Documento di progettazione

Indice:

1. Architettura del sistema

1.1. Moduli principali e responsabilità

il sistema è basato su un'architettura a tre strati logici, per permettere la manutenibilità e una separazione dei compiti.
i tre moduli principali sono:

Modulo	Package Java	Responsabilità	Dipendenze
Presentazione	com.mycompany.bibliotecainds.controller	Gestisce l'interfaccia utente e la visualizzazione dei dati e la navigazione.	le classi di questo package sono dipendenti dalle classi di com.mycompany.bibliotecainds.service.
Logica di Business	com.mycompany.bibliotecainds.service	Implementa le regole di Business e i vincoli del sistema.	le classi di questo package sono dipendenti dalle classi di com.mycompany.bibliotecainds.data e com.mycompany.bibliotecainds.model.
Accesso ai dati	com.mycompany.bibliotecainds.data	Gestisce la Persistenza su file tramite Serializzazione.	le classi di questo package sono dipendenti dalle classi di com.mycompany.bibliotecainds.model.

1.2. Pattern architetturali:

per questo progetto abbiamo utilizzato diversi pattern di progettazione:

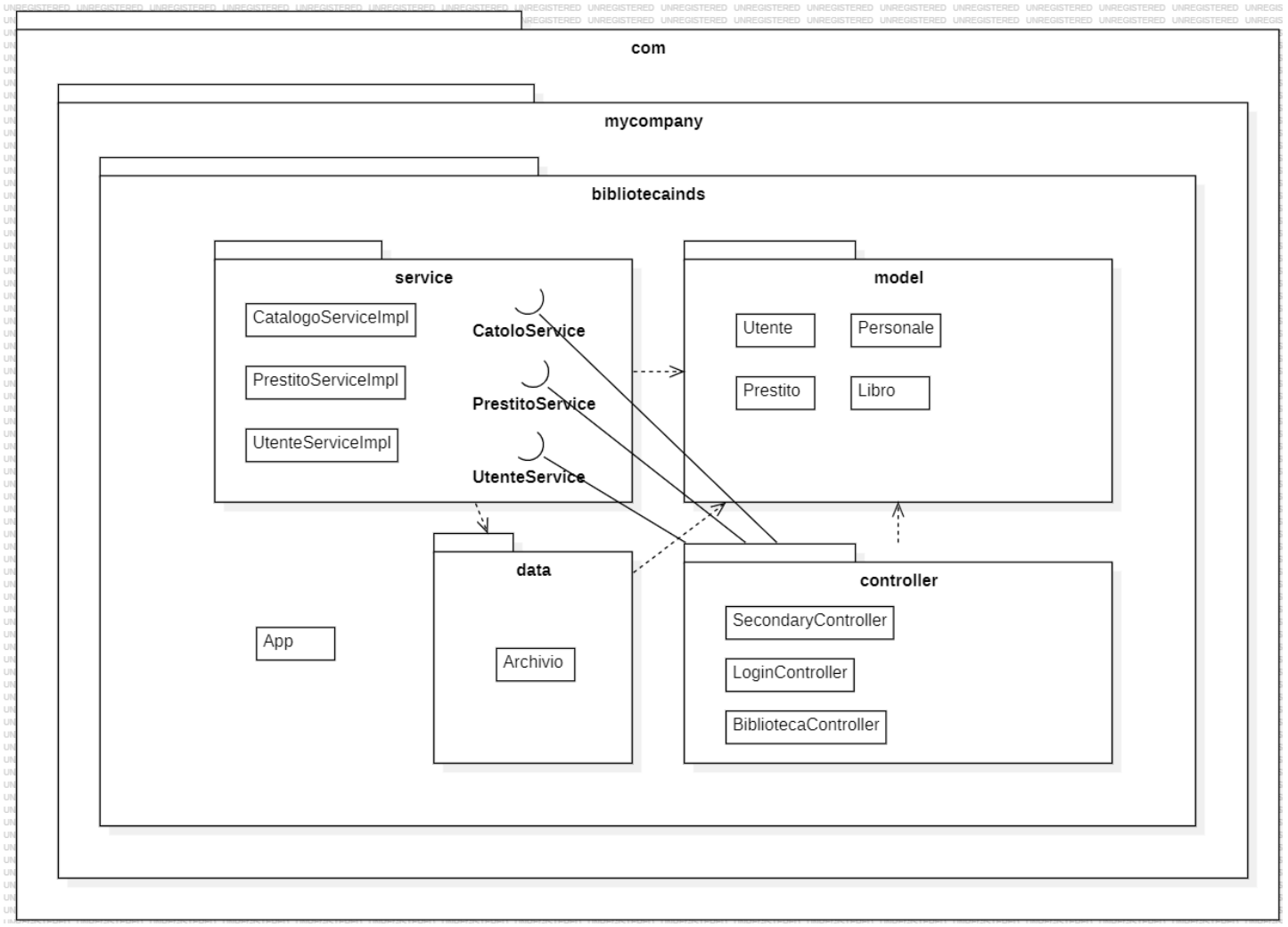
- **MVC (Model-view-Controller):**
  - Model: le classi che rappresentano i dati e le regole di business fondamentali (LIbro, Utente, Personale, Prestito)
  - View: i file FXML (login.fxml, primary.fxml, secondary.fxml)
  - Controller: le classi LoginController, BibliotecaController, SecondaryController, che gestiscono gli eventi utente e l'interazione con i servizi.
- **Singleton**: Implementato nella classe Archivio che garantisce l'esistenza di una e una sola istanza centrale che contiene tutto lo stato del sistema (catalogoLibri, registroUtenti, prestitiAttivi), unica accesso ai dati.
- **Service Layer:** Le funzionalità di Business Logic sono definite da interfacce (*Service) e implementate da classi concrete (*ServiceImpl), promuovendo il disaccoppiamento.

1.3. Commenti sulle scelte progettuali:

la progettazione del sistema adotta un'architettura multi-strato chiara, che garantisce al meglio i requisiti di manutenibilità.

