

# DOCUMENTO DI PROGETTAZIONE

## 1. Architettura Del Sistema

*Documentazione sull'utilizzo di pattern architettonici. Descrizione dei moduli principali, delle loro responsabilità e dipendenze (con diagramma dei package).*

### 1.1 Introduzione Struttura Generale dell'Architettura

*Documentazione sull'utilizzo di pattern architettonici.*

L'applicazione implementa un sistema per la gestione di una biblioteca universitaria utilizzando l'**architettura MVC** (Model-View-Controller).

La scelta di tale architettura deriva dalla necessità di realizzare un sistema modulare (ogni **modulo** si occupa di **responsabilità definite** e dai **confini precisi**), facilmente manutenibile e con una chiara separazione delle responsabilità tra i componenti.

Questa architettura risponde direttamente ai requisiti funzionali e non funzionali del progetto.

L'architettura MVC suddivide l'applicazione in tre componenti principali:

- **model** : si occupa dell'interazione con il database MySQL, della validazione delle informazioni e della struttura delle entità, è suddiviso a sua volta in:
  - **dataclass** : contiene le entità principali (Libro, Utente, EmailInfo e Autore).
  - **servizi** : contiene la logica applicativa vera e propria.
- **view** : rappresenta l'interfaccia grafica con cui il bibliotecario interagisce. Comprende le schermate di login/logout, la visualizzazione del catalogo, degli utenti, dei prestiti e i messaggi di errore o conferma.
- **controller** : intermediario tra view e model. Riceve gli input del bibliotecario, invoca la logica del model e aggiorna la view con i risultati delle operazioni. Suddiviso a sua volta in:
  - **catalogo** : si occupa della gestione del catalogo.
  - **utenti** : gestisce tutte le operazioni relative agli utenti memorizzati nel sistema.
  - **prestitorestituzione** : controlla le operazioni legate ai prestiti dei libri.
  - **modificapassword** : contiene le operazioni relative al cambio della password del bibliotecario.

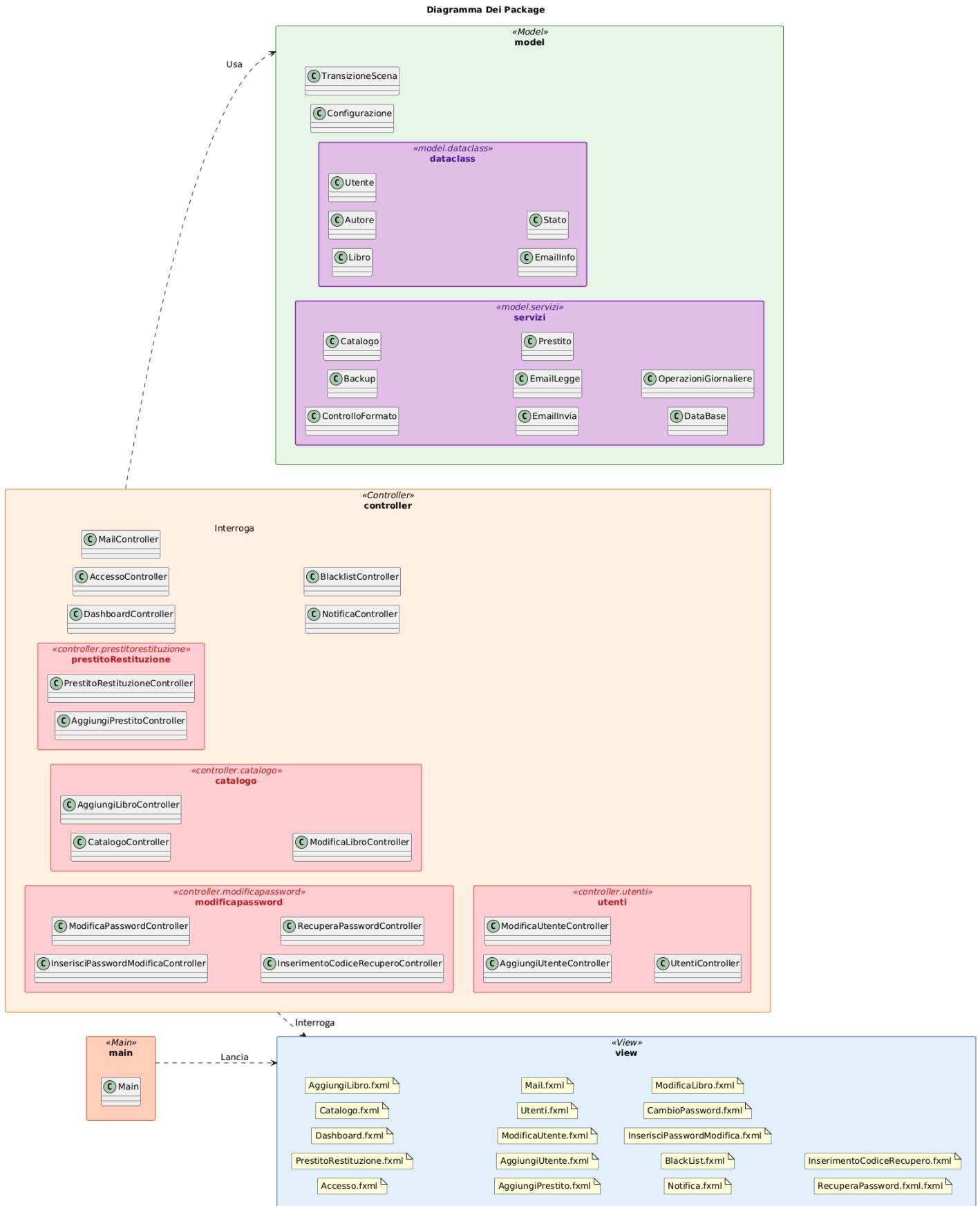
## **1.2 Moduli Principali**

*Moduli Principali, Responsabilità, Dipendenze, Ruolo nel Pattern e Requisiti associati.*

Nome	Responsabilità principali	Dipendenze	Ruolo nel Pattern (MVC)	Requisiti associati
Accesso e Gestione account	Il modulo gestisce l'accesso, l'uscita e la sicurezza dell'account del bibliotecario. Le responsabilità principali sono: registrazione, login, logout e cambio password.	Classe DataBase (MySQL)	Controller	IF-01 a IF-04; BF-01 a BF-03; DF-04; UI-01, UI-02
Gestione Catalogo	Il modulo gestisce l'intero catalogo biblioteca. Le responsabilità principali sono: inserimento, modifica, eliminazione, ricerca e visualizzazione dei libri.	Classe Libro, Classe Autore, Classe Catalogo, Classe DataBase (MySQL)	Controller	IF-05 a IF-09; BF-04, BF-05; DF-01; UI-04
Gestione Prestiti	Il modulo gestisce i prestiti, le restituzione e le estensioni della data di restituzione. Le responsabilità principali sono: registrazione prestito e restituzione, estensioni e consultazione archivio.	Classe Prestito, Classe Libro, Classe Utente, Enum Stato, Classe DataBase	Controller	IF-10 a IF-15; BF-06, BF-07; DF-01 a DF-03; UI-05
Gestione Utenti	Il modulo gestisce l'anagrafica degli utenti della biblioteca. Le responsabilità principali sono: inserimento, modifica, eliminazione utenti e ricerca nell'archivio.	Classe Utente, Classe DataBase (MySQL), Classe ControlloFormato	Controller	IF-16 a IF-18, IF-21, IF-22; BF-08, BF-09; DF-02; UI-06
Gestione Black-list	Il modulo gestisce utenti inadempienti e bloccati. Le responsabilità principali sono: aggiungere, rimuovere utenti dalla blacklist e consultazione del loro stato.	Classe Utente, Classe DataBase, Classe EmailInvia	Controller	IF-19, IF-20; BF-11; DF-02; UI-09
Gestione Notifiche	Il modulo gestisce notifiche interne ed email riguardo ai prestiti. Le sue responsabilità principali sono: notifiche scadenza e invio email agli utenti.	Classe Prestito, Classe EmailInvia, Classe EmailLegge, Servizi Email (SMTP/IMAP)	Controller	IF-23; BF-10; DF-03, DF-05; UI-08; IS-01
Gestione Backup	Il modulo crea copie di sicurezza del database. Le sue responsabilità principali sono: gestione backup del database tramite procedure MySQL.	Classe DataBase, DBMS MySQL, mysqldump	Servizio/Utility	IF-24; BF-12; IS-02
DataBase e Persistenza dati	Il modulo gestisce la memorizzazione e l'integrità dei dati. Le sue responsabilità sono la gestione delle strutture dati, le relazioni, l'integrità e la persistenza.	MySQL	Model	DF-01 a DF-04; IS-02

Interfaccia Utente (UI)	Il modulo gestisce l'area dedicata alle schermate e le interazioni del bibliotecario. Le sue responsabilità sono: visualizzazione del catalogo, utenti, prestiti, form, notifiche e messaggi di errore.	Tutti i Controller	View	Tutte le UI-01 a UI-09
Integrazione Servizi e-mail	Il modulo gestisce la comunicazione con il server SMTP per inviare e-mail. Le responsabilità principali sono: invio e-mail, gestione errori e comunicazione esterna.	Classe EmaillInvia, Server SMTP/TLS, Server IMAP	Servizio/Utility	IS-01; DF-05; IF-23

## 1.3 Diagramma Dei Package



## **2. Modello Statico**

Descrizione delle classi (responsabilità, attributi principali, metodi pubblici e relazioni rilevanti). Documentazione sulle scelte progettuali in termini di coesione, accoppiamento e principi di buona progettazione.

### **2.1 Descrizione Delle Classi**

Di seguito vengono descritte le classi principali del sistema, suddivise per package architetturale (Model, View, Controller), con i dettagli derivanti dall'implementazione attuale.

#### **PACKAGE model**

##### **Nome Classe: TransizioneScena**

- **Responsabilità:** Gestione transizioni tra view.
- **Attributi:**
  - Nessuno
- **Metodi Principali:**
  - *switchSceneEffect(Stage stage, String fxmlPath) : void* - Cambio di scena con FadeTransition.
- **Relazioni:**
  - Usato da *DashboardController* per caricare FXML.

 TransizioneScena
+switchSceneEffect(stage: Stage, fxmlPath: String) : void

##### **Nome Classe: Configurazione**

- **Responsabilità:** Gestione del caricamento e dell'accesso centralizzato ai parametri di configurazione dell'applicazione.
- **Attributi:**
  - *properties : Properties* - contenitore delle proprietà lette dal file config.properties.
- **Metodi Principali:**
  - *getEmailUsername() : String* - Restituisce lo username email.
  - *getPasswordSender() : String* - Restituisce la password del mittente.
  - *getPasswordReceiver() : String* - Restituisce la password del destinatario.
  - *getMaxUsers() : int* - Numero massimo utenti.
  - *getMaxBooks() : int* - Numero massimo libri.
  - *getMaxLoans() : int* - Numero massimo prestiti.
  - *getMaxAuthors() : int* - Numero massimo autori.
  - *getMaxWrited() : int* - Numero massimo scrittori collegati.
  - *getTimeOpen() : int[]* - Ora e minuto di apertura.
  - *getTimeClose() : int[]* - Ora e minuto di chiusura.

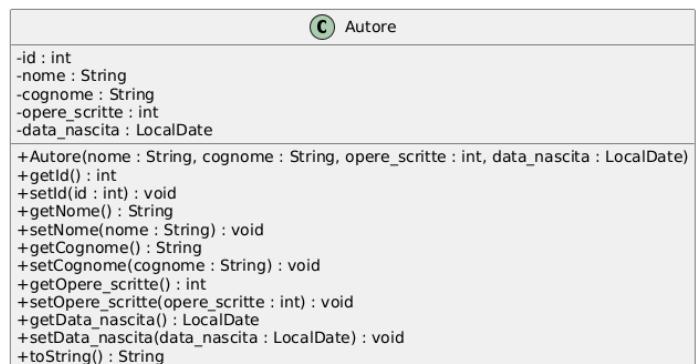
 Configurazione
-properties : Properties
+getEmailUsername() : String
+getPasswordSender() : String
+getPasswordReceiver() : String
+getMaxUsers() : int
+getMaxBooks() : int
+getMaxLoans() : int
+getMaxAuthors() : int
+getMaxWrited() : int
+getTimeOpen() : int[]
+getTimeClose() : int[]
+getSessionDuration() : int
+parseInteger(key: String, defaultValue: int) : int

- `getSessionDuration() : int` - Durata sessione utente.
- `parseInteger(String key, int defaultValue) : int` - Converte una proprietà in intero o restituisce il valore di default.
- **Relazioni:**
  - Usata da più controller dell'applicazione che necessitano di leggere valori di configurazione.
  - Non dipende da altre classi poiché sfrutta solo `java.util.Properties` e il `classpath`.

## PACKAGE `model.dataclass`

### Nome Classe: Autore

- **Responsabilità:** Rappresenta un autore di libri con i relativi dati anagrafici.
- **Attributi:**
  - `id : int` - Identificativo univoco database.
  - `nome : String` - Nome dell'autore.
  - `cognome : String` - Cognome dell'autore.
  - `opere_scritte : int` - Contatore delle opere scritte.
  - `data_nascita : LocalDate` - Data di nascita.
- **Metodi Principali:**
  - `Autore(String nome, String cognome, int opere_scritte, LocalDate data_nascita)` - Costruttore.
  - `getId() : int, setId(int id) : void` - Accesso all'ID.
  - `getNome() : String, setNome(String nome) : void` - Accesso ai dati anagrafici nome.
  - `getCognome() : String, setCognome(String cognome) : void` - Accesso ai dati anagrafici cognome.
  - `getOpere_scritte() : int, setOpere_scritte(int opere_scritte) : void` - Accesso alle opere dell'autore.
  - `getData_nascita() : LocalDate, setData_nascita(LocalDate data_nascita) : void` - Accesso alla data di nascita.
  - `toString() : String` - Rappresentazione testuale dell'oggetto.
- **Relazioni:**
  - È referenziato dalla classe `Libro`.



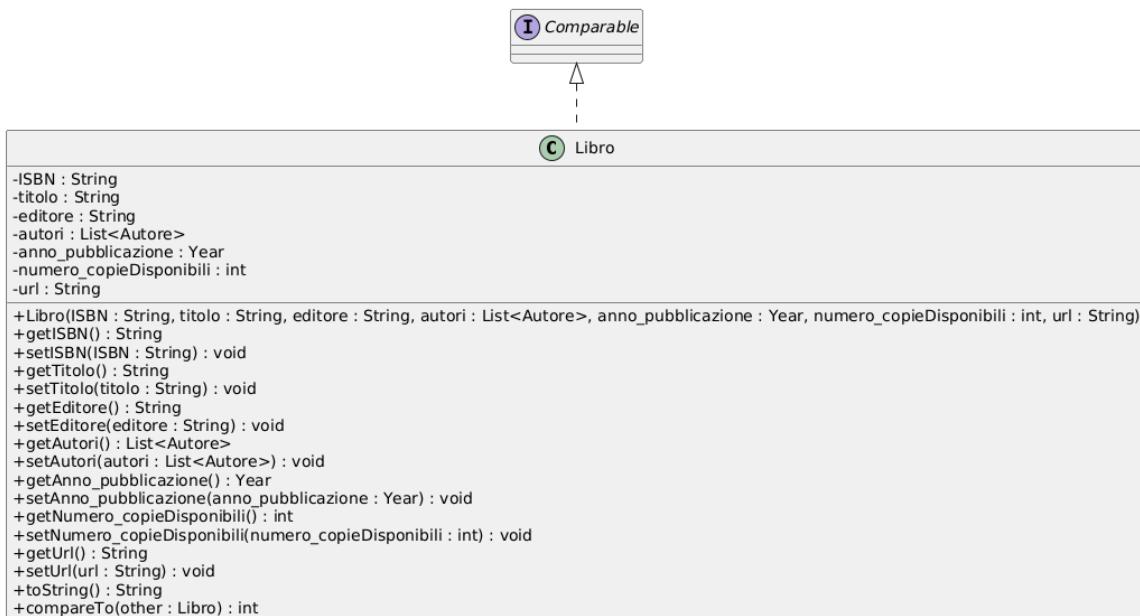
### Nome Classe: EmailInfo

- **Responsabilità:** Rappresenta le informazioni principali di una singola email, inclusi oggetto, destinatario e data di invio.
- **Attributi:**
  - `oggetto: String` - Oggetto dell'email.



- *destinatario*: String - Destinatario dell'email.
- *dataInvio*: Date - Data e ora di invio dell'email.
- **Metodi Principali:**
  - *EmailInfo(String oggetto, String destinatario, Date dataInvio)* - Costruttore che inizializza i campi della classe.
  - *getOggetto(): String* - Restituisce l'oggetto dell'email.
  - *getDestinatario(): String* - Restituisce il destinatario dell'email.
  - *getDataInvio(): Date* - Restituisce la data di invio dell'email.
  - *toString(): String* - Restituisce una rappresentazione testuale dell'email con data, destinatario e oggetto.
- **Relazioni:**
  - Utilizzata dalla classe *EmailLegge* per creare oggetti contenenti le informazioni delle email lette dal server.

## Nome Classe: Libro



- **Responsabilità:** Rappresenta un libro nel sistema.
- **Attributi:**
  - *isbn*: String - Codice identificativo univoco.
  - *titolo*: String - Titolo del libro.
  - *editore*: String - Casa editrice.
  - *autori*: List<Autore> - Lista degli autori associati.
  - *anno\_pubblicazione*: Year - Anno di pubblicazione.
  - *numero\_copieDisponibili*: int - Numero di copie fisiche disponibili.
  - *url*: String - Percorso dell'immagine di copertina.
- **Metodi Principali:**
  - *Libro(String ISBN, String titolo, String editore, List<Autore> autori, Year anno\_pubblicazione, int numero\_copieDisponibili, String url)* - Costruttore.
  - *getISBN()*: String, *setISBN(String ISBN)*: void - Gestione ISBN.
  - *getTitolo()*: String, *setTitolo(String titolo)*: void - Gestione Titolo.
  - *getEditore()*: String, *setEditore(String editore)*: void - Gestione Editore.

- `getAutori() : List<Autore>`, `setAutori(List<Autore> autori) : void` - Gestione lista autori.
- `getAnno_pubblicazione() : Year`, `setAnno_pubblicazione(Year anno_pubblicazione) : void` - Gestione Anno di Pubblicazione.
- `getNumero_copieDisponibili() : int`, `setNumero_copieDisponibili(int numero_copieDisponibili) : void` - Gestione Numero di copie disponibili.
- `getUrl() : String`, `setUrl(String url) : void` - Gestione url.
- `toString() : String` - Rappresentazione testuale dell'oggetto.
- `compareTo(Libro o) : int` - Permette di confrontare due oggetti *Libro* in base al titolo per determinare un ordine naturale.
- **Relazioni:**
  - Aggregazione con *Autore*.
  - Gestito da *Catalogo*.
  - Caricato/Salvato tramite *DataBase*.

### Nome Classe/Enum: Stato

- **Responsabilità:** Rappresentare i possibili stati di un prestito nella biblioteca.
- **Tipo:** Enumerativo (enum).
- **Valori Possibili:**
  - *ATTIVO* - Prestito attualmente in corso.
  - *RESTITUITO* - Prestito già restituito.
  - *PROROGATO* - Prestito prorogato oltre la scadenza iniziale.
  - *IN\_RITARDO* - Prestito scaduto e non ancora restituito.
- **Relazioni:**
  - Utilizzato da classi come *Prestito* e *OperazioniGiornaliere* per gestire lo stato dei prestiti.

<b>E</b>	Stato
	ATTIVO
	RESTITUITO
	PROROGATO
	IN_RITARDO

### Nome Classe: Utente

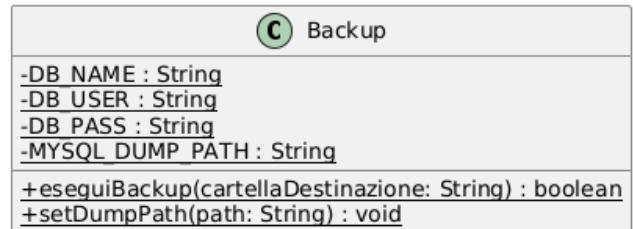
<b>U</b> Utente	
-nome : String	
-cognome : String	
-matricola : String	
-mail : String	
-bloccato : boolean	
+Utente(matricola: String, nome: String, cognome: String, mail: String, bloccato: boolean)	
+getUtentiBlackListed(utenti: ArrayList<Utente>) : ArrayList<Utente>	
+getMatricola() : String	
+setMatricola(matricola: String) : void	
+getNome() : String	
+setNome(nome: String) : void	
+getCognome() : String	
+setCognome(cognome: String) : void	
+getMail() : String	
+setMail(mail: String) : void	
+isBloccato() : boolean	
+setBloccato(bloccato: boolean) : void	
+toString() : String	

- **Responsabilità:** Rappresenta un utente con i relativi dati personali..
- **Attributi:**
  - *nome : String* - Nome dell'utente.
  - *cognome : String* - Cognome dell'utente.
  - *matricola : String* - Matricola univoca associata all'utente.
  - *mail : String* - Email dell'utente.
  - *bloccato: boolean* - Stato dell'utente (inserito o meno nella blacklist).
- **Metodi Principali:**
  - *Utente(String matricola, String nome, String cognome, String mail,boolean bloccato)* - Costruttore.
  - *getUtentiBlackListed(ArrayList<Utente> utenti) : ArrayList<Utente>* – Restituisce la lista degli utenti che hanno bloccato = true.
  - *getMatricola() : String* - Restituisce la matricola dell'utente.
  - *setMatricola(String matricola) : void* - Imposta la matricola dell'utente.
  - *isBloccato() : boolean* - Restituisce se l'utente è bloccato.
  - *setBloccato(): void* - Imposta lo stato di blocco dell'utente.
  - *getNome() : String* - Restituisce il nome dell'utente.
  - *setNome(String nome) : void* - Imposta il nome dell'utente.
  - *getCognome() : String* - Restituisce il cognome dell'utente.
  - *setCognome(String cognome) : void* - Imposta il cognome dell'utente.
  - *getMail() : String* - Restituisce l'email dell'utente.
  - *setMail(String mail) : void* - Imposta l'email dell'utente.
  - *toString() : String* - Rappresentazione testuale dell'oggetto.
- **Relazioni:**
  - È referenziato da *Prestito*.
  - È contenuto in *BlackList*.

