Documento di progetto
Biblioteca Universitaria

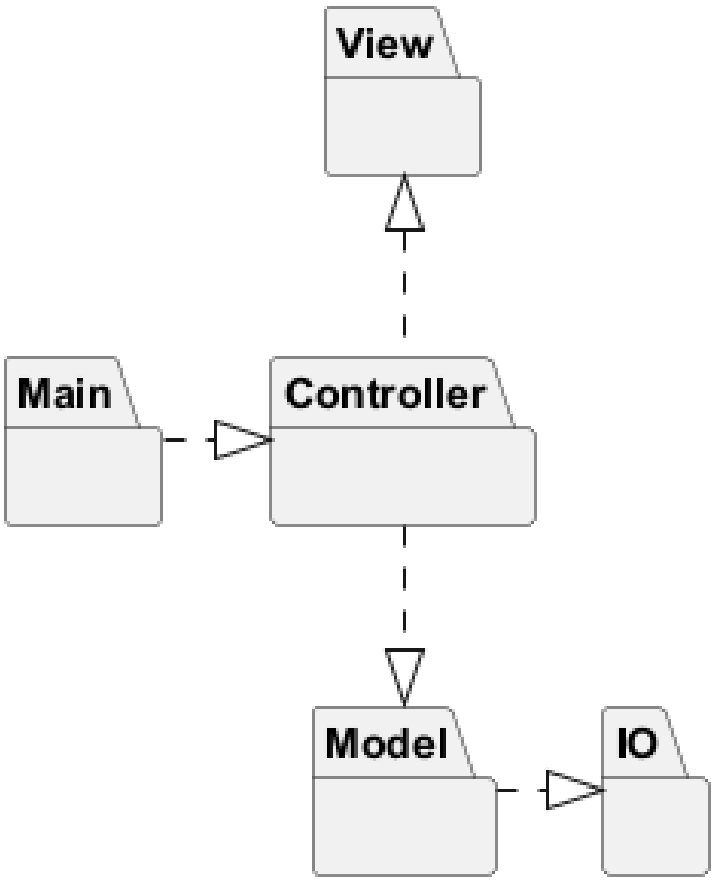
Documento di Progettazione v1.0

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

1 Architettura del Sistema

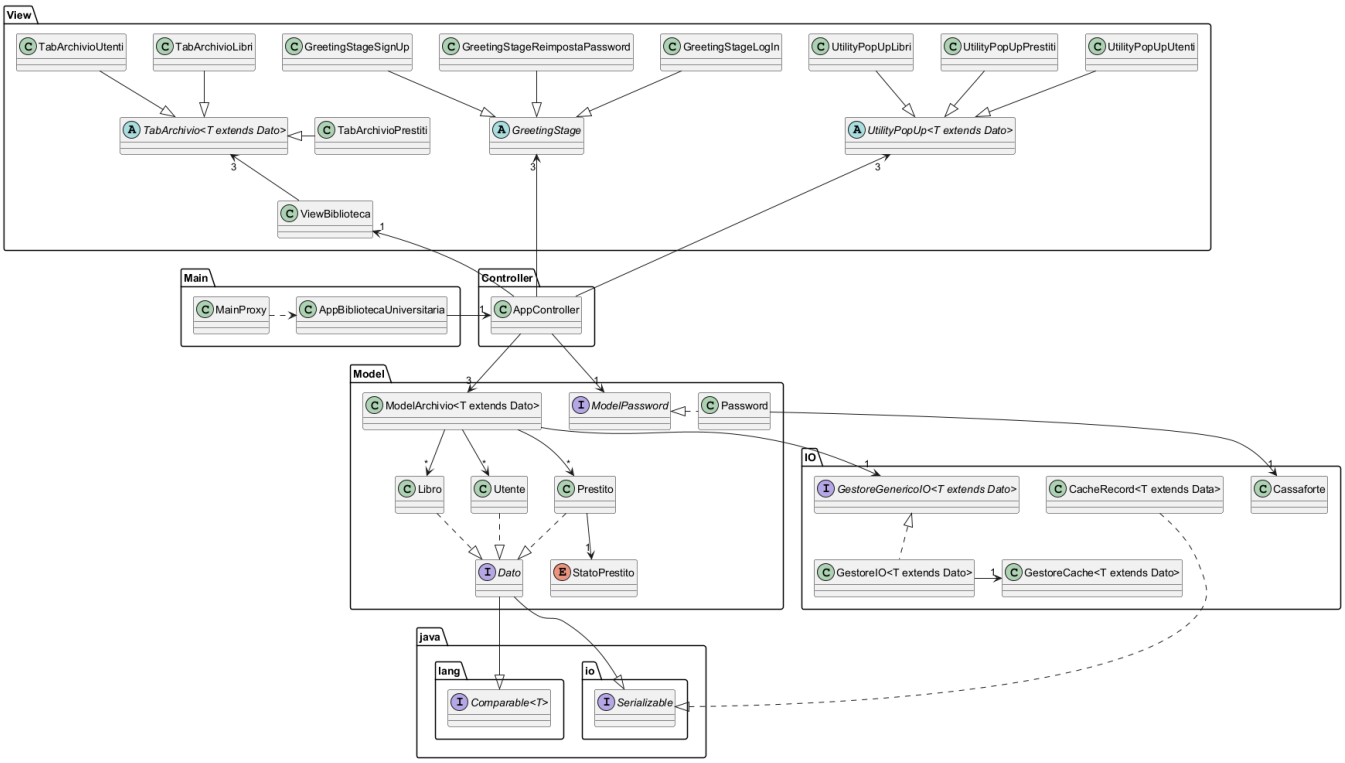
L'architettura scelta per la realizzazione del software per la gestione della biblioteca universitaria è il pattern Model-View-Controller. Questa decisione è stata presa in quanto il gruppo ha familiarità con il pattern che permette anche di progettare in modo semplice un'applicazione Java che sfrutta un'interfaccia grafica. Oltre ai package previsti dal pattern base, sono presenti due ulteriori package: il package Main ed il package IO. E' fondamentale dunque descrivere i ruoli che i singoli package interpretano:

- 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 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 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; interagendo con il modulo che si occupa dell'IO per recuperare e salvare i dati al fine di poter adempiere ai servizi che offre; per questo è dipendente dal package IO.
- 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 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, ma offre solo i propri servizi, non ha alcuna dipendenza verso il Controller.



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)`

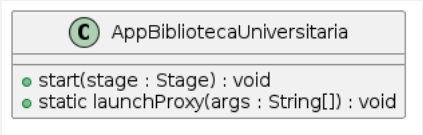
Livello di coesione

- **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`.

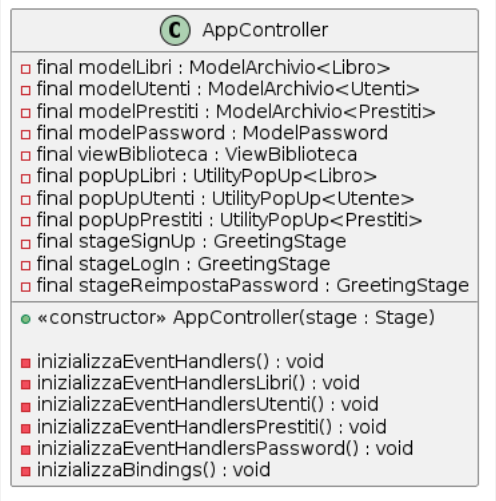
Livello di coesione

- **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



Questa classe astrae il concetto di controller del pattern MVC. Per questo motivo, l'unico servizio che offre in termini di metodi pubblici è la possibilità di essere istanziata. Tutto il lavoro che svolge invece è a suo carico. Come controller ha dunque la responsabilità di passare i dati dalla `view` al `model` e viceversa, istanziare tutti i componenti della `view` e del `model`, impostare gli event handlers e creare i bindings.

Attributi principali

- `modelLibri`: Model che astrae il concetto di archivio di libri.

- `modelUtenti`: Model che astrae il concetto di archivio degli utenti.
- `modelPrestiti`: Model che astrae il concetto di archivio dei prestiti.
- `modelPassword`: Model che astrae il concetto del contenitore 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.
- `popUpLibri`: View che astrae la finestra di pop up generica generabile dal premere uno dei bottoni nella tab dei libri.
- `popUpUtenti`: View che astrae la finestra di pop up generica generabile dal premere uno dei bottoni nella tab degli utenti.
- `popUpPrestiti`: View che astrae la finestra di pop up generica generabile dal premere uno dei bottoni nella tab dei prestiti.
- `stageSignUp`: View che astrae la finestra di sign up visualizzata prima dell'apertura della finestra principale.
- `stageLogIn`: View che astrae la finestra di log in visualizzata prima dell'apertura della finestra principale.
- `stageReimpostaPassword`: View che astrae la finestra di reimposta password visualizzata prima dell'apertura della finestra principale.

Metodi pubblici

- `AppController(stage)`: Costruttore che accetta una reference di tipo `Stage` come parametro di input; si occupa di istanziare gli attributi di `AppController` ed inizializzare gli event handlers e i bindings.

Relazioni rilevanti

- Associazione (unidirezionale) a `ModelArchivio` – 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) a `UtilityPopUp` – molteplicità 3: La classe `AppController` mantiene una reference dell'astrazione delle finestre di utilità pop up generabili dalle singole tab associate a ciascun tipo di dato gestito dall'applicazione (`Libri, Utenti, Prestiti`).
- Associazione (unidirezionale) a `GreetingStage` – molteplicità 3: La classe `AppController` mantiene una reference dell'astrazione delle finestre di benvenuto (`SignUp, LogIn, ReimpostaPassword`) per poter gestire i lori componenti grafici.

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: istanziare le parti principali e inizializzare gli event handlers ed i bindings.

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

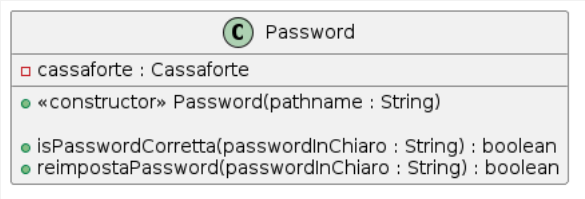
- `verificaPassword(passwordInChiaro)`: verifica che la password inserita come argomento coincida con quella memorizzata in archivio.
- `impostaPassword(passwordInChiaro)`: salva l'hash della password all'interno dell'archivio

---

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`, per questo 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.

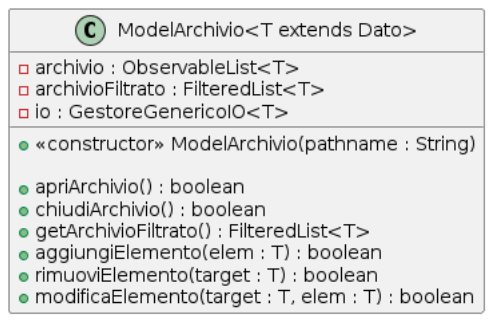
Livello di coesione

- **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.ModelArchivio`



Questa classe rappresenta il modello di archivio accessibile al 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 per potersi adattare alla necessità di astrarre il concetto di archivio dei `Libro`, degli `Utente` e dei `Prestito`.

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 accedendo a record contenuti nella cache (chiusura non corretta 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()`: Semplice 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.
- `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.
- `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.

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.

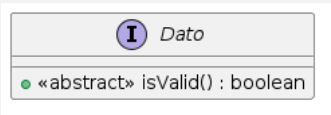
Livello di coesione

- **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.

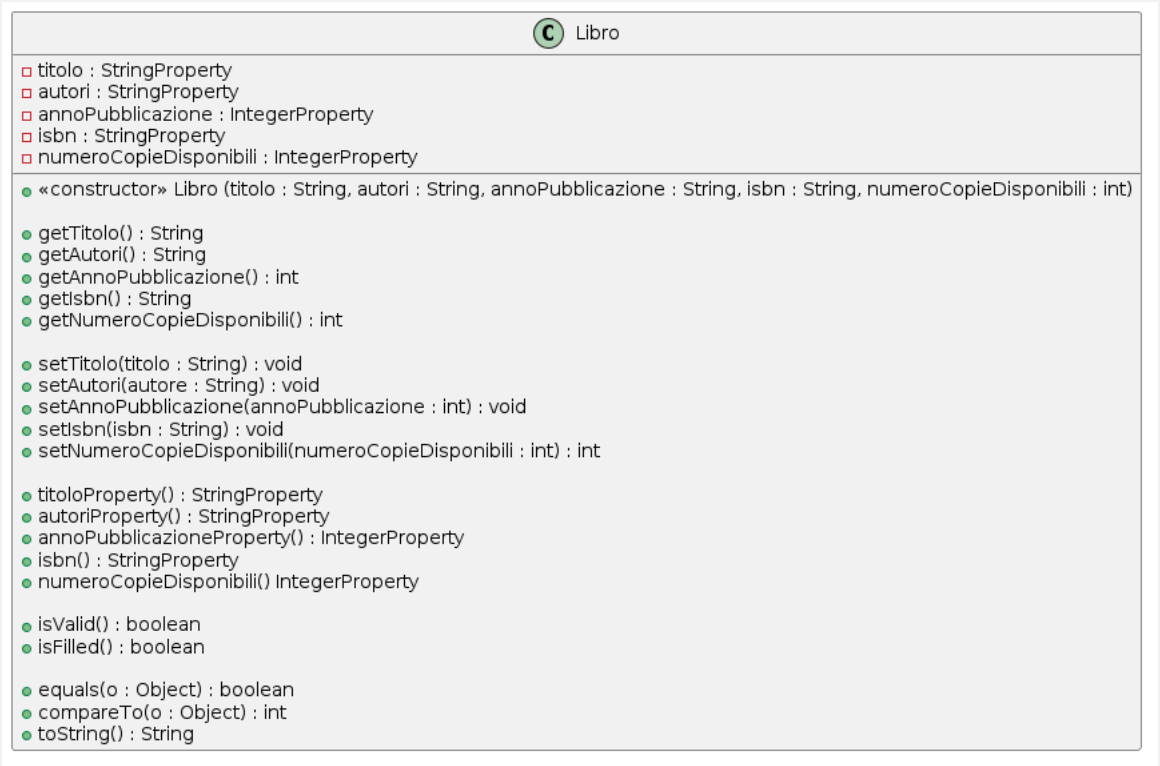
Metodi pubblici

- `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.

