# Book Recommender

## Manuale Tecnico

Autori:

Cagnetta Diana

Romano Aurora Lindangela

Caliaro Fabio

Fedeli Riccardo

# Sommario

# Premessa

Sebbene l'applicativo sia stato prodotto nell'ambito del progetto del Laboratorio B per il *corso di laurea in informatica dell'Università degli Studi dell'Insubria*, si è cercato di impiegare l'organizzazione strutturale tipica delle applicazioni di livello entreprise, adottandone il più possibile.

In questo manuale non verrà discusso il funzionamento di tutte le classi e i metodi contenuti al loro interno: verranno menzionati i componenti cruciali al funzionamento del programma e quelli di maggior rilevanza e/o complessità. Per un elenco completo e dettagliato, fare riferimento alla **Javadoc** del progetto.

# Introduzione

**"Book Recommender"** è un sistema per la valutazione e raccomandazione di libri, che permette agli utenti registrati di inserire recensioni e a tutti gli utenti di consultare le valutazioni e ricevere consigli di lettura. Per ulteriori informazioni sul funzionamento del programma, incluse le istruzioni per l'avvio, consultare il Manuale Utente.

# Software Design

Per lo sviluppo dell'applicazione si è scelto **Java 21** per sfruttare le funzionalità moderne del linguaggio, ridurre il codice boilerplate e migliorare la manutenibilità complessiva. Il sistema è stato progettato con un'architettura **client-server**: il client fornisce l'interfaccia grafica sviluppata in **JavaFX**, che favorisce la separazione tra View e logica applicativa, mentre il server espone i servizi tramite utilizzo di **socket**, al fine di semplificare la logica di connessione e rendere trasparente la gestione del multi-threading, necessario per gestire i diversi client, consentendo così di focalizzarsi sulla logica applicativa. L'applicazione è stata testata su sistemi Windows (10/11).

## Struttura dell’applicazione

Di seguito viene mostrata la struttura dei principali moduli dell’applicazione:

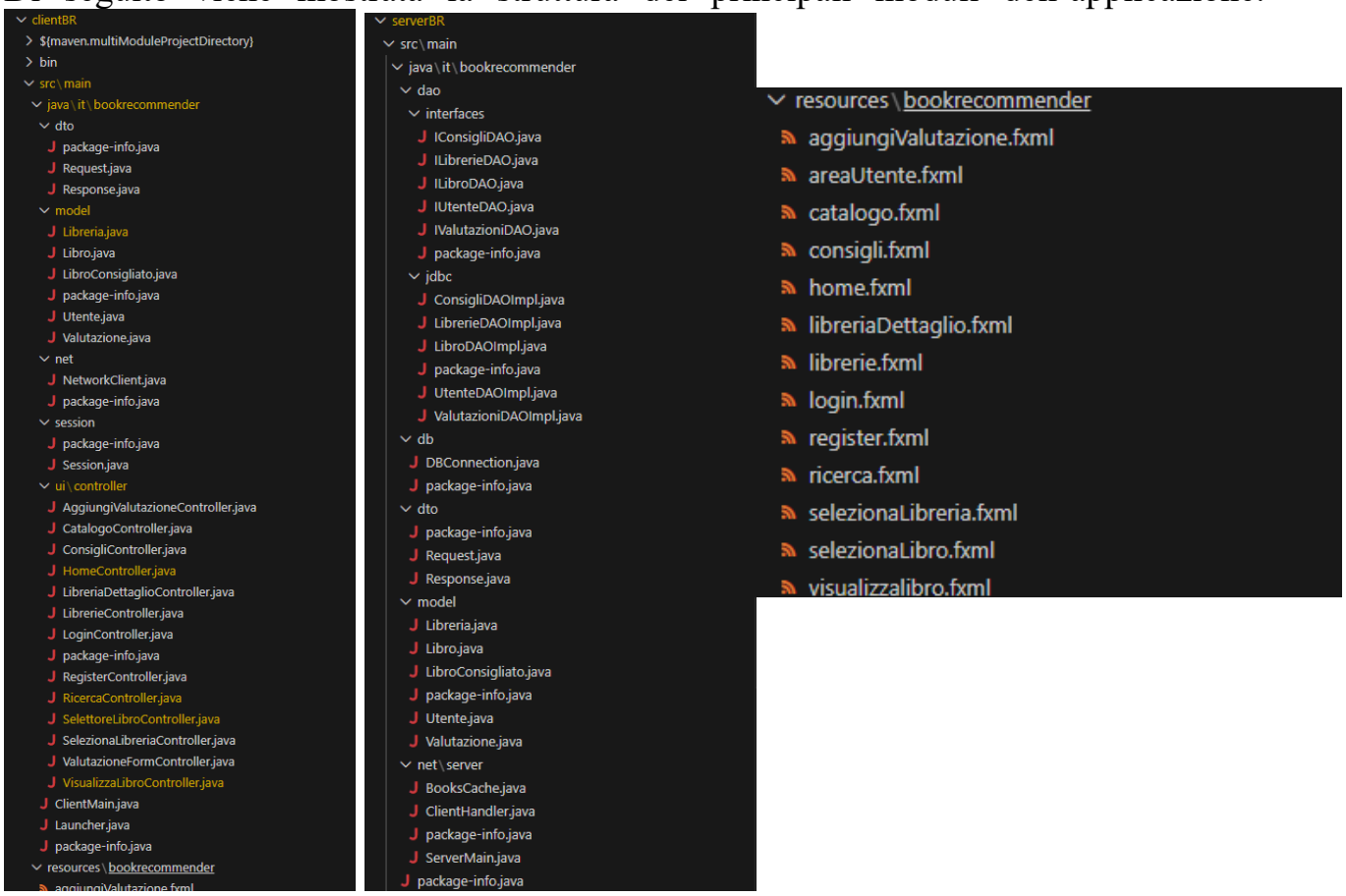


Figura 1: Struttura del progetto.

Il sistema BookRecommender è stato sviluppato con un'architettura client-server, seguendo un approccio modulare per garantire una chiara separazione delle responsabilità, facilitare la manutenzione e promuovere il riutilizzo del codice.

Il progetto è suddiviso in due moduli Maven distinti:

- **serverBR**: contiene il server;
- **clientBR**: contiene il client;

Come sistema di build è stato utilizzato **Maven**: ogni modulo possiede le sue dipendenze, specificate nel suo pom.xml, ed eventuali dipendenze condivise sono

specificate nel file pom.xml padre. I package sono stati organizzati per funzionalità, in modo da aumentare al massimo il grado di modularità.

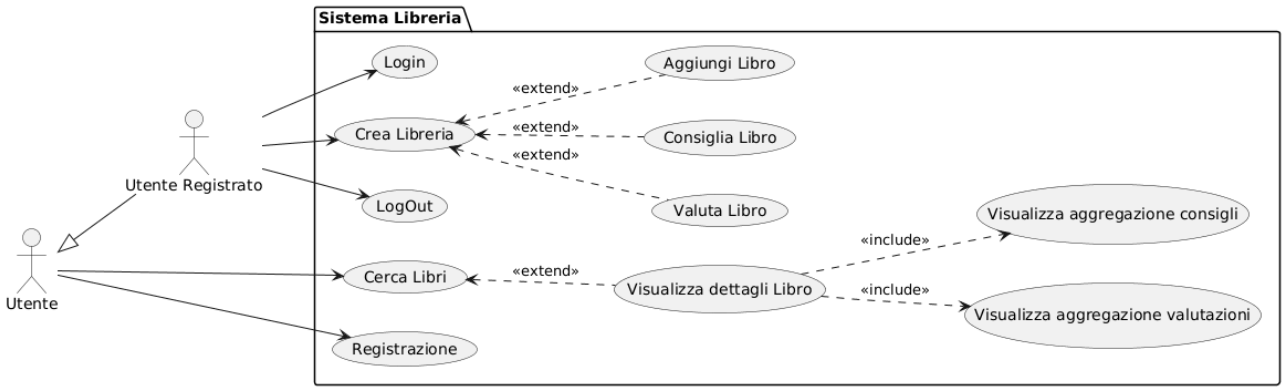



Figura 2: Use Case DiagramBook Recommender.

