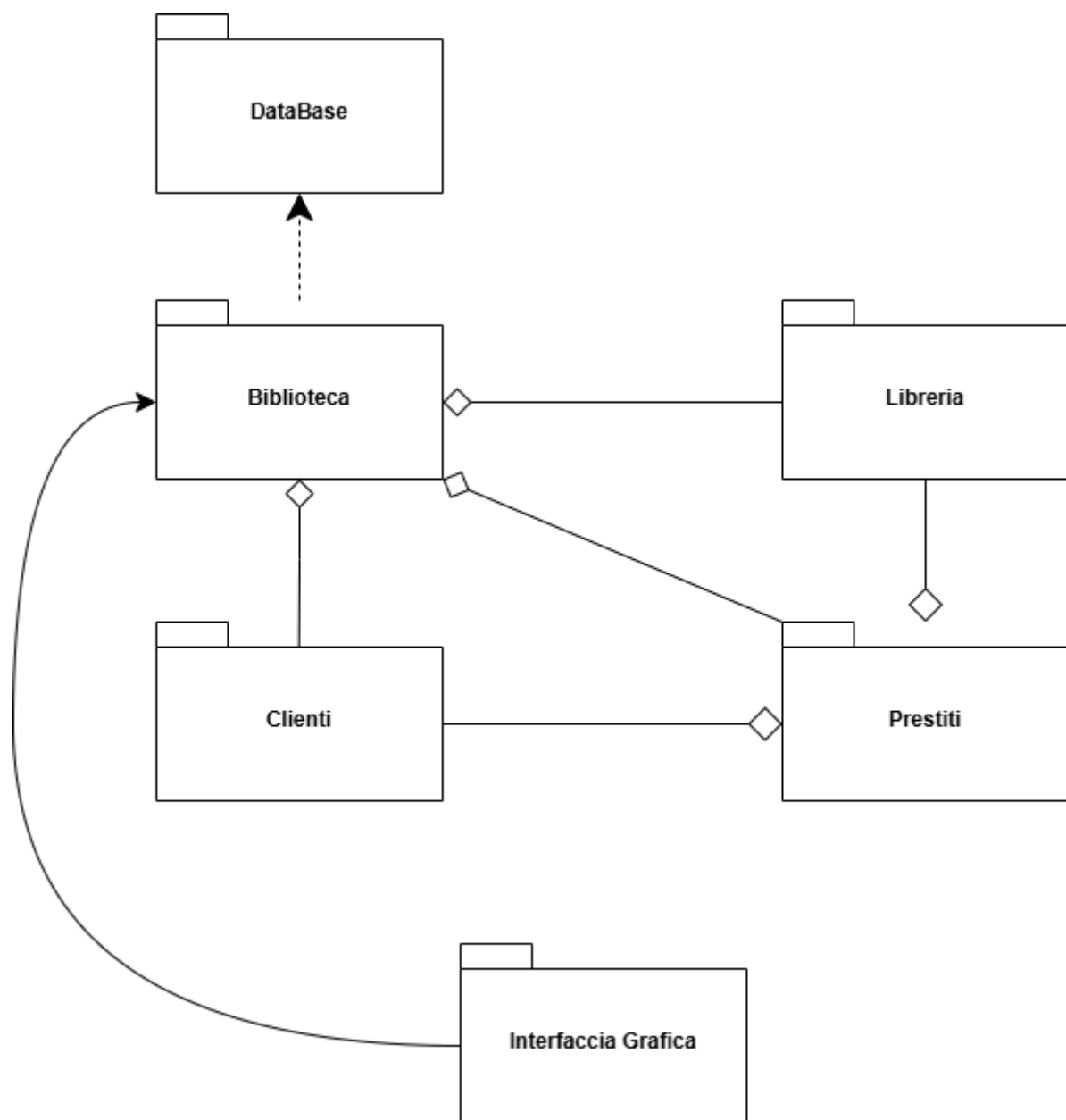
Questo package contiene il modulo “Interfaccia Utente”, che si occupa di gestire tutta la parte relativa alla “presentazione” dell’applicazione. Ha ,infatti, il compito di mostrare i dati all’utente ( dati recuperati dal package “Model” tramite il package “Control”), di catturare gli input dell’utente ( ad esempio il click del mouse ...), e di inviare delle richieste al package “Control”.

### **4. DataBase (File Esterno)**

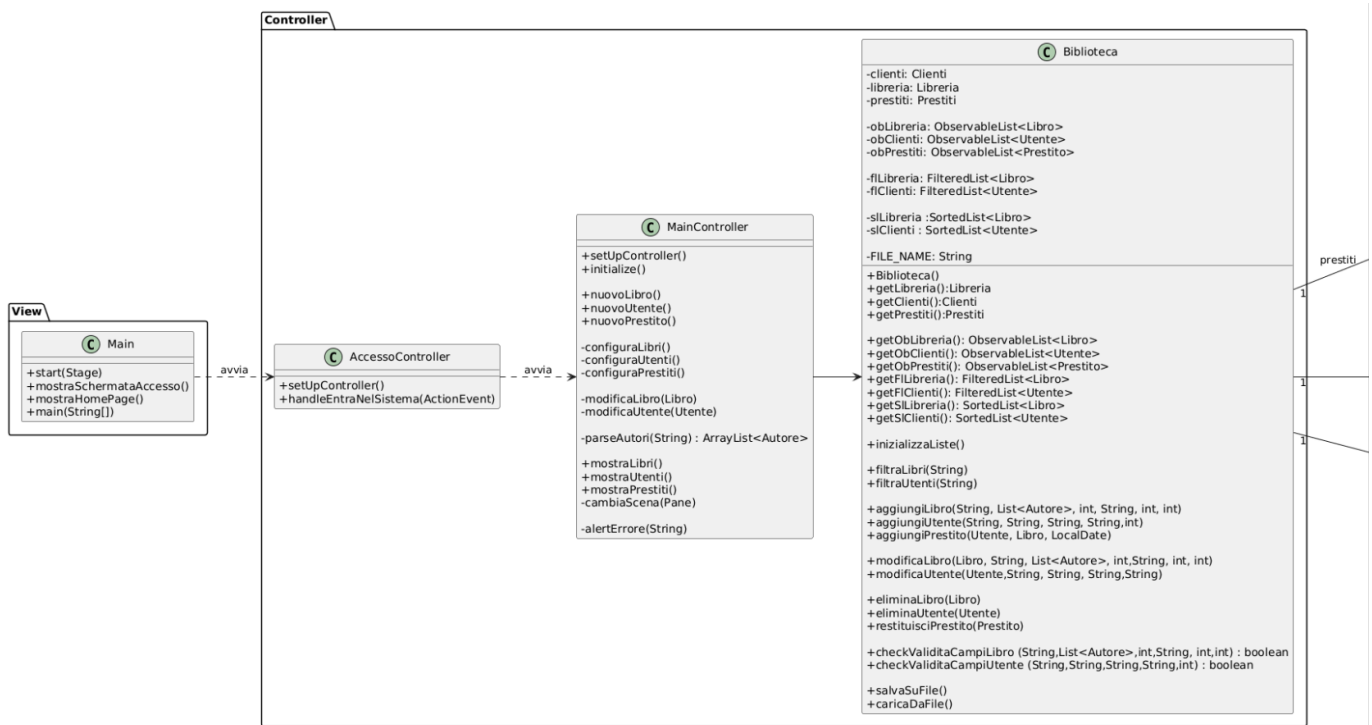
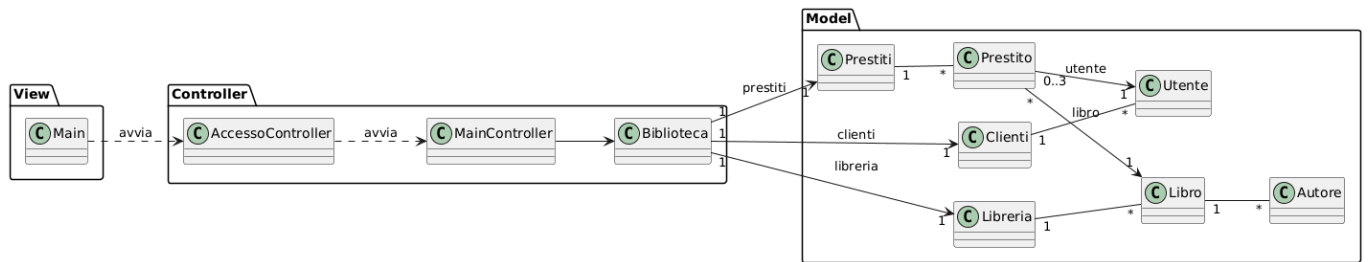
Questo package gestisce il File Esterno che viene utilizzato come un archivio di tutti i dati dell’applicazione, infatti in esso saranno memorizzate tutte le informazioni relative a Libri, Utenti e Prestiti, consentendo anche il caricamento di tali dati all’avvio del software.

