EasyLibrary

Progettazione di Dettaglio del Software

Gruppo 18
Francesco Cangianiello
Andrea Ciliberti
Serena Giannitti
Marco Giraulo

11 dicembre 2025

Indice

1 Introduzione

1.1 Scopo del documento

Lo scopo di questo di questo documento è di descrivere la progettazione del livello di dettaglio del il sistema software EasyLibrary. Il documento rappresenta la mappa per l'implementazione del sistema utilizzata dagli sviluppatori durante il processo di implementazione. Nel documento sono presenti descrizioni per i principali componenti del sistema EasyLibrary e esposizioni delle scelte di progettazione che hanno portato alla loro formalizzazione. I diagrammi di sequenza sono utilizzati per dimostrare il funzionamento dinamico del sistema e per descriverne con precisione i meccanismi di implementazione agli sviluppatori.

1.2 Processo di Progettazione

Completata la fase di analisi, è stato innanzitutto disegnato un diagramma UML generale delle classi con cui astrarre le caratteristiche del sistema per requisiti. Successivamente, al fine di migliorare la struttura del sistema secondo gli attributi di qualità studiati, sono state compiute scelte di design facendo riferimento alle capacità delle tecnologie di sviluppo (linguaggio Java). Sfruttando appieno la flessibilità dell'incapsulamento, dell'ereditarietà e dei tipi generici, si è arrivati ad una struttura del sistema maggiormente stabile e più facilmente manutenibile e testabile. A tal punto si è potuto passare all finalizzazione dei contratti delle diverse classi e dei loro metodi associati. Infine, i casi d'uso del sistema sono stati passati in rassegna e per ognuno è stato individuato un flusso di sequenza delle operazioni, da cui sono stati dedotti i diagrammi di sequenza e di attività. Durante il processo di disegno di questi diagrammi sono stati riconosciuti e risolti alcuni problemi di design, cosa che ha portato a migliorare in modo iterativo la progettazione del sistema.

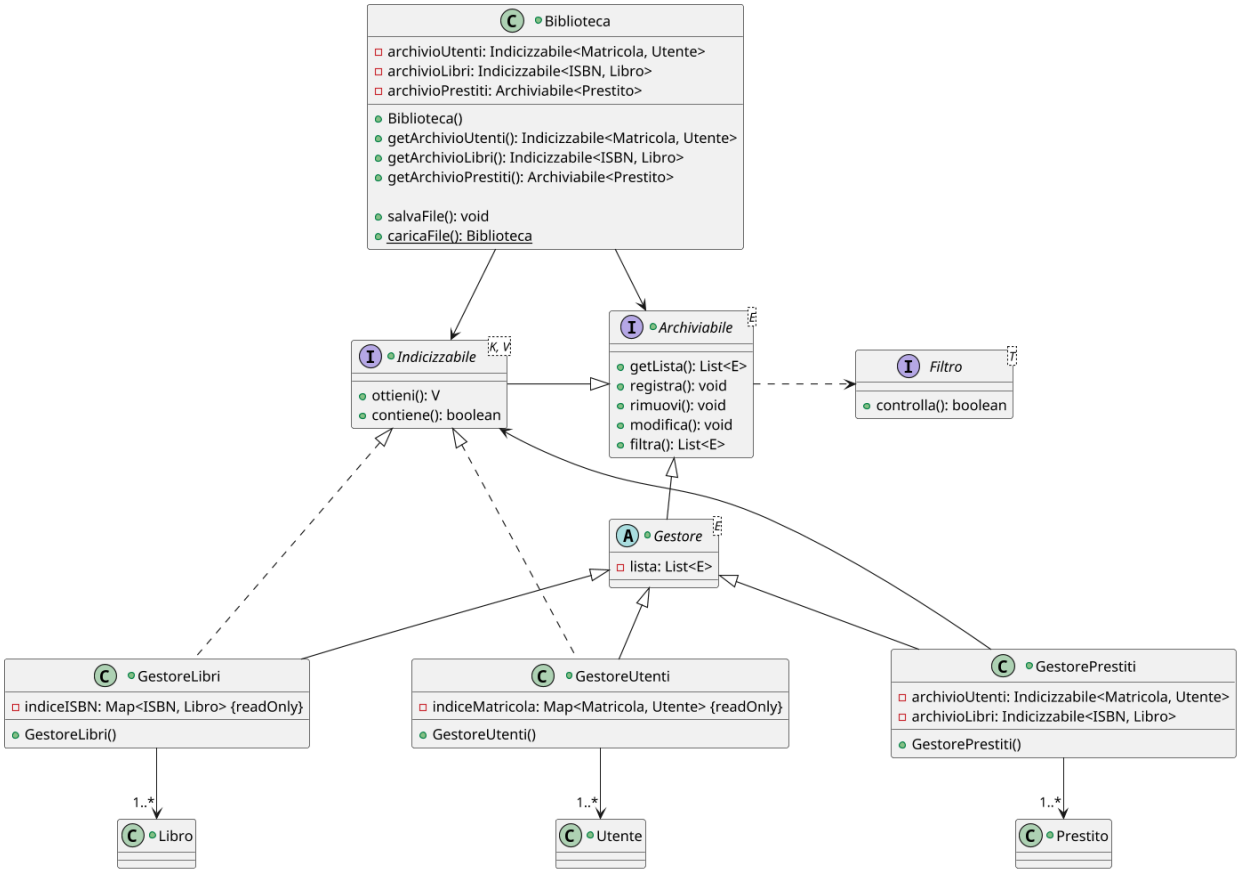
2 Progettazione di Dettaglio delle Classi

Il sistema software è stato scomposto in 3 pacchetti funzionali:

- Un pacchetto responsabile della gestione delle strutture dati che compongono l'archivio della biblioteca.
- Un pacchetto responsabile della rappresentazione dei dati elementari dell'archivio della biblioteca.
- Un pacchetto responsabile della visualizzazione dell'interfaccia grafica del sistema.