## Connessione al database

La connessione al database relazionale **PostgreSQL**, ospitato su **Render**, avviene tramite il driver **JDBC**, gestita centralmente dalla classe DBConnection, la quale implementa il pattern Static Factory Method. La classe contiene un unico campo statico Connection e costruttore privato. Per inizializzare la connessione si utilizza il seguente metodo:

*public static void init(String h, String u, String p) {*

*host = h;*

*user = u;*

*password = p;*

*}*

Dopo l'inizializzazione, l'accesso operativo al database avviene sempre tramite il metodo:

*public static Connection getConnection() throws SQLException {*

*if (host == null || user == null || password == null) {*

*throw new SQLException("Connessione DB non inizializzata. Chiama prima DBConnection.init().");*

*}*

*// URL del database su Render (usa sempre il nome corretto del tuo DB)*

*String url = "jdbc:postgresql://" + host + ":5432/bookrecommender_qt9c";*

*// Connessione con SSL abilitato (necessario per Render)*

*Connection conn = DriverManager.getConnection(url + "?sslmode=require", user, password);*

*return conn;*

*}*

## Entità relazionali

Prima di illustrare l'architettura del sistema, è utile chiarire il processo di mappatura tra le entità relazionali del database e le classi Java. Nel progetto BookRecommender, le principali entità sono:

- **Utente**: Rappresenta gli utenti registrati nel sistema.
- **Libro**: Rappresenta le opere presenti nel catalogo.
- **Libreria**: Gestisce le collezioni personali create dagli utenti.
- **Valutazione**: Contiene i voti analitici e i commenti relativi ai libri.
- **LibroConsigliato**: Modella l'aggregazione dei suggerimenti di lettura.

Queste classi corrispondono direttamente alle tabelle del **database SQL** (es. UtentiRegistrati, Libri, ConsigliLibri). Per ciascuna di esse è stata implementata una classe Java (POJO - Plain Old Java Object) nel package it.bookrecommender, dove ogni attributo privato rappresenta esattamente una colonna della tabella corrispondente.

Ad esempio, nella classe Libro, i campi libroId, titolo e autore mappano le chiavi e i dati della tabella Libri. Questa struttura garantisce una mappatura chiara e immediata, facilitando la logica di accesso ai dati tramite i **DAO**, la gestione delle sessioni e la

comunicazione tra Client e Server tramite oggetti di richiesta (Request) e risposta (Response).

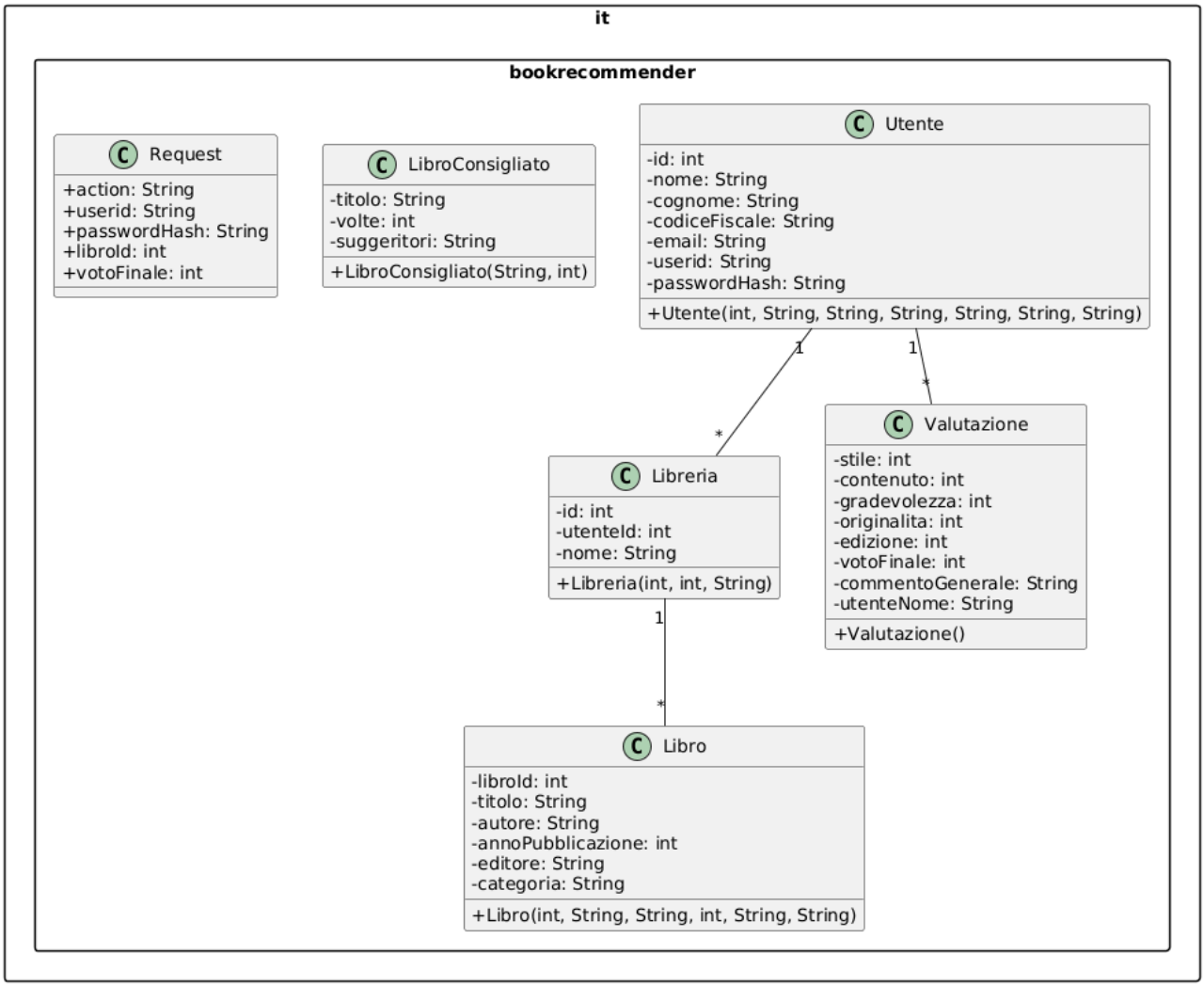



Figura 3: Entità relazionali.

## Architettura

L'applicazione adotta un'architettura distribuita multi-tier (N-Tier). La separazione tra Client e Server permette di isolare completamente l'interfaccia utente dalla logica di persistenza, garantendo scalabilità e sicurezza.

L’architettura è composta dai seguenti livelli:

1. **Presentation Layer**: GUI JavaFX con FXML, controller, Swing…

2. **Data Access Layer**: implementazioni delle entità e operazioni di persistenza.

3. **Persistence Layer (DataBase)**: il livello di persistenza dei dati, implementato utilizzando il database relazionale PostgreSQL.

