

GRUPPO 22

Lavoro svolto da: Iasevoli Lucia, Monetta Lucia, Miranda Nicola e Tamburro Michele.

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

- **Model** : contiene la logica applicativa e la gestione dei dati (si occupa dell'interazione con il database MySQL, della validazione delle informazioni e della struttura delle entità);
- **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 dell'utente, invoca la logica del Model e aggiorna la View con i risultati delle operazioni.

Il Model, per garantire maggiore **robustezza** e **separazione** tra logica applicativa e accesso ai dati, è organizzato ulteriormente in:

- **Entity Model (DataClass)**: rappresentazione delle entità principali (Libro, Utente, Prestito).

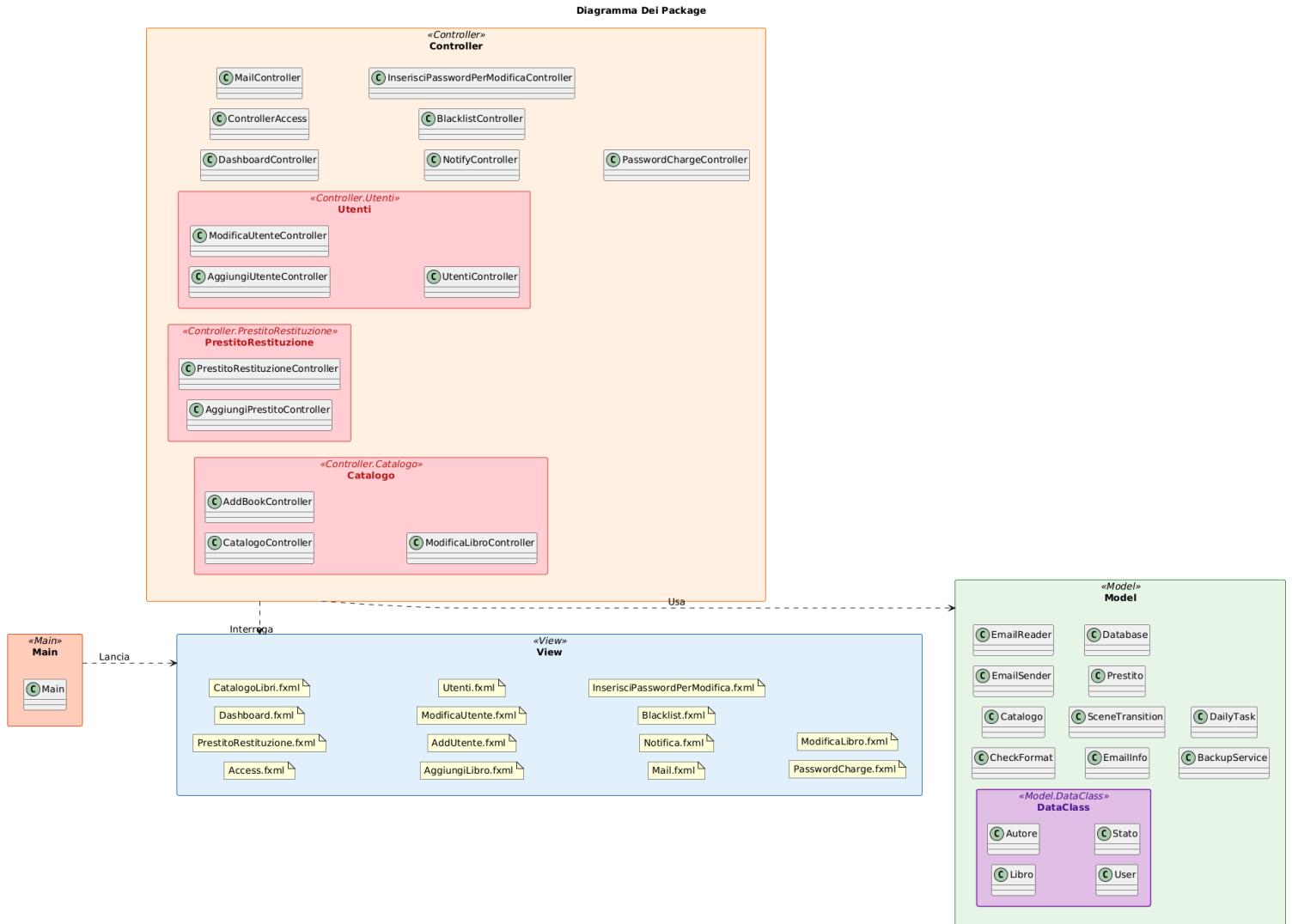
1.2 Moduli Principali

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.	Model Bibliotecario, DataBase, MySQL, UI-01, UI-02	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.	Model Libro, DataBase MySQL, UI-04	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.	Model Prestito, Model Libro, Model Utente, DataBase, UI-05	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.	Model Utente, DataBase MySQL, UI-06	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.	Model Black-list, Model Utente, DataBase, UI-09	Controller	IF-19, IF-20; BF-11; DF-05; 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.	Model Prestito, Servizio e-mail, UI-08	Controller	IF-23; BF-10; DF-03; 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.	DBMS MySQL, mysqldump	Controller	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-05; 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.	Server SMTP/TLS	Controller	IS-01; IF-23

1.3 Diagramma Dei Package



2. Modello Statico

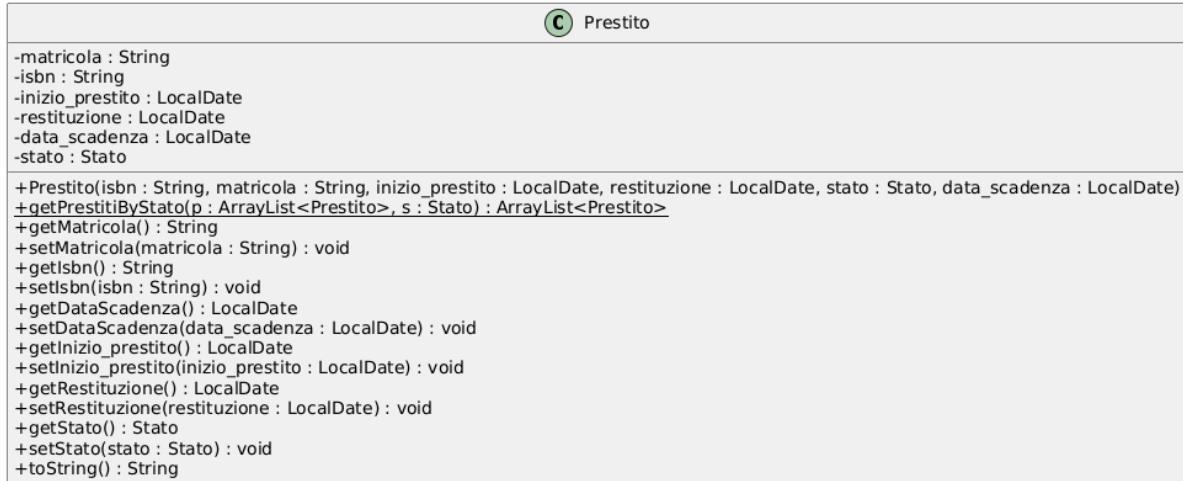
Descrizione delle classi (responsabilità, attributi principali, metodi pubblici e relazioni rilevanti). Documentazione sulle scelte progettuali in termini di coesione, accoppiamento e principi di buona progettazione.

2.1 Descrizione Delle Classi

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

PACKAGE MODEL

Nome Classe: Prestito



- **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, getRestituzione() : LocalDate, setRestituzione(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.

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.
 - *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*.

C Catalogo
-libri : List<Libro>
+Catalogo()
+aggiungiLibro(libro : Libro) : void
+rimuoviLibro(libro : Libro) : void
+getLibri() : ArrayList<Libro>
+cercaPerISBN(isbn : String) : Libro
+cercaPerTitolo(titolo : String) : List<Libro>
+cercaPerAutore(autore : String) : List<Libro>

Nome Classe: CheckFormat

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

C CheckFormat
+CheckPasswordFormat(password : String) : boolean
+CheckEmailFormat(email : String) : boolean

Nome Classe: DataBase

- **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:**
 - *DBInitialize() : void* - Stabilisce la connessione.
 - *InsertBibliotecario(String password) : boolean*, *CheckIfExistsBibliotecario() : boolean*

C DataBase
-conn : Connection
-DB_name : String
+DBInitialize() : void
+InsertBibliotecario(password : String) : boolean
+CheckIfExistsBibliotecario() : boolean
+CheckPasswordBibliotecario(password : String) : boolean
+RemoveBibliotecario() : boolean
+getCatalogo() : Catalogo
+getAutori() : ArrayList<Autore>
+addBook(l : Libro) : boolean
+addAutore(a : Autore) : boolean
+searchBook(isbn : String) : Libro
+getNum_Autori() : int
+SearchAutorByNames(nome : String, cognome : String) : Autore
+getUtenti() : ArrayList<User>
+addUser(u : User) : boolean
+getNumUser() : int
+searchUser(matricola : String) : User
+setBlackListed(matricola : String) : boolean
+UnsetBlackListed(matricola : String) : boolean
+ModifyUser(matricola : String, nome : String, cognome : String, mail : String) : boolean
+getPrestiti() : ArrayList<Prestito>
+addPrestito(p : Prestito) : boolean
+getNumLoan() : int
+Restituisci(isbn : String, matricola : String) : boolean
+CheckPrestito(isbn : String, matricola : String) : boolean
+RemovePrestito(isbn : String, matricola : String) : boolean
+getNumCopieByisbn(isbn : String) : int
+ModifyNum_copie(isbn : String, add : boolean) : boolean

- *boolean CheckPasswordBibliotecario(String password) : boolean* - Gestione autenticazione bibliotecario.
 - *RemoveBibliotecario() : 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.
 - *addBook(Libro l) : boolean, addAutore(Autore a) : boolean* - Persistenza di nuovi dati.
 - *searchBook(String ISBN) : Libro* - Cerca un libro nel catalogo.
 - *getNum_Autori() : int, SearchAutorByNames(String nome, String cognome) : Autore* - Gestiscono la ricerca/numero di autori.
 - *getUtenti() : ArrayList<User>* - Restituisce gli utenti.
 - *addUser(User u) : boolean* - Aggiunge un utente nel database.
 - *getNumUser() : int* - Restituisce numero di utenti.
 - *searchUser(String matricola) : User* - 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.
 - *ModifyUser(String matricola, String nome, String cognome, String mail) : boolean* - Aggiorna i dati di un utente.
 - *getPrestiti() : ArrayList<Prestito>* - Restituisce la lista di prestiti.
 - *addPrestito(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).
 - *CheckPrestito(String ISBN, String matricola) : boolean* - Verifica lo stato del prestito.
 - *RemovePrestito(String ISBN, String matricola) : boolean* - Rimuove un prestito dal database.
 - *getNumCopieByISBN(String ISBN) : int* - Restituisce il numero di copie.
 - *ModifyNum_copie(String ISBN, boolean add) : boolean* - Modifica il numero di copie.
 - *isISBNPresent(String ISBN) : boolean* - Controlla se l'ISBN è presente nel sistema.
 - *isMatricolaPresent(String matricola) : boolean* - Controlla se la matricola è presente nel sistema.
- **Relazioni:**
 - È utilizzata dai Controller per l'accesso ai dati persistenti.

Nome Classe: EmailSender

- **Responsabilità:** Utility per l'invio di email tramite protocollo SMTP (es. notifiche).
- **Attributi:**
 - *username: String* - Email del bibliotecario.
 - *password: String* - Password del bibliotecario.
- **Metodi Principali:**
 - *sendEmail(String recipient, String subject, String body) : void* - Configura la sessione e invia il messaggio.
- **Relazioni:**
 - È utilizzata dai Controller per inviare email.