Oltre alle classi descritte di seguente, sono inoltre da implementare i segnalatori di errore (*RuntimeException*) utilizzati per segnalare errori in casistiche in cui le precondizioni dei contratti delle funzioni non sono rispettate. Non sono incluse nei diagrammi poichè la loro presenza e funzione non è vincolante per la progettazione del sistema, ma la loro presenza è indicata nei diagrammi di sequenza dove i flussi di errore sono correttamente gestiti.

2.1 Gestori di Archivio



2.1.1 Scelte progettuali

Come mostrato nel diagramma delle classi, la distribuzione delle responsabilità funzionali dell'applicativo segue una struttura ad albero. Tale organizzazione è stata progettata per ridurre le interdipendenze tra le varie sezioni del sistema, in modo da favorire l'*ortogonalità*, secondo il principio della *separazione delle preoccupazioni*, in modo da migliorare la coesione dei singoli componenti software. La radice dell'albero è la classe *Biblioteca*, composta da diversi archivi di dati che può orchestrare per compiere azioni complesse o globali.

Dai requisiti analizzati si è potuto riconoscere un insieme di funzionalità comuni alle 3 principali collezioni di dati (archivio di utenti, libri e prestiti). Le funzionalità comuni sono state estratte in un interfaccia, *Archiviabile*,

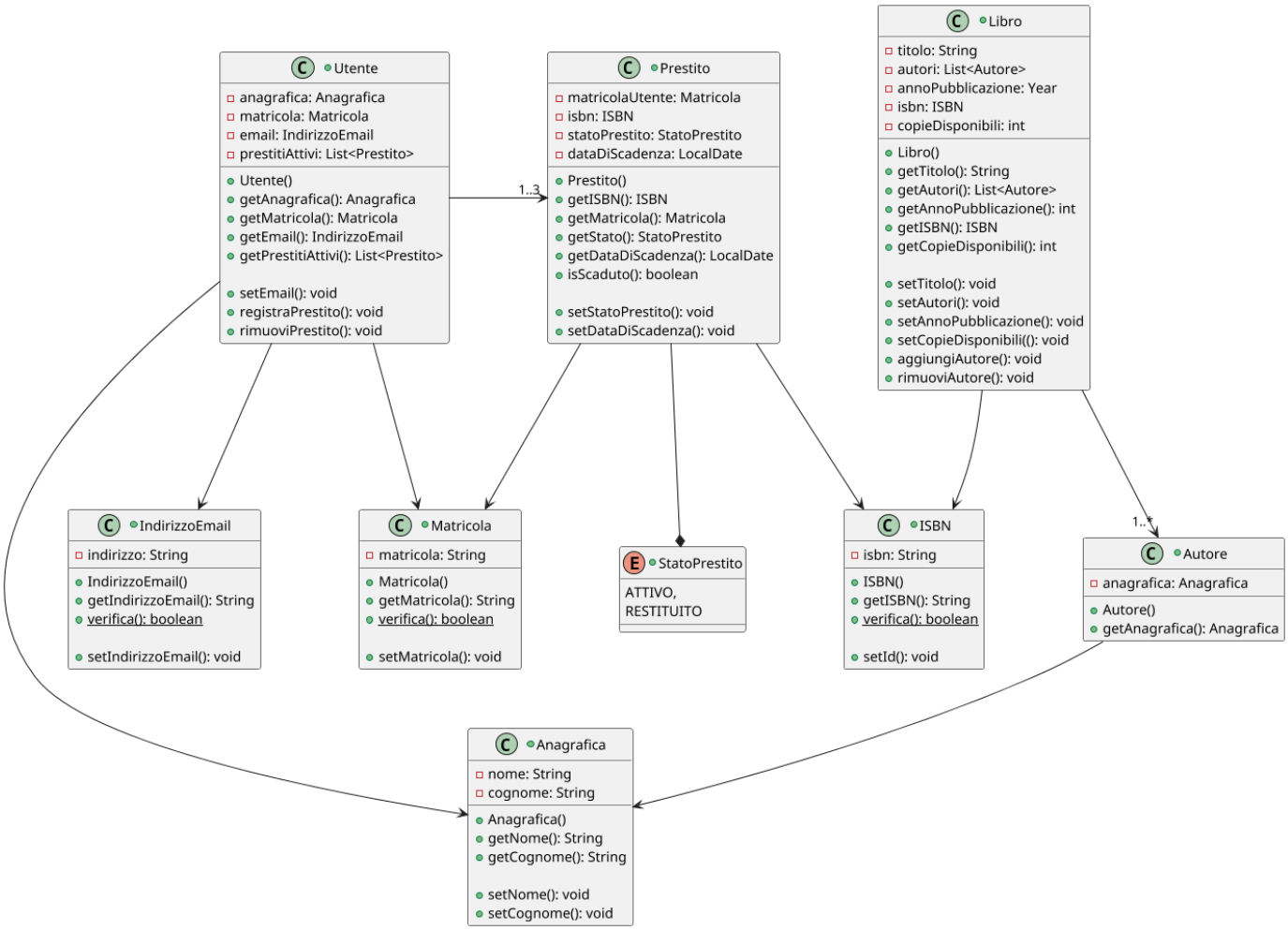
che permette la registrazione, rimozione e modifica di dati generici. Per gli archivi con indice univoco (utenti e libri), è inoltre disponibile un ulteriore interfaccia *Indicizzabile*, estensione di *Archiviabile*, che permette l'ottenimento diretto di un valore tramite chiave. Questa impostazione consente di applicare il *principio di inversione delle dipendenze*, rendendo *Biblioteca* dipendente da astrazioni piuttosto che da implementazioni concrete. Le classi che realizzano gli archivi possono così essere sostituite o estese senza impattare sul livello superiore.

