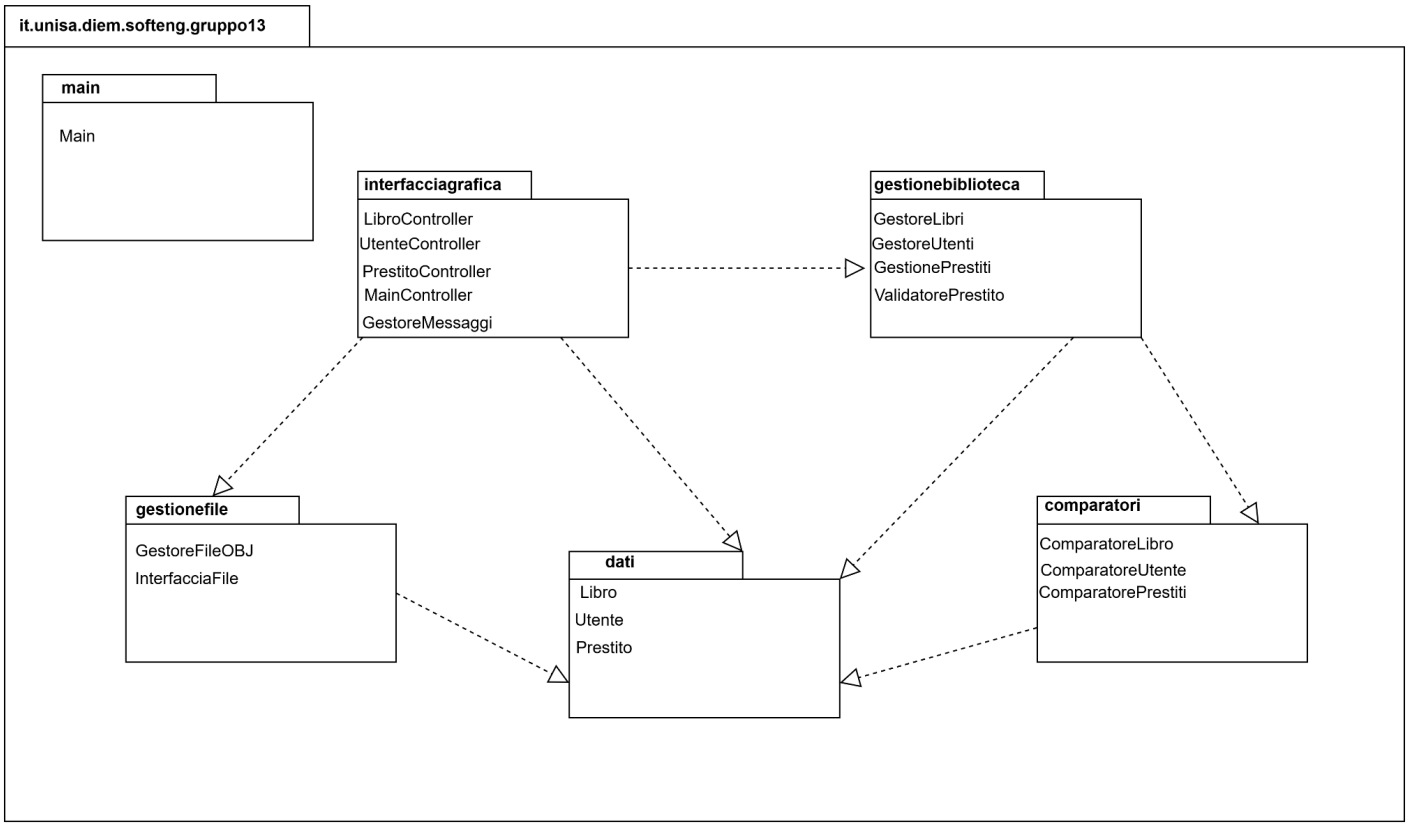
DESIGN
Diagrammi e scelte di progettazione

Gruppo 13: Bello Daniel, Capaldo Daniele, De Santis Andrea, Ferrara Stefano

INDICE

DIAGRAMMA DEI PACKAGE



COMMENTO AL DIAGRAMMA

Il sistema è strutturato secondo un'**architettura a livelli**, dove ogni package ha responsabilità specifiche. Questa suddivisione garantisce che le modifiche in un package non abbiano impatti distruttivi sugli altri, massimizzando la **manutenibilità**.

- Il package **interfacciagrafica** è il livello più alto dell'applicazione, che gestisce l'interazione con l'utente, la visualizzazione dei dati e la cattura degli eventi (click, input).

  Dipende dai package gestionebiblioteca (per eseguire operazioni), dati (per manipolare gli oggetti da visualizzare) e gestionefile (per la persistenza).

- Il package **gestionebiblioteca** rappresenta il cuore funzionale del sistema perché implementa le regole di business, la validazione dei dati e la gestione delle liste in memoria.

  Utilizza il package dati per definire le strutture su cui operare e comparatori per ordinare tali strutture. Non dipende né dalla GUI né dal gestore file.

- Il package **comparatori** fornisce i criteri di ordinamento personalizzati per le liste di oggetti.

Dipende dal package dati per accedere ai campi da confrontare.

- Il package **gestionefile** traduce gli oggetti in memoria in un formato di archiviazione e viceversa.

  Dipende solo dal package dati**,** per sapere cosa salvare. È completamente isolato dalla logica di business.

