Documento di progetto
Biblioteca Universitaria

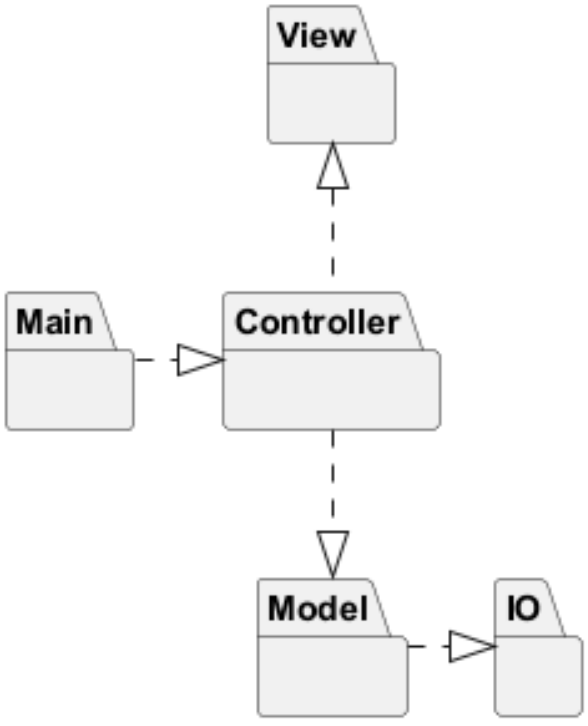
Documento di Progettazione v2.2

Gruppo 3
Francesco Pisaturo – Matr. 0612709758
Giovanni Scelzo – Matr. 0612709505
Matteo Sirignano – Matr. 0612709969
Francesco Vecchione – Matr. 0612709314

1 Architettura del Sistema

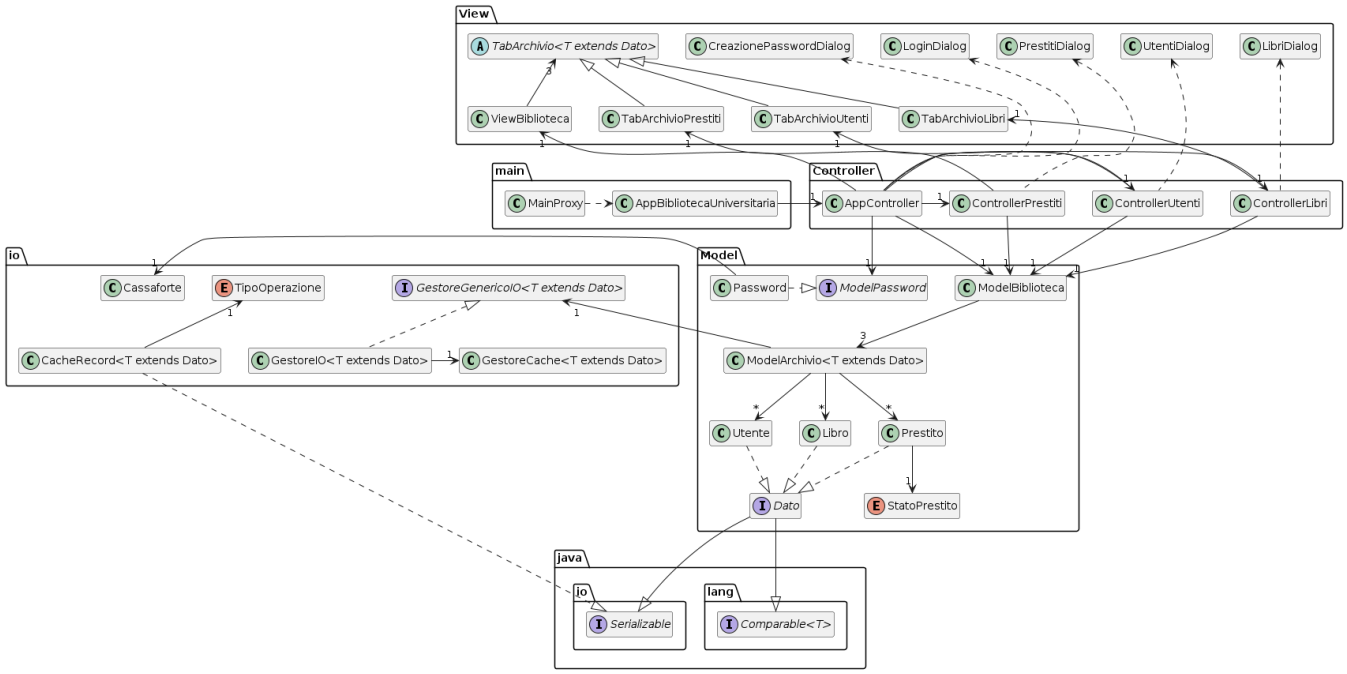
L'architettura scelta per la realizzazione del software per la gestione della biblioteca universitaria è il pattern **Model-View-Controller** in quanto offre una maggiore manutenibilità e modularità del codice, separa le responsabilità (*model* gestisce i dati e la logica, *view* la parte di visualizzazione grafica e *controller* l'interazione dell'utente con i componenti grafici e gli eventi che ne seguono). I package previsti sono:

- **Pattern MVC**
  - package `model`: mantiene i dati dell'archivio e la password di accesso, offrendo i servizi necessari al modulo del Controller per ottenere e manipolare tali dati. Si occupa inoltre di invocare i servizi del package `io` per le operazioni *read/write* su file; esiste quindi una dipendenza tra `model` e `io`.
  - package `view`: si occupa di inizializzare le componenti della vista del software, fornendo i servizi necessari per accedere alle singole parti, difatti inviando i dati raccolti dall'interazione con l'utente al modulo del Controller; poiché non richiama alcuna funzionalità del modulo del Controller, non ha alcuna dipendenza verso il Controller.
  - package `controller`: permette di lanciare le singole parti dell'applicazione, usando poi i servizi che queste offrono per rispondere all'interazione con l'utente; per questo, dipende da Model e da View.
- package `main`: permette di lanciare l'intera applicazione, difatti inizializzando il controller; per questo, l'unica dipendenza che possiede è legata al package Controller.
- package `io`: si occupa di interagire con i file, leggendo o scrivendo da essi i dati passatigli dal modulo del Model; poiché non richiama alcuna funzionalità del modulo del Model, non ha alcuna dipendenza da esso.

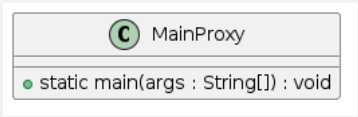


Package Diagram

2 Modello Statico



main.MainProxy



Questa classe rappresenta l'entry point del programma; l'unico servizio che offre è fare da proxy per lanciare il programma. Si è scelto di usare una classe che fa da intermediaria per risolvere un baco riscontrato nell'IDE di utilizzo, ovvero NetBeans, usando una versione di Java superiore ad 8: il baco non permette di lanciare direttamente la classe che estende `Application`, ma ha bisogno di un'altra classe che richiami il suo metodo `main` (che per facilitare la comprensione, è stato rinominato `launchProxy` nell'apposita classe).

Metodi pubblici

- `static main(args)`: lancia la classe `AppBibliotecaUniversitaria`; può ricevere argomenti da riga comando ma non vengono usati in alcun modo dal resto del programma; non ha alcun valore di ritorno.

Relazioni rilevanti

- Dipendenza da `AppBibliotecaUniversitaria`: Il metodo `main(args)` di questa classe richiama il metodo statico `launchProxy(args)`

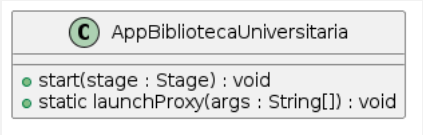
Livello di coesione

- **Coesione Funzionale**: Il modulo ha un unico metodo che svolge un unico compito ben definito.

Livello di accoppiamento

- **Accoppiamento per Dati**: Nonostante `args` sia un parametro che non viene utilizzato in nessun modulo del programma, viene comunque scambiato tra la classe `MainProxy` e la classe `AppBibliotecaUniversitaria` attraverso il metodo `launchProxy`. Solo per questo motivo non si può dire di avere nessun accoppiamento.

main.AppBibliotecaUniversitaria



Questa classe è il launcher dell'applicazione.

Metodi pubblici

- `start(stage)`: Una volta lanciata l'applicazione, si occupa di istanziare `AppController` e passargli lo `stage` principale; accetta come input un oggetto di tipo `Stage`; non ha alcun valore di ritorno.
- `launchProxy(args)`: Metodo delega di `launch` (ereditato da `Application`) che lancia una singola applicazione; gli argomenti che accetta sono passati poi al metodo `launch`; non ha alcun valore di ritorno.

Relazioni rilevanti

- Associazione (unidirezionale) ad `AppController` – molteplicità 1: Sebbene la classe non conservi in senso stretto un'istanza di `AppController`, istanzia comunque (nel metodo `start`) un oggetto dalla classe `AppController`.

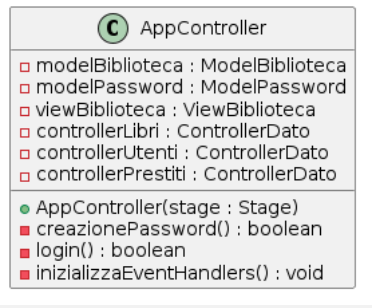
Livello di coesione

- **Coesione Funzionale**: Il modulo contiene il minimo delle operazioni fondamentali per avviare l'applicazione.

Livello di accoppiamento

- **Accoppiamento per Dati**: Il modulo passa all'`AppController` solo lo `stage` principale nel proprio metodo `start`.

controller.AppController



La classe si occupa di lanciare le finestre di accesso e di gestirne gli eventi; successivamente si occupa di lanciare la schermata principale, delegando la gestione degli eventi di quest'ultima alle altre classi di controller.

Attributi principali

- `modelBiblioteca`: Model che si occupa della gestione completa dei dati della biblioteca
- `modelPassword`: Model che si occupa della gestione della password.
- `viewBiblioteca`: View che astrae la finestra principale della pagina dove sono contenute le tab dell'archivio dei libri, degli utenti e dei prestiti.
- `controllerLibri`: Controller specifico responsabile della gestione degli eventi relativi ai libri.
- `controllerUtenti`: Controller specifico responsabile della gestione degli eventi relativi agli utenti.
- `controllerPrestiti`: Controller specifico responsabile della gestione degli eventi relativi ai prestiti.

Metodi pubblici

- `AppController(stage)`: Costruttore che accetta una reference di tipo `Stage` come parametro di input; si occupa di gestire la fase di autenticazione, istanziare gli archivi, istanziare la schermata principale e i controller che la gestiscono.

Relazioni rilevanti

- Associazione (unidirezionale) a `ModelBiblioteca` – molteplicità 3: La classe `AppController` mantiene e gestisce delle reference dell'astrazione dell'archivio per tutti e tre i dati che l'applicazione deve gestire (`Libri, Utenti, Prestiti`).
- Associazione (unidirezionale) a `ModelPassword` – molteplicità 1: La classe `AppController` mantiene e gestisce una reference dell'astrazione della password.
- Associazione (unidirezionale) a `ViewBiblioteca` – molteplicità 1: La classe `AppController` mantiene una reference della finestra principale della biblioteca per poter gestire i suoi componenti grafici.
- Associazione unidirezionale ai controller specifici (`ControllerLibri`, `ControllerUtenti`, `ControllerPrestiti`) – molteplicità 3: `AppController` coordina i controller dedicati ai singoli tipi di dato, delegando loro la gestione degli eventi specifici.

Livello di coesione

- **Coesione Funzionale**: L'unico "metodo" pubblico disponibile all'esterno è il proprio costruttore che si occupa di due compiti che concorrono a gestire l'applicazione: gestire l'autenticazione, istanziare le parti principali e i controller che ne gestiscono gli eventi.

Livello di accoppiamento

- **Accoppiamento per Dati**: Dovendo essere l'intermediario tra `model` e `view`, è necessario che il `Controller` scambi dati con la `view` e che sempre il `controller` scambi dati con il `model`; il controller può per questo motivo anche ricevere dati dal model e passarli alla view e viceversa.

`io.GestoreGenericoIO<T extends Dato>`

Questa interfaccia astrae il concetto di controller e offre un'interfaccia pubblica comune per inizializzare tutti gli event handlers relativi ad una schermata di gestione dati.

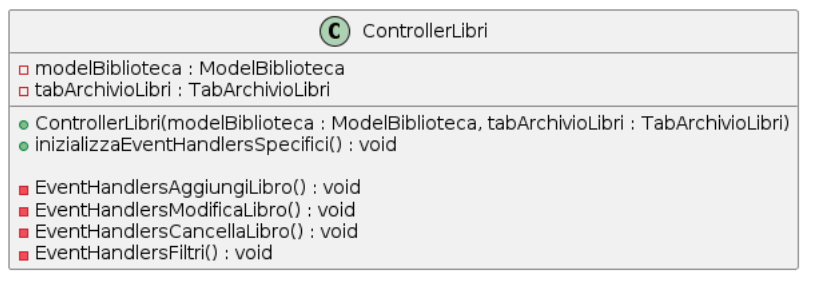
Metodi pubblici

- `inizializzaEventHandlersSpecifici()`: definisce tutti gli event handlers specifii per la tipologia di dato.

Relazioni rilevanti

- L'interfaccia è implementata da tutti i controller concreti (`ControllerLibri`, `ControllerUtenti`, `ControllerPrestiti`).

`controller.ControllerLibri`



La classe si occupa di gestire gli eventi relativi alla gestione dei libri; in particolare è responsabile del lancio delle finestre di pop up di inserimento/modifica libro, dei pop up di conferma e dei messaggi di errore, oltre che la gestione dei filtri di ricerca.

Attributi principali

- `modelBiblioteca`: Model che si occupa della gestione completa dei dati della biblioteca.
- `viewBiblioteca`: View che astrae la finestra principale dell'applicazione e fornisce l'accesso ai suoi componenti grafici (vista libri, pulsanti e filtro di ricerca).

Metodi pubblici

- `ControllerLibri(modelBiblioteca, viewBiblioteca)`: Costruttore riceve le reference di modelBiblioteca e della vista principale e inizializza le componenti atte alla gestione degli eventi.
- `inizializzaEventHandlersSpecifici()`: Metodo che implementa il contratto definito dall'interfaccia `ControllerDato` e che si occupa di inizializzare tutti gli event handlers relativi alle operazioni sui libri.

Relazioni rilevanti

- Associazione (unidirezionale) a `ModelBiblioteca` – molteplicità 1: La classe `AppController` mantiene e gestisce delle reference dell'astrazione dell'archivio della biblioteca.
- Associazione (unidirezionale) a `TabArchivioLibri` – molteplicità 1: La classe `AppController` mantiene una reference della tab relativa ai libri per controllarne le componenti grafiche.

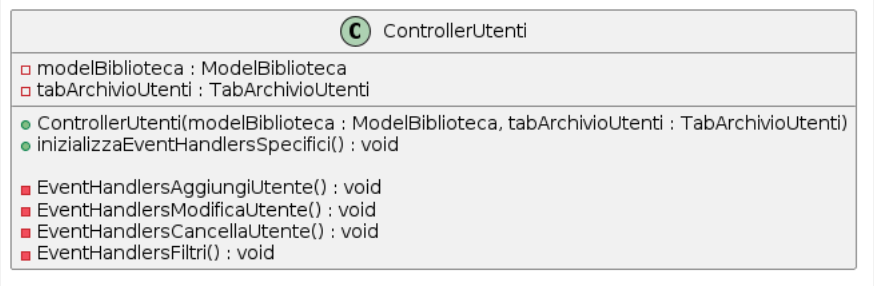
Livello di coesione

- **Coesione Funzionale**: Tutti i metodi sono finalizzati all'unico scopo di gestire tutti gli eventi dietro la finestra di gestione dei libri.

Livello di accoppiamento

- **Accoppiamento per Dati**: Dovendo essere l'intermediario tra `model` e `view`, è necessario che il `Controller` scambi dati con la `view` e che sempre il `controller` scambi dati con il `model`; il controller può per questo motivo anche ricevere dati dal model e passarli alla view e viceversa.

controller.ControllerUtenti



La classe si occupa di gestire gli eventi relativi alla gestione degli utenti; in particolare è responsabile del lancio delle finestre di pop up di registrazione/modifica utente, dei pop up di conferma e dei messaggi di errore, oltre che la gestione dei filtri di ricerca.

Attributi principali

- `modelBiblioteca`: Model che si occupa della gestione completa dei dati della biblioteca.
- `viewBiblioteca`: View che astrae la finestra principale dell'applicazione e fornisce l'accesso ai suoi componenti grafici (vista utenti, pulsanti e filtro di ricerca).

Metodi pubblici

- `ControllerUtenti(modelBiblioteca, viewBiblioteca)`: Costruttore riceve la reference di modelBiblioteca e della vista principale e inizializza le componenti atte alla gestione degli eventi.
- `inizializzaEventHandlersSpecifici()`: Metodo che implementa il contratto definito dall'interfaccia `ControllerDato` e che si occupa di inizializzare tutti gli event handlers relativi alle operazioni sugli utenti.

Relazioni rilevanti

- Associazione (unidirezionale) a `ModelBiblioteca` – molteplicità 1: La classe `AppController` mantiene e gestisce delle reference dell'astrazione dell'archivio centrale della biblioteca.
- Associazione (unidirezionale) a `TabArchivioUtenti` – molteplicità 1: La classe `AppController` mantiene una reference della tab relativa agli utenti per controllarne le componenti grafiche.

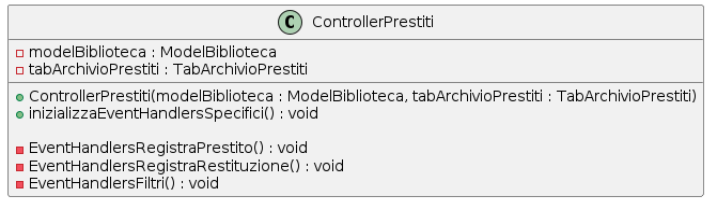
Livello di coesione

- **Coesione Funzionale**: Tutti i metodi sono finalizzati all'unico scopo di gestire tutti gli eventi dietro la finestra di gestione degli utenti.