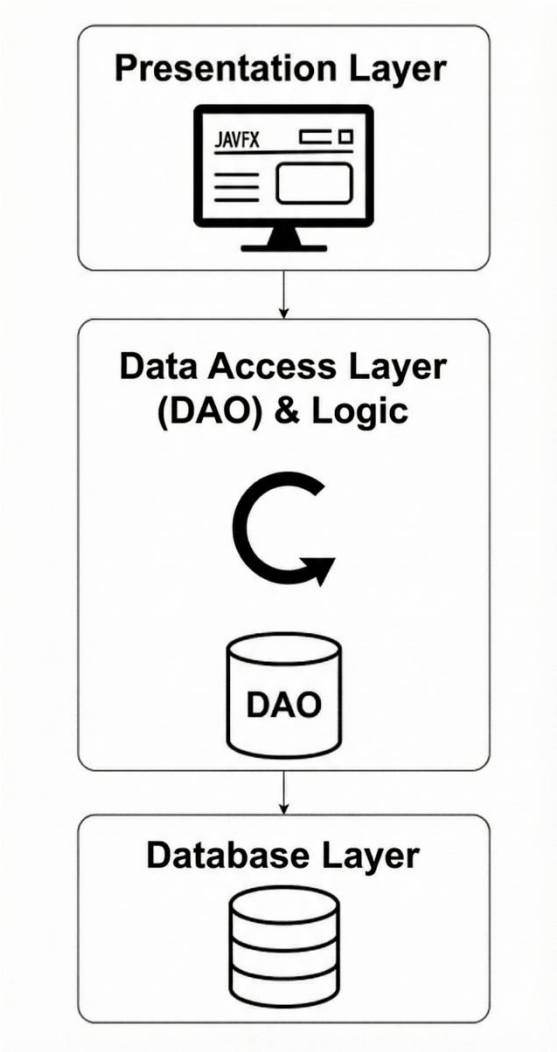
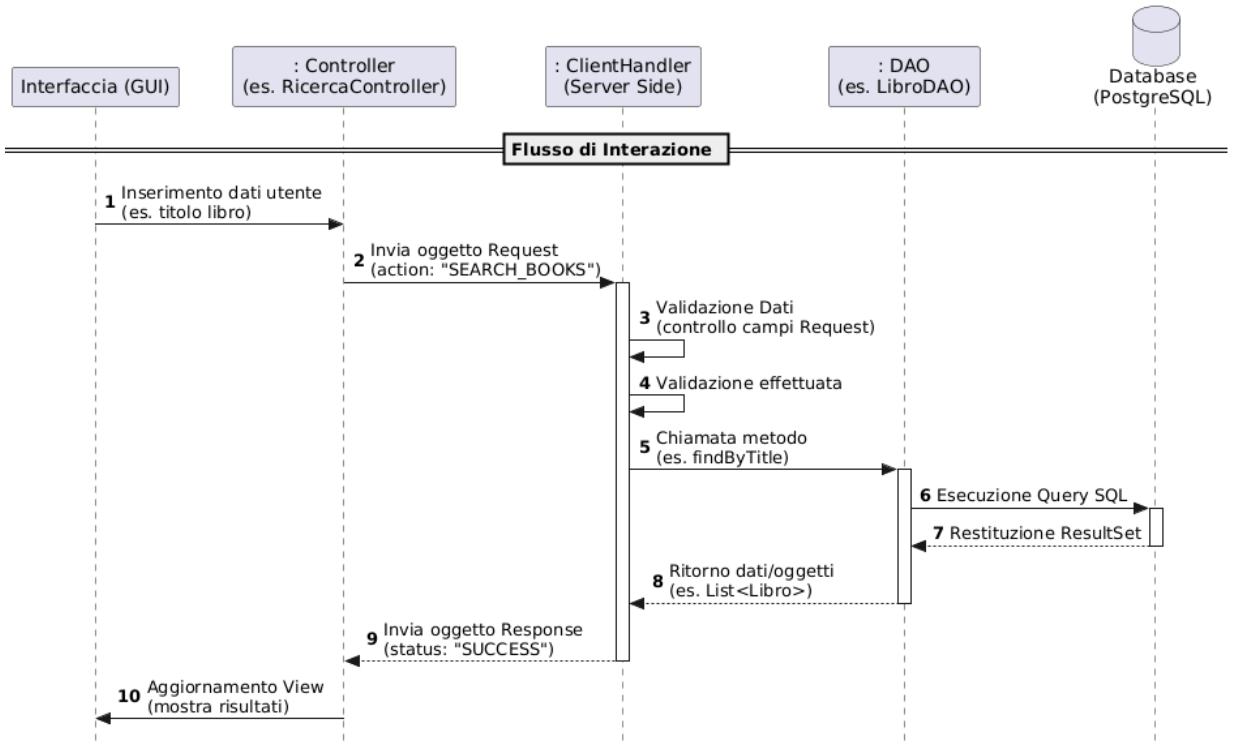


Figura 4: Architettura.

Ogni livello (ad eccezione del livello di presentazione) espone delle interfacce, ognuna delle quali viene utilizzata dalle implementazioni del livello appena superiore: il beneficio più grande di questa architettura è, infatti, che ciascun livello delega al livello sottostante le funzionalità di cui ha bisogno. Questo approccio garantisce una netta separazione delle responsabilità, facilitando così la manutenzione del sistema. Ogni livello può concentrarsi esclusivamente sul proprio compito, riducendo la complessità generale e migliorando la modularità del codice.

Figura 5: 3-Tier Architecture Sequence Diagram




## Presentation Layer

Il Presentation Layer si occupa della logica di presentazione, gestendo l'interazione diretta tra l'utente e il sistema. Per soddisfare i requisiti del progetto, è stata sviluppata un'interfaccia grafica utente (**GUI**) implementata come applicazione JavaFX, sfruttando il supporto di Scene Builder per la costruzione delle View. Questo approccio permette una netta separazione tra la struttura estetica (rappresentata in file .fxml) e la logica di gestione degli eventi, confinata nei Controller.

Ogni schermata dell'applicazione corrisponde a un file FXML gestito dal relativo controller nel modulo clientBR. I principali componenti che orchestrano la logica dell'interfaccia sono:

- **Autenticazione**: LoginController e RegisterController gestiscono l'accesso sicuro e la creazione dei nuovi profili utente.
- **Catalogo e Ricerca**: RicercaController e CatalogoController sono dedicati alla consultazione e al filtraggio dinamico del catalogo librario.
- **Collezioni Personali**: LibrerieController e LibreriaDettaglioController permettono l'organizzazione e la visualizzazione delle librerie private degli utenti.

- **Dettagli e Feedback**: VisualizzaLibroController e AggiungiValutazioneController orchestrano la visualizzazione delle schede tecniche e l'inserimento delle valutazioni analitiche.

**Componenti Grafici e Utility di Sistema**

L'interfaccia utilizza componenti standard di JavaFX (come TextField per l'input, TableView per la visualizzazione dei dati e MenuButton) integrati con utility personalizzate nel pacchetto it.bookrecommender:

- **ViewsController**: Gestisce lo switching dinamico tra le diverse scene dell'applicazione, assicurando una navigazione fluida.
- **NetworkClient**: È responsabile della comunicazione asincrona con il server, inviando oggetti Request e ricevendo i dati.
- **User Experience**: Sono presenti componenti come ParticleAnimation per arricchire l'estetica visiva e sistemi centralizzati di gestione degli alert per fornire feedback immediati sull'esito delle operazioni (es. errori di login o conferme di salvataggio).

Il Presentation Layer è responsabile dell'aggiornamento reattivo dell'interfaccia basato sul flusso di dati proveniente dal server. Una volta che il server invia un oggetto Response, il client ne estrae il contenuto e mappa i dati sugli oggetti del modello condiviso — come Libro, Valutazione e LibroConsigliato.

Grazie a questo mapping, le proprietà dei **POJO** vengono presentate graficamente nelle tabelle e nei campi di testo, permettendo all'utente di interagire con informazioni sempre sincronizzate con lo stato del database.