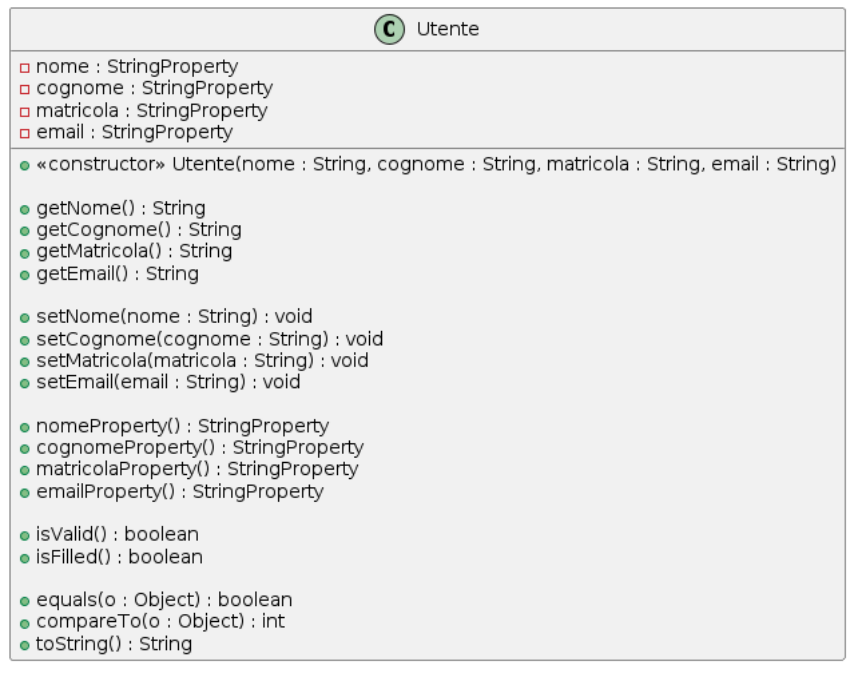
Livello di coesione

- **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.

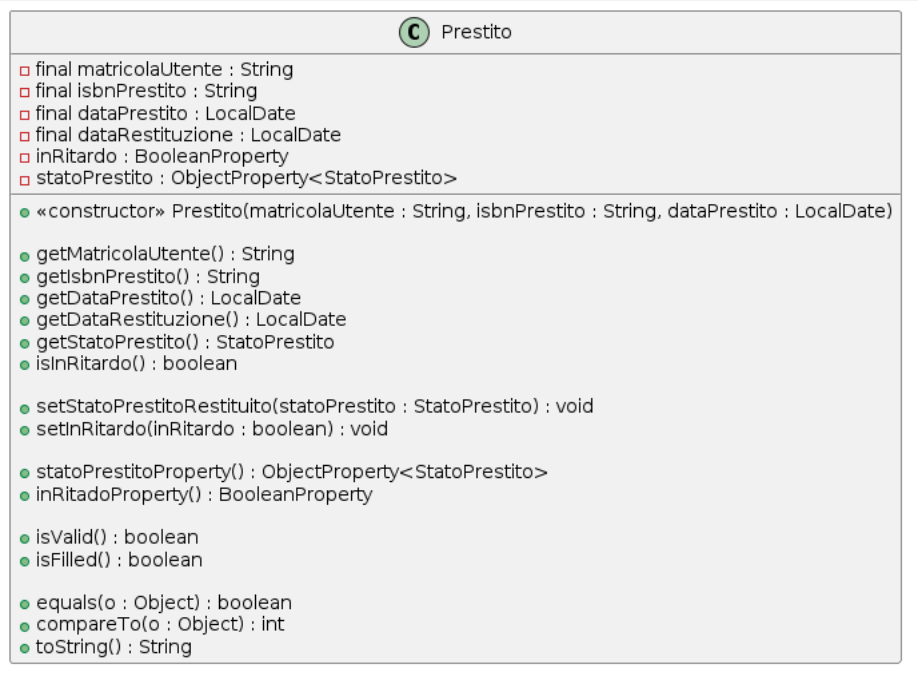
Livello di coesione

- **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.
- `inRitardo`: Flag che indica se l'utente è in ritardo nel restituire il libro.
- `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 la `matricolaUtente` rispetta lo standard e l'`isbnPrestito` rispetta il formato standard dei libri.
- `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.

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.

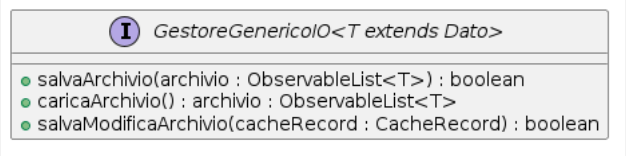
Metodi pubblici

- 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.

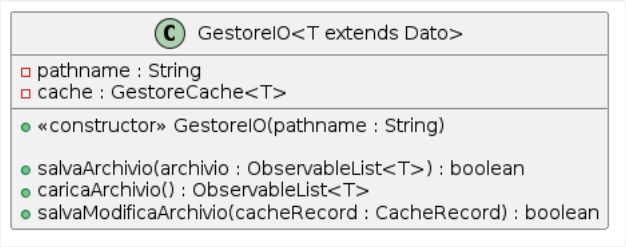
Metodi pubblici

- `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.

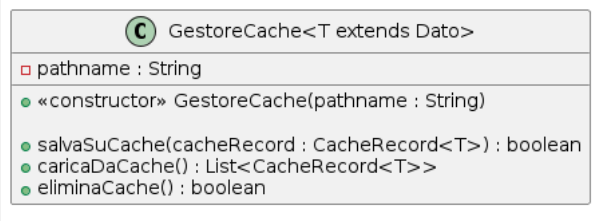
Livello di coesione

- **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; se la lista è vuota significa che il file è stato creato ex novo.
- `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.

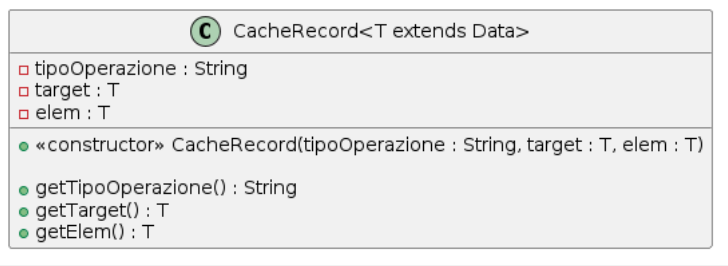
Livello di coesione

- **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

- `tipOperazione`: Stringa che 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.

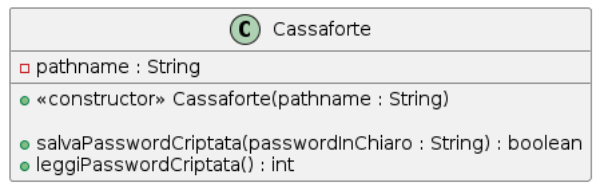
Livello di coesione

- **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.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.

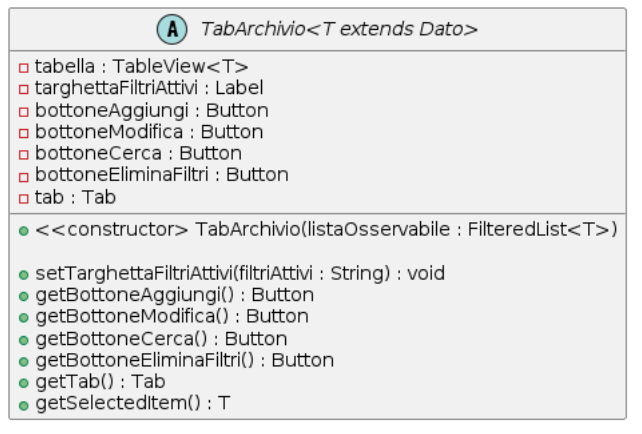
Livello di coesione

- **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.
- `targhettaFiltriAttivi`: Etichetta che indica i filtri attivi.
- `bottoneAggiungi`: Bottone per aggiungere un elemento nella tabella.
- `bottoneModifica`: Bottone per modificare un elemento nella tabella.
- `bottoneCerca`: Bottone per cercare un elemento nella tabella.
- `bottoneEliminaFiltri`: 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.
- `setTarghettaFiltriAttivi(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

- `bottoneCancella`: 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.
- `getBottoneCancella()`: Restituisce la reference al bottone di cancellazione.
- Metodi ereditati da `TabArchivio`: Getter dei bottoni, `setTarghettaFiltriAttivi()`, `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

- `bottoneCancella`: 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.
- `getBottoneCancella()`: Restituisce la reference al bottone di cancellazione.
- Metodi ereditati da `TabArchivio`: Getter dei bottoni, `setTarghettaFiltriAttivi()`, `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.

Attributi principali

- Nessun attributo aggiuntivo.

Metodi pubblici

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

Relazioni rilevanti

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

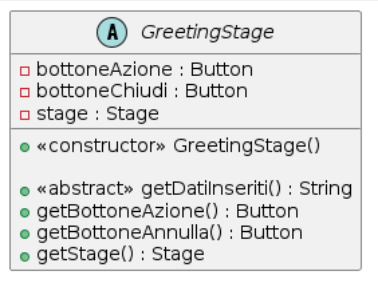
Livello di coesione

- **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.GreetingStage



Questa classe astrae il concetto di finestra di benvenuto, la cui responsabilità è permettere all'utente di accedere alla vista principale dell'applicazione.

Attributi principali

- `bottoneAzione`: Bottone per l'azione generica di accesso all'applicazione.
- `bottoneChiudi`: Bottone per la chiusura della finestra di benvenuto.
- `stage`: Lo stage associato alla finestra di benvenuto.

Metodi pubblici

- `GreetingStage()`: Costruttore che si occupa di impostare il layout base della finestra di benvenuto.
- `getDatiInseriti()`: Ottiene la stringa inserita dall'utente nel campo di testo della password.
- Getter per tutti gli attributi: Restituiscono le reference mantenute dagli attributi.

Relazioni rilevanti

- È associata (unidirezionale) ad `AppController`: Il controller mantiene una reference ad ogni istanza di `GreetingStage` per interagire con le rispettive finestre.
- È estesa da `GreetingStageSignUp`: Specializza la finestra di benvenuto per il caso di sign up.
- È estesa da `GreetingStageReimpostaPassword`: Specializza la finestra di benvenuto per il caso di reimpostazione password.
- È estesa da `GreetingStageLogIn`: Specializza la finestra di benvenuto per il caso di log in.

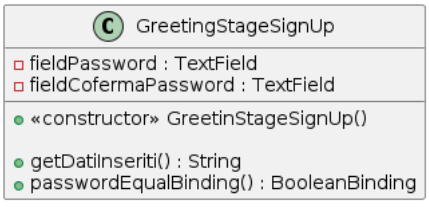
Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi servono unicamente a inizializzare la finestra di benvenuto e a recuperarne le componenti principali.

Livello di accoppiamento

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

View.GreetingStageSignUp



Questa classe specializza `GreetingStage` per adattarsi al caso di *sign up*, che si verifica quando il bibliotecario

non ha ancora registrato alcuna password per accedere all'applicazione.

Attributi principali

- `fieldPassword`: Campo di testo per inserire la password.
- `fieldConfermaPassword`: Campo di testo per reinserire la password e verificarne la correttezza.

Metodi pubblici

- `GreetingStageSignUp()`: Costruttore che aggiunge al layout base i campi di testo specifici per l'inserimento della password.
- `getDatiInseriti()`: Contratto ereditato da `GreetingStage`.
- `passwordEqualBinding()`: Restituisce una `BooleanBinding` che lega le proprietà dei due campi di testo verificando che il loro contenuto sia uguale.
- Getter per tutti gli attributi: Restituiscono le reference mantenute dagli attributi.

Relazioni rilevanti

- Estende `GreetingStage`: Questa classe concretizza la versione specializzata della finestra di benvenuto per il caso di sign up.

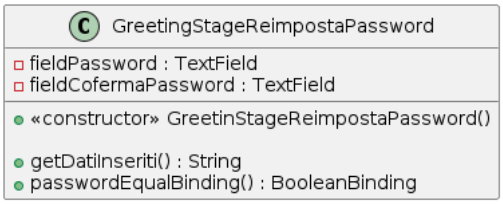
Livello di coesione

- **Coesione Funzionale**: I metodi e gli attributi sono esclusivamente dedicati all'inizializzazione e alla gestione della finestra di benvenuto per l'operazione di sign up.

Livello di accoppiamento

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

`View.GreetingStageReimpostaPassword`



Questa classe specializza `GreetingStage` per il caso di *reimpostazione della password*, che si verifica quando il bibliotecario sceglie di reimpostare la propria password dalla finestra di log in.