## PACKAGE *model.servizi*

### Nome Classe: Backup

- **Responsabilità:** Esegue il backup del database della biblioteca esportando i dati in un file .sql nella cartella specificata e gestendo automaticamente il nome del file e il percorso di destinazione.
- **Attributi:**
  - *DB\_NAME : String* - Nome del database da salvare.
  - *DB\_USER : String* - Nome utente per l'accesso al database.
  - *DB\_PASS : String* - Password dell'utente del database.
  - *MYSQL\_DUMP\_PATH : String* - Percorso completo dell'eseguibile mysqldump.exe.
- **Metodi Principali:**
  - *eseguiBackup(String cartellaDestinazione) : boolean* - Costruisce il comando per eseguire il dump del database e lo lancia come processo esterno. Restituisce true se il backup è riuscito, false altrimenti.
  - *setDumpPath(String path) : void* - Permette di cambiare a runtime il valore della variabile statica *MYSQL\_DUMP\_PATH*, cioè il percorso completo del file mysqldump.exe.
- **Relazioni:**
  - Interagisce con il file system (*File*) per creare il file di destinazione.
  - Usa classi standard Java per gestione data (*Date*, *SimpleDateFormat*), input/output (*BufferedReader*, *InputStreamReader*) e processi esterni (*ProcessBuilder*).



### Nome Classe: Catalogo

- **Responsabilità:** Gestisce la collezione in memoria dei libri e fornisce metodi di ricerca in base a vari filtri.
- **Attributi:**
  - *libri : List<Libro>* - Lista interna dei libri.
- **Metodi Principali:**
  - *Catalogo()* - Costruttore.
  - *aggiungiLibro(Libro libro) : void* - Aggiunge un libro alla lista.
  - *rimuoviLibro(Libro libro) : void* - Rimuove un libro.
  - *getLibri() : ArrayList<Libro>* - Restituisce lista libri.
  - *sort() : void* - Ordina i libri in ordine alfabetico in base al titolo.
  - *cercaPerISBN(String ISBN) : Libro* - Ricerca per codice.
  - *cercaPerTitolo(String titolo) : List<Libro>* - Ricerca per titolo.
  - *cercaPerAutore(String autore) : List<Libro>* - Ricerca per autore.
- **Relazioni:**
  - Contiene una lista di oggetti *Libro*.



## Nome Classe: ControlloFormato

- **Responsabilità:** Controlla la validità dei dati inseriti.
- **Attributi:** Nessuno.
- **Metodi Principali:**
  - *controlloFormatoPassword(String password) : boolean* - Controlla il formato della password.
  - *controlloFormatoEmail(String email) : boolean* - Controlla il formato dell'email.
- **Relazioni:**
  - Usato da *AccessoController*.

<b>C</b>	ControlloFormato
--	
+controlloFormatoPassword(password: String) : boolean	
+controlloFormatoEmail(email: String) : boolean	

## Nome Classe: DataBase

<b>C</b>	DataBase
-conn : Connection	
-DB_name : String	
+initializeDB() : void	
+inserisciBibliotecario(password: String) : boolean	
+controllaEsistenzaBibliotecario() : boolean	
+controllaPasswordBibliotecario(password: String) : boolean	
+rimuoviBibliotecario() : boolean	
+getCatalogo() : Catalogo	
+getAutori() : ArrayList<Autore>	
+aggiungiLibro(l: Libro) : boolean	
+aggiungiAutore(a: Autore) : boolean	
+cercaLibro(ISBN: String) : Libro	
+getNum_Autori() : int	
+cercaAutoreByNames(nome: String, cognome: String) : Autore	
+modificaLibro(isbn: String, titolo: String, editore: String, anno_pubblicazione: int, num_copie: int, url: String, autori: ArrayList<Autore>) : boolean	
+getUtenti() : ArrayList<Utente>	
+aggiungiUtente(u: Utente) : boolean	
+getNumUser() : int	
+cercaUtente(matricola: String) : Utente	
+setBlackListed(matricola: String) : boolean	
+unsetBlackListed(matricola: String) : boolean	
+modificaUtente(matricola: String, nome: String, cognome: String, mail: String) : boolean	
+isMatricolaPresent(matricola: String) : boolean	
+rimuoviUtente(matricola: String) : boolean	
+getPrestiti() : ArrayList<Prestito>	
+aggiungiPrestito(p: Prestito) : boolean	
+getNumLoan() : int	
+restituisci(ISBN: String, matricola: String) : boolean	
+controllaPrestito(ISBN: String, matricola: String) : boolean	
+rimuoviPrestito(ISBN: String, matricola: String) : boolean	
+getNumCopieByISBN(ISBN: String) : int	
+modificaNum_copie(ISBN: String, add: boolean) : boolean	
+setStatoPrestito(isbn: String, matricola: String, stato: Stato) : boolean	
+proroga_prestito(isbn: String, matricola: String) : boolean	
+getLibriByTitolo(titolo: String) : ArrayList<Libro>	
+rimuoviLibro(isbn: String) : boolean	
+getNumRelationsScritto_Da() : int	
+isISBNPresent(ISBN: String) : boolean	

- **Responsabilità:** Gestisce la connessione JDBC al database MySQL e l'esecuzione delle query CRUD.
- **Attributi:**
  - *conn : Connection* - Connessione attiva al DB.
  - *DB\_name : String* - Nome del database ("Biblioteca").
- **Metodi Principali:**
  - *initializeDB() : void* - Stabilisce la connessione.
  - *inserisciBibliotecario(String password) : boolean, controllaEsistenzaBibliotecario() : boolean, controllaPasswordBibliotecario(String password) : boolean* - Gestione autenticazione bibliotecario.

- *rimuoviBibliotecario() : boolean* - Elimina l'account del bibliotecario.
- *getCatalogo() : Catalogo* - Recupera l'intero catalogo e ricostruisce gli oggetti *Libro* e *Autore* dalle tabelle relazionali.
- *getAutori () : ArrayList<Autore>* - Restituisce la lista di autori.
- *aggiungiLibro(Libro l) : boolean, aggiungiAutore(Autore a) : boolean* - Persistenza di nuovi dati.
- *cercaLibro(String ISBN) : Libro* - Cerca un libro nel catalogo.
- *getNum\_Autori() : int, cercaAutoreByNames(String nome, String cognome) : Autore* - Gestiscono la ricerca/numero di autori.
- *modificaLibro(String isbn, String titolo, String editore,int anno\_pubblicazione,int num\_copie, String url,ArrayList<Autore> autori) : boolean* - Aggiorna i dati principali nella tabella Libri.
- *getLibriByTitolo(String titolo) : ArrayList<Libro>* - Restituisce una lista di libri il cui titolo contiene la stringa passata come parametro.
- *rimuoviLibro(String isbn) : boolean* - Rimuove libro partendo dall'isbn.
- *getUtenti() : ArrayList<Utente>* - Restituisce gli utenti.
- *aggiungiUtente(Utente u) : boolean* - Aggiunge un utente nel database.
- *getNumUser() : int* - Restituisce numero di utenti.
- *cercaUtente(String matricola) : Utente* - Cerca un utente conoscendo la matricola.
- *setBlackListed(String matricola) : boolean* - Inserisce un utente nella black-list.
- *unsetBlackListed(String matricola) : boolean* - Rimuove un utente dalla black-list.
- *modificaUtente(String matricola, String nome, String cognome, String mail) : boolean* - Aggiorna i dati di un utente.
- *isMatricolaPresent(String matricola) : boolean* - Controlla se la matricola è presente nel sistema.
- *rimuoviUtente(String matricola) : boolean* - Rimuove l'utente.
- *getPrestiti() : ArrayList<Prestito>* - Restituisce la lista di prestiti.
- *aggiungiPrestito(Prestito p) : boolean* - Aggiunge un prestito alla lista.
- *getNumLoan () : int* - Restituisce il numero del prestito.
- *restituisci (String ISBN, String matricola) : boolean* - Aggiorna lo stato del prestito (data restituzione).
- *controllaPrestito(String ISBN, String matricola) : boolean* - Verifica lo stato del prestito.
- *rimuoviPrestito(String ISBN, String matricola) : boolean* - Rimuove un prestito dal database.
- *modificaNum\_copie(String ISBN, boolean add) : boolean* - Modifica il numero di copie.
- *setStatoPrestito(String isbn, String matricola, Stato stato) : boolean* - Aggiorna il campo stato\_prestito di un prestito registrato nel database.
- *proroga\_prestito(String isbn, String matricola) : boolean* - Proroga il prestito.
- *getNumCopieByISBN(String ISBN) : int* - Restituisce il numero di copie.
- *getNumRelationsScritto\_Da() : int* - Restituisce il numero totale di relazioni libro-autore presenti nella tabella scritta\_da del database.
- *isISBNPresent(String ISBN) : boolean* - Controlla se l'ISBN è presente nel sistema.
- **Relazioni:**
  - È utilizzata dai Controller per l'accesso ai dati persistenti.

## Nome Classe: EmailInvia

C EmailInvia	
-username : String	
-password : String	
-SMTP HOST : String	
-SMTP PORT : String	
-ret : boolean	
+inviaEmail(recipient: String, subject: String, body: String) : void	
+inviaAvviso(recipientEmail: String, titolo: String, nome: String, cognome: String, inizioPrestito: LocalDate) : void	
+setTestConfiguration(host: String, port: String) : void	

- **Responsabilità:** Utility per l'invio di email tramite protocollo SMTP (es. notifiche).
- **Attributi:**
  - *username*: String - Email del bibliotecario.
  - *password*: String - Password del bibliotecario.
  - *SMTP\_HOST*: String - Host del server SMTP, modificabile per test.
  - *SMTP\_PORT*: String - Porta del server SMTP, modificabile per test.
  - *ret* : boolean - Flag che indica se l'invio dell'email è avvenuto con successo.
- **Metodi Principali:**
  - *inviaEmail(String recipient, String subject, String body) : void* - Configura la sessione e invia il messaggio.
  - *inviaAvviso(String recipientEmail, String titolo, String nome, String cognome, LocalDate inizioPrestito) : void* - Invia un avviso di mancata restituzione libro in modalità asincrona.
  - *setTestConfiguration(String host, String port) : void* - Permette di modificare host e porta SMTP per eseguire test senza usare Gmail reale.
- **Relazioni:**
  - È utilizzata dai Controller per inviare email.

## Nome Classe: EmailLegge

- **Responsabilità:** Utility per la lettura delle email inviate tramite protocollo IMAP, recuperando informazioni principali come destinatario, oggetto e data di invio.
- **Attributi:**
  - *username*: String - Email del mittente.
  - *password*: String - Password dell'account Gmail.
  - *IMAP\_HOST*: String - Memorizza l'host del server IMAP utilizzato per la ricezione delle email.
- **Metodi Principali:**
  - *leggiPostaInviata(): ArrayList<EmailInfo>* - Si connette al server IMAP, legge le email inviate negli ultimi 40 messaggi, crea oggetti *EmailInfo* con soggetto, destinatario e data, e li restituisce in una lista.
- **Relazioni:**
  - È utilizzata dai Controller per visualizzare le email inviate.
  - Dipende dalla classe *EmailInfo* per rappresentare i dati di ogni messaggio.

C EmailLegge	
-username : String	
-password : String	
-IMAP HOST : String	
+leggiPostaInviata() : ArrayList<EmailInfo>	

## Nome Classe: OperazioniGiornaliere

- **Responsabilità:** Gestisce l'esecuzione automatica di controlli giornalieri sui prestiti, aggiornando lo stato dei prestiti in ritardo e mostrando notifiche all'utente. Permette di avviare e fermare il task schedulato quotidianamente.
- **Attributi:**
  - *ultimoResetSessione : long* - Timestamp dell'ultimo reset della sessione utente (millisecondi).
  - *scheduler : ScheduledExecutorService* - Scheduler per eseguire task periodici.
  - *durataSessione : int* - Durata massima della sessione in ore, caricata da Configurazione.
- **Metodi Principali:**
  - *avviaTaskDiMezzanotte(): void* - Avvia il task che si esegue automaticamente ogni giorno a mezzanotte, calcolando il ritardo iniziale e il periodo di 24 ore.
  - *calcolaRitardoVersoMezzanotte(): long* - Calcola quanti secondi mancano alla prossima mezzanotte.
  - *eseguiControlliAutomatici(boolean SkipNotify): void* - Esegue i controlli sui prestiti, aggiornando lo stato dei prestiti in ritardo e mostrando la notifica se necessario.
  - *stop(): void* - Ferma lo scheduler e interrompe l'esecuzione automatica dei task.
  - *calcolaSecondiAllaProssimaOra() : long* - Restituisce i secondi mancanti alla prossima ora.
- **Relazioni:**
  - Utilizza la classe *Prestito* e *DataBase* per recuperare e aggiornare i dati dei prestiti.
  - Usa *Stato* (enum) per modificare lo stato dei prestiti.
  - Interagisce con JavaFX (*Platform, Stage, Scene, FXMLLoader*) per mostrare notifiche.

 OperazioniGiornaliere
<i>-ultimoResetSessione : long</i>
<i>-scheduler : ScheduledExecutorService</i>
<i>-durataSessione : int</i>
<i>+avviaTaskDiMezzanotte() : void</i>
<i>+calcolaRitardoVersoMezzanotte() : long</i>
<i>+eseguiControlliAutomatici(SkipNotify: boolean) : void</i>
<i>+stop() : void</i>
<i>+calcolaSecondiAllaProssimaOra() : long</i>

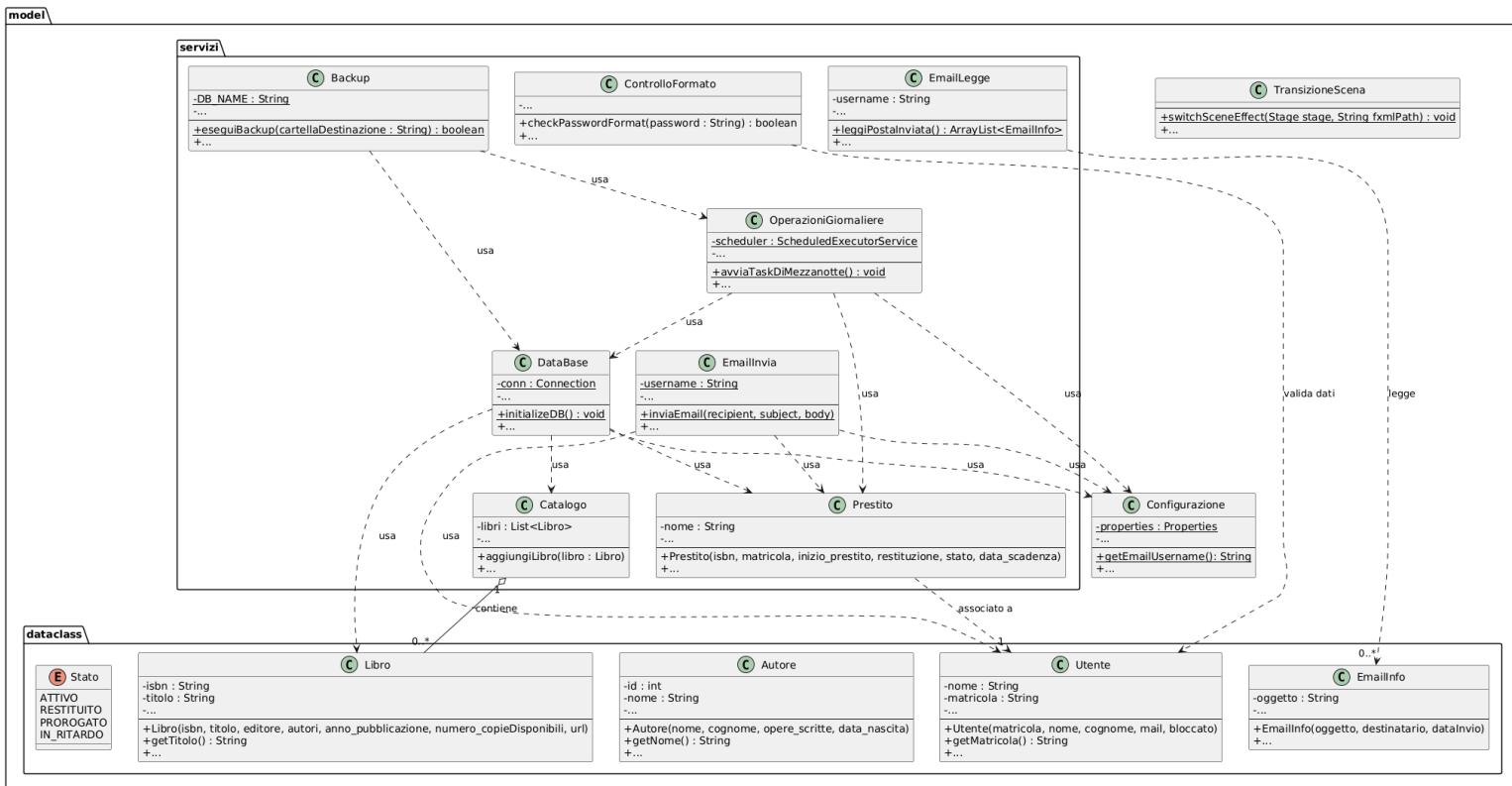
## Nome Classe: Prestito

 Prestito
<i>-matricola : String</i>
<i>-isbn : String</i>
<i>-inizio_prestito : LocalDate</i>
<i>-restituzione : LocalDate</i>
<i>-data_scadenza : LocalDate</i>
<i>-stato : Stato</i>
<i>+Prestito(isbn : String, matricola : String, inizio_prestito : LocalDate, restituzione : LocalDate, stato : Stato, data_scadenza : LocalDate)</i>
<i>+getPrestitiByStato(p : ArrayList&lt;Prestito&gt;, s : Stato) : ArrayList&lt;Prestito&gt;</i>
<i>+getMatricola() : String</i>
<i>+setMatricola(matricola : String) : void</i>
<i>+getIsbn() : String</i>
<i>+setIsbn(isbn : String) : void</i>
<i>+getDataScadenza() : LocalDate</i>
<i>+setDataScadenza(data_scadenza : LocalDate) : void</i>
<i>+getInizio_prestito() : LocalDate</i>
<i>+setInizio_prestito(inizio_prestito : LocalDate) : void</i>
<i>+getRestituzione() : LocalDate</i>
<i>+setRestituzione(restituzione : LocalDate) : void</i>
<i>+getStato() : Stato</i>
<i>+setStato(stato : Stato) : void</i>
<i>+toString() : String</i>

- **Responsabilità:** Rappresenta il prestito di un libro a un utente.
- **Attributi:**
  - *matricola : String* - Matricola univoca associata all'utente.

- *isbn*: String - Codice identificativo libro.
  - *inizio\_prestito* : LocalDate - Data inizio prestito.
  - *restituzione*: LocalDate - Data in cui è stata effettuata la restituzione.
  - *data\_scadenza*: LocalDate - Data di fine prestito.
  - *stato*: Stato - Tipo enumerativo che rappresenta lo stato corrente di un prestito (valori possibili: ATTIVO, RESTITUITO, PROROGATO, IN\_RITARDO).
- **Metodi Principali:**
  - *Prestito(String ISBN, String matricola, LocalDate inizio\_prestito, LocalDate restituzione, Stato stato, LocalDate data\_scadenza)* - Costruttore.
  - *getPrestitiByStato(ArrayList<Prestito> p, Stato s)* : ArrayList<Prestito> - Accesso al prestito.
  - *getMatricola() : String*, *setMatricola(String matricola) : void* - Accesso ai dati dell'utente.
  - *getISBN() : String*, *setISBN(String ISBN) : void* - Gestione dati libro.
  - *getDataScadenza() : LocalDate*, *setDataScadenza(LocalDate data\_scadenza) : void*,  
*getInizio\_prestito() : LocalDate*, *setInizio\_prestito(LocalDate inizio\_prestito) : void*,  
*getRiRestituzione() : LocalDate*, *setRiRestituzione(LocalDate restituzione) : void* -  
 Getter/Setter per le date.
  - *getStato() : Stato*, *setStato(Stato stato) : void* - Getter/Setter stato del prestito.
  - *toString() : String* - Rappresentazione testuale dell'oggetto.
- **Relazioni:**
  - Associazione Libro <-> Utente.
  - Utilizzata dai moduli Gestione Prestiti/Notifiche.