## **Definizione delle dipendenze**

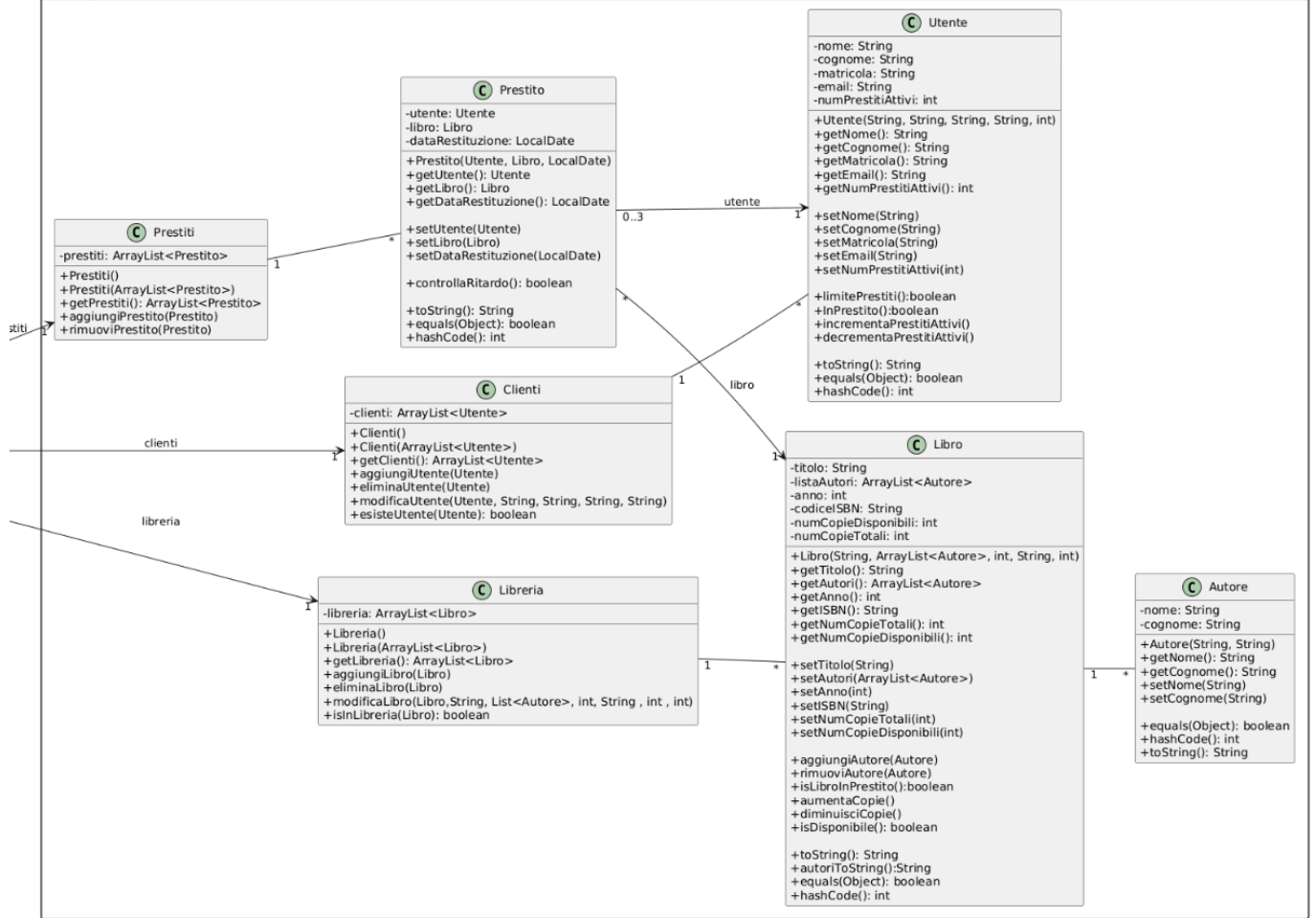
Il diagramma sottostante è un diagramma ad alto livello che mostra, in maniera generale quali sono le dipendenze tra i principali moduli che definiscono l'applicazione. Infatti è possibile notare la relazione di aggregazione tra il modulo Biblioteca e i tre moduli Clienti, Libreria e Prestiti e tra i Libreria e Prestiti e Clienti e Prestiti.



# Diagramma delle classi



# Model



## Suddivisione delle classi

- **Autore**

La classe Autore memorizza le informazioni anagrafiche (Nome,Cognome) di ciascun autore e fornisce le operazioni base per gestirli. Per quanto riguarda le relazioni, **la classe Autore è associata alla classe Libro**, questo significa che almeno un autore è contenuto in un oggetto Libro e che un autore può comparire in più libri (**relazione 1...\* Autori per ogni libro**). Possiede, inoltre, i seguenti metodi e attributi :

- **attributi:**

- **nome : String** – nome dell'Autore
- **cognome : String** – cognome dell'Autore

- **metodi:**

- **Autore(String nome, String cognome)**
- **getters()**
- **setters (String)**
- **toString() : String**
- **equals(Object o) : boolean**
- **hashCode() : int**

- **Libro**

La classe Libro memorizza le informazioni relative ad un Libro (Titolo, ISBN, Anno, Lista di Autori, Numero di Copie disponibili) e fornisce le operazioni base per gestirli. Per quanto riguarda le relazioni, **la classe Libro è una composizione della classe Autore**, questo significa che la classe Libro contiene una lista di oggetti Autore, **ma è anche associata con la classe Prestito**, perché un oggetto Libro è l'oggetto a cui si riferisce il prestito e un libro può comparire in più prestiti attivi essendoci più copie disponibili. Possiede, inoltre, i seguenti metodi e attributi :

- **attributi:**

- **titolo: String** — Titolo Libro
- **listaAutori: List<Autore>** — Autori del Libro
- **anno: int**
- **codiceISBN : String**
- **numCopieTotali : int**
- **numCopieDisponibili : int**

- **metodi:**

- **Libro(String titolo, ArrayList<Autore> autori, int anno, String codiceISBN, int numCopieTotali, int numCopieDisponibili)**
- **getters**
- **setters**
- **aggiungiAutore(Autore a)**
- **rimuoviAutore(Autore a)**
- **isDisponibile(): boolean**
- **isLibroInPrestito(): boolean**
- **aumentaCopie()**
- **diminuisciCopie()**
- **toString() : String**
- **autoriToString() : String**
- **equals(Object o) : boolean**
- **hashCode() : int**

## ● **Utente**

La classe Utente memorizza le informazioni relative all'utente (Nome, Cognome, Matricola, E-mail) . Per quanto riguarda le relazioni, ***la classe Utente è associata con la classe Prestito (un utente è soggetto di 0...\* prestiti)***, questo significa che un oggetto Utente è il soggetto del prestito nella classe Prestito. Possiede, inoltre, i seguenti metodi e attributi :

- **attributi:**
  - **nome : String**
  - **cognome : String**
  - **matricola : String**
  - **email: String**
  - **numPrestitiAttivi : int**
- **metodi:**
  - **Utente(String nome, String cognome, int matricola, String email, int numPrestitiAttivi)**
  - **getters**
  - **setters**
  - **limitePrestiti() : boolean**
  - **inPrestito() : boolean**
  - **incrementaPrestitiAttivi()**
  - **decrementaPrestitiAttivi()**
  - **toString() : String**
  - **equals(Object o) : boolean**

- hashCode() : int

- **Prestito**

La classe Prestito, rappresenta l'associazione di un Utente con un Libro per un periodo di tempo limitato. Memorizza quindi tutte le informazioni relative all'Utente, ai Libri e la relativa data di restituzione. Per quanto riguarda le relazioni, **la classe Prestito è una composizione della classe Utente e Libro**, in quanto contiene un riferimento esplicito ad un oggetto Utente e un oggetto Libro. Possiede, inoltre, i seguenti metodi e attributi :

- **attributi:**

- utente : Utente
    - libro : Libro
    - dataRestituzione : LocalDate

- **metodi:**

- Prestito(Utente utente, Libro libro, LocalDate dataRestituzione)
    - getters
    - setters
    - controllaRitardo() : boolean
    - toString() :String
    - equals(Object o) : boolean
    - hashCode() : int

- **Clienti**

La classe Clienti è la classe che gestisce una collezione di oggetti Utente, infatti mantiene l'elenco di tutti gli utenti registrati alla biblioteca, e fornisce metodi per operare sui vari Utenti. Per quanto riguarda le relazioni, **la classe Clienti è una composizione della classe Utente**, in quanto aggrega una lista di oggetti Utenti, e viene anche **usata da Biblioteca**. Possiede, inoltre, i seguenti metodi e attributi :