Attributi principali

- `fieldPassword`: Campo di testo per inserire la nuova password.
- `fieldConfermaPassword`: Campo di testo per confermare la password e verificarne la correttezza.

Metodi pubblici

- `GreetingStageReimpostaPassword()`: Costruttore che aggiunge al layout base i campi di testo necessari alla reimpostazione della password.
- `getDatiInseriti()`: Contratto ereditato da `GreetingStage`.
- `passwordEqualBinding()`: Restituisce una `BooleanBinding` che verifica l'uguaglianza tra i due campi di testo.

- Getter per tutti gli attributi: Restituiscono le reference mantenute dagli attributi.

Relazioni rilevanti

- Estende `GreetingStage`: Questa classe concretizza la finestra di benvenuto specifica per il caso di reimpostazione della password.

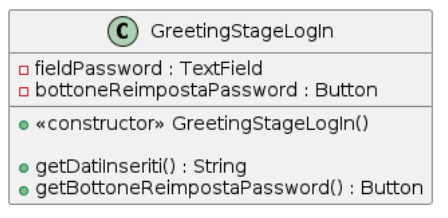
Livello di coesione

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

Livello di accoppiamento

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

`View.GreetingStageLogIn`



Questa classe specializza `GreetingStage` per il caso di *log in*, che si verifica quando il bibliotecario ha già registrato una password e deve inserirla per accedere all'applicazione.

Attributi principali

- `fieldPassword`: Campo di testo per l'inserimento della password.
- `bottoneReimpostaPassword`: Bottone che permette di accedere alla schermata di reimpostazione della password.

Metodi pubblici

- `GreetingStageLogIn()`: Costruttore che aggiunge al layout base della finestra di benvenuto gli elementi specifici del log in.
- `getDatiInseriti()`: Contratto ereditato da `GreetingStage`.
- `getBottoneReimpostaPassword()`: Restituisce la reference al bottone per reimpostare la password.
- Getter per gli altri attributi: Restituiscono le reference mantenute dagli attributi.

Relazioni rilevanti

- Estende `GreetingStage`: Questa classe concretizza la finestra di benvenuto nel caso in cui l'utente debba effettuare il log in.

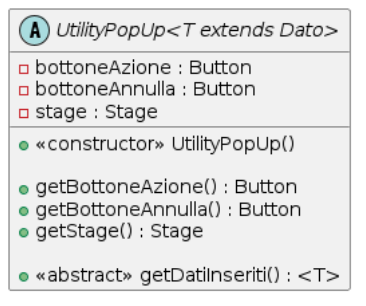
Livello di coesione

- **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.UtilityPopUp<T extends Dato>



Questa classe astrae il concetto di finestra pop up di utilità, utilizzata per permettere all'utente di inserire o modificare dati relativi agli archivi. Ha il compito di recuperare i dati inseriti dal gestore nel relativo form.

Attributi principali

- `bottoneAzione`: Bottone per eseguire l'azione richiesta sull'archivio.
- `bottoneAnnulla`: Bottone per chiudere la finestra pop up senza applicare modifiche.
- `stage`: Lo `Stage` della finestra pop up.

Metodi pubblici

- `UtilityPopUp()`: Costruttore che inizializza le componenti base della finestra pop up.
- `getDatiInseriti()`: Restituisce il dato costruito a partire dalle informazioni inserite dall'utente.
- Getter per gli attributi: Restituiscono le reference mantenute dagli attributi.

Relazioni rilevanti

- È associata ad `AppController`: Il controller mantiene reference a istanze di `UtilityPopUp` per gestire le operazioni dell'utente.
- Estesa da `UtilityPopUpLibri`: Specializzazione dedicata all'archivio dei libri.
- Estesa da `UtilityPopUpUtenti`: Specializzazione dedicata all'archivio degli utenti.
- Estesa da `UtilityPopUpPrestiti`: Specializzazione dedicata all'archivio dei prestiti.

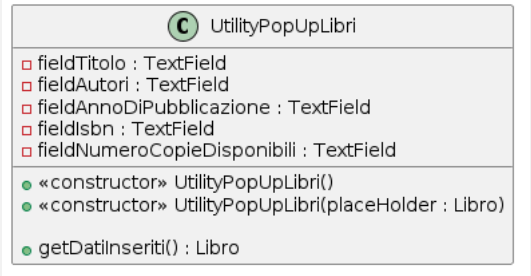
Livello di coesione

- **Coesione Funzionale**: Gli attributi e i metodi sono dedicati alla sola gestione della finestra pop up e al recupero dei dati inseriti.

Livello di accoppiamento

- **Accoppiamento per Dati**: La classe espone e utilizza solo i dati strettamente necessari all'esecuzione dei servizi previsti.

View.UtilityPopUpLibri



Questa classe specializza `UtilityPopUp` per il caso in cui la finestra di utilità venga generata dalla tab dell'archivio dei libri. La finestra ha il compito di recuperare e validare i dati inseriti dal gestore.

Attributi principali

- `fieldTitolo`: Campo di testo per il titolo del libro.
- `fieldAutori`: Campo di testo per la lista degli autori.
- `fieldAnnoDiPubblicazione`: Campo di testo per l'anno di pubblicazione.
- `fieldIsbn`: Campo di testo per il codice ISBN.
- `fieldNumeroDiCopieDisponibili`: Campo di testo per il numero di copie disponibili.

Metodi pubblici

- `UtilityPopUpLibri()`: Costruttore che imposta il layout della finestra pop up.
- `UtilityPopUpLibri(placeHolder)`: Costruttore sovraccarico che imposta il layout assegnando ai campi di testo dei valori di placeholder.
- `getDatiInseriti()`: Restituisce un oggetto `Libro` costruito con i dati inseriti dall'utente.
- Getter per tutti gli attributi: Restituiscono le reference mantenute dagli attributi.

Relazioni rilevanti

- Estende `UtilityPopUp<T extends Dato>`: La classe concretizza la finestra pop up generica specializzandola per i libri.

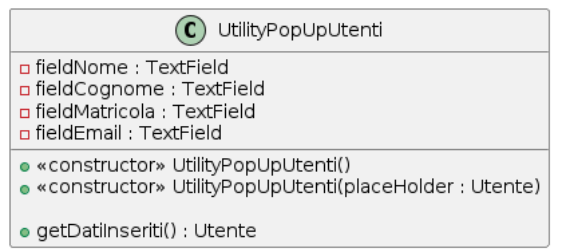
Livello di coesione

- **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.UtilityPopUpUtenti`



Questa classe specializza `UtilityPopUp` per il caso in cui la finestra di utilità venga generata dalla tab dell'archivio degli utenti. La finestra ha il compito di recuperare i dati inseriti dal gestore.

Attributi principali

- `fieldNome`: Campo di testo per il nome dell'utente.
- `fieldCognome`: Campo di testo per il cognome dell'utente.
- `fieldMatricola`: Campo di testo per la matricola dell'utente.
- `fieldEmail`: Campo di testo per l'e-mail istituzionale dell'utente.

Metodi pubblici

- `UtilityPopUpUtenti()`: Costruttore che imposta il layout della finestra pop up.
- `UtilityPopUpUtenti(placeHolder)`: Costruttore sovraccarico che imposta il layout assegnando valori di placeholder ai campi di testo.
- `getDatiInseriti()`: Restituisce un oggetto `Utente` costruito con i dati inseriti dall'utente.
- Getter per tutti gli attributi: Restituiscono le reference mantenute dagli attributi.

Relazioni rilevanti

- Estende `UtilityPopUp<T extends Dato>`: La classe concretizza la finestra pop up generica specializzandola per i dati relativi agli utenti.

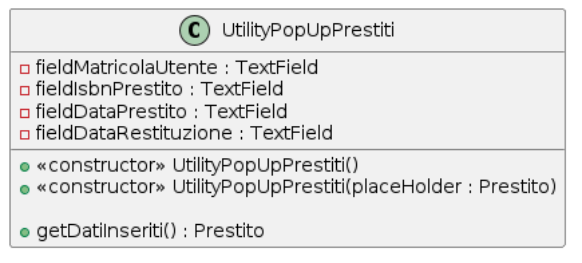
Livello di coesione

- **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.UtilityPopUpPrestiti`



Questa classe specializza `UtilityPopUp` 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

- `fieldMatricolaUtente`: Campo di testo per la matricola dell'utente.
- `fieldIsbnPrestito`: Campo di testo per l'ISBN relativo al prestito.
- `fieldDataPrestito`: Campo di testo per la data di prestito.
- `fieldDataRestituzione`: Campo di testo per la data massima di restituzione.

Metodi pubblici

- `UtilityPopUpPrestiti()`: Costruttore che imposta il layout della finestra pop up.
- `UtilityPopUpPrestiti(placeHolder)`: Costruttore sovraccarico che imposta il layout assegnando valori placeholder ai campi di testo.
- `getDatiInseriti()`: Restituisce un oggetto `Prestito` costruito con i dati inseriti dall'utente.
- Getter per tutti gli attributi: Restituiscono le reference mantenute dagli attributi.

Relazioni rilevanti

- Estende `UtilityPopUp<T extends Dato>`: La classe concretizza la finestra pop up generica specializzandola per i dati relativi ai prestiti.

Livello di coesione

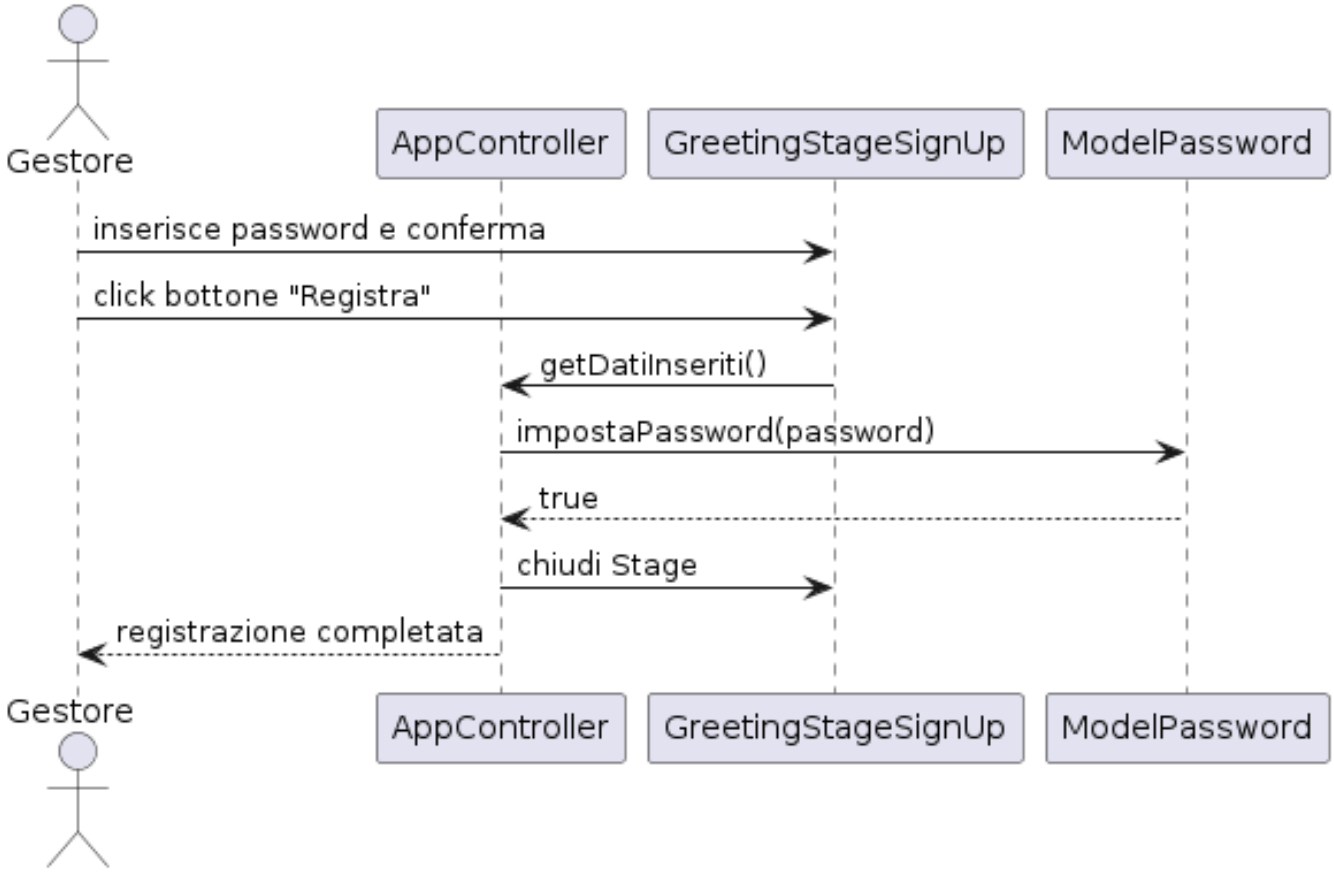
- **Coesione Funzionale**: Gli attributi e i metodi servono esclusivamente all'inizializzazione della finestra pop up e alla raccolta dei dati relativi ai prestiti.