Le funzionalità comuni ai vari gestori sono state consolidate in una singola classe astratta, *Gestore*, al fine di applicare il principio *DRY* (*Don't Repeat Yourself*). Le tre specializzazioni ereditano tale implementazione e definiscono solo il comportamento specifico, riducendo la complessità del codice secondo il principio *KISS* (*Keep It Simple*).

L'unica classe di gestione che richiede più informazioni della semplice lista di dati è *GestorePrestiti*. Per compiere i controlli di validità, è necessario che il gestore dei prestiti abbia disponibile le informazioni di contesto sugli utenti e sui libri presenti nel sistema. Nonostante ciò, anzichè creare dipendenza tra le varie classi, si è deciso di associare a *GestorePrestiti* riferimenti all'interfaccia *Indicizzabile*, che possiede tutti i servizi necessari per ottenere lo stato degli utenti data la *Matricola* (e dei libri, dato l'*ISBN*). Ciò permette di mantenere un ottima *ortogonalità* tra i diversi archivi.

Infine, un rigoroso uso dell'incapsulamento ha permesso di minimizzare l'accoppiamento tra le classi e di nascondere i dettagli implementativi, aumentando la robustezza e l'estensibilità del sistema.

2.2 Rappresentazione Dati



2.2.1 Scelte progettuali

Le scelte di progettazione compiute nell'ambito dati sono state pensate al fine di aumentare il livello di coesione e abbassare il livello di accoppiamento tra le diverse strutture dati. Ogni classe raggruppa attributi e metodi strettamente correlati al proprio scopo, aderendo al principio di *separazione delle preoccupazioni*.

Al fine di evitare *accoppiamento per timbro*, anzichè mantenere riferimenti ad altri oggetti dati in modo diretto, si è preferito (ove possibile) mantenere riferimenti a chiavi univoche, come nel caso di *Prestito*, che

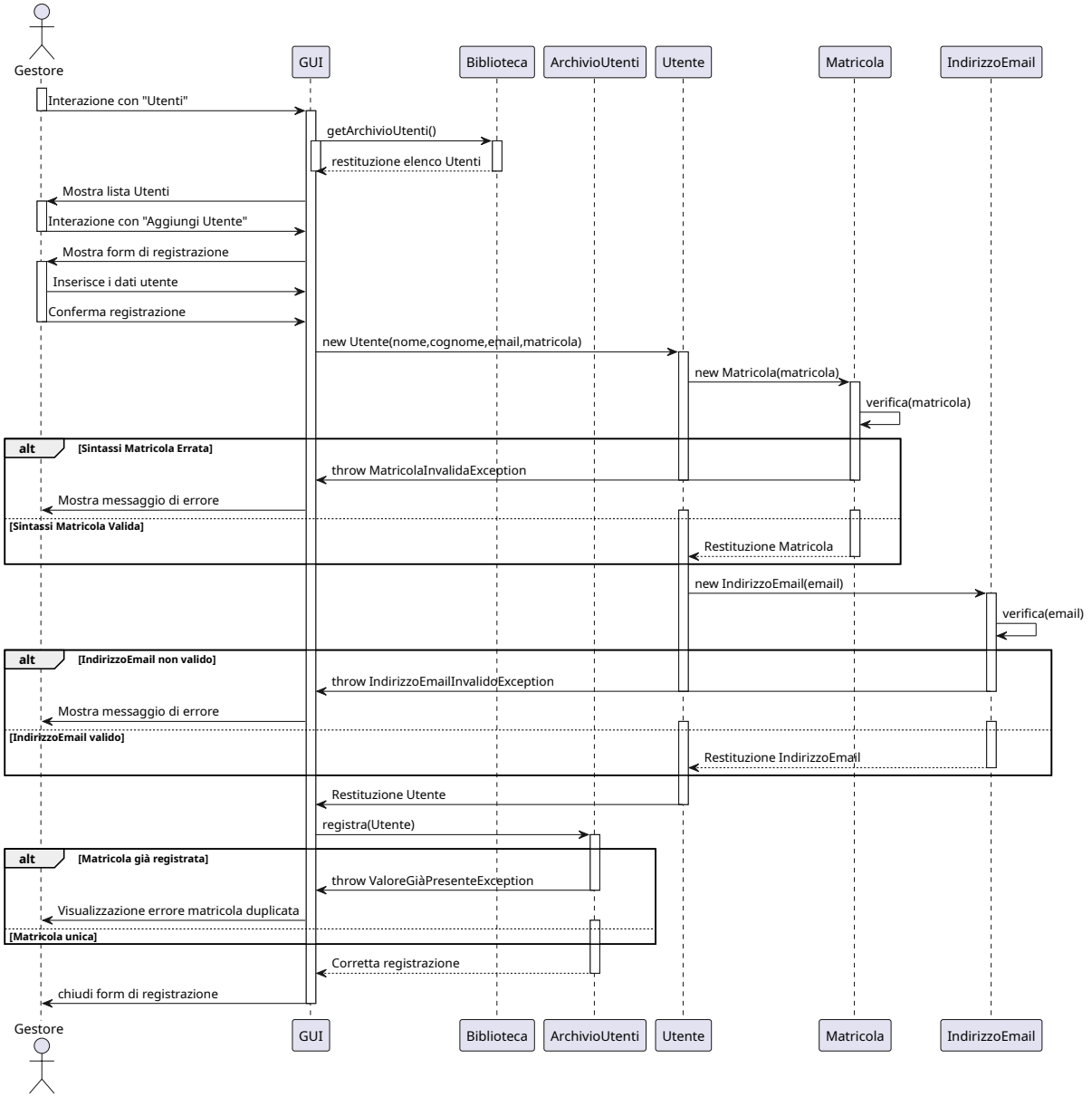
contiene riferimenti a *Matricola* e ad *ISBN*, anzichè ad *Utente* e a *Libro*. L'unico caso in cui ciò non è stato possibile è l'associazione da *Utente* a *Prestito*, poichè non è previsto un identificativo univoco per *Prestito*.