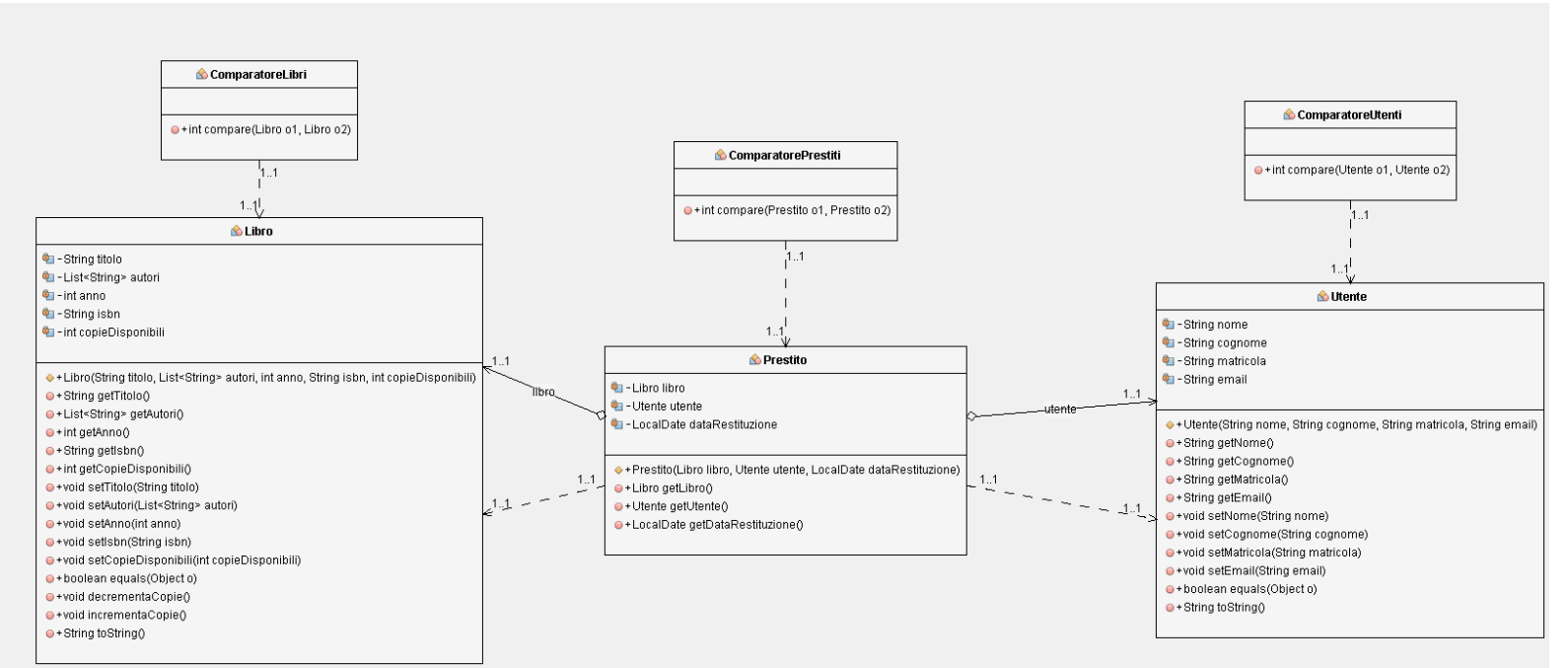
- Il package **dati** serve a definire la struttura dei dati scambiati tra tutti gli altri livelli.

  Non dipende da nessun altro package.

