Documento del Design

Titolo del Progetto: Project Library G13
Autore: Gruppo 13 (Iacuzzo Chiara, Rossino Giacomo, Risi Christian, Vessa Davide)
Data: XX/11/2025

Indice

1. Architettura del sistema

L'architettura del sistema è stata progettata seguendo un approccio modulare e stratificato, finalizzato a garantire la manutenibilità, la scalabilità e la chiara separazione delle responsabilità.
Il sistema è suddiviso nei seguenti macro-moduli:

1.1 Descrizione dei Moduli

1.1.1 Modulo InterfacciaUtente (GUI)

Responsabilità: Questo package, funge da Controller nell'architettura MVC. Poiché le interfacce grafiche sono realizzate esternamente con SceneBuilder (file FXML), le classi qui contenute (nei package GUI_Login, GUI_Libro, ecc.) si occupano di gestire l'interazione utente, quindi intercettano gli eventi (click, input), validano le richieste e invocano i metodi corretti nei moduli logici.

Dipendenze: Ogni sotto-package dipende direttamente dal modulo di logica (GestioneLibri, GestioneUtenti, GestionePrestiti, Autenticazione) verso i quali inoltrare le richieste di elaborazione dei dati. Inoltre ognuno dipende dal modulo di sorting relativo (SortingLibri, SortingUtenti, SortingPrestiti) necessario per creare l'oggetto comparator da passare.

1.1.2 Modulo GestioneLibro

Responsabilità: Si occupa della gestione delle entità Libri, permettendo tutte le modifiche sul ciclo di vita di una singola entità e gestendo nel complesso il loro insieme (aggiunta, rimozione, ricerca, incremento/decremento copie, ecc..).

Dipendenze: Questo modulo dipende da:

- SortingLibri;
- EccezioniLibri;
- SalvataggioFileLibri;

1.1.3 Modulo GestioneUtente

Responsabilità: Si occupa della gestione delle entità Utenti, permettendo tutte le modifiche sul ciclo di vita di una singola entità e gestendo nel complesso il loro insieme (aggiunta, rimozione, ricerca, ecc..)

Dipendenze: Questo modulo dipende da:

- SortingUtenti;
- EccezioniUtenti;
- SalvataggioFileUtenti;
- GestionePrestiti (Per la presenza di una lista di prestiti per ogni utente).

1.1.4 Modulo GestionePrestiti

Responsabilità: Si occupa della gestione delle entità Prestiti, che è definita dal legame più complesso Libro-Utente, permettendo tutte le modifiche sul ciclo di vita di una singola entità e gestendo nel complesso il loro insieme (aggiunta, rimozione, ricerca, ecc..). Ciò avviene grazie all'implementazione della classe GestorePrestito che ha il compito di controllare tutte le regole che permettono di approvare o meno un prestito.

Dipendenze: Questo modulo dipende da:

- SortingPrestiti;
- EccezioniPrestiti;
- SalvataggioFilePrestiti;
- GestioneLibro (Per il controllo del vincolo sui Libri);
- GestioneUtente (Per il controllo del vincolo sugli Utenti).

1.1.5 Modulo SalvataggioFile:

Responsabilità: Garantisce che i dati persistono al riavvio dell'applicazione. Si occupa di serializzare gli oggetti e salvarli su disco, e di deserializzarli all'avvio.

Dipendenze: Non ha dipendenze.

1.1.6 Modulo Autenticazione

Responsabilità: Verifica le credenziali del bibliotecario prima di concedere l'accesso alle funzionalità di sistema.

Dipendenze: Questo modulo dipende da:

- EccezioneAutenticazione.

1.1.7 Modulo Eccezioni

Responsabilità: Si occupa di gestire l'avvenimento di un'eccezione.

Dipendenze: Non ha dipendenze.

1.1.8 Modulo Sorting

Responsabilità: Fornisce le logiche di comparazione al fine di ordinare le varie liste (CatalogoLibri, ListaUtenti, ElencoPrestiti) secondo un criterio stabilito.

Dipendenze: Non ha dipendenze.

1.2 Implementazione dei componenti dello schema MVC

La struttura principale della nostra applicazione, si basa sull'uso del pattern architetturale Model-View-Controller (MVC), utile principalmente per gestire le interazioni tra la GUI e la logica di gestione.
Nello specifico l'applicazione implementa il pattern MVC nel seguente modo:

1.2.1 Model

Il model non è costituito da un insieme di classi, ma da un insieme di package che gestiscono i dati e la logica di business, ovvero: GestioneLibro, GestioneUtente, GestionePrestito (p. 1.1.2-3-4). Rappresenta i dati e la logica.

1.2.2 View

Le view è costituita dai file FXML, creati tramite SceneBuilder (è stata creata una View per ogni finestra che può essere visualizzata). Si occupano esclusivamente dell'interfaccia grafica e della presentazione dei dati all'utente (p. 4). Non contengono logica di business complessa.

1.2.3 Controller

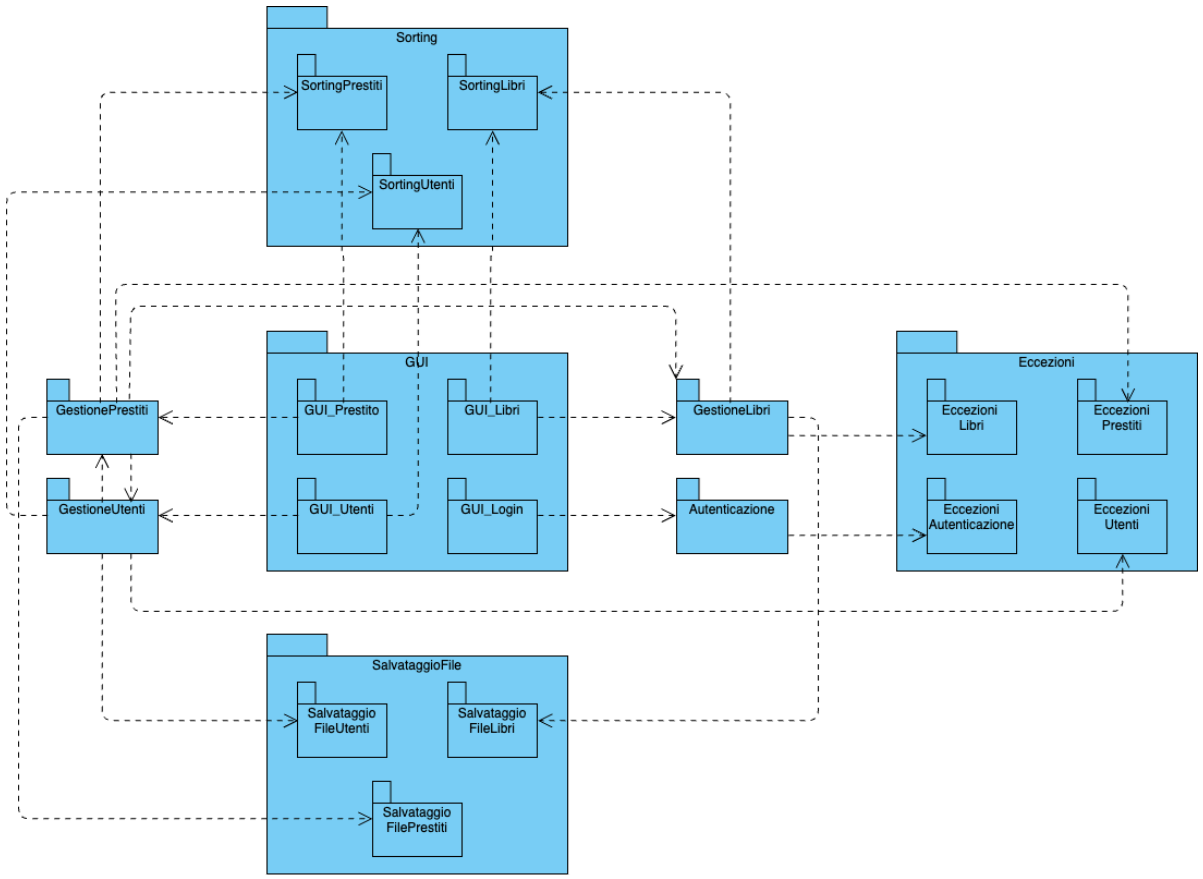
Il controller viene rappresentato dalle classi dentro il package GUI (p.1.1.1): LoginViewController, CatalogoLibroViewController, GestioneUtenteViewController e GestionePrestitoViewController. Agisce da intermediario tra il Model e la View. Riceve gli input dalla View, interpreta la richiesta e invoca i metodi appropriati nel Model; una volta aggiornati i dati, il Controller si occupa di riflettere i cambiamenti nella View.

1.3 Adozione dello Strategy Pattern

Tra i pattern utilizzati, si ha anche lo Strategy Pattern. Il package Sorting (p. 1.1.8) utilizza l'interfaccia Comparator (si veda la documentazione Oracle); ciò permette di definire una famiglia di "algoritmi" e renderli intercambiabili durante il processo di esecuzione in base alle esigenze del bibliotecario.
Il Controller può quindi selezionare dinamicamente la strategia di ordinamento più

appropriata (es. per Titolo, per Autore, per Anno) in risposta alle azioni dell'utente (il bibliotecario), senza che la logica di visualizzazione debba conoscere i dettagli dell'algoritmo utilizzato.