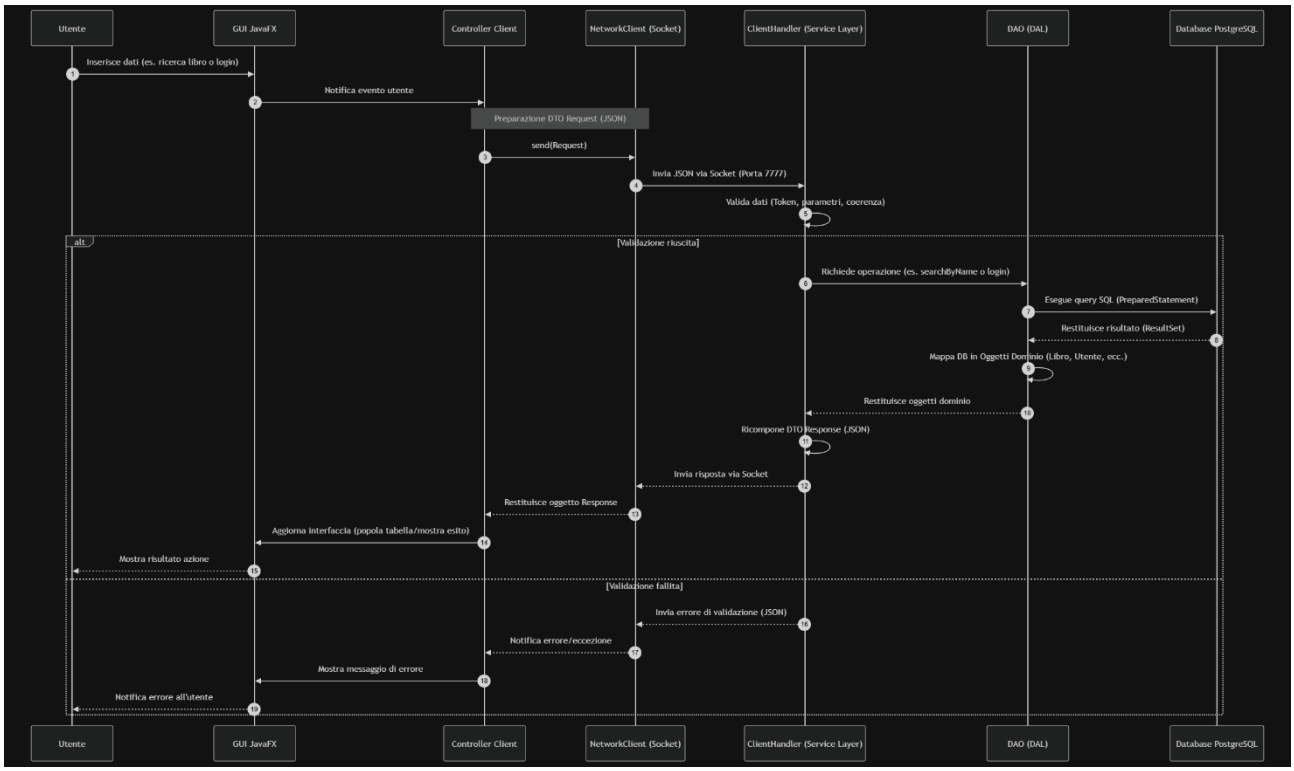



Figura 6: State Diagram che rappresenta le transizioni di stato tra le View della GUI.

Le **View** e i rispettivi controller nel progetto BookRecommender implementano le seguenti funzionalità principali:

- **login.fxml e LoginController**: Rappresentano la schermata di accesso. Consentono l'autenticazione tramite userid e passwordHash e forniscono il link alla schermata di registrazione.
- **register.fxml e RegisterController**: Gestiscono la creazione di un nuovo profilo utente, acquisendo dati quali nome, cognome, codice fiscale ed email.
- **home.fxml e HomeController**: Costituiscono il pannello principale (Area Privata) dell'utente autenticato, da cui è possibile navigare verso la ricerca, il catalogo o le proprie librerie.
- **ricerca.fxml e RicercaController**: Permettono l'interazione con il catalogo tramite filtri per titolo, autore o anno. Includono la logica per inviare oggetti Request di tipo "SEARCH_BOOKS" al server.
- **librerie.fxml e LibrerieController**: Schermata dedicata alla gestione delle collezioni personali. Permette di visualizzare l'elenco delle librerie create e di aggiungerne di nuove.

- **libreriaDettaglio.fxml e LibreriaDettaglioController**: Visualizzano i libri contenuti in una specifica libreria, interfacciandosi con i metodi del server per il recupero dei dati.
- **visualizzalibro.fxml e VisualizzaLibroController**: Mostrano la scheda tecnica del libro selezionato, integrando i dati aggregati su valutazioni medie e suggerimenti di lettura ricevuti dal server tramite l'oggetto Response.
- **aggiungiValutazione.fxml e AggiungiValutazioneController**: Forniscono il form per l'inserimento dei 5 parametri di voto e dei commenti testuali per ogni categoria (stile, contenuto, etc.)
- **catalogo.fxml e CatalogoController**: Questa View è dedicata alla visualizzazione strutturata dei volumi. Il controller gestisce il popolamento di una TableView con oggetti di tipo Libro ricevuti dal server, permettendo all'utente di scorrere il catalogo completo o i risultati filtrati.
- **consigli.fxml e ConsigliController**: Gestiscono l'interfaccia dedicata ai suggerimenti di lettura correlati. Il controller permette di visualizzare i consigli della community e consente all'utente di raccomandare nuovi volumi associandoli a un'opera target, inviando una Request con azione "ADD_CONSIGLIO" contenente l'ID del libro suggerito.

## Data Access Layer & Logic

Il **Data Access Layer (DAL**) implementa il pattern Data Access Object (DAO) e fornisce un'interfaccia astratta per l'accesso ai dati persistenti. In questa configurazione a tre livelli, lo strato non solo agisce come ponte verso PostgreSQL, ma ospita la logica di business fondamentale, garantendo che ogni operazione sui dati rispetti le regole del dominio.

Ogni DAO incapsula le query SQL e le operazioni di mapping tra le entità Java e le tabelle del database. Inoltre, il DAL si occupa della trasformazione dei dati: come mostrato nel metodo getValutazioniLibro, il livello aggrega informazioni provenienti da tabelle diverse (Join tra Valutazioni e Utenti) e le modella in oggetti pronti per la visualizzazione (es. concatenando nome e cognome dell'autore).

Per l'esecuzione di tali query, i DAO delegano la creazione della connessione alla classe DBConnection. Quest'ultima, tramite il metodo statico getConnection(), fornisce il tunnel sicuro (SSL) necessario per interagire con PostgreSQL su Render.

Questa struttura non solo facilita la manutenzione, ma centralizza la logica decisionale, riducendo la complessità del Presentation Layer che deve solo preoccuparsi di mostrare i dati già validati ed elaborati.

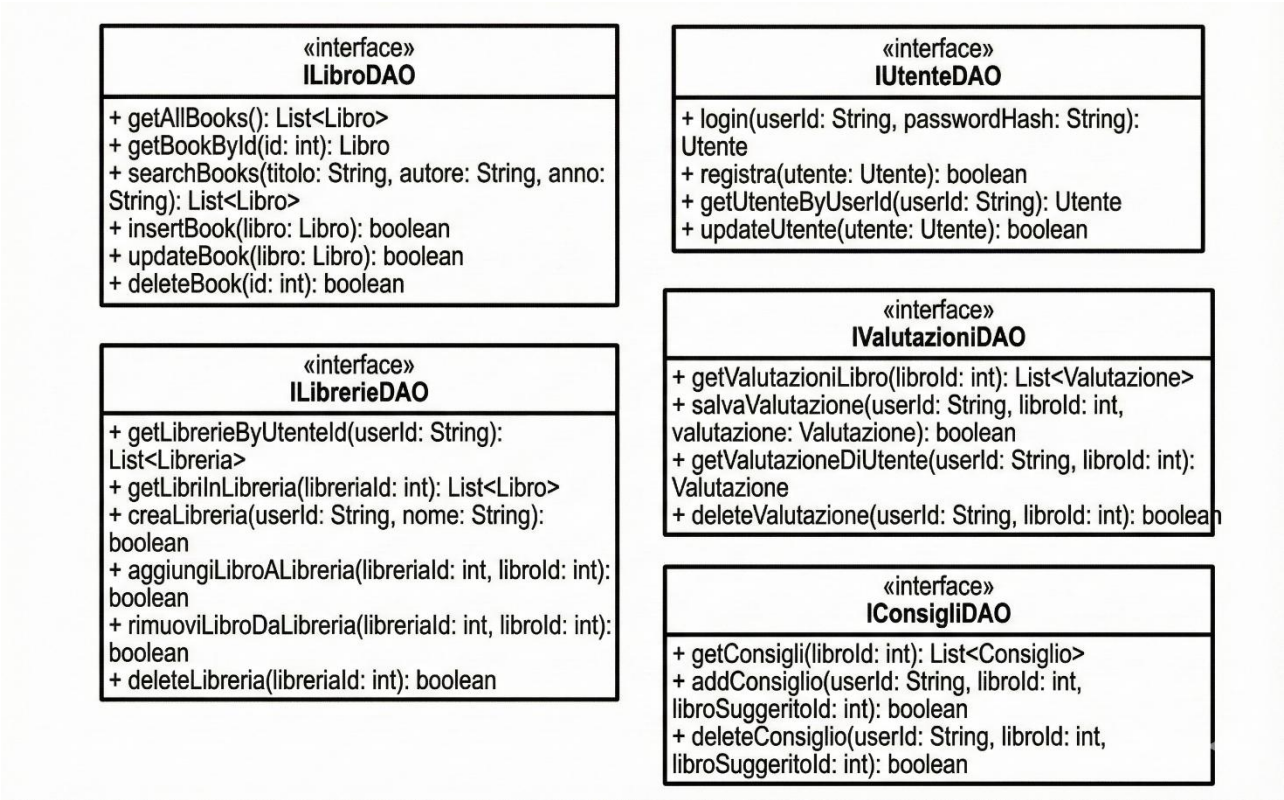


Figura 7: Struttura Data Access Layer.