Livello di accoppiamento

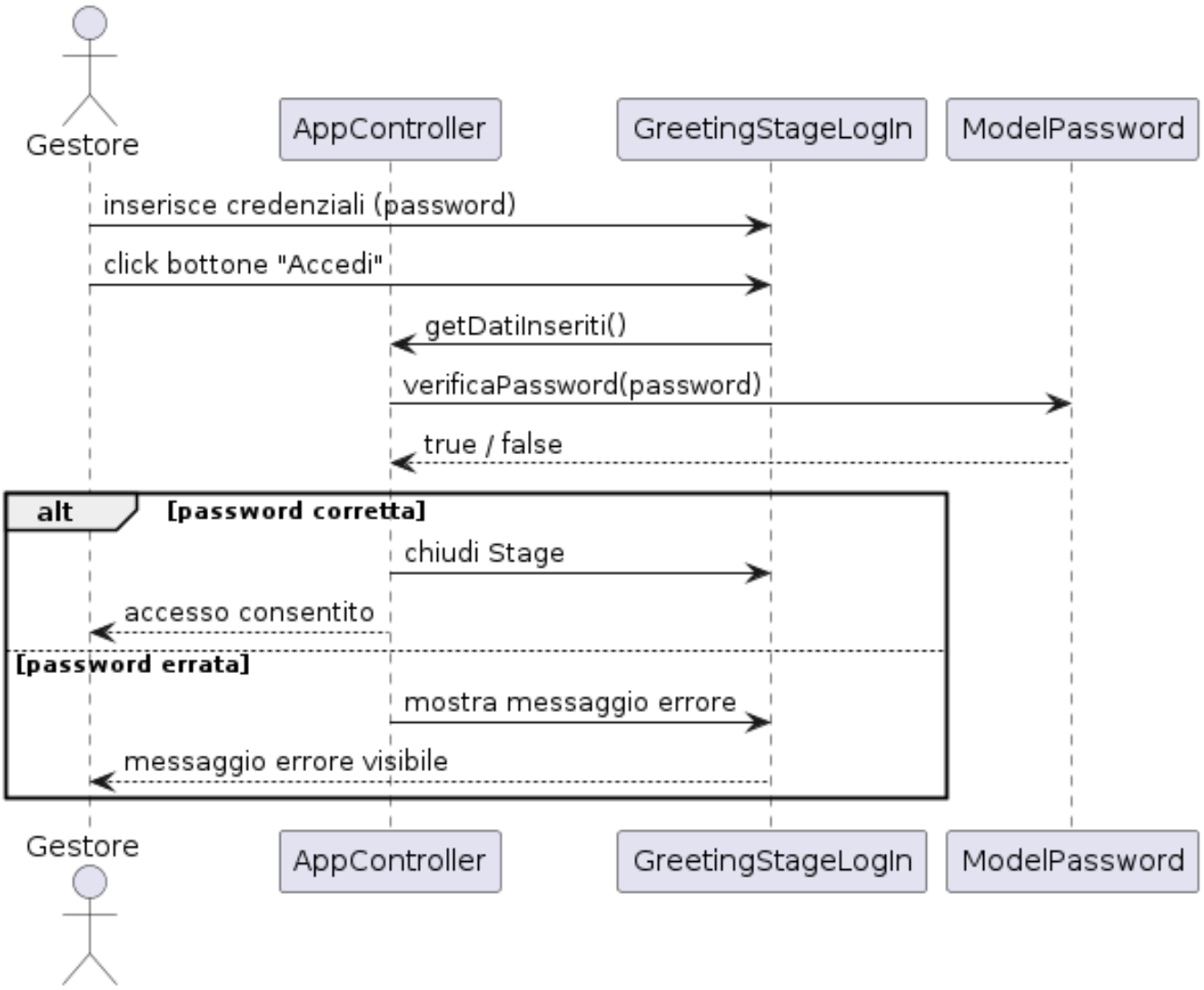
- **Accoppiamento per Dati**: La classe accetta e restituisce solo i dati strettamente necessari ai servizi previsti.