1.4 Diagramma dei Package



2. Modello statico

2.1 Principi di Buona Progettazione

La progettazione del sistema Library G13 è stata condotta con l'obiettivo di avere una buona manutenibilità e scalabilità e una complessità del progetto bassa.
Sono stati applicati i seguenti principi generali e orientati agli oggetti:

2.1.1 Principi Generali

Separazione delle preoccupazioni (SoC): Come evidenziato nel paragrafo 1.2, con l'utilizzo dell'architettura MVC, il sistema separa i diversi aspetti dell'applicazione. La logica del layout (FXML) è ortogonale alla logica del gestore.

> **Applicazione nel progetto**: Il modulo delle View (p. 1.2.1) non contiene logica per la gestione dei dati, ma essa si limita ad inoltrare gli input al Controller (p. 1.2.3). Anche il modulo del SalvataggioFile (p. 1.1.5) è separato dalla logica di gestione in memoria (p. 1.1.2-3-4), permettendo di modificare la strategia di persistenza senza impattare sulla logica applicativa del programma.

KISS (Keep It Simple, Stupid): Si è preferito adottare una struttura modulare chiara e con algoritmi lineari per evitare complessità non necessarie nel progetto. Ogni modulo presente espone solo metodi necessari per il funzionamento del sistema, riducendo per l'appunto la complessità del codice, anche per avere una manutenibilità migliore.

YAGNI (You Aren't Going to Need It): Si è preferito non inserire funzionalità non richieste, per evitare di avere un codice inutilmente più esteso e per dedicare il tempo per la realizzazione di funzionalità necessarie.

Don't Repeat Yourself (DRY): ogni funzionalità o logica deve avere un'unica astrazione di tale funzionalità, questo per evitare ripetizioni.
Nel sistema vi è una ripetizione degli algoritmi di salvataggio, quindi il principio non è sempre rispettato, in quanto abbiamo preferito prediligere un'alta coesione ed un basso accoppiamento.

2.1.2 Principi SOLID e Orientati agli oggetti

Single Responsibility Principle (SRP): Ogni modulo svolge un compito ben definito.

> **Applicazione nel progetto**: La distinzione fra il modulo GestionePerstiti (p 1.1.4) e il modulo del SalvataggioFilePrestiti (p. 1.1.5) assicura che una modifica al formato di salvataggio dei dati non vada a introdurre bug o errori per la Gestione dei Prestiti.

Open-Closed Principle (OCP): Ogni classe, modulo o metodo è chiuso alla modifica, per evitare che andando a modificare si possa avere un impatto sul codice, ma aperte all'estensione, ossia consentire modifiche che vadano ad adattare l'unità a nuovi scenari, diversi da quelli considerati all'inizio.

> **Applicazione nel progetto**: L'adozione dello Strategy Pattern nel modulo Sorting (p. 1.1.8) è l'esempio cardine di questo principio. Grazie all'utilizzo dell'interfaccia Comparator (si veda la documentazione Oracle) è possibile introdurre nuovi tipi di ordinamento (quindi una apertura verso le estensioni) senza dover modificare il codice nelle classi entità (chiusura alla modifica).

Liskov Substitution Principle (LSP): Gli oggetti possono essere sostituiti con istanze dei loro sottotipi senza alterare la correttezza del programma.

> **Applicazione nel progetto**: Rifacendoci al caso del Sorting (p 1.1.8), le classi presenti in questo modulo espongono diversi criteri di ordinamento. Il codice che esegue l'ordinamento presume che ogni comparatore restituisca un valore valido per il confronto. I diversi metodi dei comparatori sono perfettamente intercambiabili, poiché restituiscono sempre un risultato coerente con il contratto. Il sistema può sostituire la strategia di ordinamento in qualsiasi momento senza rischiare crash o malfunzionamenti.

Interface Segregation Principle (ISP): Un client non dipende da metodi che non utilizza.

> **Applicazione nel progetto**: Il sistema evita di avere una classe di gestione globale, che contiene tutti i metodi per gestire l'intero sistema. Al contrario, vi è una suddivisione dei vari moduli, in modo da garantire una indipendenza tra le varie classi, evitando così che una classe vada a dipendere da metodi o interfacce che non utilizzerà mai. Il GestioneLibroViewController dipende solo dai metodi definiti in GestioneLibro. Se venissero implementate nuove funzionalità nella gestione degli utenti, il codice del GestioneLibriViewController non verrebbe influenzato e non necessiterebbe di una ricompilazione, rispettando la segregazione delle responsabilità.

Dependency Inversion Principle (DIP): I moduli di alto livello non dipendono dai dettagli di basso livello, ma entrambi dipendono da astrazioni.

> **Applicazione nel progetto**: I Controller della GUI (alto livello) non implementano direttamente la logica algoritmica. Si pensi agli algoritmi di Sorting o al SalvataggioFile. Dipendono dall'astrazione di un'interfaccia, nel primo caso

Comparator, nel secondo caso Serializable (si veda la documentazione Oracle). Permette di modificare la logica interna di un confronto (per quanto riguarda il Sorting) senza dover toccare il codice dei Controller.

2.1.3 Principio di Robustezza

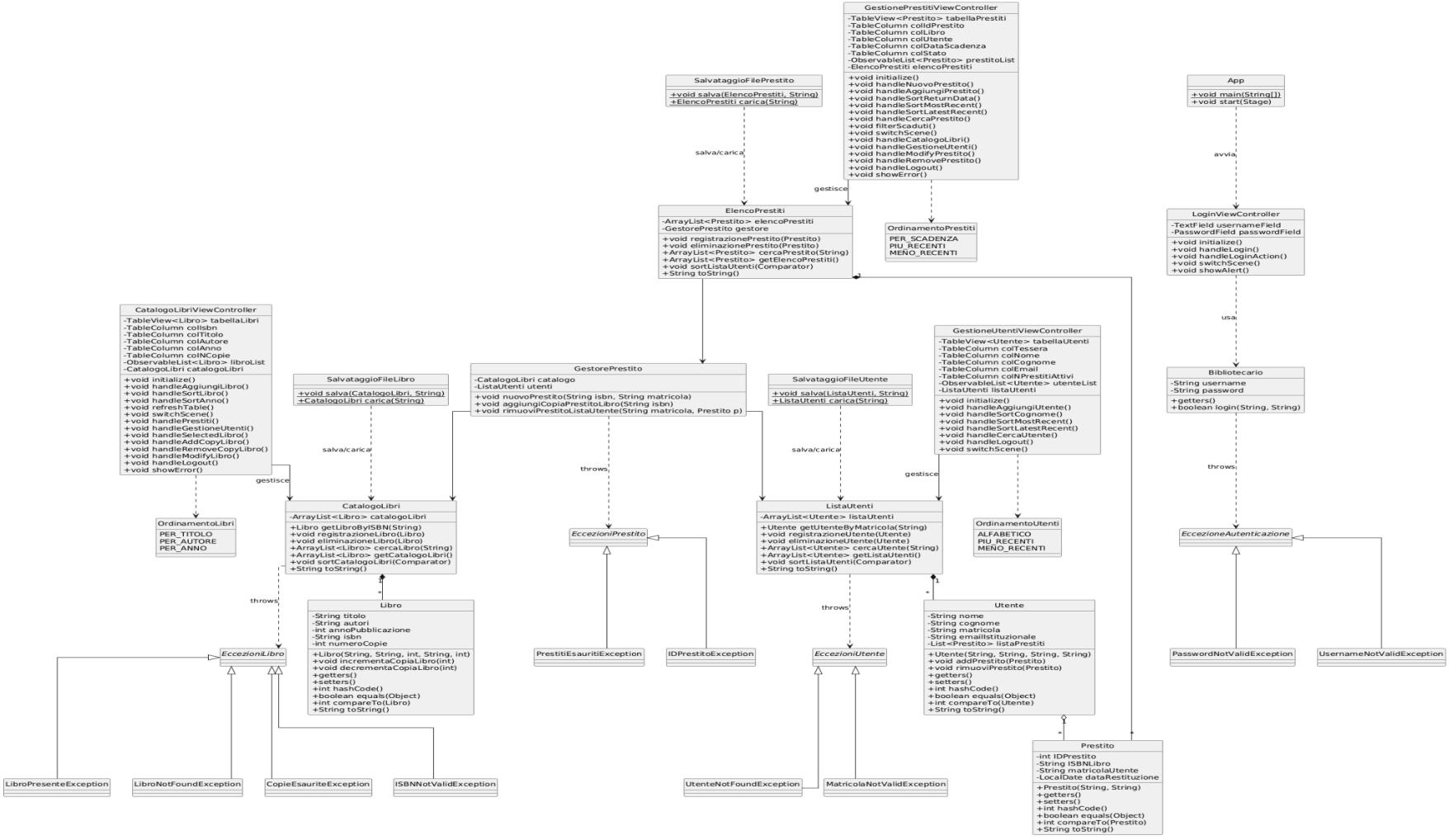
Il sistema è stato progettato per “limitare i danni" nel caso le precondizioni non vengano rispettate. L’obiettivo è quello di garantire che non si sia una perdita di dati e che il software non si comporti in modo ambiguo e inconsistente.

