DIAGRAMMA DELLE CLASSI
DESCRIZIONE

1.1 PRINCIPI DI BUONA PROGETTAZIONE

1.1.1 FRONTEND

Il front-end comprende la gestione delle interfacce grafiche e delle interazioni dell'utente attraverso l'uso di JavaFX e di FXML; la sua struttura interna segue il pattern architetturale dell'MVC. L'uso di questo pattern garantisce una separazione delle responsabilità (SoC) delle classi che lo riguardano grazie alla suddivisione in model, view e controller.
E' stata inoltre realizzata una classe astratta estesa dai vari controller del progetto; questa classe permette la condivisione di funzionalità più articolate (es. apertura di popUp) tra le diverse scene del progetto usando lo stesso codice (DRY).

1.1.2 BACKEND

Il backend comprende la gestione di utenti, prestiti, libri e della password di accesso del bibliotecario. Esso è formato dall'implementazione delle interfacce descritte precedentemente attraverso l'uso di un database relazionale.
La scelta dell'uso di un database relazionale garantisce modularità alla persistenza dei dati: si può optare per una soluzione su file (con SQLite) o si può spostare la persistenza su un DBMS (PostgreSQL, MySQL…); inoltre il formato dei dati salvati sul database NON dipende dalla serializzazione di oggetti Java e quindi la persistenza è anche modulare rispetto al linguaggio di programmazione utilizzato.

1.1.3 COMUNICAZIONE FRONTEND/BACKEND

La comunicazione tra front-end e back-end avviene con l'uso di diverse, generiche interfacce chiamate "Service" che hanno lo scopo di isolare le funzionalità richieste dal front-end (gestione utenti, prestiti…) con la loro effettiva implementazione.
Questa architettura ha come principi cardine:

- Principio della Singola Responsabilita': ogni interfaccia espone metodi relativi ad una singolo modello del progetto (utente, libro, prestito, password)
- Principio Aperto-Chiuso: se un cliente volesse modificare o estendere le funzionalità fornite da un'interfaccia può estendere una delle implementazioni fornite dal back-end senza dover modificare in alcun modo i controller presenti.
- Principio di Sostituzione di Liskov: in corrispondenza di ogni interfaccia si può sostituire una qualsiasi sua implementazione e il programma continuerà a funzionare correttamente
- Principio della Segregazione delle Interfacce: sono state realizzate diverse interfacce che permettono ad un client (es. il front-end) di dipendere solo dalla funzionalità realmente richiesta
- Principio dell'Inversione delle Dipendenze: la comunicazione tra i moduli di basso livello (es. backend) e quelli di alto livello (es. frontend) dipende esclusivamente dall'uso di queste interfacce

Inoltre il progetto comprende numerose eccezioni descrittive (es DuplicateBookByIsbnException) che permettono la gestione di errori e la comunicazione chiara con il bibliotecario.

1.2 COESIONE ED ACCOPPIAMENTO

1.2.1 COESIONE

1.2.1.1 FRONTEND
Le classi Controller presentano un livello di coesione funzionale. Ogni controller è legato a una singola scena FXML e gestisce solo gli eventi e la logica di presentazione che riguardano quella specifica schermata (es. `AddBookSceneController` gestisce solo l'aggiunta dei libri).

1.2.1.2 BACKEND
I servizi implementati dal backend presentano un livello di coesione funzionale. Le classi (es. `DatabaseUserService`) raggruppano metodi che operano sulla stessa entità di dominio e contribuiscono ad un unico compito logico (es. la gestione della persistenza per quell'entità), senza includere logiche estranee.

1.2.2 ACCOPPIAMENTO

1.2.2.1 FRONTEND
Il frontend mantiene un basso accoppiamento con il backend grazie all'uso delle interfacce dei Service. I controller dipendono dall'astrazione (`BookService`) e non dalla concretezza (`DatabaseBookService`).

1.2.1.2 BACKEND
I servizi implementati presentano un accoppiamento di tipo stamp, in quanto le implementazioni richiedono l'uso di un oggetto di tipo `Database` (wrapper di JDBI) per funzionare.