- **attributi:**

- clienti : ArrayList<Utente>

- **metodi:**

- Clienti()
    - getClienti() : ArrayList<Utente>
    - aggiungiUtente(Utente u)
    - eliminaUtente(Utente u)

- **modificaUtente**(Utente u, String nome, String cognome, String matricola, String email)
- **esisteUtente**(Utente u): boolean

## ● Libreria

La classe Libreria è la classe che gestisce una collezione di oggetti Libro, infatti mantiene il catalogo di tutti i libri della biblioteca, e fornisce metodi per operare sui vari Libri e per controllare la disponibilità delle copie in archivio. Per quanto riguarda le relazioni, **la classe Libreria è una composizione della classe Libro**, in quanto aggrega una lista di oggetti Libro, e anche questa viene **usata da Biblioteca**. Possiede, inoltre, i seguenti metodi e attributi:

- **attributi:**
  - **libreria** : ArrayList<Libro>
- **metodi:**
  - **Libreria()**
  - **getLibreria()** : ArrayList<Libro>
  - **aggiungiLibro**(Libro l)
  - **eliminaLibro**(Libro l)
  - **modificaLibro**(Libro l, String titolo, List<Autore> autori, int anno, String ISBN, int copieTot, int copieDisp)
  - **isInLibreria**(Libro l): boolean

## ● Prestiti

La classe Prestiti è la classe che gestisce una collezione di oggetti Prestito, infatti mantiene il catalogo di tutti i prestiti attivi, e fornisce metodi per operare sui vari Prestiti e per controllare la lista dei prestiti attivi. Per quanto riguarda le relazioni, **la classe Prestiti è una composizione della classe Prestito**, in quanto aggrega una lista di oggetti Prestito, e anche questa viene **usata da Biblioteca**. Possiede, inoltre, i seguenti metodi e attributi:

- **attributi:**
  - **prestiti** : ArrayList<Prestito>
- **metodi:**
  - **Prestiti()**
  - **getPrestiti()** : ArrayList <Prestito>
  - **aggiungiPrestito**(Prestito p)
  - **rimuoviPrestito**(Prestito p)

- **Biblioteca**

La classe Biblioteca rappresenta la giunzione tra il Model e l'interfaccia grafica, e in quanto tale, coordina tutte le operazioni che coinvolgono i vari gestori (Clienti, Prestiti, Libreria) e gestisce anche l'autenticazione del Bibliotecario e il salvataggio dei dati sul file esterno. Per quanto riguarda le relazioni, **la classe Biblioteca è un'aggregazione delle classi Clienti, Libreria e Prestiti in quanto presenta i loro riferimenti**. Viene **usata anche dalla classe Interfaccia Utente** per gestire tutte le richieste di sistema. Possiede, inoltre, i seguenti metodi e attributi:

- **attributi**

- **libreria : Libreria**
    - **clienti : Clienti**
    - **prestiti : Prestiti**
    - **obLibreria : ObservableList<Libro>**
    - **obClienti : ObservableList<Utente>**
    - **obPrestiti : ObservableList<Prestito>**
    - **sLibreria: SortedList<Libro>**
    - **sClienti: SortedList<Utente>**
    - **flLibreria : FilteredList<Libro>**
    - **flClienti : FilteredList<Utente>**
    - **filename : final String**

- **metodi**

- **Biblioteca()**
    - **getters**
    - **inizializzaListe()**
    - **filtraLibri(String search)**
    - **filtraUtenti(String search)**
    - **checkValiditaCampiLibro(String titolo, List<Autore> autori, int anno, String ISBN, int copieTot, int copieDisp) : boolean**
    - **checkValiditaCampiUtente(String nome, String cognome, String matricola, String email, int numPrestitiAttivi) : boolean**
    - **aggiungiLibro(String titolo, List<Autore> autori, int anno, String ISBN, int copieTot, int copieDisp)**
    - **eliminaLibro(Libro l)**
    - **modificaLibro(Libro l, String titolo, List<Autore> autori, int anno, String ISBN, int copieTot, int copieDisp)**

- **aggiungiUtente(String nome, String cognome, String matricola, String email, int numPrestitiAttivi)**
- **eliminaUtente(Utente u)**
- **modificaUtente(Utente u, String nome, String cognome, String matricola, String email)**
- **aggiungiPrestito(Utente u, Libro l, LocalDate data)**
- **restituiscePrestito(Prestito p)**
- **salvaSuFile()**
- **caricaDaFile()**

- **MainController**

La classe MainController rappresenta il Controller, e in quanto tale, si occupa di gestire gli elementi grafici ed il file FXML che rappresentano graficamente tutta la gestione della biblioteca.

- **setUpController()**
- **initialize()**
- **configuraLibri()**
- **configuraUtenti()**
- **configuraPrestiti()**
- **mostraLibri()**
- **mostraUtenti()**
- **mostraPrestiti()**
- **cambiaScena(Pane paneToShow)**
- **nuovoLibro()**
- **nuovoUtente()**
- **nuovoPrestito()**
- **modificaLibro(Libro l)**
- **modificaUtente(Utente u)**
- **parseAutori(String str) : ArrayList<Autore>**
- **alertErrore(String msg)**

- **AccessoController**

La classe AccessoController rappresenta il Controller dell'interfaccia d'accesso che gestisce gli elementi grafici ed il file FXML che rappresentano la schermata d'accesso.

- **setupController()**
- **handleEntraNelSistema(ActionEvent event)**

- **Interfaccia Utente**

L'insieme di classi "Interfaccia Utente" rappresentano la View, e in quanto tale, si occupa dell'interazione diretta con l'attore Bibliotecario. E' la classe che gestisce quindi tutti gli elementi grafici della GUI , che riceve gli input dell'utente ... Per quanto riguarda le relazioni, ***la classe Interfaccia Utente è associata con la classe Biblioteca in quanto interagisce con essa per modificare dati e inviare le azioni di sistema.*** Possiede ,inoltre, le seguenti classi, con i seguenti metodi e attributi:

- **Main**: Si occupa di avviare l'applicazione
  - **start(Stage stage)**
  - **mostraSchermataAccesso()**
  - **mostraHomePage()**
  - **main(String[] args)**
- **FXML**

## Coesione e Accoppiamento

CLASSE	COESIONE	DESCRIZIONE
Autore	Funzionale	La classe Autore incapsula nome e cognome di un autore. Si è deciso di rappresentare con una classe questa informazione per una gestione migliore di alcune operazioni di filtro, ricerca e visualizzazione.
Libro	Funzionale	La classe Libro incapsula le informazioni necessarie ad ogni libro (titolo, autori, anno, ISBN, copie, metodi per verificarne disponibilità e aggiornamento).
Utente	Funzionale	La classe Utente incapsula le informazioni necessarie ad ogni utente (identità e stato prestiti). Si è scelto di non inserire la lista dei prestiti in Utente per non creare inconsistenze tra strutture dati o duplicazioni.
Prestito	Funzionale	La classe Prestito incapsula le informazioni necessarie all'associazione dei prestiti. Si include in questa classe un metodo per la ricerca dei prestiti associati all'utente.
Libreria	Funzionale	La classe Libreria include funzionalità che svolgono il singolo compito della gestione dei libri.
Clienti	Funzionale	La classe Utenti include funzionalità che svolgono il singolo compito della gestione degli utenti.
Prestiti	Funzionale	La classe Prestiti include funzionalità che svolgono il singolo compito della gestione dei prestiti.
Biblioteca	Procedurale	È la classe principale che funge da gestore della comunicazione tra le altre classi, l'interfaccia utente ed il file esterno; per cui, il livello di coesione non risulta funzionale.
MainController	Funzionale	È la classe che si occupa di gestire tutti gli elementi grafici dell'interfaccia ed include funzionalità che svolgono solamente questo ruolo, senza eccedere in altri ricoperti da altre classi.
AccessoController	Funzionale	È la classe che si occupa di gestire gli elementi grafici presenti nella schermata d'accesso ed include funzionalità che svolgono solamente questo ruolo.

### **Livello di accoppiamento: Per dati/Per timbro.**

Il livello di accoppiamento generale è abbastanza buono. La maggior parte delle classi scambia solamente le informazioni necessarie al funzionamento di un'altra classe. Questo non avviene nella classe "Prestito": gli attributi "libro" e "utente" racchiudono tutte le informazioni relative ad essi, comprendendo anche informazioni che la classe

“Prestito” non necessita, portando così il livello di accoppiamento da “per dati” a “per timbro”.

## Principi di buona progettazione

In termini di semplicità di progettazione (KISS), di realizzabilità (SINE) e di ripetizione del codice (DRY) il progetto è generalmente ben costruito: si è cercato di suddividere il più possibile i compiti delle classi, dando ad ognuna di esse un singolo compito (Separation of Concerns), evitando di creare delle “God Class”. Nello specifico, la classe Autore è stata creata per evitare errori di parsing e inconsistenze quando si lavorerà con il filtraggio degli Autori. Gestire una Collection della classe Autore risulta molto più semplice.