Livello di accoppiamento

- **Accoppiamento per Dati**: Dovendo essere l'intermediario tra `model` e `view`, è necessario che il `Controller` scambi dati con la `view` e che sempre il `controller` scambi dati con il `model`; il controller può per questo motivo anche ricevere dati dal model e passarli alla view e viceversa.

controller.ControllerPrestiti



La classe si occupa di gestire gli eventi relativi alla gestione dei prestiti; in particolare è responsabile del lancio delle finestre di pop up di registrazione/restituzione prestito, dei pop up di conferma e dei messaggi di errore, oltre che la gestione dei filtri di ricerca.

Attributi principali

- `modelBiblioteca`: Model che si occupa della gestione completa dei dati della biblioteca.
- `viewBiblioteca`: View che astrae la finestra principale dell'applicazione e fornisce l'accesso ai suoi componenti grafici (vista utenti, pulsanti e filtro di ricerca).

Metodi pubblici

- `ControllerUtenti(modelBiblioteca, viewBiblioteca)`: Costruttore riceve le reference di modelBiblioteca e della vista principale e inizializza le componenti atte alla gestione degli eventi.
- `inizializzaEventHandlersSpecifici()`: Metodo che implementa il contratto definito dall'interfaccia `ControllerDato` e che si occupa di inizializzare tutti gli event handlers relativi alle operazioni sui prestiti.

Relazioni rilevanti

- Associazione (unidirezionale) a `ModelBiblioteca` – molteplicità 1: La classe `AppController` mantiene e gestisce delle reference dell'astrazione dell'archivio centrale della biblioteca.
- Associazione (unidirezionale) a `TabArchivioPrestiti` – molteplicità 1: La classe `AppController` mantiene una reference della tab relativa alla gestione dei prestiti.

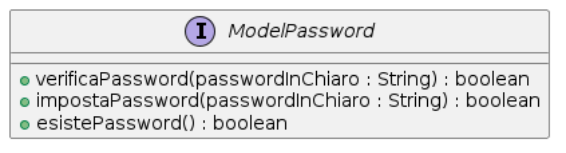
Livello di coesione

- **Coesione Funzionale**: Tutti i metodi sono finalizzati all'unico scopo di gestire tutti gli eventi dietro la finestra di gestione dei prestiti.

Livello di accoppiamento

- **Accoppiamento per Dati**: Dovendo essere l'intermediario tra `model` e `view`, è necessario che il `Controller` scambi dati con la `view` e che sempre il `controller` scambi dati con il `model`; il controller può per questo motivo anche ricevere dati dal model e passarli alla view e viceversa.

`model.ModelPassword`



Quest'interfaccia rappresenta il modello di password accessibile al controller. Il suo compito è di fornire le uniche funzioni che il controller può utilizzare ovvero quella di verificare che la password sia corretta e quella di impostare la password.

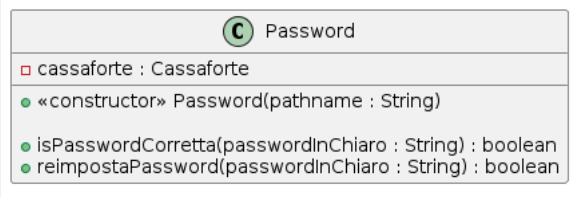
Metodi pubblici

- `verificaPassword(passwordInChiaro)`: verifica che la password inserita come argomento coincida con quella memorizzata in archivio.
- `impostaPassword(passwordInChiaro)`: salva l'hash della password all'interno dell'archivio
- `esistePassword()`: restituisce `true` se la password è registrata correttamente, `false` se non esiste

Relazioni rilevanti

- È associata (unidirezionale) ad `AppController` – molteplicità 1: `AppController` mantiene una reference di tipo `ModelPassword` per poter verificare ed impostare (o reimpostare) la password d'accesso.
- È implementata da `Password`: `Password` è la classe che concretizza l'astrazione fornita da `ModelPassword` e come tale è necessario che la implementi.

`model.Password`



La classe implementa l'interfaccia `ModelPassword` e ne eredita le responsabilità.

Attributi principali

- `cassaforte`: Reference alla classe di IO che si occupa di conservare la password.

Metodi pubblici

- `Password(pathname)`: Costruttore che si occupa di istanziare `Cassaforte` ed assegnarla al proprio attributo.
- `verificaPassword(passwordInChiaro)`: Contratto fornito dall'interfaccia `ModelPassword`.
- `impostaPassword(passwordInChiaro)`: Contratto fornito dall'interfaccia `ModelPassword`.

Relazioni rilevanti

- Implementa `ModelPassword`: La classe implementa i servizi offerti dall'interfaccia `ModelPassword` preoccupandosi dell'interazione con l'IO e rendendo il processo trasparente ad `AppController`.
- Associazione (unidirezionale) a `Cassaforte` – molteplicità 1: Al fine di salvare, verificare e reimpostare la password quando necessario, la classe `Password` necessita di una reference alla `Cassaforte` contenente la password.

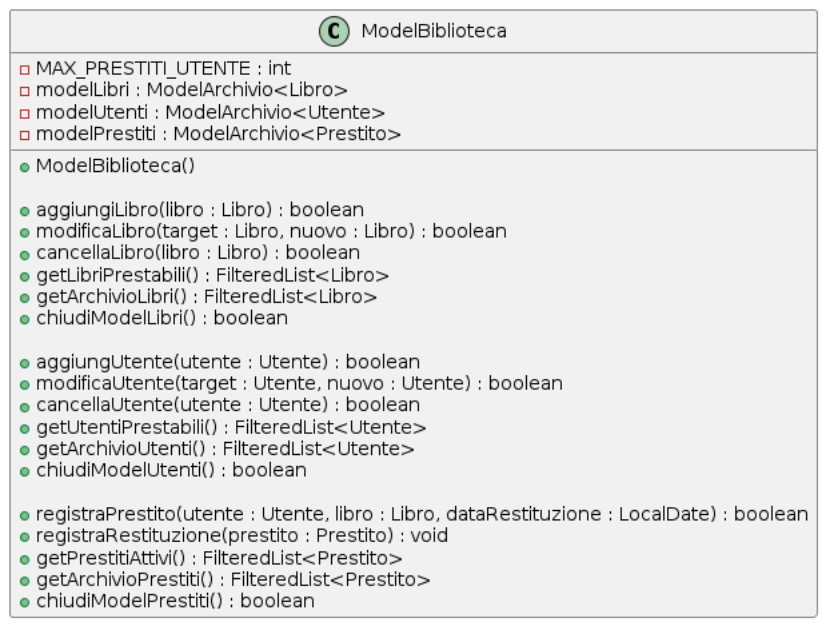
Livello di coesione

- **Coesione Funzionale**: Vengono implementati solo i metodi forniti dall'interfaccia `ModelPassword` e questi riguardano le uniche operazioni permesse per la gestione della password.

Livello di accoppiamento

- **Accoppiamento per Dati**: Vengono richiesti solo la password inserita dall'utente, ovvero il minimo necessario per eseguire le operazioni richieste.

model.ModelBiblioteca



Questa classe rappresenta l'interfaccia del controller con il model del software. L'esigenza di avere un'ulteriore classe per la gestione dei dati nasce dal fatto che alcune funzionalità necessitano della conoscenza di dati diversi (cancella libro necessita di sapere se il libro è attualmente prestato); di conseguenza, essendo queste operazioni, operazioni sui dati, è buona progettazione far si che avvengano in una classe model. Questo permette, nel caso di salvataggio dei dati tramite database relazionale (come da documento SRS), di cambiare solo le implementazioni della classe corrente per far si che tutto continui a funzionare; si ha quindi un codice modulare e facilmente manutenibile. Ulteriore vantaggio, in accordo al design MVC, è quello di tenere distaccate e indipendenti le funzionalità di Controller e View dall'implementazione concreta di Model.

Attributi principali

- `MAX_PRESTITI_UTENTE`: numero massimo di prestiti attivi in carico ad un utente.
- `modelLibri`: reference all'archivio dei libri.
- `modelUtenti`: reference all'archivio degli utenti.
- `modelPrestiti`: reference all'archivio dei prestiti.

Metodi pubblici

- `ModelBiblioteca()`: crea gli archivi dei singoli dati e li apre.
- `metodiDiAggiunta_libro/utente`: metodi delega ai metodi di `ModelArchivio`; ritornano `true` se l'operazione è andata a buon fine, `false` altrimenti.
- `metodiDiModifica_libro/utente`: metodi delega ai metodi di `ModelArchivio`; ritornano `true` se l'operazione è andata a buon fine, `false` altrimenti.
- `cancellaLibro(Libro)`: il metodo cerca di cancellare un libro; si può trovare in due condizioni: una copia del libro è in prestito, quindi azzera le copie presenti in archivio mantenendo l'oggetto (ritorna `false`), non ci sono copie del libro in prestito, quindi viene cancellato l'oggetto relativo al libro (ritorna `true`). Vengono eliminati anche i prestiti restituiti del libro da eliminare.
- `cancellaUtente(Utente)`: il metodo cerca di cancellare un utente; si può trovare in due condizioni: l'utente ha un prestito attivo, quindi l'utente non viene cancellato (ritorna `false`), l'utente non ha prestiti attivi, quindi l'utente viene cancellato (ritorna `true`). Vengono eliminati anche i prestiti restituiti dell'utente da eliminare.
- `getterLibri/utenti_prestabili`: restituiscono una lista dei soli libri/utenti che possono partecipare ad un prestito (utenti con meno di 3 prestiti all'attivo, libri con copie disponibili)

- `getter _archivi`: metodi delega ai metodi di `ModelArchivio`
- `metodi _chiudiArchivio`: metodi delega ai metodi di `ModelArchivio`
- `registraPrestito()`: metodo che inserisce un prestito in archivio dopo aver controllato che l'utente e il libro siano "prestabili"; inoltre si occupa di decrementare le copie disponibili del libro.
- `registraRestituzione()`: metodo che si occupa di cambiare lo stato del prestito in *"RESTITUITO"* e di incrementare le copie disponibili del libro.

Relazioni rilevanti

- È associata (unidirezionale) ad `AppController`: il controller mantiene reference di `ModelBiblioteca` per gestire i dati della biblioteca.
- Associazione (unidirezionale) a `ModelArchivio` – molteplicità *: `ModelBiblioteca` mantiene il reference ai tre archivi di `Libro`, `Utente`, `Prestito`.
- É associata (unidirezionale) ai controller `ControllerDato`: i controller mantengono una reference di `ModelBiblioteca` per gestire i dati della biblioteca.

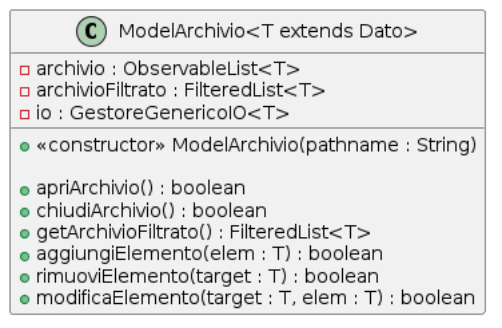
Livello di coesione

- **Coesione Funzionale**: Tutti i servizi offerti dalla classe sono necessari per operare sugli archivi della biblioteca.

Livello di accoppiamento

- **Accoppiamento per Dati**: Il controller scambia il minimo dei dati necessario con la classe per eseguire i servizi offerti.

`model.ModelArchivio`



Questa classe rappresenta il modello di un generico archivio che implementa le funzionalità base (inserimento, modifica, cancellazione) in modo sicuro, ovvero rispettando la validazione `isValid()`. I suoi servizi sono a disposizione della classe `ModelBiblioteca`, a cui fa riferimento il Controller. Le funzionalità possibili sono: aprire e chiudere l'`archivio`, ottenere la lista osservabile (filtrata) dei dati, aggiungere, rimuovere e modificare un elemento dalla lista osservabile. La classe è generica perché può gestire tipi di dati diversi. Avendo imposto il vincolo `T extends Dato`, i tipi concreti utilizzabili sono `Libro, Utente e Prestito`. Gestendo in maniera generica tutti i tipi di dati e offrendo le sole funzionalità base, è possibili sostituire questa classe e tutto il package `IO` con un database relazionale (come da documento SRS), e cambiare la sola classe di interfaccia, ovvero `ModelBiblioteca`.

Attributi principali

- `archivio`: Lista osservabile che astrae l'archivio.
- `archivioFiltrato`: Lista ordinata e filtrabile che viene visualizzata dalle tabelle.
- `io`: La reference alla cassaforte che si occupa di custodire la password.

Metodi pubblici

- `ModelArchivio(pathname)`: istanzia `GestoreIO` e inizializza le strutture che astraggono l'archivio.
- `apriArchivio()`: richiede a `GestoreIO` l'archivio salvato su file; ritorna un valore booleano che indica se il caricamento dei dati è avvenuto con successo. In caso di ritorno di `false`, è anche possibile che il caricamento dell'archivio sia avvenuto aggiornando l'archivio tramite i record presenti nella cache; ciò è sintomo di una non corretta chiusura del programma nell'esecuzione precedente.
- `chiudiArchivio()`: richiede a `GestoreIO` di salvare l'archivio in modo corretto l'archivio su file. Ritorna un valore boolean che indica se l'operazione è avvenuta con successo.
- `getArchivioFiltrato()`: Metodo accessor per `archivioFiltrato`.
- `aggiungiElemento(elem)`: Metodo delega per l'operazione di aggiunta di un elemento all'archivio, che richiede a `GestioreIO` l'aggiunta dell'operazione in cache. Ritorna un valore booleano che indica se l'operazione è avvenuta con successo. L'elemento viene aggiunto alla `ObservableList` che automaticamente aggiorna la `FilteredList`.
- `rimuoviElemento(target)`: Metodo delega per la rimozione di un elemento dall'archivio selezionato dalla tabella che richiede a `GestioreIO` l'aggiunta dell'operazione in cache. Ritorna un valore booleano che indica se l'operazione è avvenuta con successo. L'elemento viene aggiunto alla `ObservableList` che automaticamente aggiorna la `FilteredList`.
- `modificaElemento(target, elem)`: Metodo per la modifica di un elemento selezionato dalla tabella (`target`) nell'archivio che richiede a `GestioreIO` l'aggiunta dell'operazione in cache. Ritorna un valore booleano che indica se l'operazione è avvenuta con successo. L'elemento viene aggiunto alla `ObservableList` che automaticamente aggiorna la `FilteredList`.

Relazioni rilevanti

- È associata (unidirezionale) ad `AppController`: il controller mantiene reference di `ModelArchivio` per ogni tipo di dato gestito (`Libro`, `Utente`, `Prestito`).
- Associazione (unidirezionale) a `Libro` – molteplicità *: contiene l'archivio dei libri.
- Associazione (unidirezionale) a `Utente` – molteplicità *: contiene l'archivio degli utenti.
- Associazione (unidirezionale) a `Prestito` – molteplicità *: contiene l'archivio dei prestiti.
- Associazione (unidirezionale) a `GestoreGenericoIO<T>` – molteplicità 1: mantiene una reference al gestore di IO per leggere/scrivere dati.

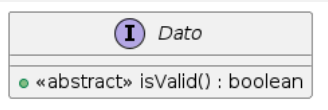
Livello di coesione

- **Coesione Funzionale**: Tutti i servizi offerti dalla classe sono necessari per operare sull'archivio.

Livello di accoppiamento

- **Accoppiamento per Dati**: Il controller scambia il minimo dei dati necessario con la classe per eseguire i servizi offerti.

`model.Dato`



Questa interfaccia astrae il concetto di dato che la biblioteca deve gestire.

Metodi pubblici

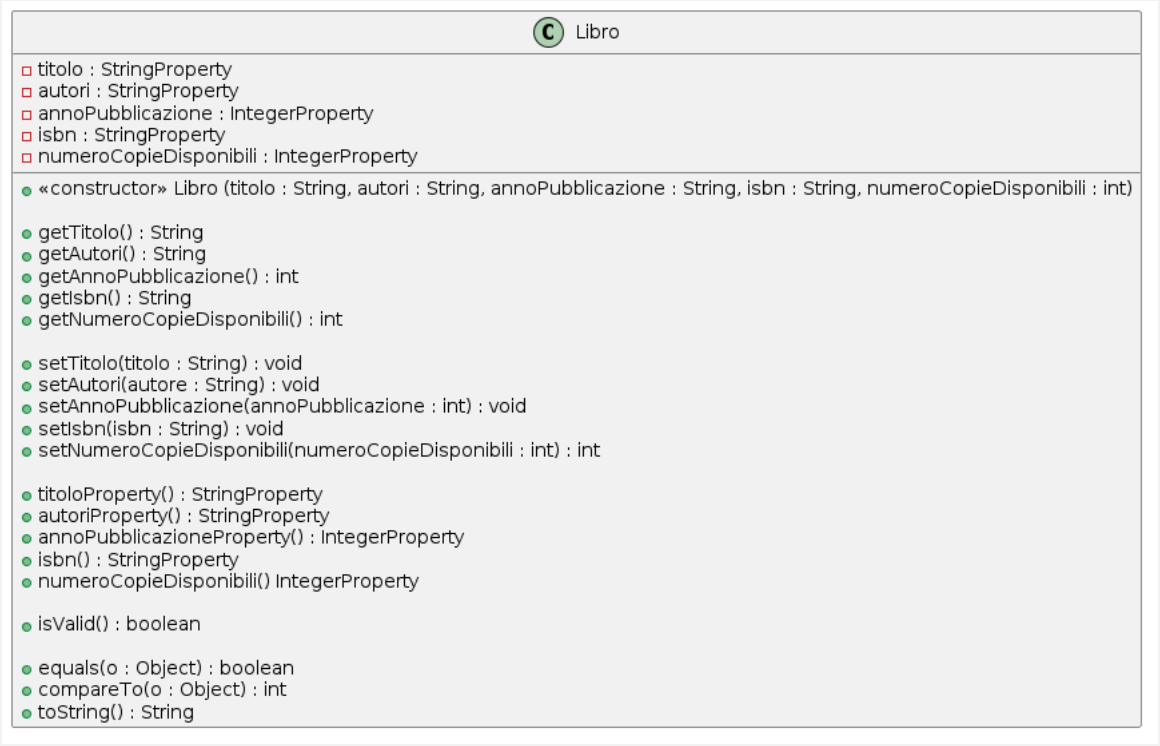
- `isValid()`: Metodo invocato su una particolare istanza per verificare se tale istanza è corretta secondo i

principi della classe di appartenenza.