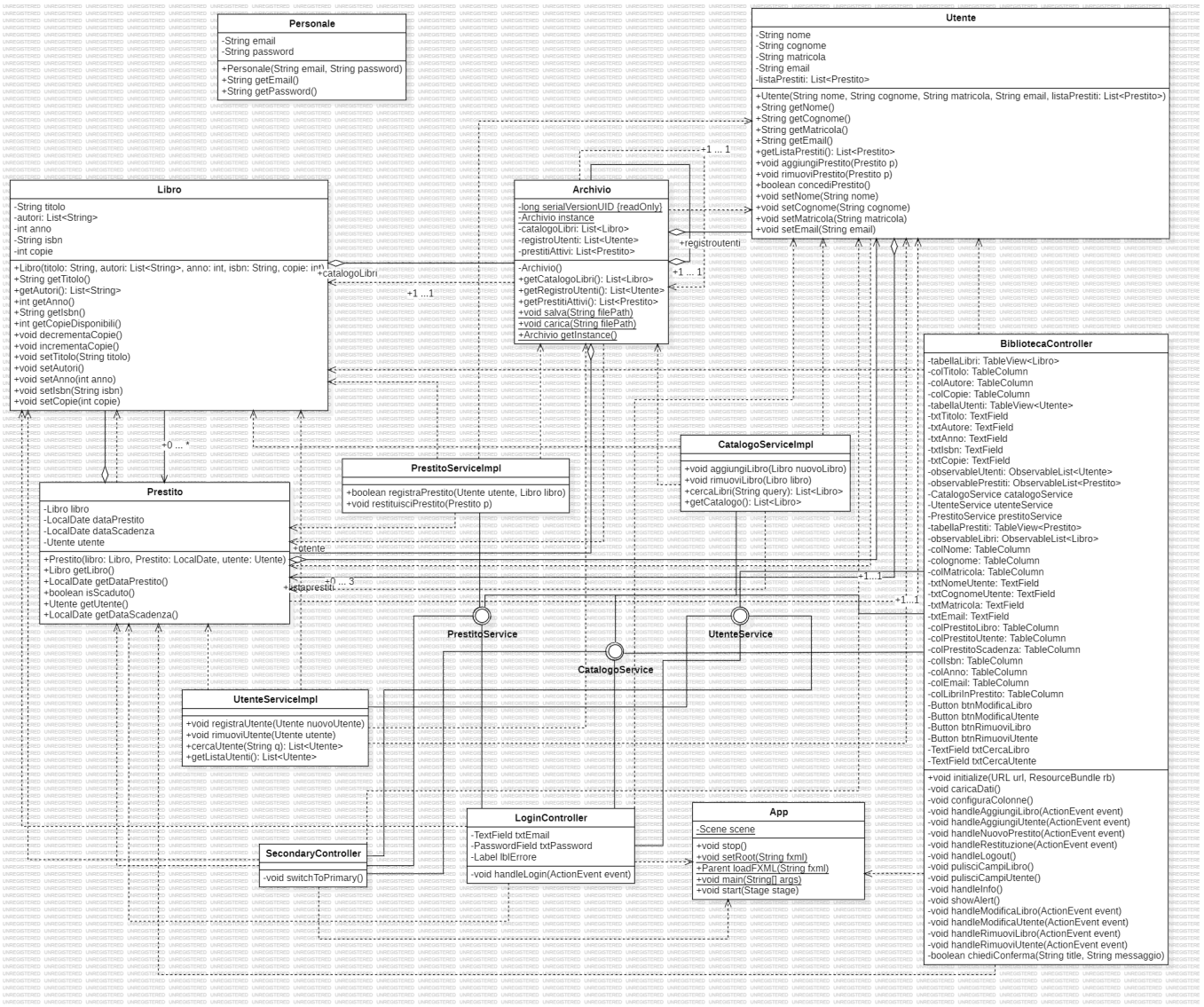
- **Basso accoppiamento:**
  - **Inversione di dipendenza (DIP):** Lo stato di presentazione (BibliotecaController) non dipende dalle implementazioni della logica di business (tutti i *ServiceImpl), ma solo dalle interfacce astratte (CatalogoService, PrestitoService, UtenteService). Facendo questo il Controller non dipende dai cambiamenti della logica interna e si migliora la riutilizzabilità del codice.
  - **Separazione della persistenza:** Il solo strato di Service dipende dalla classe Archivio. Essa ha una singola responsabilità (gestione I/O su file), in modo tale da isolare completamente il Controller e il Modello dal meccanismo di salvataggio.
  - **Iniezione di Dipendenza (Implicita):** Le dipendenze tra i vari strati vengono gestite tramite Iniezioni del Costruttore, assicurando che le vari classi non creino direttamente le loro dipendenze.
- **Elevata Coesione:**
  - **Coesione dello Strato Service:** Ogni implementazione Service (*ServiceImpl) incapsula la totalità della logica di business per il suo dominio (ad es. PrestitoServiceImpl gestisce tutti i controlli di vincolo per la registrazione di un prestito).
  - **Coesione nel Modello:** Le classi di dominio (Libro, Utente, Prestito) contengono sia i dati che i metodi essenziali per la gestione del loro stato interno.
  - **Coesione dello Strato Dati:** La classe Archivio è coesa nella sua singola funzione di gestire le collezioni di oggetti e l'I/O su file, senza coinvolgere la logica di business.
- **Principi di buona Progettazione e Astrazione:**
  - **Uso di Astrazione (Interfacce):** L'utilizzo diffuso di interfacce Service è il principale meccanismo di astrazione per migliorare la riutilizzabilità. Il Service Layer è l'unico punto in cui il Controller deve interagire per eseguire le funzionalità.
  - **Gestione delle eccezioni:** I diagrammi di sequenza illustrano una gestione formale delle eccezioni (es. throw CopieNonDisponibiliException). Esse sono gestite dal Service e catturate dal Controller per fornire feedback visivo (mostraErrore()).
  - **Assocazioni con vincoli:** La relazione tra Utente e Prestito è formalizzata nel diagramma delle classi con una molteplicità 0…3. Questo vincolo di business è integrato nel design e viene verificato dalla logica nel Service Layer (metodo condediPrestito()).

1.4. Diagramma dei package



2. Modello statico

2.1. Diagramma delle classi



2.2. Descrizione delle classi

2.2.1. Package com.mycompany.bibliotecainds.model:

Classe	Attributi	Metodi e Logica di business
Libro	private String titolo, private String isbn, private int copie, private List<String> autori, private int anno	decrementaCopie()/incrementaCopie(): definiscono la disponibilità fisica dei libri del catalogo, utilizzati da PrestitoServiceImpl per aggiornare le copie in seguito di resituzioni/prestiti.
Utente	private String nome, private String cognome,	concediPrestito(): Ritorna true se listaPrestiti.size() < 3.

	private String matricola, private String email, private List<Prestito> listaPrestiti	aggiungiPrestito(p) / rimuoviPrestito(p): Gestione della lista prestiti interna.
Prestito	private Libro libro, private Utente utente, private LocalDate dataPrestito, private LocalDate dataScadenza	isScaduto(): Ritorna true se LocalDate.now().isAfter(dataScadenza). La dataScadenza è impostata a 30 giorni dopo la dataPrestito.
Personale	private String email, private String password	

2.2.2. Package com.mycompany.bibliotecainds.data:

Classe	**Attributi**	**Metodi e Logica di business**
Archivio	private static final long serialVersionUID private List<Libro> catalogoLibri private List <Utente> registroUtenti private static Archivio instance	getIstance(): Restituisce l'unica istanza dell'archivio. Se l'istanza ancora non esiste essa viene creata salva(String filePath): Salva l'intero archivio su file tramite serializzazione carica(String filePath): Carica l'intero archivio da un file se esiste

2.2.3. Package com.mycompany.bibliotecainds.service:

Interfaccia/classe	**Metodi e Logica di business**
UtenteService/UtenteServiceImpl	registraUtente(Utente nuovoUtente): Registra un nuovo utente nel sistema. Controlla che matricola ed email non siano già associate ad altri utenti. In caso contrario il sistema blocca l'operazione e avvisa tramite un errore. rimuoviUtente(Utente utente): rimuove un utente registrato. Un utente non può essere eliminato se ha prestiti ancora attivi. cercaUtente(String q): cerca utenti tramite testo libero. La query viene confrontata con nome, cognome o matricola. Se la stringa è vuota o nulla, viene restituita l'intera lista degli utenti.