Si è cercato inoltre di limitare al minimo le funzionalità superflue (YAGNI), di tenere collegati solo i moduli Libri-Autore (Ortogonalità) e di evitare di utilizzare l’ereditarietà quando la composizione era sufficiente.

Il codice è accuratamente formattato: si è deciso di utilizzare il camelCase, di usare l’italiano per i metodi e i commenti e di mantenere sempre lo stesso numero di righe vuote tra blocchi di codice non strettamente correlati. Il codice è ampiamente commentato, sia con commenti classici che con commenti per Doxygen, rispettando il principio della minima sorpresa.

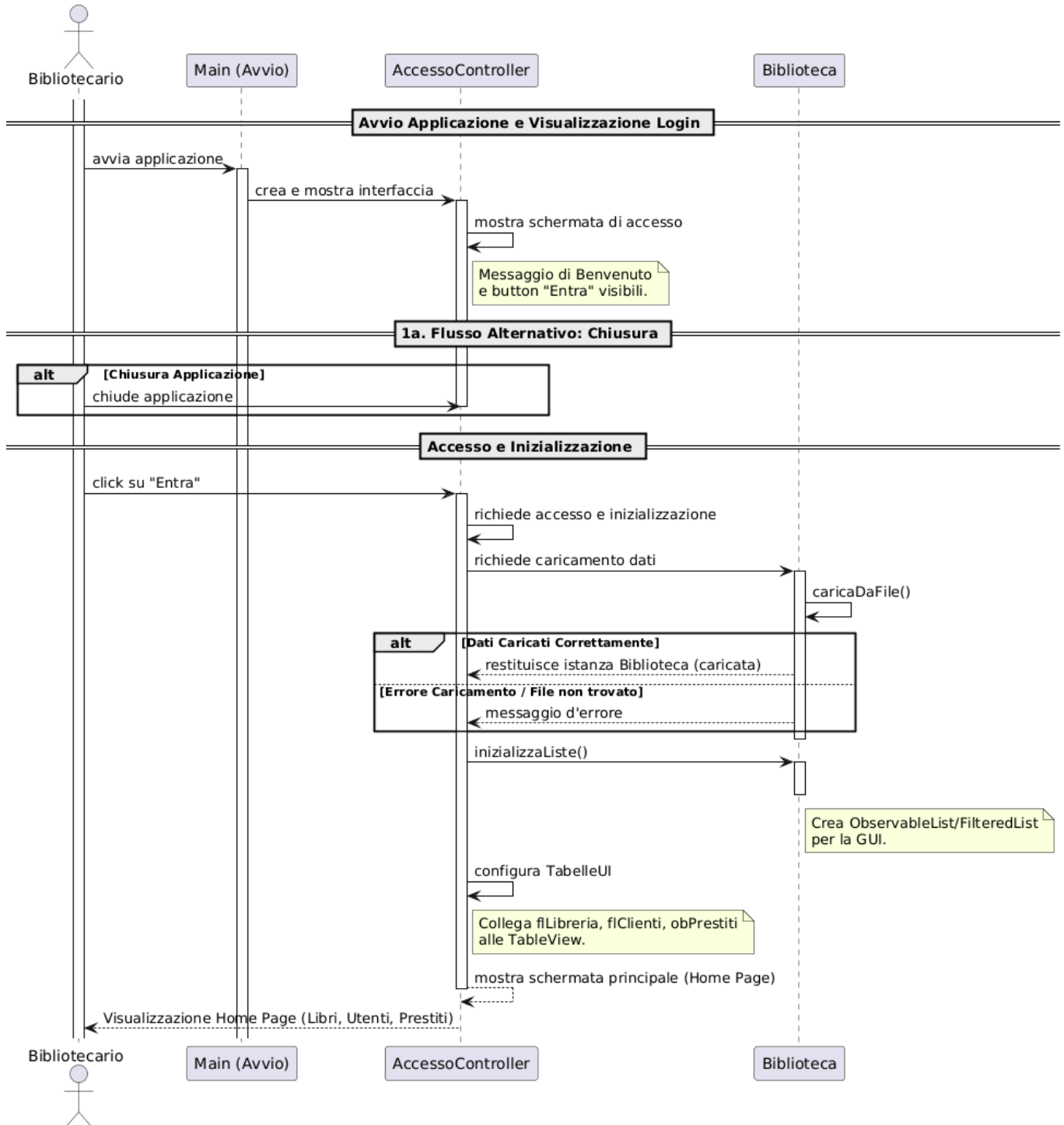
## Principi SOLID

- **Single responsibility:** ogni modulo ha un solo scopo ben definito.
- **Open-closed:** l’incapsulamento inserito in ogni classe (attributi private, metodi getter e setter, se necessari) garantisce la chiusura alle modifiche, ma lascia spazio all’estensione.
- **Liskov substitution:** non avendo ereditarietà, questo principio non può essere violato.
- **Interface segregation:** si è scelto di non definire interfacce. Non ci sono metodi così simili da poterne usare senza compromettere il principio KISS.
- **Dependency inversion:** la non definizione di interfacce comporta anche la comunicazione diretta tra moduli di alto e basso livello.

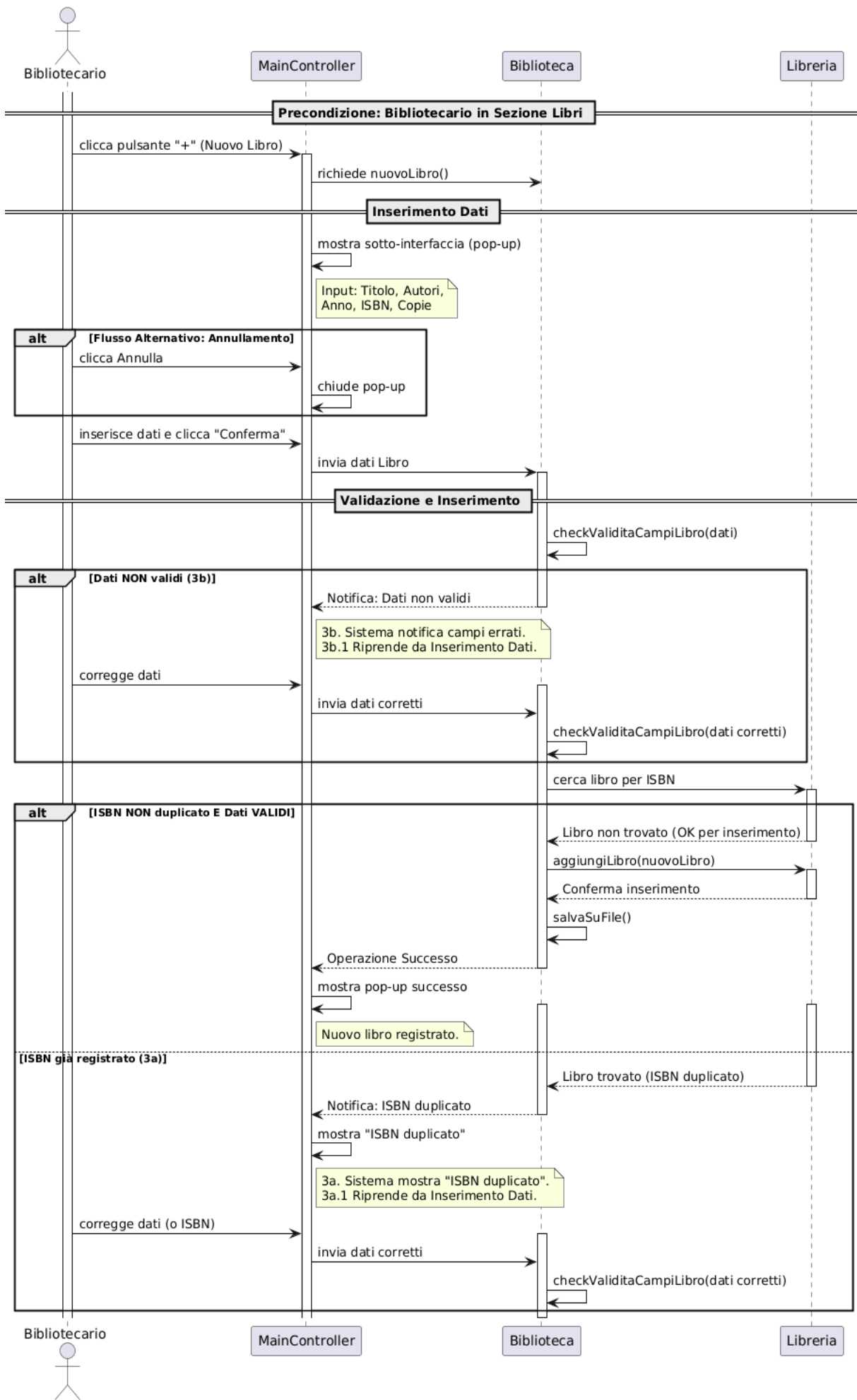
Generalmente, il rispetto dei principi SOLID è buono.