 EmailSender
-username : String
-password : String

+sendEmail(recipient : String, subject : String, body : String) : void

Nome Classe: EmailReader

- **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.
- **Metodi Principali:**
 - *leggiPostalInvia()*: *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	EmailReader
-username : String	
-password : String	
+leggiPostalInvia() : ArrayList<EmailInfo>	

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 *EmailReader* per creare oggetti contenenti le informazioni delle email lette dal server.

C	EmailInfo
-oggetto : String	
-destinatario : String	
-dataInvio : Date	
+EmailInfo(oggetto : String, destinatario : String, dataInvio : Date)	
+getOggetto() : String	
+getDestinatario() : String	
+getDataInvio() : Date	
+toString() : String	

Nome Classe: SceneTransition

- **Responsabilità:** Gestione transizioni tra view.
- **Attributi:**
 - *Nessuno*
- **Metodi Principali:**

C	SceneTransition
+switchSceneEffect(stage : Stage, fxmlPath : String) : void	

- `switchSceneEffect(Stage stage, String fxmlPath) : void` - Cambio di scena con FadeTransition.
- **Relazioni:**
 - Usato da *DashboardController* per caricare FXML.

Nome Classe: DailyTask

- **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:**
 - `scheduler: ScheduledExecutorService` - Executor per pianificare task ricorrenti a intervalli fissi.
- **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.
- **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.

 DailyTask
-scheduler : ScheduledExecutorService
+avviaTaskDiMezzanotte() : void
+calcolaRitardoVersoMezzanotte() : long
+eseguiControlliAutomatici(SkipNotify : boolean) : void
+stop() : void

Nome Classe: BackupService

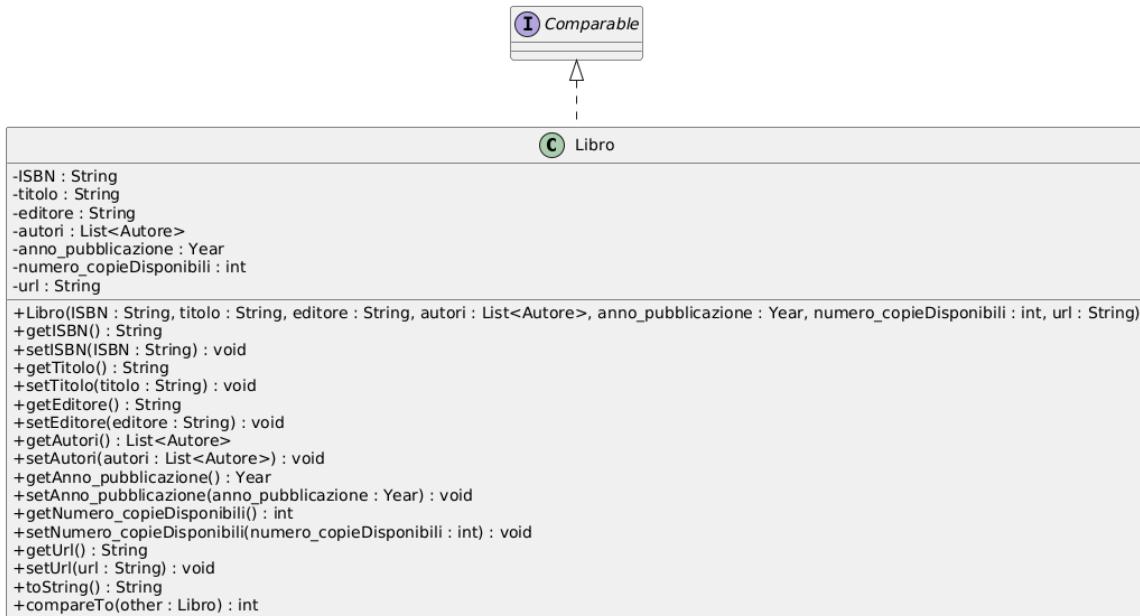
- **Responsabilità:** Esegue il backup del database della biblioteca esportando i dati in un file .sql nella cartella specificata, gestendo automaticamente nome del file e percorso di destinazione.
- **Attributi:**
 - `DB_NAME: String` - Nome del database da salvare.
 - `DB_USER: String` - Nome utente 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.
- **Relazioni:**

 BackupService
-DB_NAME : String
-DB_USER : String
-DB_PASS : String
-MYSQL_DUMP_PATH : String
+eseguiBackup(cartellaDestinazione : String) : boolean

- 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).

PACKAGE MODEL.DATACLASS

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.
- **Relazioni:**
 - Aggregazione con *Autore*.
 - Gestito da *Catalogo*.
 - Caricato/Salvato tramite *DataBase*.

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.
 - *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* - Gestione 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*.

```

    Autore
    -id : int
    -nome : String
    -cognome : String
    -opere_scritte : int
    -data_nascita : LocalDate

    +Autore(nome : String, cognome : String, opere_scritte : int, data_nascita : LocalDate)
    +getId() : int
    +setId(id : int) : void
    +getNome() : String
    +setNome(nome : String) : void
    +getCognome() : String
    +setCognome(cognome : String) : void
    +getOpere_scritte() : int
    +setOpere_scritte(opere_scritte : int) : void
    +getData_nascita() : LocalDate
    +setData_nascita(data_nascita : LocalDate) : void
    +toString() : String

```

Nome Classe: User

- **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.

```

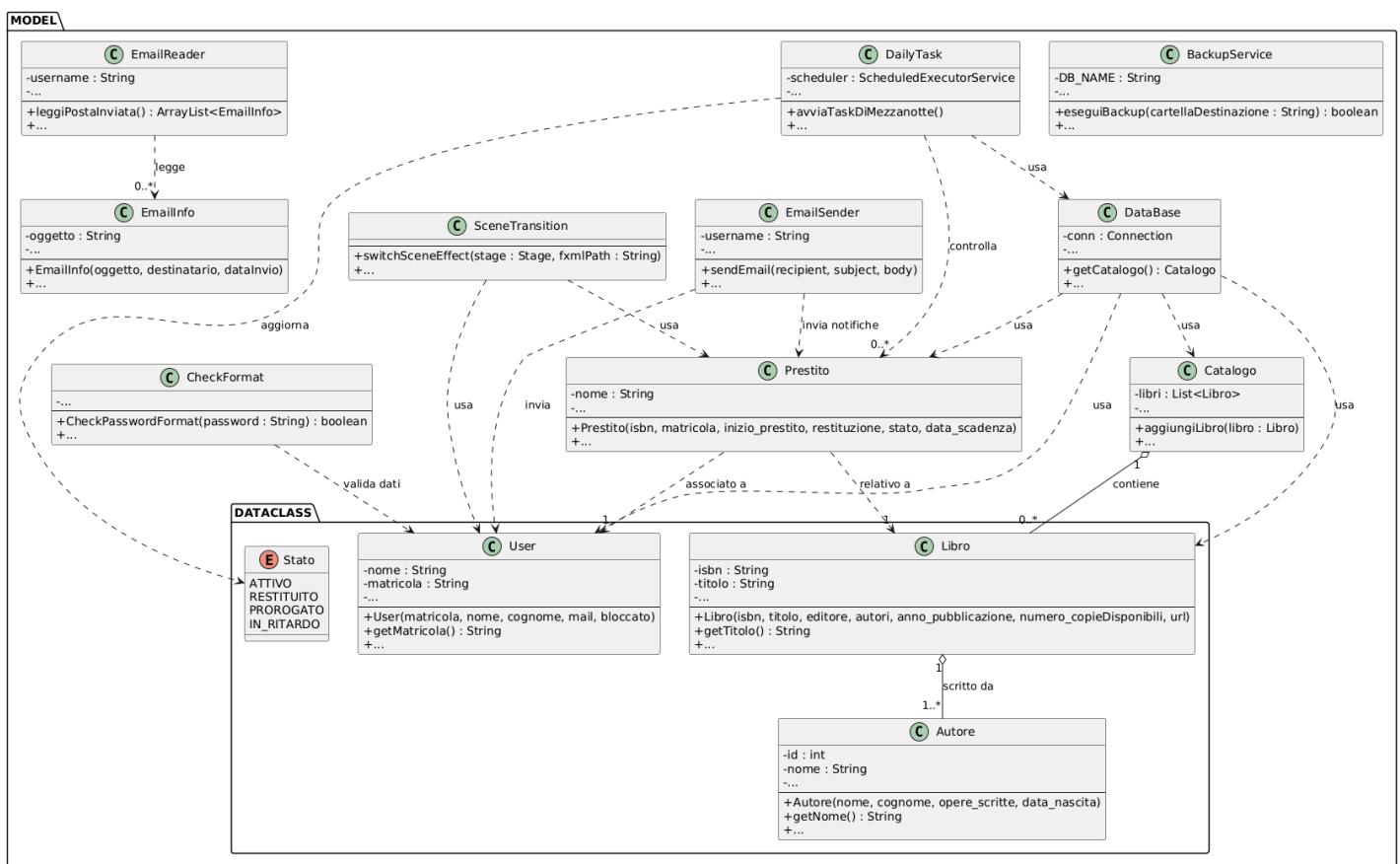
    User
    -nome : String
    -cognome : String
    -matricola : String
    -mail : String
    -bloccato : boolean

    +User(matricola : String, nome : String, cognome : String, mail : String, bloccato : boolean)
    +getNome() : String
    +setNome(nome : String) : void
    +getCognome() : String
    +setCognome(cognome : String) : void
    +getMatricola() : String
    +setMatricola(matricola : String) : void
    +getMail() : String
    +setMail(mail : String) : void
    +isBloccato() : boolean
    +setBloccato(bloccato : boolean) : void
    +toString() : String

```

- *bloccato: boolean* - Stato dell'utente (inserito o meno nella blacklist).
- **Metodi Principali:**
 - *User(String matricola, String nome, String cognome, String mail, boolean bloccato)* - Costruttore.
 - *getNome() : String, setNome(String nome) : void, getCognome() : String, setCognome(String cognome) : void* - Accesso ai dati anagrafici.
 - *getMatricola() : String, setMatricola(String matricolare) : void, isBloccato() : boolean, setBloccato(boolean bloccato) : void, getMail() : String, setMail(String mail) : void* - Accesso ai dati personali.
 - *toString() : String* - Rappresentazione testuale dell'oggetto.
- **Relazioni:**
 - È referenziato da *Prestito*.
 - È contenuto in *BlackList*.

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



PACKAGE CONTROLLER

Nome Classe: ControllerAccess

- **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.
 - *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 bottoni 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.
- **Relazioni:**

 ControllerAccess	
-LoginPane : Pane -RegisterForm : Pane -Sliding : Pane -SlidingButton : Button -SwitchLabel : Label -direction : boolean -ts : TranslateTransition -tr : TranslateTransition -slidingTiming : double -PassRegister : PasswordField -PassRegisterVisible : TextField -PassConRegister : PasswordField -PassConRegisterVisible : TextField -ChekShowPassRegister : CheckBox -PassLogin : PasswordField -PassLoginVisible : TextField -ChekShowPassLogin : 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	

- Dipende da Model, da View (*ControllerAccess* usa *CheckFormat*, *DataBase* e *EmailSender*).