Si è fatto uso rigoroso di incapsulamento per rendere accessibile il comportamento delle classi solo tramite metodi, al fine di evitare *coesione per contenuti*.

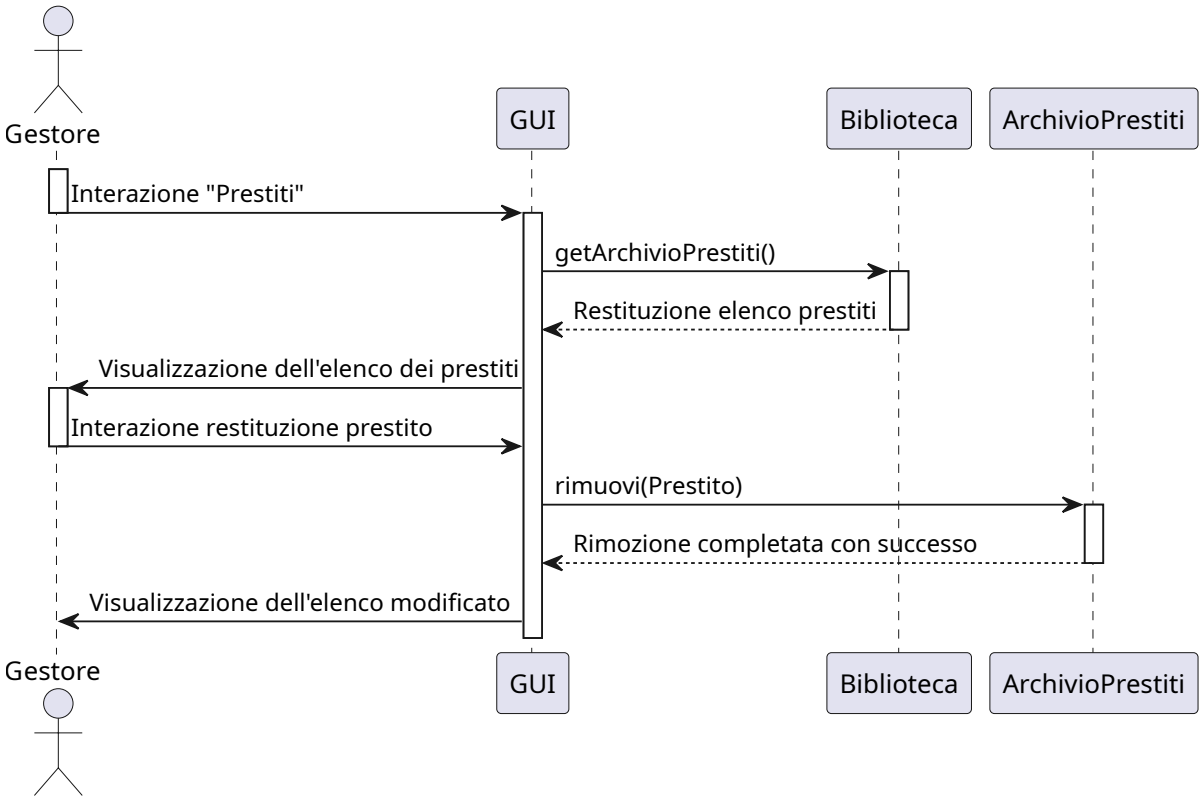
Ogni classe espone solo funzionalità strettamente associate alla propria funzione, raggiungendo un'alta *coesione funzionale*.