Applicazione nel progetto: Questo principio è stato implementato grazie al modulo delle Eccezioni (p. 1.1.7) e grazie ai moduli dei gestori (1.1.2-3-4).

Gestione di ambiguità e inconsistenza: Nei moduli di gestione, vengono verificate le precondizioni (ad esempio che il libro non sia già presente nel catalogo). Se una operazione fallisce, viene sollevata un’eccezione specifica e l’operazione viene abortita prima di modificare i dati.

Limitazione dei danni: Il sistema intercetta l’errore tramite i blocchi try-catch. Questo impedisce i crash dell’applicazione e fornisce un feedback visivo all’utente, in modo da non interrompere il funzionamento del software

2.2 Diagramma delle Classi



2.3 Scelte Progettuali in termini di Coesione ed Accoppiamento

Classi di interfaccia grafica GUI (LoginViewController, MainDashboard, CatalogoLibriViewController, GestioneUtentiViewController e GestionePrestitiViewController):

Appartenendo al livello di interfaccia grafica è possibile definire lo stesso livello di coesione e accoppiamento:

- **Coesione Logica**: Queste classi raggruppano operazioni differenti ma collegate logicamente alla gestione dell'interazione con l'utente.
- **Accoppiamento per Controllo**: Le classi controllano il flusso dell'applicazione, decidendo quali operazioni invocare, quali viste caricare e quali moduli attivare in risposta alle azioni dell'utente.

Classe App:

- **Coesione Temporale**: Poiché tutte le sue operazioni sono concentrate nella fase di avvio dell'applicazione.
- **Accoppiamento per Timbro**: La classe dipende dalla struttura dati dell'oggetto Stage per funzionare correttamente. Inoltre, esiste un accoppiamento implicito con le risorse esterne in quanto la classe "conosce" e richiede la presenza di specifici percorsi.

Classi di Gestione di una entità (Bibliotecario, Libro, Utente, Prestito):

Svolgendo gli stessi compiti ma su entità diverse, è possibile definire lo stesso livello di coesione e accoppiamento:

- **Coesione Funzionale**: Il metodo delle varie classi Bibliotecario, Libro, Utente, Prestito si occupano solo di una singola entità, e tutti i loro metodi gestiscono solo quella specifica entità.
- **Accoppiamento per Dati**: Le interazioni con le altre classi avvengono esclusivamente tramite lo scambio di parametri e valori di ritorno, senza controllo diretto del flusso di esecuzione di altri moduli.

Classi di Gestione delle Collezioni (CatalogoLibri, ListaUtenti, ElencoPrestiti):

Classi CatalogoLibri, ElencoPrestiti

- **Coesione Comunicazionale**: I metodi delle classi CatagoloLibri e ElencoPrestito lavorano sugli stessi dati e cioè rispettivamente i libri e prestiti.
- **Accoppiamento per Dati**: Le classi contengono al proprio interno solamente le informazioni strettamente necessarie per il suo funzionamento, che non comprendono solo tipi primitivi ma anche oggetti delle classi Libro e Prestito.

Classe ListaUtenti:

- **Coesione Comunicazionale**: I metodi della classe ListaUtenti lavorano sugli stessi dati, vale a dire gli Utenti.
- **Accoppiamento per Timbro**: La classe non dipende solo da dati primitivi e dalla classe Utente, necessarie al suo funzionamento, ma anche dalla struttura dati completa di un'altra classe, ovvero Prestito, poiché contiene al proprio interno una lista di tali oggetti.

Classe GestorePrestito:

- **Coesione Funzionale**: la classe svolge un singolo compito ben definito: la validazione e l'autorizzazione di un prestito.
- **Accoppiamento per Timbro**: Per effettuare i controlli sulla validazione di un prestito, la classe accede a due strutture dati complete di altre classi, ovvero: catalogoLibri e listaUtenti.

Classi Salvataggio File (SalvataggioFileLibro, SalvataggioFileUtente, SalvataggioFilePrestito):

Le classi di salvataggio si occupano tutte delle persistenza dei dati, quindi è possibile definire lo stesso livello di coesione e accoppiamento:

- **Coesione Funzionale**: Le classi hanno il solo compito di salvare le informazioni passate, in modo serializzato, in file binari specifici.
- **Accoppiamento per Dati**: Le classi ricevono in ingresso l'intero oggetto da salvare (CatalogoLibri, ListaUtenti o ElencoPrestiti) , necessario per implementare il loro funzionamento.

Classi Ordinamento (OrdinemantoLibri, OrdinamentoPrestiti, OrdinamentoUtenti):

Poiché tutte le classi di ordinamento differiscono solo per l'entità che vanno ad ordinare possiamo definire lo stesso livello di coesione ed accoppiamento per le tre classi:

- **Coesione Funzionale**: Tutte le classi di ordinamento si occupano esclusivamente di fornire criteri di confronto (Comparator) per una specifica entità del dominio. Ogni classe ha un solo scopo ben definito: permettere l'ordinamento degli oggetti.
- **Accoppiamento per Dati**: Le classi di ordinamento restituiscono collezioni ordinate senza controllare direttamente altre parti del sistema.

Classi Eccezioni (EccezioneAutenticazione, EccezioniLibro, EccezioniPrestito, EccezioniUtente):

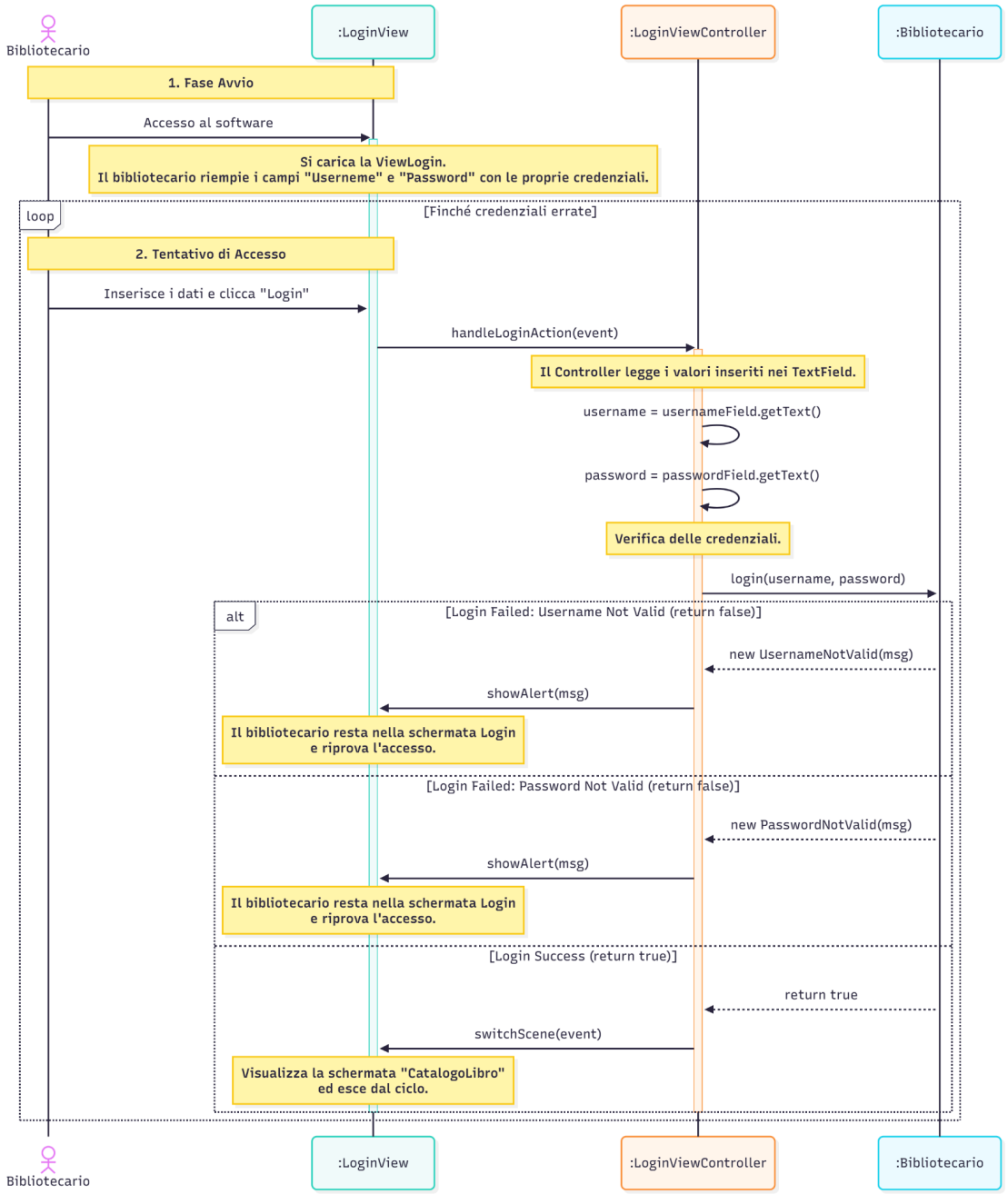
Poiché tutte le classi per la gestione delle eccezioni differiscono solo per l'entità che gestiscono, è possibile definire lo stesso livello di coesione ed accoppiamento:

- **Coesione Funzionale**: Tutte le classi di eccezione hanno un unico compito ben definito, rappresentare e segnalare uno specifico errore del sistema. Ogni classe gestisce una sola tipologia di anomalia.
- **Accoppiamento per Dati**: Le classi di eccezione trasportano esclusivamente informazioni sull'errore (messaggio, tipo di violazione) e vengono utilizzate come parametri o come oggetti lanciati (throw) dalle altre classi.