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 aprire Catalogo Libri.
 - *mailButton : Button* - Bottone per aprire gestione email/notifiche.
 - *BLButton : Button* - Bottone per aprire la schermata Blacklist.
 - *DashboardButton : Button* - Bottone della home principale.
 - *utentiButton : Button* - Bottone gestione utenti.
 - *PrestitiRestituzioniButton : Button* - Bottone gestione prestiti e restituzioni.
 - *DashboardBox : VBox* - Contenitore verticale della sidebar.
 - *DashboardScrollPane : ScrollPane* - Scroll della dashboard.
 - *HomeBorderPane : BorderPane* - Contenitore centrale che cambia contenuto (caricamento FXML).
 - *menuButtons : List<Button>* - Lista di tutti i pulsanti del menu per gestire l'evidenziazione.
- **Metodi Principali:**
 - *initialize() : void* - Configura il menu.
 - *ButtonInitialize() : void* - Definisce la logica di navigazione (caricamento FXML) al click dei pulsanti.
 - *evidenziaBottone(Button bottoneAttivo) : void* - Gestisce lo stile grafico del pulsante attivo.
- **Relazioni:**
 - Carica le viste gestite da *CatalogoController*, *MailController*, ecc.

C	DashboardController
-CatalogoLibriButton : Button	
-mailButton : Button	
-BLButton : Button	
-DashboardButton : Button	
-utentiButton : Button	
-PrestitiRestituzioniButton : Button	
-DashboardBox : VBox	
-DashboardScrollPane : ScrollPane	
-HomeBorderPane : BorderPane	
-menuButtons : List<Button>	
+initialize() : void	
+ButtonInitialize() : void	
+evidenziaBottone(b : 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.

C	MailController
-emailContainer : VBox	
-lblTotalUsers : Label	
+initialize() : void	
+caricaEmailInviate() : void	
+aggiungiCardEmail(mail : EmailInfo) : void	

- **Metodi Principali:**
 - *initialize()* - 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 *EmailReader* 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: PasswordChargeController

- **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.
- **Metodi Principali:**
 - *initialize() : void* - Inizializza la view, impostando checkbox e buttoni.
 - *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.
- **Relazioni:**
 - Usa *DataBase* per aggiornare la password del bibliotecario.
 - Usa *CheckFormat* per validare la sicurezza della password.
 - Interagisce con JavaFX (PasswordField, TextField, Button, CheckBox, Label, Alert, DialogPane) per gestire la UI.

 PasswordChargeController
-NewPass : PasswordField
-ConfirmPass : PasswordField
-NewPassVisible : TextField
-ConfirmPassVisible : TextField
-CheckShowPass : CheckBox
-BtnSalva : Button
+initialize() : void
+SetButtonFunction() : void
+SetCheckBox() : void
+ShowPassword(yes : boolean) : void

Nome Classe: NotifyController

- **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.

C	NotifyController
-	numRit : Label
+	initialize() : void

Nome Classe: InserisciPasswordPerModificaController

- **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 button 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 *CheckFormat* per eventuali controlli sul formato della password (se necessario).

C	InserisciPasswordPerModificaController
-	NewPass : PasswordField -NewPassVisible : TextField -CheckShowPass : CheckBox -BtnSalva : Button -BtnAnnulla : Label
+	initialize() : void +SetButtonFunction() : void +SetCheckBox() : void +ShowPassword(yes : boolean) : void

- Interagisce con JavaFX (PasswordField, TextField, CheckBox, Button, Label, Stage, Scene) per gestire l'interfaccia grafica.

Nome Classe: BlacklistController

BlacklistController	
-blacklistContainer : VBox	
-lblTotalBlocked : Label	
-UnLockAllButton : Button	
-searchUser : TextField	
+initialize() : void	
+SearchFunction() : void	
+updateUtentiList(utenti : ArrayList<User>) : 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<User>) : 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 *User* per ottenere la lista degli utenti e filtrare quelli bloccati.
 - Usa *DataBase* per sbloccare utenti e ottenere la lista completa degli utenti.
 - Usa *EmailSender* per inviare avvisi agli utenti.
 - Interagisce con JavaFX (VBox, HBox, Label, Button, TextField, Tooltip, DialogPane) per la gestione dell'interfaccia grafica.

PACKAGE CONTROLLER.CATALOGO

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.
 - *addButton* : *Button* - Bottone per aprire la finestra di inserimento di un nuovo libro.
 - *MAX_BOOKS* : *short* - Costante che definisce il numero massimo di libri consentiti nel sistema (1000).
- **Metodi Principali:**
 - *updateCatalogo()* : *void* - Recupera i libri dal DataBase e popola la griglia.
 - *initialize()* : *void* - Inizializza il catalogo.
 - *creaLibroAnimato(Libro libro)* : *VBox* - Crea dinamicamente un oggetto grafico (*StackPane*) per ogni libro, con effetti 3D al passaggio del mouse e gestione del click.
 - *SearchFunction()* : *void* - Esegue la ricerca di libri per ISBN o titolo e aggiorna la vista del catalogo.
- **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.

C	CatalogoController
-	<i>containerLibri</i> : <i>GridPane</i>
-	<i>LibriScrollPane</i> : <i>ScrollPane</i>
-	<i>btnCerca</i> : <i>Button</i>
-	<i>searchBar</i> : <i>TextField</i>
-	<i>addButton</i> : <i>Button</i>
-	<i>MAX_BOOKS</i> : <i>short</i>
+ <i>initialize()</i>	: <i>void</i>
+ <i>updateCatalogo(libri : Catalogo)</i>	: <i>void</i>
+ <i>creaLibroAnimato(libro : Libro)</i>	: <i>VBox</i>
+ <i>SearchFunction()</i>	: <i>void</i>

Nome Classe: AddBookController

- **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.

C	AddBookController
-	<i>txtTitolo</i> : <i>TextField</i>
-	<i>txtISBN</i> : <i>TextField</i>
-	<i>spinAnno</i> : <i>Spinner<Integer></i>
-	<i>spinCopie</i> : <i>Spinner<Integer></i>
-	<i>menuAutori</i> : <i>MenuButton</i>
-	<i>imgAnteprima</i> : <i>ImageView</i>
-	<i>txtEditore</i> : <i>TextField</i>
-	<i>ScegliFileButton</i> : <i>Button</i>
-	<i>AnnullaButton</i> : <i>Button</i>
-	<i>SalvaButton</i> : <i>Button</i>
-	<i>RimuoviCopButton</i> : <i>Button</i>
-	<i>MAX_AUTORS</i> : <i>short</i> = 1000
-	<i>MAX_WRITED</i> : <i>short</i> = 5000
-	<i>urlIM</i> : <i>String</i>
+ <i>initialize()</i>	: <i>void</i>
+ <i>SettingForm()</i>	: <i>void</i>
+ <i>UpdateAutori()</i>	: <i>void</i>
+ <i>ButtonInitialize()</i>	: <i>void</i>
+ <i>SpinnerInitialize()</i>	: <i>void</i>

- *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* : *short* - Numero massimo di autori consentiti nel sistema (1000).
- *MAX_WRITED* : *short* - 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*.

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.
 - *ISBN* : *String* - Codice ISBN del libro in modifica.
- **Metodi Principali:**
 - *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.

C	ModificaLibroController
-	<i>txtTitolo</i> : <i>TextField</i>
-	<i>menuAutori</i> : <i>MenuButton</i>
-	<i>txtEditore</i> : <i>TextField</i>
-	<i>spinAnno</i> : <i>Spinner<Integer></i>
-	<i>spinCopie</i> : <i>Spinner<Integer></i>
-	<i>imgAnteprima</i> : <i>ImageView</i>
-	<i>ScegliFileButton</i> : <i>Button</i>
-	<i>AnnullaButton</i> : <i>Button</i>
-	<i>SalvaButton</i> : <i>Button</i>
-	<i>RimuoviCopButton</i> : <i>Button</i>
-	<i>urlIM</i> : <i>String</i>
-	<i>isbn</i> : <i>String</i>
+ <i>initialize()</i>	: <i>void</i>
+ <i>SettingForm()</i>	: <i>void</i>
+ <i>ButtonInitialize()</i>	: <i>void</i>
+ <i>UpdateAutori()</i>	: <i>void</i>
+ <i>SpinnerInitialize()</i>	: <i>void</i>

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

C	AggiungiPrestitoController
-txtISBN :	TextField
-txtMatricola :	TextField
-isbnCheck :	Label
-matricolaCheck :	Label
-isbnCheckButton :	Button
-matricolaCheckButton :	Button
-AnnullaButton :	Button
-SalvaButton :	Button
-dateInizio :	DatePicker
-dateScadenza :	DatePicker
-CompletedCheckISBN :	boolean
-CompletedCheckMatricola :	boolean
+initialize() :	void
+InitializeProperty() :	void
+ButtonInitialize() :	void
+ButtonCheckingInitialize() :	void

Nome Classe: PrestitoRestituzioneController

C PrestitoRestituzioneController	
-loansContainer : VBox	
-NewLoanButton : Button	
-FilterButton : MenuButton	
-lblActiveLoans : Label	
+initialize() : void	
+MenuButtonInitialize() : 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.

- **Metodi Principali:**

- *initialize() : void* - Inizializza la vista, aggiorna i prestiti e i filtri.
- *MenuButtonInitialize() : void* - Inizializza il menu di filtraggio per stato.
- *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*, *User* 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.

C AggiungiUtenteController	
-txtMatricola : TextField	
-txtNome : TextField	
-txtCognome : TextField	
-txtEmail : TextField	
-AnnullaButton : Button	
-SalvaButton : Button	
+initialize()	
+ButtonInitialize()	

- *txtEmail* : *TextField* - Email utente.
- *AnnullaButton* : *Button* - Chiude la finestra.
- *SalvaButton* : *Button* - Avvia validazione e salvataggio.
- **Metodi Principali:**
 - *initialize()* : *void* - Inizializza bottoni e listener richiamando *ButtonInitialize()*.
 - *ButtonInitialize()* : *void* - Chiude la finestra ed esegue i controlli.
- **Relazioni:**
 - Dipende da Model e da View.

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 *ModifyUser, CheckFormat*.
 - Interagisce con JavaFX (Label, TextField, Button, Alert, Stage).