3 Modello Dinamico

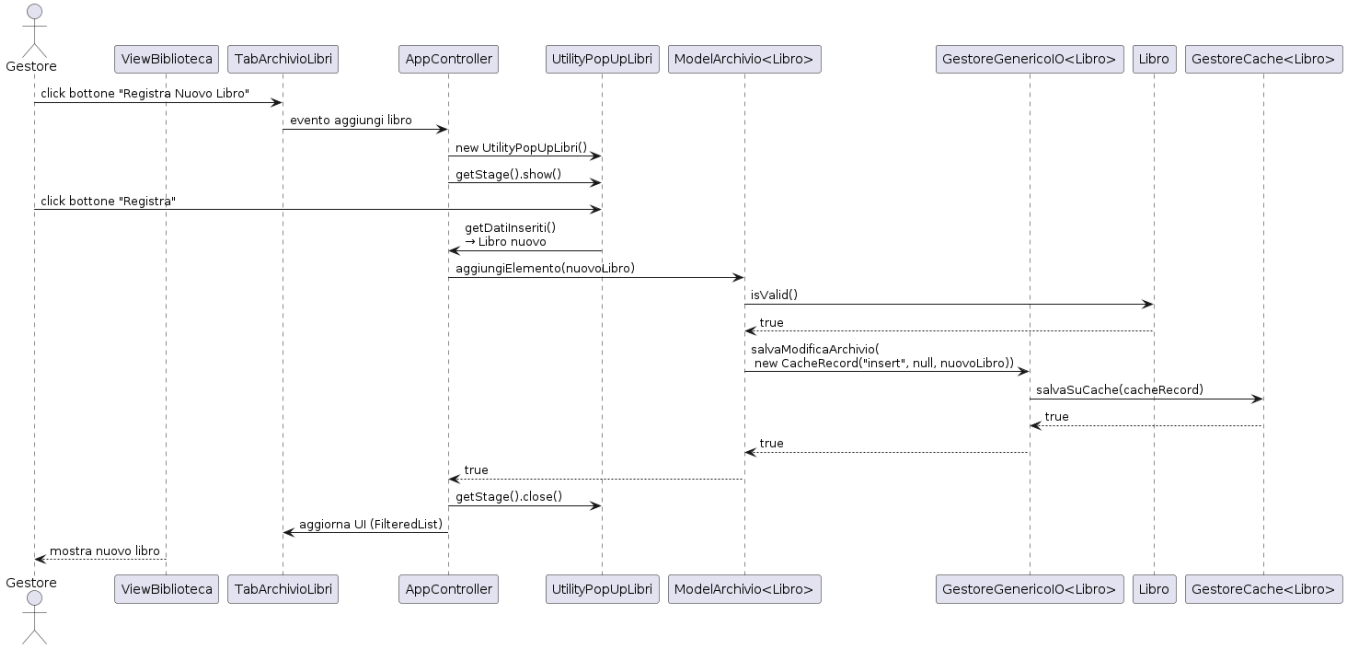
3.1 CU-1.1 - Sign Up



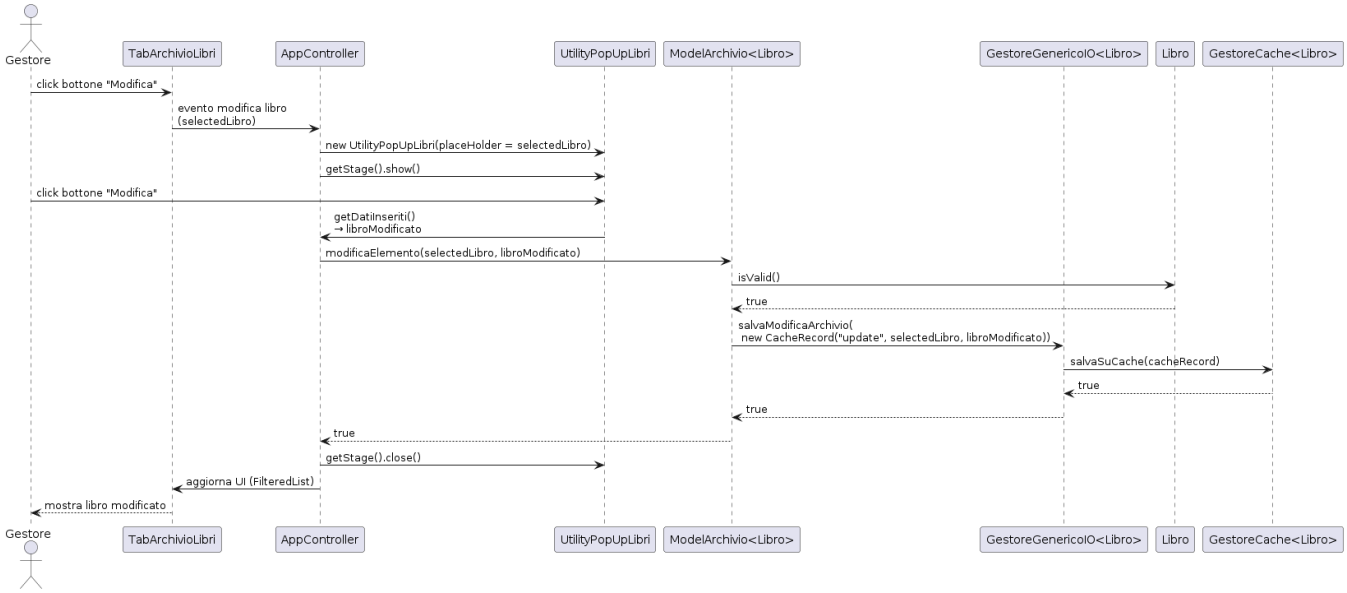
3.2 CU-1.2 - Login



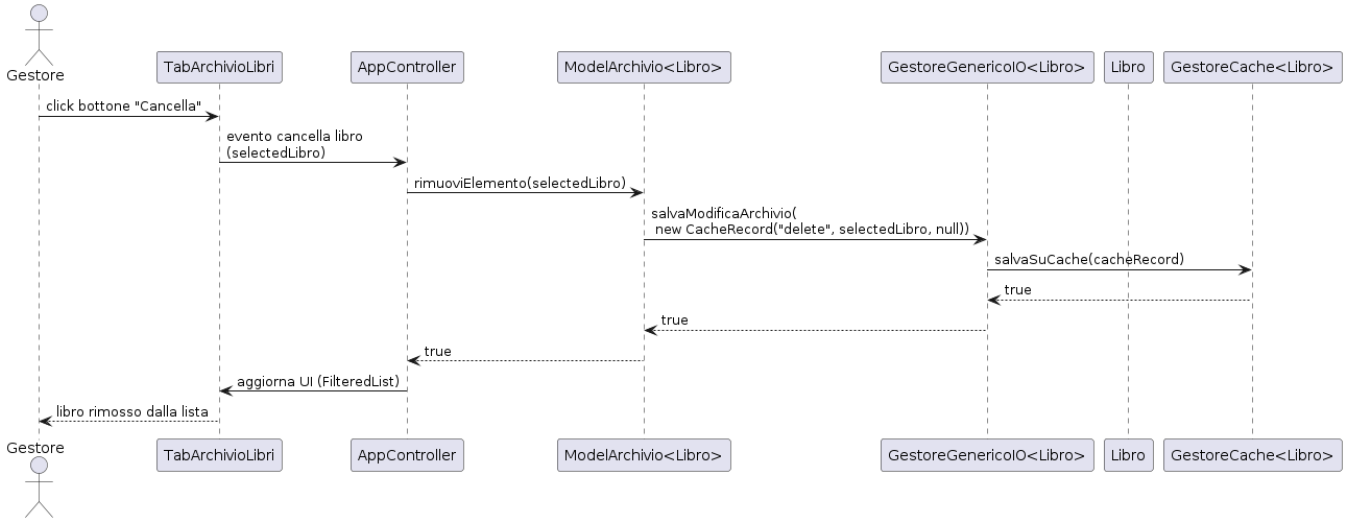
3.3 CU-2.1 - Inserimento Libro



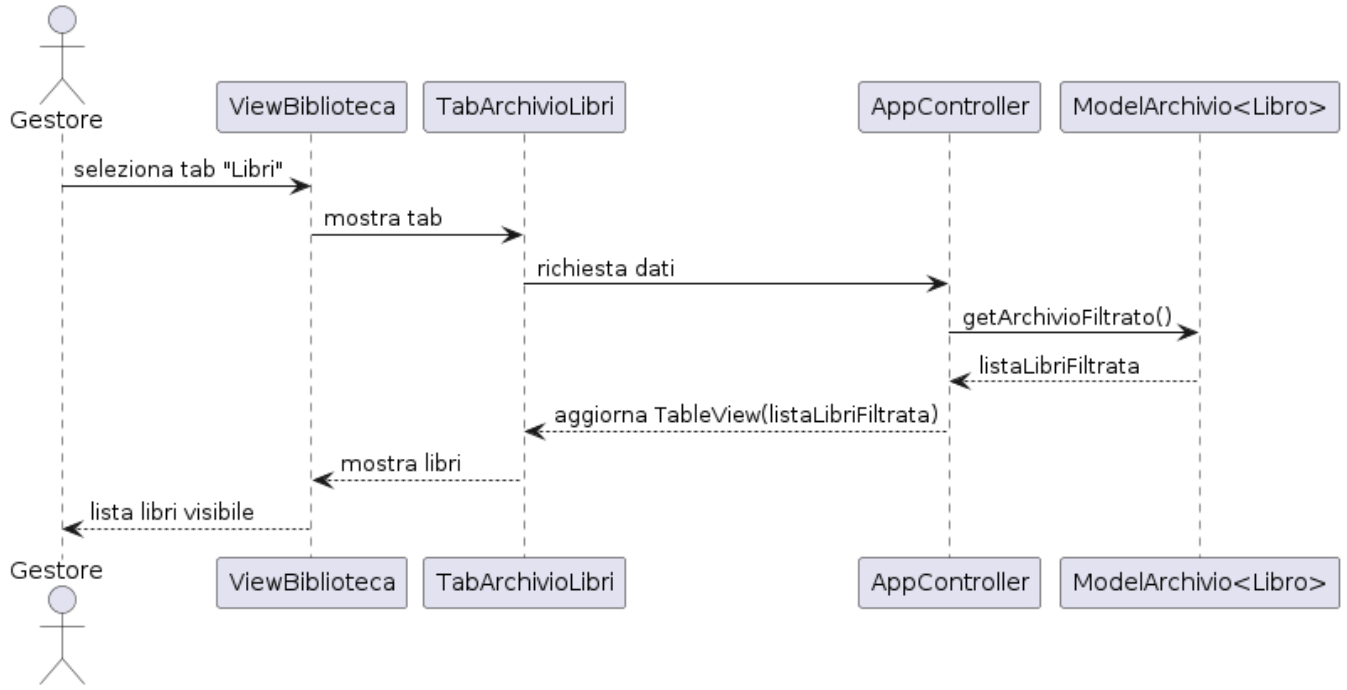
3.4 CU-2.2 - Modifica Dati Libro



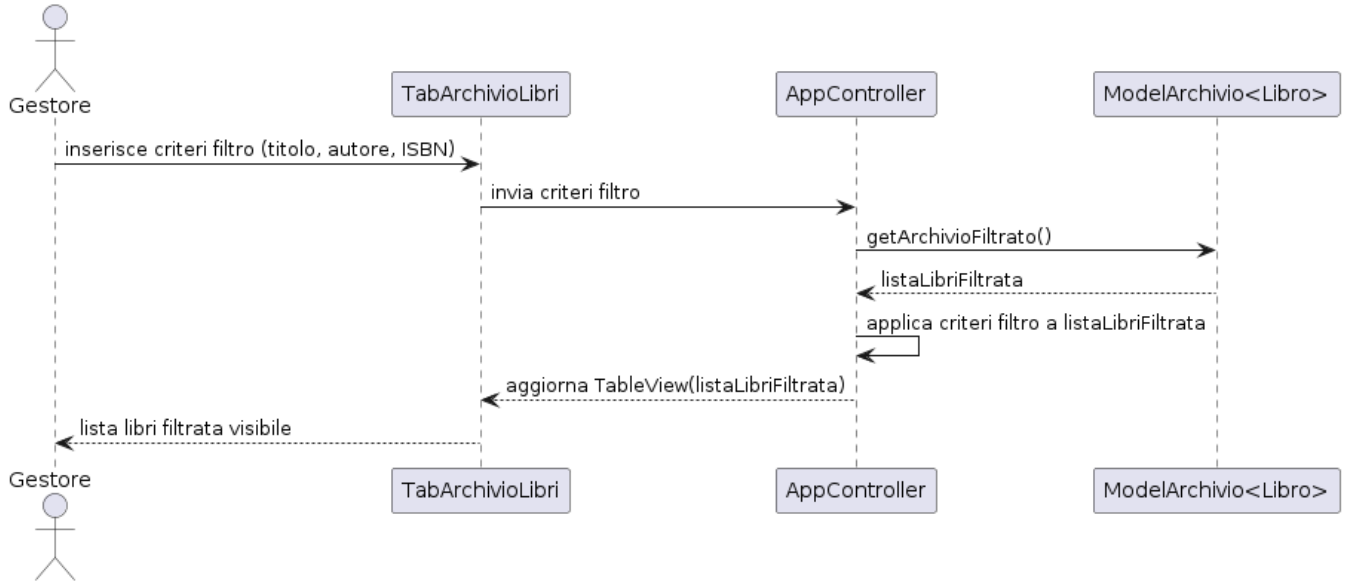
3.5 CU-2.3 - Cancellazione Libro



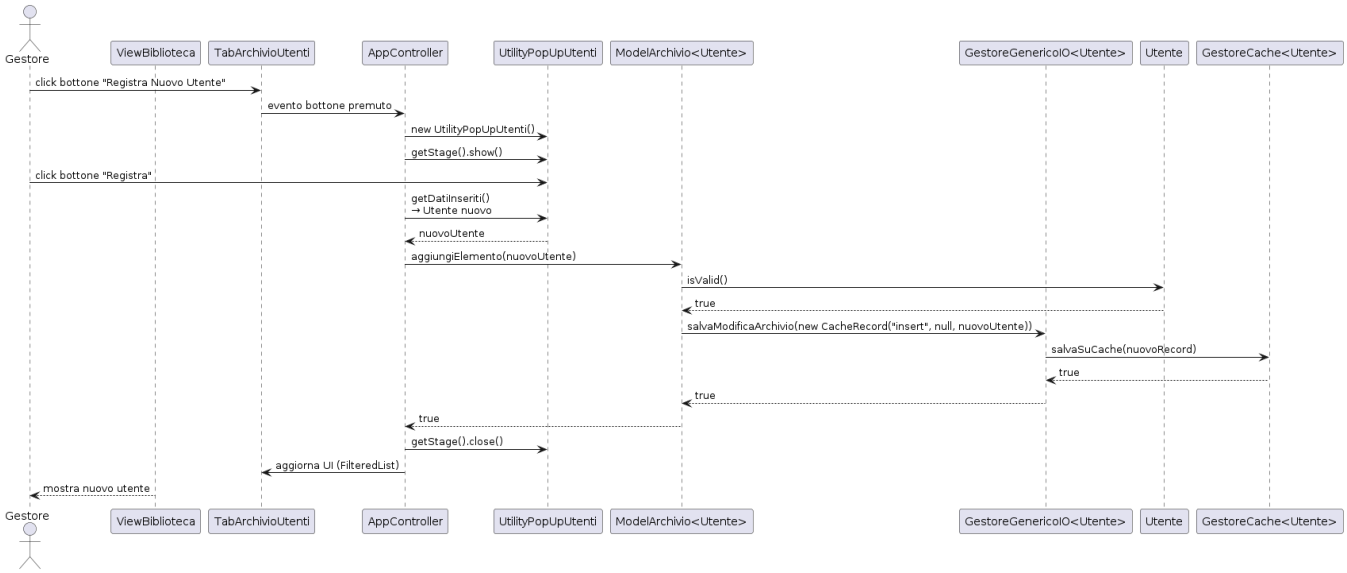
3.6 CU-2.4 - Visualizzazione Archivio Libri