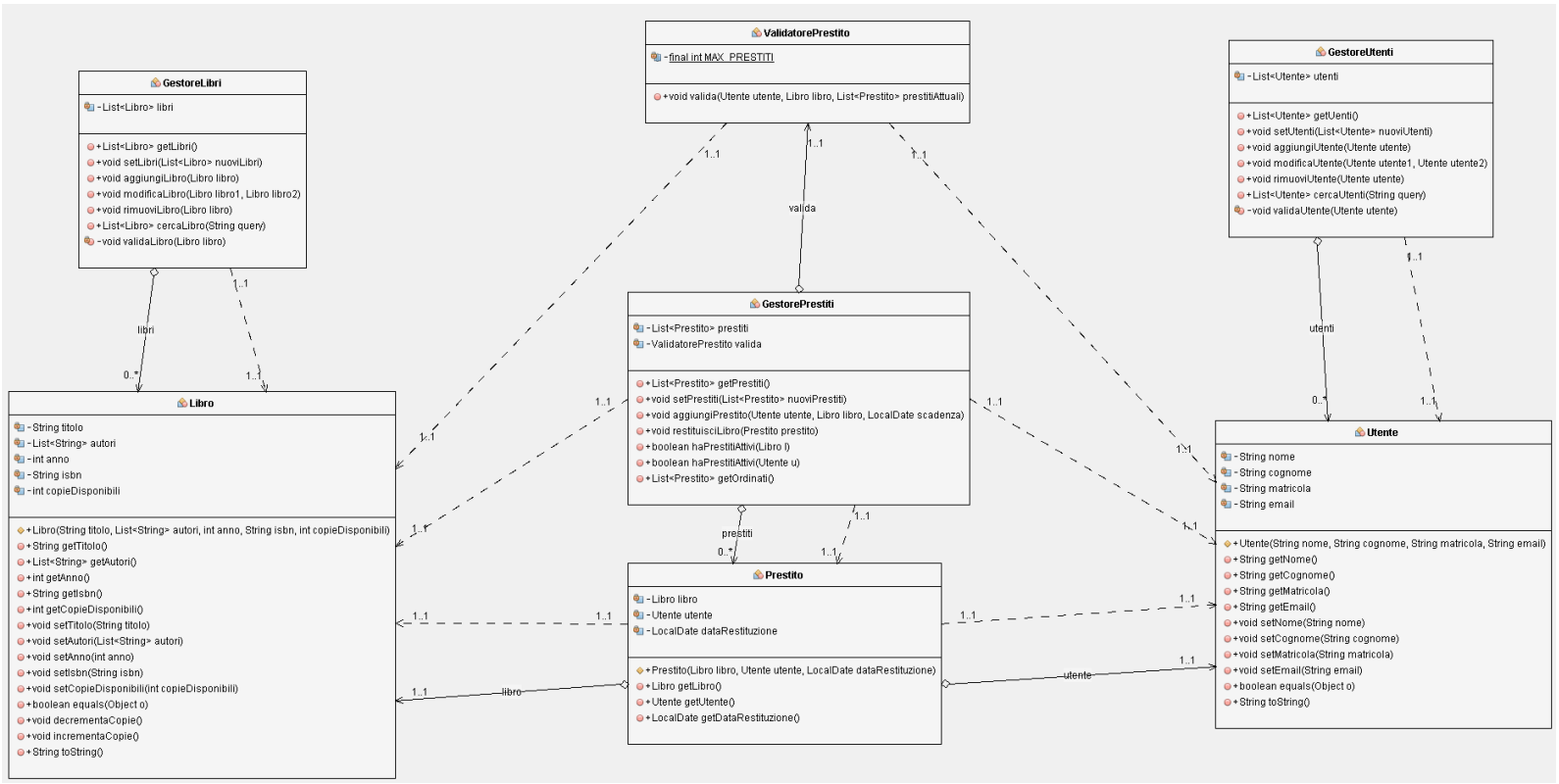
DIAGRAMMA DELLE CLASSI

Il Diagramma delle Classi illustra le relazioni statiche tra i componenti del sistema. Per maggiore chiarezza il diagramma principale è stato diviso in più parti.

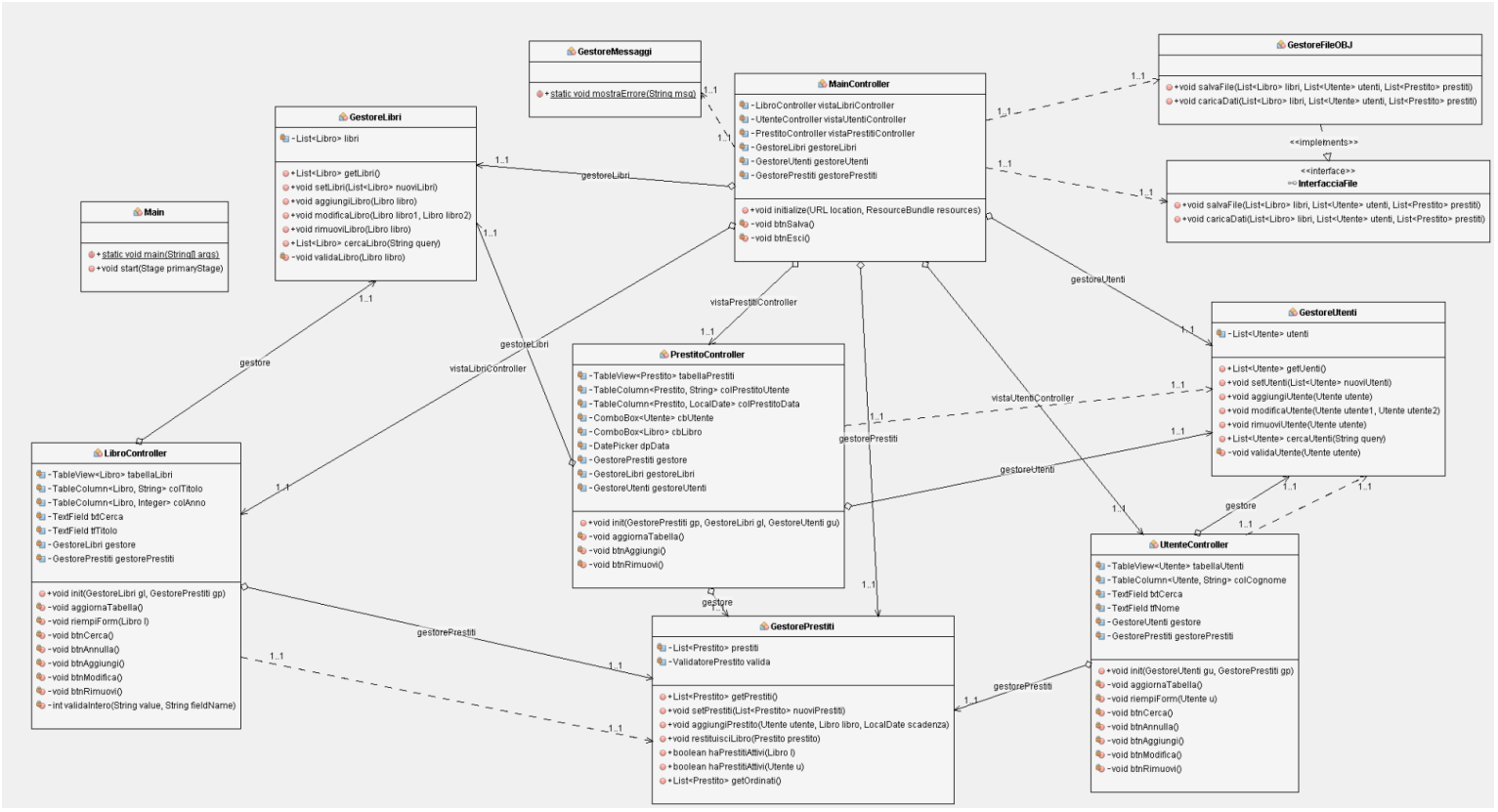
Il diagramma sottostante mostra le interazioni tra i Dati e i Comparatori.



Il diagramma sottostante mostra le relazioni tra Dati e Gestori.



Il diagramma sottostante mostra le relazioni tra Controller, FIle e Gestori.



SCELTE DI PROGETTAZIONE

Per la progettazione architetturale è stato usato un approccio Object Oriented, suddividendo le classi in base alle entità che rappresentano il dominio del problema, come Libro, Utente e Prestito. Le operazioni delle classi invece sono divise in metodi con approccio funzionale, basato sulle operazioni che devono essere realizzate, come l'aggiunta, la modifica, la ricerca e l'eliminazione.

