Documento di progettazione

Indice:

1. Architettura del sistema
   - 1.1. Moduli principali e responsabilità
   - 1.2. Pattern architetturali
   - 1.3. Diagramma dei package
2. Modello statico
   - 2.1. Diagramma delle classi
   - 2.2. Descrizione delle classi
   - 2.3. Scelte progettuali
3. Modello dinamico
   - 3.1. Casi d’uso significativi
   - 3.2. Diagrammi di sequenza
4. Design dell’interfaccia utente
   - 4.1. Moke-up e Wireframe delle schermate principali
   - 4.2. Descrizioni interazioni

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	le classi di questo package sono dipendenti alla classe Archivio
Accesso ai dati	com.mycompany.bibliotecainds.data com.mycompany.bibliotecainds.model	Gestisce la Persistenza su file tramite Serializzazione	Nessuna dipendenza

1.2. Pattern architetturali:

per questo progetto abbiamo utilizzato il pattern MVC (Model-view-Controller)

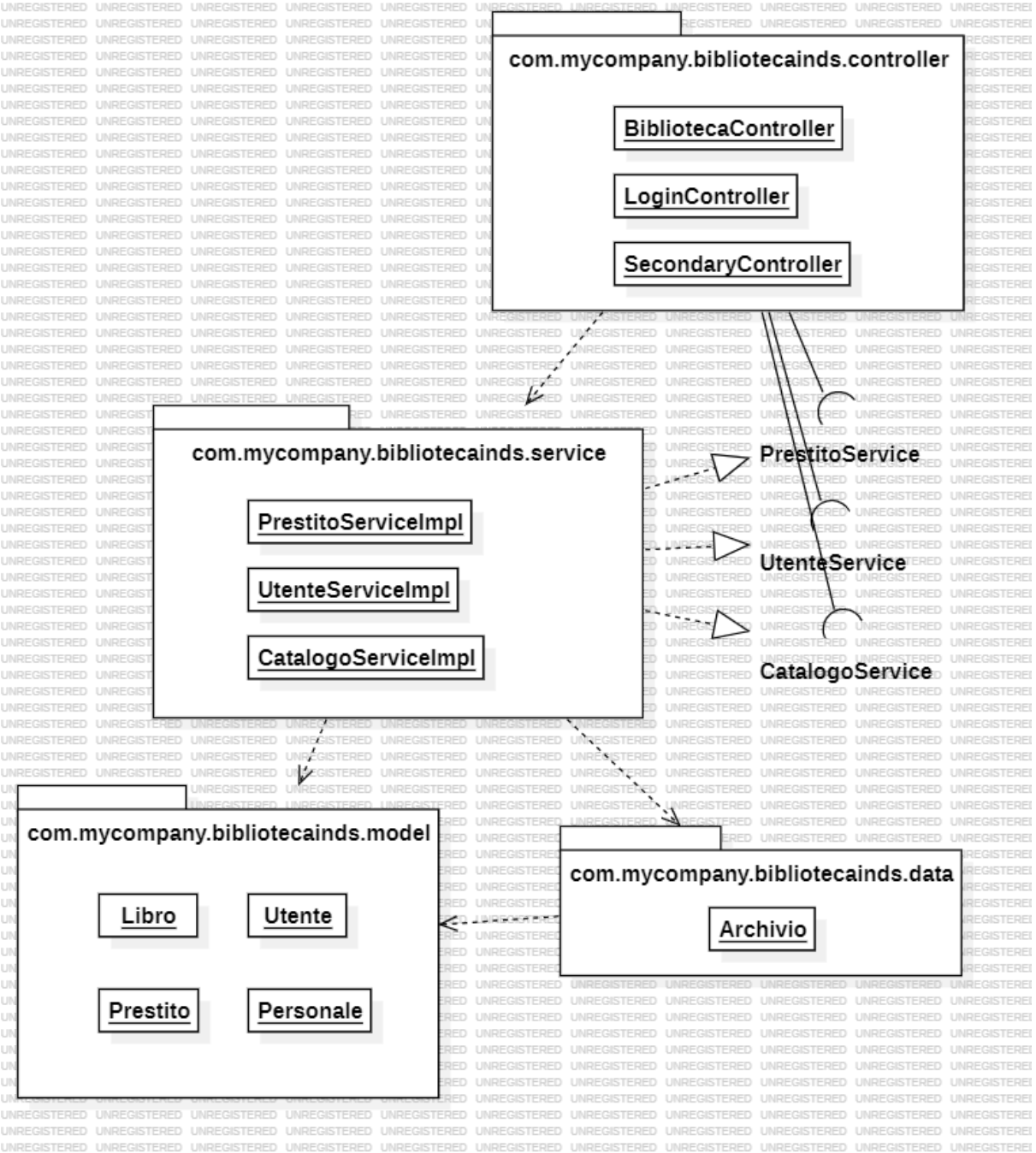
- Model: le classi di dominio (LIbro, Utente, Personale, Prestito) e il gestore dei dati (Archivio)
- View: i file FXML (login.fxml, primary.fxml, secondary.fxml)
- Controller: le classi LoginController, BibliotecaController, SecondaryController.

