**Library Manager**

**Progetto Software**

**Autore:***Gabriele Vianello*

**Anno Accademico** 2024 – 2025

Politecnico di Milano

**Indice**

**Introduzione**

Il progetto prevede la realizzazione di un applicativo software, denominato *Library Manager*, che permetta di gestire una biblioteca. Per gestione si intende l’aggiunta, modifica, ricerca, eliminazione di libri ed utenti, così come per i prestiti dei libri agli utenti; le informazioni saranno gestite da un istanza locale di database SQL relazionale.

Poiché l’applicativo è realizzato esclusivamente in java potrà lavorare su differenti sistemi operativi/piattaforme[[1]](#footnote-1), gestendo anche più connessioni simultaneamente. L’interfaccia console sarà utilizzata per messaggi di debug e log (sia lato server che client), mentre la GUI sarà realizzata utilizzando JavaFX.

Saranno realizzati appositi test di funzionamento per tutti i modelli e quelle classi che saranno predisposte per la loro manipolazione.

Ulteriore documentazione, come JavaDoc, diagramma UML completo del sistema e percentuali di coverage dei test si trovano nell’apposita cartella all’interno del progetto.

**Architettura generale**

L’applicativo implementato ha due sottocomponenti eseguibili differenti, uno per il dispositivo client ed uno server. Il protocollo di comunicazione usato dai due moduli prevede lo scambio di oggetti *Message* che come le classi dei modelli estende *Serializable*. Quindi i modelli dei libri, utenti prestiti possono essere facilmente trasmessi attraverso *ObjectInputStream* (ricezione) *e ObjectOutputStream* (trasmissione).

Ogni connessione server client è indipendente e viene gestita lato server come un thread che riceve e risponde alle richieste del client; dato che la maggior parte delle richieste comportano un accesso al DB, e nell’ipotesi che si possano richiedere accessi simultanei ad esso ma con client differenti, sono stati resi synchronized i metodi che accedono direttamente al DB[[2]](#footnote-2)

Per garantire la modularità del sistema sono stati adottati il DAO Pattern per l’accesso al database e l’MVC per un interazione ottimale ed efficiente sia con utente finale che con il server.

UML generale.

**Modelli:** quelli previsti ed implementati (da come si può vedere dalle classi nell’UML precedente) sono *Books*, *Lends* e *Customers*. Contengono oltre ai necessari attributi anche – eventualmente – metodi per l’elaborazione dei dati come nel caso dei prestiti, dove è stato implementato un metodo che restituisce il valore booleano *true* nel caso in cui vi sia un ritardo nella riconsegna del libro.

**Gerarchie:** tutte le operazioni previste dovranno essere richieste dal client al server, il tutto passando per delle classi intermedie per l’elaborazione dei dati, incapsulando la richiesta in un messaggio che sarà serializzato e spedito al server. Allo stesso modo, le risposte in ingresso ripercorrono la stessa strada ma al contrario; in quest’ultimo caso, a rimanere sarà l’oggetto richiesto.

Descrizione dell'architettura client/server utilizzata, con i principali componenti e il flusso delle operazioni.

**MVC (Model-View-Controller)**

L'MVC è un pattern architetturale utilizzato principalmente nelle applicazioni con interfaccia utente (UI). Divide l'applicazione in tre componenti principali:

1. **Model (Modello):** rappresenta la logica di business e i dati dell'applicazione. È responsabile della gestione dello stato dell'applicazione dove gli oggetti rappresentano entità, che in questo caso possono essere *libri*, *prestiti* o *utenti*.
2. **View (Vista):** è la parte che gestisce l'interfaccia utente, mostra i dati all'utente e invia i comandi dell'utente al *Controller*; un esempio di *View* è la finestra JavaFX.
3. **Controller (Controllore):** gestisce l'interazione tra il Model e la View; riceve input dall'utente tramite la View, elabora i dati (eventualmente con l'aiuto del Model) e aggiorna la View. Concettualmente, lavora come un intermediario, tra la view ed il modello ottenuto a partire dai dati contenuti nel Database.

In questo modo – oltre a garantire la separazione della logica di business da quella di visualizzazione – facilita il mantenimento e la testabilità del codice, oltre ad una maggiore flessibilità in caso di aggiornamenti futuri.

**DAO (Data Access Object)**

Il DAO è un pattern che si concentra sull'accesso ai dati. È una classe o un insieme di classi che fornisce un'interfaccia astratta per interagire con un database o un'altra fonte di dati.

**Componenti principali del DAO:**

1. **Interfaccia DAO:** definisce i metodi per operazioni CRUD (Create, Read, Update, Delete). Un esempio di interfaccia è *BookDAOInterface* che conterrà le dichiarazioni dei metodi che saranno utilizzati per l’accesso/estrazione dei dati dal DB.
2. **Implementazione DAO:** contiene l'implementazione concreta dell'interfaccia, interagendo con il database tramite JDBC[[3]](#footnote-3). Nasconde la complessità della gestione del database, come connessioni, query e chiusura delle risorse.

L’adozione ed implementazione del DAO favorisce la riutilizzabilità del codice; dona anche una maggiore modularità al sistema oltre a garantire un certo grado di facilità nel testing.

**Modulo di gestione libri (Books)**

* Descrizione delle entità e delle operazioni CRUD (Create, Read, Update, Delete) sui libri
* Implementazione del DAO per l'interfaccia con il database
* Implementazione del controller e della view per le operazioni sui libri

**Modulo di gestione prestiti (Lends)**

* Descrizione delle entità e delle operazioni CRUD sui prestiti di libri
* Implementazione del DAO per l'interfaccia con il database
* Implementazione del controller e della view per la gestione dei prestiti

**Modulo di gestione clienti (Customers)**

* Descrizione delle entità e delle operazioni CRUD sui clienti
* Implementazione del DAO per l'interfaccia con il database
* Implementazione del controller e della view per la gestione dei clienti

**Server**

**Disconnessione sicura:** in caso di disconnessione, che sia da parte del client o del server, sarà gestita tramite le eccezioni. Una volta generata sarà usata per richiamare i metodi predisposti per la chiusura dei socket e per …RIVEDERE.

UML, clientHandler, gestione dei messaggi e connessione.

**Client**

UML, gestione dei messaggi, richieste a server, MVC.

**Interfaccia grafica**

GUI, spiegazione.

**Tecnologie utilizzate**

Elenco delle principali tecnologie e librerie utilizzate nell'applicativo:

* Java
* JavaFX
* Database (es. MySQL, PostgreSQL)

**Macchina e Ambiente di Sviluppo:** il software è stato sviluppato interamente su macchina Windows 10 Pro con istanza di MySql8.0 e utilizzando IntelliJ IDEA come ambiente di sviluppo. Le query SQL sono state scritte su MySql WorkBench.

**Installazione e configurazione**

Istruzioni per l'installazione e la configurazione dell'applicativo, inclusi i requisiti di sistema e le dipendenze.

**Considerazioni finali**

Eventuali note aggiuntive o considerazioni sull'applicativo.

1. A patto che abbiano una Java Virtual Machine installata. [↑](#footnote-ref-1)
2. Cioè tutti i metodi/funzioni che effettuano query al DB nelle classi che riportano la dicitura “…DAO.java”, come previsto dal DataAccessObject pattern. [↑](#footnote-ref-2)
3. È la libreria per l’accesso al DB utilizzata nel progetto. [↑](#footnote-ref-3)