ATTRIBUTI DI QUALITÀ

L'architettura realizzata garantisce la realizzazione di alcuni attributi di qualità (**Q.A.**) **interni** come la manutenibilità e la riusabilità.
La **Separazione delle Preoccupazioni** è stato il passo principale. Grazie alla divisione tra GUI, logica e dati, è garantita una certa **manutenibilità**, così che il software possa essere facilmente modificato o esteso. In più, l'ulteriore ripartizione di Controller e Gestori, mira a raggiungere una buona **modularità**, in modo che ogni classe abbia il minimo impatto sull'altro. Ciò ha permesso di realizzare classi generiche e **riutilizzabili**, come i comparatori, i gestori di file e delle finestre di dialogo,

Per quanto riguarda gli attributi di qualità **esterni**, gli obiettivi principali sono la **robustezza** e l'**usabilità**, in modo che l'utente possa interagire senza difficoltà con il sistema per raggiungere i suoi obiettivi.

COESIONE E ACCOPPIAMENTO

COESIONE

L'obiettivo del design è stato quello di **massimizzare la coesione** verso il livello più alto, quindi facendo in modo che le parti dei moduli fossero molto legate tra loro.

Le classi dei dati (**Libro, Utente, Prestito**) hanno **coesione funzionale**, siccome incapsulano solo gli attributi e i metodi strettamente pertinenti alla definizione di quell'entità, come i Getter e i Setter.

Anche le classi **GestoreLibro** e **GestoreUtente** e **GestorePrestito** presentano una **coesione funzionale.** Ogni modulo include funzionalità che lavorano insieme per realizzare un singolo compito ben definito: la gestione del ciclo di vita di una specifica entità. Queste classi infatti contengono i metodi di aggiunta, modifica, ricerca o rimozione dei rispettivi modelli Libro, Utente e Prestito.

ValidaPrestito, ComparatoreLibro, ComparatoreUtente, ComparatorePrestito, GestoreMessaggi sono ottimi esempi dell'applicazione del **Principio di Singola Responsabilità**, oltre che di **coesione funzionale**. Ognuna di esse ha un singolo scopo: verificare la validità di un prestito, comparare un libro, un utente o un prestito, creare finestre di dialogo per l'interfaccia.

Il **GestoreFile,** che permette di salvare/caricare dati su/da un file esterno, ha **coesione sequenziale.**

Nell'interfaccia grafica, **LibroController**, **UtenteController**, **PrestitoController** hanno diverse funzionalità che lavorano sui dati di input e quindi una **coesione comunicazionale**.

Il **MainController** ha coesione temporale dal momento che il metodo initialize() ha una forte coesione temporale: esegue una serie di operazioni (creare i gestori, iniettarli nei controller, caricare i dati) che non sono strettamente legate dalla logica, ma dal fatto che devono accadere tutte insieme all'avvio del programma. Tuttavia, considerando che è una classe che agisce come Composition Root, si può anche considerare come funzionale.

Il **Main** è **funzionale** dal momento in cui contiene i metodi per l'esecuzione dell'applicazione.

ACCOPPIAMENTO

Per quanto riguarda l'interdipendenza tra i moduli di un sistema, l'obiettivo primario è stato quello di raggiungere il livello ideale di Accoppiamento per Dati (**Data Coupling**), evitando forme più rigide attraverso anche l'uso di un corretto incapsulamento e variabili non globali.

Il **MainController** ha il maggior numero di dipendenze ma sono necessarie. Conosce e istanzia tutte le classi dei **Gestori** dei Dati**,** il **GestoreFile** e conosce tutti i **sotto-controller**. Ha un accoppiamento per controllo
permette a tutte le altre classi di avere un accoppiamento basso.

LibroController, **UtenteController** e **PrestitoController** hanno un **accoppiamento per dati** con i **Gestori** (dipendono da essi attraverso i loro metodi pubblici) e con i **Dati**, che permettono lo scambio di informazioni.

Le classi di **GestoreLibri** e **GestoreUtenti** presentano un **accoppiamento per dati** perché dipendono esclusivamente da **Libro** o **Utenti** e dai rispettivi **Comparatori**.

La stessa logica è stata seguita per **GestorePrestiti**, che ricava da **Libro**, **Utenti, Prestito** e **ValidatorePrestito** unicamente le informazioni strettamente necessarie per il funzionamento.

ValidatorePrestito e **Comparatori** hanno un **accoppiamento per dati** con le strutture dati per cui operano.

La classe del **GestoreFile** ha bisogno della struttura dei **Dati** per poter caricare o salvare, con un livello di **accoppiamento per dati**.

PRINCIPI SEGUITI

Il progetto è stato sviluppato ponendo particolare attenzione alla qualità del design software, applicando i principi di buona progettazione.

La semplicità è stata preferita alla complessità non necessaria (**KISS**) e sono state evitate ripetizioni e ridondanze (**DRY**). Da qui l'idea di implementare dei metodi privati con le validazioni e classi separate per la gestione di finestre e comparazioni.