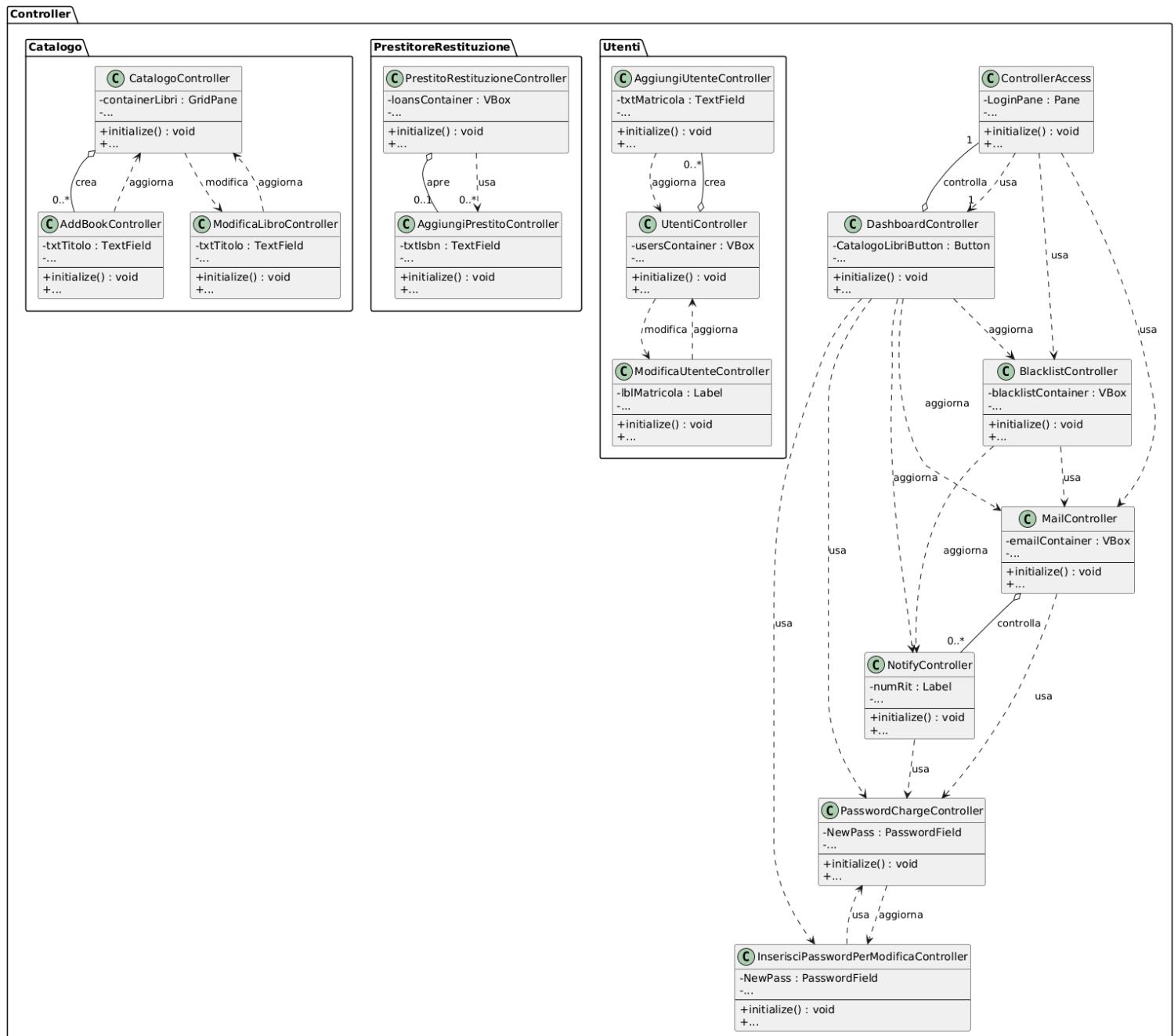
C	ModificaUtenteController
-	<i>lblMatricola</i> : <i>Label</i>
-	<i>txtNome</i> : <i>TextField</i>
-	<i>txtCognome</i> : <i>TextField</i>
-	<i>txtEmail</i> : <i>TextField</i>
-	<i>btnSalva</i> : <i>Button</i>
-	<i>btnAnnulla</i> : <i>Button</i>
+	<i>matricola</i> : <i>String</i> «static»
+	<i>initialize()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

Nome Classe: UtentiController

C	UtentiController
-	<i>usersContainer</i> : <i>VBox</i>
-	<i>btnAddUser</i> : <i>Button</i>
-	<i>lblTotalUsers</i> : <i>Label</i>
-	<i>FilterButton</i> : <i>MenuButton</i>
+	<i>initialize()</i>
+	<i>MenuButtonInitialize()</i>
+	<i>LabelInitialize()</i>
+	<i>ButtonInitialize()</i>
+	<i>updateUtentiList(utenti : ArrayList<User>)</i>
+	<i>aggiungiCardUtente(nome : String, cognome : String, matricola : String, email : String, isBlacklisted : boolean)</i>

- **Responsabilità:** Gestisce la visualizzazione e la gestione degli utenti, inclusi filtri (tutti, attivi, bloccati), aggiunta di nuovi utenti, modifica, blocco/sblocco 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).
- **Metodi Principali:**
 - *initialize()* : *void* - Inizializza componenti, aggiorna lista utenti e imposta pulsanti e menu.
 - *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<User> 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 *User* 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: Access.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 *ControllerAccess*.
 - Utilizza componenti JavaFX (Pane, Button, TextField, PasswordField, CheckBox, Label).
 - Usa il foglio di stile *StyleAccess.css*.

Nome Interfaccia: CatalogoLibri.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: AddPrestito.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: AddUtente.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 *AddBookController*.
 - 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 searchUser*.
 - *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).

- Stile definito nel foglio CSS: StyleUtenti.css.

Nome Interfaccia: Utenti.fxml

- **Responsabilità:** Definisce l'interfaccia grafica per la gestione utenti, inclusa la visualizzazione della lista utenti, la ricerca, laggiunta 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 *NotifyController*.
 - 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 *User* per ottenere informazioni sugli utenti.
 - Usa *DataBase* per sbloccare utenti e recuperare la lista completa.
 - Usa *EmailSender* 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: InserisciPasswordPerModifica.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 bottoni.
 - *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 *InserisciPasswordPerModificaController*.
 - Usa *DataBase* per verificare la password corrente.
 - Usa *CheckFormat* 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: PasswordCharge.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 bottoni.
 - *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".

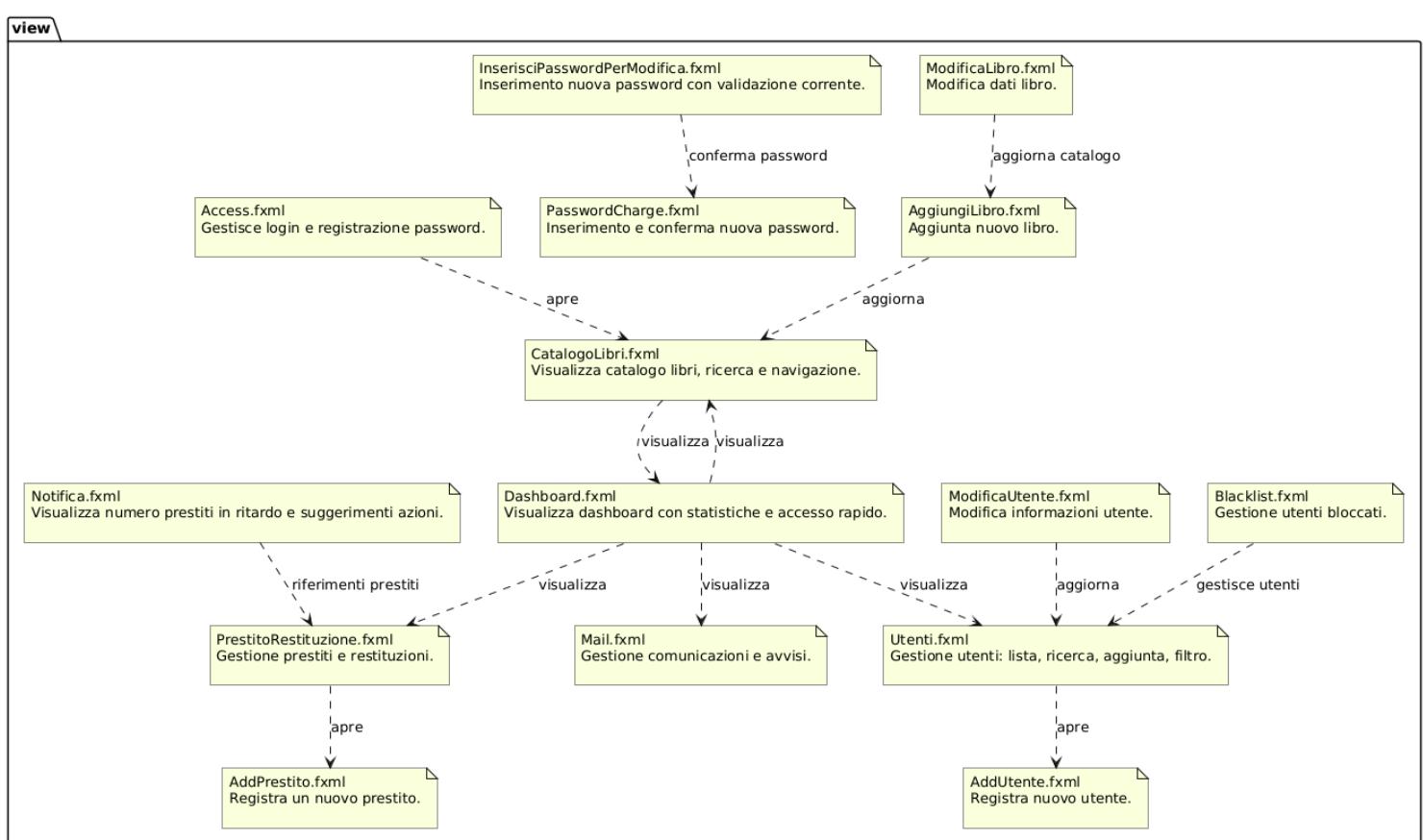
- *PasswordField_ConfirmPass* - Campo password nascosto per confermare la password.
- *TextField_ConfirmPassVisible* - Campo testo visibile per confermare la password (toggle visibilità).
- *CheckBox_ShowPass* - Checkbox per mostrare/nascondere entrambe le password.
- *Button_BtnSalva* - Pulsante per 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 *PasswordChargeController*.
 - Usa *DataBase* per aggiornare la password del bibliotecario.
 - Usa *CheckFormat* 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

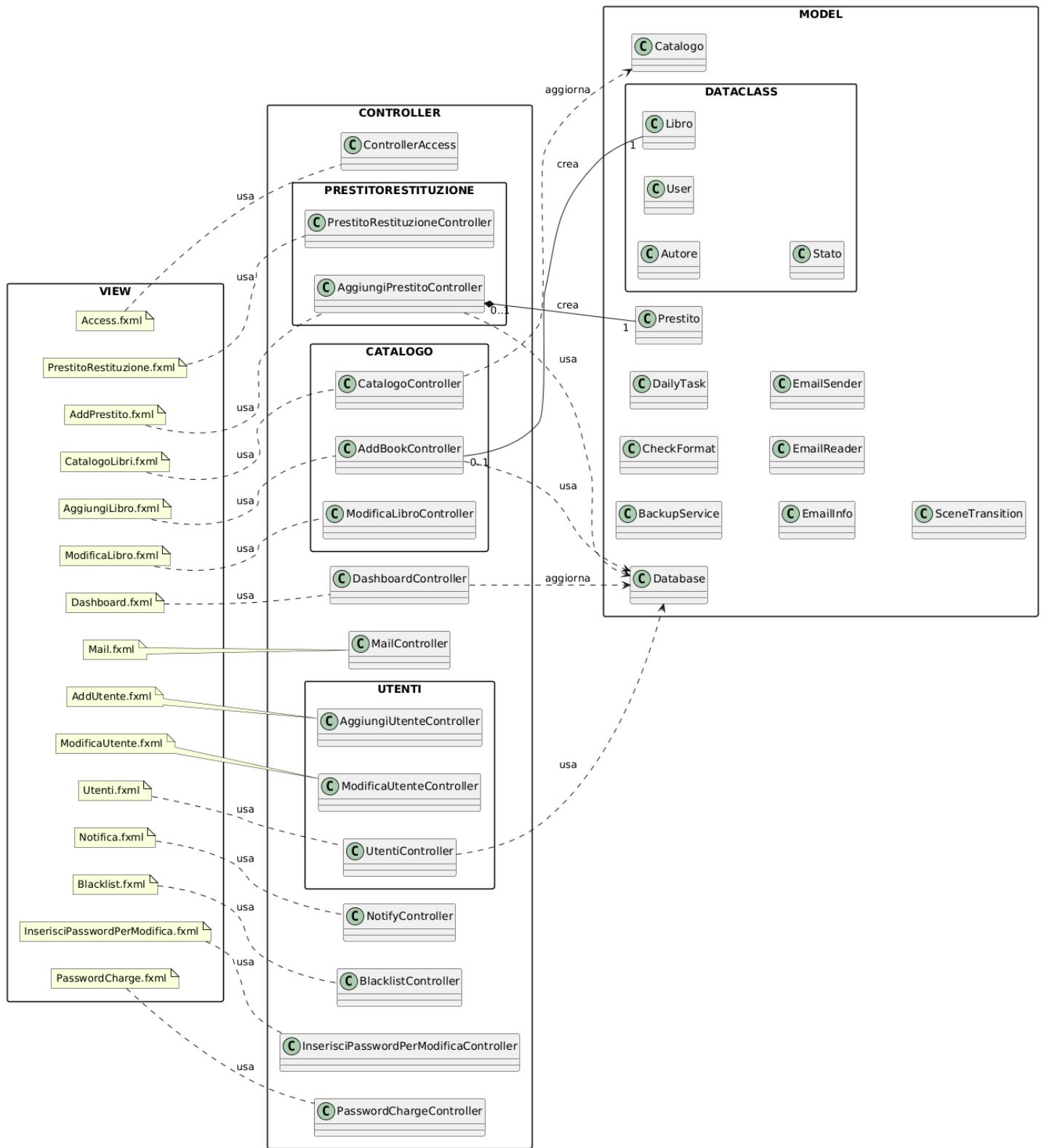
- **Responsabilità:** Gestisce la modifica dei dati di un libro all'interno del catalogo, inclusi titolo, autori, casa editrice, anno di pubblicazione, numero di copie e immagine di copertina. Permette di salvare o annullare le modifiche.
- **Componenti grafici principali:**
 - *StackPane_Principale* - 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.*
- *Annulloamento 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.