# Diagrammi di Sequenza

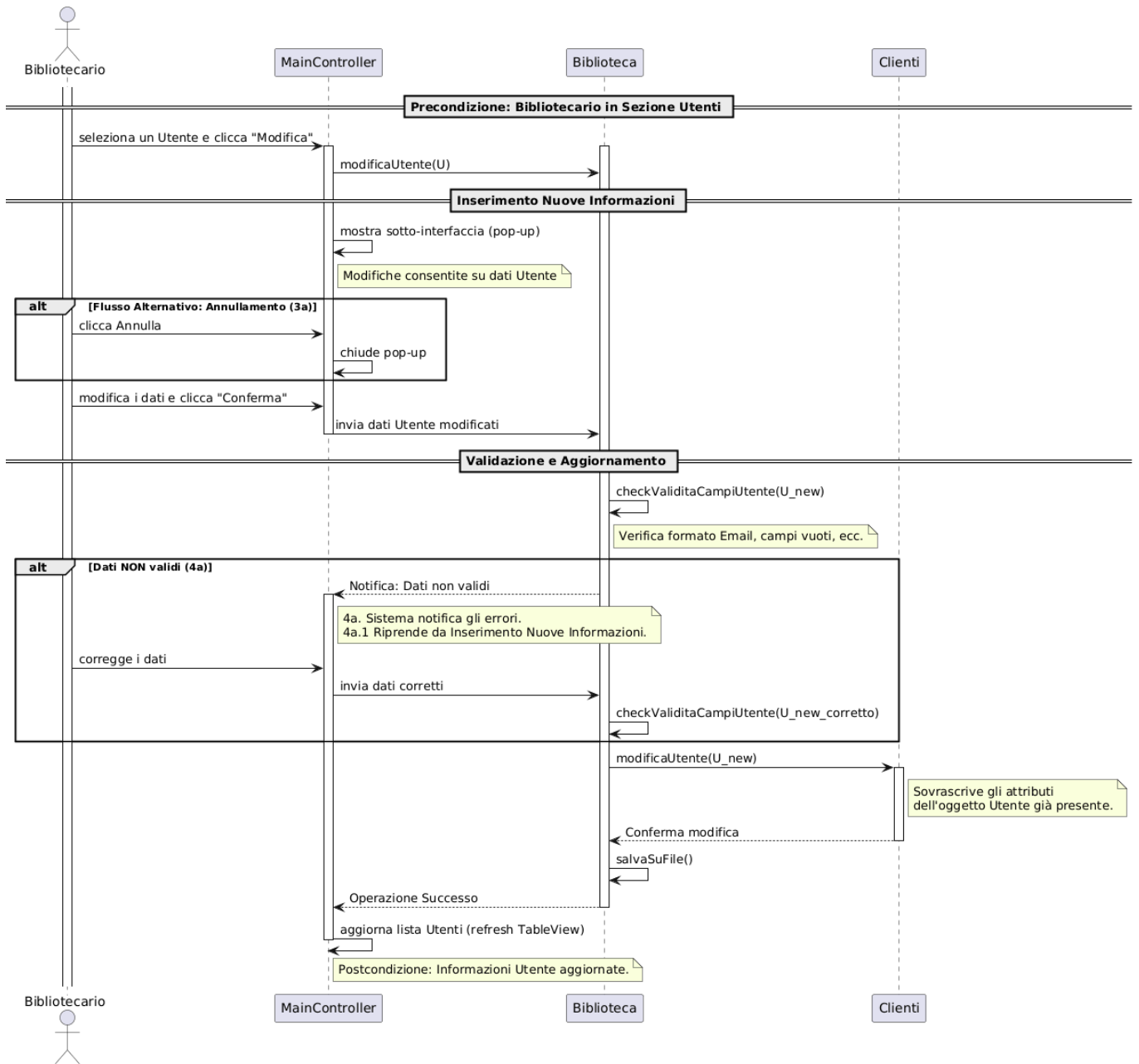
## UC-1: Accesso all'Applicazione



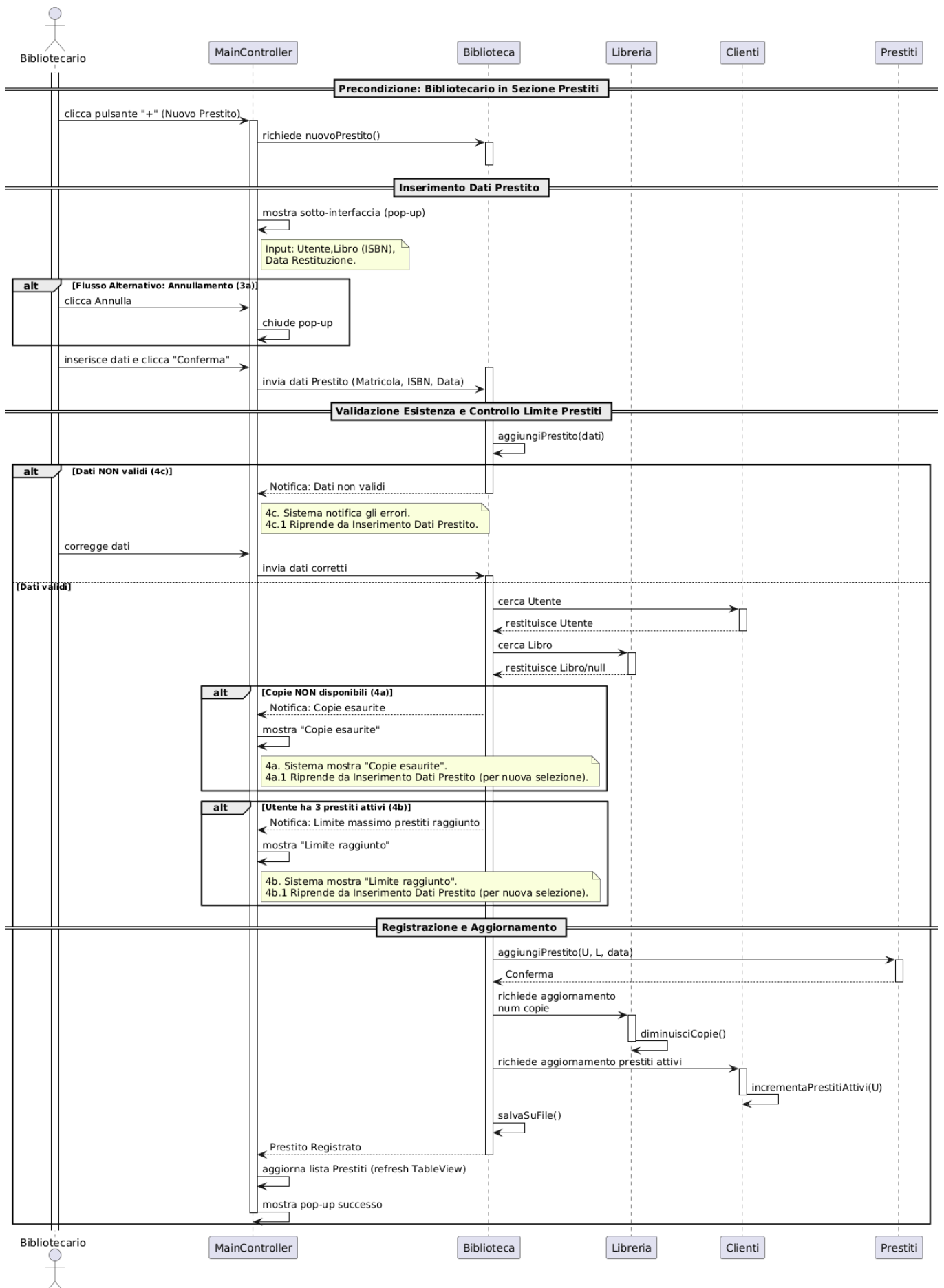
## UC-8: Inserimento di un Libro