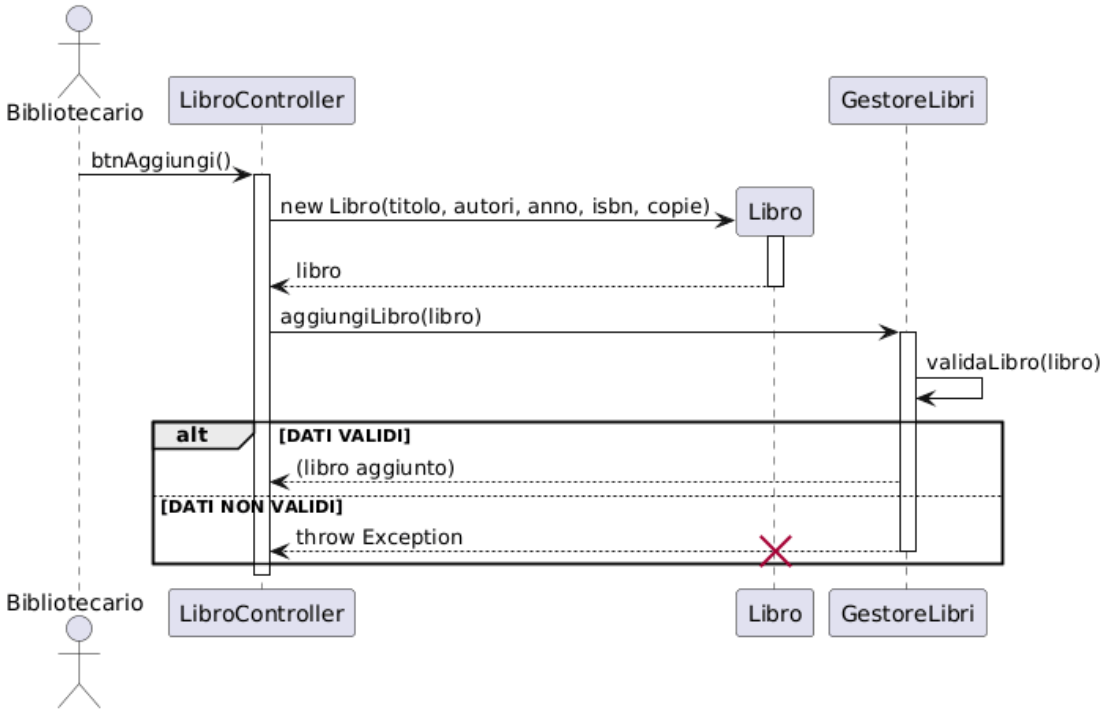
Queste caratteristiche sono conformi al **Single Responsibility Principle.** Ad esempio, l'uso di un validatore di prestito esterno permette di modificare i criteri di conferma di un prestito senza dover modificare il codice delle classi del modello o dei gestori.

Il **principio di Liskov** è stato rispettato con l'interfaccia dei file: il Controller dipende solo da questa astrazione, così da rendere possibile il cambiamento del formato file senza modificare il codice del Controller.

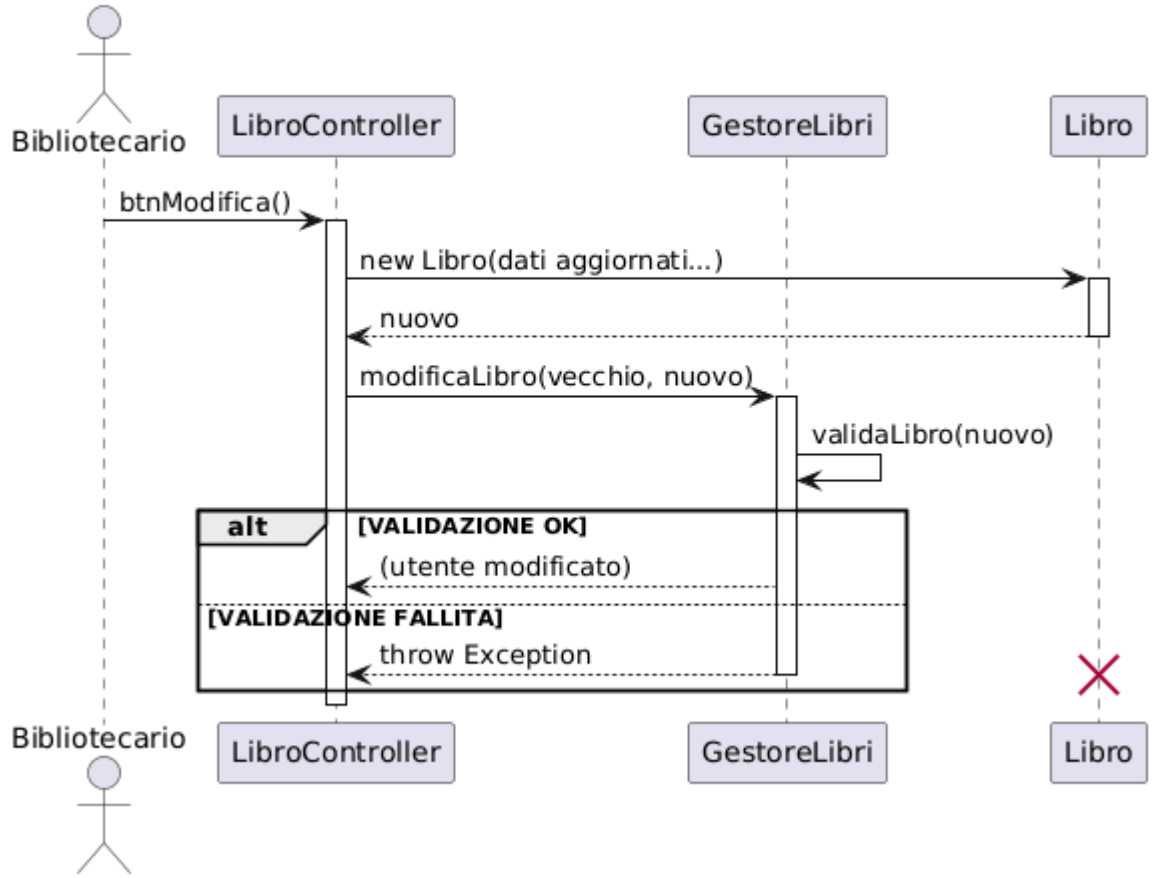
DIAGRAMMI DI SEQUENZA

I diagrammi di sequenza illustrano il **flusso logico** e le interazioni tra i componenti dell'architettura. Per semplicità sono stati omessi alcuni dettagli inerenti a metodi interni e all'interfaccia grafica, come l'aggiornamento delle tabelle e la comparsa dei messaggi di errore collegati alle eccezioni.
Come scenari più significativi sono stati considerati le aggiunte, le modifiche, le eliminazioni e il salvataggio su file.