- **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 *User*, *UtentiController*, *AggiungiUtenteController* e *ModificaUtenteController* si occupano esclusivamente della gestione degli utenti (inserimento, modifica, ricerca, visualizzazione e gestione blacklist).
 - *User* 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 *AddBookController* 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 *ControllerAccess* e *CheckFormat* si occupano delle operazioni relative al profilo del bibliotecario.
 - *ControllerAccess* si occupa dell'autenticazione, registrazione e controllo del profilo.
 - La classe *CheckFormat* 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 *EmailSender* 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 (es. *CatalogoController*, *PrestitoRestituzioneController* e *UtentiController*, ...) non gestiscono direttamente query SQL, ma si limitano a invocare metodi come *getCatalogo()*, *addPrestito()*, *searchUser()* o *ModifyUser()*. Ciò riduce fortemente la dipendenza tra la logica applicativa e il livello fisico di memorizzazione dei dati.
- **Model:** le entità di dominio (*Libro*, *Autore*, *User*, *Prestito*, *Stato*) contengono solo dati e metodi di accesso (getter/setter), senza dipendere da controller, database o view. Mantenendo zero accoppiamento con gli altri livelli.
- **Servizi esterni:** funzioni come invio email (*EmailSender*) o transizioni grafiche (*SceneTransition*) sono isolate in moduli dedicati, senza introdurre dipendenze circolari.
- **Validazioni:** la classe *CheckFormat* 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.*

- **Model.DataClass:** *Libro, Utente, Prestito* contengono solo dati e metodi strettamente legati alle entità del dominio.
- **Controller:** *CatalogoController, PrestitoRestituzioneController, UtentiController* gestiscono solo l'interazione con la View e il coordinamento tra Model e UI.
- **DataBase:** *DataBase* gestisce esclusivamente la persistenza dei dati.
- **Utility:** *EmailSender* e *SceneTransition* 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 *User* 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. *BackupService* e *SMSNotificationService*).

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 *User* ovunque fossero previsti utenti, mantenendo invariata la logica dei prestiti (*Prestito*) e delle notifiche (*EmailSender*).
- 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 *User* e i metodi di gestione utenti del DataBase.
- Servizi specializzati come *EmailSender* o *SceneTransition* 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*, *EmailSender* ed *EmailReader*). 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.

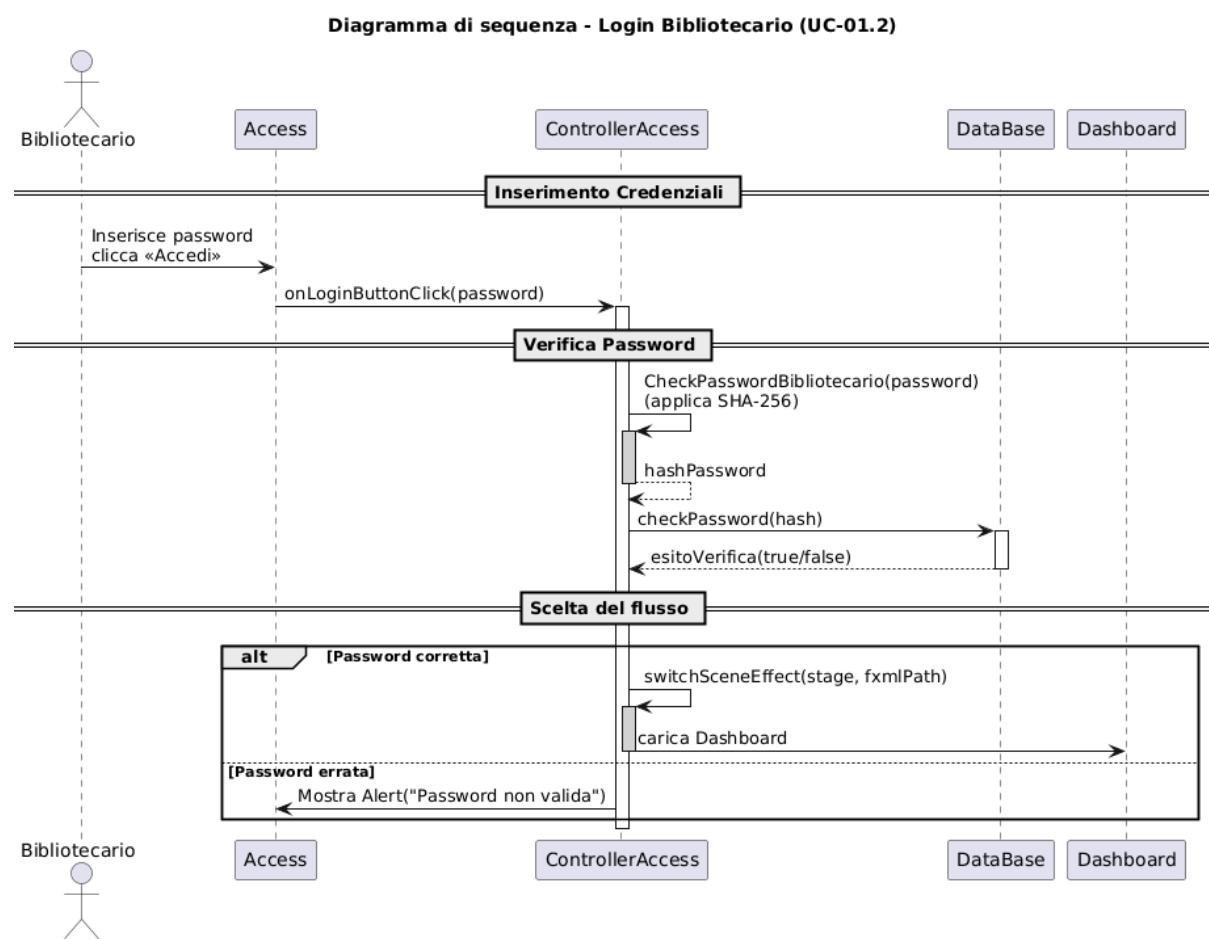
Gestione Accesso e Account

Caso d'uso: Login (UC-01.2)

Il diagramma descrive il processo di autenticazione del bibliotecario.

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

Il *ControllerAccess* 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 *Access.fxml* e preme il pulsante "Accedi".
- L'interfaccia *Access.fxml* invia i dati al *ControllerAccess* attraverso l'handler del bottone.
- Il *ControllerAccess* richiama il metodo *CheckPasswordBibliotecario(String password)*
- Il metodo provvede a:

- applicare l'algoritmo SHA-256 alla password inserita.
 - confrontare l'hash ottenuto con quello memorizzato dal database tramite *CheckPasswordBibliotecario(String password)*.
5. Se la verifica ha successo, il *ControllerAccess* 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 *CheckPasswordBibliotecario(String password)* rileva che l'hash della password non coincide con quello memorizzato.

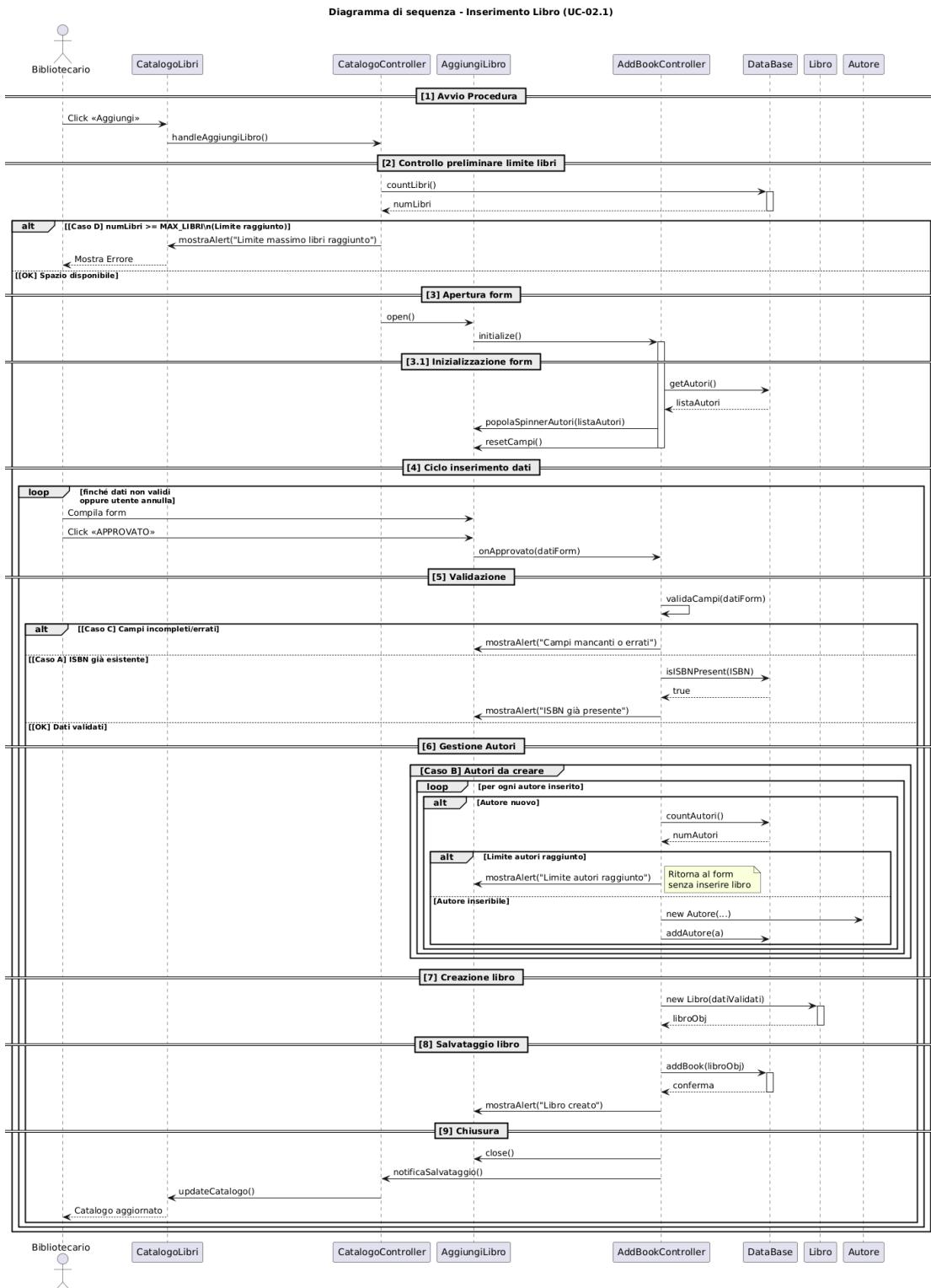
3a.1 Il Controller mostra un messaggio di errore tramite Alert.