- **Diagramma delle classi nel package model:**



## PACKAGE controller

### Nome Classe: AccessoController

- **Responsabilità:** Gestisce l'autenticazione del bibliotecario, la registrazione e le animazioni della schermata di login.
- **Attributi:**
  - *LoginPane : Pane* - Pannello del form di login.
  - *RegisterForm : Pane* - Pannello del form di registrazione.
  - *Sliding : Pane* - Barra laterale che scorre per passare da login a registrazione.
  - *SlidingButton : Button* - Bottone che attiva lo sliding.
  - *SwitchLabel : Label* - Testo che cambia in base alla schermata attiva.
  - *direction : boolean* - Definisce il verso dello sliding (valore: true).
  - *ts : TranslateTransition* - Transizione della barra Sliding.
  - *tr : TranslateTransition* - Transizione del pannello RegisterForm.
  - *slidingTiming : double* - Durata dello sliding.
  - *LoginButton : Button* - Bottone di login.
  - *RegisterButton : Button* - Bottone per registrarsi.
  - *PassRegister : PasswordField / PassRegisterVisible : TextField* - Password nascosta e visibile.
  - *PassConRegister : PasswordField / PassConRegisterVisible : TextField* - Conferma password nascosta/visibile.
  - *CheckShowPassRegister : CheckBox* - Mostra/nasconde password.
  - *PassLogin : PasswordField/ PassLoginVisible : TextField* - Password nascosta/visibile.
  - *CheckShowPassLogin : CheckBox* - Mostra/nasconde password.
- **Metodi Principali:**
  - *initialize() : void* - Avvia la configurazione iniziale di buttoni e checkbox.
  - *buttonInitialize() : void* - Collega i pulsanti alle loro funzioni, gestisce sliding, registrazione e login.
  - *checkboxInitialize() : void* - Gestisce il passaggio tra password visibile/nascosta.
  - *setSliding() : void* - Prepara l'animazione di sliding per passare tra login e registrazione.
  - *showPassword(boolean yes, boolean login) : void* - Alterna tra *PasswordField* e *TextField* per mostrare/nascondere password.
  - *cleanForm(boolean login) : void* - Pulisce campi e checkbox quando si passa tra schermate.
  - *typewriterEffectLabel(Label label, String text) : void* private - Effetto scrittura per la label.
  - *typewriterEffectButton(Button b, String text) : void* private - Effetto scrittura per il pulsante sliding.

 AccessoController
-LoginPane : Pane
-RegisterForm : Pane
-Sliding : Pane
-SlidingButton : Button
-SwitchLabel : Label
-direction : boolean
-ts : TranslateTransition
-tr : TranslateTransition
-slidingTiming : double
-LoginButton : Button
-RegisterButton : Button
-PassRegister : PasswordField
-PassRegisterVisible : TextField
-PassConRegister : PasswordField
-PassConRegisterVisible : TextField
-CheckShowPassRegister : CheckBox
-PassLogin : PasswordField
-PassLoginVisible : TextField
-CheckShowPassLogin : CheckBox
+initialize() : void
+buttonInitialize() : void
+checkboxInitialize() : void
+setSliding() : void
+showPassword(yes: boolean, login: boolean) : void
+cleanForm(login: boolean) : void
-typewriterEffectLabel(label: Label, text: String) : void
-typewriterEffectButton(b: Button, text: String) : void

- **Relazioni:**
  - Dipende da Model ( usa *ControlloFormato*, *DataBase* ed *EmailInvia*) e da View.

## Nome Classe: BlacklistController

C BlacklistController	
-blacklistContainer : VBox	
-lblTotalBlocked : Label	
-UnLockAllButton : Button	
-searchUser : TextField	
+initialize() : void	
+searchFunction() : void	
+updateUtentiList(utenti: ArrayList<Utente>) : void	
-aggiungiCardUtente(nome: String, cognome: String, matricola: String, email: String, isBlacklisted: boolean) : void	

- **Responsabilità:** Gestisce la visualizzazione e la gestione degli utenti bloccati (blacklist), permette di sbloccare tutti o singoli utenti, cercare utenti nella lista e inviare avvisi via email.
- **Attributi principali:**
  - *blacklistContainer* : *VBox* - Contenitore grafico che ospita le righe degli utenti nella blacklist.
  - *lblTotalBlocked* : *Label* - Label che mostra il numero totale di utenti bloccati.
  - *UnLockAllButton* : *Button* - Bottone per sbloccare tutti gli utenti presenti nella blacklist.
  - *searchUser* : *TextField* - Campo di testo per cercare un utente specifico per nome, cognome, email o matricola.
- **Metodi Principali:**
  - *initialize()* : *void* - Inizializza la vista caricando gli utenti bloccati, configura i listener del bottone sblocca tutto e del campo di ricerca.
  - *searchFunction()* : *void* - Cerca un utente nella lista blacklist in base al testo inserito nel campo di ricerca e aggiorna la lista visualizzata.
  - *updateUtentiList(utenti : ArrayList<Utente>)* : *void* - Aggiorna il contenitore grafico con le righe degli utenti forniti, aggiornando il contatore totale.
  - *aggiungiCardUtente(nome : String, cognome : String, matricola : String, email : String, isBlacklisted : boolean)* : *void* - Crea e aggiunge una riga grafica per un singolo utente, con icona, stato, bottone email e bottone blocca/sblocca.
- **Relazioni:**
  - Usa *Utente* per ottenere la lista degli utenti e filtrare quelli bloccati.
  - Usa *DataBase* per sbloccare utenti e ottenere la lista completa degli utenti.
  - Usa *EmailInvia* per inviare avvisi agli utenti.
  - Interagisce con JavaFX (*VBox*, *HBox*, *Label*, *Button*, *TextField*, *Tooltip*, *DialogPane*) per la gestione dell'interfaccia grafica.

## **Nome Classe: DashboardController**

- **Responsabilità:** Controller principale che gestisce il menu di navigazione laterale e il caricamento dinamico delle viste nel pannello centrale.
- **Attributi:**
  - *CatalogoLibriButton : Button* - Bottone per accedere al catalogo dei libri.
  - *mailButton : Button* - Bottone per accedere alla sezione mail.
  - *BLButton : Button* - Bottone per la sezione blacklist utenti.
  - *DashboardButton : Button* - Bottone per tornare alla home/dashboard.
  - *utentiButton : Button* - Bottone per la sezione utenti.
  - *PrestitiRestituzioniButton : Button* - Bottone per la sezione prestiti e restituzioni.
  - *numLibri : Label* - Mostra il numero totale di libri presenti.
  - *numLoanAttivi : Label* - Mostra il numero di prestiti attivi.
  - *numUsers : Label* - Mostra il numero di utenti registrati.
  - *numScaduti : Label* - Mostra il numero di prestiti in ritardo o scaduti.
  - *DashboardBox : VBox* - Contenitore verticale per elementi della GUI.
  - *DashboardScrollPane : ScrollPane* - Contenitore con supporto scroll verticale e orizzontale.
  - *HomeBorderPane : BorderPane* - Contenitore centrale dinamico per caricare diverse viste.
  - *menuButtons : List<Button>* - Lista dei bottoni principali del menu per la gestione degli stili.
  - *LogoutButton : Button* - Bottone per uscire dalla dashboard e tornare al login.
  - *modPassButton : Button* - Bottone per modificare la password dell'utente.
  - *BackupButton : Button* - Bottone per eseguire il backup dei dati della biblioteca.
- **Metodi Principali:**
  - *initialize() : void* - Configura il menu.
  - *labelInitialize() : void* - Aggiorna i valori delle label (libri, prestiti attivi, utenti, prestiti scaduti) leggendo i dati dal database.
  - *buttonInitialize() : void* - Configura tutti i bottoni della dashboard, associando le azioni (navigazione, backup, logout, modifica password).
  - *evidenziaBottone(Button bottoneAttivo) : void* - Evidenzia il bottone attivo nel menu laterale rimuovendo l'evidenziazione dagli altri.
- **Relazioni:**
  - Carica le viste gestite da *CatalogoController*, *MailController*, ecc.

<b>C</b>	DashboardController
-CatalogoLibriButton : Button	
-mailButton : Button	
-BLButton : Button	
-DashboardButton : Button	
-utentiButton : Button	
-PrestitiRestituzioniButton : Button	
-numLibri : Label	
-numLoanAttivi : Label	
-numUsers : Label	
-numScaduti : Label	
-DashboardBox : VBox	
-DashboardScrollPane : ScrollPane	
-HomeBorderPane : BorderPane	
-menuButtons : List<Button>	
-LogoutButton : Button	
-modPassButton : Button	
-BackupButton : Button	
+initialize() : void	
+labelInitialize() : void	
+buttonInitialize() : void	
+evidenziaBottone(bottoneAttivo: Button) : void	

## Nome Classe: MailController

- **Responsabilità:** Gestisce la visualizzazione della lista utenti a cui inviare email e la generazione dinamica delle righe nella UI.
- **Attributi principali:**
  - *emailContainer : VBox* - Contenitore grafico dell'e-mail.
  - *lblTotalUsers : Label* - Label per mostrare il numero di email o messaggi di stato.
- **Metodi Principali:**
  - *initialize()* : void - Inizializza la vista e avvia il caricamento delle email.
  - *caricaEmailInviate()* : void - Scarica le email in un thread separato e aggiorna la GUI.
  - *aggiungiCardEmail(mail : EmailInfo) : void* - Crea e aggiunge una card visiva per una singola email.
- **Relazioni:**
  - *MailController* usa *EmailLegge* per leggere le email.
  - *MailController* dipende da *EmailInfo* per rappresentare i dati di ogni email.
  - Interagisce con JavaFX (*VBox*, *HBox*, *Label*, *ProgressIndicator*, ecc.) per gestire la visualizzazione.



## Nome Classe: NotificaController

- **Responsabilità:** Permette all'utente di visualizzare il numero di prestiti in ritardo e suggerisce eventuali azioni, come l'invio di notifiche.
- **Attributi principali:**
  - *numRit : Label* - Label per mostrare il numero di prestiti in ritardo e un messaggio informativo.
- **Metodi Principali:**
  - *initialize()* : void - Recupera la lista dei prestiti dal database, conta quelli in ritardo e aggiorna la label numRit con un messaggio informativo.
- **Relazioni:**
  - Usa *DataBase* per ottenere la lista dei prestiti.
  - Dipende da *Prestito* e dall'enum *Stato* per identificare i prestiti in ritardo.
  - Interagisce con JavaFX (*Label*) per la visualizzazione del messaggio nella UI.



## PACKAGE controller.catalogo

### Nome Classe: AggiungiLibroController

- **Responsabilità:** Gestisce il form di inserimento di un nuovo libro.
- **Attributi:**
  - *txtTitolo : TextField* , *txtISBN : TextField*, *spinAnno : Spinner<Integer>*, *spinCopie : Spinner<Integer>* - Campi di input.
  - *menuAutori : MenuButton* - Menu a discesa con CheckBox per selezione autori multipli.
  - *imgAnteprima : ImageView* - Visualizza la copertina selezionata.
  - *txtEditore : TextField* - Campo editore.
  - *ScegliFileButton : Button* - Pulsante per scegliere un file.
  - *AnnullaButton : Button* - Pulsante per annullare l'operazione.
  - *SalvaButton : Button* - Pulsante per salvare.
  - *RimuoviCopButton : Button* - Pulsante per rimuovere copia.
  - *MAX\_AUTORS : int* - Numero massimo di autori consentiti nel sistema (1000).
  - *MAX\_WRITED : int* - Numero massimo di relazioni “scritto da” libro-autore (5000).
  - *urlIM : String* - Percorso dell’immagine della copertina del libro.
- **Metodi Principali:**
  - *initialize() : void* - Configura il form.
  - *settingForm() : void* - Setta il form.
  - *updateAutori() : void* - Popola il menu autori recuperando i dati dal DB.
  - *buttonInitialize() : void* - Gestisce il salvataggio (SalvaButton) e la scelta file (ScegliFileButton).
  - *spinnerInitialize() : void* - Inizializza il campo Spinner.
- **Relazioni:**
  - Crea oggetti *Libro* e *Autore* e li salva tramite *DataBase*.

<b>C</b>	AggiungiLibroController
-txtTitolo : TextField	
-txtISBN : TextField	
-spinAnno : Spinner<Integer>	
-spinCopie : Spinner<Integer>	
-menuAutori : MenuButton	
-imgAnteprima : ImageView	
-txtEditore : TextField	
-ScegliFileButton : Button	
-AnnullaButton : Button	
-SalvaButton : Button	
-RimuoviCopButton : Button	
-MAX_AUTORS : int	
-MAX_WRITED : int	
-urlIM : String	
+initialize() : void	
+settingForm() : void	
+updateAutori() : void	
+buttonInitialize() : void	
+spinnerInitialize() : void	

### Nome Classe: CatalogoController

- **Responsabilità:** Gestisce la visualizzazione a griglia del catalogo libri e le interazioni con le card dei libri.
- **Attributi:**
  - *containerLibri: GridPane* - Controlla la disposizione dei libri.
  - *LibriScrollPane: ScrollPane* - Pannello scorrevole.
  - *btnCerca : Button* - Bottone per la ricerca.
  - *searchBar : TextField* - Campo di input per la ricerca dei libri per titolo o ISBN.

<b>C</b>	CatalogoController
-containerLibri : GridPane	
-LibriScrollPane : ScrollPane	
-btnCerca : Button	
-searchBar : TextField	
-addButton : Button	
-MAX_BOOKS : int	
+initialize() : void	
+searchFunction() : void	
+updateCatalogo() : void	
-creaLibroAnimato(libro: Libro) : VBox	
+launchAggiungiLibroForm() : void	

- *addButton : Button* - Bottone per aprire la finestra di inserimento di un nuovo libro.
- *MAX\_BOOKS : int* - Costante che definisce il numero massimo di libri consentiti nel sistema (1000).
- **Metodi Principali:**
  - *initialize() : void* - Inizializza il catalogo.
  - *searchFunction() : void* - Esegue la ricerca di libri per ISBN o titolo e aggiorna la vista del catalogo.
  - *updateCatalogo() : void* - Recupera i libri dal DataBase e popola la griglia.
  - *creaLibroAnimato(Libro libro) : VBox* - Crea dinamicamente un oggetto grafico (StackPane) per ogni libro, con effetti 3D al passaggio del mouse e gestione del click.
  - *launchAggiungiLibroForm() : void* - Apre la finestra modale per aggiungere un nuovo libro; controlla il limite massimo di libri consentiti.
- **Relazioni:**
  - Usa *Catalogo* per gestire la lista dei libri.
  - Usa *Libro* per rappresentare ogni libro e accedere ai dati.
  - Usa *DataBase* per ricerca, aggiunta, modifica e rimozione dei libri.
  - Usa *ModificaLibroController* per aprire la finestra di modifica del libro.
  - Interagisce con JavaFX (*ScrollPane*, *GridPane*, *VBox*, *HBox*, *Button*, *Label*, *ImageView*, *StackPane*, *Alert*, *DialogPane*) per gestire la UI e le animazioni.

## **Nome Classe: ModificaLibroController**

- **Responsabilità:** Gestisce la modifica dei dati di un libro all'interno del catalogo, inclusi titolo, autore, editore, anno, numero di copie e immagine di copertina. Permette di aggiornare i dati nel database e gestisce l'interfaccia grafica per la selezione dei campi.
- **Attributi:**
  - *txtTitolo : TextField* - Campo di testo per il titolo del libro.
  - *menuAutori : MenuButton* - Menu per selezionare o aggiungere gli autori del libro.
  - *txtEditore : TextField* - Campo di testo per il nome dell'editore.
  - *spinAnno : Spinner<Integer>* - Spinner per selezionare l'anno di pubblicazione.
  - *spinCopie : Spinner<Integer>* - Spinner per selezionare il numero di copie disponibili.
  - *imgAnteprima : ImageView* - Visualizza l'immagine della copertina del libro.
  - *ScegliFileButton : Button* - Bottone per scegliere un file immagine come copertina.
  - *AnnullaButton : Button* - Bottone per chiudere la finestra senza salvare le modifiche.
  - *SalvaButton : Button* - Bottone per salvare le modifiche nel database.
  - *RimuoviCopButton : Button* - Bottone per ripristinare la copertina di default.
  - *urlIM : String* - Percorso dell'immagine attualmente selezionata.
  - *lib : Libro* - Oggetto libro corrente
  - *isbn : String* - Codice ISBN del libro in modifica.
- **Metodi Principali:**

 <b>ModificaLibroController</b>
- <i>txtTitolo : TextField</i>
- <i>menuAutori : MenuButton</i>
- <i>txtEditore : TextField</i>
- <i>spinAnno : Spinner&lt;Integer&gt;</i>
- <i>spinCopie : Spinner&lt;Integer&gt;</i>
- <i>imgAnteprima : ImageView</i>
- <i>ScegliFileButton : Button</i>
- <i>AnnullaButton : Button</i>
- <i>SalvaButton : Button</i>
- <i>RimuoviCopButton : Button</i>
- <i>urlIM : String</i>
- <i>lib : Libro</i>
- <i>isbn : String</i>
+ <i>initialize() : void</i>
+ <i>settingForm() : void</i>
+ <i>buttonInitialize() : void</i>
+ <i>updateAutori() : void</i>
+ <i>spinnerInitialize() : void</i>

- *initialize() : void* - Inizializza il form impostando immagine di default, autori e spinner.
- *settingForm() : void* - Configura il form con immagine predefinita, autori disponibili e spinner per anno e copie.
- *buttonInitialize() : void* - Configura i bottoni della UI (scegli file, rimuovi copertina, salva e annulla) e le rispettive azioni.
- *updateAutori() : void* - Aggiorna il menu degli autori esistenti e aggiunge campi per nuovi autori.
- *spinnerInitialize() : void* - Inizializza gli spinner per anno di pubblicazione e numero di copie.
- **Relazioni:**
  - Usa *Libro* per rappresentare e modificare i dati del libro.
  - Usa *Autore* per gestire la selezione e creazione degli autori.
  - Usa *DataBase* per salvare le modifiche e cercare autori.
  - Interagisce con JavaFX (*TextField*, *MenuButton*, *Spinner*, *ImageView*, *Button*, *CustomMenuItem*, *CheckBox*, *DialogPane*, *Alert*) per gestire l'interfaccia utente.

## PACKAGE controller.modificapassword

### Nome Classe: CambioPasswordController

- **Responsabilità:** Gestisce l'interfaccia per l'inserimento e aggiornamento della password del bibliotecario, includendo controllo di sicurezza, conferma password e visibilità del campo.
- **Attributi principali:**
  - *NewPass : PasswordField* - Campo password per l'inserimento della nuova password.
  - *ConfirmPass : PasswordField* - Campo password per confermare la nuova password.
  - *NewPassVisible : TextField* - Campo testo visibile della nuova password (per toggle visibilità).
  - *ConfirmPassVisible : TextField* - Campo testo visibile della conferma password (per toggle visibilità).
  - *CheckShowPass : CheckBox* - Checkbox per abilitare/disabilitare la visualizzazione delle password.
  - *BtnSalva : Button* - Bottone per salvare la nuova password.
  - *BtnAnnulla : Button* - Bottone per annullare l'operazione.
- **Metodi Principali:**
  - *initialize() : void* - Inizializza la view, impostando checkbox e bottoni.
  - *setButtonFunction() : void* - Configura la logica di salvataggio e annullamento della password, inclusi controlli e alert.
  - *setCheckBox() : void* - Configura il comportamento della checkbox per mostrare/nascondere le password.
  - *showPassword(yes : boolean) : void* - Mostra o nasconde i campi password in base al valore booleano passato.

CambioPasswordController
- <i>NewPass : PasswordField</i> - <i>ConfirmPass : PasswordField</i> - <i>NewPassVisible : TextField</i> - <i>ConfirmPassVisible : TextField</i> - <i>CheckShowPass : CheckBox</i> - <i>BtnSalva : Button</i> - <i>BtnAnnulla : Button</i>
+ <i>initialize() : void</i> + <i>setButtonFunction() : void</i> + <i>setCheckBox() : void</i> + <i>showPassword(yes : boolean) : void</i>

- **Relazioni:**
  - Usa *DataBase* per aggiornare la password del bibliotecario.
  - Usa *ControlloFormato* per validare la sicurezza della password.
  - Interagisce con JavaFX (*PasswordField*, *TextField*, *Button*, *CheckBox*, *Label*, *Alert*, *DialogPane*) per gestire la UI.