3 Diagrammi di Sequenza

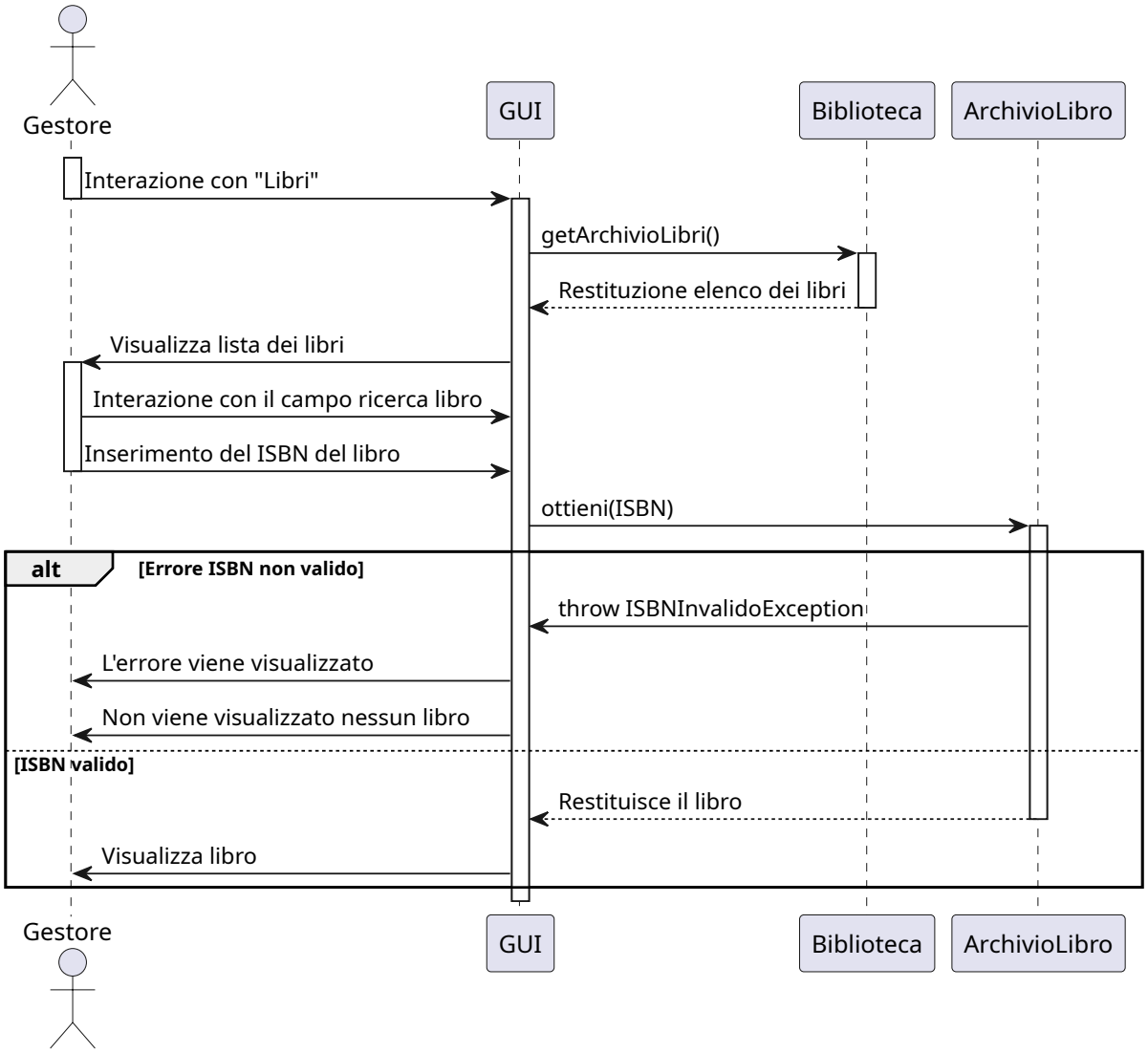
3.1 UC-6 – Registrazione di un utente



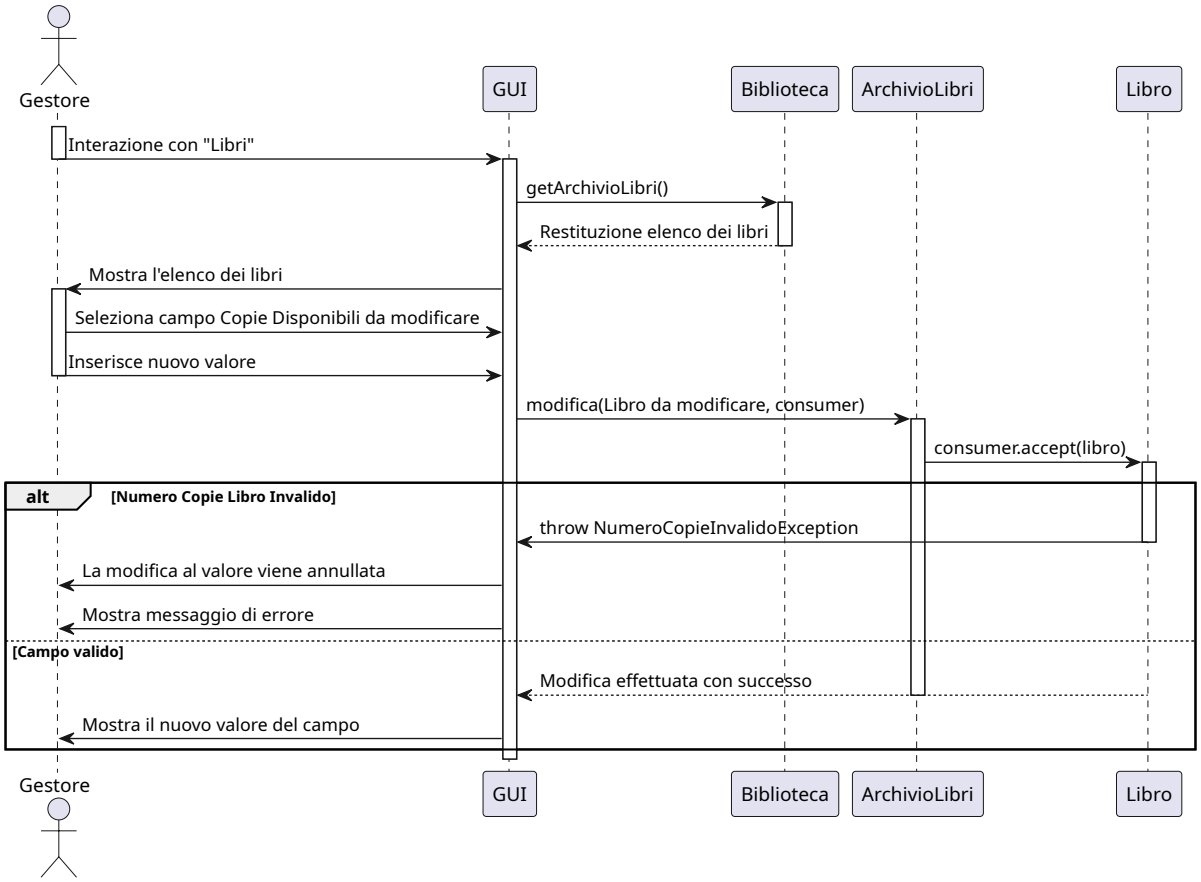
3.2 UC-11 – Restituzione di un prestito