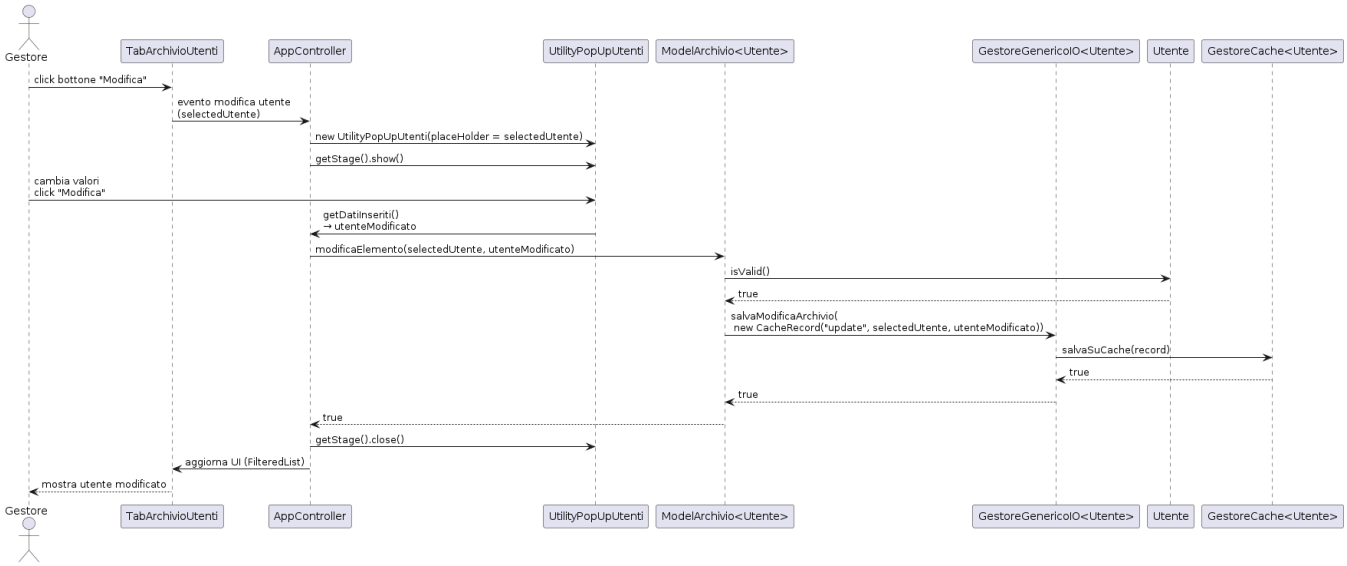
3.7 CU-2.5 - Ricerca Libro



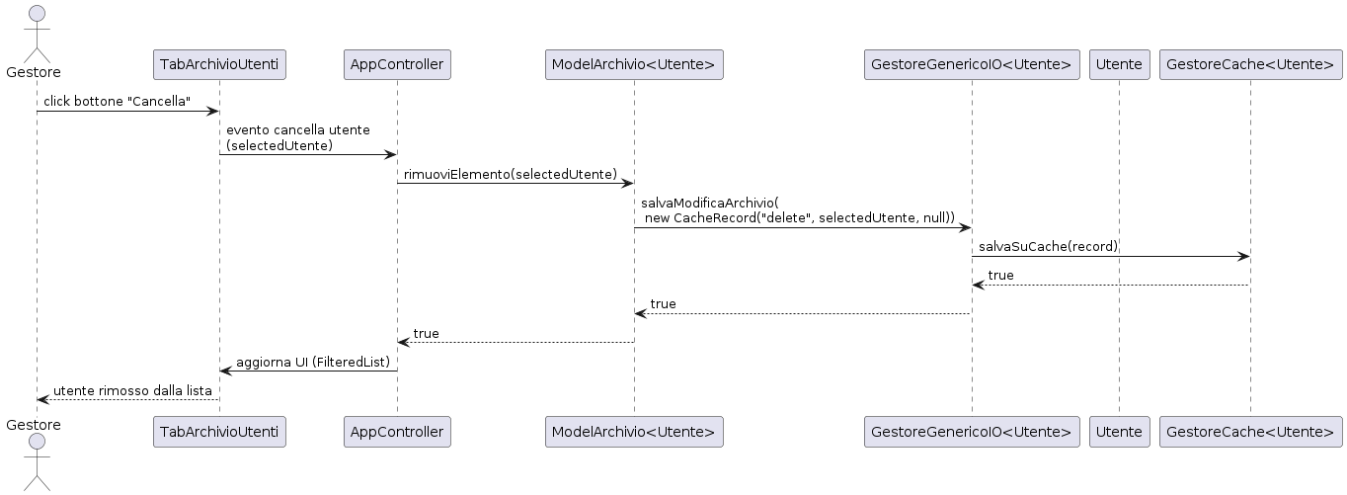
3.8 CU-3.1 - Inserimento Nuovo Utente



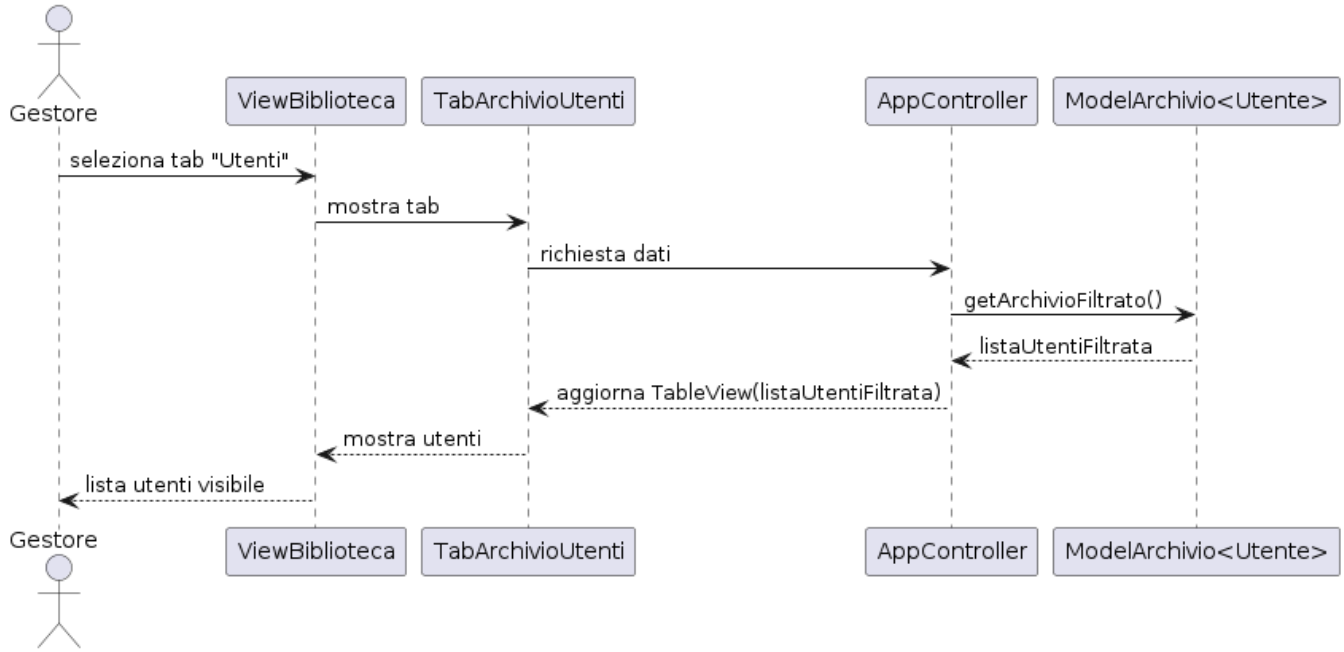
3.9 CU-3.2 - Modifica Dati Utente



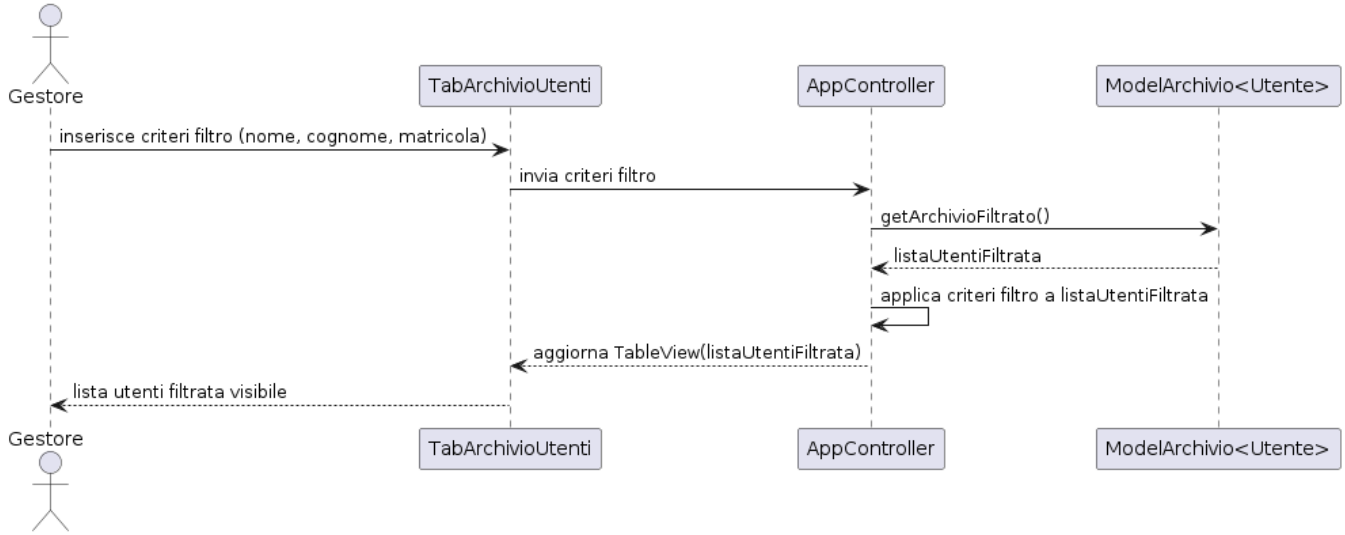
3.10 CU-3.3 - Cancellazione Utente



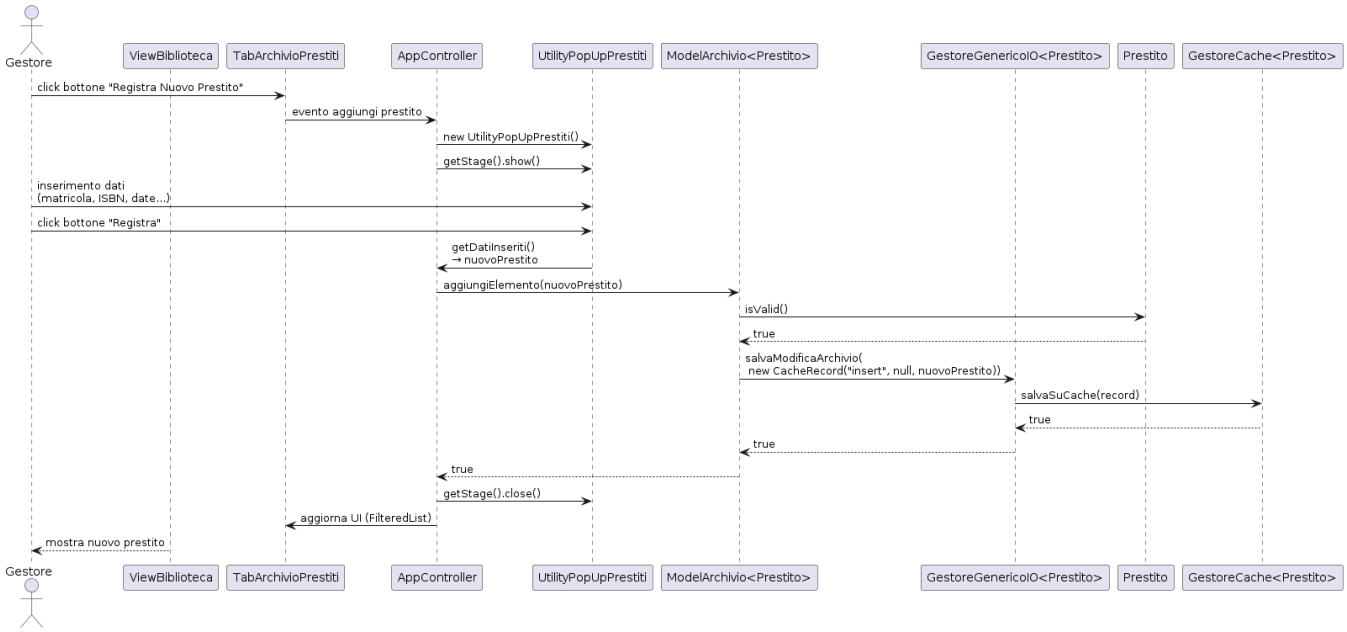
3.11 CU-3.4 - Visualizzazione Archivio Utenti



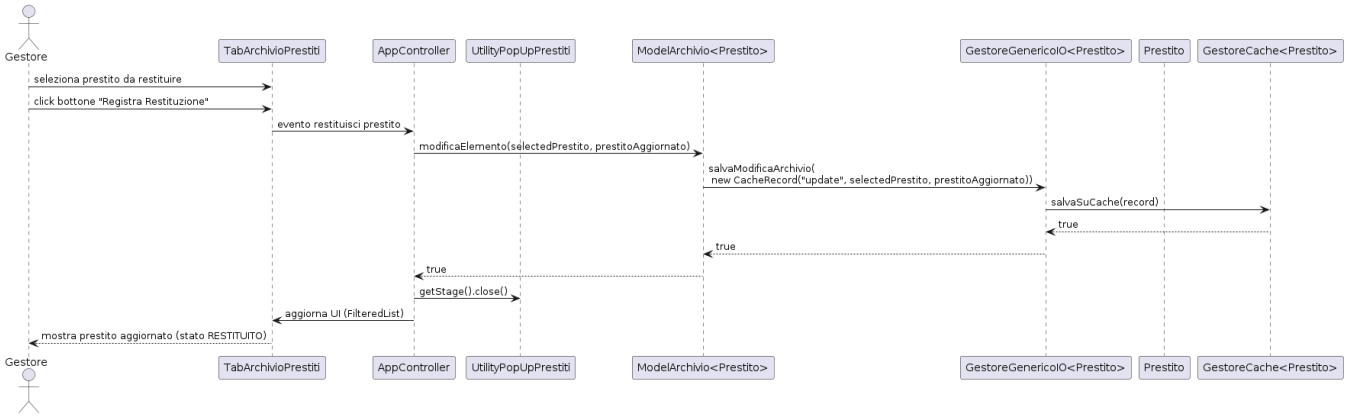
3.12 CU-3.5 - Ricerca Utente



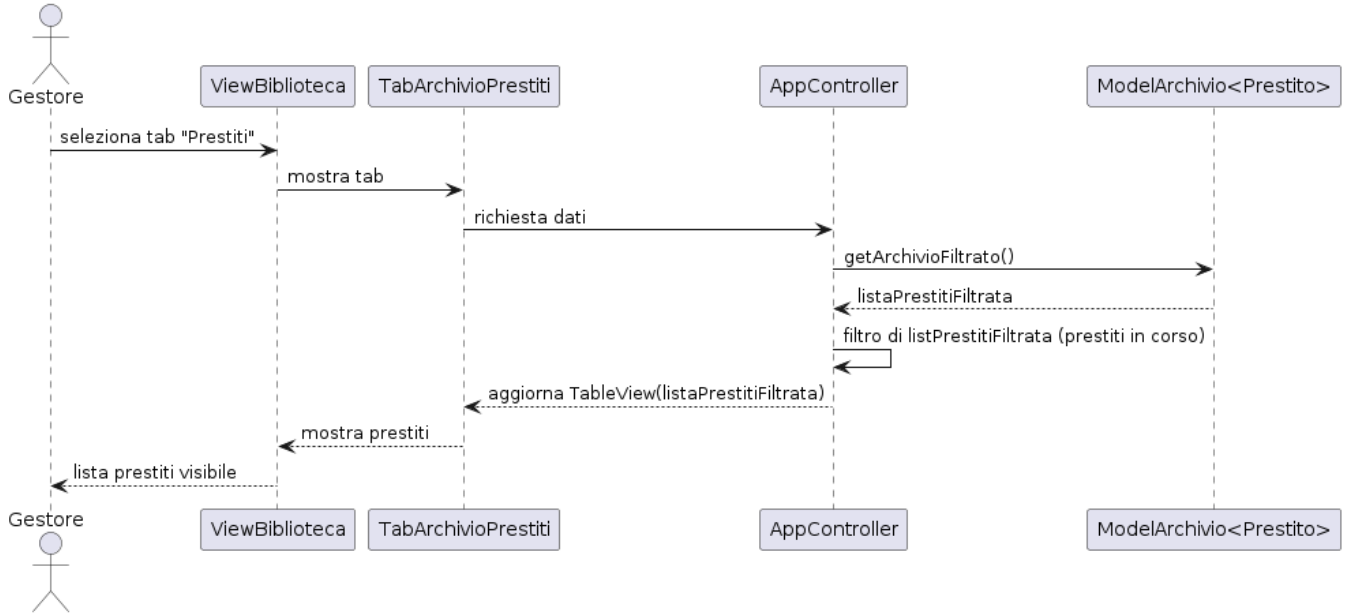
3.13 CU-4.1 - Registrazione Nuovo Prestito



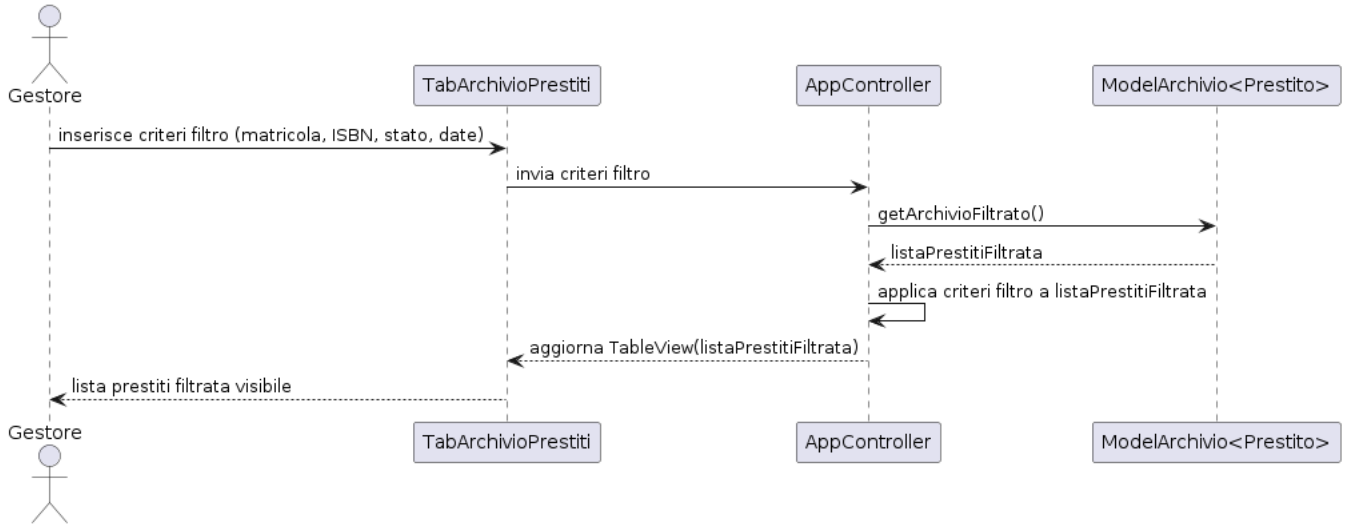
3.14 CU-4.2 - Registrazione Restituzione Libro