Di seguito è riportato un metodo estratto dalla classe ValutazioniDAOImpl, a titolo di esempio:

```
public List<Valutazione> getValutazioniLibro(int libroId) throws Exception {
    List<Valutazione> lista = new ArrayList<>();
    // Query che unisce la tabella Valutazioni con quella degli Utenti per mostrare chi ha scritto cosa
    String sql = "SELECT v.*, u.nome, u.cognome FROM br.ValutazioniLibri v " +
            "LEFT JOIN br.utentiregistrati u ON v.utente_id = u.id WHERE v.libro_id = ?";

    try (Connection conn = DBConnection.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql)) {

      ps.setInt(1, libroId);
```

```
        ResultSet rs = ps.executeQuery();

        while (rs.next()) {
            // Costruzione dell'oggetto Valutazione mappando le colonne del DB
            Valutazione v = new Valutazione(
                rs.getInt("stile"), rs.getInt("contenuto"), rs.getInt("gradevolezza"),
                rs.getInt("originalita"), rs.getInt("edizione"), rs.getInt("voto_finale"),
                rs.getString("commento_stile"), rs.getString("commento_contenuto"),
                rs.getString("commento_gradevolezza"), rs.getString("commento_originalita"),
                rs.getString("commento_edizione"), rs.getString("commento_generale")
            );

            // Concatenazione nome e cognome per l'autore della recensione.  Il DAL si occupa di trasformare i dati "grezzi" del database (es. due colonne separate nome e cognome) in informazioni pronte per essere consumate dal Presentation Layer (un unico campo autore).
            v.setUtenteNome(rs.getString("nome") + " " + rs.getString("cognome"));
            lista.add(v);
        }
    }
    return lista;
}
```

Il DAL implementa un modello di gestione robusto che cattura le eccezioni tecniche (es. SQLException) e le trasforma in eccezioni di business comprensibili per il client. Registra i dettagli dell'errore tramite logging e restituisce messaggi appropriati al Presentation Layer, proteggendo l'utente dalla complessità tecnica del server.

## Database

Lo schema del database di BookRecommender è stato progettato per riflettere in modo puntuale ed efficace i casi d'uso dell'applicazione, garantendo al contempo robustezza e scalabilità. Le entità core, quali Libri e UtentiRegistrati, sono nettamente separate dalle azioni dinamiche dell'utente — come valutazioni, consigli e gestione delle librerie — attraverso un modello relazionale normalizzato e facilmente leggibile.

Durante la fase di progettazione il database è stato inizialmente istanziato in ambiente locale mediante PostgreSQL e pgAdmin, così da permettere una rapida iterazione sullo schema e la verifica delle funzionalità applicative.

In una fase successiva il database è stato deployato su infrastruttura cloud (Render), al fine di supportare un'architettura client–server distribuita e consentire l'accesso concorrente da più macchine, superando i limiti di un database locale legato a una singola postazione.

L'utilizzo di un database remoto consente inoltre una maggiore aderenza a scenari reali di deployment e semplifica le attività di integrazione e test del sistema.

Tutte le tabelle sono contenute nello schema:

*CREATE SCHEMA br;*

Lo schema br permette di isolare logicamente le entità dell'applicazione e migliora l'organizzazione e la manutenibilità del database.

### Integrità e Relazioni

L'integrità di entità è garantita tramite le chiavi primarie, mentre l'integrità referenziale del sistema è garantita dalla presenza di chiavi esterne (FK) esplicite che collegano logicamente ogni azione al relativo attore o oggetto. Un elemento cruciale è la tabella di giunzione librerielibri, caratterizzata da:

- **Chiave Primaria Composita**: formata dalle coppie libreriaId e libroId, assicurando che non ci siano duplicati all'interno della stessa collezione.
- **Cancellazione a Cascata**: permette una gestione fedele delle librerie personali, eliminando automaticamente i riferimenti ai libri quando una libreria viene rimossa.

### Logica Applicativa e Vincoli

Per ottimizzare le prestazioni e garantire la coerenza dei dati, parte della logica di business è stata delegata direttamente al database PostgreSQL:

- **Vincoli di Unicità**: Username, email e codice fiscale sono protetti da vincoli di unicità (UNIQUE) per rafforzare l'identità univoca dell'utente.

- **Tracciabilità Naturale**: L'uso di timestamp di default per ogni inserimento fornisce una base per ordinamenti temporali e audit, facilitando la tracciabilità per la creazione di librerie e consigli.
- Ogni tabella con **chiave primaria intera** utilizza una sequenza dedicata per la generazione automatica degli identificativi (SERIAL esplicitato) garantendo la coerenza e la corretta gestione del ciclo di vita degli identificatori, per esempio:

```
CREATE SEQUENCE br.utentiregistrati_id_seq;
AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
ALTER SEQUENCE br.utentiregistrati_id_seq OWNED BY br.utentiregistrati.id;
```