Relazioni rilevanti

- È implementata da `Libro`: `Libro` è uno dei tre tipi di dato gestiti dalla biblioteca.
- È implementata da `Utente`: `Utente` è uno dei tre tipi di dato gestiti dalla biblioteca.
- È implementata da `Prestito`: `Prestito` è uno dei tre tipi di dato gestiti dalla biblioteca.
- Estende l'interfaccia `Comparable<T>`: permette di definire una relazione d'ordine sui dati; le classi che implementano `Dato` devono quindi implementare anche `Comparable`.
- Estende l'interfaccia `Serializable`: i dati devono poter essere salvati su file in modo serializzabile.

`model.Libro`



Questa classe astrae l'oggetto libro.

Attributi principali

- `titolo`: Titolo del libro.
- `autori`: Lista di uno o più autori separati da una virgola.
- `annoPubblicazione`: Anno di pubblicazione del libro.
- `isbn`: Codice identificativo univoco del libro. Deve essere formato da 13 cifre le cui prime tre devono necessariamente essere 978 (o 979).
- `numeroCopieDisponibili`: Numero di copie disponibili del libro.

Metodi pubblici

- `Libro(titolo, autori, annoPubblicazione, isbn, numeroCopieDisponibili)`: Costruttore che si occupa di inizializzare gli attributi.
- `getter per tutti gli attributi`: Getter per i valori degli attributi.
- `getter specifici per le property per tutti gli attributi`: Getter per le property degli attributi.
- `setter per tutti gli attributi`: Setter per i valori degli attributi.

- `isValid()`: Contratto fornito dall'interfaccia `Dato`. Un oggetto `Libro` è valido se ha un numero di copie maggiore di zero e l'`isbn` rispetta il formato standard.
- `equals(o)`: Contratto fornito dalla classe `Object`. Due oggetti `Libro` sono uguali se hanno `isbn` uguale.
- `compareTo(o)`: Contratto fornito dall'interfaccia `Comparable<T>`. Due oggetti `Libro` sono confrontati sulla base del `titolo`.
- `toString()`: Contratto fornito dalla classe `Object`.

Relazioni rilevanti

- È associata (unidirezionale) a `ModelArchivio<T extends Dato>`: `ModelArchivio` mantiene una struttura dati che può essere di uno dei tre tipi di dato gestiti dalla biblioteca (`Libro`, `Utente`, `Prestito`).
- Implementa l'interfaccia `Dato`: Un `Libro` è un dato che viene gestito dalla biblioteca.

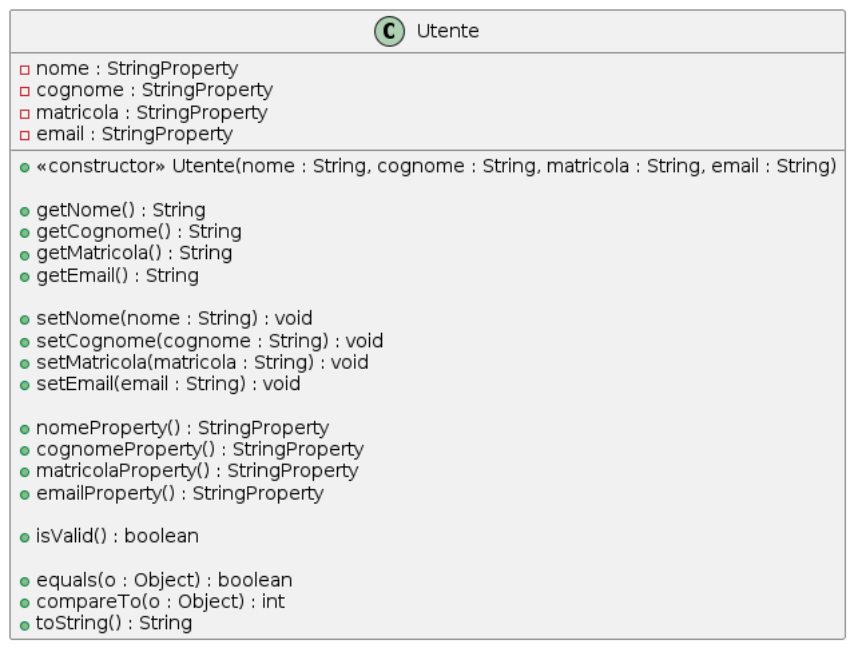
Livello di coesione

- **Coesione Funzionale**: Vengono forniti i metodi e mantenuti gli attributi che servono unicamente a creare e modificare un `Libro`, mantenere un archivio di libri ordinato e visualizzabile da una tabella.

Livello di accoppiamento

- **Accoppiamento per Dati**: Questa classe accetta e fornisce solo lo stretto necessario dei dati richiesti dai singoli servizi che offre.

`model.Utente`



Questa classe astrae l'oggetto `Utente`.

Attributi principali

- `nome`: Nome dell'utente.
- `cognome`: Cognome dell'utente.
- `matricola`: Matricola universitaria dell'utente. Deve essere composta da 10 cifre.
- `email`: Email istituzionale dell'università. Formato: prima lettera del nome, punto, cognome, eventuale numero, @studenti.unisa.it (esempio: f.vecchione12@studenti.unisa.it).

Metodi pubblici

- `Utente(nome, cognome, matricola, email)`: Costruttore che si occupa di inizializzare gli attributi.
- `getter per tutti gli attributi`: Getter per i valori degli attributi.
- `getter specifici per le property per tutti gli attributi`: Getter per le property degli attributi.
- `setter per tutti gli attributi`: Setter per i valori degli attributi.
- `isValid()`: Contratto fornito dall'interfaccia `Dato`. Un oggetto `Utente` è valido se la `matricola` rispetta lo standard e la `email` ha formato istituzionale.
- `equals(o)`: Contratto fornito dalla classe `Object`. Due oggetti `Utente` sono uguali se hanno la stessa `matricola`.
- `compareTo(o)`: Contratto fornito dall'interfaccia `Comparable<T>`. Due oggetti `Utente` sono confrontati sulla base del `cognome` e quindi del `nome`.
- `toString()`: Contratto fornito dalla classe `Object`.

Relazioni rilevanti

- È associata (unidirezionale) a `ModelArchivio<T extends Dato>`: `ModelArchivio` mantiene una struttura dati che può essere di uno dei tre tipi di dato gestiti dalla biblioteca (`Libro`, `Utente`, `Prestito`).
- Implementa l'interfaccia `Dato`: Un `Utente` è un dato gestito dalla biblioteca.

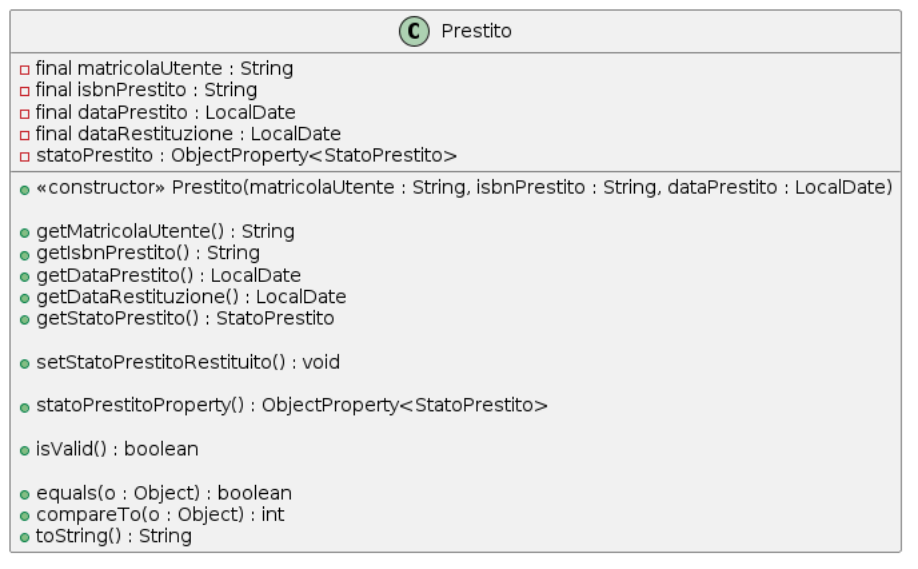
Livello di coesione

- **Coesione Funzionale**: Vengono forniti i metodi e mantenuti gli attributi che servono unicamente a creare e modificare un `Utente`, mantenere un archivio di utenti ordinato e visualizzabile da una tabella.

Livello di accoppiamento

- **Accoppiamento per Dati**: Questa classe accetta e fornisce solo lo stretto necessario dei dati richiesti dai singoli servizi che offre.

model.Prestito



Questa classe astrae l'oggetto `Prestito`.

Attributi principali

- `matricolaUtente`: Matricola dell'utente che ha richiesto il prestito.

- `isbnPrestito`: Libro richiesto in prestito dall'utente.
- `dataPrestito`: Data della richiesta del prestito.
- `dataRestituzione`: Data prevista per la restituzione del prestito.
- `statoPrestito`: Flag che indica lo stato del prestito (`"ATTIVO"` o `"RESTITUITO"`).

Metodi pubblici

- `Prestito(matricolaUtente, isbnPrestito, dataPrestito)`: Costruttore che si occupa di inizializzare gli attributi.
- `getter per tutti gli attributi`: Getter per i valori degli attributi.
- `getter specifici per le property per tutti gli attributi`: Getter per le property degli attributi.
- `setter per tutti gli attributi`: Setter per i valori degli attributi.
- `isValid()`: Contratto fornito dall'interfaccia `Dato`. Un oggetto `Prestito` è valido se l'utente ha un numero di libri in prestito inferiore a 3.
- `equals(o)`: Contratto fornito dalla classe `Object`. Due oggetti `Prestito` sono uguali se hanno stessa `matricolaUtente` e stesso `isbnPrestito`.
- `compareTo(o)`: Contratto fornito dall'interfaccia `Comparable<T>`. Due oggetti `Prestito` sono confrontati sulla base della `dataRestituzione`.
- `toString()`: Contratto fornito dalla classe `Object`.

Relazioni rilevanti

- È associata (unidirezionale) a `ModelArchivio<T extends Dato>`: `ModelArchivio` mantiene una struttura dati di tipo `Prestito`.
- Associazione (unidirezionale) a `StatoPrestito` – molteplicità 1: La classe mantiene l'informazione dello stato del prestito.
- Implementa l'interfaccia `Dato`: Un `Prestito` è un dato gestito dalla biblioteca.

Livello di coesione

- **Coesione Funzionale**: Vengono forniti i metodi e mantenuti gli attributi che servono unicamente a creare e modificare un `Prestito`, mantenere un archivio di prestiti ordinato e visualizzabile da una tabella.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e fornisce solo lo stretto necessario dei dati richiesti dai servizi che offre.

`model.StatoPrestito`

Questa enumerazione elenca gli stati possibili di un prestito.

Attributi principali

- `ATTIVO`: Istanza che codifica lo stato "attivo" del prestito.
- `RESTITUITO`: Istanza che codifica lo stato "restituito" del prestito.

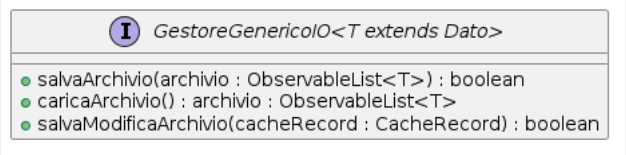
Metodi pubblici

- I metodi presenti appartengono al contratto base di una enumerazione.

Relazioni rilevanti

- È associata (unidirezionale) a `Prestito`: Un prestito ha uno stato che può essere o `ATTIVO` o `RESTITUITO`.

io.GestoreGenericoIO<T extends Dato>



Questa interfaccia astrae il concetto di gestore generico di IO. Fornisce i servizi necessari al modello di archivio per salvare le modifiche fatte all'archivio effettivo nella cache, caricare e salvare l'archivio in maniera trasparente rispetto a crash e chiusure inaspettate dell'applicazione.

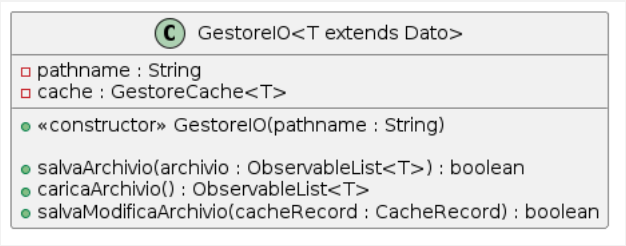
Metodi pubblici

- `salvaArchivio(archivio)`: Salva in un file l'archivio passato come input; ritorna un valore booleano che indica se l'operazione sia andata a buon fine.
- `caricaArchivio()`: Recupera l'archivio da file, gestendo eventuali chiusure non corrette della precedente esecuzione; ritorna un valore booleano che indica il successo dell'operazione.
- `salvaModificaArchivio(cacheRecord)`: Salva sulla cache la modifica dell'archivio passata in input; ritorna un valore booleano che indica se l'operazione sia andata a buon fine.

Relazioni rilevanti

- È associata (unidirezionale) a `ModelArchivio<T extends Dato>`: ModelArchivio mantiene una reference al gestore generico IO per garantire i propri servizi.
- È implementata da `GestoreIO<T extends Dato>`: La classe concreta concretizza l'interfaccia implementandone i metodi.

io.GestoreIO<T extends Dato>



Questa classe concretizza l'interfaccia `GestoreGenericoIO`.

Attributi principali

- `pathname`: Stringa che mantiene il percorso del file di archivio.
- `cache`: Reference ad un oggetto di tipo `GestoreCache` per utilizzare i relativi servizi.

Metodi pubblici

- `GestoreIO(pathname)`: Costruttore che istanzia l'oggetto `GestoreCache` e memorizza il pathname dell'archivio.
- `salvaArchivio(archivio)`: Contratto fornito dall'interfaccia `GestoreGenericoIO`.
- `caricaArchivio()`: Contratto fornito dall'interfaccia `GestoreGenericoIO`.
- `salvaModificaArchivio(cacheRecord)`: Contratto fornito dall'interfaccia `GestoreGenericoIO`.

Relazioni rilevanti

- Implementa `GestoreGenericoIO<T extends Dato>`: La classe implementa i servizi dell'interfaccia gestendo l'interazione con i file di archivio e cache in modo trasparente rispetto a `ModelArchivio`.
- Associazione (unidirezionale) a `GestoreCache<T extends Dato>` – molteplicità 1: Mantiene una reference al gestore della cache per l'archivio.

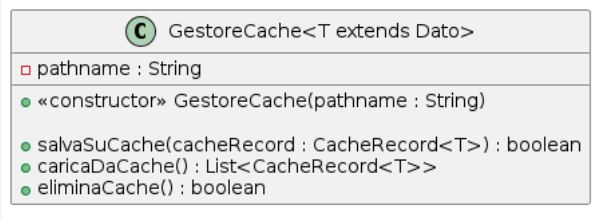
Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi servono esclusivamente a gestire le interazioni con archivi file e cache.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e fornisce solo i dati strettamente necessari ai servizi che offre.

`io.GestoreCache<T extends Dato>`



Questa classe rappresenta un gestore della cache. Per cache s'intende un file dove vengono salvate le operazioni svolte sull'archivio della biblioteca durante l'esecuzione del programma. La cache viene creata all'inizio dell'esecuzione e cancellata alla fine; se il file esiste da esecuzioni precedenti significa che il programma non è stato chiuso correttamente.

Attributi principali

- `pathname`: Stringa che mantiene il percorso del file di cache.

Metodi pubblici

- `GestoreCache(pathname)`: Costruttore che salva una copia del pathname passato in input.
- `salvaSuCache(cacheRecord)`: Salva su cache un aggiornamento qualsiasi dell'archivio; ritorna `boolean` per indicare se l'operazione ha avuto successo.
- `caricaDaCache()`: Recupera la lista dei record contenuti nella cache; restituisce `null` se il file è vuoto
- `eliminaCache()`: Elimina il file di cache associato all'oggetto corrente; ritorna `boolean` per indicare se l'operazione ha avuto successo.

Relazioni rilevanti

- È associata (unidirezionale) a `GestoreIO<T extends Dato>`: `GestoreIO` mantiene una reference alla cache per adempiere ai propri servizi.

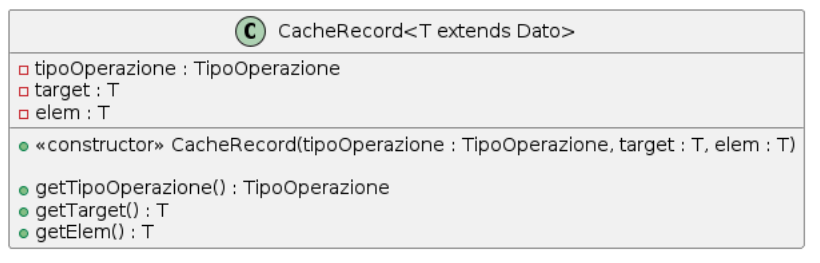
Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi servono esclusivamente a gestire le interazioni con il file di cache.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e fornisce solo i dati strettamente necessari ai servizi che offre.

io.CacheRecord<T extends Dato>



Questa classe rappresenta un record della cache. Un record della cache è una notifica di aggiornamento dell'archivio che può essere un'aggiunta, una cancellazione o una modifica.

Attributi principali

- `tipoOperazione`: indica il tipo di operazione svolta.
- `target`: Elemento cancellato o modificato; `null` in caso di aggiunta.
- `elem`: Elemento aggiunto o che modifica un elemento esistente; `null` in caso di cancellazione.

Metodi pubblici

- `CacheRecord(tipOperazione, target, elem)`: Costruttore che inizializza gli attributi del record.
- Getter per tutti gli attributi: Permettono di ottenere i valori degli attributi `tipOperazione`, `target` e `elem`.