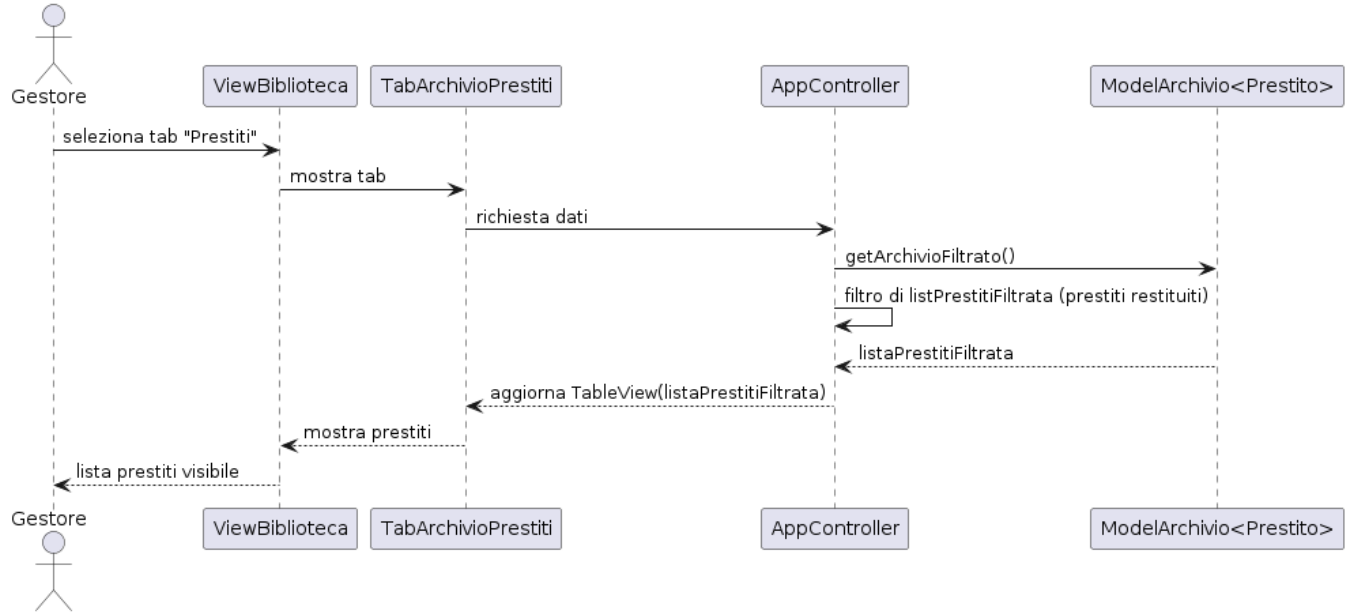
3.15 CU-4.3 - Visualizzazione Archivio Prestiti in Corso