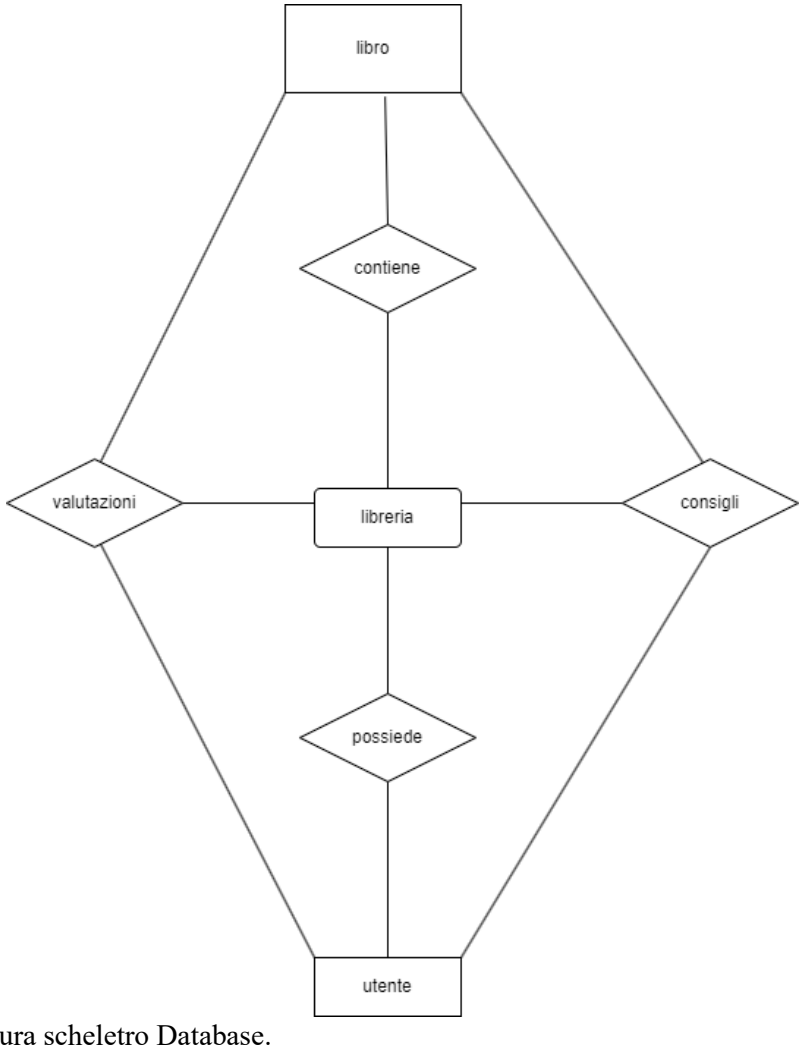


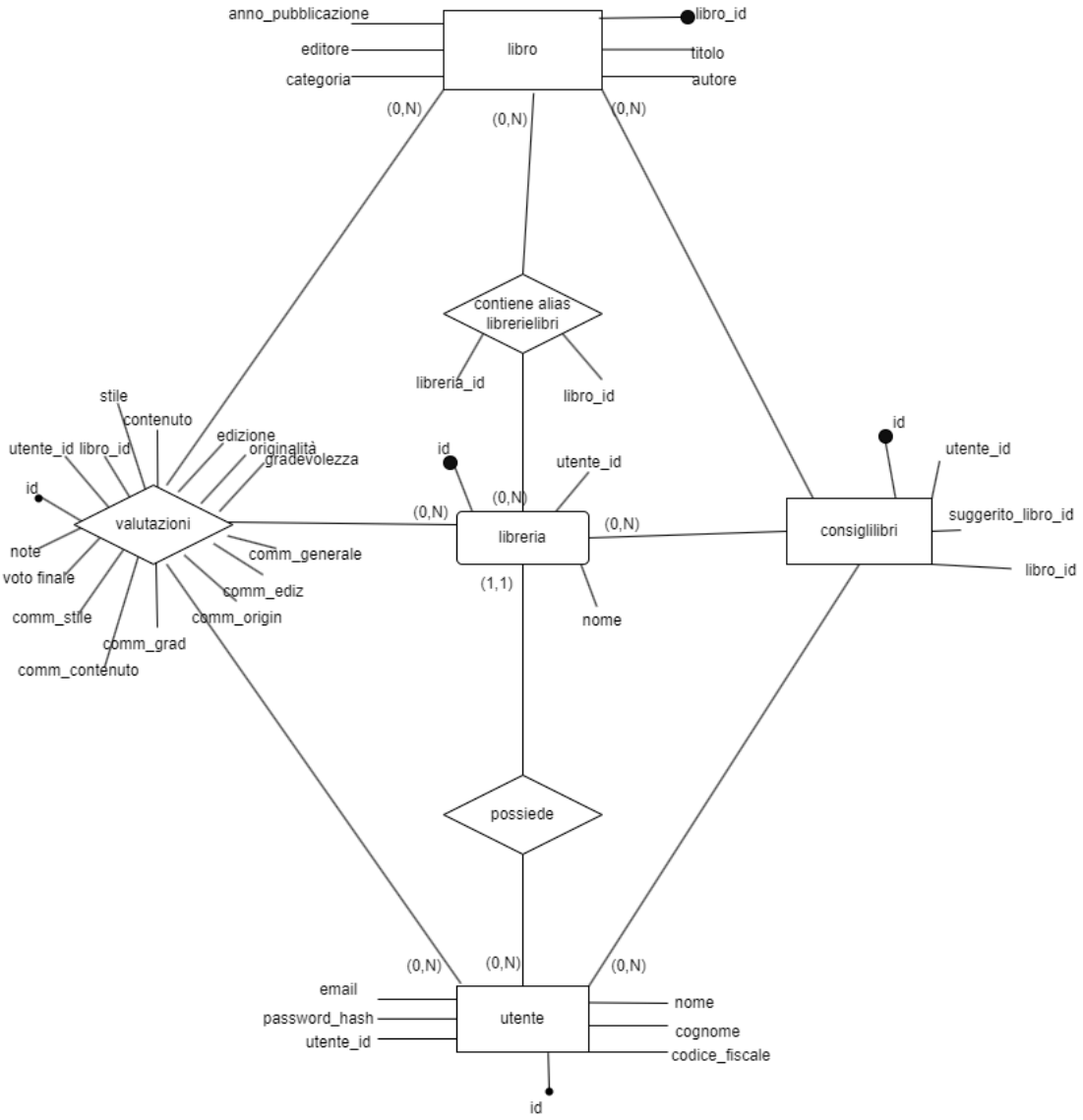

Figura 8: Struttura scheletro Database.

Le **relazioni** individuate nello schema ER sono le seguenti:

- **Possiede**: un utente registrato possiede nessuna o più librerie personali; ogni libreria appartiene a un singolo utente registrato.
- **Contiene**: una libreria contiene uno o più libri; un libro può essere contenuto in una o più librerie.
- **Valutazioni** : associazione ternaria tra Utente, Libreria e Libro. Una valutazione è espressa da un singolo utente su un singolo libro; un utente può rilasciare zero o più valutazioni, e un libro può riceverne molte da diversi utenti. I punteggi sono interi nell'intervallo [1..5] e i commenti sono ≤ 256 caratteri.
- **Consigli**: associazione ternaria tra Utente, Libreria e Libro. Un consiglio indica che un utente, nel contesto della propria libreria, suggerisce un determinato libro; uno stesso utente può inserire più consigli (max 3), e uno stesso libro può essere suggerito da più utenti. Vincoli principali: ogni tupla ConsigliLibri(user, libreria, libro, …) è ammessa solo se Possiede(user,libreria) e Contiene(libreria,libro).

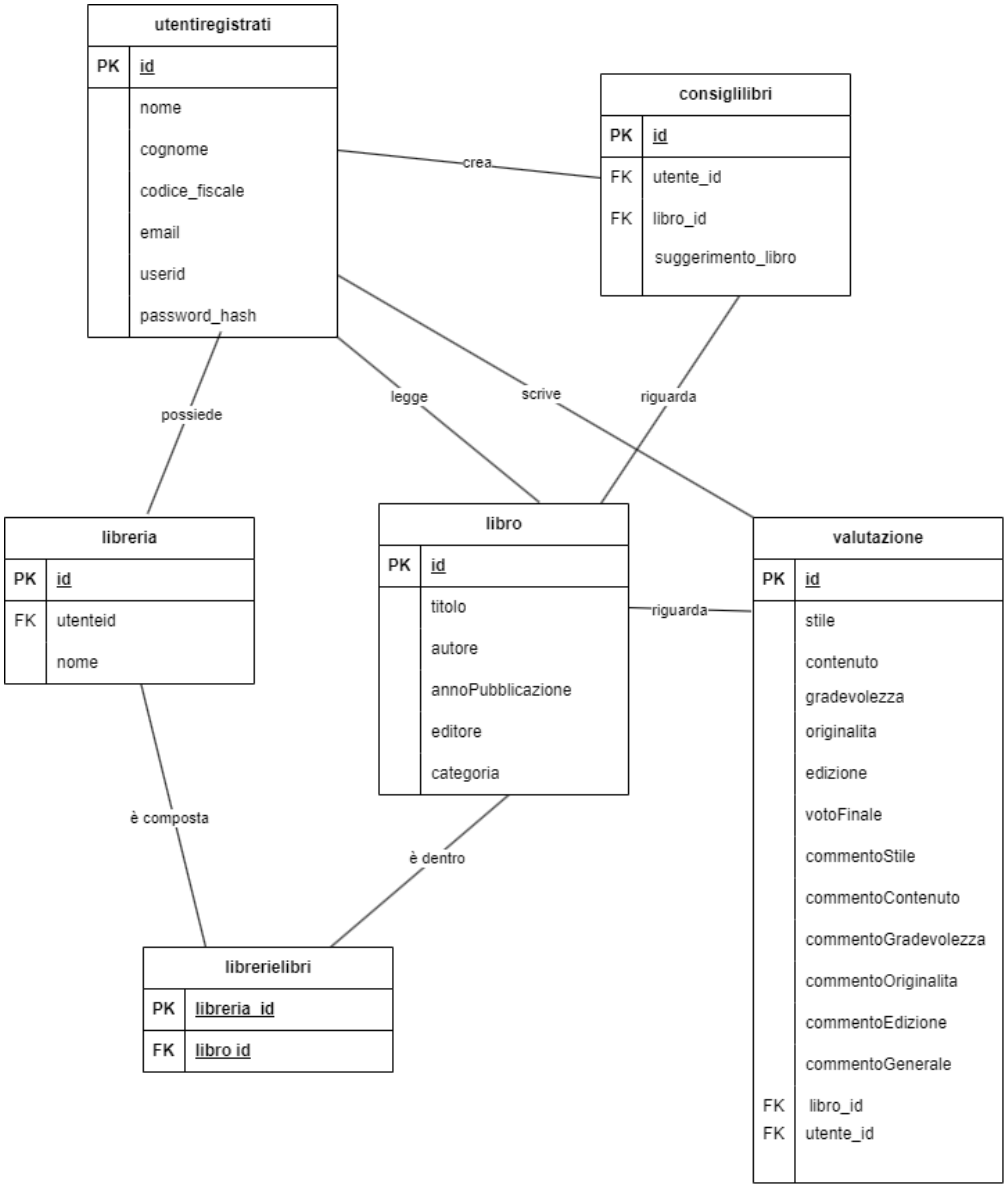
Successivamente si è passati dallo schema scheletro allo **schema concettuale** aggiungendo attributi, vincoli di cardinalità e vincoli di identificazione.