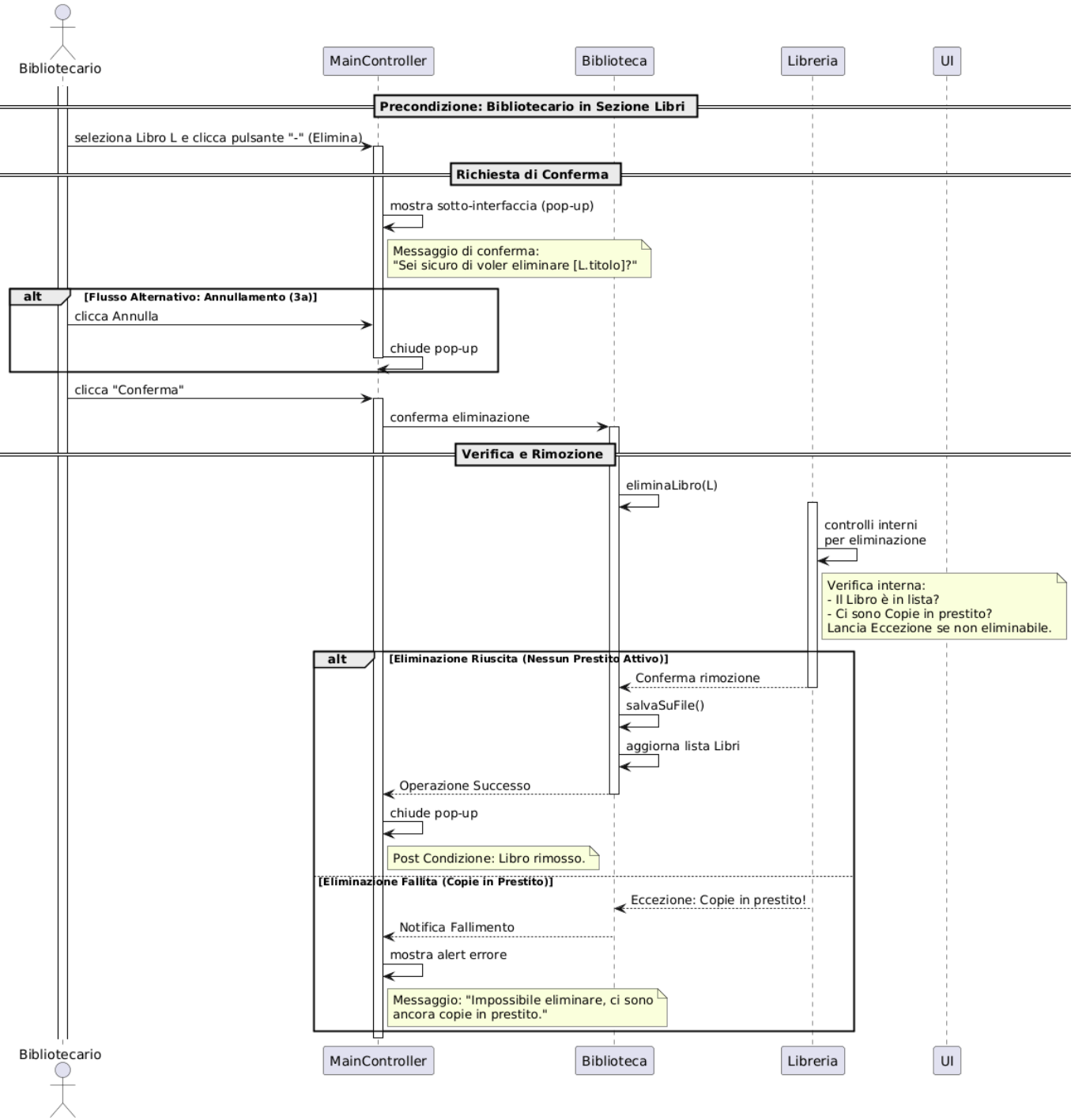
# UC-12: Modifica di un Utente



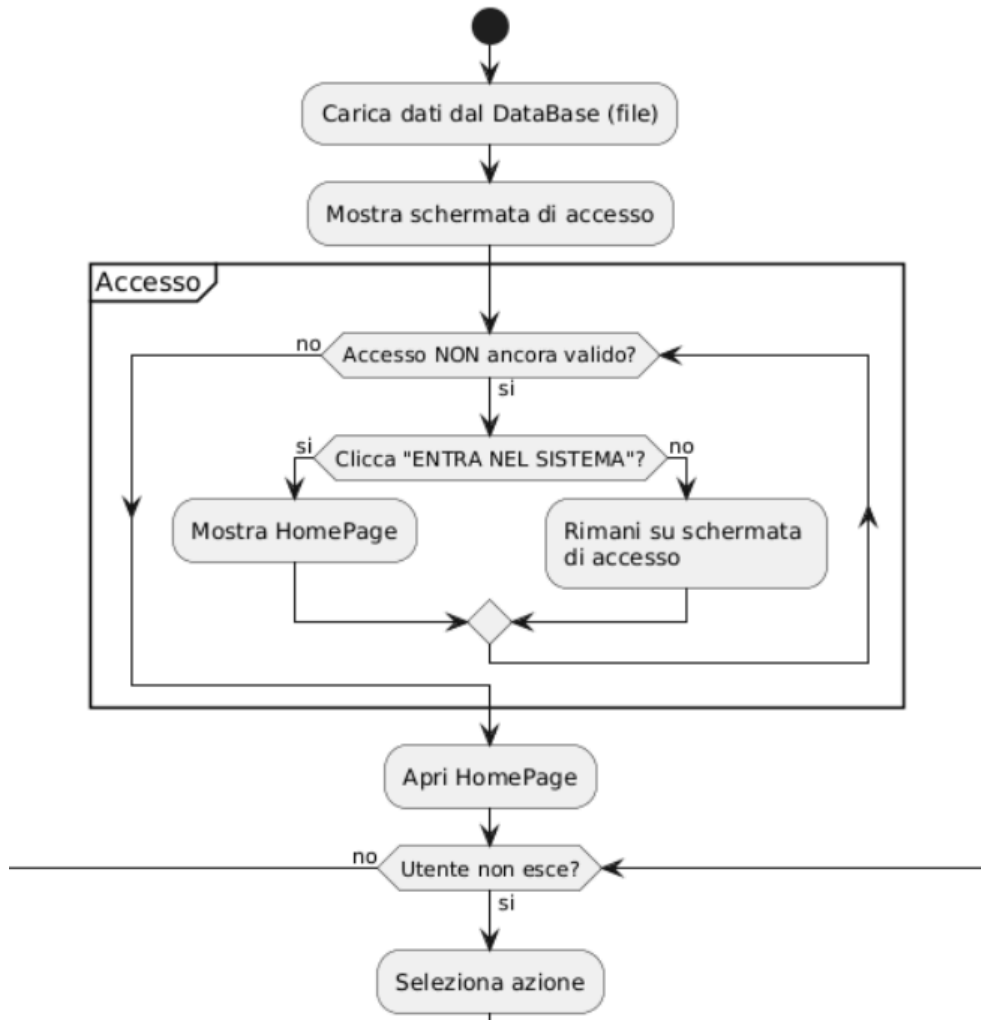
# UC-14: Registrazione di un Prestito

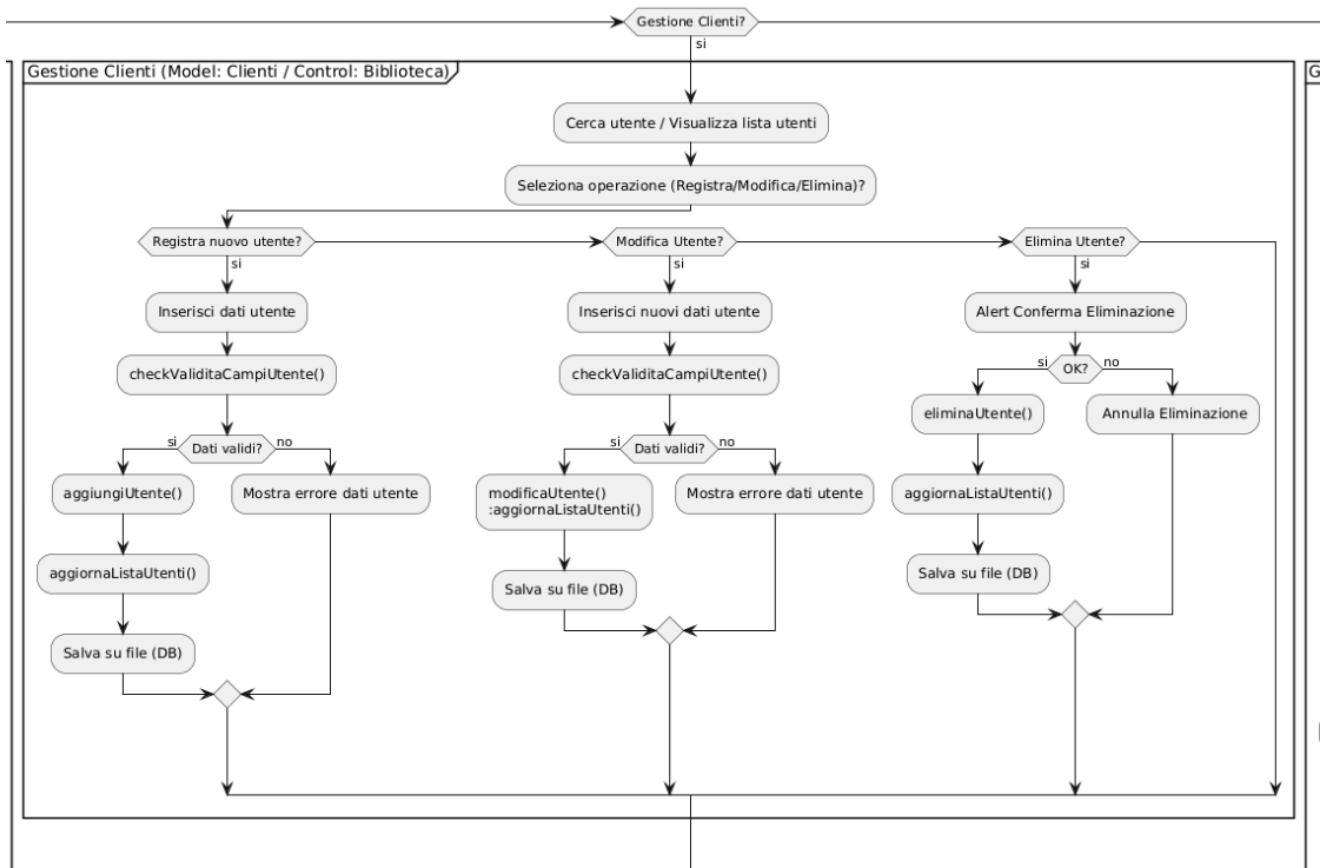