3. Modello dinamico

3.1 Diagrammi di sequenza

3.1.1 Login:

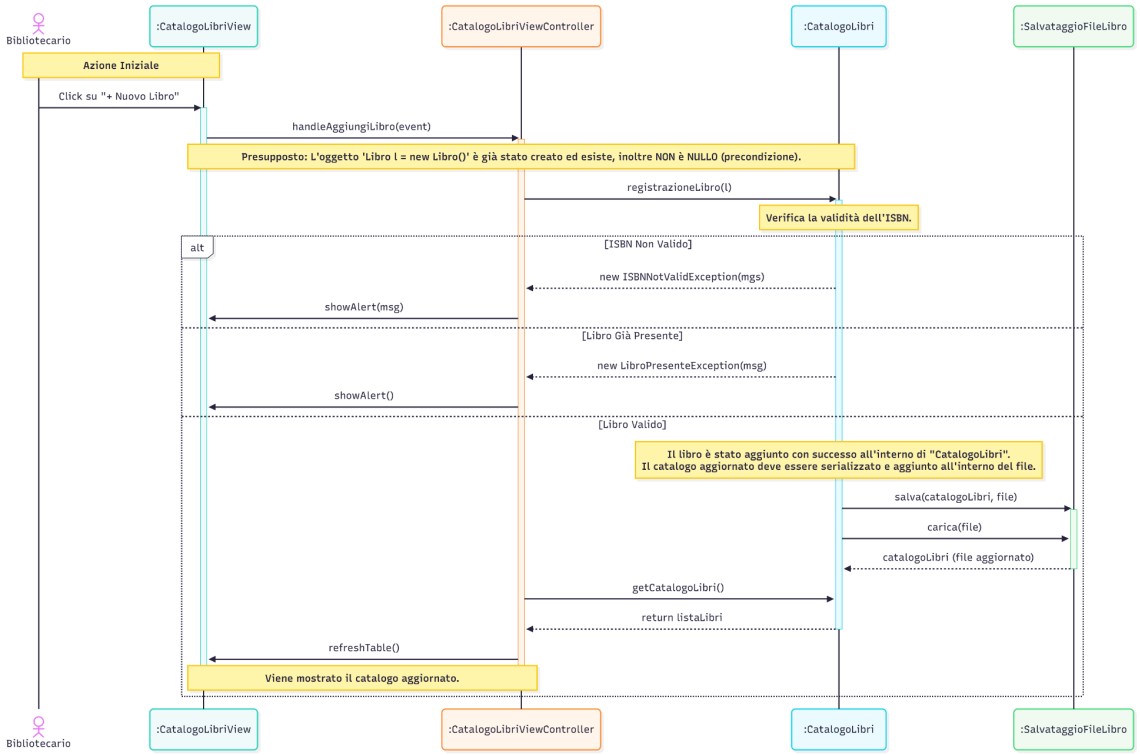


Commento:

Descrizione del diagramma: Questo diagramma di sequenza descrive il processo di autenticazione che viene effettuato quando il bibliotecario accede al software.

1. **Azione iniziale**: Il processo viene avviato dall'azione dell'attore "Bibliotecario". Quando il bibliotecario accede al sistema viene mostrata la finestra di Login, ovvero la "LoginView".
1. **Processo di autenticazione**: La LoginView mostra due TextField (Username e Password) dove il bibliotecario deve inserire le proprie credenziali e un bottone "Login". Quando il bibliotecario riempie i campi e clicca il bottone la View passa il controllo al LoginViewController. Attraverso il metodo "handleLoginAction()", il Controller salva i dati inseriti dal bibliotecario nei due attributi definiti nella classe e poi li verifica. Per effettuare questa verifica si invoca il metodo "login()" della classe Bibliotecario, quindi il controllo viene passato al Model.
2. **Esiti del Login**:
    a. **Credenziali non valide**: Se le credenziali inserite dal bibliotecario sono errate, il Controller mostra l'errore e rimane nel loop, ovvero viene mostrata sempre la finestra di Login. Inoltre a seconda delle credenziali errate possono essere lanciate due eccezioni diverse: "UsernameNotValidException" se viene inserito un username errato e/o "PasswordNotValidException" se viene inserita una password errata.
    b. **Credenziali valide**: Se le credenziali inserite sono valide (sia l'username che la password sono giusti) viene invocato il metodo switchScene() del Controller che permette di mostrare la finestra "CatalogoLibri" e si esce dal loop.

3.1.2 Registrazione Libro:



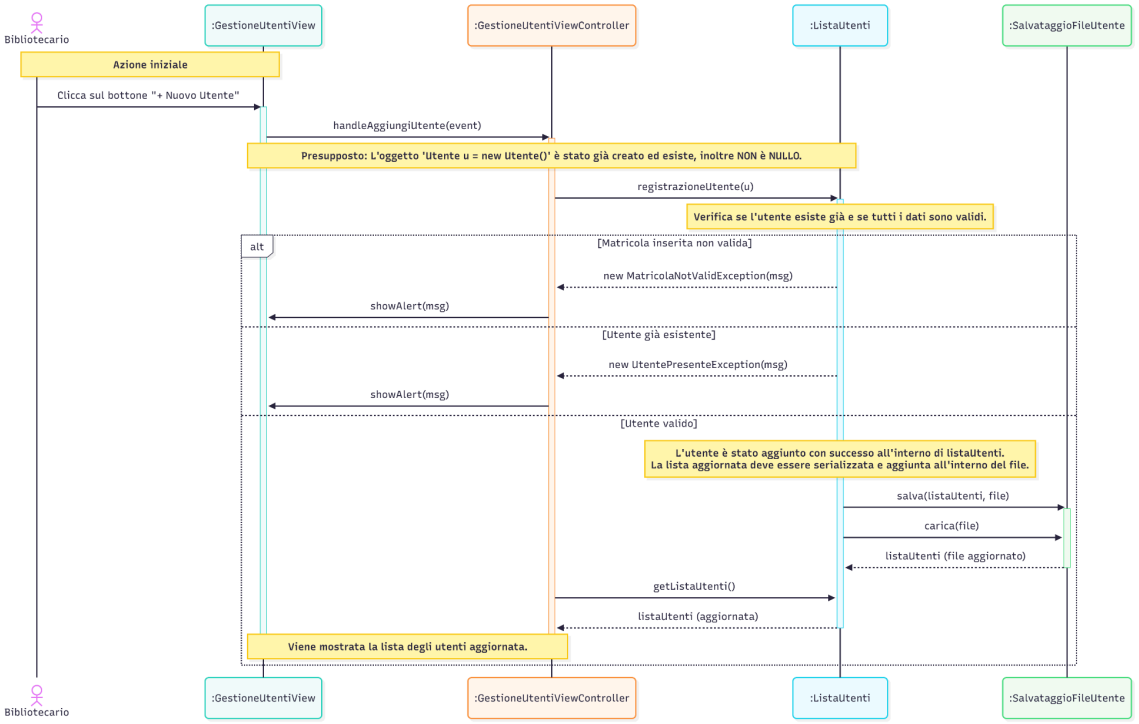
Commento:

Descrizione del diagramma: Questo diagramma descrive il processo per registrare un nuovo libro all'interno del catalogo. La procedura inizia mediante l'azione dell'attore "Bibliotecario".

1. **Azione iniziale**: Il bibliotecario si trova nella finestra "CatalogoLibriView" e clicca sul bottone "+ Nuovo Libro". La View allora attiva il "GestioneLibriViewController" attraverso l'azione "handleAggiungiLibro()".
2. **Gestione della registrazione**: Si suppone che l'oggetto Libro (l) sia già stato istanziato con i dati inseriti dal bibliotecario e non sia nullo. Il Controller quindi, attraverso il messaggio "registrazioneLibro()", passa il controllo al Model "CatalogoLibri".
3. **Gestione delle eccezioni**: Il Model controlla la validità dell ISBN inserito dal bibliotecario per verificare se il libro può essere registrato o meno.
    a. **ISBNNotValidException**: Se l'ISBN non è valido (non è corretto) viene lanciata un'eccezione "ISBNNotValidException" e il Controller mostra nella View un messaggio di errore (showAlert()).
    b. **LibroPresenteException**: Se il libro è già presente allora viene lanciata questa eccezione e il Controller mostra nella View il messaggio di errore attraverso il metodo showAlert().

4. **Registrazione Libro**: Se la registrazione supera i controlli, allora il libro viene effettivamente aggiunto all'interno del catalogo. A questo punto, il controllo passa a "SalvataggioFileLibro", il quale deve serializzare il catalogo aggiornato e caricarlo sul file.
5. **Visualizzazione catalogo aggiornato**: Il Controller chiede al Model il catalogo aggiornato e lo mostra sulla View.

3.1.3 Registrazione Utente:



Commento:

Descrizione del diagramma: Questo diagramma di sequenza illustra il processo per registrare un nuovo utente, che viene attivato dall'azione del bibliotecario.