Figura 9: struttura concettuale Database




Infine è stato prodotto lo **schema logico** come traduzione dello schema concettuale nelle finali tabelle del DataBase:

Figura 10: schema logico del Database




## Package by layer

Il sistema BookRecommender adotta un approccio **"Package by Layer"** per l'organizzazione del codice, garantendo una chiara separazione dei componenti in base al loro ruolo architetturale e alla responsabilità tecnica. Questa strutturazione permette una netta divisione delle competenze e facilita sia la manutenzione che l'evoluzione del sistema attraverso il disaccoppiamento dei vari moduli.

All'interno di ciascun layer, i package sono organizzati per riflettere le aree funzionali del dominio, migliorando la leggibilità e la navigazione del progetto:

- **Presentation layer:**
  - it.bookrecommender.ui.controller
  - it.bookrecommender.net.server
- **Data access e logic:**
  - it.bookrecommender.net
  - it.bookrecommender.model
  - it.bookrecommender.dto
  - it.bookrecommender.dao.jdbc
  - it.bookrecommender.session
- **Persistence layer**
  - it.bookrecommender.db (DBConnection)

Questa strutturazione favorisce la manutenibilità e la scalabilità, permettendo ad esempio di modificare la logica di persistenza nel Data Access Layer senza impattare sul Presentation Layer, grazie all'uso di interfacce intermedie e DTO.

## Strutture dati

Nel progetto BookRecommender, la gestione dei dati si affida a strutture standard della **Java Collections Framework**, ottimizzate per le specifiche necessità di ogni layer architettonico.

Nel client JavaFX, la collezione centrale è l'ObservableList, utilizzata come modello per le componenti grafiche come le TableView (ad esempio per mostrare le liste di oggetti Libro). Rispetto a una List tradizionale, essa consente un binding reattivo: operazioni di inserimento, rimozione e ordinamento aggiornano automaticamente la UI senza dover implementare logica aggiuntiva di sincronizzazione tra il modello dati e l'interfaccia. Per l'elaborazione locale dei risultati di ricerca si utilizza tipicamente l'ArrayList, che garantisce un accesso indicizzato con complessità O(1) e costi di iterazione minimi.

Nel layer dei servizi (ClientHandler) e nel Data Access Layer (DAO), prevalgono le interfacce List e le implementazioni ArrayList come contenitori per le righe mappate dai ResultSet del database. Questa scelta è ottimale poiché la cardinalità dei dati è

variabile e l'uso dominante è la scansione sequenziale; l'overhead di una LinkedList non porterebbe benefici prestazionali e viene quindi evitato.

Per la gestione dei filtri opzionali nelle query di ricerca, vengono impiegate le **Map** (come *Map<String, Object>*), che permettono di convogliare parametri variabili in modo flessibile. Per garantire l'unicità dei dati ed evitare la duplicazione client-side (ad esempio per non proporre lo stesso libro più volte nelle liste della libreria), viene impiegata l'interfaccia Set.

Le classi di dominio condivise situate nel package it.bookrecommender (come Libro, Valutazione, Libreria, Utente) insieme ai DTO Request e Response, fungono da contenitori di dati serializzabili in formato JSON tramite la libreria **GSON** per la comunicazione su Socket. L'uso di campi tipizzati riduce la necessità di mappe basate su stringhe a favore di oggetti *type-safe*. Dove un risultato può mancare (ad esempio nel recupero dei dettagli opzionali di un libro), l'uso di Optional chiarisce l'assenza del dato a livello di API senza ricorrere a valori null.

Infine, le classi **ENUM** (come quelle ipotizzate per la gestione delle viste o delle icone) svolgono un ruolo di centralizzazione delle risorse:

- **FXMLtype**: raccoglie i riferimenti ai file FXML, evitando stringhe "hardcoded" nei controller (come LoginController o RicercaController) e rendendo la navigazione tra le schermate più sicura e leggibile.
- **IMGtype**: standardizza le risorse grafiche come icone e immagini funzionali, prevenendo errori di percorso e semplificando l'aggiornamento estetico dell'applicazione.

Queste scelte favoriscono la manutenibilità e la chiarezza del presentation layer, riducendo drasticamente la possibilità di incoerenze nell'interfaccia utente.

## Design pattern

Questa sezione elenca i principali design pattern utilizzati nel progetto, con lo scopo di fornire una panoramica architetturale delle soluzioni adottate durante la progettazione dell'applicazione. Ecco i principali design pattern utilizzati:

1. **Data Access Object (DAO)**

È il pattern predominante nel layer di persistenza. Separa la logica di accesso ai dati dalla logica di business, permettendo al resto dell'applicazione di ignorare come i dati siano effettivamente memorizzati nel database PostgreSQL.

- **Implementazione**: Classi come UtenteDAO, LibriDAO, ValutazioniDAO, LibrerieDAO e ConsigliDAO incapsulano tutte le query SQL.
- **Vantaggio**: Se si decidesse di cambiare database, basterebbe modificare queste classi senza impattare sul resto del server.

2. **Data Transfer Object (DTO)**

Questo pattern viene utilizzato per trasportare i dati tra il client e il server attraverso il socket. Invece di inviare singoli parametri, i dati vengono raggruppati in oggetti serializzabili.

- **Implementazione**: Le classi Request (che contiene l'azione e i parametri della richiesta) e Response (che contiene l'esito e i dati restituiti) sono i DTO del sistema.
- **Vantaggio**: Riduce il numero di messaggi scambiati sulla rete e rende il protocollo di comunicazione più strutturato.

3. **Front Controller / Dispatcher**

Il server utilizza un punto di ingresso unico per gestire tutte le richieste dei client.

- **Implementazione**: La classe ClientHandler agisce come un dispatcher. Riceve ogni Request JSON, analizza il campo action tramite un costrutto switch e smista l'esecuzione verso il metodo DAO appropriato.
- **Vantaggio**: Centralizza la logica di controllo, la validazione e la gestione della sicurezza.

4. **Singleton (o Static Utility)**

Viene utilizzato per gestire risorse che devono essere uniche o condivise globalmente nell'applicazione.

- **Implementazione**:
    - DBConnection: Gestisce centralmente la connessione al database.
    - BooksCache: Fornisce un unico punto di accesso in memoria al catalogo libri, caricato all'avvio del server.
- **Vantaggio**: Garantisce che non vengano sprecate risorse ricreando oggetti pesanti o connessioni multiple non necessarie.

5. **Model-View-ViewModel (MVVM)**

Utilizzato nel modulo client per gestire l'interfaccia grafica JavaFX.

- **Implementazione**: I file FXML rappresentano la View, le classi POJO (come Libro o Utente) rappresentano il Model, e i Controller (come RicercaController o LibreriaDettaglioController) fungono da ViewModel gestendo le proprietà osservabili.
- **Vantaggio**: Permette il binding dei dati tra l'interfaccia e il codice Java, assicurando che la GUI si aggiorni automaticamente quando arrivano i dati dal server.

6. **Proxy (Network Client)**

Il client non comunica direttamente con il database, ma utilizza un intermediario per le operazioni remote.

- **Implementazione**: La classe NetworkClient funge da proxy per il server. I controller chiamano i metodi del NetworkClient, che si occupa di inviare la richiesta sulla rete e restituire la risposta.
- **Vantaggio**: Nasconde la complessità della comunicazione socket ai controller della GUI.