Relazioni rilevanti

- Implementa `Serializable`: I record della cache devono poter essere salvati su file in modo serializzabile.

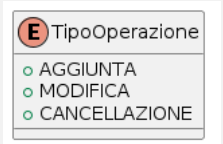
Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi servono esclusivamente ad identificare un record della cache attraverso l'operazione svolta e gli elementi coinvolti.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e fornisce solo i dati strettamente necessari ai servizi che offre.

io.TipoOperazione



Questa enumerazione elenca le operazioni possibili su un dato.

Attributi principali

- `AGGIUNTA`
- `MODIFICA`
- `CANCELLAZIONE`

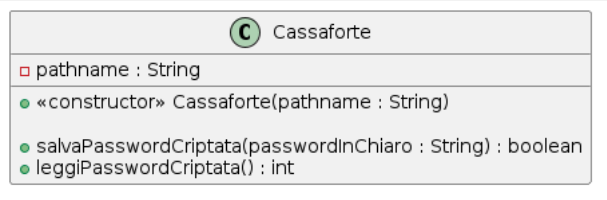
Metodi pubblici

- I metodi presenti appartengono al contratto base di una enumerazione.

Relazioni rilevanti

- È associata (unidirezionale) a `CacheRecord`: Un `CacheRecord` ha uno stato che può essere o `AGGIUNTA`, o `MODIFICA` o `CANCELLAZIONE`.

io.Cassaforte



Questa classe astrae il concetto di cassaforte dove è mantenuta la password. Si occupa di salvarla criptandola su file e di restituirla.

Attributi principali

- `pathname`: Stringa che mantiene il percorso del file dove è salvata la password.

Metodi pubblici

- `Cassaforte(pathname)`: Costruttore che salva una copia del `pathname` passato in input.
- `salvaPasswordCriptata(passwordInChiaro)`: Cripta la password e la salva su file. Ritorna un booleano per indicare il successo dell'operazione.
- `leggiPasswordCriptata()`: Recupera la password criptata dal file e la restituisce al chiamante.

Relazioni rilevanti

- È associata (unidirezionale) a `Password`: La classe `Password` mantiene una reference a `Cassaforte` per poter adempiere ai propri servizi.

Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi sono strettamente necessari per compiere le operazioni di base su una cassaforte.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e fornisce solo i dati strettamente necessari ai servizi che offre.

view.ViewBiblioteca

Questa classe astrae la vista principale dell'applicazione, contenente le tre tab per i corrispondenti archivi (libri, utenti e prestiti). Si occupa di inizializzare il layout generale e le singole tab, passando a loro i relativi archivi sotto forma di `FilteredList`.

Attributi principali

- `stage`: Lo stage della vista principale.
- `tabLibri`: La tab della gestione dell'archivio dei libri.
- `tabUtenti`: La tab della gestione dell'archivio degli utenti.
- `tabPrestiti`: La tab della gestione dell'archivio dei prestiti.

Metodi pubblici

- `ViewBiblioteca(stage, listaOsservabileLibri, listaOsservabileUtenti, listaOsservabilePrestiti)`: Costruttore che istanzia gli attributi dell'oggetto ed imposta il layout generale della vista principale.
- `getter` per tutti gli attributi: Permettono di accedere alle reference mantenute dagli attributi.

Relazioni rilevanti

- È associata (unidirezionale) a `AppController`: Il controller mantiene una reference a `ViewBiblioteca` per interagire con la vista principale.
- Associazione (unidirezionale) a `TabArchivio<T extends Dato>` – molteplicità 3: Mantiene tre reference alle tab di gestione dell'archivio per libri, utenti e prestiti.

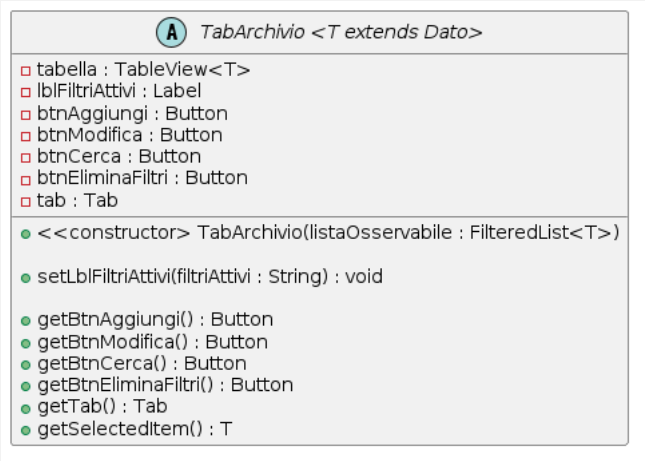
Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi servono unicamente a inizializzare la vista e recuperare le proprie componenti.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e fornisce solo lo stretto necessario dei dati richiesti dai servizi che offre.

`view.TabArchivio<T extends Dato>`



Questa classe astrae il concetto di tab per l'applicazione di gestione della biblioteca. Ogni tab deve avere una tabella, i bottoni per le azioni principali ed un'etichetta che riporta i filtri attivi. La classe è astratta anche se non contiene metodi astratti, per forzare il cliente a istanziare oggetti dalle classi concrete che la estendono.

Attributi principali

- `tabella`: Tabella che visualizza l'archivio.
- `lblFiltriAttivi`: Etichetta che indica i filtri attivi.
- `btnAggiungi`: Bottone per aggiungere un elemento nella tabella.

- `btnModifica`: Bottone per modificare un elemento nella tabella.
- `btnCerca`: Bottone per cercare un elemento nella tabella.
- `btnEliminaFiltri`: Bottone per eliminare i filtri applicati.
- `tab`: Reference alla tab.

Metodi pubblici

- `TabArchivio(listaOsservabile)`: Costruttore che imposta il layout generico della tab istanziando le componenti.
- `getter` per i bottoni e per la tab: Permettono di accedere alle reference mantenute dagli attributi.
- `setLblDiltriAttivi(filtriAttivi)`: Imposta il testo visibile della targhetta che mostra i filtri attivi.
- `getSelectedItem()`: Restituisce l'elemento selezionato dall'utente nella tabella.

Relazioni rilevanti

- È associata (unidirezionale) a `ViewBiblioteca`: La vista principale mantiene tre reference a oggetti `TabArchivio` per consentire al controller di accedere agli aspetti essenziali delle singole tab.
- È estesa da `TabArchivioLibri`, `TabArchivioUtenti` e `TabArchivioPrestiti`: Le classi concrete specializzano la tab generica di gestione di un archivio.

Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi servono unicamente a inizializzare la tab e recuperare le proprie componenti.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e fornisce solo lo stretto necessario dei dati richiesti dai servizi che offre.

`view.TabArchivioLibri`

Questa classe specializza `TabArchivio` per la gestione dell'archivio dei Libri. L'offerta di questa tab è ampliata permettendo di recuperare l'istanza di un bottone di cancellazione.

Attributi principali

- `btnCancella`: Bottone per la cancellazione di un elemento nella tabella.

Metodi pubblici

- `TabArchivioLibri(listaOsservabileLibri)`: Costruttore che aggiunge al layout standard di una tab generica il bottone di cancellazione.
- `getBtnCancella()`: Restituisce la reference al bottone di cancellazione.
- Metodi ereditati da `TabArchivio`: Getter dei bottoni, `setLblFiltriAttibi()`, `getSelectedItem()`.

Relazioni rilevanti

- Estende `TabArchivio<T extends Dato>`: La classe aggiunge al servizio base di `TabArchivio` la funzionalità del bottone di cancellazione.

Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi servono unicamente a inizializzare la tab e recuperare le proprie componenti.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e fornisce solo lo stretto necessario dei dati richiesti dai servizi che offre.

`view.TabArchivioUtenti`

Questa classe specializza `TabArchivio` per la gestione dell'archivio degli Utenti. L'offerta di questa tab è ampliata permettendo di recuperare l'istanza di un bottone di cancellazione.

Attributi principali

- `btnCancella`: Bottone per la cancellazione di un elemento nella tabella.

Metodi pubblici

- `TabArchivioUtenti(listaOsservabileUtenti)`: Costruttore che aggiunge al layout standard di una tab generica il bottone di cancellazione.
- `getBtnCancella()`: Restituisce la reference al bottone di cancellazione.
- Metodi ereditati da `TabArchivio`: Getter dei bottoni, `setLblFiltriAttivi()`, `getSelectedItem()`.

Relazioni rilevanti

- Estende `TabArchivio<T extends Dato>`: La classe aggiunge al servizio base di `TabArchivio` la funzionalità del bottone di cancellazione.

Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi servono unicamente a inizializzare la tab e recuperare le proprie componenti.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e fornisce solo lo stretto necessario dei dati richiesti dai servizi che offre.

`view.TabArchivioPrestiti`

Questa classe specializza `TabArchivio` per la gestione dell'archivio dei Prestiti. L'offerta rimane la stessa prevista dalla classe `TabArchivio`, senza l'aggiunta di ulteriori componenti.

Metodi pubblici

- `TabArchivioPrestiti(listaOsservabilePrestiti)`: Costruttore che richiama il costruttore della superclasse `TabArchivio`, senza aggiungere elementi ulteriori.
- Metodi ereditati da `TabArchivio`: Getter dei bottoni, `setLblFiltriAttivi()`, `getSelectedItem()`.

Relazioni rilevanti

- Estende `TabArchivio<T extends Dato>`: La classe concretizza la tab generica senza aggiungere funzionalità extra.

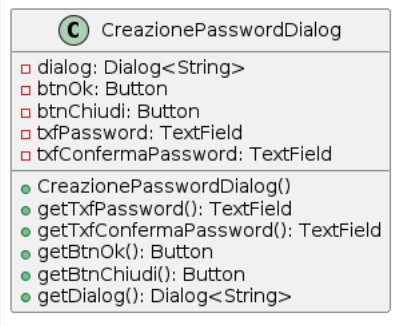
Livello di coesione

- **Coesione Funzionale**: I metodi e la struttura sono quelli strettamente necessari per inizializzare la tab e fornire l'accesso agli elementi base.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e fornisce solo lo stretto necessario dei dati richiesti dai servizi della superclasse.

`view.CreazionePasswordDialog`



La classe ha come utilità quella di visualizzare una finestra di pop up utile alle funzionalità *registrazione* e *reimposta password*. Essendo le due interfacce uguali in funzionalità e layout grafico, si è deciso di modellare le due finestre in una sola classe.

Attributi principali

- `dialog`: Oggetto di tipo `Dialog`, la finestra in se
- `txfPassword`: Campo di testo per inserire la password.
- `txfConfermaPassword`: Campo di testo per reinserire la password e verificarne la correttezza.
- `btnOK`: Bottone per andare avanti
- `btnChiudi`: Bottone per chiudere l'applicazione

Metodi pubblici

- `CreazionaPasswordDialog()`: Costruttore che aggiunge al layout base i campi di testo specifici e i pulsanti per la creazione della password.
- `getter` per tutti gli attributi: Restituiscono le reference mantenute dagli attributi.

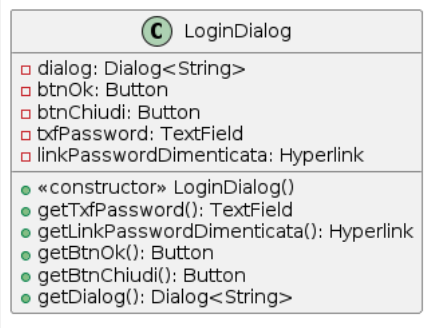
Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi sono esclusivamente dedicati alla gestione della finestra per la creazione delle password.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe richiede e fornisce solo i dati minimi necessari per svolgere i compiti richiesti.

`view.LoginDialog`



Questa classe definisce la finestra che appare quando il bibliotecario ha già registrato una password e deve autenticarsi per accedere all'applicazione.

Attributi principali

- `dialog`: Stage del contenitore della finestra.
- `btnOk`: Bottone per confermare.
- `btnChiudi`: Bottone per chiudere la finestra.
- `txfPassword`: Textfield dove inserire la password.
- `linkPasswordDimenticata`: Link che porta alla registrazione di una nuova password in caso il bibliotecario se la dimentichi.

Metodi pubblici

- `LoginDialog()`: Costruttore che aggiunge al layout base della finestra di benvenuto gli elementi specifici del log in.
- `getter` per gli altri attributi: restituiscono le reference mantenute dagli attributi.

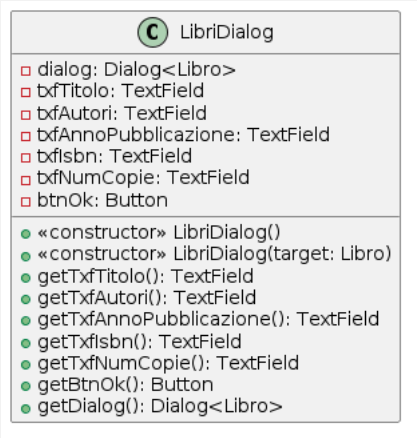
Livello di coesione

- **Coesione Funzionale**: Attributi e metodi sono dedicati esclusivamente alla gestione della finestra di log in.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe scambia solo i dati strettamente necessari ai servizi che offre.

`view.LibriDialog`



Classe che gestice la finestra che ha il compito di recuperare e validare i dati inseriti dal gestore.

Attributi principali

- `dialog`: Stage del contenitore della finestra.
- `txfTitolo`: Campo di testo per il titolo del libro.
- `txfAutori`: Campo di testo per la lista degli autori.
- `txfAnnoPubblicazione`: Campo di testo per l'anno di pubblicazione.
- `txfIsbn`: Campo di testo per il codice ISBN.
- `btnOk`: Bottone di conferma.
- `txfNumCopie`: Campo di testo per il numero di copie disponibili.

Metodi pubblici

- `LibriDialog(Libro target)`: Costruttore che imposta il layout della finestra nel caso in cui si scelga di modificare un libro.
- `LibriDialog()`: Costruttore che imposta il layout della finestra pop up.
- `getter` per tutti gli attributi: restituiscono le reference mantenute dagli attributi.

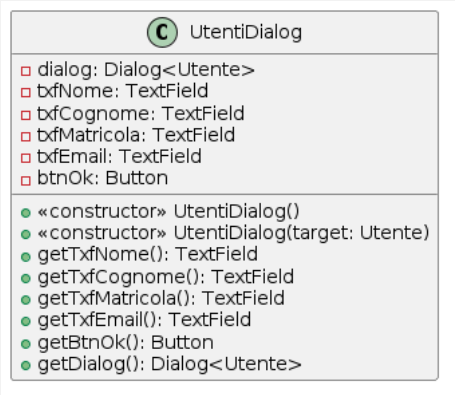
Livello di coesione

- **Coesione Funzionale**: Gli attributi e i metodi sono dedicati esclusivamente all'inizializzazione della finestra pop up e all'acquisizione dei dati relativi ai libri.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e restituisce solo i dati necessari ai servizi previsti.

`view.UtentiDialog`



Questa classe gestisce la finestra di dialog nel caso in cui venga generata dalla tab dell'archivio degli utenti. La finestra ha il compito di recuperare i dati inseriti dal gestore.

Attributi principali

- `dialog`: Stage del contenitore della finestra.
- `txfNome`: Campo di testo per il nome dell'utente.
- `txfCognome`: Campo di testo per il cognome dell'utente.
- `txfMatricola`: Campo di testo per la matricola dell'utente.
- `txfEmail`: Campo di testo per l'e-mail istituzionale dell'utente.
- `btnOk`: Bottone di conferma.

Metodi pubblici

- `UtentiDialog()`: Costruttore che imposta il layout della finestra pop up nel caso in cui si scelga un utente da modificare.
- `UtentiDialog(Utente target)`: Costruttore che imposta il layout della finestra pop up.
- `getter` per tutti gli attributi: restituiscono le reference mantenute dagli attributi.

Livello di coesione

- **Coesione Funzionale**: Gli attributi e i metodi servono esclusivamente all'inizializzazione della finestra pop up e alla raccolta dei dati relativi agli utenti.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e restituisce solo i dati strettamente necessari ai servizi previsti.

view.PrestitiDialog

Questa classe gestisce il dialog per il caso in cui la finestra di utilità venga generata dalla tab dell'archivio dei prestiti. La finestra ha il compito di recuperare i dati inseriti dal gestore.

Attributi principali

- `dialog`: Stage del contenitore della finestra.
- `btnOk`: Bottone di conferma.
- `btnChiudi`: Bottone di chiusura.
- `cbUtenti`: combobox per la selezione dell'utente.
- `cbLibri`: combobox per la selezione del libro.
- `datePicker`: datepicker per selezionare la data.

Metodi pubblici

- `PrestitiDialog(FilteredList<Libro> libri, FilteredList<Utente> utenti)`: Costruttore che imposta il layout della finestra pop up.
- `getter` per tutti gli attributi: restituiscono le reference mantenute dagli attributi.

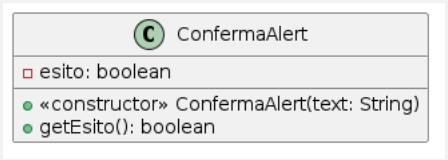
Livello di coesione

- **Coesione Funzionale**: Gli attributi e i metodi servono esclusivamente all'inizializzazione della finestra pop up e alla raccolta dei dati relativi ai prestiti.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe accetta e restituisce solo i dati strettamente necessari ai servizi previsti.

view.ConfermaAlert



Questa classe modella una una finestra di conferma con i soli pulsanti *"OK"* e *"Annulla"*, oltre che un'etichetta di avviso.

Attributi Principali

- `esito`: Restituisce `true` se è stata confermata l'operazione, `false` altrimenti

---

Metodi pubblici

- `ConfermaAlert()`: Costruttore che inizializza i componenti grafici della finestra.
- `getEsito()`: restiutisce `esito`

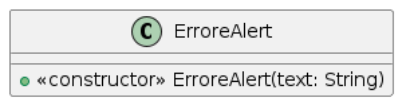
Livello di coesione

- **Coesione Funzionale**: Attributi e metodi sono dedicati esclusivamente alla gestione della finestra di conferma.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe scambia solo i dati strettamente necessari ai servizi che offre.