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 *CatalogoView*, *CatalogoController* (permette l'apertura del form), *AddBookView*, *AddBookController* (gestisce l'intera logica dell'inserimento) e *DataBase* (gestisce la memorizzazione dei libri e degli autori).



Descrizione dell'interazione:

Flusso Principale:

1. Il bibliotecario si trova nel *CatalogoLibri.fxml* e clicca sul pulsante “Aggiungi” (o icona “+” in fondo al catalogo).
2. L'evento viene intercettato dal *CatalogoController* che apre la *aggiungiLibro.fxml*, che visualizza il form per l'inserimento di un nuovo libro.
3. L'*AddBookController* 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”. La *aggiungiLibro.fxml* invia i dati all'*AddBookController* tramite l'handler del bottone.
5. L'*AddBookController* 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'*AddBookController* chiama la funzione *addAutore(Autore a)* della classe database, verificando prima di inserire ogni autore se questa operazione causa problemi di memoria insufficiente.
7. L'*AddBookController* crea un nuovo oggetto *Libro* con tutti i dati validati.
8. Il controller invia il nuovo oggetto Libro al DataBase tramite *DataBase.addBook(Libro libro)*, che lo registra nel catalogo.
9. Il form viene chiuso e il *CatalogoController* aggiorna l'elenco dei libri mostrati nella *CatalogoLibri.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'*AddBookController* 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.

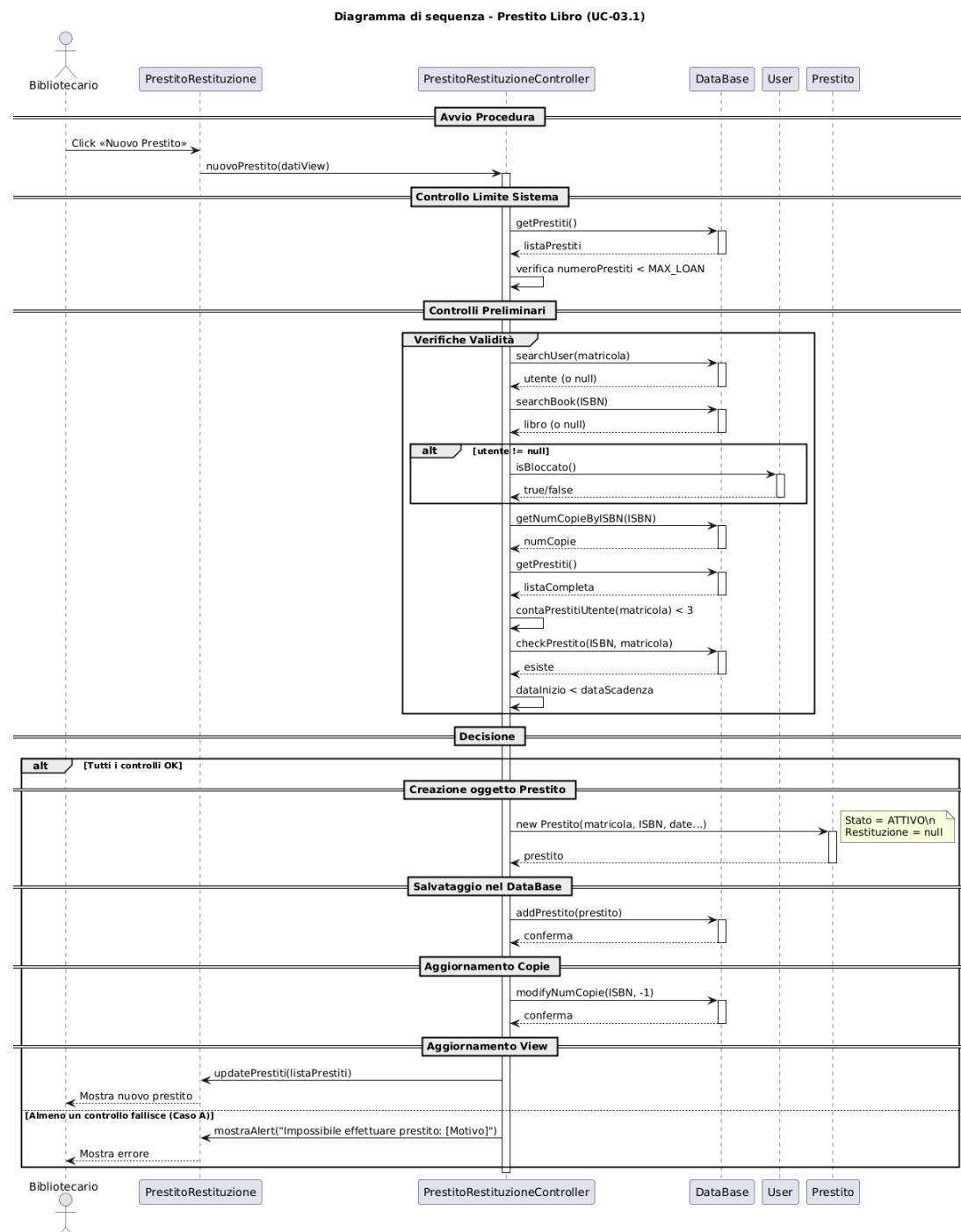
3d.2 Il bibliotecario chiude l'alert e si ritrova al passo 1.

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 *SearchBook(String ISBN)* e *SearchUser(String matricola)*.
 - che l'utente non sia in blacklist tramite il metodo *SearchUser(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 *CheckPrestito(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.addPrestito(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.ModifyNum_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 *ControllerPrestiti* mostra un Alert informando l'operatore dell'impossibilità di effettuare prestiti, specificando il motivo.

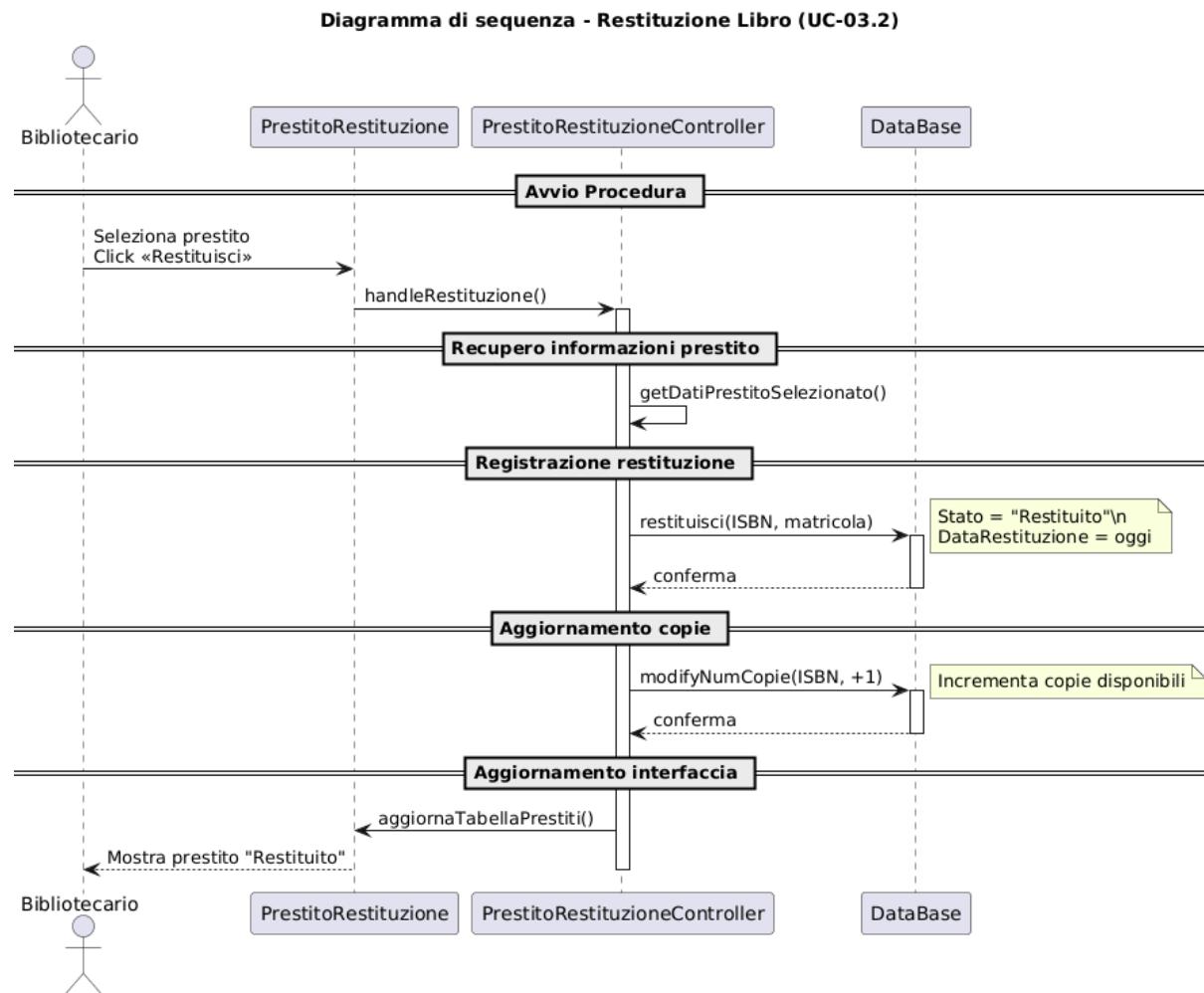
3a.1 Si ritorna al passo 1.

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 *PrestitiView*, *PrestitiController* 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 “Restituisci”.
- L'evento viene intercettato dal *PrestitoRestituzioneController*, che recupera le informazioni del prestito e aggiorna lo stato a “Restituito”, registrando:
 - la data di restituzione (quella attuale al momento del click sul bottone)
 - la chiusura formale del prestito
 tutto implementato dalla funzione *Restituisci(String ISBN, String matricola)*