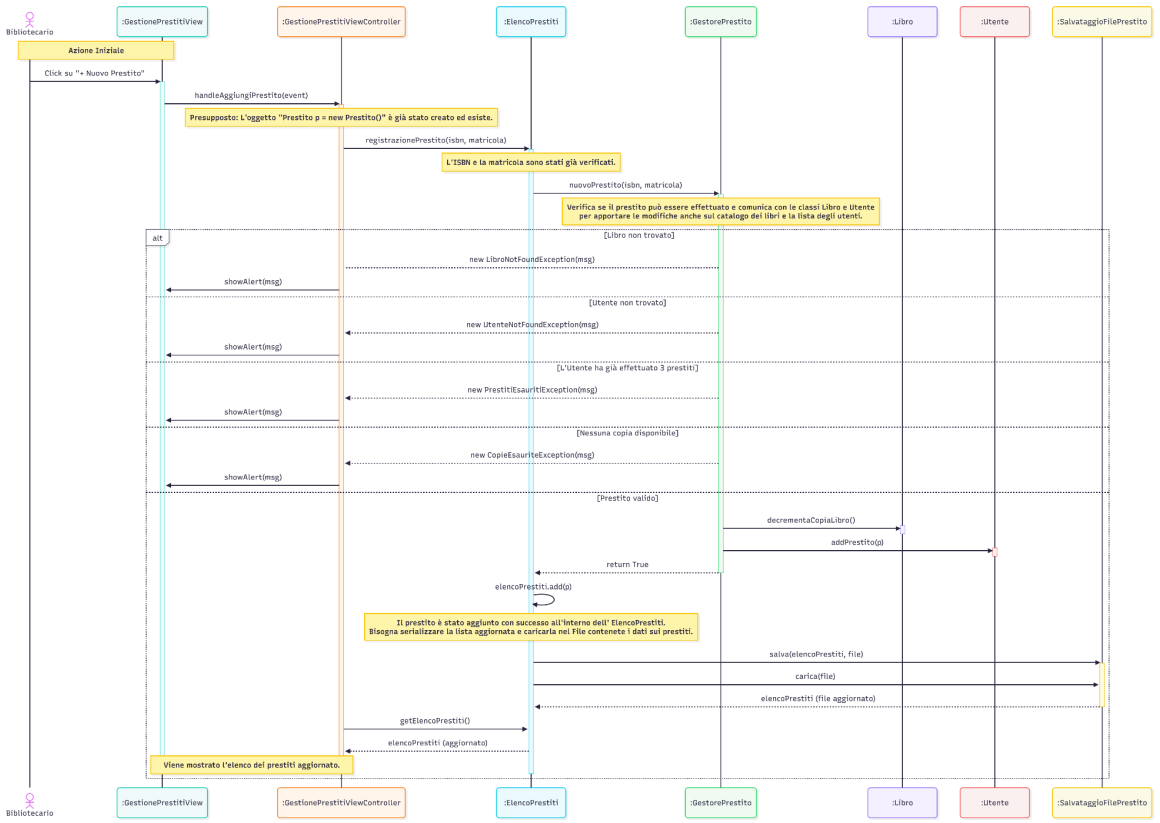
1. **Azione iniziale**: L'attore "Bibliotecario" si trova nella finestra "GestioneUtenteView" e il processo si avvia nel momento in cui clicca il bottone "+ Nuovo Prestito".
2. **Gestione dell'utente**: A questo punto la View dà il controllo al "GestioneUtentiViewController" attraverso la richiesta "handleAggiungiUtente()". Si suppone che l'oggetto Utente (u) sia stato già istanziato con i dati inseriti dal bibliotecario. Quindi il controllo passa al Model "ListaUtenti" attraverso la richiesta "registrazioneUtente()".
3. **Gestione delle Eccezioni**: La registrazione di un utente deve essere rifiutata nel caso in cui il bibliotecario inserisca un utente nullo, una matricola non valida o un

utente già esistente all'interno di "ListaUtenti". In tutti i casi viene lanciata una specifica eccezione:

a. **IllegalArgumentException**: Questa eccezione viene lanciata nel caso in cui il bibliotecario inserisca un utente nullo e il cOntroller mostra un messaggio di errore nella View attraverso il metodo showAlert().
b. **MatricolaNotValidException**: In questo caso viene lanciata un'eccezione dal Model e, quando il Controller la cattura, mostra nella View un messaggio di errore attraverso il metodo showAlert().
c. **UtentePresenteException**: Quando l'utente inserito dal bibliotecario è già presente all'interno della "ListaUtenti" allora viene lanciata questa eccezione e il Controller mostra un messaggio di errore nella View attraverso il messaggio showAlert().

4. **Salvataggio del nuovo utente**: Quando l'utente viene aggiunto con successo all'interno di "ListaUtenti", la lista modificata deve essere serializzata e aggiunta nel file. Quindi il Model passa il controllo a "SalvataggioFileUtente" attraverso i messaggi "salva()" e "carica()".
5. **Visualizzazione della lista aggiornata**: Infine il Controller mostra nella View la lista aggiornata che contiene il nuovo utente registrato.

3.1.4 Registrazione Prestito:



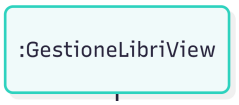
Commento:

Descrizione del diagramma: Questo diagramma di sequenza mostra il processo che viene effettuato per registrare un nuovo prestito e viene avviato dall'azione dell'attore "Bibliotecario".

1. **Azione iniziale**: Il bibliotecario si trova sulla finestra "PrestitiView" e clicca sul bottone "+ Nuovo Prestito". Quindi l'attore interagisce con la View, la quale trasferisce il controllo al "GestionePrestitiViewController" attraverso la richiesta "handleAggiungiPrestito()".
2. **Aggiunta del Prestito**: Si presuppone che l'oggetto Prestito sia stato già istanziato con i dati inseriti dall'attore bibliotecario. Il Controller invia la richiesta dell'aggiunta di un nuovo prestito al Model "ElencoPrestiti" attraverso la chiamata al metodo "registrazionePrestito()".
3. **Gestione del Prestito**: Il Model per verificare se il prestito può essere effettivamente registrato o no dà il controllo al "GestorePrestiti", il quale si occupa proprio di valutare ogni prestito da aggiungere.
4. **Gestione delle Eccezioni**: Esistono 4 casi per cui il prestito richiesto non può essere registrato:
    a. **LibroNotFoundException**: Il bibliotecario inserisce un nome di un libro che non è presente all'interno del "CatalogoLibri". Il Gestore quindi lancia l'eccezione e il Controller mostra nella View un messaggio di errore.
    b. **UtenteNotFoundException**: Il bibliotecario inserisce un utente che non è presente all'interno di "ListaUtenti". Il Gestore allora lancia questa eccezione e il Controller mostra nella View un messaggio di errore.
    c. **PrestitiEsauritiException**: L'utente inserito dal bibliotecario ha già effettuato 3 prestiti e quindi non può richiederne un altro. Il Gestore lancia quindi questa eccezione che viene catturata dal Controller, il quale mostra nella View un messaggio di errore.
    d. **CopieEsauriteException**: Le copie del libro inserito dal bibliotecario non sono sufficienti per effettuare il prestito. Il Gestore allora lancia questa eccezione e il Controller mostra nella View un messaggio di errore.
5. **Aggiornamento delle liste**: Se vengono superati tutti i controlli, il prestito può essere effettuato. In questo caso il Controller richiama l'oggetto Libro per decrementare il numero di copie del libro prestato e l'oggetto Utente per aggiungere il prestito che ha appena effettuato come una sua informazione. Dopo di che il Gestore dà nuovamente il controllo al Model.
6. **Aggiunta del prestito**: Il Model registra il nuovo prestito all'interno di "ElencoPrestiti". Avendo ora l'elenco aggiornato, il Model passa il controllo all'oggetto "SalvataggioFilePrestito", il quale deve serializzare l'elenco modificato e caricarlo all'interno del file.
7. **Visualizzazione dell'elenco aggiornato**: Infine il Controller mostra nella View l'elenco dei prestiti aggiornato col nuovo prestito registrato.

Legenda:

Attore che svolge l'azione iniziale che darà via al processo.



Oggetto della classe GestioneLibriView che comunica con gli oggetti delle altre classi, come l'oggetto che rappresenta il Controller o quello che rappresenta il Model. Ogni rettangolo rappresenta quindi un partecipante del processo.

La freccia continua rappresenta il messaggio che l'oggetto della classe da cui parte (mittente) invia ad un altro oggetto. L'oggetto destinatario può essere un oggetto di un'altra classe oppure l'oggetto mittente stesso (in questo caso la freccia parte ed arriva nello stesso oggetto come nel caso di elencoPrestiti.add()).

La freccia discontinua rappresenta il messaggio di risposta che l'oggetto destinatario può inviare all'oggetto mittente. Non sempre viene utilizzata.

Commento per sottolineare determinate fasi.

LINEA VERTICALE: La linea verticale che parte da ogni oggetto (rettangolo) rappresenta la sua "*linea di vita*". In particolare per tutto il tempo in cui l'oggetto è attivo la linea si evidenzia (linea spessa colorata), per poi ritornare normale quando l'oggetto non è più attivo.