`view.ErroreAlert`



Questa classe modella una finestra di errore che mostra un messaggio di avviso con il pulsante *"OK"* per chiudere la finestra.

Metodi pubblici

- `ErroreAlert()`: Costruttore che inizializza e visualizza il componente grafico dell'alert di errore.

Livello di coesione

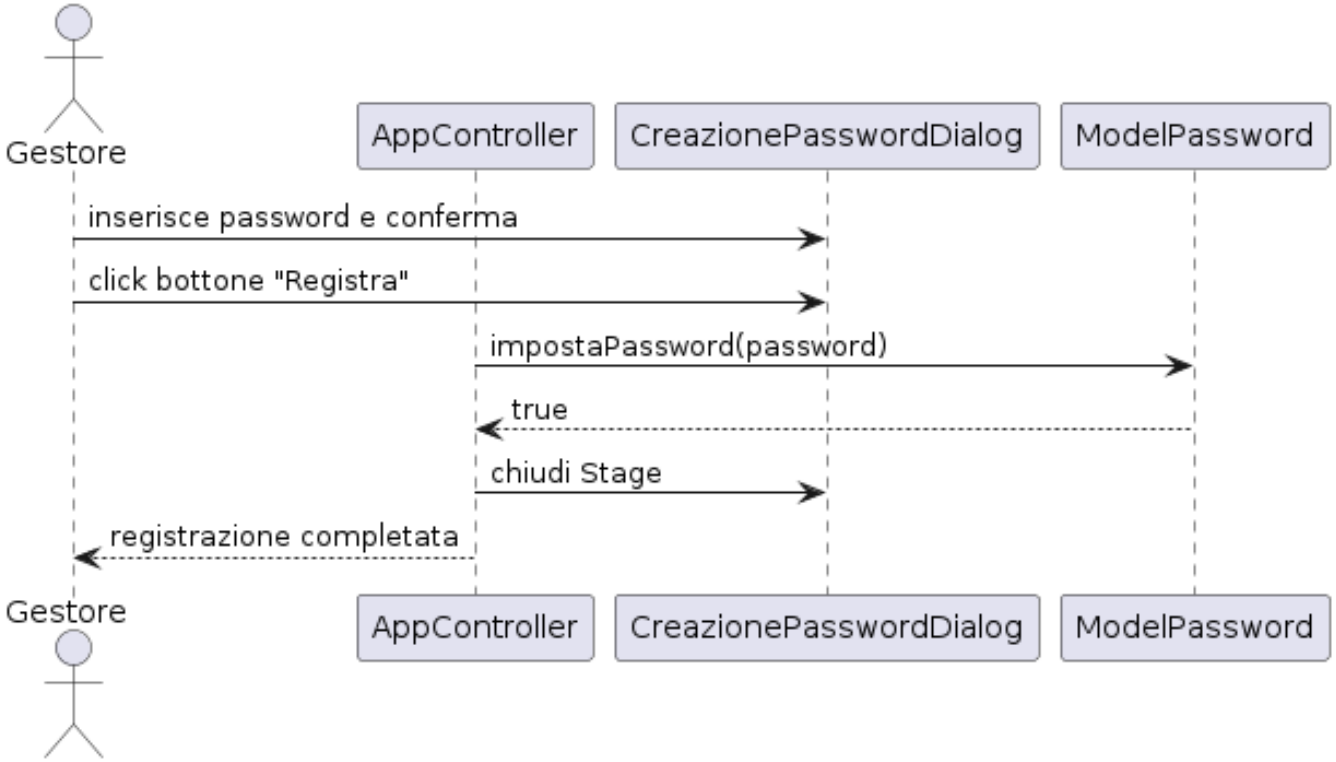
- **Coesione Funzionale**: La classe è completamente dedicata alla gestione della finestra di errore.

Livello di accoppiamento

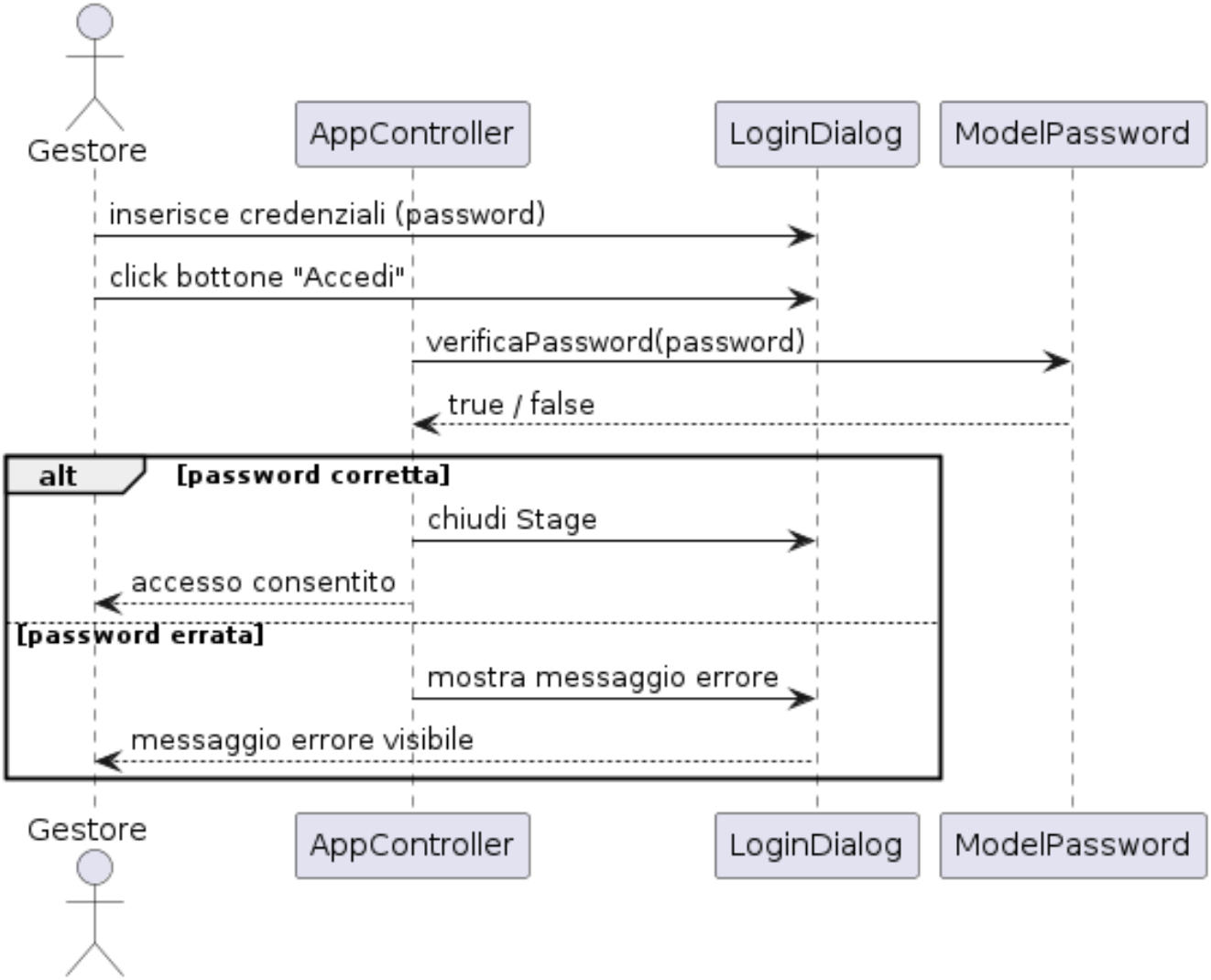
- **Accoppiamento per Servizi**: La classe scambia solo i dati strettamente necessari ai servizi che offre.