### Nome Classe: InserisciPasswordModificaController

- **Responsabilità:** Gestisce la modifica della password del bibliotecario, mostrando l'interfaccia per l'inserimento della nuova password e validando quella corrente.
- **Attributi principali:**
  - *NewPass* : *PasswordField* - Campo per inserire la nuova password (nascosta).
  - *NewPassVisible* : *TextField* - Campo per visualizzare la password se l'utente seleziona "Mostra Password".
  - *CheckShowPass* : *CheckBox* - Checkbox per alternare tra visualizzazione e nascondimento della password.
  - *BtnSalva* : *Button* - Bottone per confermare e salvare la modifica della password.
  - *BtnAnnulla* : *Label* - Label utilizzata come pulsante per annullare l'operazione.
- **Metodi Principali:**
  - *initialize()* : *void* - Inizializza il controller, impostando i listener per il checkbox e i bottoni.
  - *setButtonFunction()* : *void* - Configura le azioni dei bottoni Salva e Annulla, controlla la password inserita e apre la finestra per la modifica se valida.
  - *setCheckBox()* : *void* - Configura il comportamento del checkbox per mostrare o nascondere la password.
  - *showPassword(yes : boolean)* : *void* - Alterna tra i campi *PasswordField* e *TextField* per mostrare o nascondere la password inserita.
- **Relazioni:**
  - Usa *DataBase* per verificare la password corrente del bibliotecario.
  - Usa *ControlloFormato* per eventuali controlli sul formato della password (se necessario).
  - Interagisce con JavaFX (*PasswordField*, *TextField*, *CheckBox*, *Button*, *Label*, *Stage*, *Scene*) per gestire l'interfaccia grafica.

<b>C</b>	InserisciPasswordModificaController
	-NewPass : <i>PasswordField</i>
	-NewPassVisible : <i>TextField</i>
	-CheckShowPass : <i>CheckBox</i>
	-BtnSalva : <i>Button</i>
	-BtnAnnulla : <i>Label</i>
	+initialize() : <i>void</i>
	+setButtonFunction() : <i>void</i>
	+setCheckBox() : <i>void</i>
	+showPassword(yes: <i>boolean</i> ) : <i>void</i>

### Nome Classe: InserimentoCodiceRecuperoController

- **Responsabilità:** Gestisce l'inserimento e la validazione del codice di recupero (OTP) composto da 6 cifre, fornendo un'interfaccia che guida l'utente nella digitazione sequenziale del codice e abilita la verifica solo quando tutte le cifre sono state inserite correttamente.

- **Attributi principali:**

- *digit1: TextField*- Campo per l'inserimento della prima cifra del codice di recupero.
- *digit2: TextField* - Campo per l'inserimento della seconda cifra del codice di recupero.
- *digit3: TextField* - Campo per l'inserimento della terza cifra del codice di recupero.
- *digit4: TextField* - Campo per l'inserimento della quarta cifra del codice di recupero.
- *digit5: TextField* - Campo per l'inserimento della quinta cifra del codice di recupero.
- *digit6: TextField* - Campo per l'inserimento della sesta cifra del codice di recupero.
- *VerifyButton : Button* - Bottone per confermare e verificare il codice di recupero inserito.
- *ResendLabel : Label* - Label utilizzata per consentire il reinvio del codice di recupero.

<b>C</b>	InserimentoCodiceRecuperoController
-	digit1 : TextField
-	digit2 : TextField
-	digit3 : TextField
-	digit4 : TextField
-	digit5 : TextField
-	digit6 : TextField
+initialize() : void	
+setButtonFunction() : void	
+setEffect() : void	
+getFullCode() : String	

- **Metodi Principali:**

- *initialize() : void* - Inizializza il controller impostando gli effetti sui campi di input e la logica associata ai bottoni.
- *setButtonFunction() : void* - Configura il comportamento del bottone di verifica del codice (attualmente predisposto per la gestione della conferma).
- *setEffect() : void* - Imposta la logica di input sui campi del codice: consente solo numeri, limita a una cifra per campo, gestisce l'auto-avanzamento del cursore e il ritorno al campo precedente tramite backspace o frecce direzionali.
- *getFullCode() : String* - Restituisce il codice di recupero completo come stringa concatenando le sei cifre inserite.

- **Relazioni:**

- Usa un servizio di verifica del codice di recupero (es. DataBase o servizio di backend) per controllare la validità del codice OTP inserito dall'utente.
- Usa la logica di controllo dell'input per garantire che ogni campo contenga una sola cifra numerica.
- Interagisce con JavaFX (TextField, Button, Label, KeyEvent) per gestire l'interfaccia grafica e l'interazione dell'utente con i campi di inserimento del codice.

## Nome Classe: RecuperaPasswordController

- **Responsabilità:** Gestisce la procedura di recupero della password del bibliotecario, consentendo l'inserimento dell'email, la generazione del codice di recupero e l'avvio del flusso di recupero della password.

<b>C</b>	RecuperaPasswordController
-	mailField : TextField
-	RecoveryButton : Button
+code : String	
+Email : String	
+initialize() : void	
+setButtonFunction() : void	
+codeGeneration() : String	

- **Attributi principali:**

- *mailField : TextField*- Campo per l'inserimento dell'email del bibliotecario.
- *RecoveryButton : Button* - Bottone per avviare la procedura di recupero della password.
- *code: String* - Codice di recupero generato per la verifica dell'identità dell'utente.

- *Email : String* - Email del bibliotecario utilizzata per la procedura di recupero.
- **Metodi Principali:**
  - *initialize() : void* - Inizializza il controller impostando il comportamento del bottone di recupero.
  - *setButtonFunction() : void* - Configura l'azione del bottone di recupero, avviando la generazione del codice e il flusso di verifica dell'email.
  - *codeGeneration() : String* - Genera un codice numerico casuale di 6 cifre da utilizzare per il recupero della password.
- **Relazioni:**
  - Usa DataBase per verificare l'esistenza dell'email del bibliotecario nel sistema.
  - Usa un servizio di invio email per spedire il codice di recupero all'indirizzo inserito.
  - Usa ControlloFormato per verificare la correttezza del formato dell'email inserita.
  - Interagisce con JavaFX (TextField, Button, Alert, Stage, Scene) per gestire l'interfaccia grafica e le notifiche all'utente.

## PACKAGE controller.prestitorestituzione

### Nome Classe: AggiungiPrestitoController

- **Responsabilità:** Gestisce l'inserimento di un nuovo prestito, con controlli su ISBN, matricola, date e disponibilità delle copie.
- **Attributi:**
  - *txtISBN : TextField* - Inserimento ISBN.
  - *txtMatricola : TextField* - Inserimento matricola studente.
  - *IsbnCheck : Label* - Mostra esito verifica ISBN.
  - *matricolaCheck : Label* - Mostra esito verifica matricola.
  - *IsbnCheckButton : Button* - Avvia controllo ISBN.
  - *MatricolaCheckButton : Button* - Avvia controllo matricola.
  - *AnnullaButton : Button* - Chiude la finestra di inserimento.
  - *SalvaButton : Button* - Conferma e salva il prestito.
  - *dateInizio : DatePicker* - Data di inizio prestito.
  - *dateScadenza : DatePicker* - Data di scadenza prestito.
  - *CompletedCheckISBN : boolean* - Esito controllo ISBN (*false*).
  - *CompletedCheckMatricola : boolean* - Esito controllo matricola (*false*).
- **Metodi Principali:**
  - *initialize() : void* - Inizializzazione componenti.
  - *initializeProperty() : void* - Inizializza property.
  - *buttonInitialize() : void* - Gestione pulsanti Salva e Annulla.
  - *buttonCheckingInitialize() : void* - Controllo ISBN e matricola.
- **Relazioni:**

 AggiungiPrestitoController
- <i>txtISBN : TextField</i>
- <i>txtMatricola : TextField</i>
- <i>IsbnCheck : Label</i>
- <i>matricolaCheck : Label</i>
- <i>IsbnCheckButton : Button</i>
- <i>MatricolaCheckButton : Button</i>
- <i>AnnullaButton : Button</i>
- <i>SalvaButton : Button</i>
- <i>dateInizio : DatePicker</i>
- <i>dateScadenza : DatePicker</i>
- <i>CompletedCheckISBN : boolean</i>
- <i>CompletedCheckMatricola : boolean</i>
+ <i>initialize() : void</i>
+ <i>initializeProperty() : void</i>
+ <i>buttonInitialize() : void</i>
+ <i>buttonCheckingInitialize() : void</i>

- Usa *DataBase*, crea *Prestito*, utilizza *Utente* e *Stato*, interagisce con JavaFX.

## Nome Classe: PrestitoRestituzioneController

C PrestitoRestituzioneController	
-loansContainer : VBox	
-NewLoanButton : Button	
-FilterButton : MenuButton	
-lblActiveLoans : Label	
-searchLoan : TextField	
-MAX_LOAN : int	
+initialize() : void	
+menuButtonInitialize() : void	
+searchFunction() : void	
+buttonInitialize() : void	
+updatePrestiti(p1: ArrayList<Prestito>) : void	
+aggiungiRigaPrestito(titoloLibro: String, ISBN: String, nomeUtente: String, dataScadenza: String, statoEnum: Stato) : void	

- **Responsabilità:**

Gestisce la visualizzazione dei prestiti, il filtraggio per stato (attivi, in ritardo, restituiti) e le operazioni di restituzione e creazione di un nuovo prestito.

- **Attributi:**

- *loansContainer* : *VBox* - Contenitore grafico dei prestiti.
- *NewLoanButton* : *Button* - Bottone per aggiungere un nuovo prestito.
- *FilterButton* : *MenuButton* - Menu per filtrare i prestiti.
- *lblActiveLoans* : *Label* - Visualizza il numero dei prestiti attivi.
- *searchLoan* : *TextField* - Campo di testo della GUI dove l'utente può inserire un termine di ricerca relativo ai prestiti
- *MAX\_LOAN* : *int* - Costante che rappresenta il numero massimo di prestiti consentiti contemporaneamente nel sistema.

- **Metodi Principali:**

- *initialize()* : *void* - Inizializza la vista, aggiorna i prestiti e i filtri.
- *menuButtonInitialize()* : *void* - Inizializza il menu di filtraggio per stato.
- *searchFunction()* : *void* - Ricerca prestiti nel database in base al testo inserito dall'utente nella GUI
- *buttonInitialize()* : *void* - Gestisce l'apertura della finestra per un nuovo prestito.
- *updatePrestiti(ArrayList<Prestito> p1)* : *void* - Aggiorna la lista dei prestiti visualizzati.
- *aggiungiRigaPrestito(String titoloLibro, String ISBN, String nomeUtente, String dataScadenza, Stato statoEnum)* : *void* - Crea graficamente una riga di prestito con stato e azioni.

- **Relazioni:**

- Usa *DataBase* per la gestione dei prestiti e delle restituzioni.
- Usa *Prestito*, *Libro*, *Utente* e l'enumerazione *Stato*.
- Apre la vista gestita da *AggiungiPrestitoController*.
- Interagisce con componenti JavaFX.

## PACKAGE controller.utenti

### Nome Classe: AggiungiUtenteController

- **Responsabilità:** Gestisce la creazione di un nuovo utente. Esegue i controlli su matricola, nome, cognome e email, e inserisce l'utente nel database.
- **Attributi:**
  - *txtMatricola : TextField* - Matricola utente.
  - *txtNome : TextField* - Nome utente.
  - *txtCognome : TextField* - Cognome utente.
  - *txtEmail : TextField* - Email utente.
  - *AnnullaButton : Button* - Chiude la finestra.
  - *SalvaButton : Button* - Avvia validazione e salvataggio.
- **Metodi Principali:**
  - *initialize() : void* - Inizializza buttoni e listener richiamando *buttonInitialize()*.
  - *buttonInitialize() : void* - Chiude la finestra ed esegue i controlli.
- **Relazioni:**
  - Dipende da Model e da View.

<b>C</b> AggiungiUtenteController
-txtMatricola : TextField
-txtNome : TextField
-txtCognome : TextField
-txtEmail : TextField
-AnnullaButton : Button
-SalvaButton : Button
+initialize() : void
+buttonInitialize() : void

### Nome Classe: ModificaUtenteController

- **Responsabilità:** Gestisce la modifica dei dati di un utente (nome, cognome, email) con controlli su campi vuoti e validità dell'email. Mostra alert personalizzati e aggiorna i dati nel database.
- **Attributi:**
  - *lblMatricola : Label* - Visualizza la matricola dell'utente.
  - *txtNome : TextField* - Inserimento/modifica del nome.
  - *txtCognome : TextField* - Inserimento/modifica del cognome.
  - *txtEmail : TextField* - Inserimento/modifica dell'email.
  - *btnSalva : Button* - Conferma e salva le modifiche.
  - *btnAnnulla : Button* - Annulla e chiude la finestra.
  - *matricola : String* - Matricola dell'utente da modificare.
- **Metodi Principali:**
  - *initialize() : void* - Inizializza componenti, imposta matricola, gestisce eventi pulsanti e controlli sui campi (nome, cognome, email).
- **Relazioni:**
  - Usa *ModificaUtenteController* e *ControlloFormato*.
  - Interagisce con JavaFX (*Label*, *TextField*, *Button*, *Alert*, *Stage*).

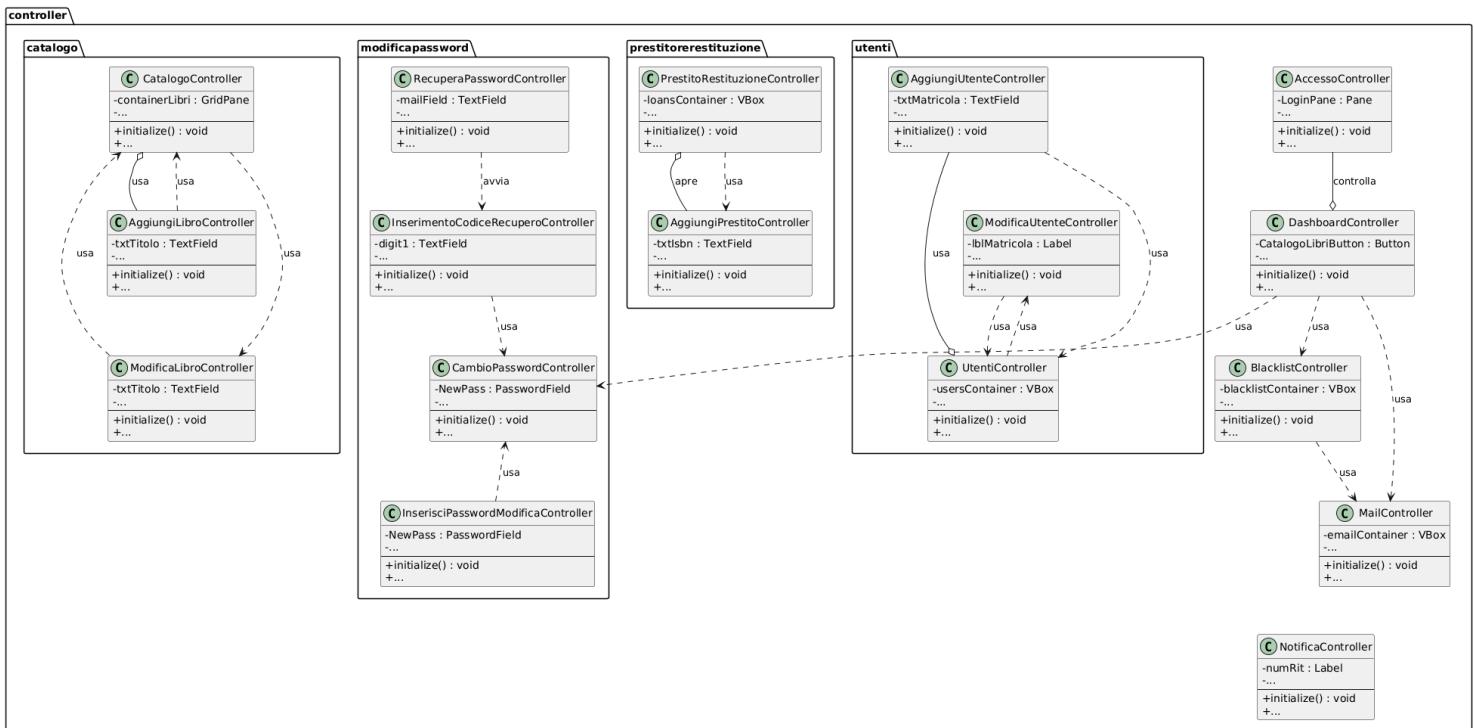
<b>C</b> ModificaUtenteController
-lblMatricola : Label
-txtNome : TextField
-txtCognome : TextField
-txtEmail : TextField
-btnSalva : Button
-btnAnnulla : Button
+matricola : String «static»
+initialize()

## **Nome Classe: UtentiController**

UtentiController	
-usersContainer : VBox	
-btnAddUser : Button	
-lblTotalUsers : Label	
-FilterButton : MenuButton	
-searchUser : TextField	
-MAX_USERS : int	
+initialize() : void	
+searchFunction() : void	
+menuButtonInitialize() : void	
+labelInitialize() : void	
+buttonInitialize() : void	
+updateUtentiList(utenti: ArrayList<Utente>) : void	
+aggiungiCardUtente(nome: String, cognome: String, matricola: String, email: String, isBlacklisted: boolean) : void	

- **Responsabilità:** Gestisce la visualizzazione e la gestione degli utenti, inclusi filtri (tutti, attivi, bloccati), aggiunta di nuovi utenti, modifica, blocco/sblocca e aggiornamento della lista utente.
- **Attributi:**
  - *usersContainer* : *VBox* - Contenitore verticale per le righe utente.
  - *btnAddUser* : *Button* - Pulsante per aprire la finestra di aggiunta utente.
  - *lblTotalUsers* *Label* - Visualizza il numero totale di utenti iscritti.
  - *FilterButton* : *MenuButton* - Menu a tendina per filtrare gli utenti (tutti, attivi, bloccati).
  - *searchUser*: *TextField* - Campo di ricerca per nome, cognome, email o matricola dell'utente.
  - *MAX\_USERS*: *int* - Costante che indica il numero massimo di utenti consentiti nel sistema (lettura da Configurazione).
- **Metodi Principali:**
  - *initialize()* : *void* - Inizializza componenti, aggiorna lista utenti e imposta pulsanti e menu.
  - *searchFunction()* : *void* - Ricerca utenti in base al testo inserito (matricola, nome, cognome, email) e aggiorna la visualizzazione.
  - *menuButtonInitialize()* : *void* - Configura le voci del menu filtro e gestisce la selezione.
  - *labelInitialize()* : *void* - Aggiorna l'etichetta del numero totale utenti.
  - *buttonInitialize()* : *void* - Configura il pulsante di aggiunta utente e l'apertura della finestra modale.
  - *updateUtentiList(ArrayList<Utenti> utenti)* : *void* - Aggiorna la lista degli utenti nel contenitore verticale.
  - *aggiungiCardUtente(String nome, String cognome, String matricola, String email, boolean isBlacklisted)* : *void* - Crea e aggiunge una riga utente con dati, icona, stato e pulsanti di azione (modifica, blocca/sblocca).
- **Relazioni:**
  - Usa *DataBase* per gestione utenti e stato blacklist e *Utente* per dati utente.
  - Interagisce con JavaFX (*VBox*, *HBox*, *Button*, *Label*, *MenuButton*, *Stage*, *Scene*, *FXMLLoader*, *Modality*).

- **Diagramma delle classi nel package controller:**



## PACKAGE **view**

**(Classi associate ai file FXML)**

Le **interfacce** dell'applicazione sono realizzate tramite **file FXML** e costituiscono lo strato **View** dell'architettura MVC.

Le interfacce dell'applicazione non contengono logica applicativa, ma definiscono esclusivamente:

- La **struttura grafica** delle schermate.
- I **componenti di input/output**.
- Le **interazioni** dell'utente.

Ogni View è associata a un **Controller**, al quale delega la logica di gestione degli eventi.

### **Nome Interfaccia: Accesso.fxml**

- **Responsabilità:** Gestisce l'interfaccia di accesso e registrazione della password, permettendo al bibliotecario di effettuare il login o creare una nuova password.
- **Componenti grafici principali:**
  - *Pane* - Pannello per il login.
  - *Pane* - Pannello per la registrazione.
  - *Pane* - Pannello di transizione tra login e registrazione.
  - *PasswordField* - Campo password login.
  - *TextField* - Visualizzazione password login.
  - *CheckBox* - Mostra/nasconde la password di login.
  - *Button* - Bottone di accesso.
  - *PasswordField* - Campo nuova password.
  - *TextField* - Visualizzazione nuova password.
  - *PasswordField* - Conferma nuova password.
  - *TextField* - Visualizzazione conferma password.
  - *CheckBox* - Mostra/nasconde la password di registrazione.
  - *Button* - Bottone di salvataggio nuova password.
  - *Button* - Bottone per passare tra login e registrazione.
  - *Label* - Testo informativo di cambio modalità.
- **Interazioni supportate:**
  - Inserimento della password per il login.
  - Inserimento e conferma della password per la registrazione.
  - Visualizzazione della password.
  - Passaggio animato tra login e registrazione.
- **Relazioni:**
  - Collegata al controller *AccessoController*.
  - Utilizza componenti JavaFX (*Pane*, *Button*, *TextField*, *PasswordField*, *CheckBox*, *Label*).
  - Usa il foglio di stile *StyleAccess.css*.