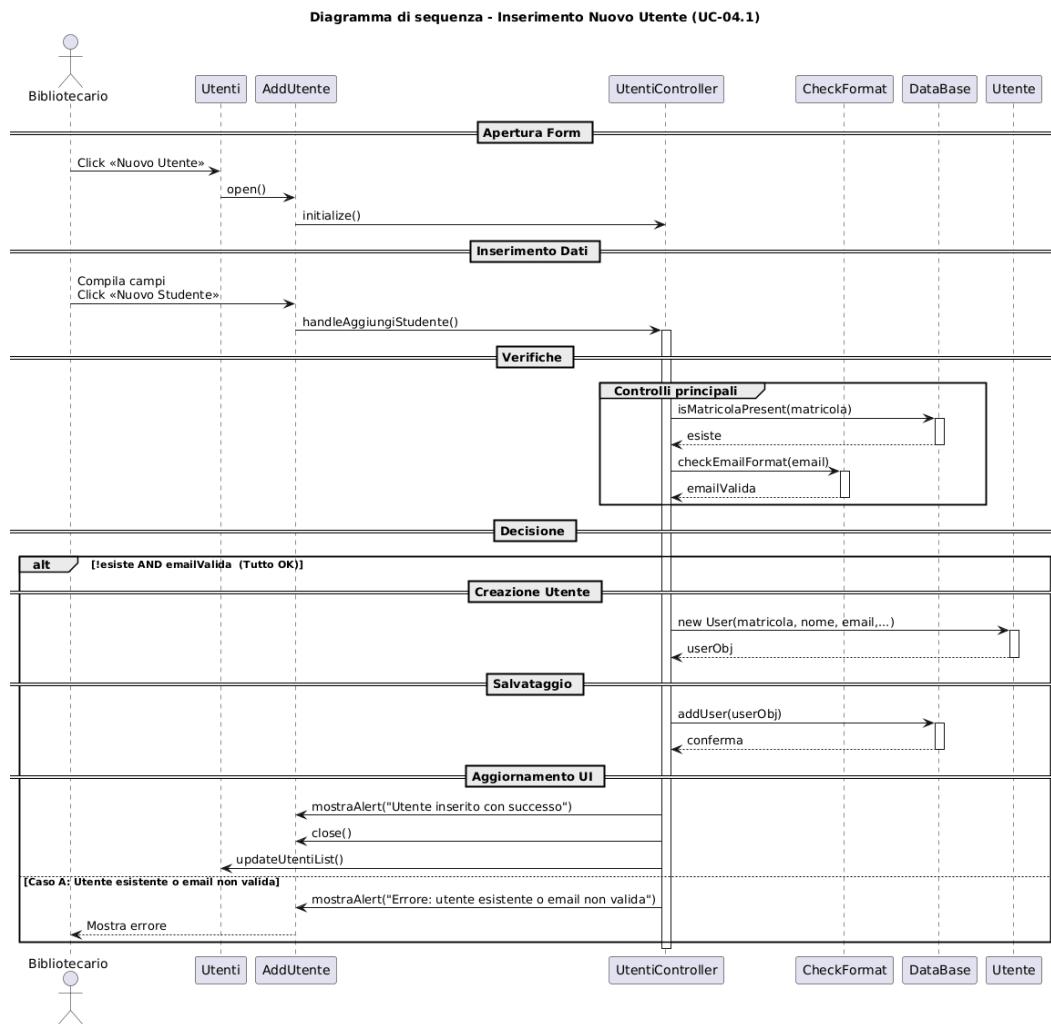
3. Il *PrestitoController* aggiorna la disponibilità del libro associato, incrementando il numero di copie disponibili nel catalogo tramite la funzione *ModifyNum_Copie(String ISBN, boolean add)*.
4. Il Controller invia tutta le modifiche al DataBase, che:
 - salva l'aggiornamento dello stato del prestito
 - 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 *addUtente.fxml*(interfaccia grafica), *addUtenteController* (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:

1. Il bibliotecario clicca il pulsante “Nuovo utente” che apre *addUtente.fxml* che inizializza i campi del form.
2. Il bibliotecario inserisce i campi e clicca sul bottone “Aggiungi Studente”.
3. 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 *CheckEmailFormat(String email)*.
 - Vari controlli (campo matricola non vuoto, matricola a 10 cifre e solo numeri)
4. Se la verifica ha esito positivo, il controller crea l’oggetto *utente* e con *addUser(User u)* lo inserisce nel database.
5. 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.

Lista interfacce:

A) Interfaccia Login:

- Campo password.
- Casella di selezione per mostrare la password.
- Tasto accedi.
- Tasto per passare alla modalità di registrazione.

B) Interfaccia Registrazione:

- Campo password.
- Campo Conferma password.
- Casella di selezione per mostrare la password.
- Tasto Salva e Accedi.
- Tasto per passare alla modalità di login.

C) Interfaccia dashboard:

- Informazioni generali sui dati del sistema.

- Bottoni per passare alle sezioni successive dell'app.
- Bottoni per logout, modifica password, Backup dei dati.

D) Interfaccia Catalogo:

- Card dei libri, con pulsanti per aggiungere/rimuovere copie, modificare o eliminare il libro selezionato.
- Tasto aggiungi libro.
- Barra di ricerca.
- Bottone Cerca.

D.1) Interfaccia aggiungi libro:

- Campi da compilare per aggiungere il libro (Campi di testo, bottone per allegare l'immagine)
- Tasto per aggiungere il libro

E) Interfaccia Prestiti:

- Card dei prestiti, con tasti per gestire restituzione, proroga, mandare avvisi.
- Tasto per filtrare i prestiti in base allo stato.
- Tasto per registrare un nuovo prestito.
- Barra di Ricerca.

E.1) Interfaccia regista prestito:

- Campi per inserire ISBN e matricola.
- Bottoni per verificare l'esistenza di ISBN e matricola nel sistema.
- Campi per impostare data inizio prestito e la scadenza.
- Bottoni per confermare o annullare le operazioni.

F) Interfaccia Utenti:

- Card degli utenti, con tasti per modificare, eliminare, bloccare, sbloccare.
- Bottone per filtrare gli utenti in base allo stato in BlackList.
- Bottone per aggiungere un nuovo utente.
- Barra di ricerca.

F.1) Interfaccia aggiungi utente

- Campi dell'utente da compilare.
- Bottoni per confermare o annullare.

F.2) Interfaccia aggiungi utente

- Campi dell'utente da compilare
- Bottoni per confermare o annullare

G) Interfaccia BlackList:

- Card degli utenti bloccati (presenti nella blacklist), tasto per sbloccarli e mandare avviso.
- Bottone per sbloccare tutti gli utenti.
- Barra di ricerca.

H) Interfaccia E-mail:

- Lista delle e-mail inviate dall'utente.

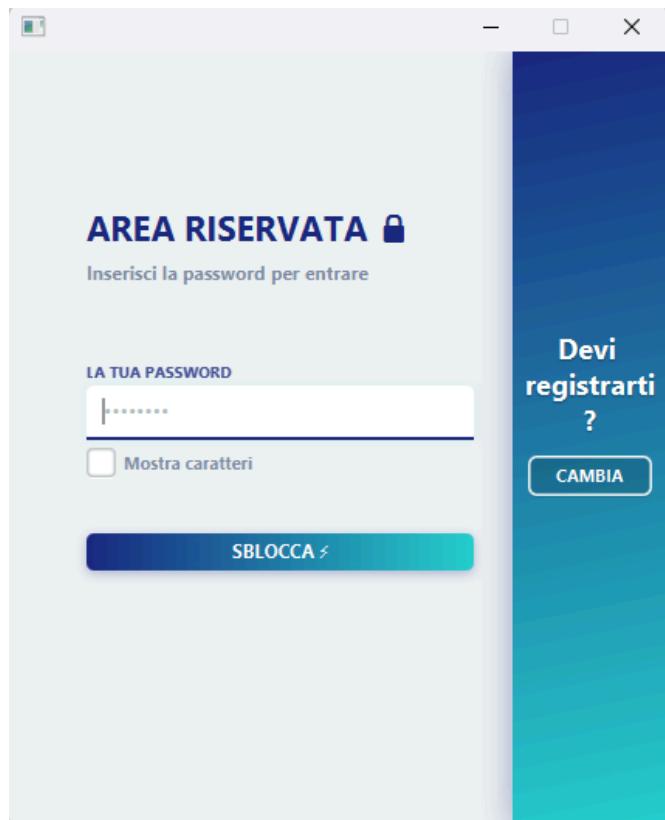
Mock-up:

LOGIN

Il bibliotecario accede all'interfaccia principale inserendo la password e cliccando su "SBLOCCA". In caso di password non corretta viene visualizzata una notifica di errore.

Il bibliotecario può spuntare la casella "Mostra caratteri" per visualizzare la password trascritta.

Inoltre, se non è registrato può cliccare su "CAMBIA" per passare al form di registrazione.



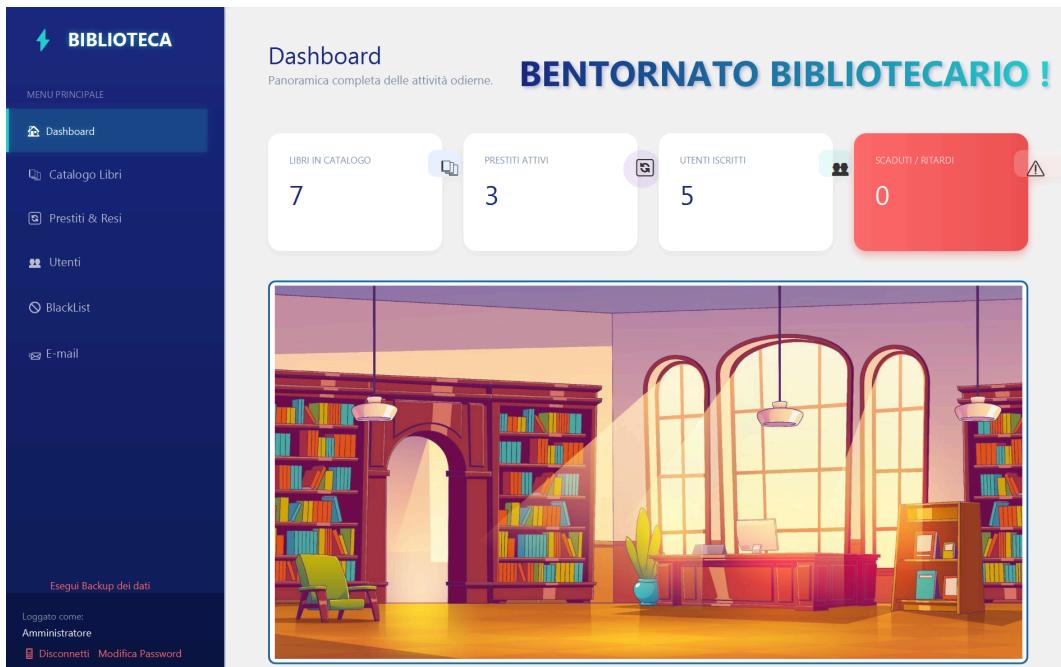
REGISTRAZIONE

Il bibliotecario (non ancora registrato) può registrarsi nel sistema inserendo la password e confermandola. Cliccando sul pulsante "Mostra caratteri" è possibile visualizzare la password inserita. Cliccando su "SALVA E ACCEDI" l'account viene registrato nel database e il bibliotecario accede all'interfaccia principale. Cliccando sul pulsante "Accedi" è invece possibile tornare al form di Login.



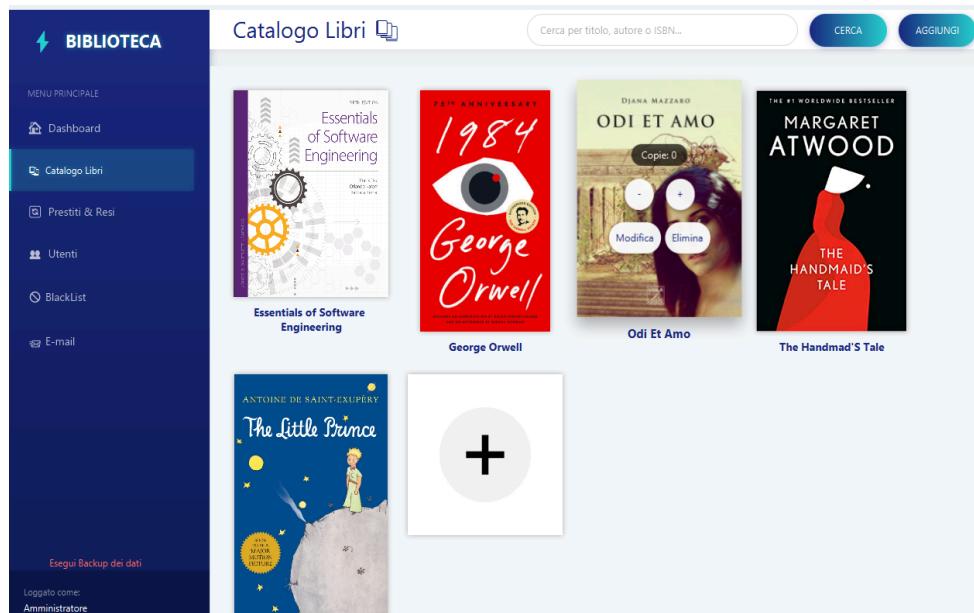
DASHBOARD

Nella Dashboard principale si possono effettuare backup dei dati, modificare credenziali, fare logout e sulla sinistra è presente un menù che permette di navigare tra le varie sezioni del sistema. Inoltre, si visualizza un riepilogo generale sui dati del sistema.