PrestitoService/PrestitoServiceImpl	registraPrestito(Utente utente, Libro libro): Registra un nuovo prestito. Controlla la disponibilità del libro e se l'utente può prendere nuovi prestiti. Se si, crea un nuovo oggetto Prestito, aggiorna copie disponibili e registra il prestito nell'archivio centrale. restituisciPrestito(Prestito p): Registra la restituzione di un prestito. Ripristina una copia disponibile del libro e rimuove il prestito sia dall'utente che dell'archivio dei prestiti attivi
CatalogoService/CatalogoServiceImpl	aggiungiLibro(Libro nuovoLibro): Aggiunge un nuovo libro al catalogo. Verifica che non esista già un libro con lo stesso codice ISBN. rimuoviLibro(Libro libro): Rimuove un libro dal catalogo. cercaLibri(String query): Cerca libri nel catalogo in base a una stringa. La ricerca avviene su: Titolo, Nome degli autori, Codice ISBN. Se la query è vuota o nulla, restituisce l'intero catalogo.

2.3. Scelte progettuali:

- **Organizzazione della Logica:**
  - **Service Layer con disaccoppiamento:** la logica di business è separata dai Controller tramite l'utilizzo delle interfacce e delle classi che le implementano, queste creano il service layer, questo per garantire un alto grado di disaccoppiamento che permetterà in futuro di sostituire l'implementazione della logica senza alterare il codice dell'interfaccia utente.
  - **Pattern Singleton per la persistenza:** grazie alla classe archivio viene implementato il pattern Singleton. esiste una sola istanza globale che accede e modifica i dati critici del sistema e non permette che diverse parti dell'applicazione modifichino i dati.
- **Gestione dei dati:**
  - **Persistenza tramite Serializzazione:** tutte le classi del package model implementano l'interfaccia java.io.Serializable, questo perchè l'intera istanza Singleton dell'Archivio viene salvata e caricata in blocco da un unico file binario (archivio.dat) tramite i metodi Archivio.salva() e Archivio.carica(), che vengono eseguiti rispettivamente all'arresto (App.stop()) e all'avvio (App.start()) dell'applicazione