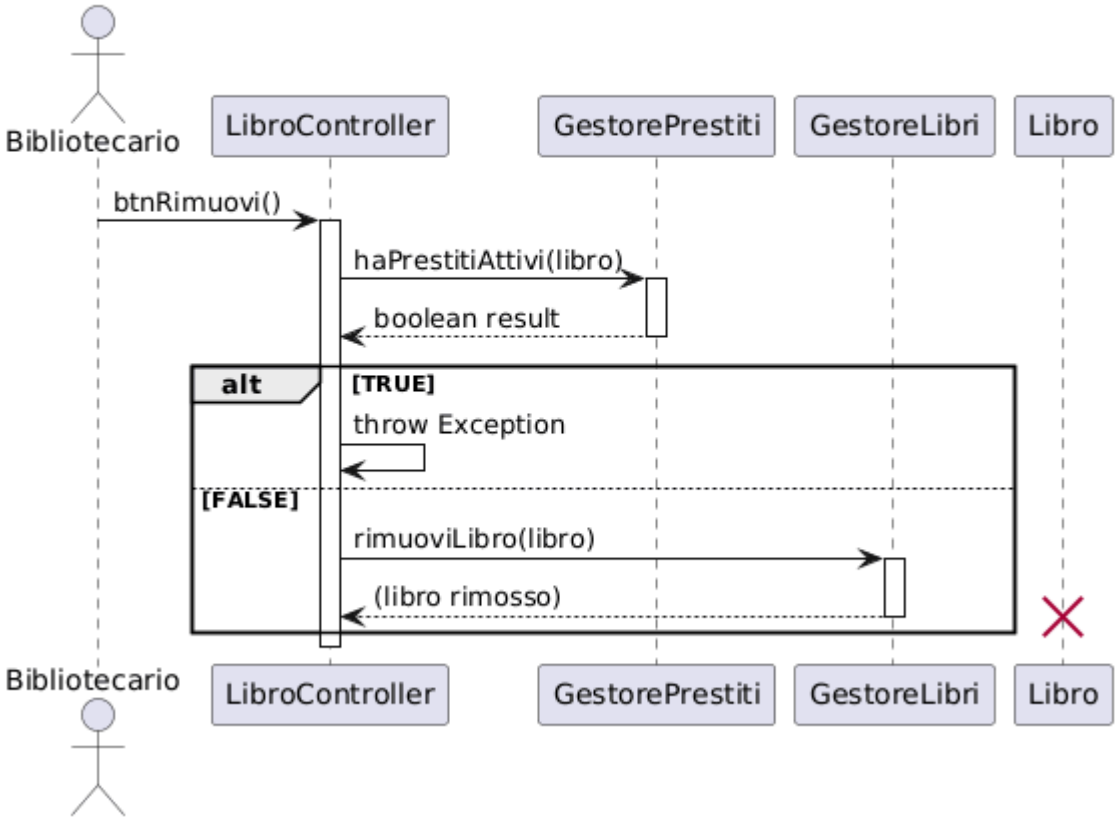
AGGIUNTA DI UN NUOVO LIBRO



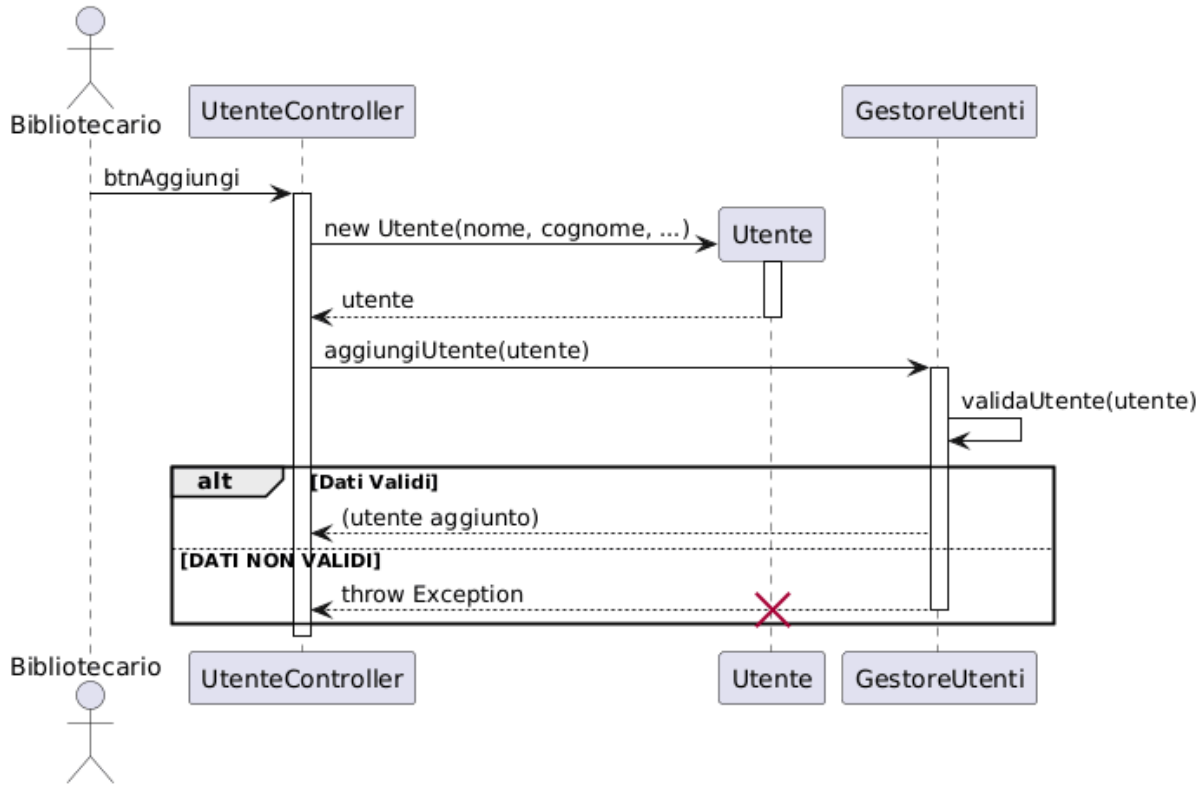
MODIFICA DI UN LIBRO



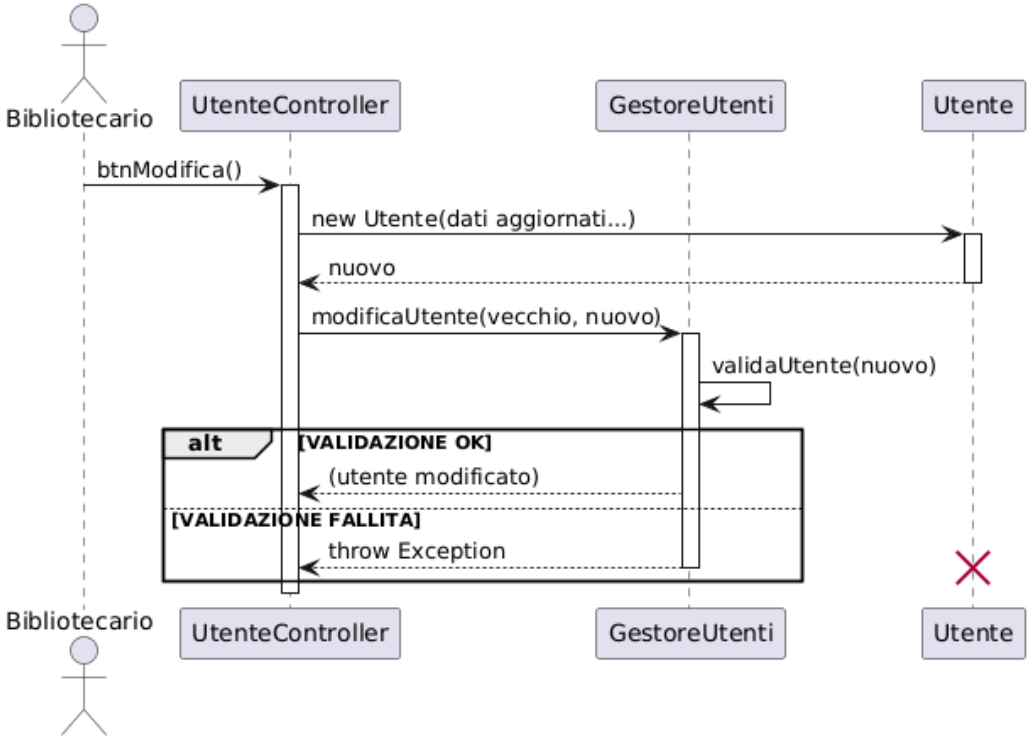