3 Modello Dinamico

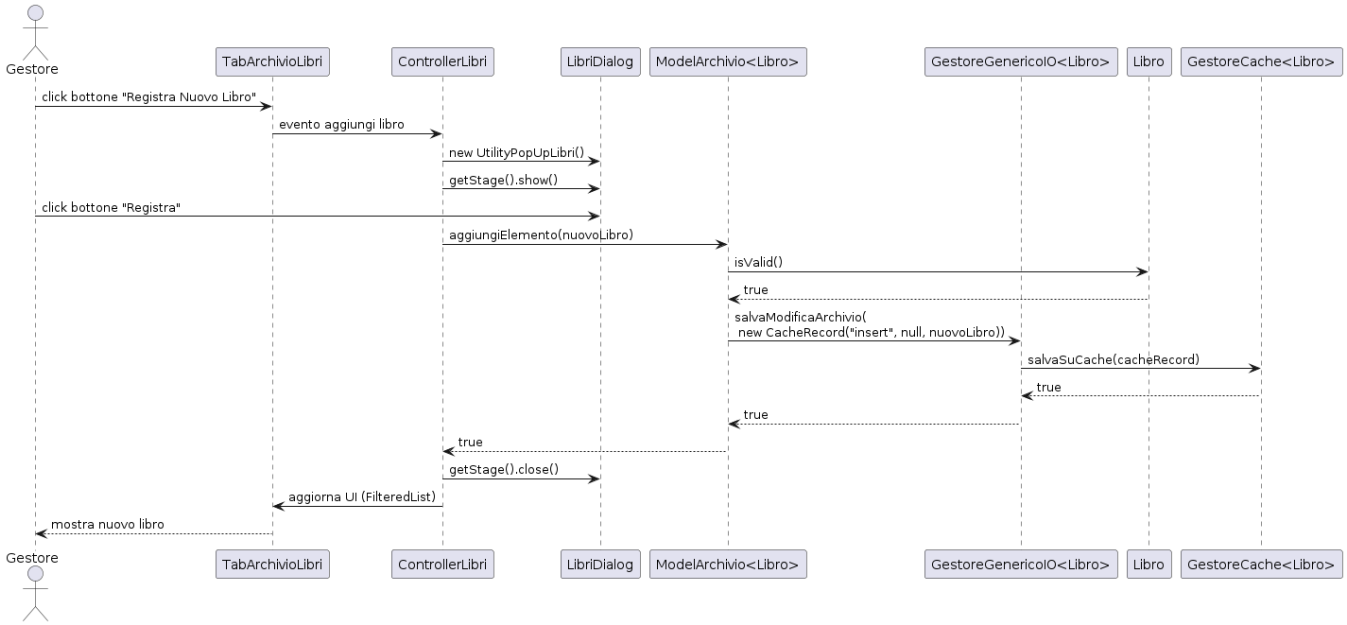
3.1 CU-1.1 - Sign Up



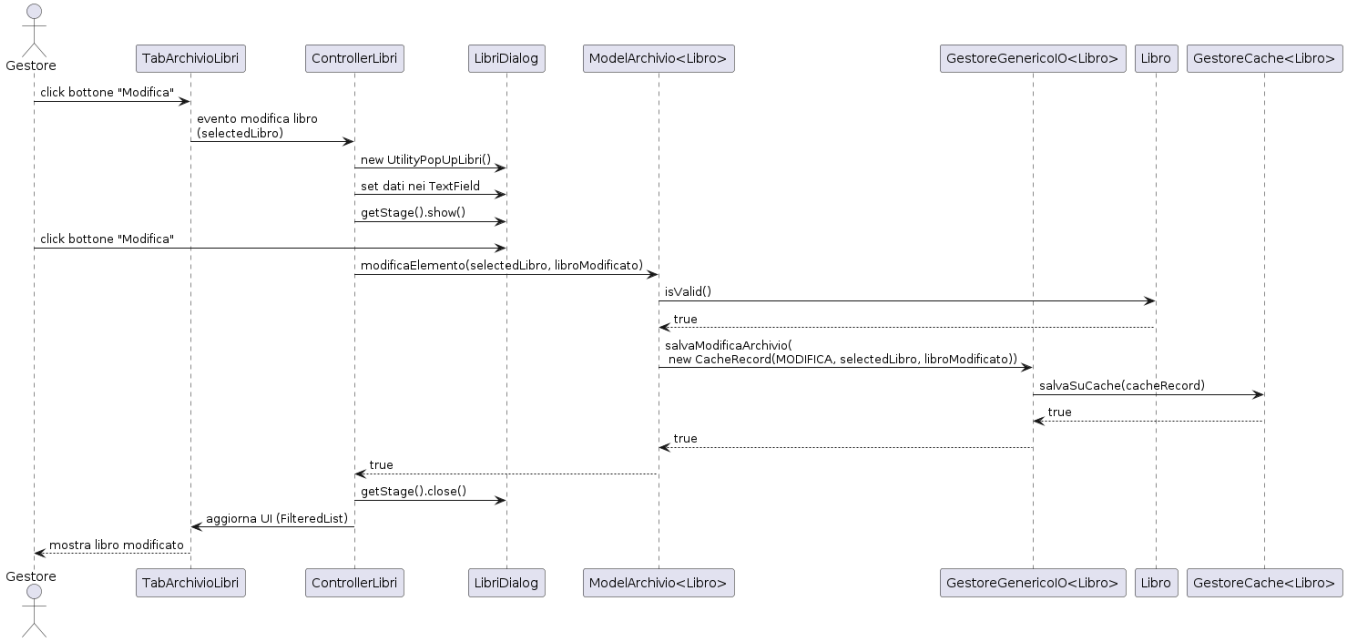
3.2 CU-1.2 - Login



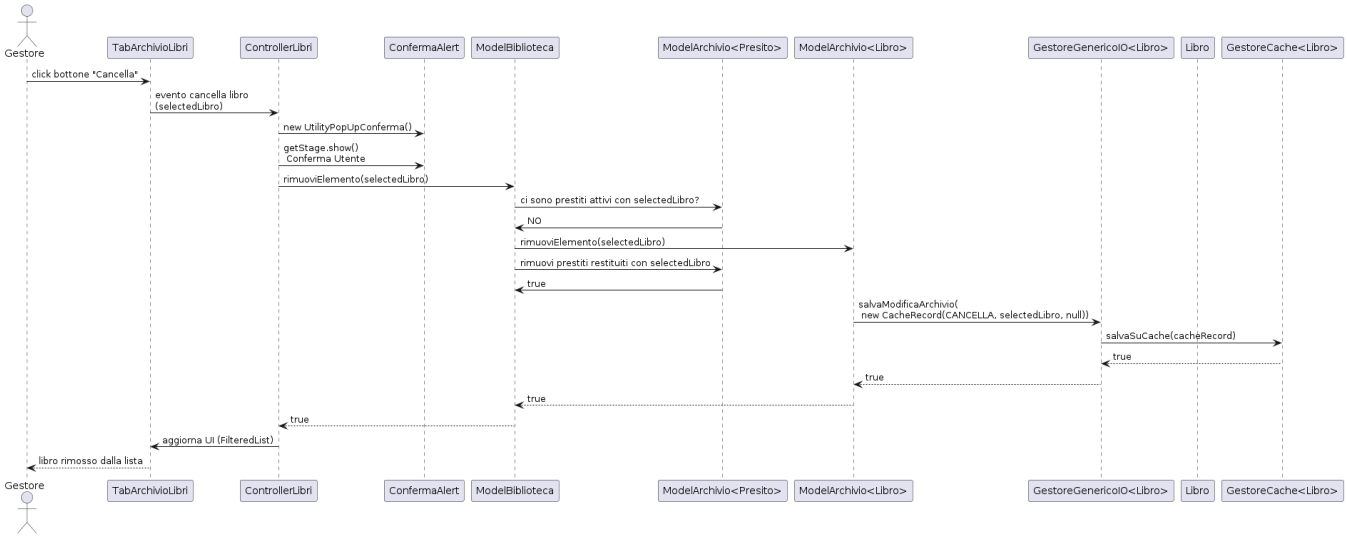
3.3 CU-2.1 - Inserimento Libro



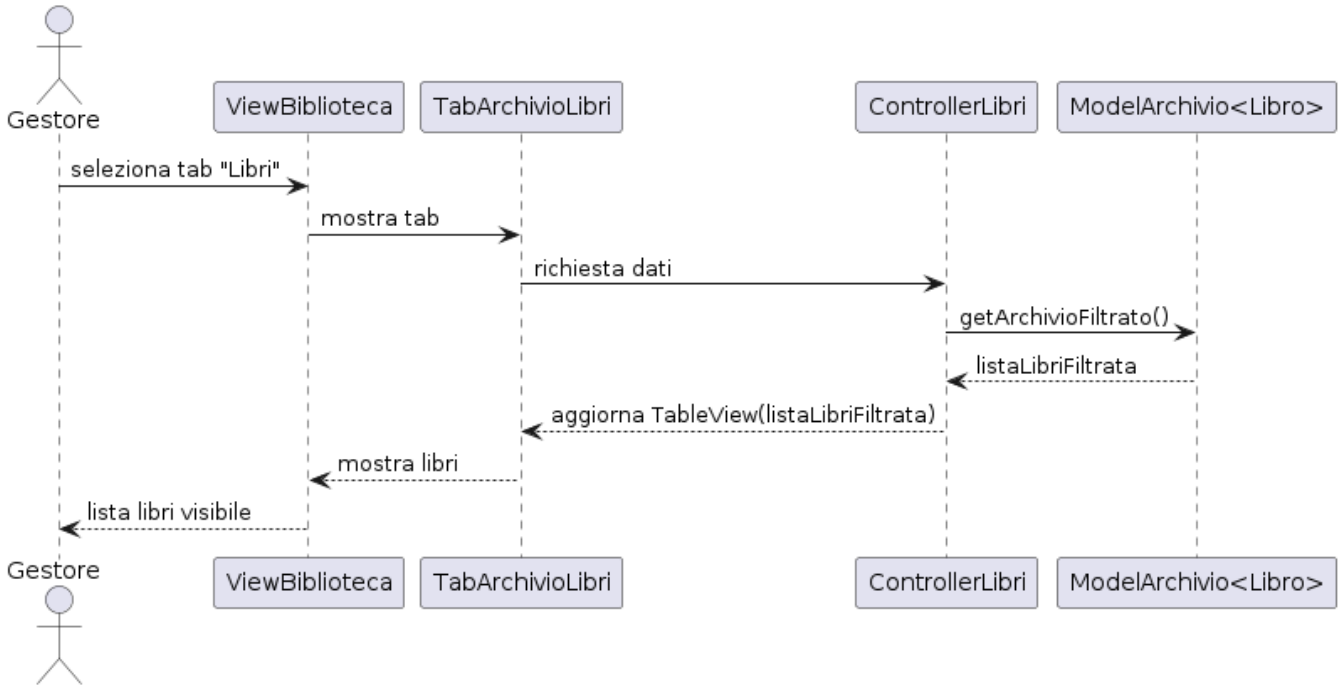
3.4 CU-2.2 - Modifica Dati Libro



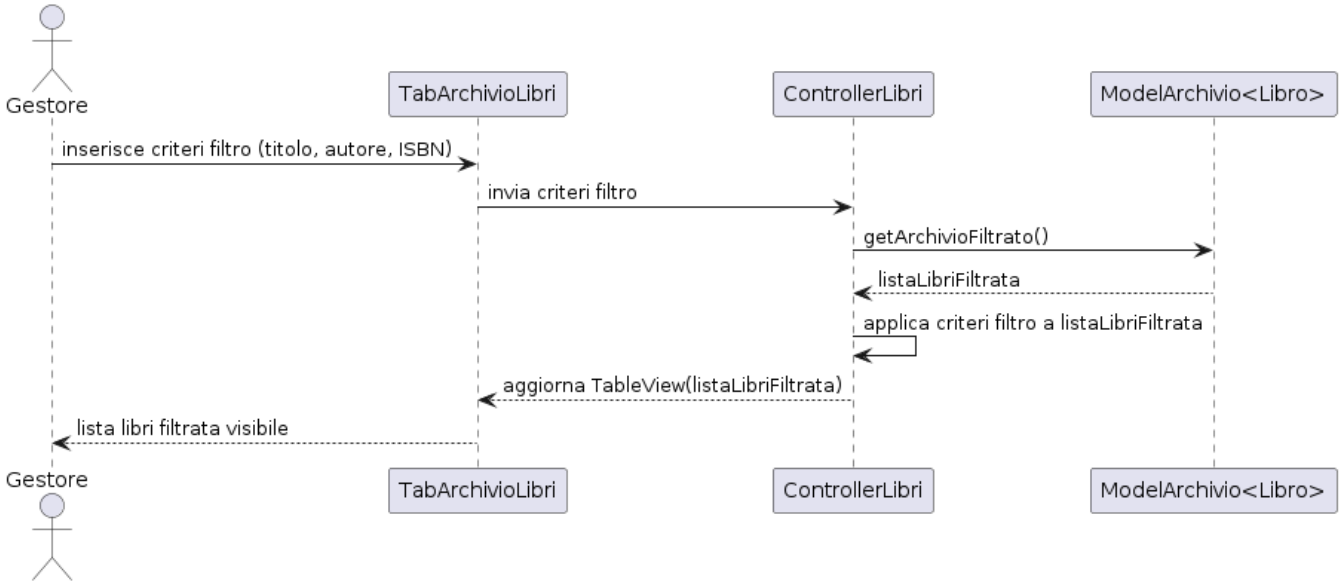
3.5 CU-2.3 - Cancellazione Libro



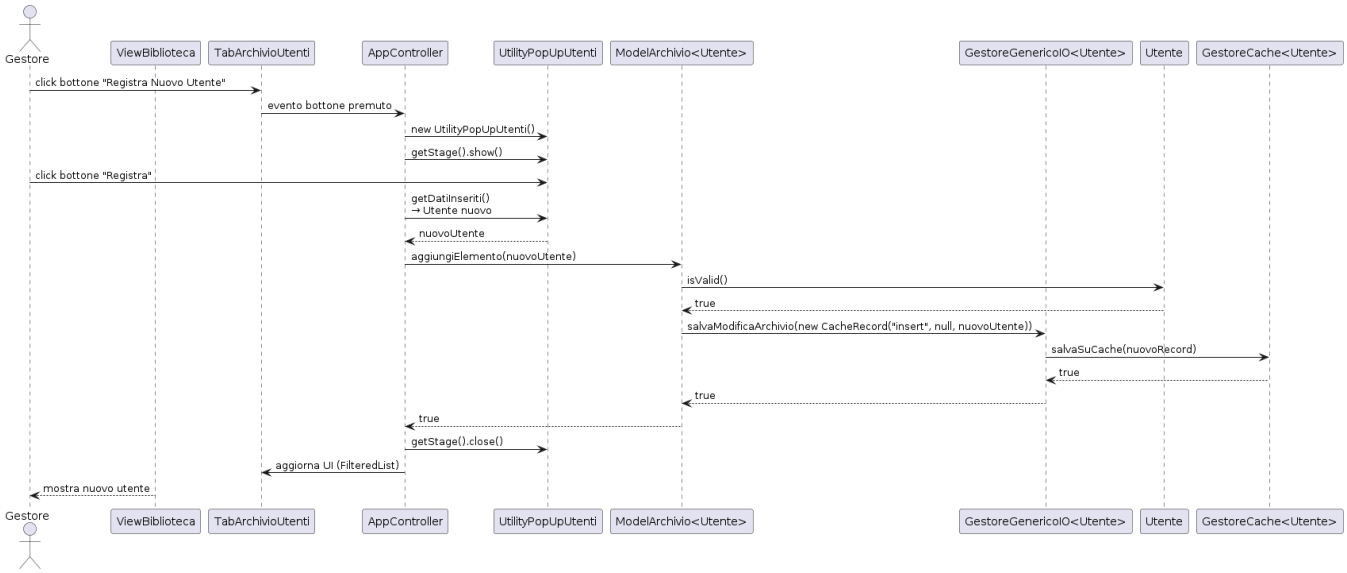
3.6 CU-2.4 - Visualizzazione Archivio Libri



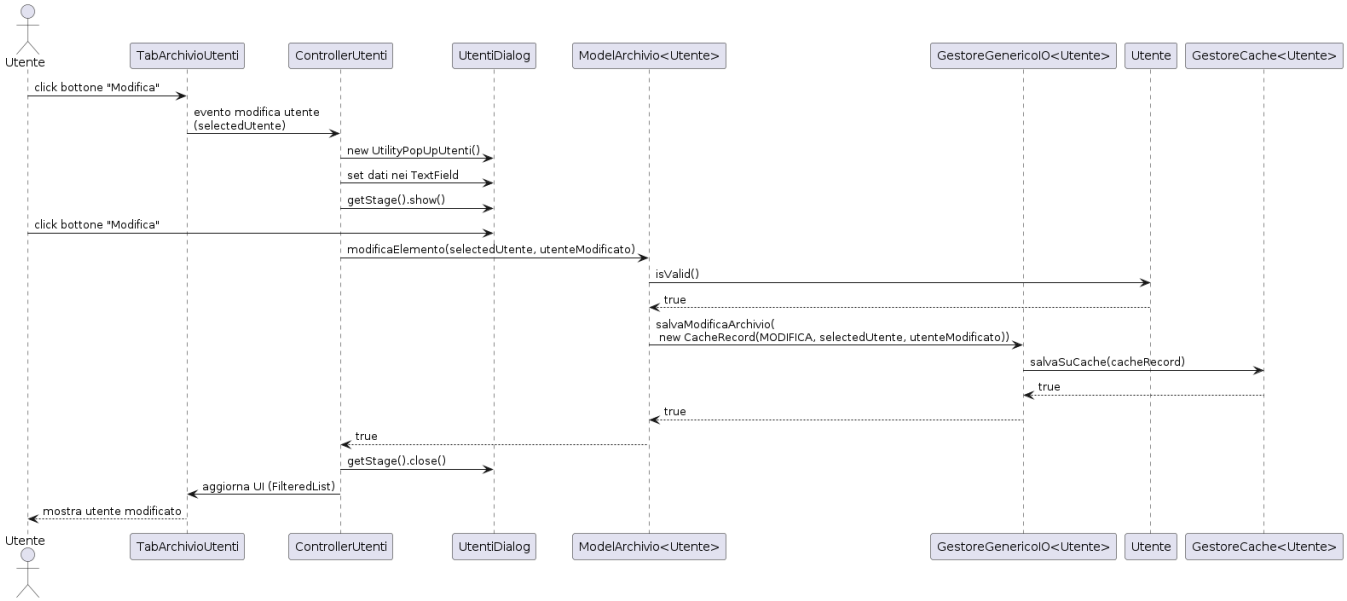
3.7 CU-2.5 - Ricerca Libro



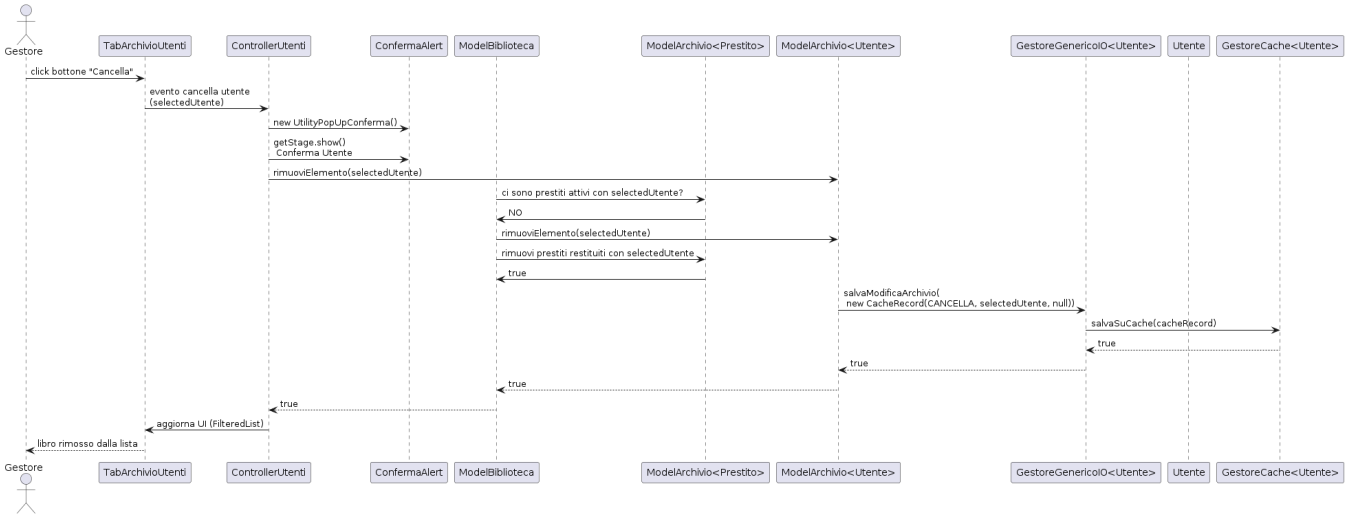
3.8 CU-3.1 - Inserimento Nuovo Utente



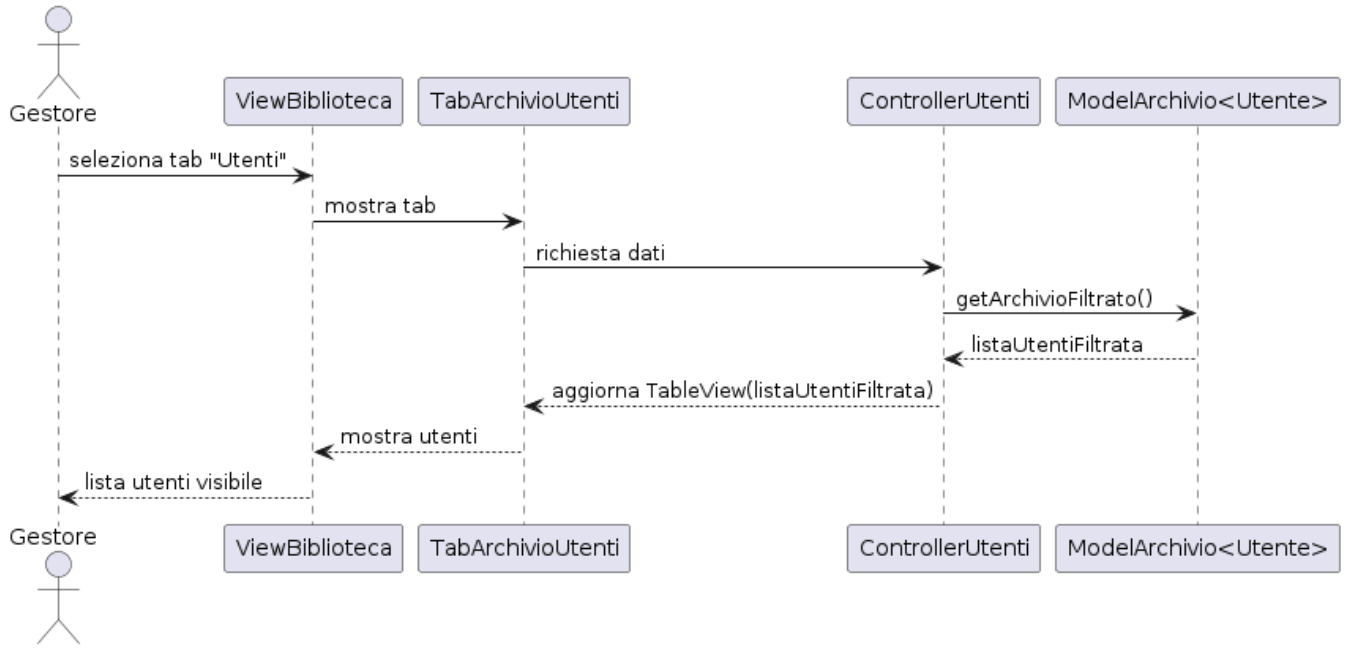
3.9 CU-3.2 - Modifica Dati Utente



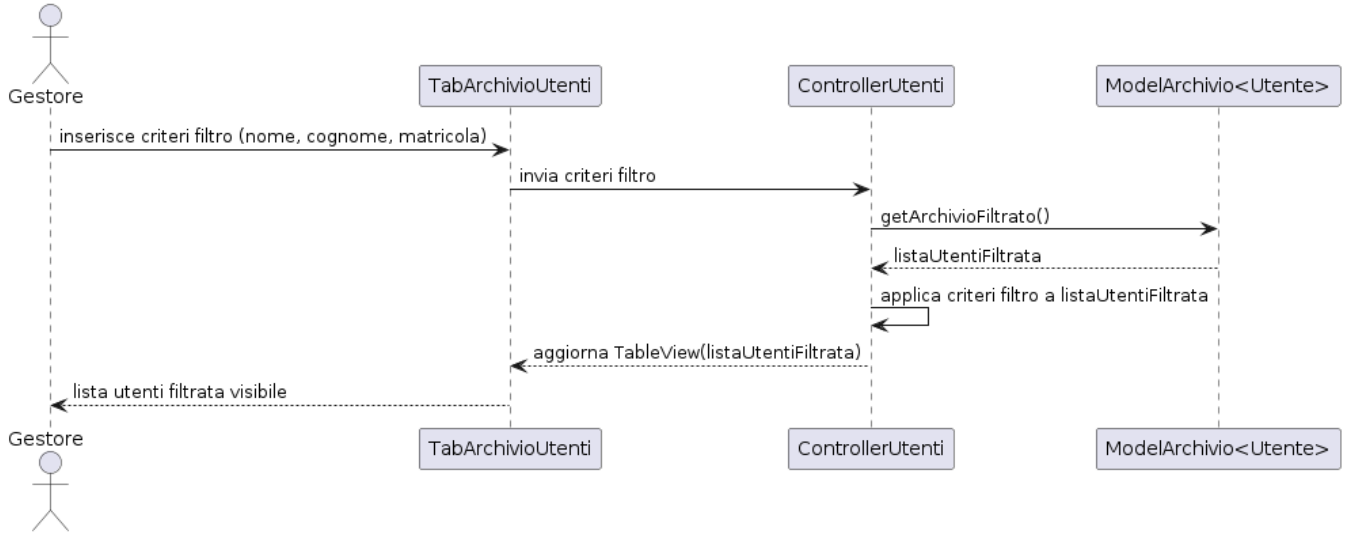
3.10 CU-3.3 - Cancellazione Utente



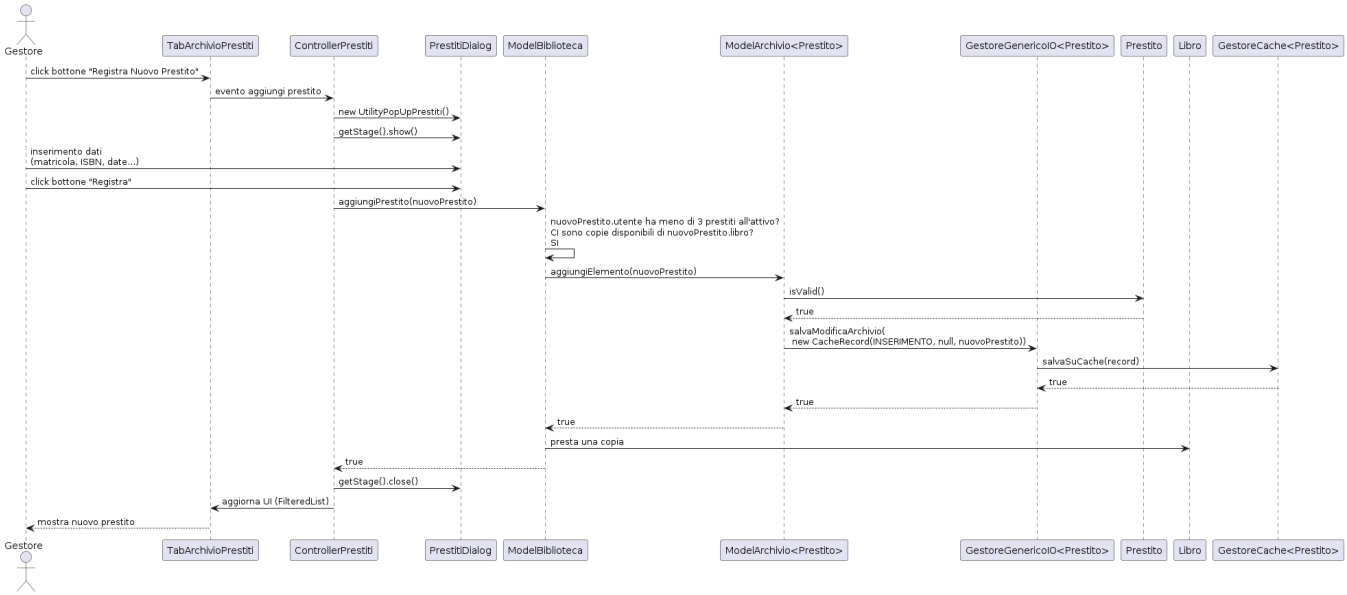
3.11 CU-3.4 - Visualizzazione Archivio Utenti