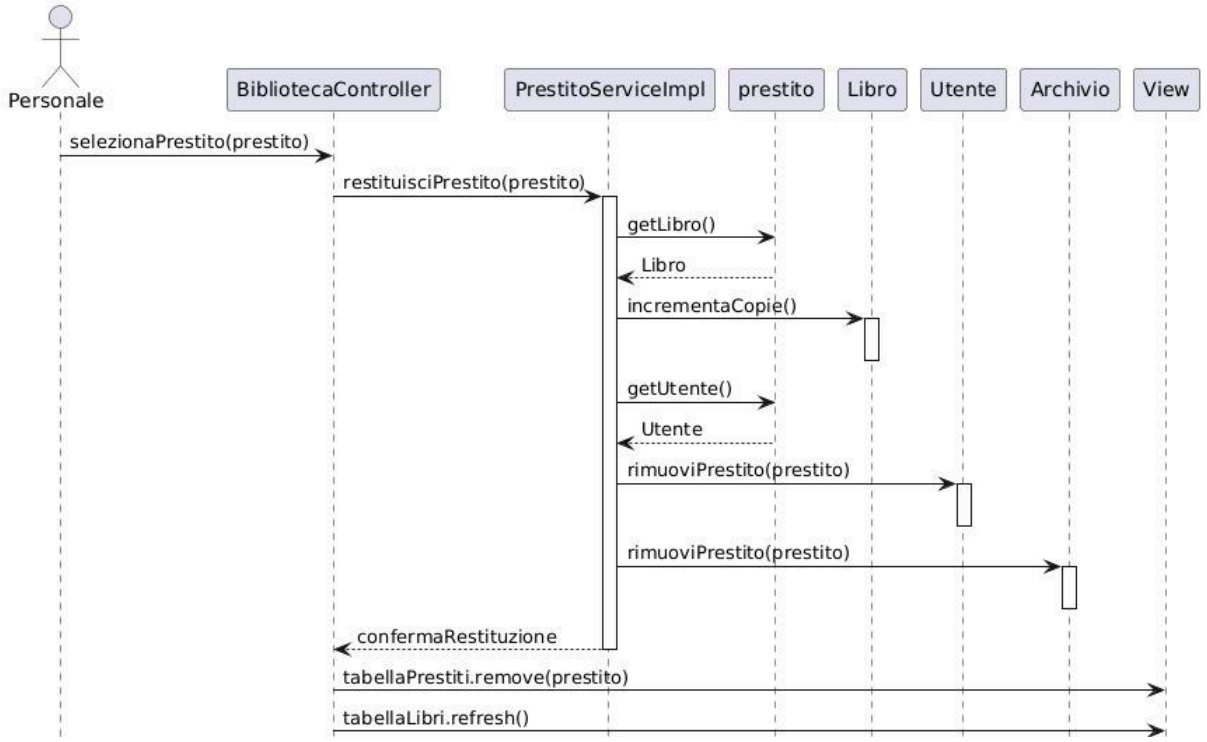
3. Modello dinamico

3.1. Casi d’uso significativi

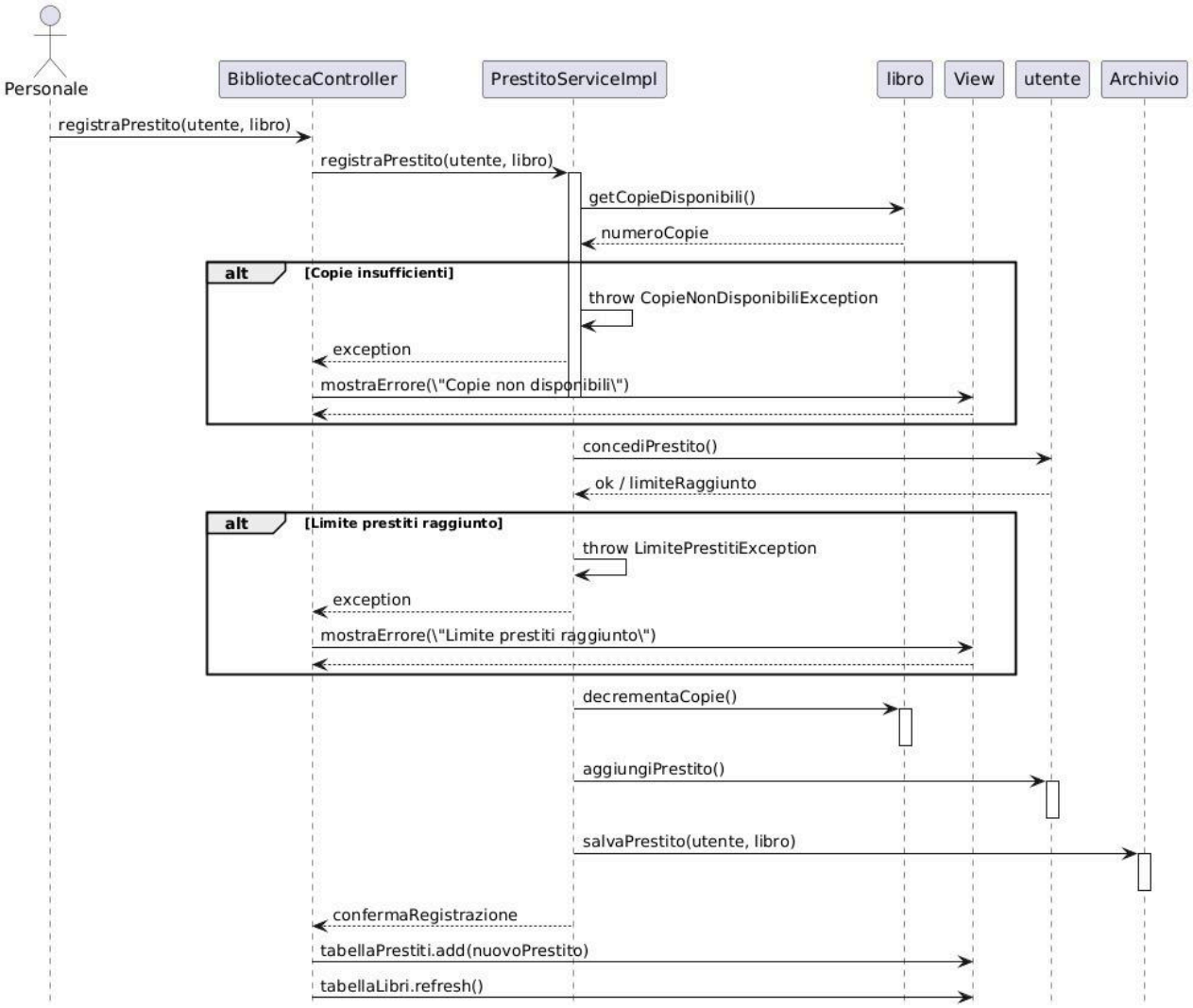
- **UC-4.10: Registrazione Prestito:** L'azione viene avviata dal Personale e dal BibliotecaController. Il Controller delega immediatamente la logica a PrestitoServiceImpl.registraPrestito(utente, libro). Il Service Layer è l'unico responsabile dell'applicazione delle regole di business: verifica la disponibilità delle copie (libro.getCopieDisponibili() > 0) e il limite prestiti dell'utente (utente.concediPrestito()). Se i controlli hanno successo, il Service chiama i metodi del Model (libro.decrementaCopie(), utente.aggiungiPrestito()) e persistendo il nuovo oggetto in Archivio. Qualsiasi fallimento nella logica (es. limite prestiti raggiunto) genera una Exception specifica, che viene catturata dal Controller per notificare l'utente tramite un Alert.
- **UC-4.11: Registrazione Restituzione:** Il Personale seleziona il prestito e il BibliotecaController invoca PrestitoServiceImpl.restituisciPrestito(prestito). Il Service Layer esegue l'operazione inversa rispetto alla registrazione, garantendo la coerenza dello stato dei dati: chiama