### **Nome Interfaccia: Catalogo.fxml**

- **Responsabilità:** Visualizza il catalogo dei libri con ricerca e disposizione a griglia, permettendo all'utente di cercare libri e navigare tra le schede dei titoli.
- **Componenti grafici principali:**
  - *TextField* - Campo per cercare libri per titolo, autore o ISBN.
  - *Button* - Bottone per avviare la ricerca.
  - *GridPane* - Contenitore dei libri disposti a griglia.
  - *ScrollPane* - Pannello scorrevole per contenere la griglia di libri.
- **Interazioni supportate:**
  - Ricerca per titolo, autore o ISBN.
  - Visualizzazione schede libro.
  - Scorrimento catalogo.
- **Relazioni:**
  - Collegata al controller *CatalogoController*.
  - Utilizza componenti JavaFX (*VBox*, *HBox*, *GridPane*, *ScrollPane*, *TextField*, *Button*, *Label*, *Pane*).
  - Usa il foglio di stile *StyleDashboard.css*.

### **Nome Interfaccia: PrestitoRestituzione.fxml**

- **Responsabilità:** Visualizza e gestisce i prestiti e le restituzioni dei libri, permettendo all'utente di filtrare per stato, cercare prestiti e registrare nuovi prestiti.
- **Componenti grafici principali:**
  - *TextField* - Campo per cercare prestiti, utenti o libri.
  - *Button* - Bottone per aggiungere un nuovo prestito.
  - *MenuButton* - Menu per filtrare i prestiti per stato (tutti, in corso, in ritardo, restituiti).
  - *Label* - Visualizza il numero dei prestiti attivi.
  - *VBox* - Contenitore verticale dei prestiti visualizzati.
  - *ScrollPane* - Pannello scorrevole per navigare tra i prestiti.
- **Interazioni supportate:**
  - Ricerca dinamica dei prestiti tramite *searchLoan*.
  - Filtraggio dei prestiti tramite *FilterButton*.
  - Visualizzazione dei prestiti nel contenitore *loansContainer*.
  - Apertura di un nuovo prestito tramite *NewLoanButton*.
  - Scorrimento verticale per navigare tra i prestiti.
- **Relazioni:**
  - Collegata al controller *PrestitoRestituzioneController*.
  - Usa componenti JavaFX (*BorderPane*, *VBox*, *HBox*, *ScrollPane*, *TextField*, *Button*, *Label*, *MenuButton*, *MenuItem*, *Pane*).
  - Usa il foglio di stile *StylePrestitoRestituzione.css*.

## **Nome Interfaccia: AggiungiPrestito.fxml**

- **Responsabilità:** Permette di registrare un nuovo prestito, inserendo i dati del libro e dell'utente, con controllo di validità e gestione delle date di inizio e scadenza.
- **Componenti grafici principali:**
  - *TextField* - Campo per inserire o scansionare il codice ISBN del libro.
  - *Button* - Bottone per verificare l'esistenza dell'ISBN.
  - *Label* - Etichetta per visualizzare lo stato della verifica ISBN.
  - *TextField* - Campo per inserire la matricola dell'utente.
  - *Button* - Bottone per verificare l'esistenza della matricola.
  - *Label* - Etichetta per visualizzare lo stato della verifica matricola.
  - *DatePicker* - Data di inizio prestito.
  - *DatePicker* - Data di scadenza prestito.
  - *Button* - Bottone per annullare l'operazione.
  - *Button* - Bottone per confermare il prestito.
- **Interazioni supportate:**
  - Inserimento e verifica dell'ISBN del libro.
  - Inserimento e verifica della matricola dell'utente.
  - Scelta della data di inizio e della scadenza del prestito.
  - Conferma o annullamento del prestito tramite i rispettivi buttoni.
- **Relazioni:**
  - Collegata al controller *AggiungiPrestitoController*.
  - Utilizza componenti JavaFX (*StackPane*, *VBox*, *HBox*, *Pane*, *Label*, *Button*, *TextField*, *DatePicker*, *Separator*, *ImageView*).
  - Usa il foglio di stile *StyleAddPrestito.css*.

## **Nome Interfaccia: AggiungiUtente.fxml**

- **Responsabilità:** Permette di registrare un nuovo utente/studente, inserendo matricola, nome, cognome e email istituzionale, con possibilità di confermare o annullare l'operazione.
- **Componenti grafici principali:**
  - *TextField* - Campo per inserire la matricola dell'utente.
  - *TextField* - Campo per inserire il nome dello studente.
  - *TextField* - Campo per inserire il cognome dello studente.
  - *TextField* - Campo per inserire l'email istituzionale.
  - *Button* - Bottone per annullare la registrazione.
  - *Button* - Bottone per salvare e aggiungere l'utente.
- **Interazioni supportate:**
  - Inserimento e validazione dei dati dell'utente.
  - Conferma o annullamento della registrazione tramite i rispettivi buttoni.
- **Relazioni:**
  - Collegata al controller *AggiungiUtenteController*.
  - Utilizza componenti JavaFX (*StackPane*, *VBox*, *HBox*, *Pane*, *Label*, *Button*, *TextField*, *ImageView*).
  - Usa il foglio di stile *StyleAddUser.css*.

### **Nome Interfaccia: AggiungiLibro.fxml**

- **Responsabilità:** Permette di aggiungere un nuovo libro al catalogo, inserendo titolo, autore, editore, ISBN, anno di pubblicazione, numero di copie e allegando la copertina.
- **Componenti grafici principali:**
  - *TextField* - Campo per inserire il titolo del libro.
  - *MenuButton* - Menu per selezionare o aggiungere autori.
  - *TextField* - Campo per inserire la casa editrice.
  - *TextField* - Campo per inserire il codice ISBN.
  - *Spinner* - Spinner per l'anno di pubblicazione.
  - *Spinner* - Spinner per il numero di copie disponibili.
  - *ImageView* - Anteprima della copertina.
  - *Button* - Bottone per caricare un'immagine di copertina.
  - *Button* - Bottone per rimuovere l'immagine allegata.
  - *Button* - Bottone per annullare l'operazione.
  - *Button* - Bottone per salvare e aggiungere il libro.
- **Interazioni supportate:**
  - Inserimento e validazione dei dati del libro.
  - Gestione dell'allegato dell'immagine di copertina.
  - Conferma o annullamento dell'aggiunta tramite i rispettivi buttoni.
- **Relazioni:**
  - Collegata al controller *AggiungiLibroController*.
  - Utilizza componenti JavaFX (*StackPane*, *VBox*, *HBox*, *Pane*, *Label*, *TextField*, *Button*, *Spinner*, *ImageView*, *MenuButton*, *MenuItem*).
  - Usa il foglio di stile *StyleAddBook.css*.

### **Nome Interfaccia: Dashboard.fxml**

- **Responsabilità:** Visualizza la dashboard dell'applicazione, mostrando statistiche principali (libri in catalogo, prestiti attivi, utenti iscritti, scadenze in ritardo) e una tabella con le scadenze imminenti. Fornisce accesso rapido alle altre sezioni tramite la sidebar.
- **Componenti grafici principali:**
  - *ScrollPane* - Contiene l'intero contenuto scorrevole della dashboard.
  - *VBox* - Contenitore principale dei widget e delle tabelle.
  - *TableView* - Tabella che mostra le scadenze in arrivo.
  - *Button* - Bottoni della sidebar per navigare.
  - *TextField* - Campo per la ricerca rapida dei libri.
  - *KPI Card Labels* (es. *libri in catalogo*, *prestiti attivi*, *utenti iscritti*, *scadenze/ritardi*) - Label per visualizzare i valori statistici.
- **Interazioni supportate:**
  - Navigazione tra le varie sezioni tramite sidebar.
  - Aggiornamento dinamico delle statistiche e della tabella delle scadenze.
  - Ricerca rapida dei libri tramite la search bar.
- **Relazioni:**
  - Collegata al controller *DashboardController*.

- Utilizza componenti JavaFX (*BorderPane*, *VBox*, *HBox*, *ScrollPane*, *GridPane*, *TableView*, *Label*, *Button*, *Pane*, *TableColumn*).
- Usa il foglio di stile *StyleDashboard.css*.

### **Nome Interfaccia: Mail.fxml**

- **Responsabilità:** Visualizza l'area di gestione delle comunicazioni e avvisi agli studenti. Consente di inviare email o promemoria e di filtrare e ordinare gli utenti in base a cognome, matricola o numero di prestiti attivi.
- **Componenti grafici principali:**
  - *TextField* - Campo per cercare uno studente o una matricola.
  - *Label* - Mostra il numero di utenti filtrati.
  - *VBox* - Contenitore che visualizza le email o le notifiche agli utenti.
  - *MenuButton* - Permette di ordinare l'elenco utenti per cognome, matricola o prestiti attivi.
  - *Button* - Pulsante per aprire il form di invio di una nuova comunicazione.
- **Interazioni supportate:**
  - Ricerca dinamica degli utenti tramite *cercaUtente()*.
  - Ordinamento dell'elenco utenti tramite il *MenuButton*.
  - Invio di nuove email o promemoria tramite il pulsante “NUOVA MAIL”.
- **Relazioni:**
  - Collegata al controller *MailController*.
  - Usa componenti JavaFX (*BorderPane*, *VBox*, *HBox*, *ScrollPane*, *Label*, *TextField*, *Button*, *MenuButton*, *MenuItem*, *Pane*).
  - Utilizza il foglio di stile *DashboardStyle.css*.

### **Nome Interfaccia: ModificaUtente.fxml**

- **Responsabilità:** Permette di modificare le informazioni di un utente già registrato nella biblioteca, come nome, cognome ed email istituzionale. Mostra anche la matricola dell'utente come riferimento non modificabile.
- **Componenti grafici principali:**
  - *Label* - Visualizza la matricola dell'utente.
  - *TextField* - Campo per modificare il nome dell'utente.
  - *TextField* - Campo per modificare il cognome dell'utente.
  - *TextField* - Campo per modificare l'email istituzionale dell'utente.
  - *Button* - Pulsante per annullare le modifiche.
  - *Button* - Pulsante per salvare le modifiche.
- **Interazioni supportate:**
  - Inserimento/modifica dei dati utente nei campi di testo.
  - Salvataggio delle modifiche tramite *btnSalva*.
  - Chiusura o annullamento della modifica tramite *btnAnnulla*.
- **Relazioni:**
  - Collegata al controller *ModificaUtenteController*.
  - Usa layout JavaFX (*StackPane*, *VBox*, *HBox*, *Pane*).

- Utilizza il foglio di stile *StyleUtenti.css*.

### **Nome Interfaccia: Utenti.fxml**

- **Responsabilità:** Definisce l'interfaccia grafica per la gestione utenti, inclusa la visualizzazione della lista utenti, la ricerca, l'aggiunta di nuovi utenti e il filtro tra tutti, attivi e bloccati.
- **Componenti grafici principali:**
  - *BorderPane* - Layout principale della scena.
  - *VBox* - Contiene titolo pagina, sottotitolo, barra di ricerca e pulsante per aggiungere nuovo utente.
  - *Label* - Titolo pagina "Gestione Utenti".
  - *Label* - Sottotitolo con descrizione funzionalità.
  - *TextField* - Barra di ricerca per nome, matricola o email.
  - *Button* - Pulsante "+ NUOVO UTENTE" per aprire la finestra di aggiunta utente.
  - *VBox* - Contenitore verticale per le card degli utenti.
  - *HBox* - Contiene etichetta "Filtra vista:", *MenuButton* per selezione filtro e *Label* con numero totale utenti.
  - *MenuButton* - Menu a tendina con voci "Tutti gli Utenti", "Solo Attivi", "Solo Blacklist".
  - *Label* - Mostra il numero totale di utenti iscritti.
  - *ScrollPane* - Consente lo scorrimento verticale delle card utenti nel VBox centrale.
- **Interazioni supportate:**
  - Ricerca utenti.
  - Filtro per stato.
  - Apertura form nuovo utente.
  - Scorrimento elenco.
- **Relazioni:**
  - Collegata al controller *UtentiController*.
  - Interagisce con JavaFX (*BorderPane*, *VBox*, *HBox*, *Button*, *Label*, *TextField*, *MenuButton*, *ScrollPane*) e con i CSS personalizzati "/CSS/*StyleUtenti.css*".

### **Nome Interfaccia: Notifica.fxml**

- **Responsabilità:** Visualizza all'utente il numero di prestiti in ritardo e suggerisce eventuali azioni, come l'invio di notifiche.
- **Componenti grafici principali:**
  - *Pane\_Principale* - Contenitore principale della card di notifica.
  - *Label\_Titolo* - Mostra il titolo della card: "AZIONE SUGGERITA".
  - *Label\_NumRit* - Mostra il numero di prestiti in ritardo con messaggio informativo.
- **Interazioni supportate:**
  - Visualizzazione dinamica del numero di prestiti in ritardo.

- Aggiornamento automatico del messaggio in base ai dati del database tramite il controller.
- **Relazioni:**
  - Collegata al controller *NotificaController*.
  - Usa *DataBase* per recuperare la lista dei prestiti.
  - Dipende da *Prestito* e da *Stato* per identificare i prestiti in ritardo.
  - Interagisce con componenti JavaFX (*Pane*, *Label*) per la visualizzazione.

### **Nome Interfaccia: BlackList.fxml**

- **Responsabilità:** Gestisce la visualizzazione e la gestione degli utenti bloccati (blacklist), consente di sbloccare tutti o singoli utenti, cercare utenti nella lista e inviare avvisi via email.
- **Componenti grafici principali:**
  - *BorderPane\_Principale* - Layout principale della scena.
  - *VBox\_Top* - Contiene titolo, sottotitolo, barra di ricerca e pulsante per sbloccare tutti gli utenti.
  - *Label\_Titolo* - "Blacklist Utenti".
  - *Label\_Sottotitolo* - "Gestione restrizioni e riammissione utenti."
  - *TextField\_SearchUser* - Barra di ricerca per filtrare utenti bloccati.
  - *Button\_UnLockAll* - Pulsante per sbloccare tutti gli utenti.
  - *VBox\_Center* - Contiene la lista utenti e informazioni di stato.
  - *Label\_TotalBlocked* - Mostra il numero totale di utenti bloccati.
  - *ScrollPane* - Permette lo scorrimento verticale delle card utenti.
  - *VBox\_BlacklistContainer* - Contenitore verticale che ospita le righe degli utenti bloccati.
- **Interazioni supportate:**
  - Ricerca dinamica degli utenti nella blacklist tramite *TextField\_SearchUser*.
  - Sblocco singolo o globale degli utenti tramite *Button\_UnLockAll* e interazioni sulle card.
  - Visualizzazione aggiornata del numero totale utenti bloccati tramite *Label\_TotalBlocked*.
  - Scorrimento verticale della lista utenti tramite *ScrollPane*.
- **Relazioni:**
  - Collegata al controller *BlacklistController*.
  - Usa *Utente* per ottenere informazioni sugli utenti.
  - Usa *DataBase* per sbloccare utenti e recuperare la lista completa.
  - Usa *EmailInvia* per inviare eventuali avvisi via email.
  - Interagisce con componenti JavaFX (*BorderPane*, *VBox*, *HBox*, *ScrollPane*, *Label*, *Button*, *TextField*, *Pane*) e con i CSS personalizzati */CSS/StyleBlacklist.css*.

### **Nome Interfaccia: InserisciPasswordModifica.fxml**

- **Responsabilità:** Gestisce l'inserimento della nuova password del bibliotecario e la validazione della password corrente, permettendo di mostrare/nascondere la password e confermare o annullare l'operazione.
- **Componenti grafici principali:**
  - *Pane\_Principale* - Contenitore principale della scena.
  - *Pane\_Interno* - Contiene titolo, sottotitolo, campi password e buttoni.
  - *Label\_Titolo* - "Nuova Password".
  - *Label\_Sottotitolo* - "Scegli una password sicura".
  - *Label\_AttualePassword* - Etichetta "PASSWORD ATTUALE".
  - *PasswordField\_NewPass* - Campo password nascosto per inserimento della nuova password.
  - *TextField\_NewPassVisible* - Campo testo visibile per la nuova password (toggle visibilità).
  - *CheckBox\_ShowPass* - Checkbox per mostrare/nascondere la password.
  - *Button\_BtnSalva* - Pulsante per confermare e salvare la nuova password.
  - *Label\_BtnAnnulla* - Etichetta utilizzata come pulsante per annullare l'operazione.
- **Interazioni supportate:**
  - Mostra/nasconde la password tramite *CheckBox\_ShowPass*.
  - Conferma modifica password tramite *Button\_BtnSalva*.
  - Annulla operazione tramite *Label\_BtnAnnulla*.
  - Visualizzazione dinamica tra *PasswordField* e *TextField* per la password.
- **Relazioni:**
  - Collegata al controller *InserisciPasswordModificaController*.
  - Usa *DataBase* per verificare la password corrente.
  - Usa *ControlloFormato* per eventuali controlli di sicurezza sulla password.
  - Interagisce con componenti JavaFX (*Pane*, *Label*, *TextField*, *PasswordField*, *Button*, *CheckBox*) e con CSS personalizzati *CSS/StyleAccess.css*.

### **Nome Interfaccia: CambioPassword.fxml**

- **Responsabilità:** Gestisce l'inserimento e la conferma della nuova password del bibliotecario, permettendo di mostrare/nascondere le password e salvare o annullare l'operazione.
- **Componenti grafici principali:**
  - *Pane\_Principale* - Contenitore principale della scena.
  - *Pane\_Interno* - Contiene titolo, sottotitolo, campi password e buttoni.
  - *Label\_Titolo* - "Nuova Password".
  - *Label\_Sottotitolo* - "Scegli una password sicura".
  - *Label\_NuovaPassword* - Etichetta "NUOVA PASSWORD".
  - *PasswordField\_NewPass* - Campo password nascosto per la nuova password.
  - *TextField\_NewPassVisible* - Campo testo visibile per la nuova password (toggle visibilità).
  - *Label\_ConfermaPassword* - Etichetta "CONFERMA PASSWORD".

- *PasswordField\_ConfirmPass* - Campo password nascosto per confermare la password.
- *TextField\_ConfirmPassVisible* - Campo testo visibile per confermare la password (toggle visibilità).
- *CheckBox\_ShowPass* - Checkbox per mostrare/nascondere entrambe le password.
- *Button\_BtnSalva* - Pulsante per salvare la nuova password.
- *Label\_BtnAnnulla* - Etichetta utilizzata come pulsante per annullare l'operazione.
- **Interazioni supportate:**
  - Mostra/nasconde le password tramite *CheckBox\_ShowPass*.
  - Conferma la nuova password tramite *Button\_BtnSalva*.
  - Annulla operazione tramite *Label\_BtnAnnulla*.
  - Alternanza tra *PasswordField* e *TextField* per i campi password.
- **Relazioni:**
  - Collegata al controller *CambioPasswordController*.
  - Usa *DataBase* per aggiornare la password del bibliotecario.
  - Usa *ControlloFormato* per validare la sicurezza della password.
  - Interagisce con componenti JavaFX (*Pane*, *Label*, *TextField*, *PasswordField*, *Button*, *CheckBox*) e con CSS personalizzati *CSS/StyleAccess.css*.