ELIMINAZIONE DI UN LIBRO



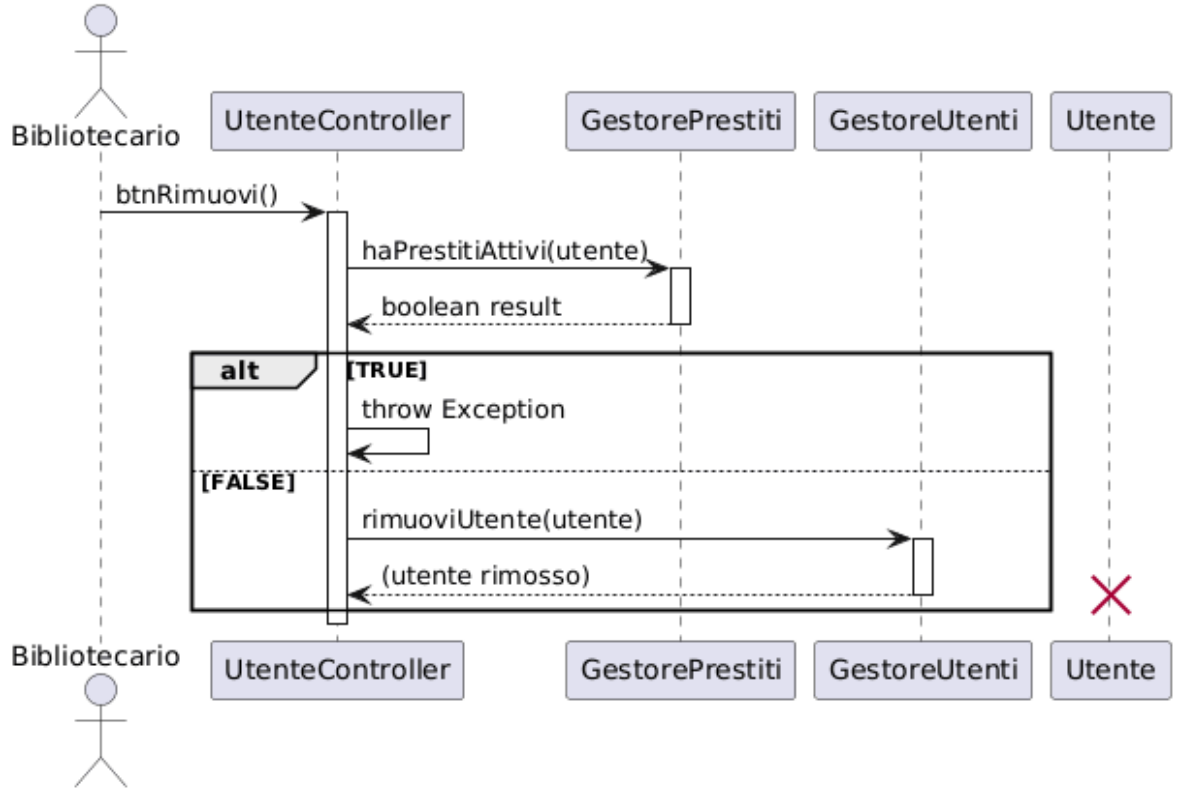
AGGIUNTA DI UN NUOVO UTENTE



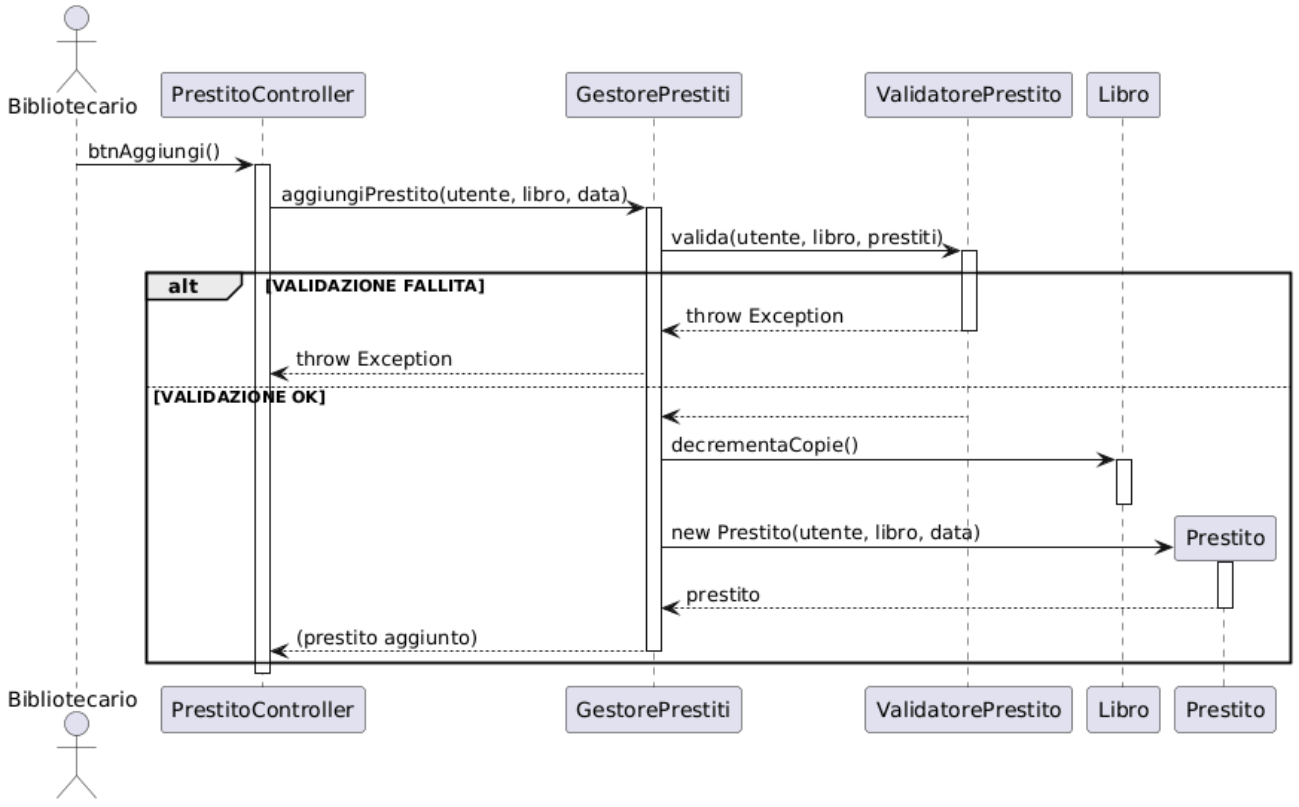