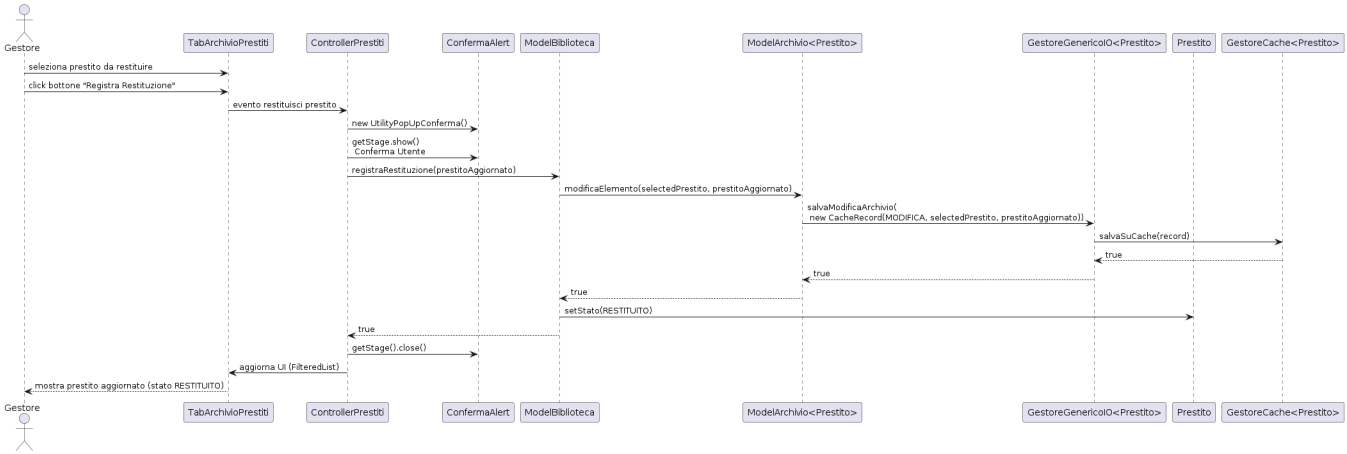
3.12 CU-3.5 - Ricerca Utente



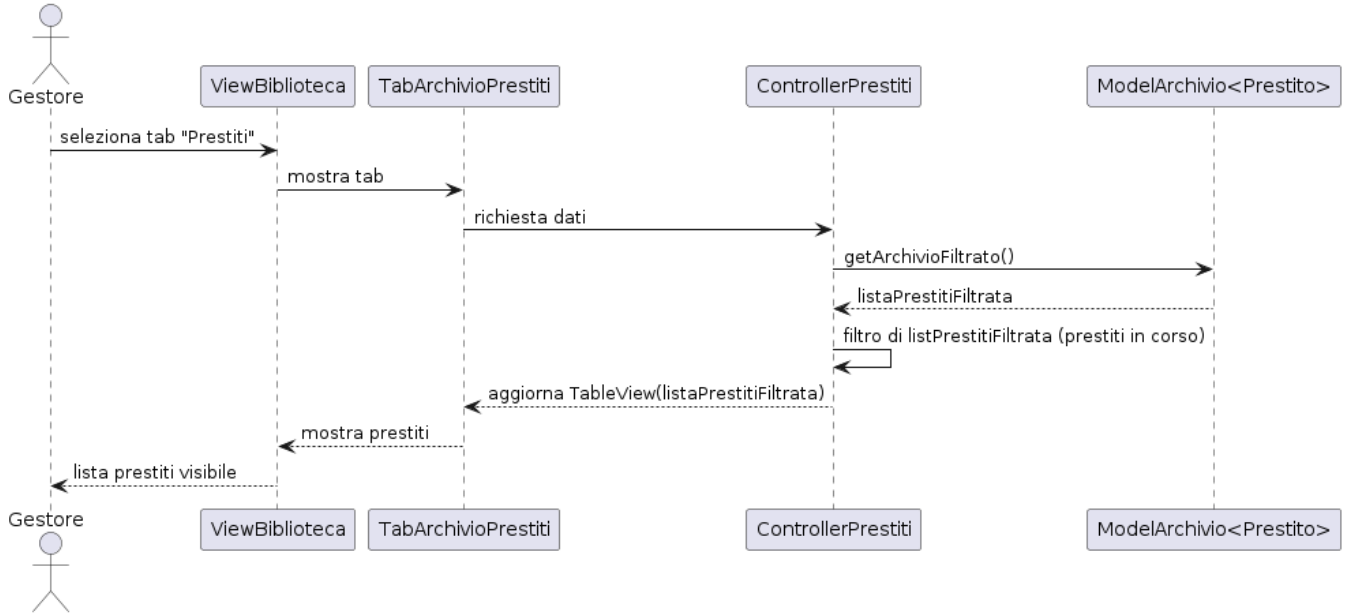
3.13 CU-4.1 - Registrazione Nuovo Prestito



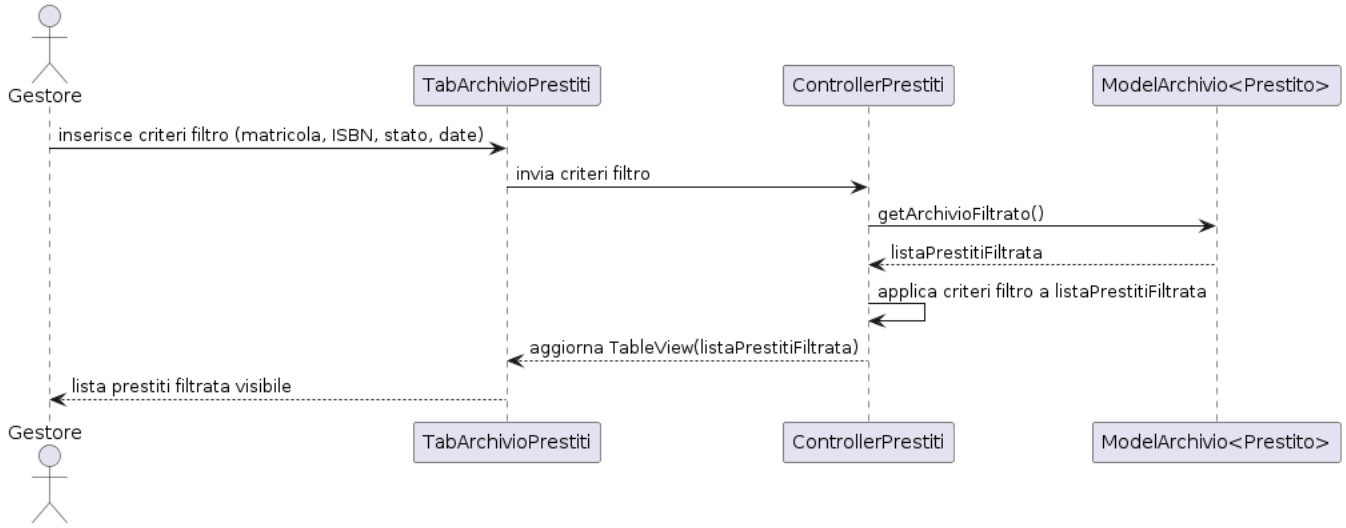
3.14 CU-4.2 - Registrazione Restituzione Libro



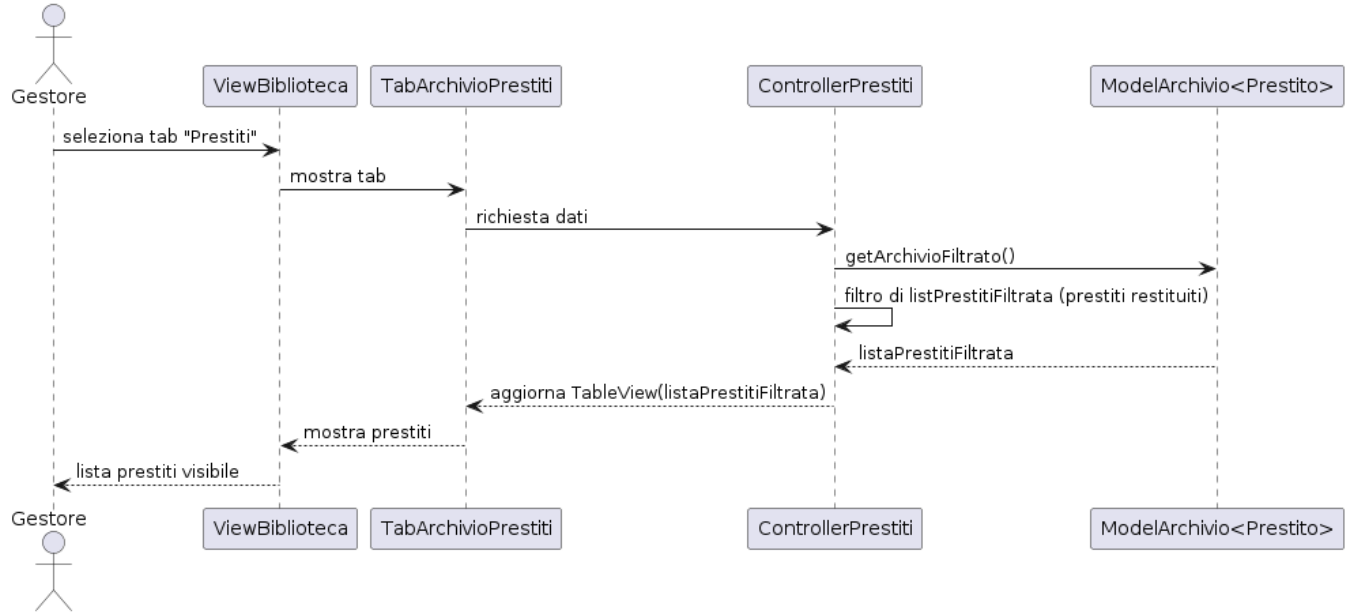
3.15 CU-4.3 - Visualizzazione Archivio Prestiti in Corso