Altri pattern rilevanti sono:

- Singleton: Implementato nella classe Archivio con il metodo statico Archivio.getInstance(). Garantisce un'unica fonte di verità e un accesso centralizzato ai dati.

- Dependency Inversion Principle (DIP): Applicato nel Service Layer, dove i Controller dipendono dalle interfacce anziché dalle implementazioni concrete.
- Repository Pattern: La classe Archivio funge da repository in memoria, fornendo liste centralizzate ai Service Layer.

1.3. Diagramma dei package



2. Modello statico

2.1. Diagramma delle classi

2.2. Descrizione delle classi

Classe	**Attributi**	**Metodi pubblici e Logica di business**	**Relazioni**
Libro	private String titolo, private String isbn, private int copie, private List<String> autori, private int anno	getCopieDisponibili(): restituisce copie. decrementaCopie()/incrementaCopie(): metodi di modifica stato (utilizzati da PrestitoServiceImpl).	Aggregato in Archivio. Associato a Prestito.
Utente	private String nome, private String cognome, private String matricola, private String email, private List<Prestito> listaPrestiti	concediPrestito(): Ritorna true se listaPrestiti.size() < 3. aggiungiPrestito(p) / rimuoviPrestito(p): Gestione della lista prestiti interna.	Aggregato in Archivio. Composizione con Prestito.
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.	Riferisce 1 Libro e 1 Utente. Aggregato in Archivio.
Personale	private String email, private String password	getEmail(), getPassword(), modificaCatalogo(), gestisciUtenti()	
Archivio	private List<Libro> catalogoLibri, private List<Utente> registroUtenti, private List<Prestito> prestitiAttivi, private static Archivio instance	getInstance(), salva(), carica()	Contenitore (Repository) di tutte le classi Model.
PrestitoServiceImpl		registraPrestito(u, l): Lancia Exception se	Implementa PrestitoService.

		copie <= 0 O se !u.concediPrestito(). Chiama libro.decrementaCopie() e utente.aggiungiPrestito(). restituisciPrestito(p): Chiama libro.incrementaCopie() e utente.rimuoviPrestito().	Dipende da Archivio.
UtenteServiceImpl		registraUtente(u): Lancia Exception se Matricola o Email sono già presenti nell'Archivio. rimuoviUtente(u): Lancia Exception se u.getListaPrestiti().isEmpty() è false (non si cancella chi ha prestiti attivi).	Implementa UtenteService. Dipende da Archivio.
CatalogoServiceImpl		aggiungiLibro(l): Lancia Exception se l'ISBN è già presente. cercaLibri(q): Implementa la ricerca filtrando per Titolo, ISBN o Autori (case-insensitive).	Implementa CatalogoService. Dipende da Archivio.