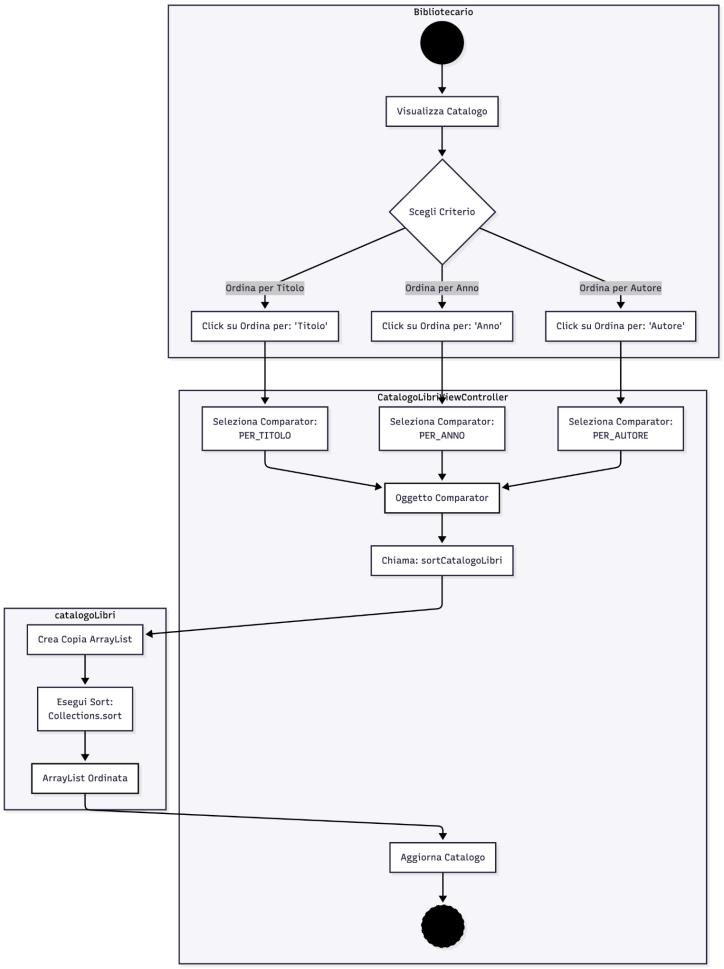
CICLI E CONDIZIONI:

In questi diagrammi si può notare la presenza di un "CICLO LOOP" (vedi 3.1.1) che indica un frammento che viene ripetuto più volte in base ad una determinata condizione.

Inoltre in molti diagrammi sono presenti le "CONDIZIONI ALT" che rappresentano frammenti alternativi multipli (ovvero coincidono all'if-else), ognuno dei quali viene eseguito solo se è soddisfatta la propria condizione.

3.2 Diagrammi delle attività

3.2.1 Ordinamento del Catalogo:



Commento:

Il diagramma di attività illustra il flusso di esecuzione relativo alla funzionalità di ordinamento del catalogo libri.

Attori:

- **Bibliotecario**: Rappresenta l'interazione umana. Il bibliotecario avvia il processo visualizzando il catalogo e sceglie un criterio di ordinamento.
- **CatalogoLibriViewController**: Agisce da intermediario, in base all'input ricevuto dal bibliotecario, invoca la logica relativa.
- **CatalogoLibri**: Esegue l'elaborazione effettiva dei dati, garantendo che l'ordinamento avvenga.

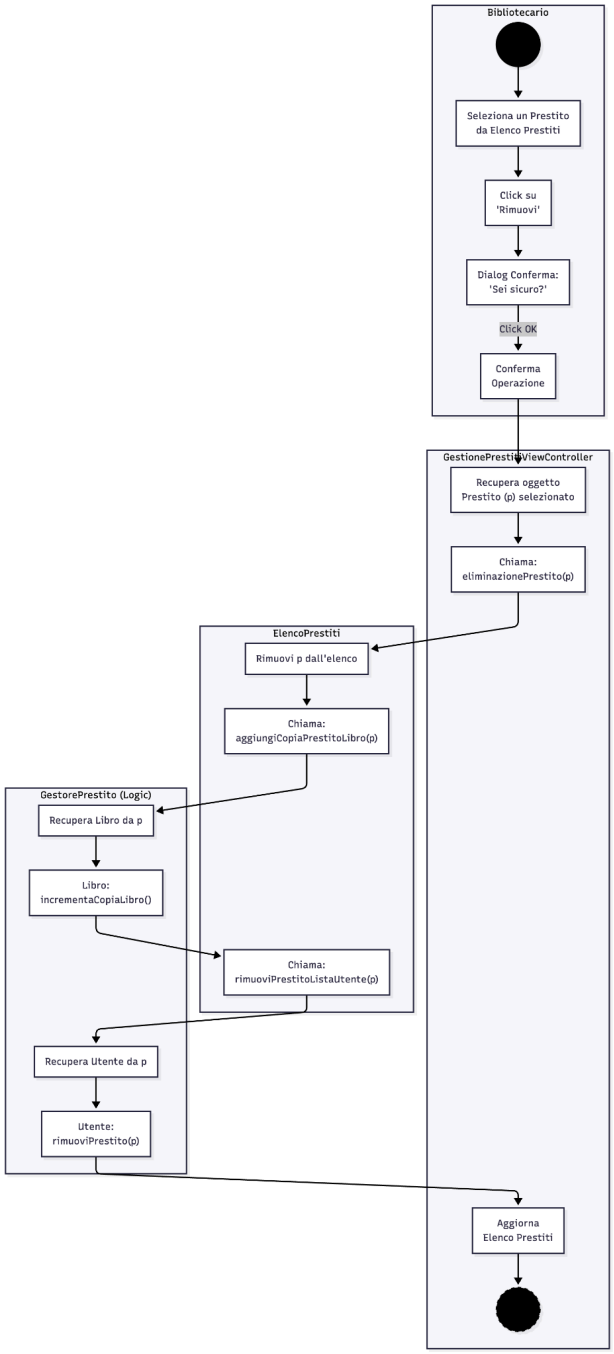
Flusso degli Eventi:

1. **Avvio e Scelta del Criterio**: Il processo inizia con il bibliotecario che visualizza il catalogo ed in base alla scelta del criterio, il flusso si dirama in tre percorsi, ognuno dei quali eseguito in base al pulsante premuto.
2. **Selezione della Strategia**: Il CatalogoLibriViewController intercetta l'evento e, in base alla scelta effettuata, seleziona l'oggetto Comparator corrispondente:
    - PER_TITOLO;
    - PER_ANNO;
    - PER_AUTORE.

    Questi flussi confluiscono in un unico nodo oggetto, che rappresenta il criterio di ordinamento scelto.

3. **Elaborazione dei Dati**: Il controller invoca il metodo sortCatalogoLibri passando il comparatore. A questo punto catalogoLibri entra in controllo dell'esecuzione e svolge le seguenti operazioni:
    - **Creazione e Copia**: Viene creata una copia del TreeSet originale in un ArrayList, per permettere di ordinare la lista.
    - **Ordinamento Lista**: Viene invocato il metodo .sort() sull'arrayList.
    - **Output**: Si ritorna un'ArrayList ordinata.
4. **Aggiornamento della Vista**: L'esecuzione torna al controller che aggiorna il catalogo Libri, presente nell'interfaccia grafica, mostrando i libri nell'ordine selezionato.

3.2.2 Eliminazione Prestito:



Commento:

Il diagramma di attività illustra il flusso di esecuzione relativo alla funzionalità di ordinamento di eliminazione/restituzione di un Prestito.

Attori:

- **Bibliotecario**: Rappresenta l'interazione umana. Il bibliotecario avvia il processo selezionando un prestito e cliccando sul tasto di rimozione.

- **GestionePrestitiViewController**: Agisce da intermediario, recupera l'entità da eliminare e invoca la logica relativa.
- **ElencoPrestiti**: Contiene la lista principale di tutti i prestiti attivi; avvia l'effettiva rimozione e coordina le operazioni esterne aggiuntive.
- **GestorePrestito**: Si occupa di aggiornare lo stato delle entità collegate.

Flusso degli Eventi:

1. **Avvio e Conferma**: Il processo inizia con il Bibliotecario che seleziona un prestito specifico dalla tabella. Al click sul pulsante "Rimuovi", il sistema visualizza un Dialog di Conferma. Se il bibliotecario clicca ok, si continua l'esecuzione, in caso contrario l'operazione viene annullata e si torna alla schermata dell'elenco.
2. **Invocazione**: Il GestionePrestitiViewController recupera l'oggetto Prestito (p) selezionato e invoca il metodo eliminazionePrestito(p).
3. **Aggiornamento**: Questa fase è caratterizzata da un'interazione stretta tra ElencoPrestiti e GestorePrestito:
    - Rimozione dall'elenco: ElencoPrestiti rimuove fisicamente l'oggetto (p) dalla propria lista interna.
    - Aggiornamento Copie Libro: ElencoPrestiti richiede a GestorePrestito di aggiornare il numero di copie del libro restituito, aggiungendone una.
    - Aggiornamento Utente: ElencoPrestiti richiede a GestorePrestito di aggiornare la lista dei prestiti relativa all'utente che ha restituito il prestito, eliminando tale libro.
4. **Chiusura**: Al termine delle operazioni, il controllo ritorna al Controller che mostra l'elenco dei prestiti aggiornato sull'interfaccia grafica.