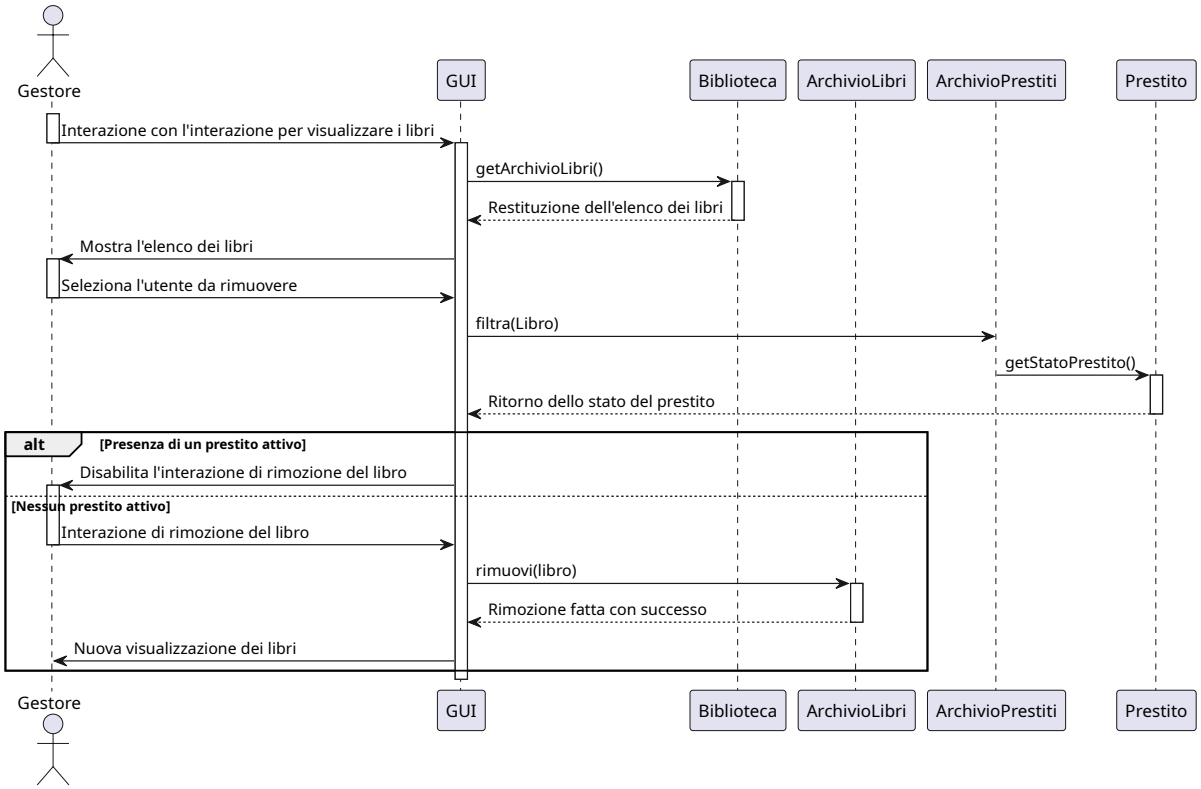
3.3 UC-12 – Ricerca di libri



3.4 UC-14 – Modifica di un libro

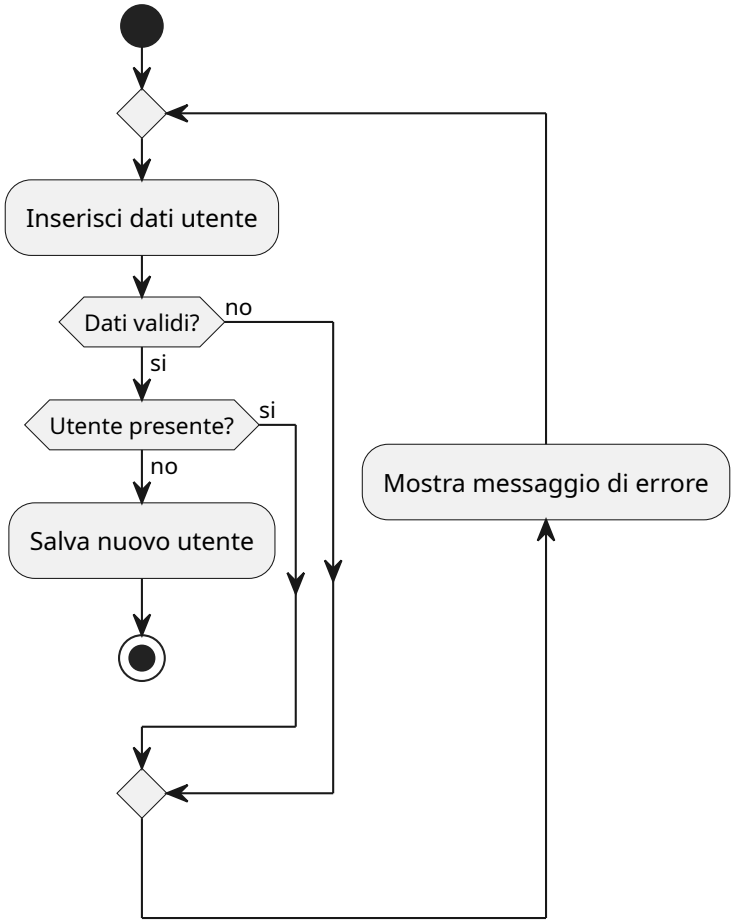


3.5 UC-15 – Rimozione di un libro