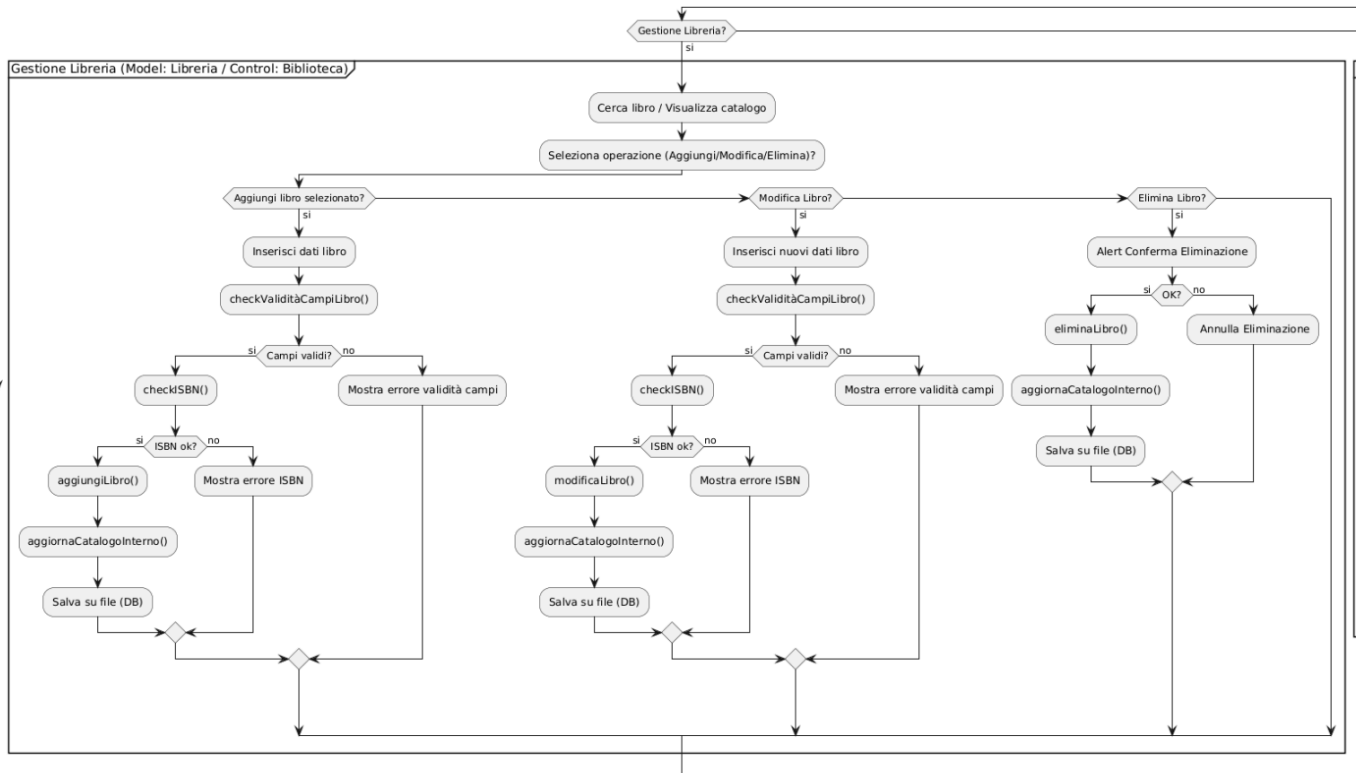


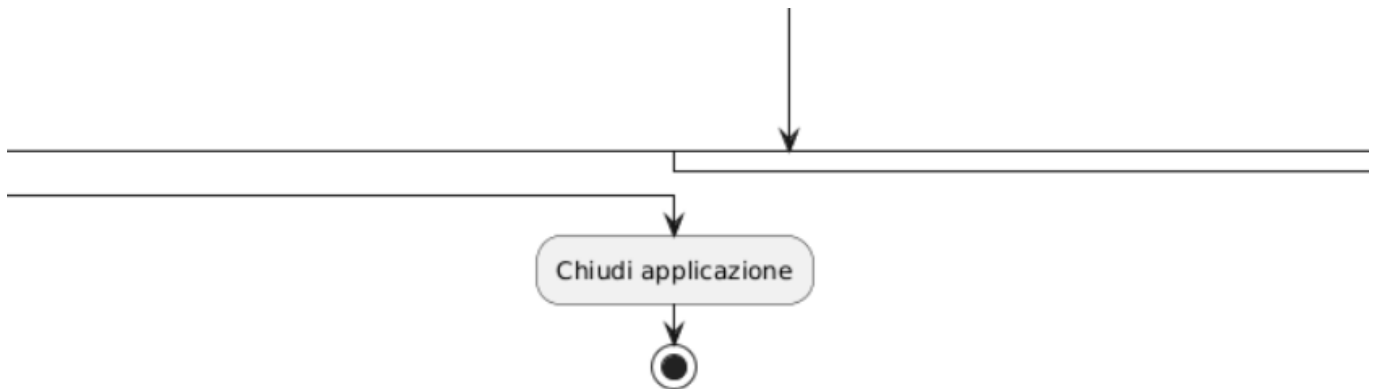
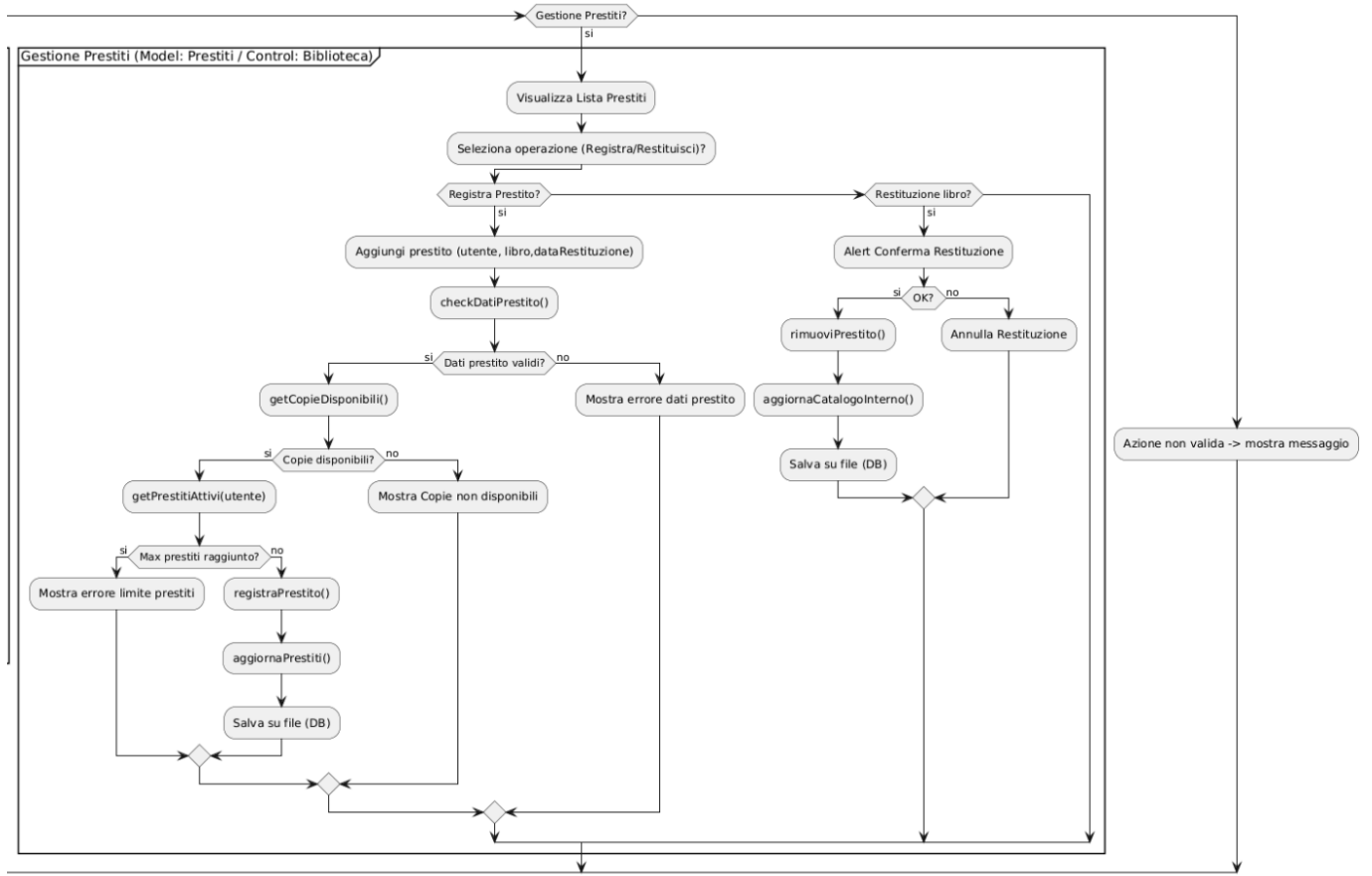
UC-10: Eliminazione di un Libro