4. Design dell’interfaccia utente

Il design dell’interfaccia utente è stato progettato seguendo un approccio minimalista, per essere il più semplice e intuitivo possibile, con l’obiettivo di garantire un’esperienza d’uso immediata e priva di distrazioni non necessarie. Si è scelto di eliminare gli elementi superflui, puntando su una struttura chiara e pulita, mantenendo una struttura simile per le varie interfacce.
Dal punto di vista tecnico, la progettazione e la costruzione delle interfacce sono state realizzate mediante SceneBuilder.

4.1 Schermata di Login:

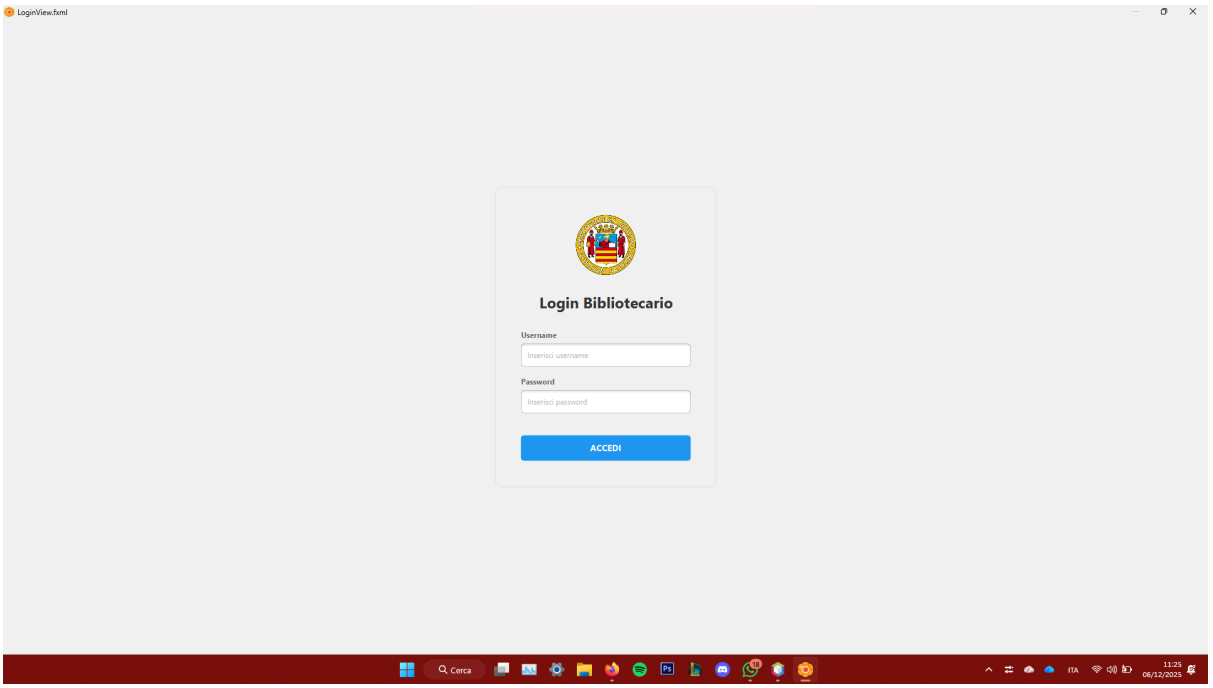


fig. 4.1.1

Le principali interazioni con l’utente:

L’interfaccia del Login presenta un layout essenziale per focalizzare l’attenzione sull’autenticazione. L’utente interagisce con i campi di input per l’inserimento di Username e Password. Successivamente l’attenzione dell’utente è richiamata dal pulsante azzurro di Login, che permette al sistema di verificare se sono stati inseriti dei dati all’interno dei campi citati prima, verificarli e, se corretti, portare l’utente nella dashboard di default. In caso contrario mostrerà un pop-up di alert come feedback.

4.2 Schermata Catalogo Libri:

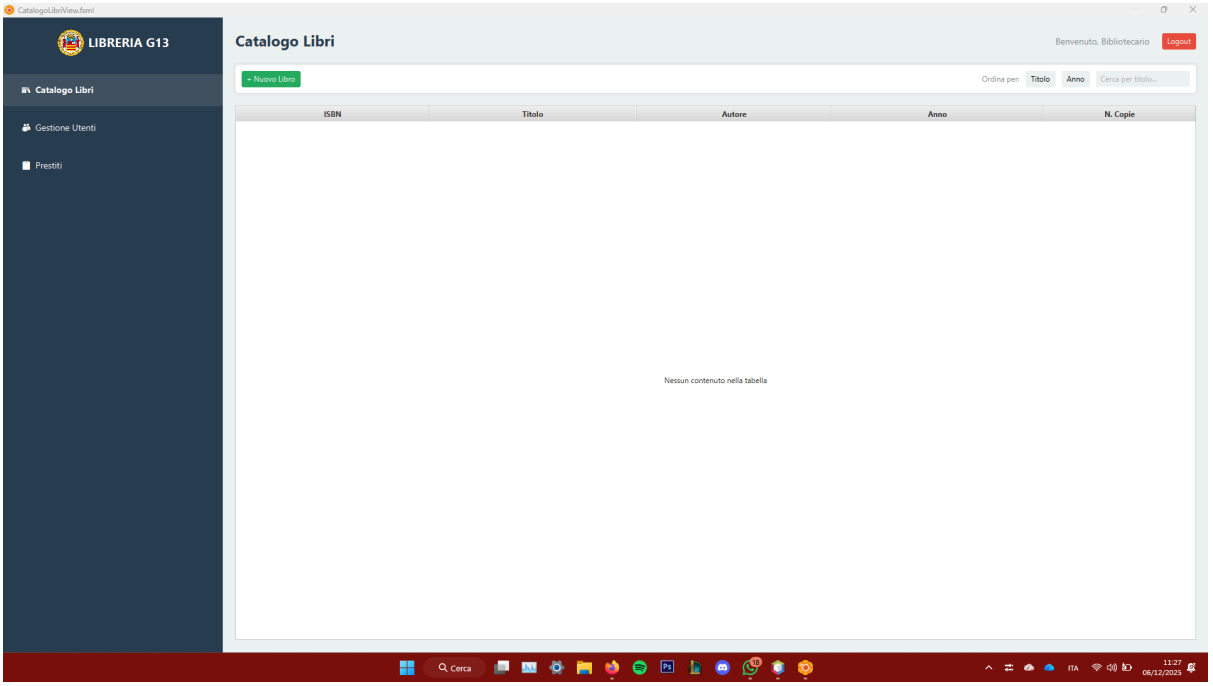


fig. 4.2.1

Le principali interazioni con l'utente:

Anche in questo caso, si è deciso di implementare un'interfaccia (fig. 4.2.1) minimale e intuitiva, in modo tale che un approccio con tale applicazione sia il più semplice possibile. La sezione del catalogo libri è divisa in due parti:

- Un menù pratico che permette all'utente di passare da una sezione all'altra, grazie ai pulsanti "Catalogo Libri", "Gestione Utenti" e "Prestiti", che permettono al sistema di cambiare sezione se premuti.
- La sezione del Catalogo Libri, formato da una barra contenente diversi pulsanti per la gestione del catalogo, come "+ Nuovo Libro" che permette di visualizzare una nuova Scena (fig. 4.2.2) con i vari campi per l'inserimento di un nuovo libro.
    - La scena in fig. 4.2.2 è composta da semplici label, campi e due pulsanti. L'utente può scrivere nei campi i dati relativi al libro che vuole aggiungere e successivamente cliccare il pulsante "Salva Libro" per confermare l'aggiunta del libro all'interno del catalogo. Il metodo collegato a questo pulsante, oltre ad aggiungere il libro nel catalogo (se sono stati inseriti i dati corretti, altrimenti verrà mostrato un errore), va anche a richiamare il metodo di refresh della tabella presente nell'interfaccia della fig. 4.2.1 e a chiudere l'interfaccia dell'aggiunta del nuovo libro. Per quanto riguarda il salvataggio del Catalogo in modo persistente, viene gestito all'interno del metodo di registrazioneLibro().

Sono presenti anche due pulsanti che permettono di avere un ordinamento della tabella per: “Titolo” e “Anno”.
Infine è presente un pulsante rosso di Logout, che permette di terminare la sessione e tornare all’interfaccia di Login (fig. 4.1.1).