p.getLibro().incrementaCopie() e p.getUtente().rimuoviPrestito(p), per poi rimuovere il prestito dalla lista centrale dell'Archivio. Il Controller completa l'operazione aggiornando la View (rimozione immediata dalla tabellaPrestiti).
- **UC-4.6: Registrazione Utente:** L’azione viene avviata dal Personale che inserisce i dati nelle rispettive caselle di testo che vengono controllate dal BibliotecaController con la funzione handleAggiungiUtente(ActionEvent event), crea un nuovo utente e richiama la funzione utenteService.registraUtente(nuovoUtente), l'implementazione dell’interfaccia UtenteService controlla che i dati per accedere dell’utente non siano già presenti nel sistema in caso di errore parte un'eccezione che mostra un messaggio, il bibliotecaController aggiorna la lista visibile degli utenti e pulisciCampiUtente() se non ci sono errori viene mostrato uno showAlert per indicare il successo dell’operazione o parte un'eccezione.

3.2. Diagrammi di sequenza

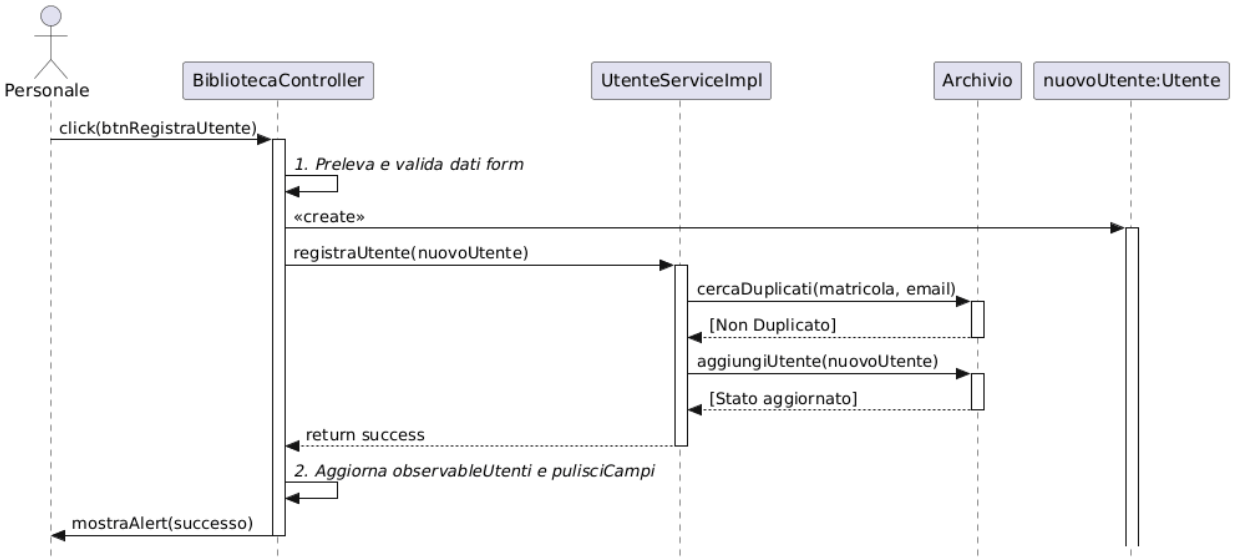
3.2.1. UC-4.10: Registrazione Prestito



3.2.2. UC-4.11: Registrazione Restituzione



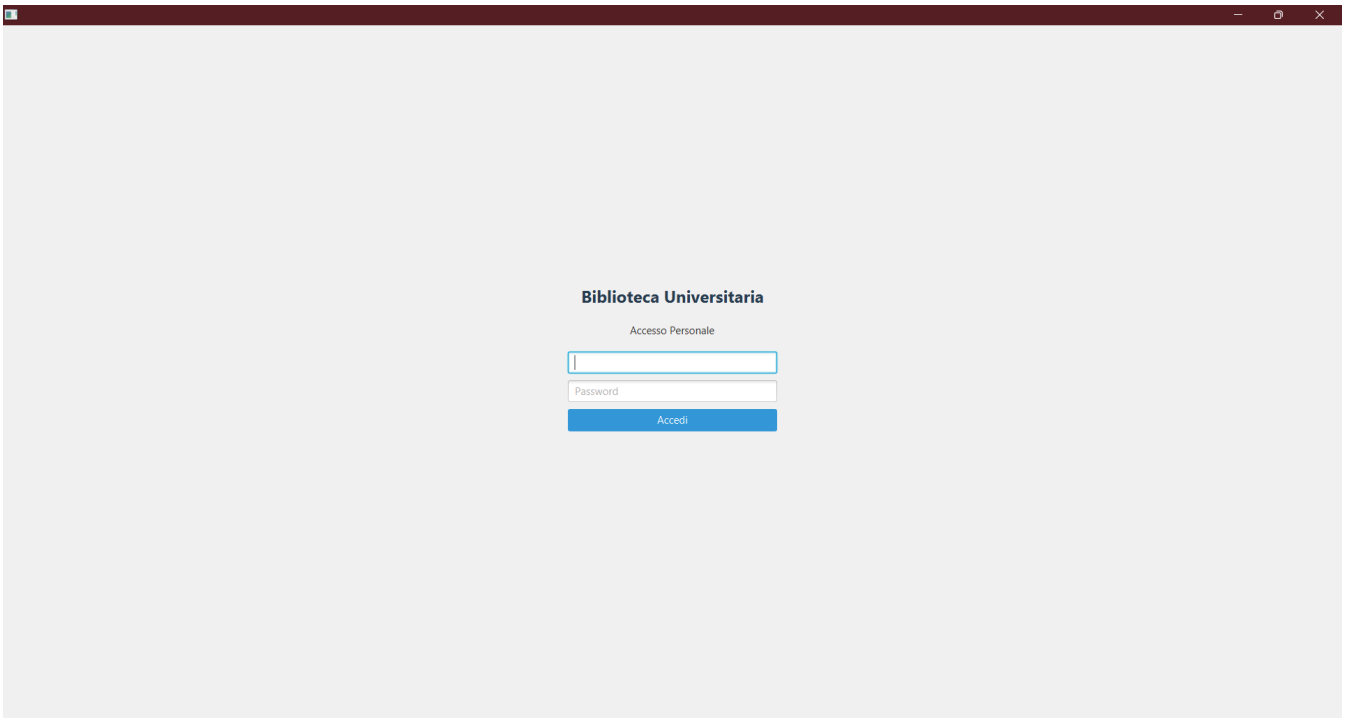
3.2.3. UC-4.6: Registrazione Utente:



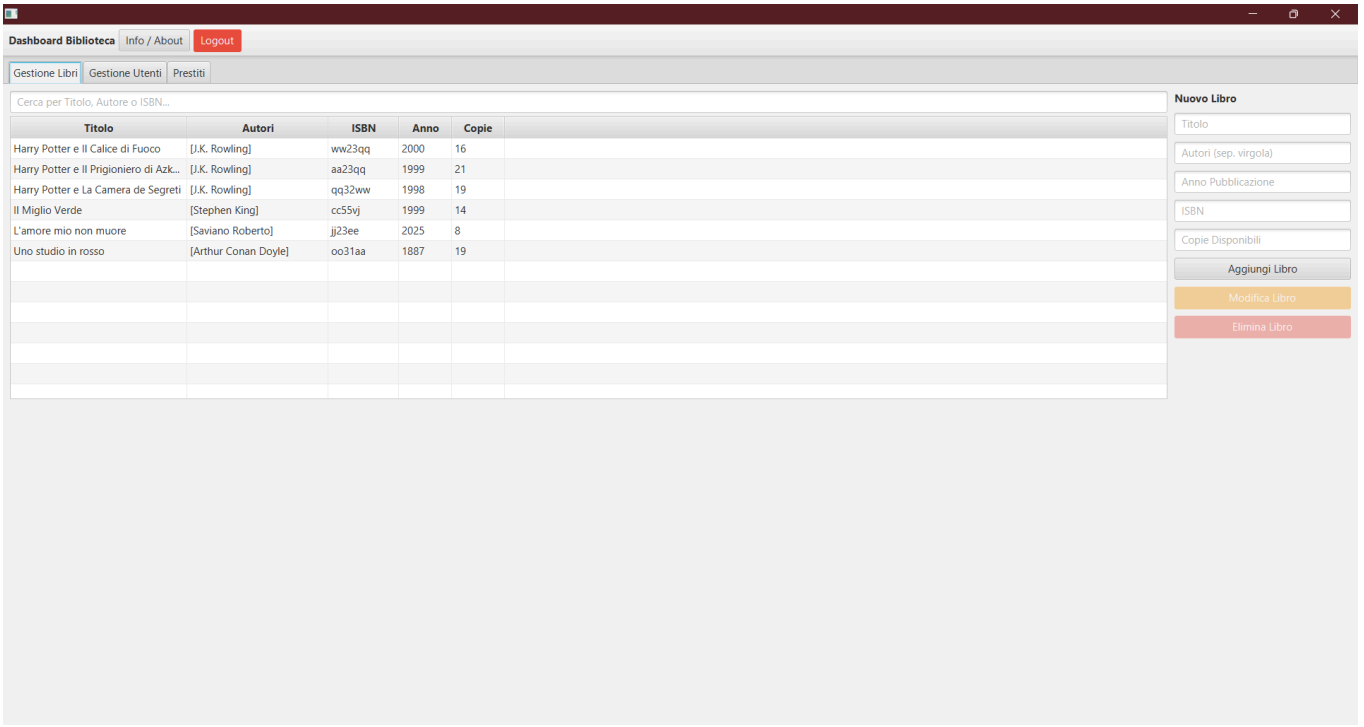
4. Design dell’interfaccia utente

4.1. Mokeup e Wireframe delle schermate principali

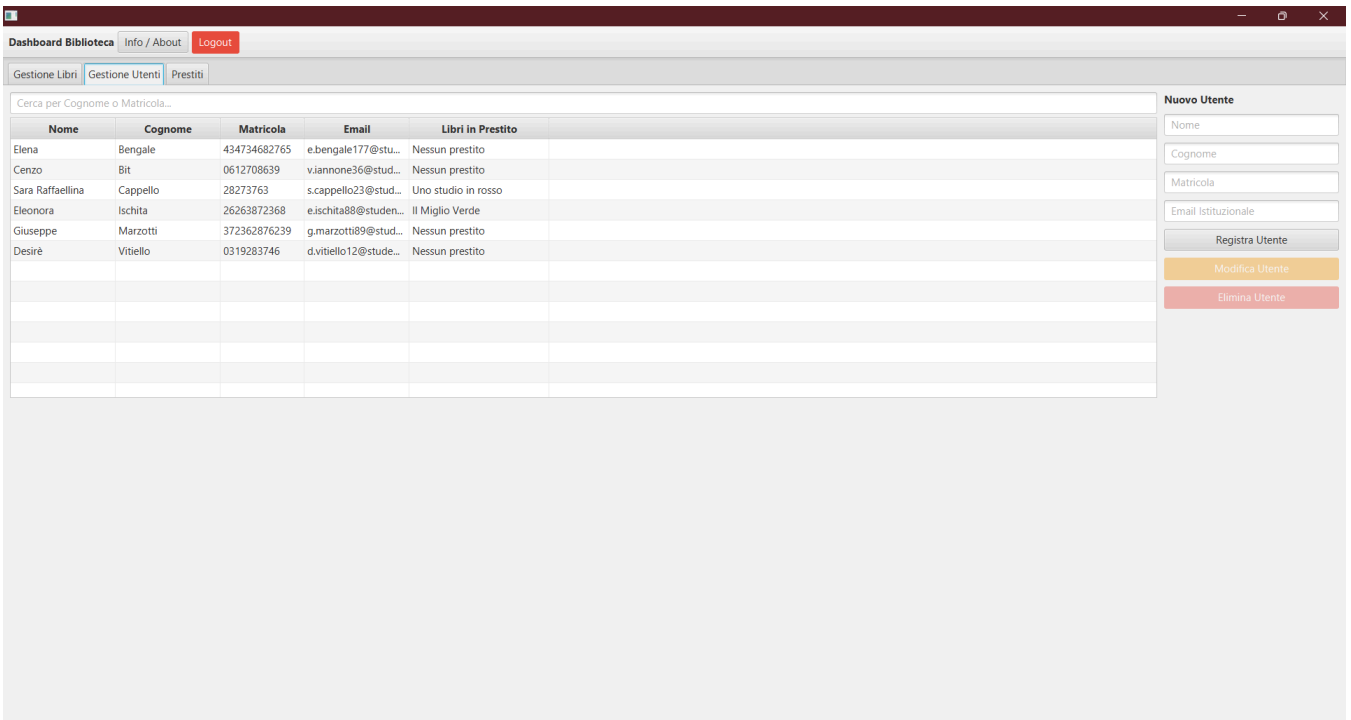
- **Schermata di login (login.fxml):** L’accesso esclusivo per il Personale, contiene un titolo fisso “Biblioteca Universitaria”, la label “Accesso Personale”, i campi txtEmail, txtPassword e un pulsante btnLogin. tutto questo è gestito dal controller LoginController.