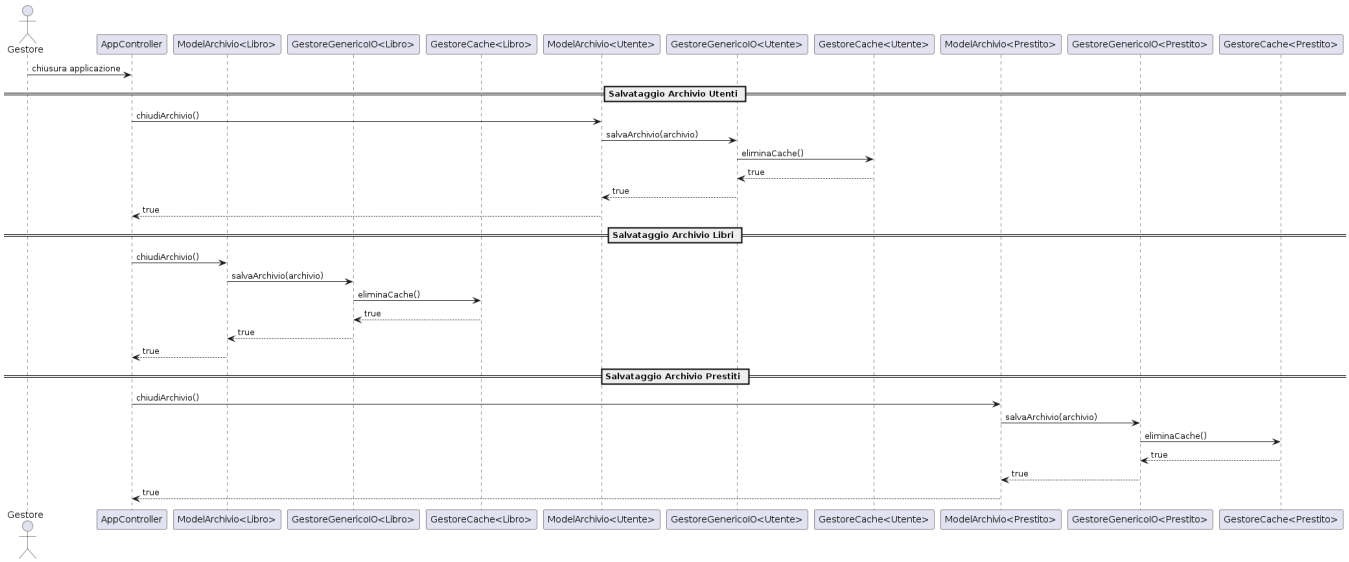
3.16 CU-4.4 - Ricerca Prestito



3.17 CU-4.5 - Visualizzazione Storico Prestito

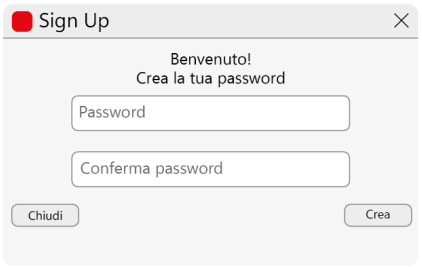


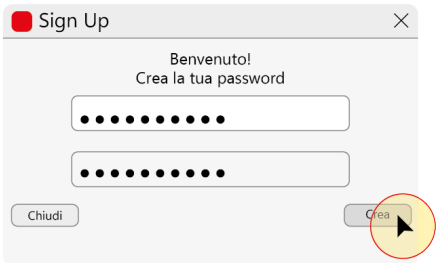
3.18 CU-5.1 - Chiusura Applicazione

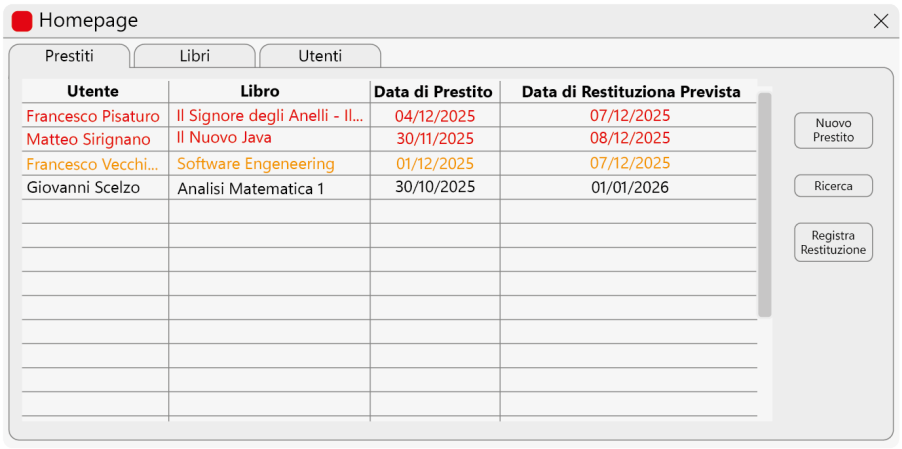


4 Design dell'Interfaccia Utente

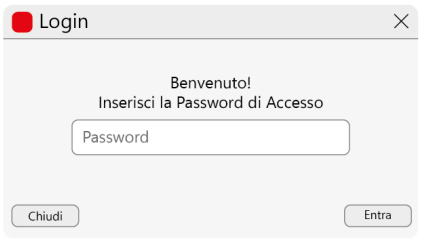
4.1 CU-1.1 - Sign Up

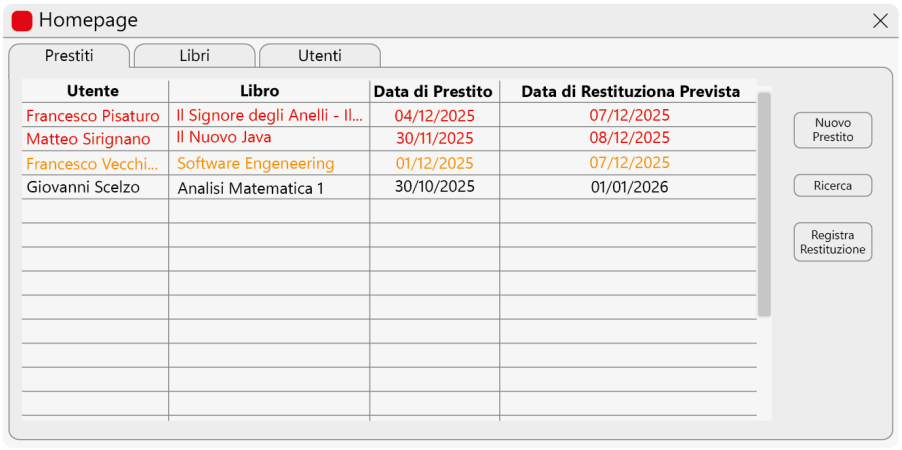




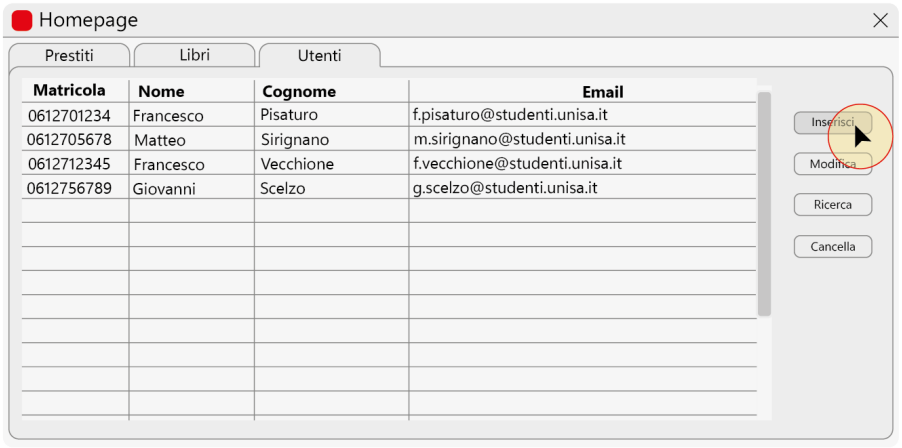


4.2 CU-1.2 - Login



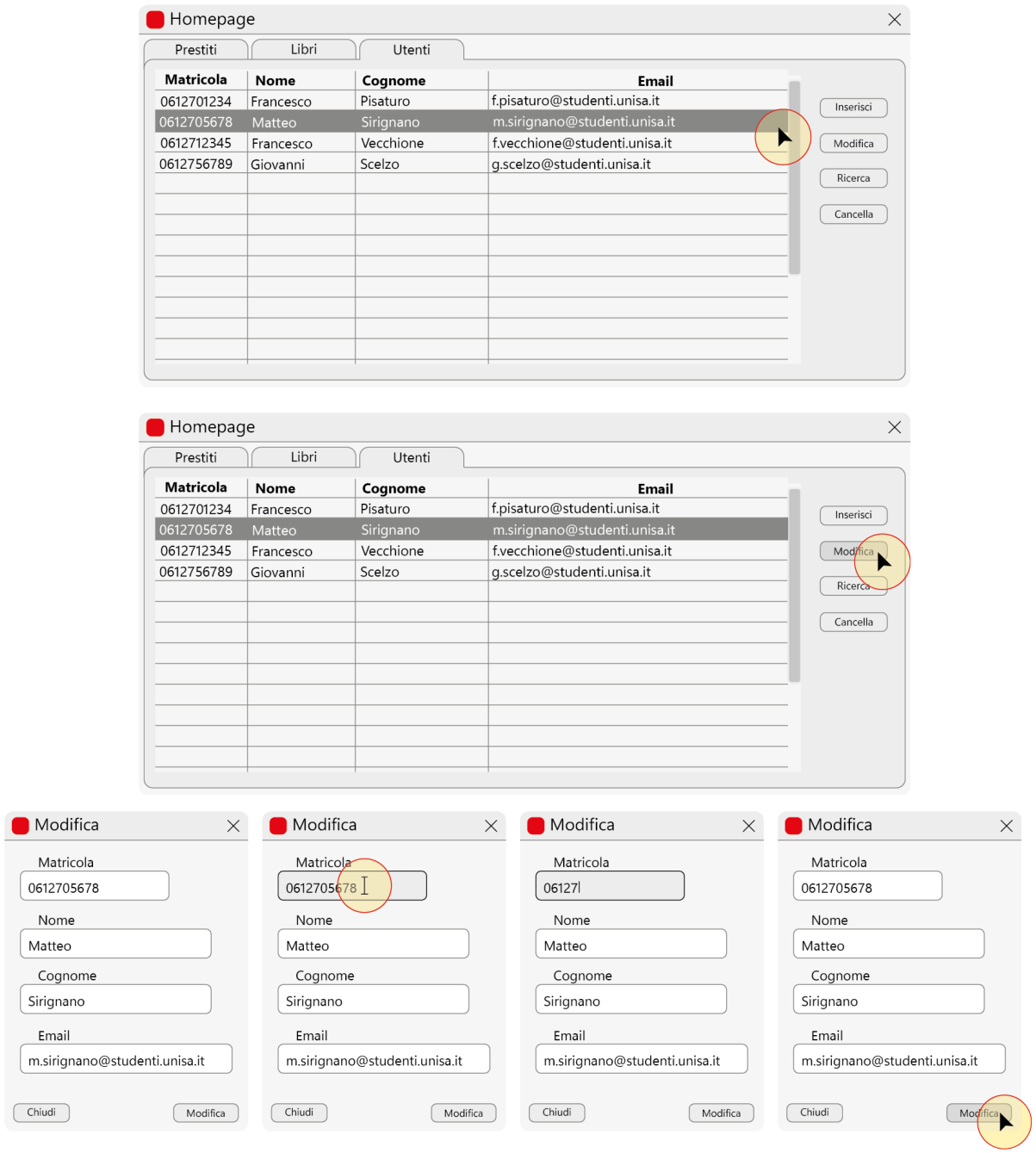


4.3 CU-3.1 - Inserimento Nuovo Utente



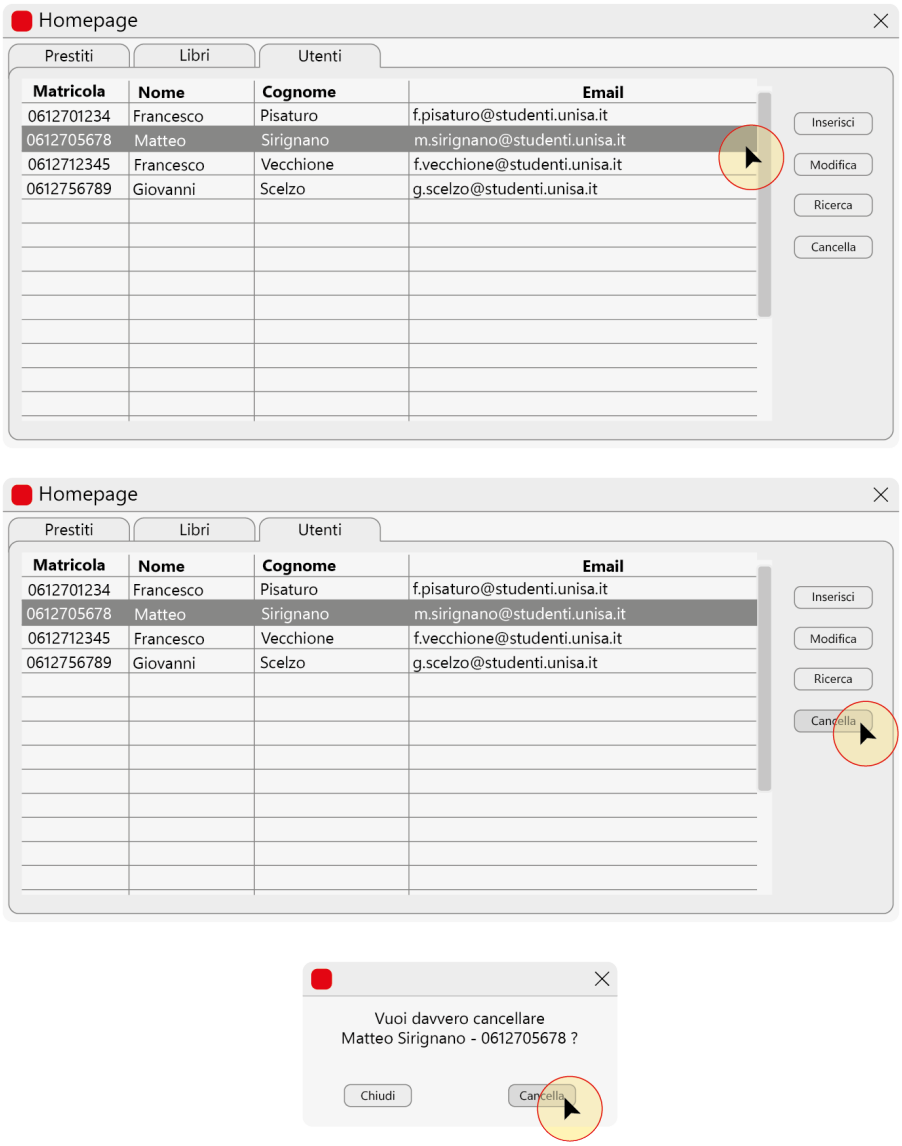
l'interazione con l'interfaccia è la medesima per il caso d'uso CU-2.1

4.4 CU-3.2 - Modifica Dati Utente



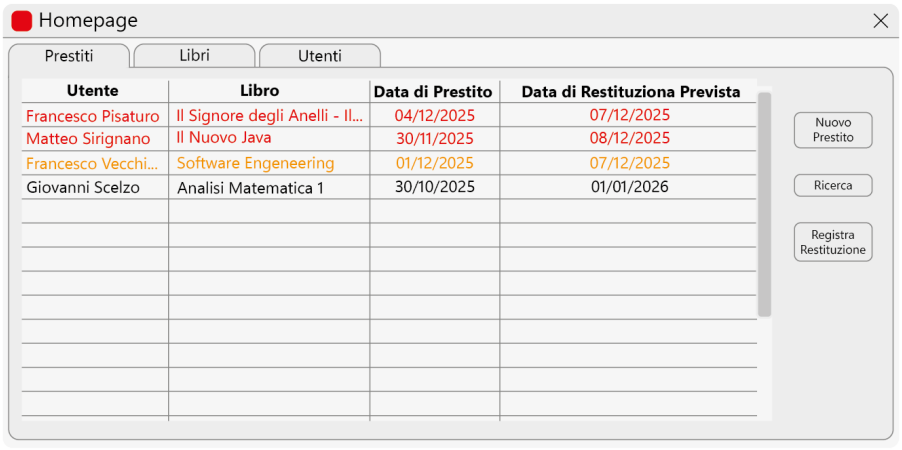
l'interazione con l'interfaccia è la medesima per il caso d'uso CU-2.2

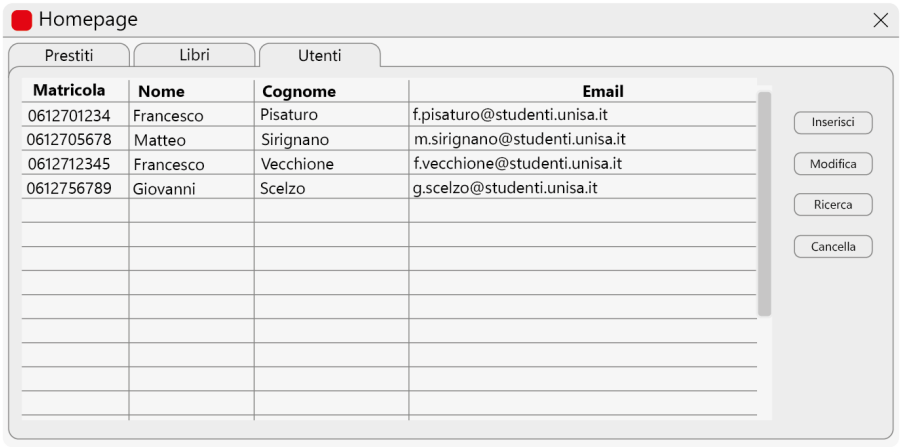
4.5 CU-3.3 - Cancellazione Utente



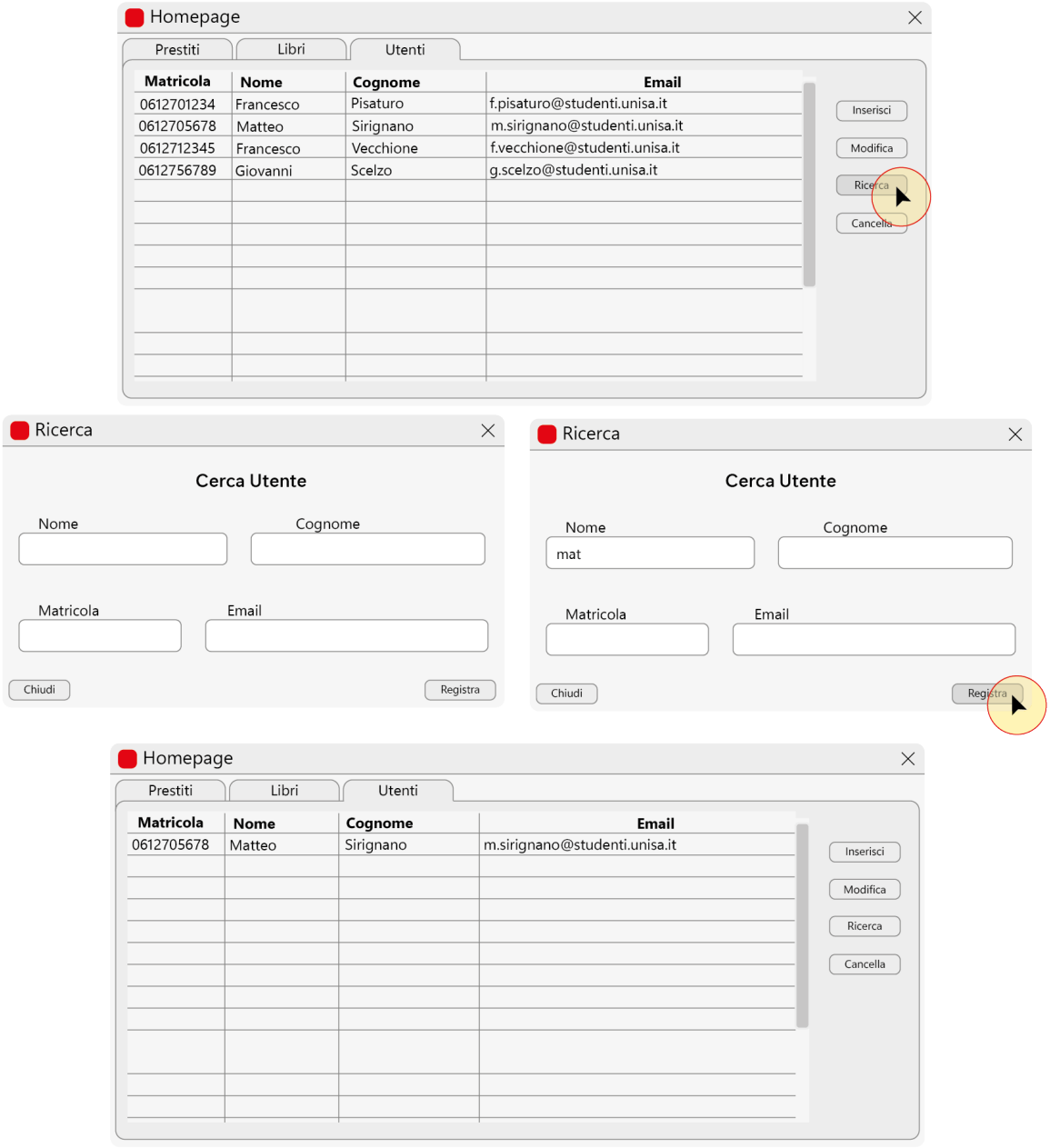
l'interazione con l'interfaccia è la medesima per il caso d'uso CU-2.3

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



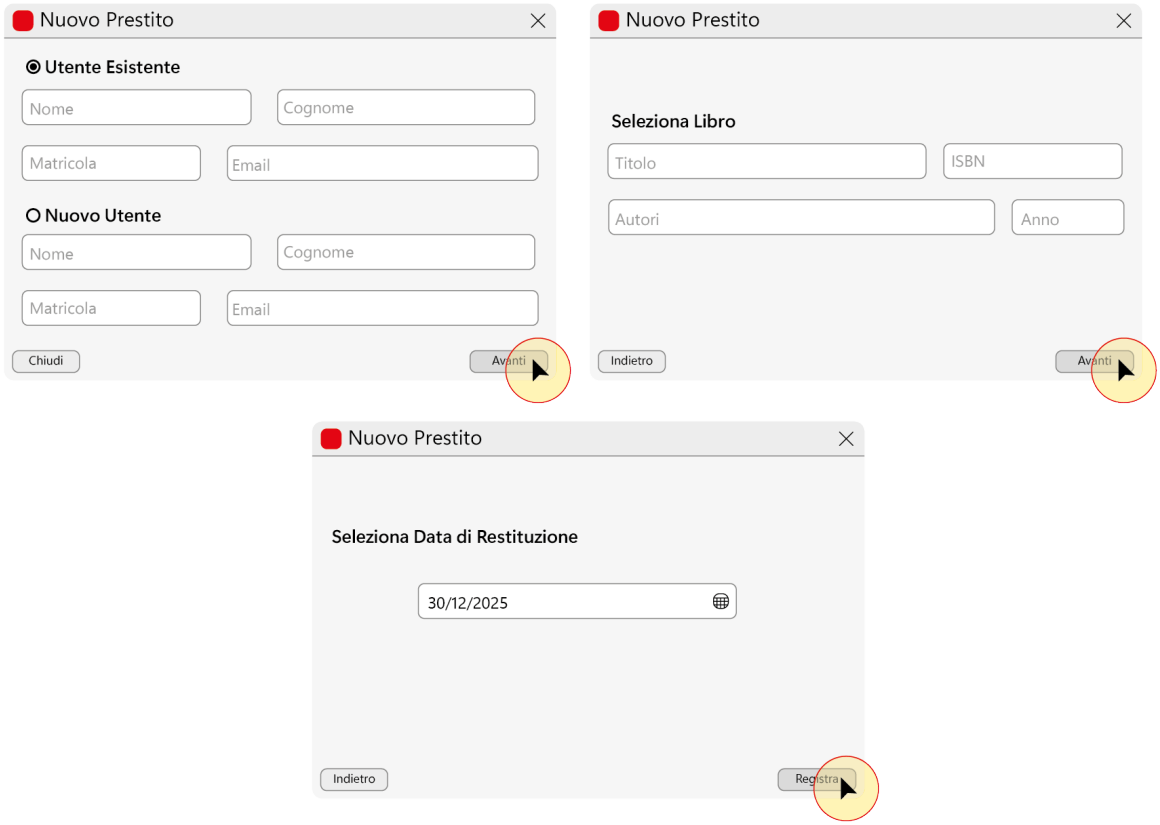


4.7 CU-3.5 - Ricerca Utente



*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