### **Nome Interfaccia: ModificaLibro.fxml**

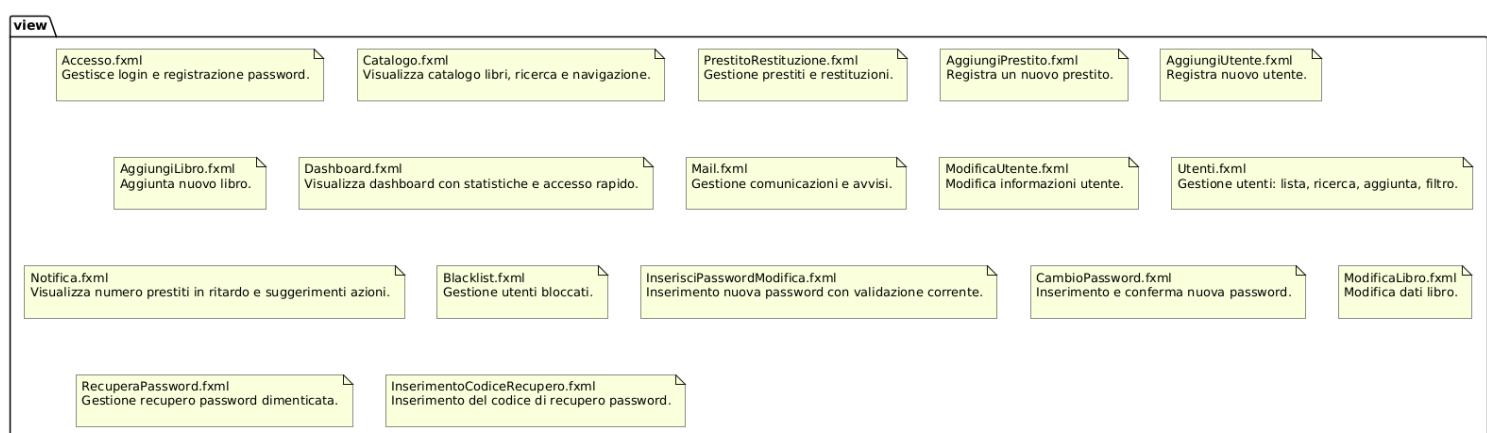
- **Responsabilità:** Gestisce la modifica dei dati di un libro all'interno del catalogo, inclusi titolo, autori, casa editrice, anno di pubblicazione, numero di copie e immagine di copertina. Permette di salvare o annullare le modifiche.
- **Componenti grafici principali:**
  - *StackPane\_Principale* - Contenitore principale della scena.
  - *ImageView\_Sfondo* - Immagine di sfondo della finestra.
  - *HBox\_Layout* - Contiene due colonne principali: dati libro e dettagli aggiuntivi.
  - *Colonna Sinistra (VBox)*:
    - *Label\_TitoloSezione* - "Registro Nuovi Arrivi".
    - *Label\_Separatore* - Linea decorativa.
    - *VBox\_TitoloLibro* - Etichetta + *TextField\_txtTitolo*.
    - *VBox\_Autori* - Etichetta + *MenuBar\_menuAutori* con MenuItem multipli.
    - *VBox\_Editore* - Etichetta + *TextField\_txtEditore*.
  - *Colonna Destra (VBox)*:
    - *HBox\_AnnoCopie* - Contiene:
      - *Spinner\_spinAnno* - Selezione anno pubblicazione.
      - *Spinner\_spinCopie* - Selezione numero copie.
    - *VBox\_AnteprimaCopertina* - Etichetta + *StackPane\_polaroid-frame* con *ImageView\_imgAnteprima*.
    - *Pane\_BottoniFile* - Pulsanti *ScegliFileButton*, *RimuoviCopButton*.
    - *Pane\_BottoniSalvaAnnulla* - Pulsanti *AnnullaButton*, *SalvaButton*.
- **Interazioni supportate:**
  - Modifica titolo, autori, editore, anno e numero copie.
  - Scegliere una nuova immagine di copertina o rimuoverla.

- Salvataggio dei dati modificati tramite *SalvaButton*.
  - Annullamento della modifica tramite *AnnullaButton*.
- **Relazioni:**
  - Collegata al controller *ModificaLibroController*.
  - Usa *Libro* per rappresentare i dati del libro.
  - Usa *Autore* per la gestione degli autori.
  - Usa *DataBase* per salvare le modifiche.
  - Interagisce con JavaFX (*StackPane*, *HBox*, *VBox*, *TextField*, *Spinner*, *MenuButton*, *MenuItem*, *Button*, *ImageView*) e CSS personalizzati */CSS/StyleAddBook.css*.
  
  
  
  
  
  
  
  
  
- Nome Interfaccia: InserimentoCodiceRecupero.fxml**
- **Responsabilità:** Gestisce l'inserimento del codice di sicurezza a 6 cifre inviato all'email dell'utente, permettendo di verificare l'identità e richiedere l'invio del codice nuovamente se necessario.
- **Componenti grafici principali:**
  - *Pane\_Principale* - Contenitore principale della scena.
  - *Pane\_Centrale* - Contiene titolo, sottotitolo, campi per inserire il codice, etichette di supporto e pulsante di verifica.
  - *Label\_Titolo* - "Verifica Identità".
  - *Label\_Sottotitolo* - "Abbiamo inviato un codice a 6 cifre alla tua email."
  - *HBox\_Codice* - Contiene i sei campi di testo per l'inserimento del codice.
  - *TextField\_Digit1* - Primo campo del codice.
  - *TextField\_Digit2* - Secondo campo del codice.
  - *TextField\_Digit3* - Terzo campo del codice.
  - *TextField\_Digit4* - Quarto campo del codice.
  - *TextField\_Digit5* - Quinto campo del codice.
  - *TextField\_Digit6* - Sesto campo del codice.
  - *Label\_RichiestaNuovoCodice* - "Non hai ricevuto il codice?".
  - *Label\_ResendLabel* - Link cliccabile "Invia di nuovo" per richiedere un nuovo codice.
  - *Button\_VerifyButton* - Pulsante per confermare il codice inserito.
- **Interazioni supportate:**
  - Inserimento del codice a 6 cifre tramite i campi *TextField\_Digit1–Digit6*.
  - Verifica del codice tramite *Button\_VerifyButton*.
  - Richiesta invio nuovo codice tramite *Label\_ResendLabel*.
- **Relazioni:**
  - Collegata al controller *InserimentoCodiceRecuperoController*.
  - Utilizza il database o il servizio di invio email per validare il codice.
  - Interagisce con componenti JavaFX (*Pane*, *Label*, *TextField*, *HBox*, *Button*) e con CSS personalizzati *CSS/InsertCodePassRecovery.css*.

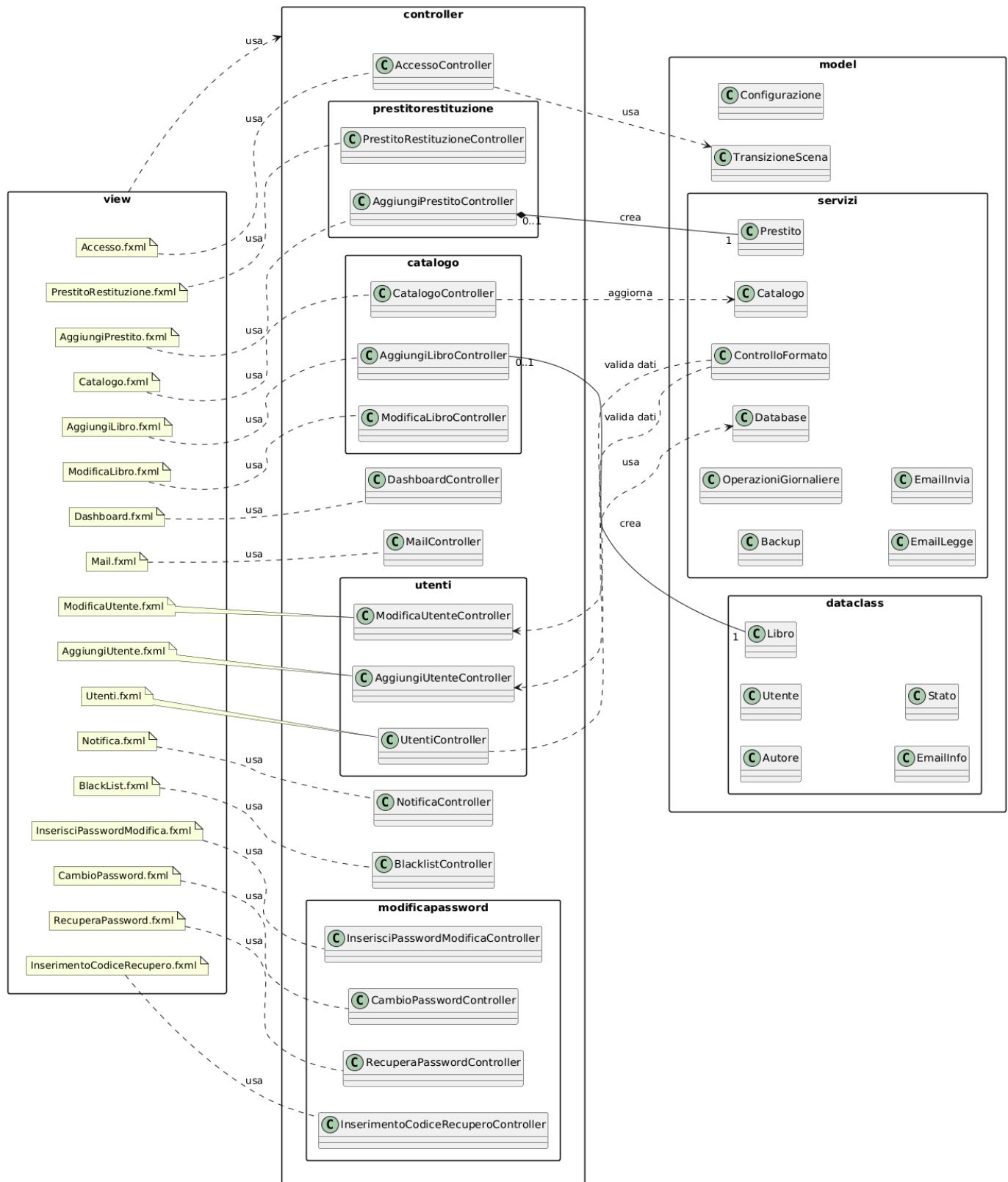
## **Nome Interfaccia: RecuperaPassword.fxml**

- **Responsabilità:** Gestisce l'inserimento dell'indirizzo email dell'utente per l'invio di un codice di sicurezza, permettendo di ricevere il codice per il recupero della password.
- **Componenti grafici principali:**
  - *Pane\_Principale* - Contenitore principale della scena.
  - *Pane\_Centrale* - Contiene titolo, sottotitolo, campo email, bottone e testo descrittivo.
  - *Label\_Titolo* - "Recupero Password".
  - *Label\_Sottotitolo* - "INDIRIZZO EMAIL".
  - *TextField\_MailField* - Campo di testo per inserire l'indirizzo email.
  - *Button\_RecoveryButton* - Pulsante "INVIAMI IL CODICE" per richiedere l'invio del codice di sicurezza.
  - *Label\_Descrizione* - Testo descrittivo in basso.
- **Interazioni supportate:**
  - Inserimento dell'indirizzo email tramite *TextField\_MailField*.
  - Invio del codice di sicurezza tramite *Button\_RecoveryButton*.
- **Relazioni:**
  - Collegata al controller *RecuperaPasswordController*.
  - Utilizza il database per verificare e inviare il codice di sicurezza.
  - Interagisce con componenti JavaFX (*Pane*, *Label*, *TextField*, *Button*) e con CSS personalizzati *CSS/StyleAccess.css*.

- **Diagramma delle interfacce grafiche nel package View:**



## 2.2 Diagramma delle classi:



## **2.3 Scelte Progettuali**

*Documentazione sulle scelte progettuali in termini di coesione, accoppiamento e principi di buona progettazione.*

La progettazione del sistema si basa sui principi di **coesione**, **accoppiamento** e sulle linee guida di buona progettazione (**SOLID**), al fine di ottenere un'architettura manutenibile, chiara ed estendibile.

**COESIONE:** *Misura quanto le funzionalità di un modulo sono correlate tra loro.*

Nel sistema la coesione è mantenuta **alta**, grazie alla suddivisione in moduli dalle responsabilità ben definite.

Ogni modulo, infatti, realizza un **unico e coerente obiettivo**.

- **Modulo Gestione Utenti:** le classi *Utente*, *UtentiController*, *AggiungiUtenteController* e *ModificaUtenteController* si occupano esclusivamente della gestione degli utenti (inserimento, modifica, ricerca, visualizzazione e gestione blacklist).
  - *Utente* rappresenta l'entità del dominio.
  - I controller gestiscono la logica di presentazione e interazione con la View.
  - Nessuna classe in questo modulo si occupa di prestiti, catalogo o notifiche.
- **Modulo Gestione Libri:** le classi *Libro*, *Autore*, *Catalogo*, *CatalogoController* e *AggiungiLibroController* si occupano esclusivamente della gestione dei libri.
  - *Libro* contiene solo dati editoriali.
  - *Autore* gestisce solo informazioni sugli autori.
  - *Catalogo* gestisce esclusivamente la collezione di libri e le ricerche.
  - I controller si limitano alla gestione dell'interfaccia e alla validazione dei dati, senza introdurre logiche estranee.
- **Modulo Gestione Prestiti:** le classi *Prestito*, *AggiungiPrestitoController* e *PrestitoRestituzioneController* si occupano soltanto del ciclo di vita dei prestiti.
  - *Prestito* rappresenta i dati del prestito.
  - I controller gestiscono:
    - creazione del prestito;
    - controlli (copie disponibili, limite prestiti, black-list);
    - gestione restituzioni;
    - filtri per stato (*ATTIVO*, *RESTITUITO*, *PROROGATO*, *IN\_RITARDO*).
- **Modulo Accesso e Account:** le classi *AccessoController* e *ControlloFormato* si occupano delle operazioni relative al profilo del bibliotecario.
  - *AccessoController* si occupa dell'autenticazione, registrazione e controllo del profilo.
  - La classe *ControlloFormato* si dedica unicamente al controllo del formato di password ed email.

- **Modulo DataBase:** la classe *DataBase* centralizza tutte le operazioni CRUD su utenti, libri, autori e prestiti. Tale scelta deriva dalla volontà di semplificare la gestione del database nel progetto.
  - Incapsula completamente la persistenza.
  - Nessun controller accede direttamente ai dati interni.
- **Modulo Notifiche:** la classe *EmailInvia* si occupa solo dell'invio email tramite SMTP. La coesione è massima perché non contiene logiche estranee.

**ACCOPIAMENTO:** *Rappresenta il grado di dipendenza tra moduli o componenti di un sistema software.*

Nel sistema l'accoppiamento è mantenuto **basso**, grazie a una chiara divisione tra Model, View e Controller e all'uso di interfacce funzionali ben definite tra le componenti.

- **Controller / Model:** I controller interagiscono solo tramite metodi pubblici del Model e non accedono direttamente ai dati. Tutte le operazioni di persistenza sono incapsulate nella classe *DataBase*.  
Le classi, come *CatalogoController*, *PrestitoRestituzioneController* e *UtentiController*, non gestiscono direttamente query SQL, ma si limitano a invocare metodi come *getCatalogo()*, *aggiungiPrestito()*, *cercaUtente()* o *modificaUtente()*. Ciò riduce fortemente la dipendenza tra la logica applicativa e il livello fisico di memorizzazione dei dati.
- **Model:** le entità di dominio (*Libro*, *Autore*, *Utente*, *EmailInfo*) contengono solo dati e metodi di accesso (getter/setter), senza dipendere da controller, database o view. Ciò garantisce zero accoppiamento con gli altri livelli.
- **Servizi esterni:** funzioni come invio email (*EmailInvia*) o transizioni grafiche (*TransizioneScena*) sono isolate in moduli dedicati, senza introdurre dipendenze circolari.
- **Validazioni:** la classe *ControlloFormato* centralizza le verifiche e viene utilizzata solo dai controller, evitando duplicazioni.
- **Struttura MVC:** garantisce che ogni livello comunichi solo con quelli adiacenti, riducendo le dipendenze incrociate:  
View → Controller  
Controller → Model  
Model → DataBase

## **PRINCIPI DI BUONA PROGETTAZIONE (SOLID)**

**Single Responsibility Principle (SRP):** *Ogni classe del sistema ha una responsabilità unica e ben definita.*

- *Libro, Utente, Autore e EmailInfo* contengono solo dati e metodi strettamente legati alle entità del dominio.
- I controller gestiscono solo l'interazione con la View e il coordinamento tra Model e UI.
- *DataBase* gestisce esclusivamente la persistenza dei dati.
- *EmailInvia* e *TransizioneScena* si occupano solo delle notifiche e delle transizioni tra View.

**Open-Closed Principle (OCP):** *Le classi sono progettate per poter subire estensioni future senza comportare modifiche ai componenti esistenti.*

- Si possono aggiungere nuove entità (es. *Docente, Studente*) estendendo *Utente* senza alterare la gestione dei prestiti o delle notifiche.
- Possono essere creati nuovi servizi, come backup avanzati o nuovi tipi di notifiche, senza toccare i controller o le classi del model esistenti (es. *Backup*).

**Liskov Substitution Principle (LSP):** *Le sottoclassi possono sostituire le classi base senza modificare il comportamento.*

- Nel progetto attuale non sono presenti controlli sui tipi concreti delle sottoclassi, ciò garantisce la possibilità di estendere le classi senza dover modificare il codice esistente.
- Esempio: una futura classe *Studente* o *Docente* potrebbe sostituire *Utente* ovunque fossero previsti utenti, mantenendo invariata la logica dei prestiti (*Prestito*) e delle notifiche (*EmailInvia*).
- I metodi pubblici come *getMatricola()* o *getMail()* manterrebbero la stessa validità anche in eventuali estensioni.

**Interface Segregation Principle (ISP):** *I controller e i servizi dipendono solo dalle funzionalità necessarie.*

- *CatalogoController* interagisce esclusivamente con le componenti del *DataBase* relative ai libri, senza accedere alle operazioni su utenti o prestiti,
- *UtentiController* interagisce esclusivamente con *Utente* e i metodi di gestione utenti del *DataBase*.
- Servizi specializzati come *EmailInvia* o *TransizioneScena* forniscono solo le interfacce necessarie per notifiche o animazioni, senza costringere i controller a dipendere da metodi inutilizzati.

**Dependency Inversion Principle (DIP):** *I moduli di alto livello non dipendono direttamente da quelli di basso livello.*

- L'utilizzo dell'**architettura MVC** permette una parziale implementazione del **DIP**.
- Tuttavia, molti Controller dipendono direttamente da classi concrete (*DataBase*, *EmailInvia* ed *EmailLegge*). L'introduzione di interfacce per l'accesso ai dati e ai servizi permetterebbe di rispettare pienamente il principio ma comporterebbe un aumento della complessità strutturale.

### **3. Modello Dinamico**

Documentazione sulle interazioni tra le classi nei casi d'uso più significativi e diagrammi di sequenza.

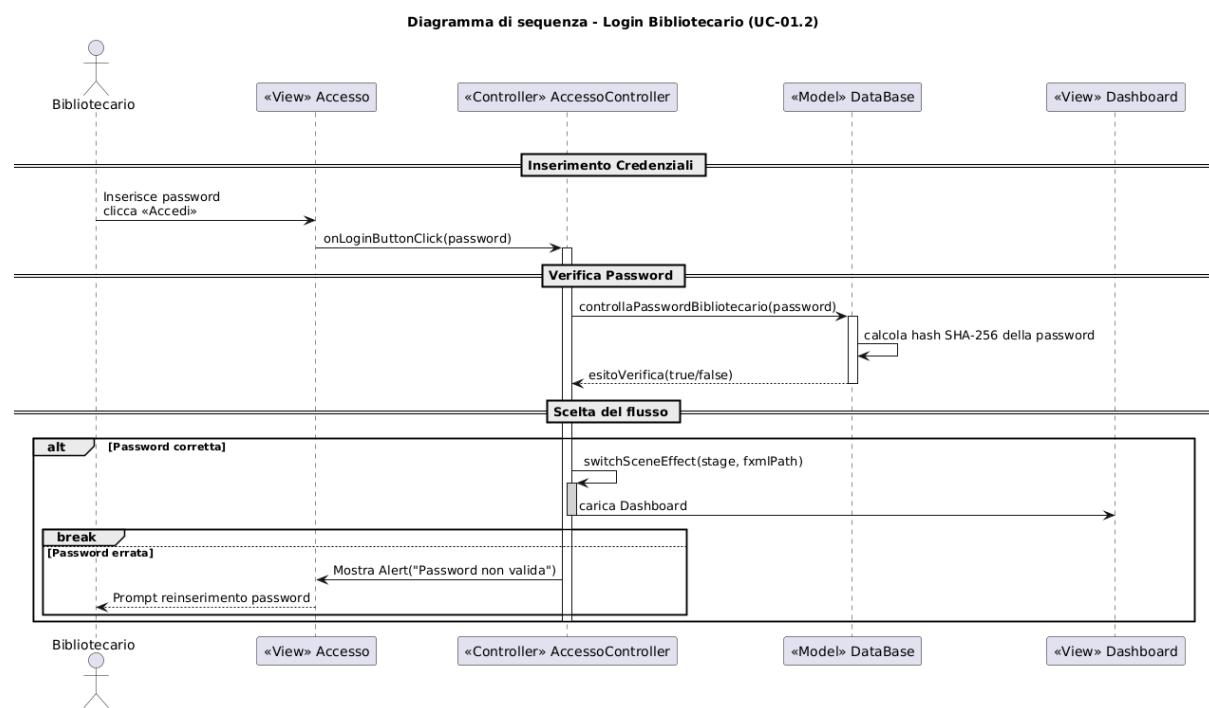
## **Gestione Accesso e Account**

### **Caso d'uso: Login (UC-01.2)**

Il diagramma descrive il processo di autenticazione del bibliotecario.

L'interazione avviene tra *Accesso.fxml* (interfaccia grafica), *AccessoController* (controller responsabile del flusso di accesso) e *DataBase* (che memorizza l'hash della password del bibliotecario).

*AccessoController* funge da mediatore: riceve le credenziali dalla View e delega al database la verifica dell'accesso.



### **Descrizione dell'interazione:**

#### **Flusso Principale:**

- Il bibliotecario inserisce le credenziali nella pagina di accesso *Accesso.fxml* e preme il pulsante "Accedi".
- L'interfaccia *Accesso.fxml* invia i dati all' *AccessoController* attraverso l'handler del bottone.
- AccessoController* richiama il metodo *controllaPasswordBibliotecario(String password)*
- Il metodo provvede a:
  - applicare l'algoritmo SHA-256 alla password inserita.
  - confrontare l'hash ottenuto con quello memorizzato dal database tramite *controllaPasswordBibliotecario(String password)*.
- Se la verifica ha successo, *AccessoController* esegue la transizione alla schermata principale utilizzando: *switchSceneEffect(Stage stage, String fxmlPath)*.