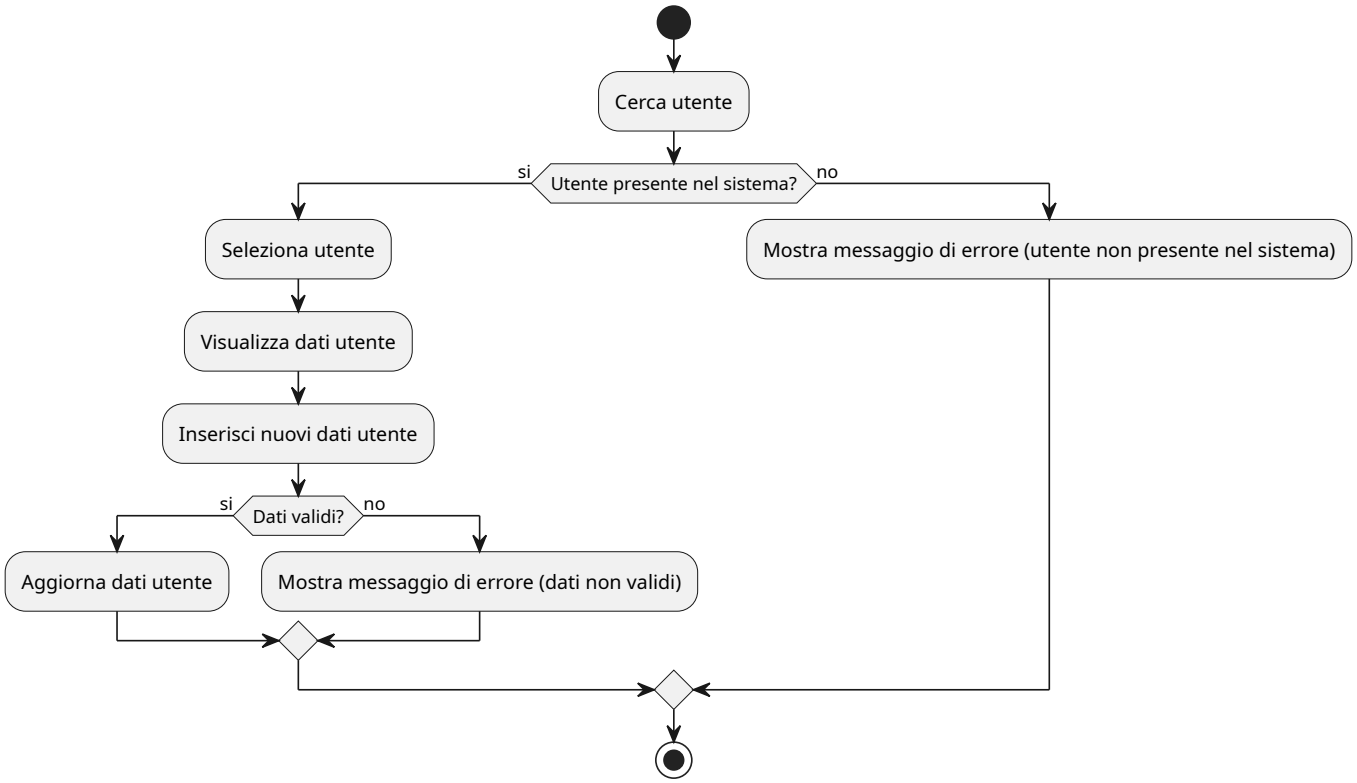


4 Diagrammi di Attività

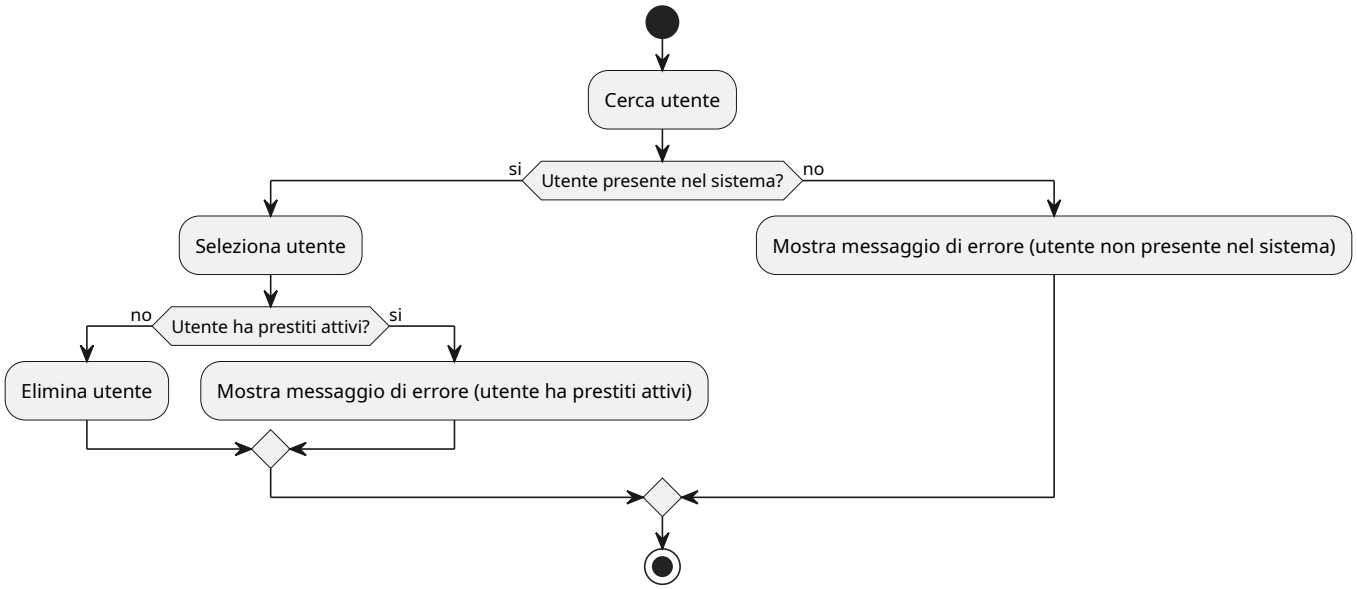
4.1 UC-6 – Registrazione di un utente



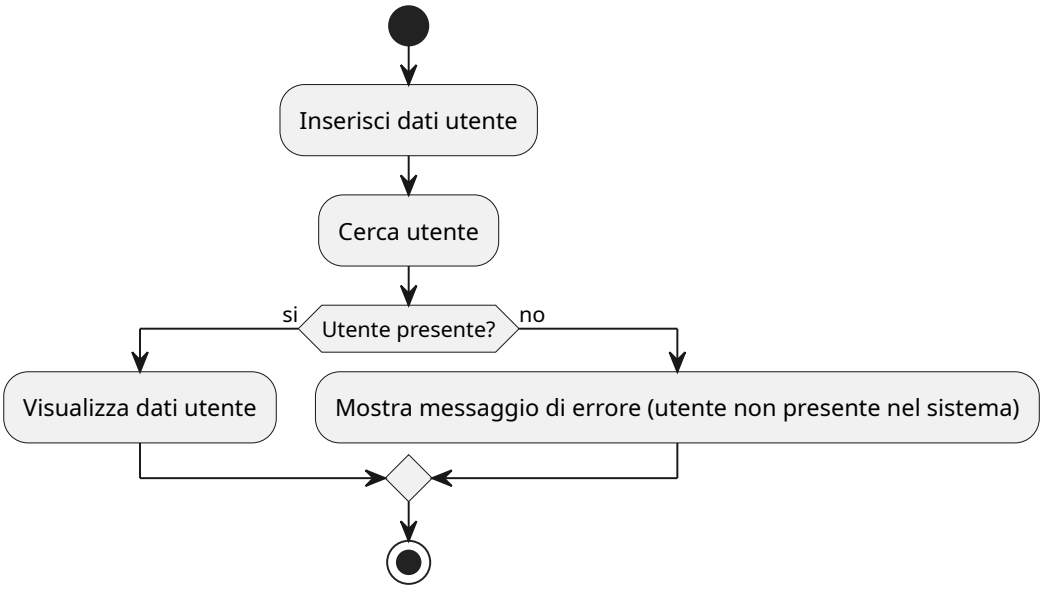
4.2 UC-7 – Modifica di un utente