2.3. Scelte progettuali:

- Basso Accoppiamento: il service layer e le relative interfacce servono per non unire i Controller del model al meccanismo di accesso ai dati gestito dalla classe Archivio.
- Alta coesione: ogni classe ha una responsabilità ben definita:
    - Libro: gestisce solo il proprio stato
    - Archivio: gestisce solo i dati e l'accesso alle liste

    - PrestitoServiceImpl: gestisce la logica di prestito/restituzione
- Principio Single Responsibility: Controller si limitano alla gestione della View e non alla logica di business
- Principio di Inversione delle Dipendenze: Il Controller dipende dalle interfacce e non dalle classi concrete. Questo rispetta la regola di non dipendere da dettagli, aumentando la Manutenibilità.
- Affidabilità: La logica di validazione è centralizzata nei Service Layer. In caso di fallimento di una regola di business, il Service lancia una Exception specifica. Questo impedisce che l'archivio venga modificato in uno stato inconsistente, garantendo l'Affidabilità.

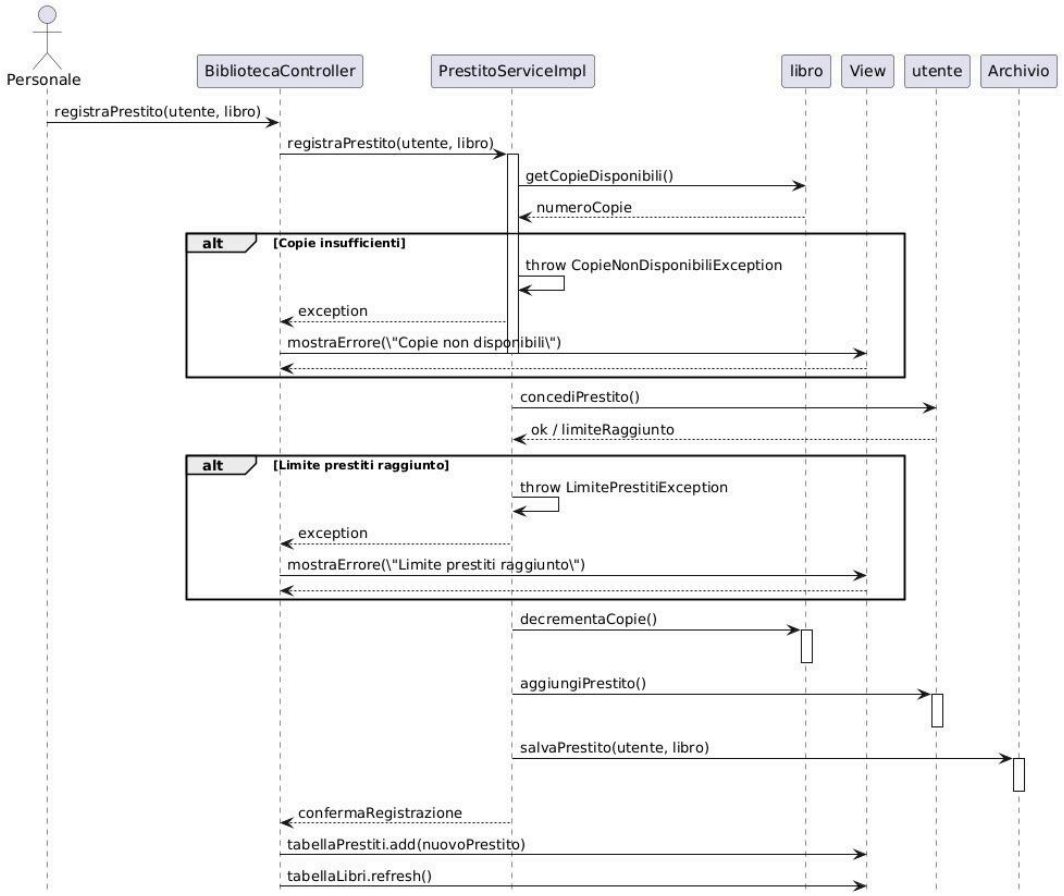
3. Modello dinamico

3.1. Casi d'uso significativi

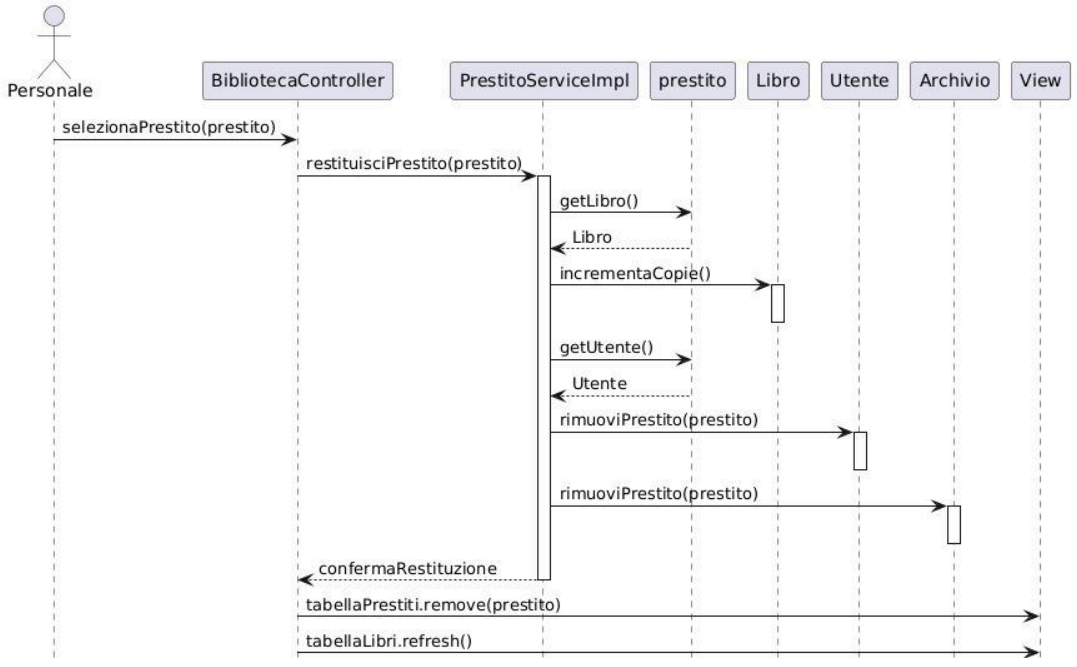
- UC-4.6: Registrazione Prestito (Personale): L'azione viene avviata dal Personale e dal BibliotecaController. Il Controller delega immediatamente la logica a PrestitoServiceImpl.registraPrestito(utente, libro). Il Service Layer è l'unico responsabile dell'applicazione delle regole di business: verifica 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-4.9: Registrazione Restituzione (Personale): 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 e aggiornamento del conteggio copie nella tabellaLibri.refresh()).
- UC-4.7: Modifica Catalogo Libri - Aggiunta (Personale): Il BibliotecaController delega l'azione di aggiunta a CatalogoServiceImpl.aggiungiLibro(nuovoLibro). Il Service verifica che l'ISBN del nuovo libro non sia già presente nel catalogo. In caso di duplicato, viene lanciata una Exception. Se l'ISBN è unico, l'oggetto Libro viene aggiunto all'Archivio, e il Controller aggiorna la tabellaLibri.