6. La Dashboard viene caricata correttamente e il bibliotecario accede al sistema.

**Flussi alternativi:**

*Caso A : Password errata*

3a. La funzione *controllaPasswordBibliotecario(String password)* rileva che l'hash della password non coincide con quello memorizzato.

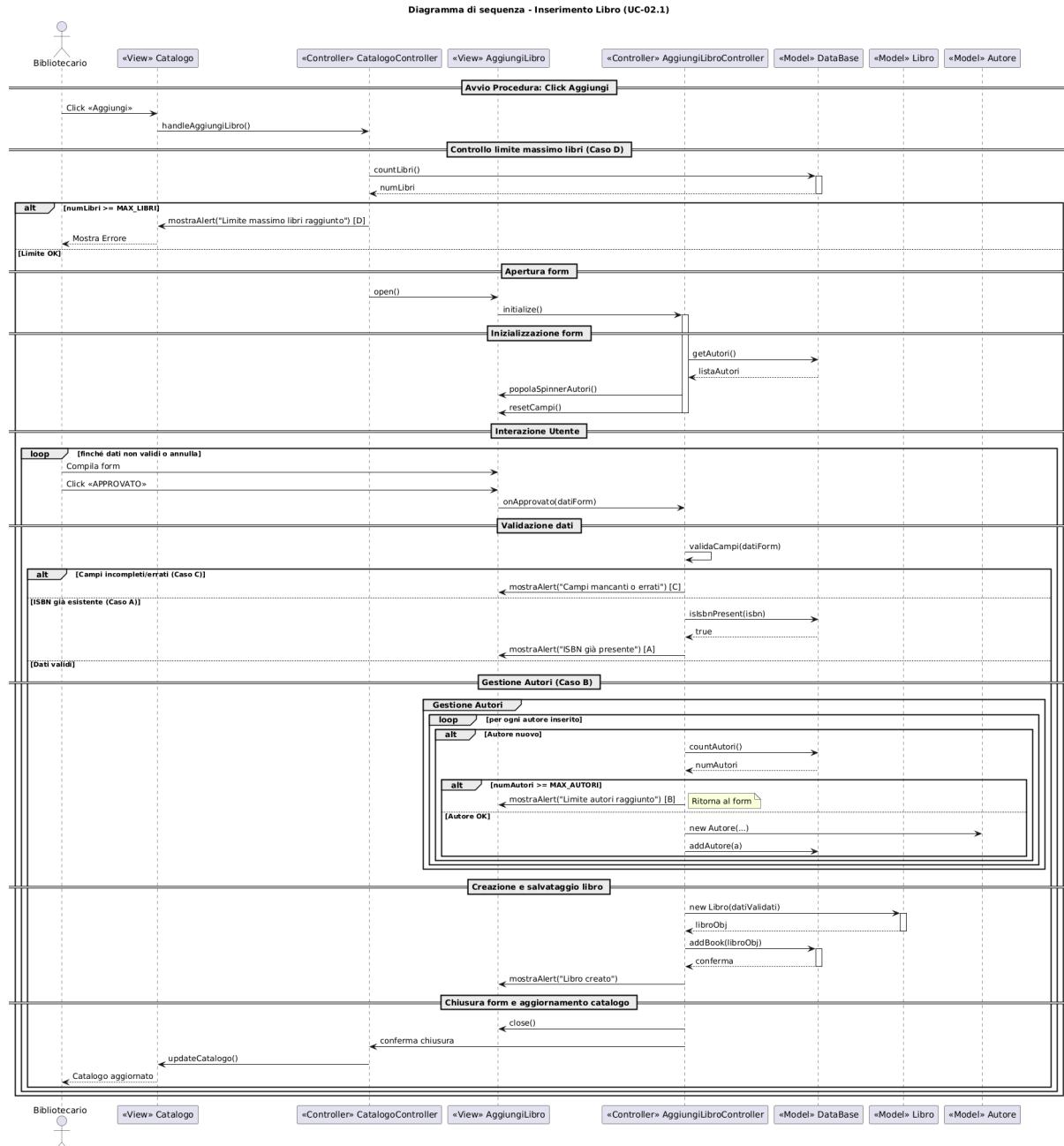
3a.1 Il Controller mostra un messaggio di errore tramite Alert.

3a.2 Si ritorna al passo 1.

# Gestione Libro

## Caso d'uso: Inserimento Libro (UC-02.1)

Il diagramma descrive il processo di inserimento di un nuovo libro nel catalogo da parte del bibliotecario. L'interazione avviene tra *Catalogo.fxml*, *CatalogoController* (permette l'apertura del form), *AggiungiLibro.fxml*, *AggiungiLibroController* (gestisce l'intera logica dell'inserimento) e *DataBase* (gestisce la memorizzazione dei libri e degli autori).



## Descrizione dell'interazione:

### Flusso Principale:

- Il bibliotecario si trova nel *Catalogo.fxml* e clicca sul pulsante “Aggiungi” (o icona “+” in fondo al catalogo).
- L'evento viene intercettato dal *CatalogoController* che apre la *AggiungiLibro.fxml*, che visualizza il form per l'inserimento di un nuovo libro.

3. L'*AggiungiLibroController* inizializza il form caricando eventuali dati necessari:
  - elenco degli autori già presenti.
  - settaggio degli spinner e campi numerici.
  - pulizia dei campi di testo.
4. Il bibliotecario compila i campi richiesti (Titolo, Autori, ISBN, Copie...) e preme il pulsante “APPROVATO”. L'*AggiungiLibro.fxml* invia i dati all'*AggiungiLibroController* tramite l'handler del bottone.
5. L'*AggiungiLibroController* effettua le verifiche sui dati inseriti (ISBN valido, campi obbligatori non vuoti, max libri catalogo...)
6. Per ogni autore non presente nel database ma indicato nel form, l'*AggiungiLibroController* chiama la funzione *aggiungiAutore(Autore a)* della classe database, verificando prima di inserire ogni autore se questa operazione causa problemi di memoria insufficiente.
7. L'*AggiungiLibroController* crea un nuovo oggetto *Libro* con tutti i dati validati.
8. Il controller invia il nuovo oggetto *Libro* al DataBase tramite *DataBase.aggiungiLibro(Libro libro)*, che lo registra nel catalogo.
9. Il form viene chiuso e il *CatalogoController* aggiorna l'elenco dei libri mostrati nella *Catalogo.fxml*. Ora il bibliotecario ritornerà al catalogo con il nuovo libro visualizzato.

#### **Flussi alternativi:**

*Caso A : ISBN già esistente*

5a. La verifica rileva che l'ISBN inserito è già presente nel database.

5a.1 L'*AggiungiLibroController* mostra un messaggio di errore tramite Alert.

5a.2 Si ritorna al passo 4.

*Caso B : Autore nuovo ma limite massimo raggiunto*

6b. L'inserimento di un nuovo autore supererebbe il limite massimo consentito.

6b.1 Il sistema mostra un Alert informando l'utente dell'impossibilità di aggiungere ulteriori autori.

6b.2 Si ritorna al passo 4.

*Caso C : Campi incompleti o non validi*

5c. Una verifica fallisce (campo mancante, formato errato, ecc.).

5c.1 Il controller mostra un Alert.

5c.2 Si ritorna al passo 4.

*Caso D : Memoria insufficiente per aggiungere nuovi libri*

1d. Il sistema nota che il numero massimo di libri consentiti è stato raggiunto.

2d.1 Viene mostrato un alert che comunica il messaggio.

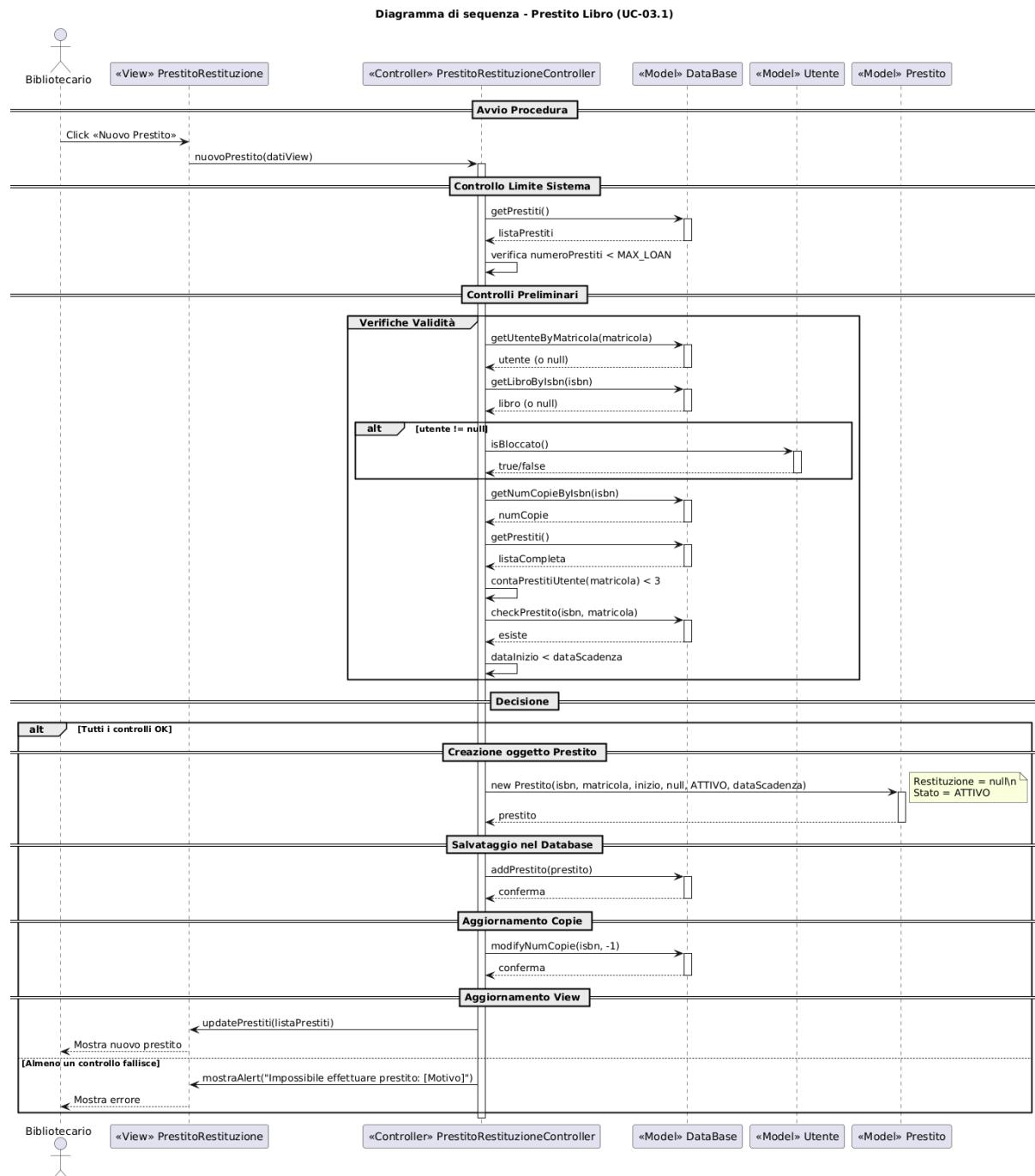
3d.2 Il bibliotecario chiude l'alert e si ritrova al passo 1.

## Caso d'uso: *Prestito Libro (UC-03.1)*

Il diagramma descrive il processo con cui un bibliotecario assegna un libro a un utente attraverso il modulo *Prestiti*.

L'interazione coinvolge *PrestitoRestituzione.fxml*, *PrestitoRestituzioneController* e *DataBase*, che gestisce la disponibilità dei libri e lo stato dei prestiti.

È fondamentale eseguire una serie di controlli applicativi "Business Rules" (copie disponibili, stato utente) prima di confermare l'operazione.



## Descrizione dell'interazione:

### Flusso Principale:

1. Il bibliotecario si trova nella *PrestitoRestituzione.fxml* e clicca sul pulsante “Nuovo Prestito”.
2. Il controller verifica che l'aggiunta di un ulteriore prestito non superi il limite massimo di prestiti gestibili dal sistema sfruttando la funzione *DataBase.getPrestiti()* (applicando *.size()* ).
3. L'evento viene intercettato dal *PrestitoRestituzioneController* che riceve la richiesta e avvia in controlli preliminari interrogando il database:
  - che la matricola e l'ISBN inseriti siano registrati nel sistema utilizzando le funzioni *cercaLibro(String ISBN)* e *cercaUtente(String matricola)*.
  - che l'utente non sia in blacklist tramite il metodo *cercaUtente(String matricola)*, il quale restituisce l'utente (o null) su cui sarà invocato il metodo *getter isBloccato()*.
  - che il libro selezionato abbia almeno una copia disponibile utilizzando la funzione *getNumCopieByISBN(String ISBN)*.
  - che l'utente non abbia già 3 prestiti attivi, controllo ricavato sfruttando la *getPrestiti()*.
  - che l'utente non abbia già in prestito lo stesso libro usando *controllaPrestito(String ISBN, String matricola)*.
  - data di inizio del prestito deve venire prima della data di scadenza (*isBefore()*).
4. Se tutti i controlli hanno esito positivo, il controller:
  - crea un nuovo oggetto *Prestito* contenente: matricola, ISBN, data di Inizio, data di scadenza (questi 4 attributi li ottiene tramite la view), imposta automaticamente stato del prestito su attivo 'ATTIVO' e data restituzione a null.
  - invia l'oggetto al database tramite un'operazione di inserimento *DataBase.aggiungiPrestito(Prestito prestito)*.
5. Il DataBase salva il nuovo prestito e conferma l'operazione al controller.
6. Il controller aggiorna la disponibilità del libro, richiamando un metodo (*DataBase.modificaNum\_copie(String ISBN, boolean add)*) che riduce di 1 il numero di copie disponibili.
7. La *PrestitoRestituzione.fxml* viene aggiornata con i nuovi record dei prestiti.

### Flussi alternativi:

Caso A : Almeno un controllo che effettua il controller fallisce

3a.1 Il *PrestitoRestituzioneController* mostra un Alert informando l'operatore dell'impossibilità di effettuare prestiti, specificando il motivo.

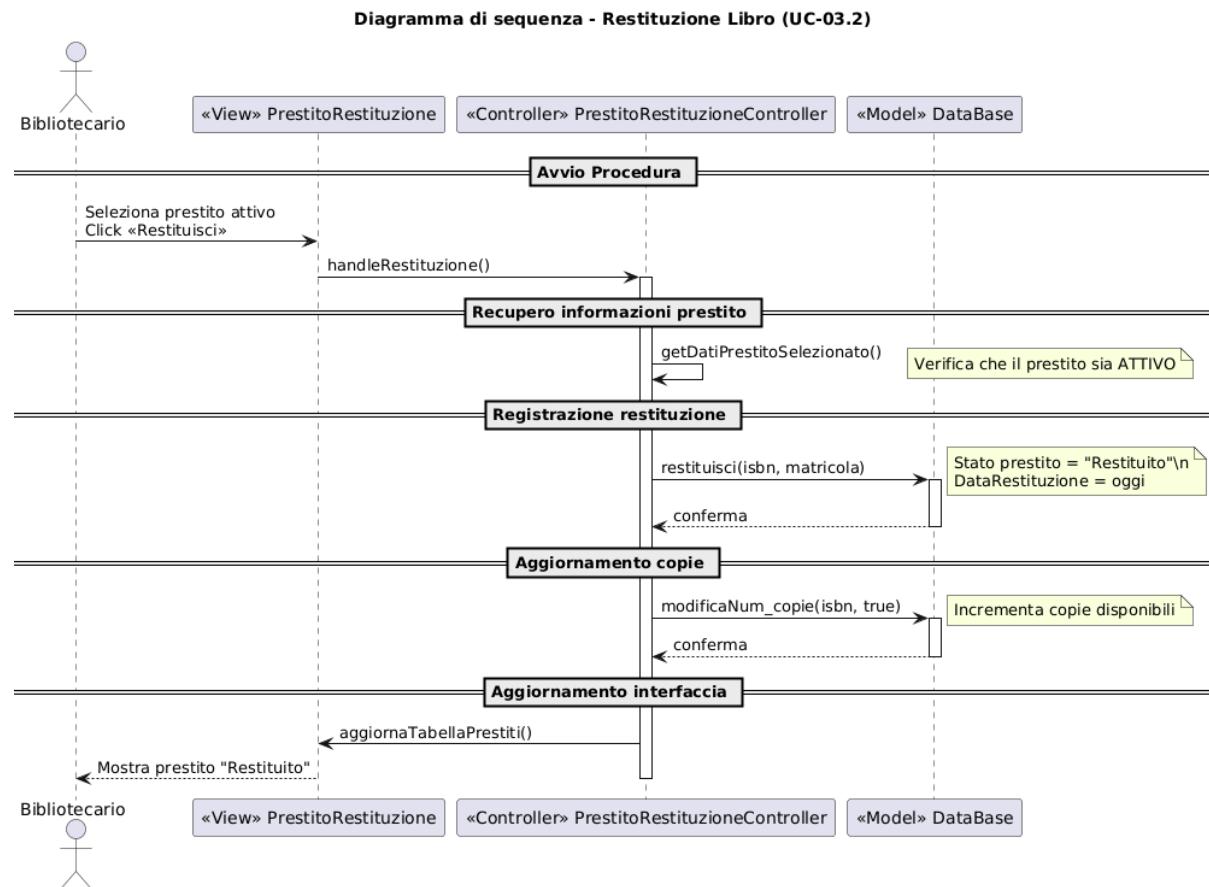
3a.1 Si ritorna al passo 1.

## Caso d'uso: Restituzione Libro (UC-03.2)

Il diagramma descrive il processo con cui un bibliotecario registra la restituzione di un libro precedentemente prestato.

L'interazione coinvolge *PrestitoRestituzione.fxml*, *PrestitoRestituzioneController* e *DataBase* che memorizza lo stato dei prestiti e la disponibilità dei libri.

L'operazione comporta sia la modifica dei dati del prestito sia l'aggiornamento del numero di copie del libro.



### Descrizione dell'interazione:

#### Flusso Principale:

- Il bibliotecario, nella *PrestitoRestituzione.fxml*, seleziona un prestito attivo e clicca sul pulsante “Ritorna”.
- L'evento viene intercettato dal *PrestitoRestituzioneController*, che recupera le informazioni del prestito e aggiorna lo stato a “Ritornato”, registrando:
  - la data di restituzione (quella attuale al momento del click sul bottone)
  - la chiusura formale del prestito
 tutto implementato dalla funzione *restituisce(String ISBN, String matricola)*
- Il *PrestitoRestituzioneController* aggiorna la disponibilità del libro associato, incrementando il numero di copie disponibili nel catalogo tramite la funzione *modificaNum\_Copie(String ISBN, boolean add)*.

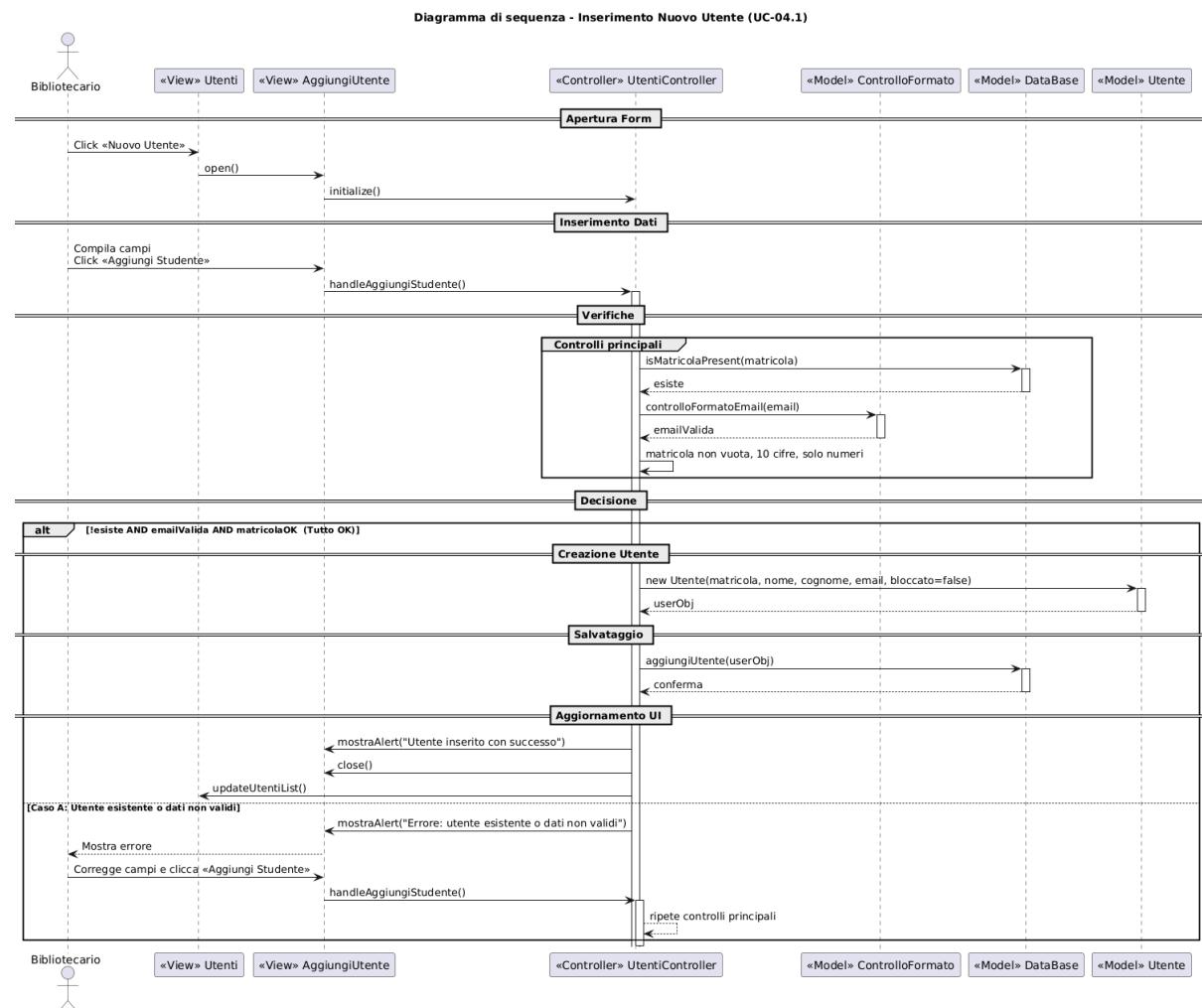
4. Il Controller invia tutta le modifiche al DataBase, che:
  - salva l'aggiornamento dello stato del prestito
  - salva l'incremento di copie del libro
  - conferma la riuscita dell'operazione
5. La *PrestitoRestituzione.fxml* viene aggiornata mostrando il prestito come restituito.