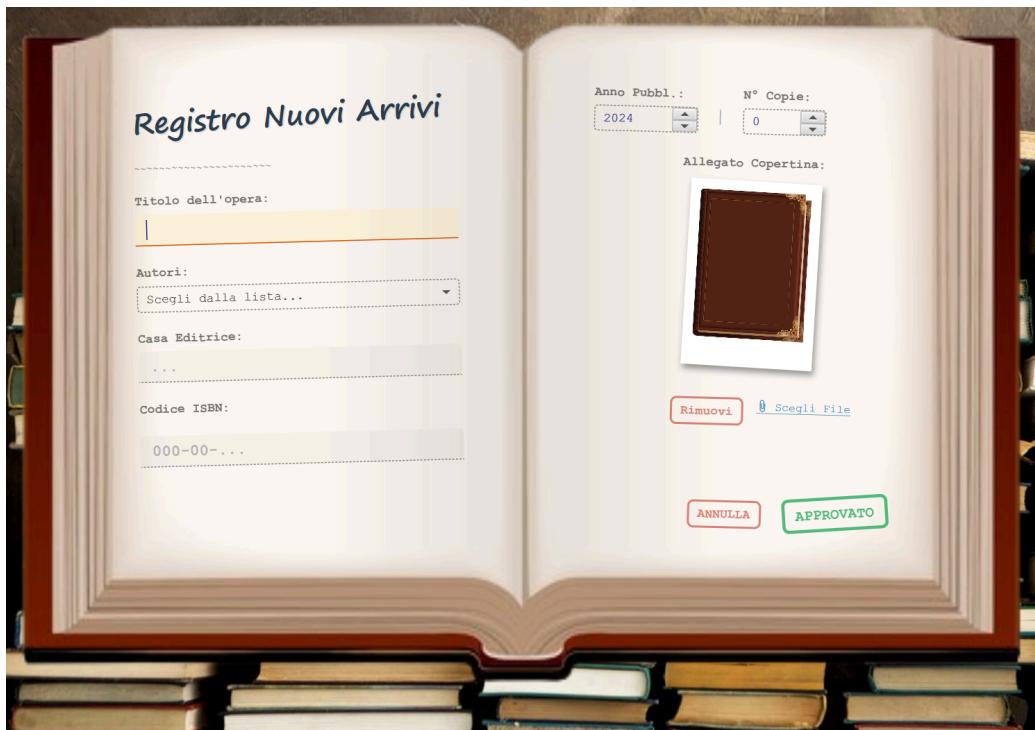
CATALOGO

Il bibliotecario può visualizzare il catalogo con i libri presenti nel sistema (provvisti di copertina se opportunamente inserita e il titolo), può cercare o aggiungere un libro con i rispettivi tasti messi a disposizione. Spostandosi con il mouse su un libro può aggiungere/rimuovere copie, eliminare il libro dal catalogo o modificare i suoi dati (tranne ISBN).



AGGIUNGI LIBRO

Contiene il form dove l'utente, alla fine della compilazione, può registrare il libro nel sistema.



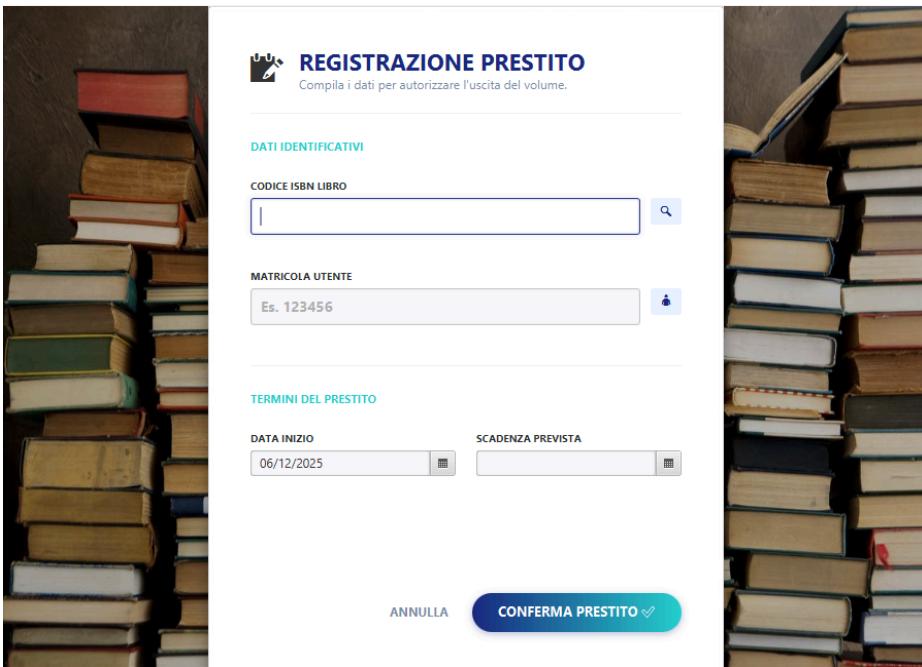
PRESTITI DASHBOARD

Mostra tutti i prestiti registrati nella biblioteca, un pulsante per registrare un nuovo prestito, una barra di ricerca per filtrare, un altro pulsante per cambiare lo stato del prestito in RESTITUITO. In base allo stato compaiono delle icone a schermo: se il prestito è attivo o in ritardo. Sono presenti: una clessidra che una volta cliccata proroga il prestito e un'icona di posta per inviare un avviso all'utente in caso di prestito in ritardo.

Titolo	Autore	ISBN	Prestato da	Scadenza	Status	Azione
George Orwell	Nicola (1234567890)	1234567890123	In Corso	2025-12-17	Restituito	
The Handmad'S Tale	Lucia (1234567892)	1234567890124	In Corso	2026-01-08	Restituito	
The Little Prince	Lucia (1234567893)	1234567890125	In Corso	2026-01-08	Restituito	

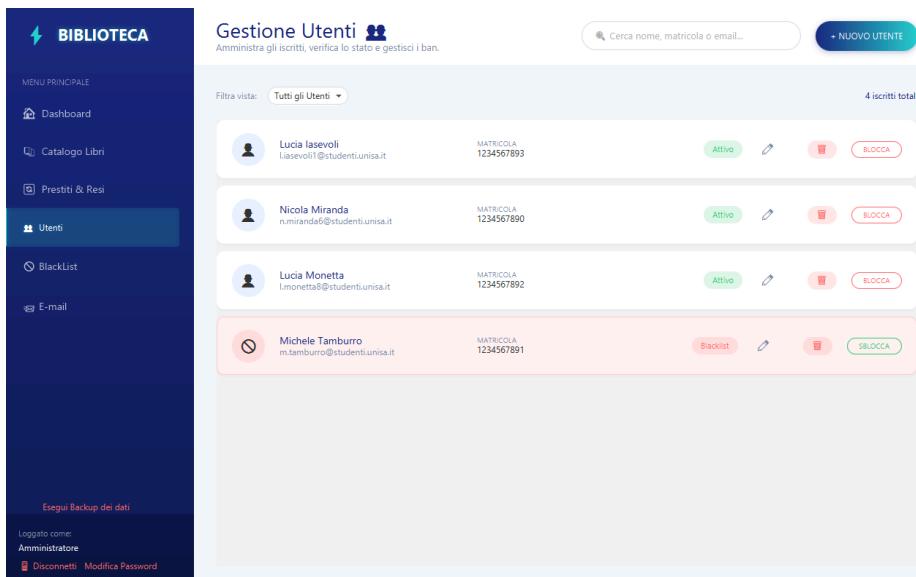
AGGIUNGERE UN PRESTITO

Permette al bibliotecario di registrare un nuovo prestito compilando gli appositi campi. I pulsanti presenti ai lati dei campi di matricola e ISBN servono per verificare la presenza dei codici nel sistema, mentre gli altri due campi sono datepicker.



UTENTI DASHBOARD

Contiene e mostra l'elenco degli studenti presenti nel sistema. È presente un pulsante per aggiungere un nuovo studente. Gli studenti si possono filtrare sia tramite barra di ricerca che tramite il menù di filtri sulla sinistra. Ogni riga mostra le informazioni dello studente e ha diverse icone: la matita per modificare i dati, il cestino per eliminare uno studente dal sistema e un tasto “Blocca”/“Sblocca” (tasto scorciatoia) per inserirlo/rimuoverlo dalla BlackList. Si possono bloccare gli utenti da questo pannello, mentre il pannello della BlackList permette solo di sbloccarli.



AGGIUNGERE UN UTENTE

Presenta un form molto semplice di dati da compilare con le informazioni dello studente, con pulsanti per confermare l'operazione o annullarla.

BIBLIOTECA UNIVERSITARIA
NUOVO STUDENTE

MATRICOLA (ID UTENTE)

NOME: MARIO COGNOME: ROSSI

E-MAIL ISTITUZIONALE: mario.rossi@studenti.it

ANNULLA AGGIungi STUDENTE

MODIFICARE UN UTENTE

Interfaccia per cambiare i dati dell'utente, contiene pulsanti per salvare le modifiche o annullare le operazioni.

MODIFICA UTENTE

MATRICOLA: 1234576890

NOME

COGNOME: Rossi

E-MAIL ISTITUZIONALE: mario.rossi@studenti.it

ANNULLA SALVA MODIFICHE

BLACKLIST

Pannello che elenca gli utenti presenti nella BlackList. È possibile sbloccare tutti gli utenti con un tasto o cercarne uno in particolare tramite la barra di ricerca integrata.

Le operazioni possibili su ogni riga sono: rimuovere un utente dalla BlackList utilizzando il pulsante “Sblocca” (è possibile farlo da scorciatoia anche nella sezione Utenti) e inviare un'email cliccando sull'icona della posta (per informare l'utente che non ha restituito una o più copie di proprietà della biblioteca entro la data di scadenza).

The screenshot shows the 'Blacklist Utenti' page. On the left is a sidebar with a blue header 'BIBLIOTECA' containing links: Dashboard, Catalogo Libri, Prestiti & Resi, Utenti, BlackList (which is highlighted in blue), and E-mail. Below the sidebar are buttons for 'Esegui Backup dei dati', 'Loggato come: Amministratore', and 'Disconnetti Modifica Password'. The main area has a light gray background with a white header bar containing a search input 'Cerca matricola bloccata...' and a red button 'SBLOCCA TUTTI' with a trash icon. Below the header, a message says 'Utenti attualmente sospesi: 1 Utenti Bloccati'. A single user entry is listed in a card: Michele Tamburro (matricola 1234567891), with icons for 'Blacklist' (red), 'Email' (blue), and 'Sblocca' (green).

E-MAIL

Interfaccia che mostra gli ultimi 40 avvisi inviati, utile per controllare quali avvisi non sono stati inviati da diverso tempo.

The screenshot shows the 'Comunicazioni Ritardi Prestiti' page. The sidebar is identical to the one in the previous screenshot. The main area has a light gray background with a white header bar containing the title 'Comunicazioni Ritardi Prestiti' and a trash icon. To the right of the header is a small text '11 Email Inviate'. Below the header, a list of 11 email notifications is shown in a scrollable area. Each notification includes a small profile icon, the subject of the email ('Mancata Restituzione del/dei libro/i'), the recipient ('A: m.tamburro@studenti.unisa.it' or similar), and the date and time it was sent ('INVITATA IL 06/12/2025 18:36').

NOTIFICA

Pop-up che suggerisce di inviare avvisi per utenti inadempienti, compare sullo schermo allo scattare della mezzanotte della data di scadenza del prestito o quando si accede al sistema.

AZIONE SUGGERITA

Ci sono 2 prestiti scaduti dove non sono state restituite le copie, si suggerisci di inviare avvisi agli interessati