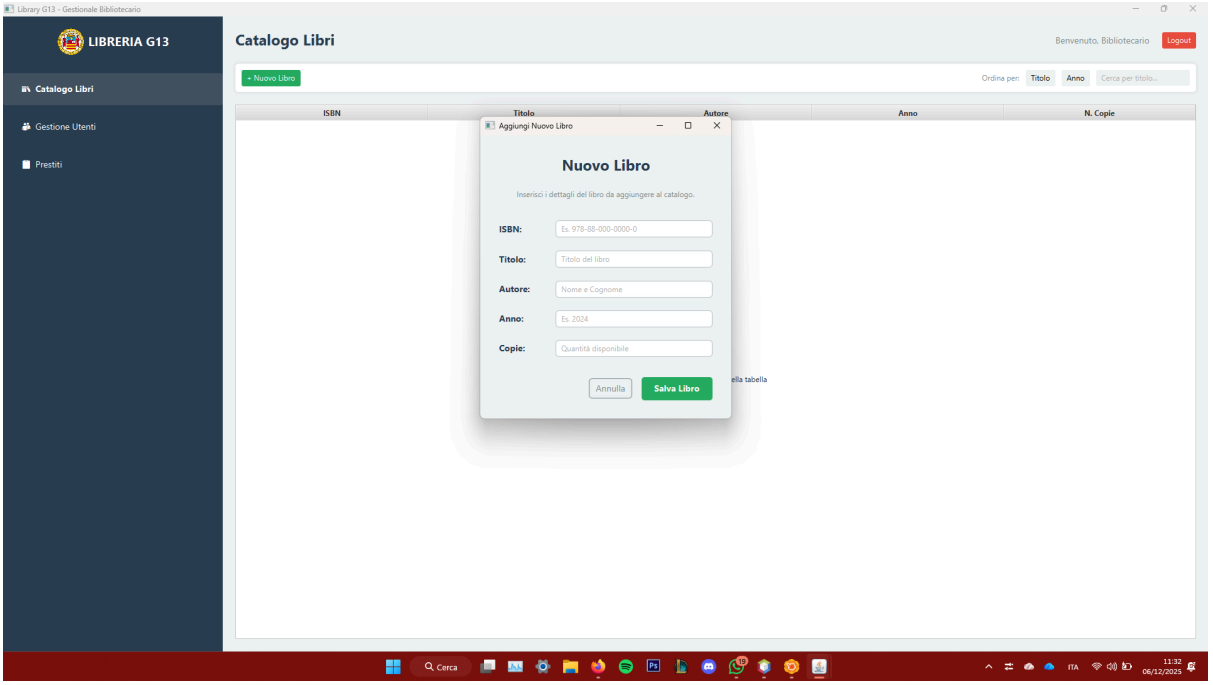


fig. 4.2.2

4.3 Schermata Gestione Utenti:

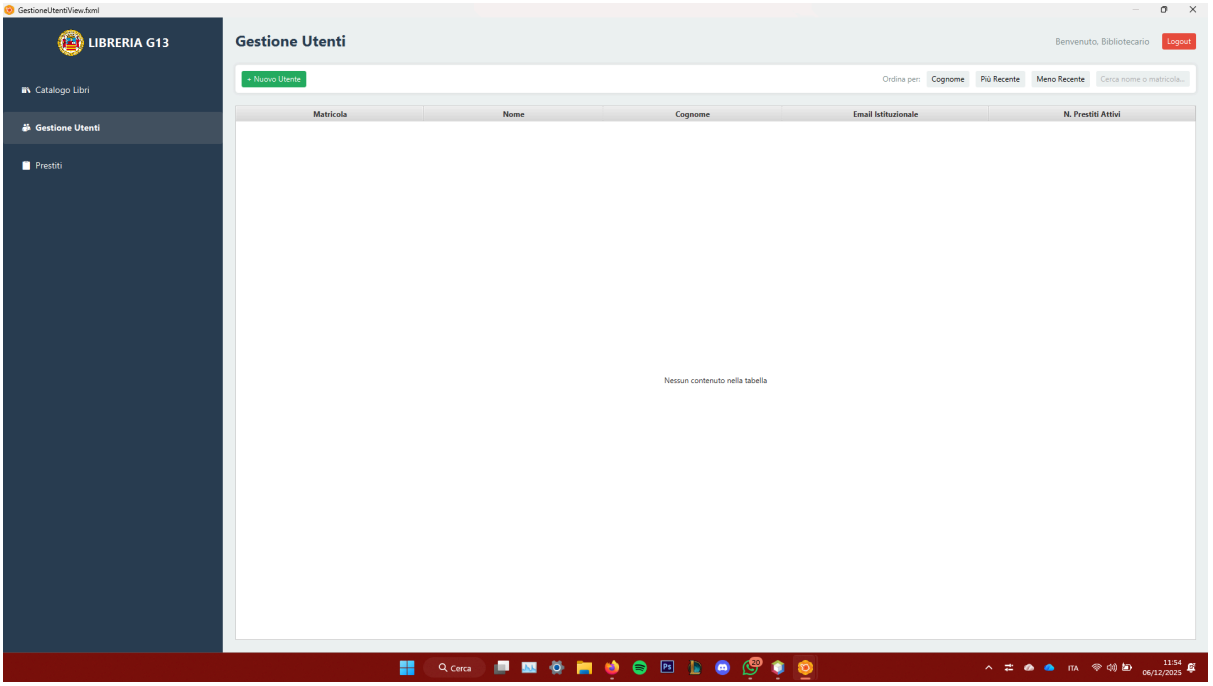


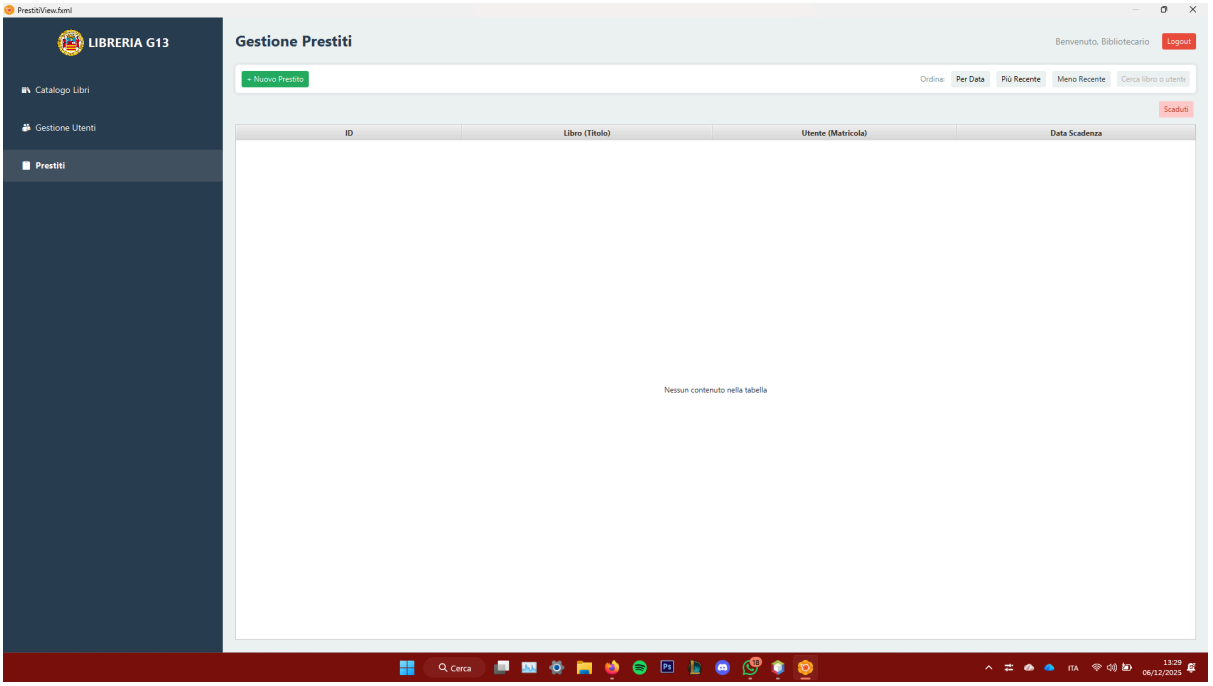
fig. 4.3.1

Le principali interazioni con l'utente:

Mantenendo la coerenza stilistica con la schermata del Catalogo Libri (fig. 4.2.1), la dashboard per la gestione degli utenti (fig 4.3.1) adotta uno stile pulito per facilitare la navigazione. Dal momento che le due sezioni sono state già trattate nel paragrafo della schermata Catalogo Libri (si veda p. 4.2), si passa direttamente alle interazioni che differiscono:

- L'utente può interagire con i pulsanti presenti nella sezione della gestione utenti quali:
    - "+Nuovo Utente" che, come per il catalogo libri, richiama l'interfaccia che permette all'utente di inserire i dati per l'aggiunta di un nuovo utente.
    - Ordinamento per: "Cognome", "Meno Recente" e "Più Recente".
    - Infine è presente un pulsante rosso di Logout, che permette di terminare la sessione e tornare all'interfaccia di Login (fig. 4.1.1).

4.4 Schermata Gestione Prestiti:



Le principali interazioni con l'utente:

Mantenendo la coerenza stilistica con la schermata del Catalogo Libri (fig. 4.2.1), la dashboard per la gestione dei prestiti (fig 4.4.1) adotta uno stile pulito per facilitare la navigazione. Dal momento che le due sezioni sono state già trattate nel paragrafo della schermata Catalogo Libri (si veda p. 4.2), si passa direttamente alle interazioni che differiscono:

- L'utente può interagire con i pulsanti presenti nella sezione della gestione prestiti quali:
    - "+Nuovo Prestito" che, come per il catalogo libri, richiama un'interfaccia che permette all'utente di inserire i dati per l'aggiunta di un nuovo prestito.
    - Ordinamento per: "Per Data", "Meno Recente" e "Più Recente".
    - Vi è un pulsante per visualizzare i prestiti scaduti
    - Infine è presente un pulsante rosso di Logout, che permette di terminare la sessione e tornare all'interfaccia di Login (fig. 4.1.1).