# Diagramma delle attività

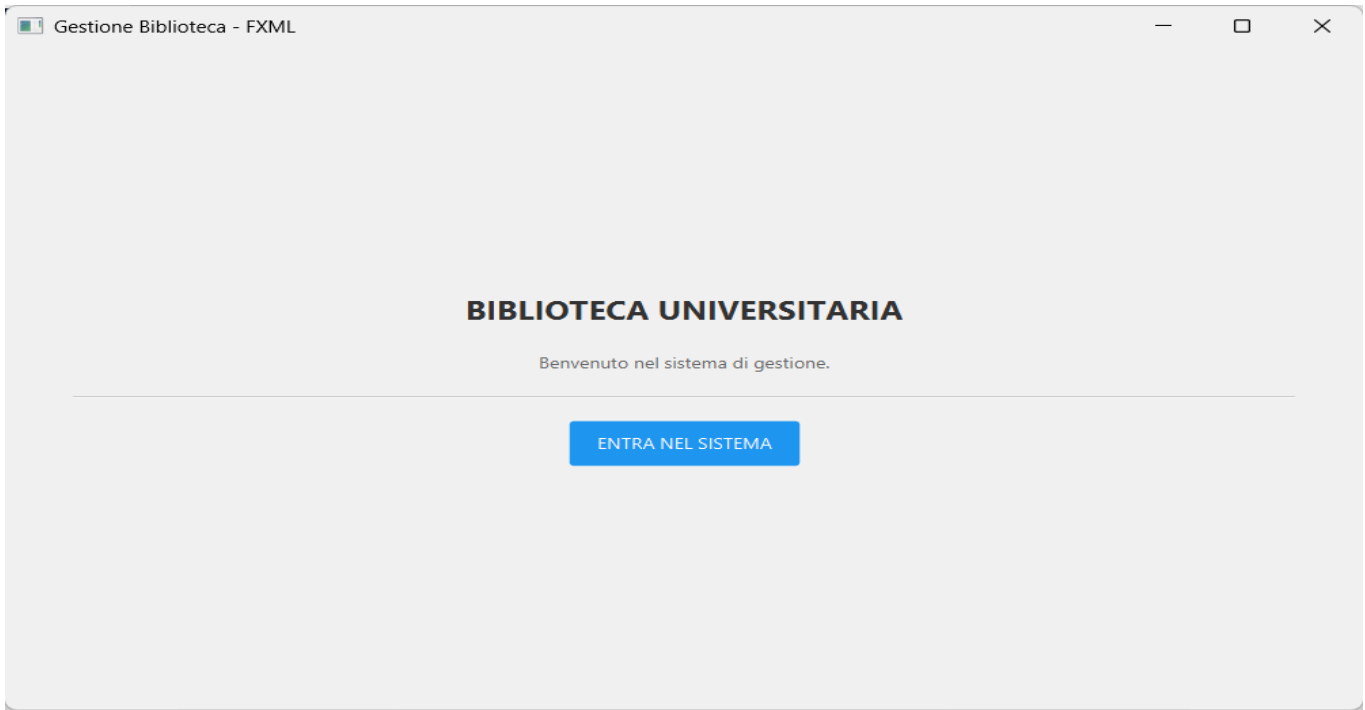




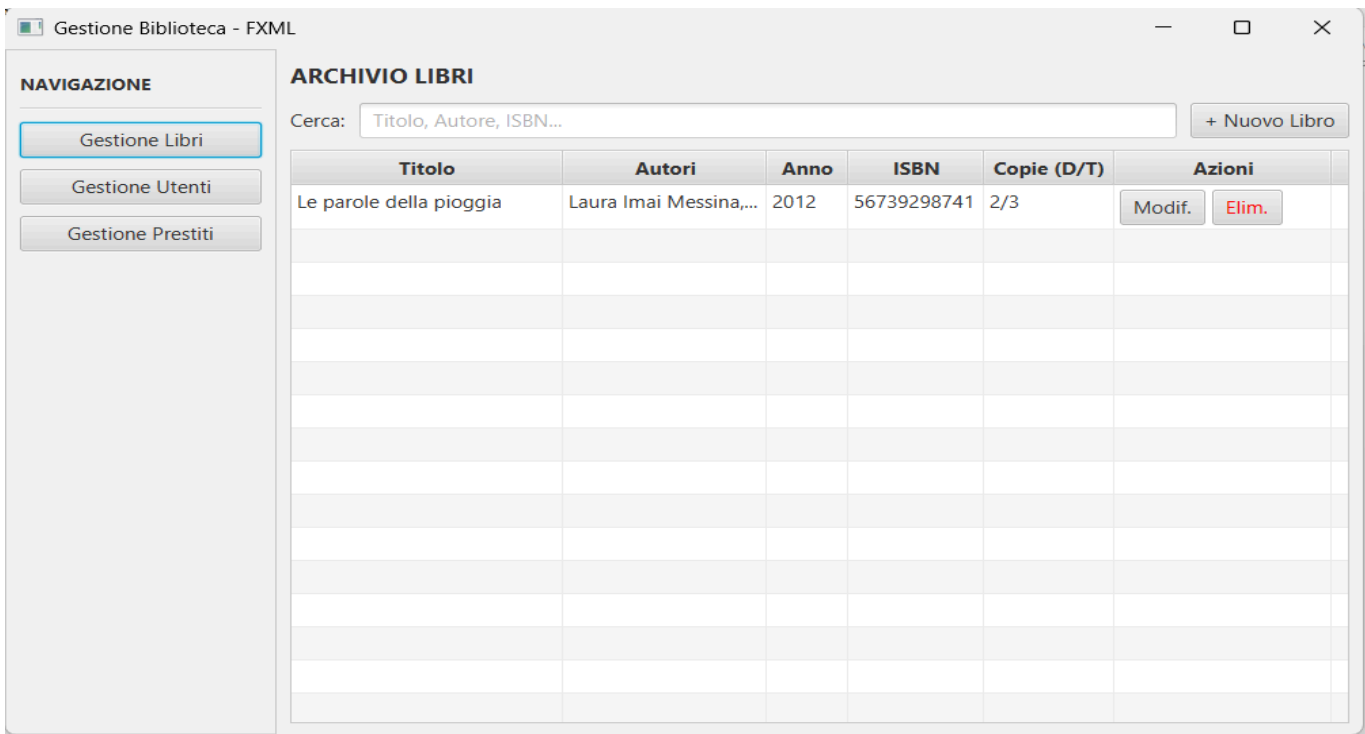


# Mock-up delle interfacce

**Accesso:** Cliccando il pulsante “Entra nel sistema”, l’utente potrà accedere alle schermate successive.



**Libri:** Mediante la seguente interfaccia, l’utente può aggiungere un nuovo libro, modificare o eliminare quelli presenti, ricercare un libro in base a: titolo, autori e codice ISBN, e cambiare pagina, accedendo ad Utenti o Prestiti.



**Utenti:** Mediante la seguente interfaccia, l'utilizzatore può registrare un nuovo utente, modificare o eliminare quelli presenti, ricercare in base a: nome, cognome e numero di matricola, e cambiare pagina, accedendo ad Libri o Prestiti.

[illegible]

**Prestiti:** L'utente può registrare un nuovo prestito selezionando utente, libro e data di restituzione e cancellare un prestito di cui è stata effettuata una restituzione.

[illegible]