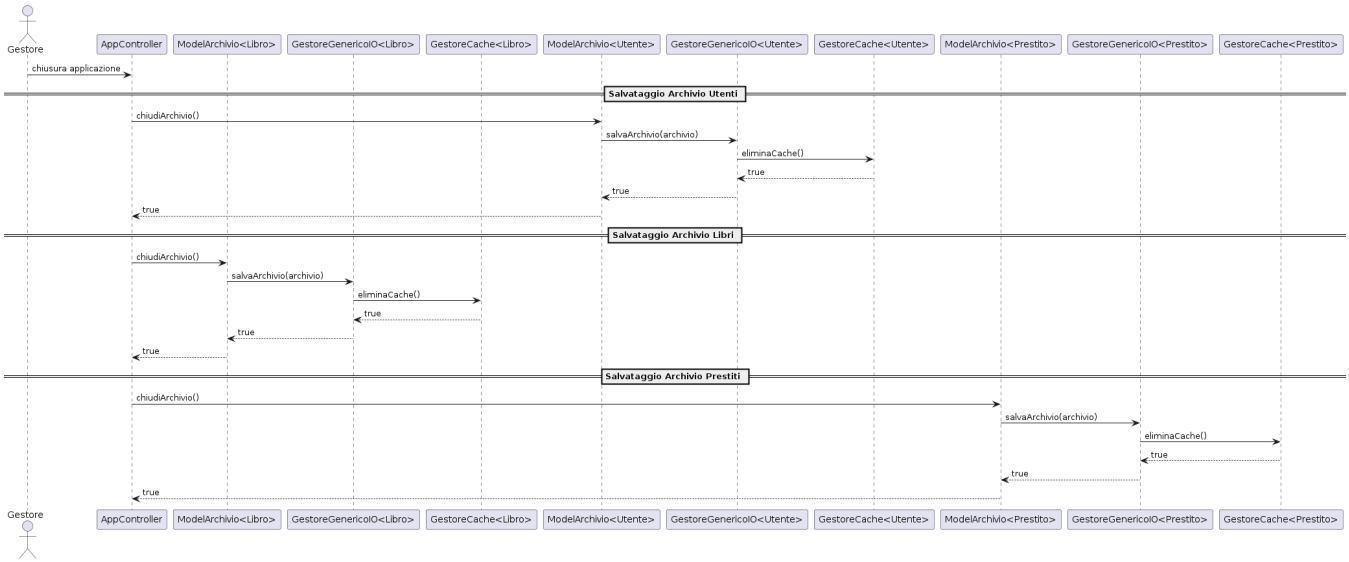
3.16 CU-4.4 - Ricerca Prestito



3.17 CU-4.5 - Visualizzazione Storico Prestito

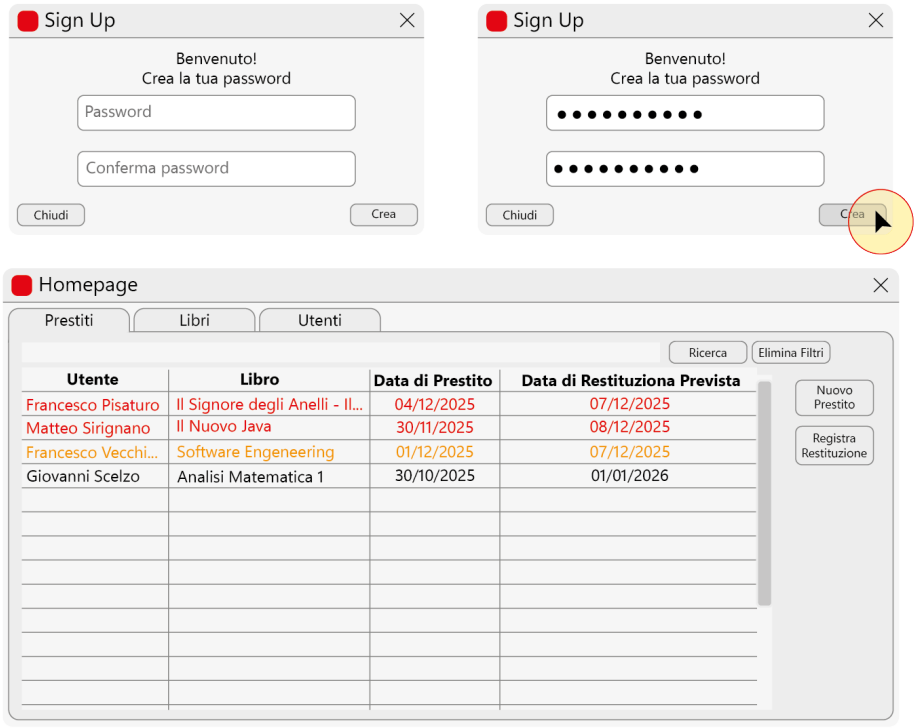


3.18 CU-5.1 - Chiusura Applicazione

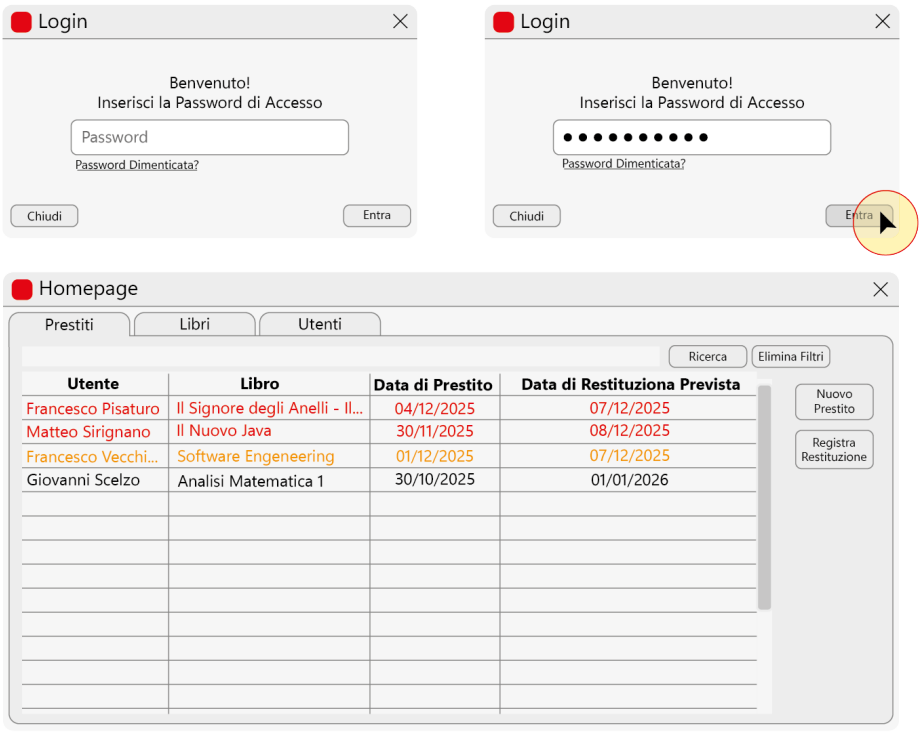


4 Design dell'Interfaccia Utente

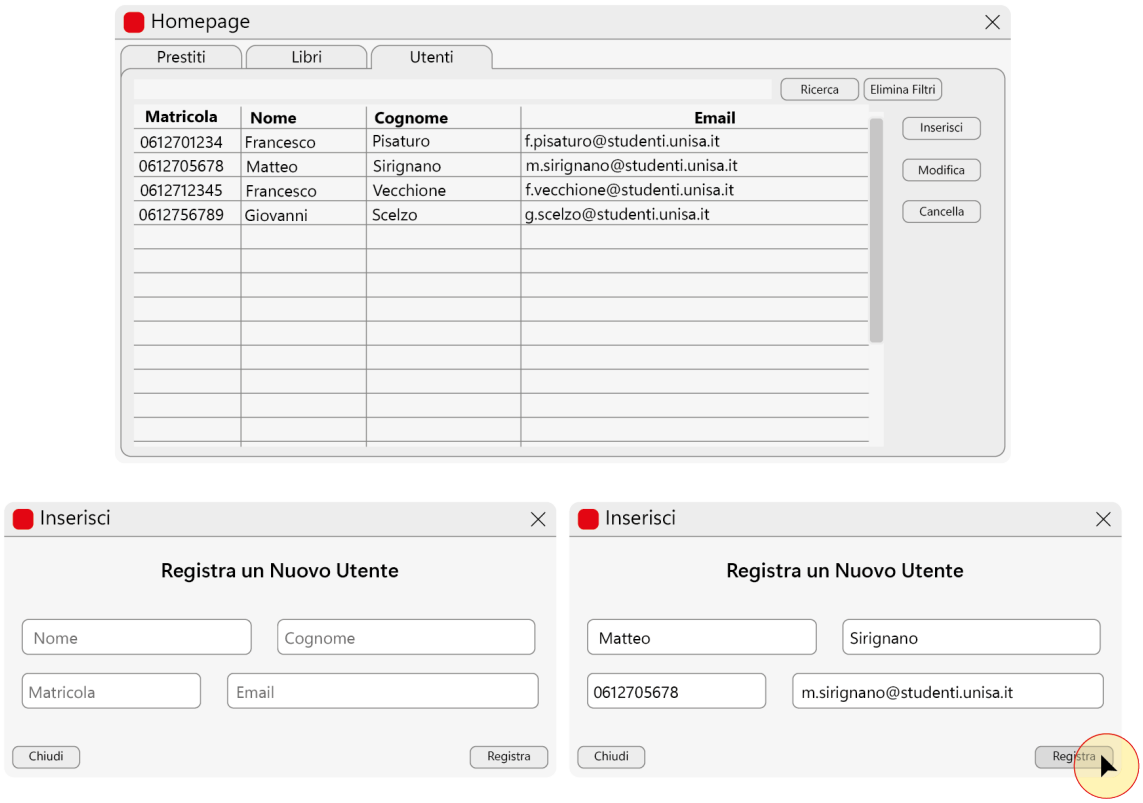
4.1 CU-1.1 - Sign Up



4.2 CU-1.2 - Login

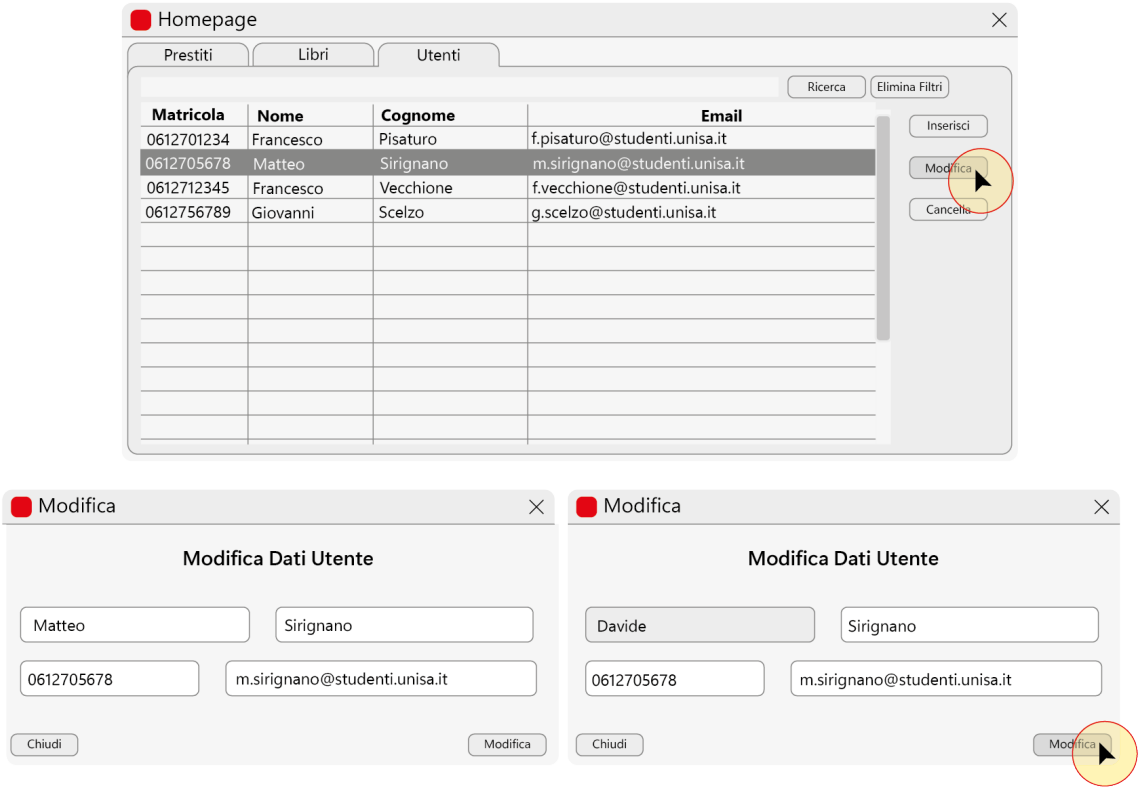


4.3 CU-3.1 - Inserimento Nuovo Utente



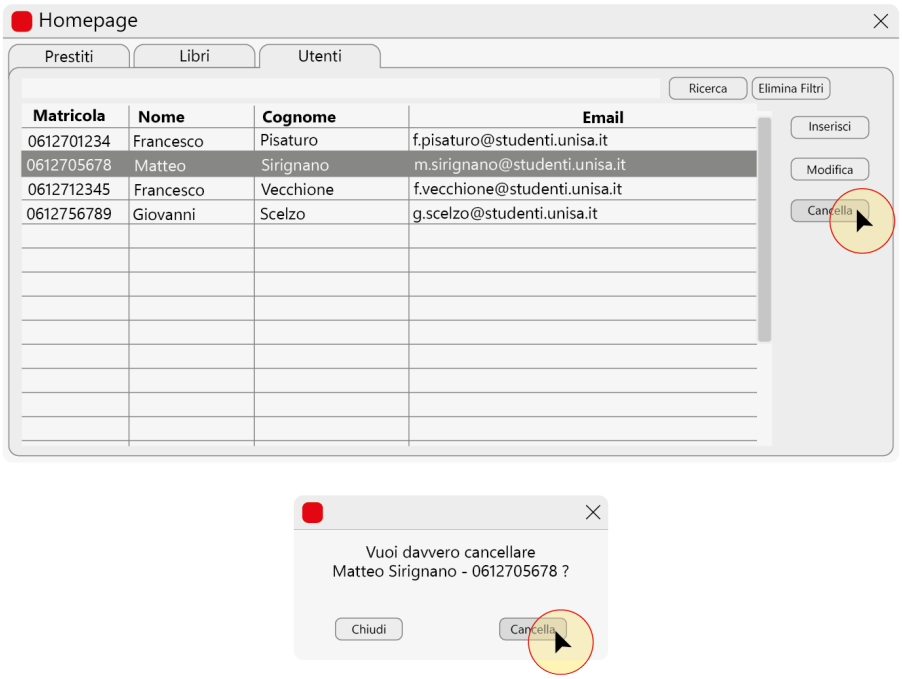
l'interazione con l'interfaccia è la medesima per il caso d'uso CU-2.1

4.4 CU-3.2 - Modifica Dati Utente



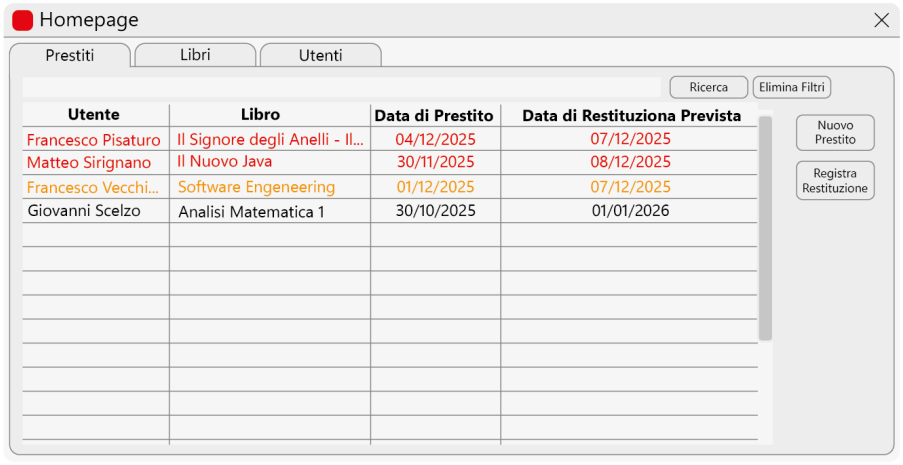
l'interazione con l'interfaccia è la medesima per il caso d'uso CU-2.2

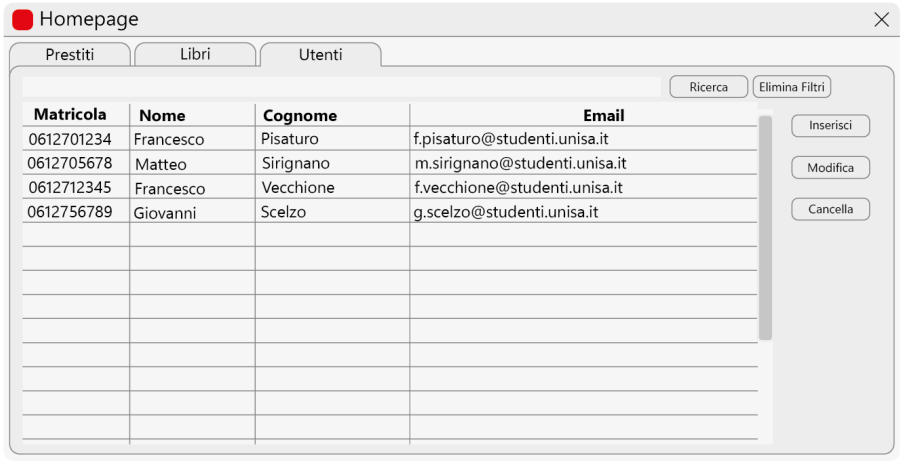
4.5 CU-3.3 - Cancellazione Utente



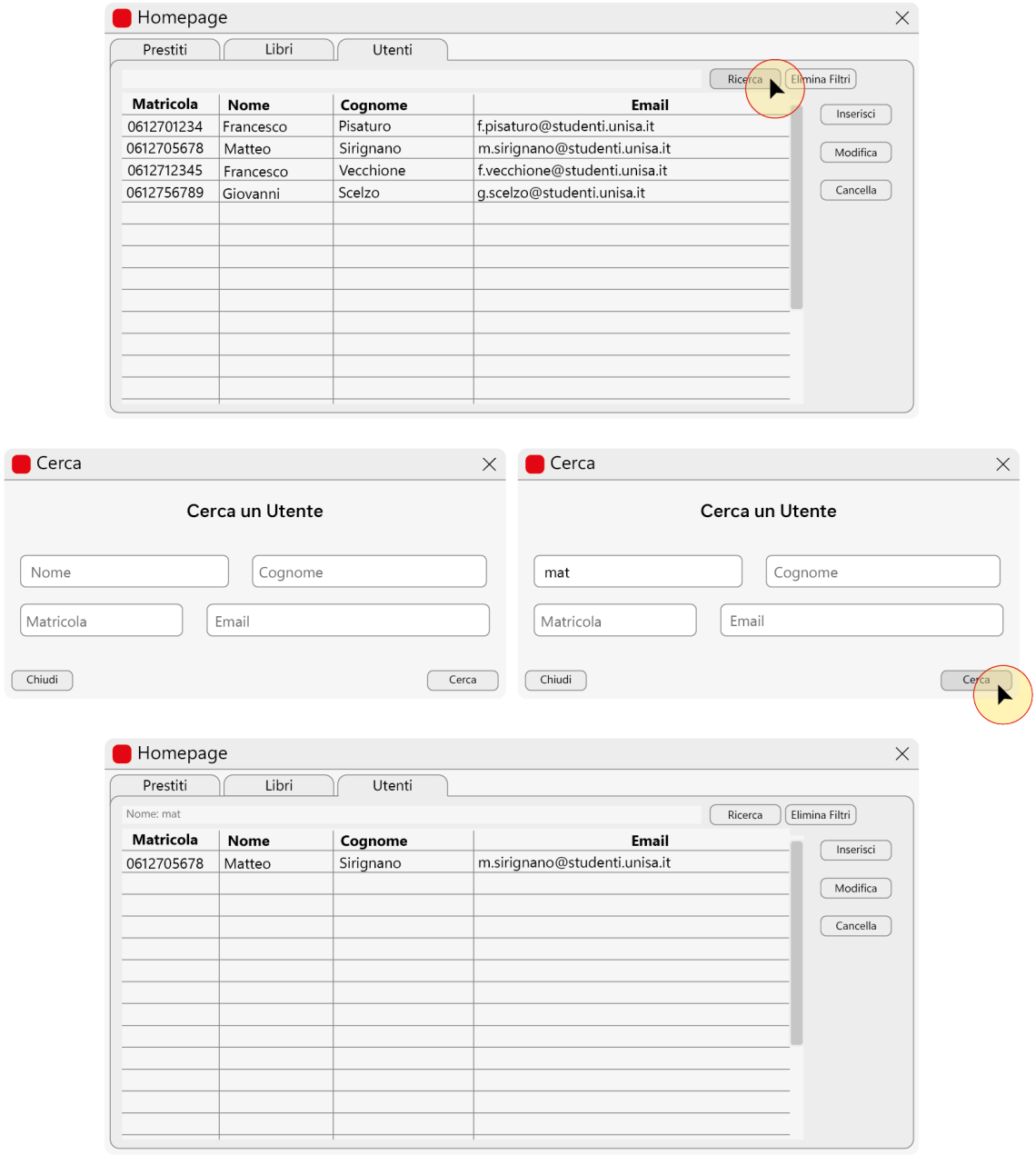
l'interazione con l'interfaccia è la medesima per il caso d'uso CU-2.3

4.6 CU-2.4, CU-3.4, CU-4.3 - Visualizzazione Dati





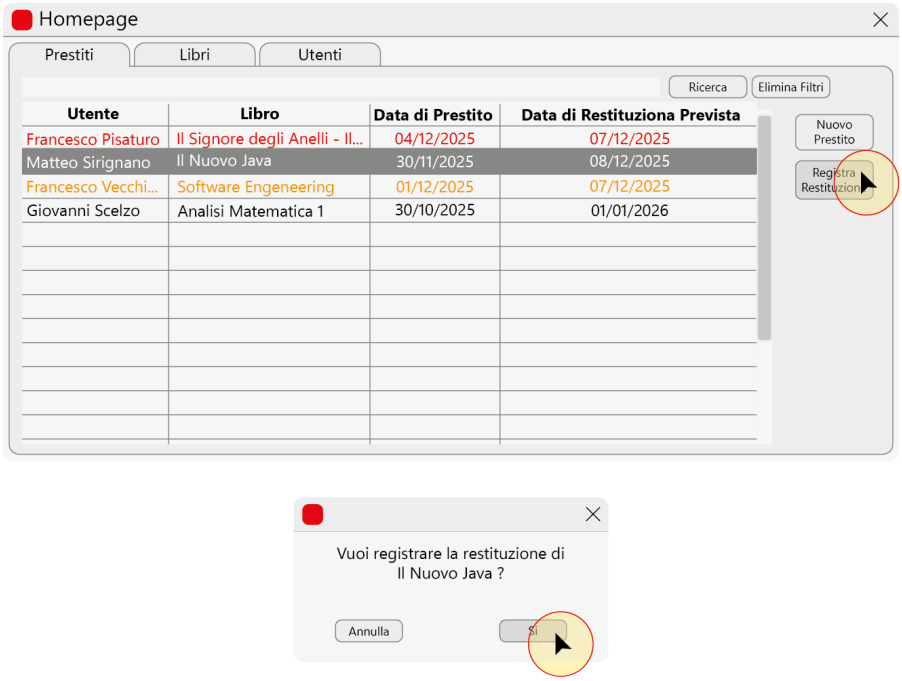
4.7 CU-3.5 - Ricerca Utente



**l'interazione con l'interfaccia è la medesima per il caso d'uso CU-2.5, CU-4.4*

4.8 CU-4.1 - Registrazione Nuovo Prestito

4.9 CU-4.2 - Registrazione Restituzione Libro



Modifiche Apportate

1. **Versione 2.0**
   - **Data della Modifica**: 10/12/2025
   - **Autore**: Giovanni Scelzo
   - **Descrizione**:
     (a) modifica del class diagram dopo alcune valutazioni durante la fase di implementazione
     (b) aggiornamento della descrizione del modello statico in accordo al nuovo CD
     (c) aggiornamento del modello dinamico in accordo al nuovo CD
     (d) aggiornamento dei mock up dell'interfaccia grafica al nuovo CD
2. **Versione 2.1**
   - **Data della Modifica**: 15/12/2025
   - **Autore**: Giovanni Scelzo
   - **Descrizione**:
     (a) modifica del class diagram dopo alcune valutazioni durante la fase di implementazione (più classi controller per la gestione degli eventi, implementazioni diverse per le finestre di pop up)
     (b) aggiornamento della descrizione del modello statico in accordo al nuovo CD
     (c) aggiornamento del modello dinamico in accordo al nuovo CD
     (d) aggiornamento dei mock up dell'interfaccia grafica al nuovo CD
3. **Versione 2.2**
   - **Data della Modifica**: 16/12/2025
   - **Autore**: Giovanni Scelzo
   - **Descrizione**:
     (a) modifica del class diagram dopo alcune valutazioni durante la fase di implementazione (aggiunta della classe `ModelBiblioteca` che migliora l'eleganza e la correttezza progettuale del codice)
     (b) aggiornamento della descrizione del modello statico in accordo al nuovo CD
     (c) aggiornamento del modello dinamico in accordo al nuovo CD