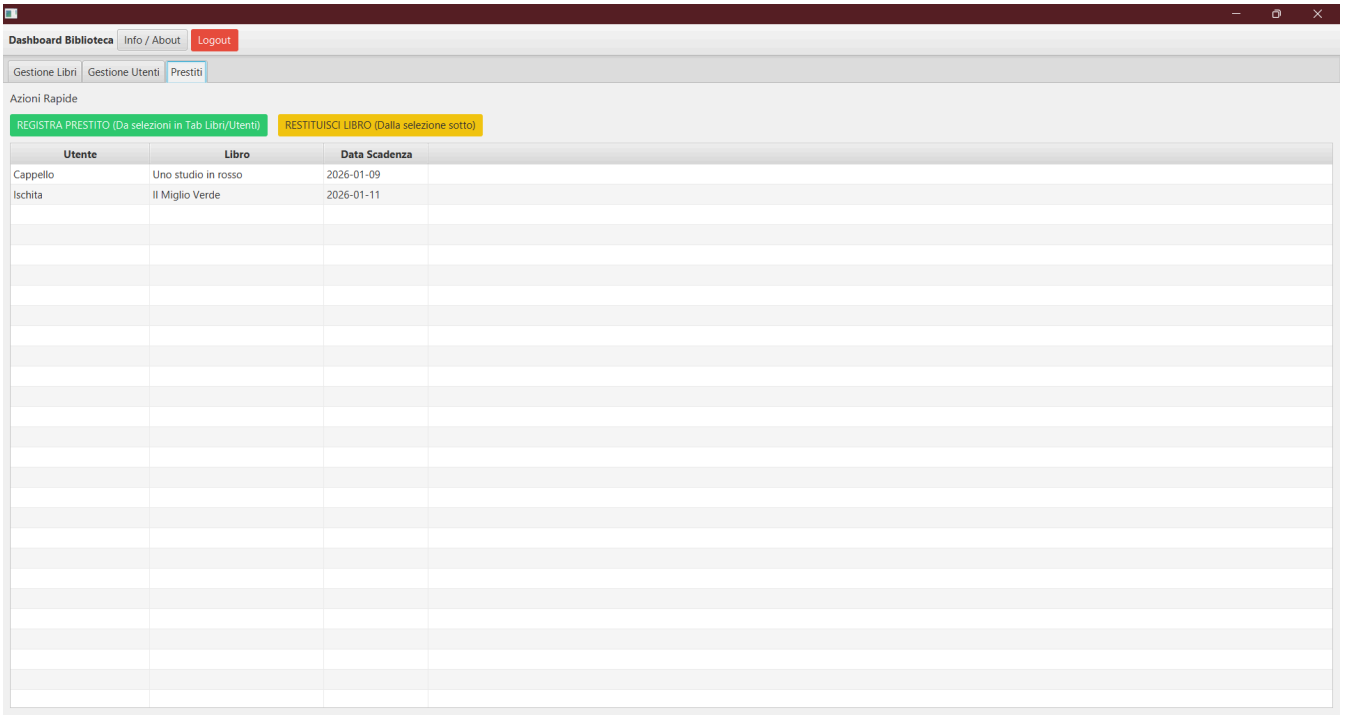
- **Dashboard Principale (primary.fxml):** Dashboard multifunzionale per le operazioni del personale. Utilizza un BorderPane con una ToolBar in alto (pulsanti Logout, Info) e un TabPane al centro per organizzare i dati e le funzionalità. Gestita da BibliotecaController. Visualizza tre schede:
    - **Scheda 1:** Contiene i form per Aggiunta/Modifica/Elimina Libro e la tabellaLibri con colonne per Titolo, Autore, ISBN, Anno e Copie disponibili.



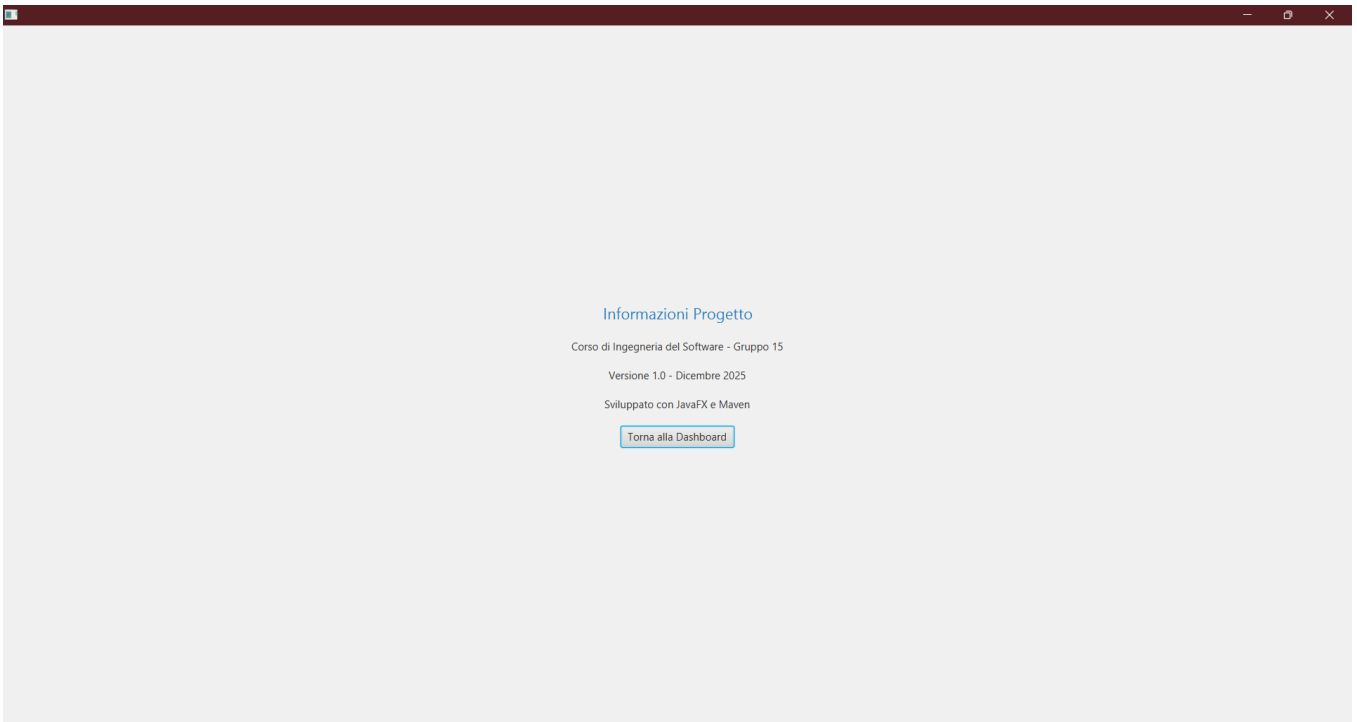
- **Scheda 2:** Contiene i form per Registrazione/Modifica/Eliminazione Utente e la tabellaUtenti con colonne Nome, Cognome, Matricola, Email e Libri in prestito.