## Caso d'uso: Inserimento Nuovo Utente (UC-04.1)

Il diagramma descrive il processo di registrazione di un nuovo utente.

L'interazione avviene tra *AggiungiUtente.fxml*(interfaccia grafica), *AggiungiUtenteController* (controller responsabile della logica di inserimento) e *DataBase* (che memorizza le informazioni degli utenti).

Il Controller funge da mediatore: riceve i dati dalla View e delega al database la verifica e il salvataggio.



## Descrizione dell'interazione:

**Flusso Principale:**

- Il bibliotecario clicca il pulsante “Nuovo utente” che apre *AggiungiUtente.fxml* che inizializza i campi del form.
- Il bibliotecario inserisce i campi e clicca sul bottone “Aggiungi Studente”.
- Il controller procede a verificare vari controlli:
  - Matricola già registrata, controllo effettuato con il metodo *isMatricolaPresent(String matricola)* della classe *DataBase*.
  - Controllo formato email con *controlloFormatoEmail(String email)*.
  - Vari controlli (campo matricola non vuoto, matricola a 10 cifre e solo numeri)
- Se la verifica ha esito positivo, il controller crea l’oggetto *utente* e con *aggiungiUtente(Utente u)* lo inserisce nel database.
- Il database conferma l’inserimento mostrando un alert di successo.

#### **Flussi alternativi:**

Caso A : Utente già presente / dati non validi

3a. Il database segnala che l’utente esiste già o che i dati non rispettano i vincoli di validità (es. formato e-mail).

3a.1 Il Controller mostra un alert nella View con l’errore.

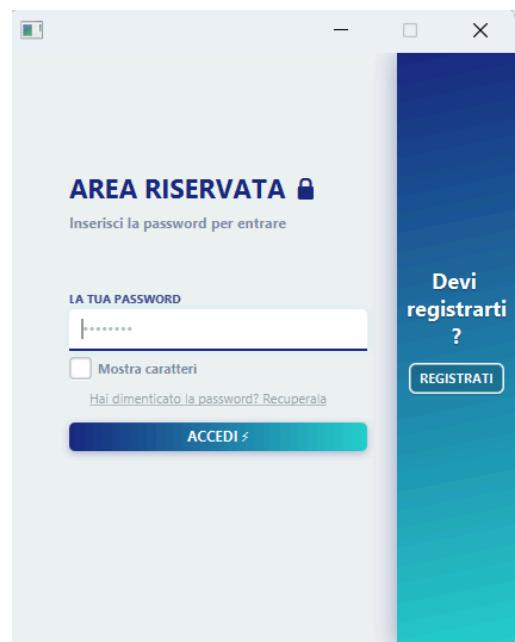
3a.2 L’utente corregge i dati ritrovandosi al passo 2.

## **4. Design Dell’Interfaccia Utente**

Mock-up delle schermate principali, descrizione e illustrazione delle principali interazioni con l’utente.

### **1) INTERFACCIA LOGIN**

- Scopo:** Consentire al bibliotecario di accedere al sistema.
- Elementi principali:**
  - Campo password.
  - Casella di selezione “Mostra caratteri”.
  - Pulsante “Accedi”.
  - Pulsante “Registrati” (per passare alla modalità di registrazione).
- Interazioni:**
  - Inserendo la password e cliccando “Accedi”, il bibliotecario entra nel sistema.
  - Se la password è errata, viene mostrata una notifica di errore.
  - Se il bibliotecario non è registrato, cliccando “Registrati” passa al form di registrazione.



## 2) INTERFACCIA REGISTRAZIONE

- **Scopo:** Permettere al bibliotecario non registrato di creare un account.
- **Elementi principali:**
  - Campo email.
  - Campo nuova password.
  - Campo conferma password.
  - Casella di selezione “Mostra caratteri”.
  - Pulsante “Salva e Accedi”.
  - Pulsante “Accedi” (torna al login).
- **Interazioni:**
  - Inserendo l'email e la password e confermando la password, cliccando “Salva e Accedi”, il sistema registra il bibliotecario il quale può accedere alla dashboard.
  - Cliccando “Accedi” il bibliotecario torna alla schermata di login.



## 3) INTERFACCIA RECUPERO PASSWORD

- **Scopo:** Permettere al bibliotecario di richiedere un codice di verifica per resettare la password.
- **Elementi principali:**
  - Campo di testo per inserire l'indirizzo email registrato.
  - Pulsante “Inviami il codice”.
- **Interazioni:**
  - Il bibliotecario inserisce la propria email e clicca “Inviami il codice”.
  - Se l'email è presente nel sistema, il sistema invia un codice di verifica a 6 cifre all'indirizzo indicato e passa alla schermata di verifica identità.
  - Se l'email non è presente, il sistema mostra il messaggio “Email non presente nel sistema!”; l'utente preme “OK” e può reinserire l'email.



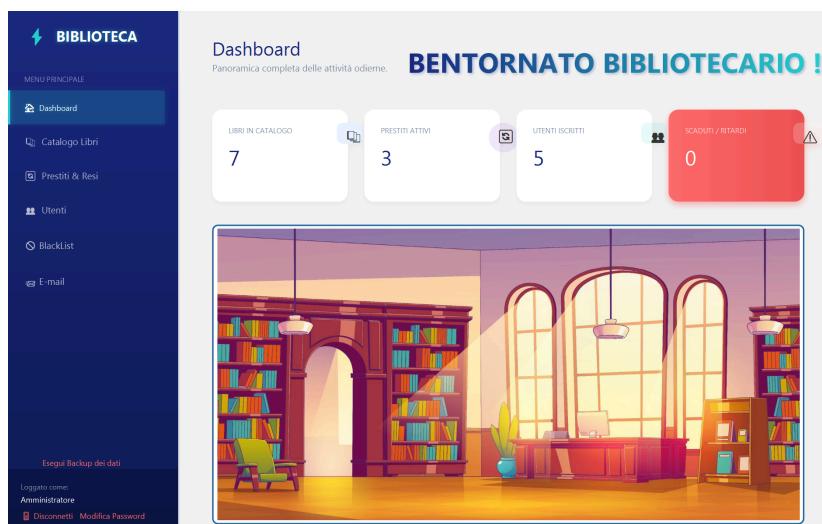
#### 4) INTERFACCIA VERIFICA IDENTITÀ

- Scopo:** Consentire al bibliotecario di confermare la propria identità inserendo il codice ricevuto via email.
- Elementi principali:**
  - Testo informativo: "Abbiamo inviato un codice a 6 cifre alla tua email".
  - Campo di inserimento del codice di verifica.
  - Testo secondario: "Non hai ricevuto il codice? Invia di nuovo".
  - Pulsante "Verifica codice".
- Interazioni:**
  - Il bibliotecario inserisce il codice ricevuto e clicca "Verifica codice".
  - Se il codice è corretto, il sistema reindirizza alla schermata di inserimento nuova password.
  - Se il codice non è corretto, il sistema mostra un messaggio di errore e permette di reinserirlo.
  - Cliccando su "Invia di nuovo", il sistema genera e invia un nuovo codice all'email registrata.



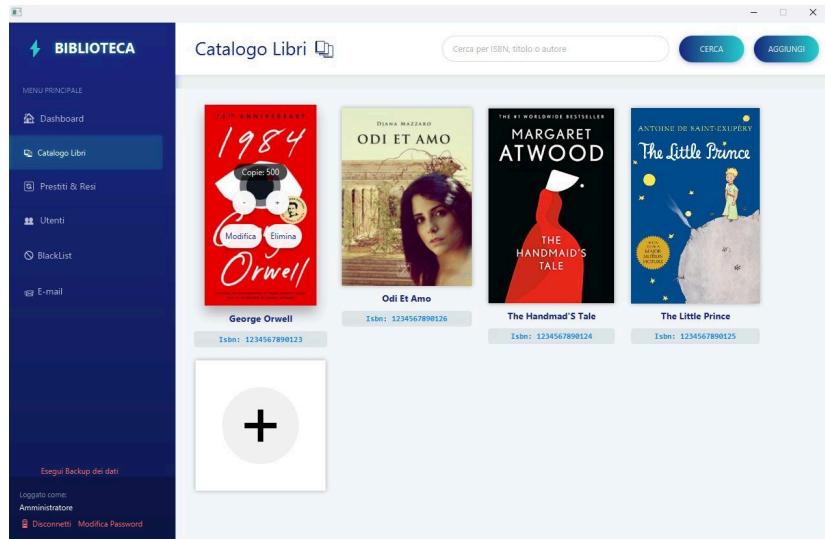
#### 5) INTERFACCIA DASHBOARD

- Scopo:** Visualizzare informazioni generali e navigare tra le sezioni principali.
- Elementi principali:**
  - Riepilogo generale dei dati del sistema.
  - Menu di navigazione laterale.
  - Pulsanti: "Logout", "Modifica password", "Backup Dati".
- Interazioni:**
  - Il menu laterale permette di passare alle sezioni Catalogo, Prestiti, Utenti, BlackList e E-mail.
  - I pulsanti consentono di effettuare operazioni rapide sulla gestione dell'account e dei dati.



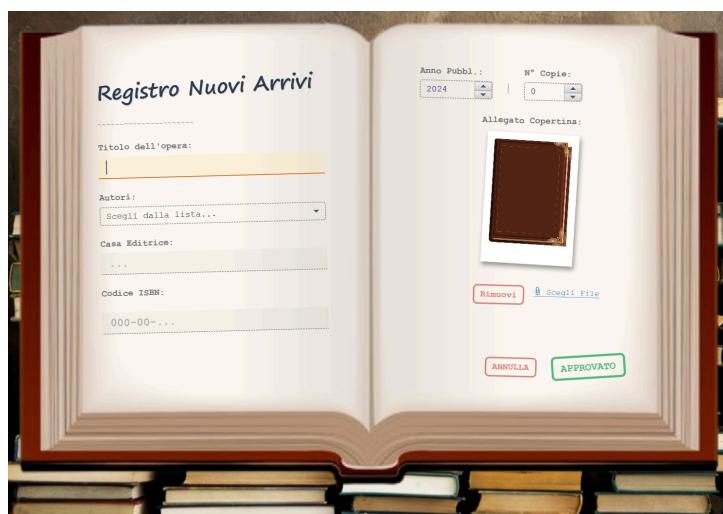
## 6) INTERFACCIA CATALOGO

- **Scopo:** Visualizzare e gestire i libri presenti nel sistema.
- **Elementi principali:**
  - Card dei libri (copertina, titolo).
  - Pulsanti per aggiungere/rimuovere copie, modificare o eliminare il libro.
  - Barra di ricerca per ISBN, titolo o autore.
  - Pulsante “Cerca”.
  - Pulsante “Aggiungi”.
- **Interazioni:**
  - Passando il mouse sopra una card, si mostrano le opzioni di modifica/eliminazione.
  - La barra di ricerca filtra i libri in base a titolo o autore.



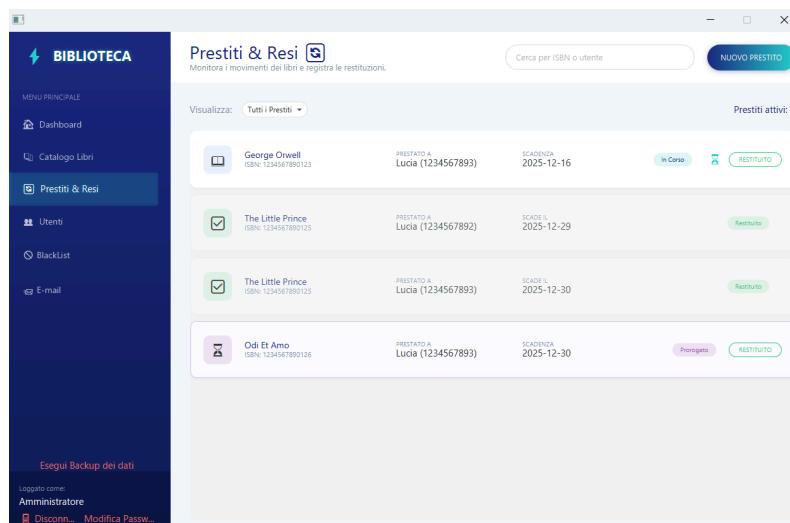
### 6.1) INTERFACCIA AGGIUNGI LIBRO

- **Elementi principali:**
  - Campi di testo per titolo, autore, ISBN, ecc.
  - Bottone per allegare copertina (“Scegli file” e “Rimuovi”).
  - Pulsanti “Approvato” e “Annulla”.
- **Interazioni:**
  - Dopo aver compilato i campi, cliccando “Approvato” il libro viene salvato nel sistema.



## 7) INTERFACCIA DASHBOARD PRESTITI

- **Scopo:** Gestire i prestiti dei libri agli utenti.
- **Elementi principali:**
  - Card dei prestiti (utente, libro, date, stato).
  - Icône per restituzione, proroga, invio avviso.
  - Pulsanti per filtrare prestiti per stato.
  - Pulsante “Nuovo Prestito”.
  - Barra di ricerca.
- **Interazioni:**
  - Le icône mostrano lo stato del prestito: attivo, in ritardo, restituito.
  - Cliccando sull’icona della clessidra si proroga il prestito.
  - Cliccando sull’icona della posta si invia un avviso.



### 7.1) INTERFACCIA REGISTRAZIONE PRESTITO

- **Elementi principali:**
  - Campi ISBN e matricola utente.
  - Pulsanti per verificare ISBN e matricola nel sistema.
  - Datepicker per data inizio e scadenza prestito.
  - Pulsanti “Conferma prestito” e “Annulla”.
- **Interazioni:**
  - Verifica automatica dei codici prima di confermare il prestito.

A screenshot of a modal window titled 'Concedi Prestito' with the sub-section 'REGISTRAZIONE PRESTITO'. The form is titled 'Compila i dati per autorizzare l'uscita del volume.' It has three main sections: 'DATI IDENTIFICATIVI' (ISBN input field with a search icon), 'MATRICOLA UTENTE' (user ID input field with a person icon), and 'TERMINI DEL PRESTITO' (date pickers for 'DATA INIZIO' [15/12/2025] and 'SCADENZA PREVISTA'). At the bottom are 'ANNULLA' and 'CONFERMA PRESTITO' buttons. The background features two images of stacks of books.

## 8) INTERFACCIA UTENTI

- **Scopo:** Gestire gli utenti della biblioteca.
- **Elementi principali:**
  - Card utenti (nome, matricola, stato).
  - Icone Modifica, Elimina, Blocca/Sblocca.
  - Pulsante Aggiungi utente.
  - Barra di ricerca e filtri per stato.
- **Interazioni:**
  - Possibilità di modificare, eliminare o bloccare gli utenti.
  - Il filtro consente di visualizzare solo gli utenti attivi o nella BlackList.

Nome	Email	Matricola	Stato	Azioni
Lucia Isevoli	Isevoli@studenti.unisa.it	1234567893	Attivo	<a href="#">Modifica</a> <a href="#">Elimina</a> <a href="#">Blocca</a>
Nicola Miranda	n.miranda@studenti.unisa.it	1234567889	Attivo	<a href="#">Modifica</a> <a href="#">Elimina</a> <a href="#">Blocca</a>
Lucia Monetta	l.monetta@studenti.unisa.it	1234567892	Attivo	<a href="#">Modifica</a> <a href="#">Elimina</a> <a href="#">Blocca</a>
Michele Tamburro	m.tamburro@studenti.unisa.it	1234567891	Blacklist	<a href="#">Modifica</a> <a href="#">Elimina</a> <a href="#">Sblocca</a>

### 8.1) INTERFACCIA AGGIUNGI UTENTE

- **Elementi principali:**
  - Form con campi nome, cognome, matricola, email, ecc.
  - Pulsanti “Aggiungi studente” e “Annulla”.
- **Interazioni:**
  - Il bibliotecario può inserire nuovi utenti immettendo i dati nell'apposito form e cliccando “Aggiungi studente”.

BIBLIOTECA UNIVERSITARIA

NUOVO STUDENTE

MATRICOLA (ID UTENTE)

NOME: MARIO

COGNOME: ROSSI

EMAIL ISTITUZIONALE: mario.rossi@studenti.it

ANNULLA **AGGIUNGI STUDENTE**

## 8.2) INTERFACCIA MODIFICA UTENTE

- **Elementi principali:**

- Stessa struttura del form di aggiunta, con pulsanti “Salva modifiche” e “Annulla”.

- **Interazioni:**

- Il bibliotecario inserisce i dati aggiornati negli appositi campi.
- Il sistema memorizza i cambiamenti dopo che il bibliotecario ha cliccato “Salva modifiche”.

MODIFICA UTENTE  
MATRICOLA: 1234567890

NOME  
|

COGNOME  
Rossi

E-MAIL ISTITUZIONALE  
mario.rossi@studenti.it

ANNULLA SALVA MODIFICHE

## 9) BLACKLIST

- **Scopo:** Gestire gli utenti bloccati per inadempienze.

- **Elementi principali:**

- Card utenti bloccati.
- Pulsante “Sblocca” utente per singolo utente.
- Pulsante “Sblocca tutti” gli utenti.
- Barra di ricerca.

- **Interazioni:**

- Possibilità di sbloccare singoli utenti o tutti insieme.
- Invio notifiche via email per informare l’utente dello sblocco o di prestiti in ritardo.

BIBLIOTECA

MENU PRINCIPALE

- Dashboard
- Catalogo Libri
- Prestiti & Resi
- Utenti
- BlackList**
- E-mail

Esegui Backup dei dati

Loggato come:  
Amministratore

Conn... Modifica Passw...

Blacklist Utenti ❌  
Gestione restrizioni e riammissione utenti.

Cerca per matricola bloccata

SBLOCCA TUTTI

Utenti attualmente sospesi: 1 Utenti Bloccati

Michele Tamburro  
m.tamburro@studenti.unisa.it  
MATRICOLA  
1234567891

Blacklist Sblocca

## 10) E-MAIL

- **Scopo:** Visualizzare gli avvisi inviati agli utenti.
- **Elementi principali:**
  - Lista degli ultimi 40 avvisi inviati.
- **Interazioni:**
  - Controllo rapido degli avvisi non inviati o più datati.

The screenshot shows the 'BIBLIOTECA' application interface. On the left, there is a dark sidebar with the title 'BIBLIOTECA' at the top, followed by 'MENU PRINCIPALE' and several menu items: 'Dashboard', 'Catalogo Libri', 'Prestiti & Resi', 'Utenti', 'BlackList', and 'Email' (which is highlighted in blue). Below these, there are links for 'Esegui Backup dei dati', 'Logout come: Amministratore', and 'Disconnetti Modifica Password'. The main content area is titled 'Comunicazioni Ritardi Prestiti' with a small info icon. It displays a list of 11 sent emails, each with a small profile icon, the subject (e.g., 'Mancata Restituzione del/dei libro/i'), the recipient (e.g., 'A: m.tamburo@studentiunisa.it'), and the send date ('INVITATA IL 06/12/2025 18:36').

## 11) NOTIFICA

- **Scopo:** Avvisare il bibliotecario di prestiti in scadenza o utenti inadempienti.
- **Elementi principali:**
  - Pop-up che compare alla mezzanotte della data di scadenza del prestito o all'accesso al sistema.
- **Interazioni:**
  - Suggerimento per inviare avvisi via email.