3.2. Diagrammi di sequenza

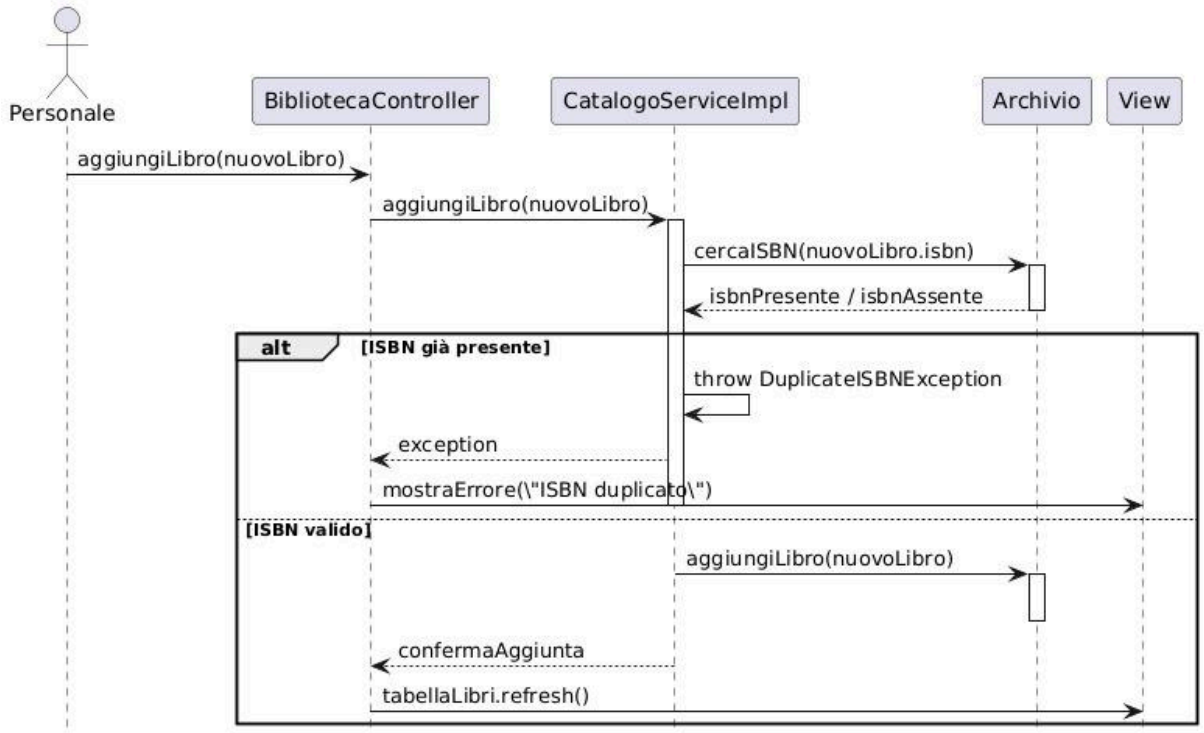
- UC-4.6: Registrazione Prestito (Personale):



- UC-4.9: Registrazione Restituzione (Personale):



- UC-4.7: Modifica Catalogo Libri - Aggiunta (Personale):



4. Design dell'interfaccia utente

4.1. Moke-up 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. visualizza due schede:
    - Scheda 1: Contiene i form per Aggiunta/Modifica Libro e la tabellaLibri con colonne per Titolo, Autore e Copie disponibili.
    - Schema 2: Contiene le Azioni Rapide (registra prestito, restituisci libro), la tabellaPrestiti (Utente, Libro, Data Scadenza) e (presumibilmente) la tabellaUtenti

4.2. Descrizioni interazioni

Interazione	Attore	Azione del Sistema

Login	Personale	Azione: Clic su pulsante Login. Flusso: Controlla che txtEmail e txtPassword non siano vuoti (logica placeholder). Risposta: Se successo, App.setRoot("primary") (passaggio alla dashboard). Se fallimento, lblErrore.setText(...).
Registra Prestito	Personale	Azione: Clic su pulsante REGISTRA PRESTITO. Flusso: Chiama prestitoService.registraPrestito(). Risposta: In caso di Successo, showAlert("Successo") e aggiornamento forzato delle tabelle (tabellaLibri.refresh(), observablePrestiti.add()). In caso di Errore (es. limite 3 prestiti), showAlert("Operazione Fallita", e.getMessage()).
Ricerca Libro	Personale	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 (come definito in CatalogoServiceImpl).
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).
Logout	Personale	Azione: Clic su pulsante Logout nella ToolBar. Flusso: App.setRoot("login"). Risposta: Ritorno alla schermata di Login.
Aggiungi un libro	Personale	Azione: Clic su pulsante per l'aggiunta. Flusso: Chiama catalogoService.aggiungiLibro(). Risposta: Successo: observableLibri.add() e

		svuotamento campi form. Errore (es. ISBN duplicato): showAlert("Operazione Fallita").
Info / About	Personale	Azione: Clic su pulsante Info. Flusso: App.setRoot("secondary"). Risposta: Passaggio alla schermata con le informazioni del progetto (secondary.fxml).
Torna alla Dashboard	Personale	Azione: Clic sul pulsante nella schermata Info. Flusso: App.setRoot("primary"). Risposta: Ritorno alla dashboard principale.
Salvataggio Dati		Azione: Chiamato in un punto critico del sistema (es. prima della chiusura dell'applicazione). Flusso: Serializzazione dell'istanza Singleton dell'Archivio su file.