MODIFICA DI UN UTENTE



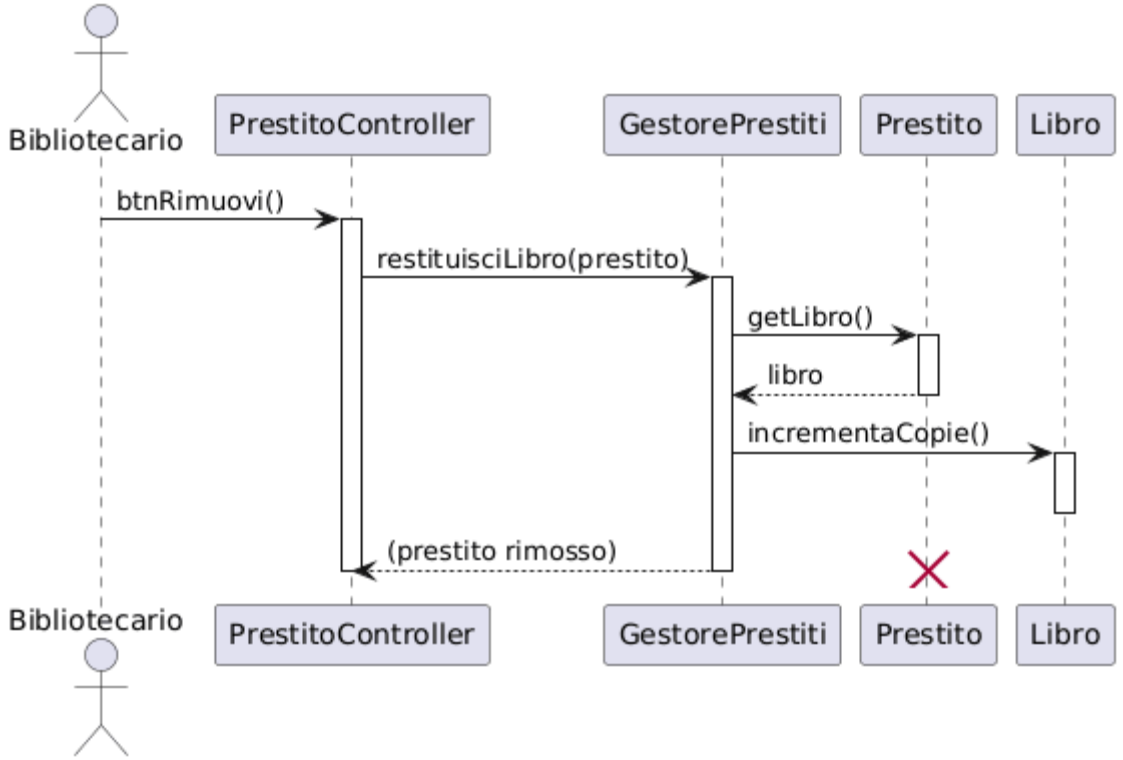
ELIMINAZIONE DI UN UTENTE



REGISTRAZIONE DI UN PRESTITO



ELIMINAZIONE DI UN PRESTITO



SALVATAGGIO SU FILE