# Algoritmi utilizzati

In questa sezione descriviamo gli algoritmi e le tecniche implementate nel progetto, spiegandone il ruolo funzionale e fornendo riferimenti al codice sorgente. Per ogni algoritmo indicheremo i file di riferimento in cui il comportamento è implementato e documentato

Ecco i principali algoritmi utilizzati:

1. **Algoritmi di Ricerca e Caching**

La ricerca del catalogo è ottimizzata tramite un approccio ibrido:

- **SQL Pattern Matching**: Nei componenti LibriDAO e UtenteDAO, la ricerca flessibile sfrutta l'operatore SQL LIKE combinato con la funzione LOWER(). Questo algoritmo di partial matching permette ricerche *case-insensitive* su titoli e autori, garantendo risultati anche con inserimenti parziali.

- **In-Memory Caching**: Per ridurre il carico sul database, la classe BooksCache implementa un algoritmo di caching all'avvio del server. Il catalogo viene caricato in una ArrayList sincronizzata, consentendo al ClientHandler di servire le richieste GET_BOOKS istantaneamente senza ulteriori query SQL.

2. **Parsing e Validazione dell'Input**

Il sistema adotta routine di controllo robuste per prevenire errori di runtime e garantire l'integrità referenziale:

- **Risoluzione degli Identificativi**: Nei DAO (come ValutazioniDAO e ConsigliDAO), prima di ogni inserimento, viene eseguito un algoritmo di risoluzione ID. Il sistema interroga la tabella UtentiRegistrati per ottenere l'ID numerico (id) a partire dallo userid testuale fornito dal client, validando l'esistenza dell'utente prima di procedere con la persistenza.
- **Parsing Sicuro**: Lato client, i controller (come RicercaController) effettuano il trim degli input e la validazione dei formati (es. anni di pubblicazione) prima di incapsulare i dati nell'oggetto Request.

3. **Serializzazione e Mapping DTO**

La comunicazione tra i moduli avviene tramite la trasformazione sistematica dei dati:

- **Mapping JSON**: Il ClientHandler utilizza la libreria GSON per implementare algoritmi di serializzazione e deserializzazione. Gli oggetti Request vengono mappati in comandi operativi, mentre i risultati estratti dai DAO vengono convertiti in oggetti Response per il trasporto su socket.
- **Conversione verso la View:** I dati restituiti (come oggetti Libro o Valutazione) vengono mappati dai controller JavaFX sulle proprietà osservabili delle TableView, garantendo che l'interfaccia rifletta sempre lo stato aggiornato del modello dati.

4. **Algoritmi di Persistenza e Aggregazione (SQL)**

Il server delega al DBMS PostgreSQL le operazioni computazionalmente più onerose tramite query strutturate:

- **Algoritmo di Raccomandazione**: ConsigliDAO e UtenteDAO implementano la logica dei suggerimenti tramite COUNT(*) e GROUP BY, ordinando i libri per frequenza di raccomandazione. Inoltre, viene utilizzato l'operatore

STRING_AGG per raggruppare i nominativi dei suggeritori in un'unica stringa descrittiva per la GUI.

- **Gestione JDBC**: Tutti i DAO seguono il pattern try-with-resources per la gestione sicura del ciclo di vita delle connessioni e dei PreparedStatement, evitando memory leak e saturazione del pool di connessioni.

5. **Sicurezza e Animazione Grafica**

- **Hashing delle Credenziali**: Il sistema implementa un algoritmo di hashing (SHA-256) per la gestione delle password. Sia in fase di login che di registra, il database confronta esclusivamente il campo *password_hash*, garantendo che le password in chiaro non vengano mai memorizzate né trasmesse se non al momento della creazione.

- **Particle Engine**: Per migliorare l'esperienza utente, è presente la classe ParticleAnimation, che gestisce un ciclo di animazione leggero basato su una ArrayList di particelle aggiornate in tempo reale, garantendo fluidità estetica senza appesantire il thread principale della UI.

# Gestione della concorrenza

Nell'applicazione non si è reso necessario introdurre meccanismi espliciti di gestione della concorrenza. Grazie all'auto-commit predefinito offerto da JDBC, ogni richiesta dei client verso il database viene trattata come transazione autonoma e affidata al sistema di gestione della concorrenza del DBMS. Inoltre, al momento della stesura del presente manuale, non sono presenti risorse condivise a livello applicativo tra i diversi client che richiedano ulteriori forme di sincronizzazione.

# Logging

"Il sistema gestisce il monitoraggio degli eventi operativi tramite messaggi sintetici e livelli di severità coerenti (**INFO** per le operazioni standard, **WARNING** e **SEVERE** per le anomalie), tracciando eventi critici quali:

- Gestione della Rete: l'accettazione di nuove connessioni sulla porta 7777 e l'avvio dei relativi thread di gestione (ClientHandler).

- Comunicazione JSON: la ricezione, il parsing delle stringhe JSON in oggetti Request e la successiva serializzazione delle Response inviate al client.

- Interazione con il DB: l'inizializzazione della connessione tramite DBConnection e l'esecuzione delle query SQL all'interno dei vari DAO.
- Errori e Eccezioni: la cattura di SQLException o interruzioni improvvise della socket, garantendo che il server rimanga operativo anche in caso di disconnessione dei client."

## Strumenti, librerie e linguaggi utilizzati

Per lo sviluppo dell'applicazione "BookRecommender" e i suoi artefatti sono stati utilizzati i seguenti **strumenti**:

- JDK SE 17
- IDE: Visual Studio Code
- Apache Maven
- PostgreSQL come DBMS.
- Gluon Scene Builder per la realizzazione delle interfacce grafiche.
- Script di utilità Windows per setup/avvio
- Microsoft Word per la scrittura di manuale tecnico e utente.
- Mermaid, draw.io e PlantUML per la realizzazione di diagrammi UML e ER
- Packaging/assembly: JAR eseguibili prodotti con plugin Maven
- Git e Github per il versioning dell'applicazione e la gestione delle varie branch dei membri del gruppo;

I **linguaggi** utilizzati sono:

- Java per la realizzazione effettiva dell'applicazione, sia client che server;
- FXML per le view JavaFX
- CSS per lo styling dei componenti della GUI;
- SQL per la definizione e interrogazione del database
- Batch (.bat) per script Windows Librerie e plugin

• JavaFX (org.openjfx: javafx-controls, javafx-fxml, javafx-graphics, javafx-base), versione 17.0.2 • PostgreSQL JDBC Driver (org.postgresql:postgresql), versione 42.7.3

• Log4j2 (org.apache.logging.log4j:log4j-core e log4j-api), versione 2.20.0

• JFoenix (componenti Material Design per JavaFX), versione 9.0.10

• FontAwesomeFX (de.jensd:fontawesomefx-fontawesome), versione 4.7.0-9.1.2

• Maven plugins principali: o maven-compiler-plugin, versione 3.11.0 (configurato per Java 17). o maven-shade-plugin, versione 3.4.1 o maven-assembly-plugin, versione 3.7.0

# Limiti dell'applicazione e conclusioni

Sebbene l'applicazione sia stata creata con l'intento di renderla il più similare possibile ad un'applicazione di livello enterprise, resta sostanzialmente un progetto con finalità didattica. Presenta pertanto le limitazioni discusse in precedenza. Nel complesso il lavoro fornisce una base per interventi incrementali e non invasivi che ne miglioreranno scalabilità e manutenzione senza stravolgere l'architettura.

# Bibliografia


• Oracle. Java Platform, Standard Edition 17 Documentation. Disponibile su: https://docs.oracle.com/en/java/javase/17

• Oracle. JavaFX Documentation. Disponibile su: https://openjfx.io

• Oracle. Java Remote Method Invocation (RMI) – Guida ufficiale. Disponibile su: https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/index.html

• PostgreSQL Global Development Group. PostgreSQL Documentation (versione 14+). Disponibile su: https://www.postgresql.org/docs/

• Oracle. Java Logging Overview (java.util.logging). Disponibile su: https://docs.oracle.com/javase/8/docs/technotes/guides/logging/

• GitHub, Inc. GitHub Documentation – Best practices for repositories, branching e collaboration. Disponibile su: https://docs.github.com

• Fowler, M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

• Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

• Freeman, E., Freeman, E. Head First Design Patterns. O'Reilly Media, 2004.