- **Schema 3:** Contiene le Azioni Rapide (registra prestito, restituisci libro), la tabellaPrestiti (Utente, Libro, Data Scadenza).



- **Dashboard informativa (secondary.fxml):** Schermata informativa accessibile dalla Dashboard tramite il pulsante "Info". Contiene dettagli sul progetto, come la Versione, l'Anno e le Tecnologie utilizzate (JavaFX e Maven). Gestita da SecondaryController.

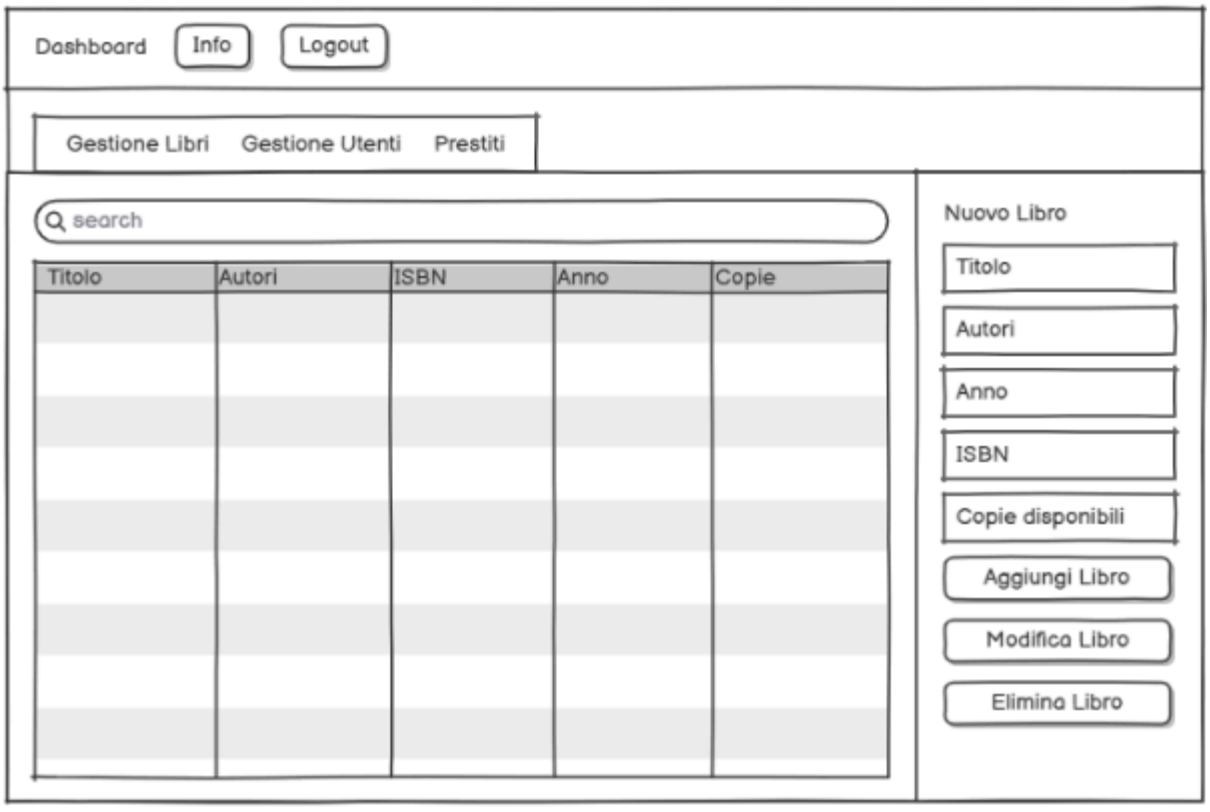


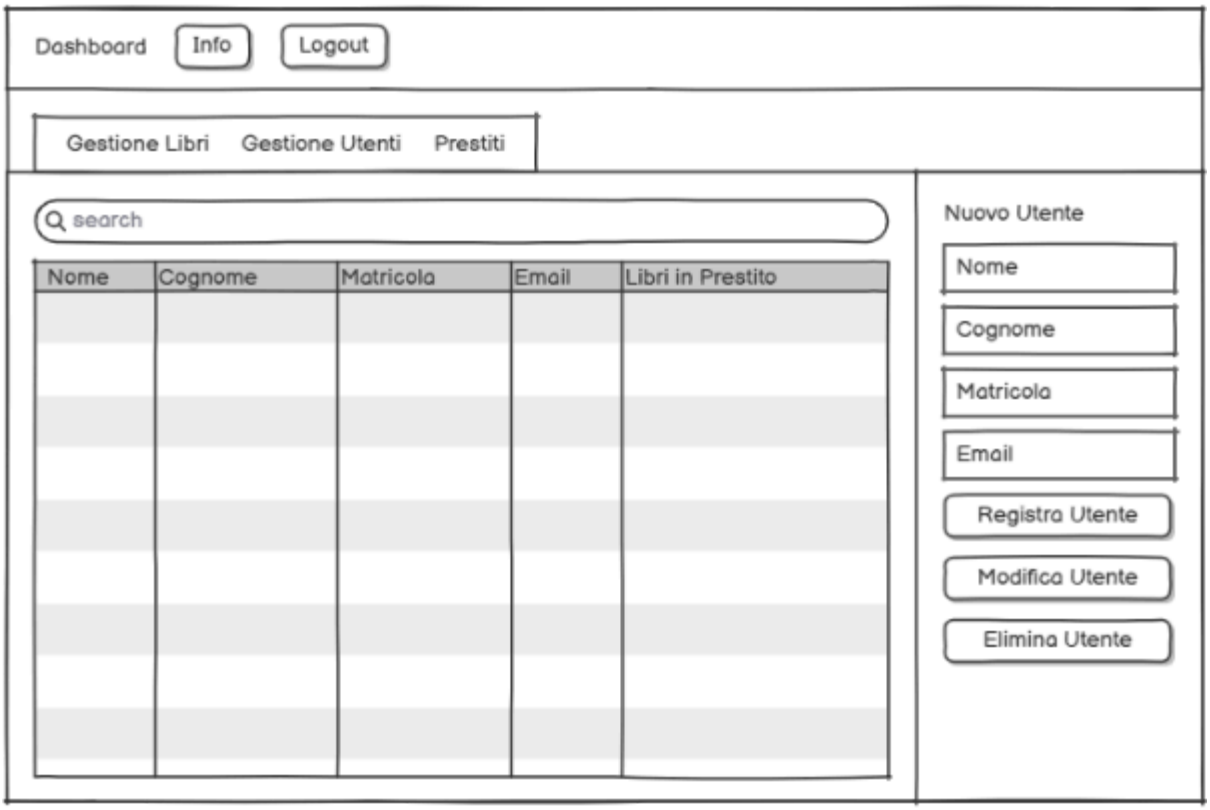
Biblioteca Universitaria

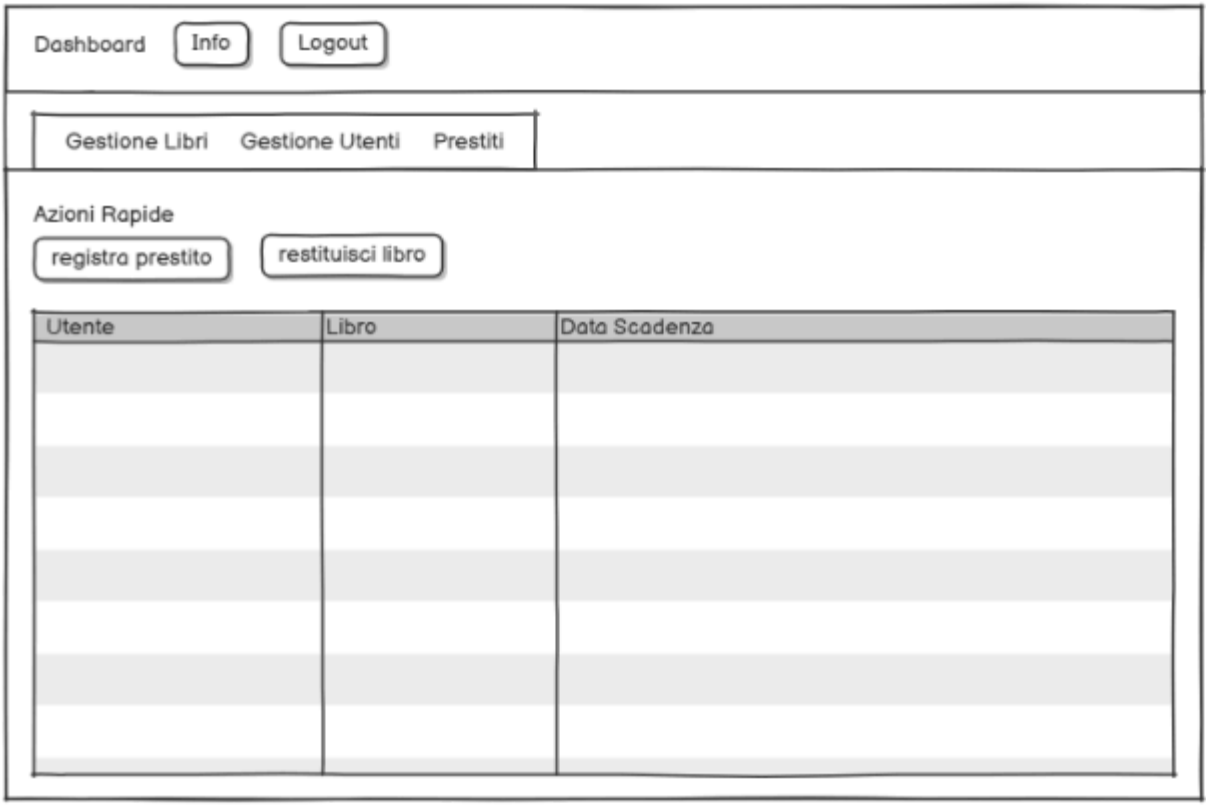
email

password

accesso




Dashboard
Info
Logout
Gestione Libri
Gestione Utenti
Prestiti
search
Nome
Cognome
Matricola
Email
Libri in Prestito
Nuovo Utente
Nome
Cognome
Matricola
Email
Registra Utente
Modifica Utente
Elimina Utente


Dashboard
Info
Logout
Gestione Libri
Gestione Utenti
Prestiti
Azioni Rapide
registra prestito
restituisci libro
Utente
Libro
Data Scadenza

4.2. Descrizioni interazioni

Interazione Utente	**Controller Responsabile**	**Flusso di interazione**
Login	LoginController	Azione: Clic sul pulsante Login. Flusso:il controller controlla che txtEmail e txtPassword non siano vuoti. Risposta: Se successo, App.setRoot("primary") (passaggio alla dashboard). Se fallimento, lblErrore.setText(...).
Registrazione Utente	BibliotecaController	Azione: Clic sul pulsante REGISTRA UTENTE. Flusso: il controller Preleva i dati del form e invoca utenteService.registraUtente(nuovoUtente) in un blocco try-catch. Risposta: se successo, aggiorna observableUtenti e svuota i campi, se fallimento cattura l'eccezione (es. "Matricola/Email duplicata") lanciata dal Service e la mostra tramite showAlert.
Registra Prestito	BibliotecaController	Azione: Clic su pulsante REGISTRA PRESTITO. Flusso: Chiama prestitoService.registraPrestito(). Risposta: In caso di successo, aggiornamento forzato delle tabelle (observableLibri.refresh()). In

		caso di Errore il controller cattura l'eccezione lanciata dal Service (es. "Libro non disponibile" o "Limite prestiti raggiunto") e la mostra tramite showAlert.
Ricerca Libro	BibliotecaController	Azione: Digita nel TextField di ricerca. Flusso: Chiama catalogoService.cercaLibri(query). Risposta: La tabellaLibri viene aggiornata con la lista filtrata per Titolo, Autore o ISBN
Registrazione Restituzione	Personale	Azione: Clic su RESTITUISCI LIBRO (necessaria selezione in tabellaPrestiti). Flusso: Chiama prestitoService.restituisciPrestito(). Risposta: Aggiornamento diretto della GUI: observablePrestiti.remove(prestitoSelezionato) e tabellaLibri.refresh() (per riflettere l'incremento delle copie).
Rimozione Utente	BibliotecaController	Azione: L'utente seleziona un utente e clicca sul pulsante RIMUOVI UTENTE. Flusso: Il controller mostra un pop-up di conferma (chiediConferma) e, se approvato, invoca utenteService.rimuoviUtente(utenteSelezionato) Risposta: Se successo rimuove l'utente da observableUtenti; se fallimento cattura l'eccezione (vincolo prestiti attivi non rispettato) e la mostra tramite showAlert.
Selezione in tabella	BibliotecaController	Azione: L'utente clicca su una riga in tabellaUtenti o tabellaLibri. Flusso: Il Controller aggiorna lo stato interno della selezione. Risposta: I pulsanti dipendenti da tale selezione (es. RIMUOVI UTENTE, PRESTA) vengono abilitati o disabilitati di conseguenza.
Logout	BibliotecaController	Azione: Clic su pulsante Logout nella ToolBar. Flusso: App.setRoot("login"). Risposta: Ritorno alla schermata di Login.
Aggiungi un libro	BibliotecaController	Azione: Clic su pulsante per l'aggiunta. Flusso: Chiama catalogoService.aggiungiLibro(). Risposta: se successo, observableLibri.add() e svuotamento campi form. se errore (es. ISBN duplicato): showAlert("Operazione Fallita").
Info / About	BibliotecaController	Azione: Clic su pulsante Info.

		Flusso: App.setRoot("secondary"). Risposta: Passaggio alla schermata con le informazioni del progetto (secondary.fxml).
Torna alla Dashboard	SecondaryController	Azione: Clic sul pulsante nella schermata Info. Flusso: App.setRoot("primary"). Risposta: Ritorno alla dashboard principale.
Salvataggio Dati	Gestito da App	Azione: Non è un'azione utente. Viene attivato da un evento di sistema critico. Flusso: Il metodo App.stop(), eseguito dal ciclo di vita JavaFX alla chiusura dell’applicazione, invoca Archivio.salva("archivio.dat") per la persistenza completa dello stato del sistema (serializzazione).