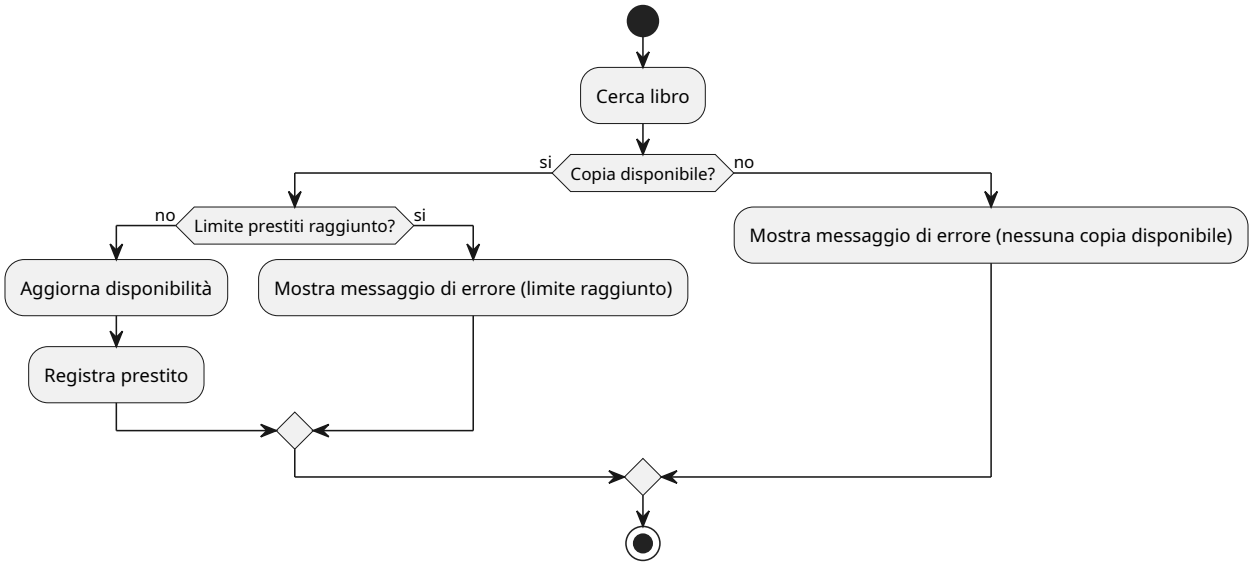
4.3 UC-8 – Rimozione di un utente



4.4 UC-9 – Ricerca di utenti



4.5 UC-10 – Restituzione di un prestito



4.6 UC-11 – Restituzione di un prestito