PROGETTO LABORATORIO B
BOOKRECOMMENDER

Manuale Tecnico

Autore: *Matteo Ferrario, Ionut Puiu, Richard Zefi*

Anno Accademico: *2025/2026*

Sommario

1. Introduzione

Il presente documento tecnico si prefigge lo scopo di illustrare in maniera esaustiva le specifiche funzionali, le scelte architettoniche e le soluzioni implementative adottate durante l'intero ciclo di sviluppo del sistema software denominato Book Recommender.

L'applicazione è stata concepita e realizzata con l'obiettivo primario di offrire una piattaforma per la gestione centralizzata e collaborativa di un vasto catalogo librario. Le funzionalità principali messe a disposizione dell'utenza includono la gestione completa del ciclo di vita dell'account (dalla registrazione al login), la possibilità di organizzare le proprie preferenze letterarie tramite la creazione di librerie personali customizzabili, e un avanzato sistema di feedback che consente la valutazione delle opere secondo metriche qualitative specifiche. Inoltre, il sistema promuove l'interazione sociale attraverso un meccanismo dedicato allo scambio di suggerimenti di lettura tra gli iscritti.

Sotto il profilo infrastrutturale, il sistema è stato progettato per operare efficacemente in un ambiente distribuito, basato su un'architettura di rete che separa la logica di presentazione da quella di business. L'implementazione garantisce la robusta persistenza dei dati su supporti relazionali e assicura la corretta gestione dell'accesso concorrente alle risorse condivise, preservando l'integrità delle informazioni anche a fronte di molteplici richieste simultanee.

2. Premessa

L'architettura fondamentale del sistema è stata progettata e sviluppata aderendo rigorosamente al paradigma Client-Server, separando nettamente la logica di presentazione da quella di elaborazione dati.

Per quanto concerne il livello di comunicazione, l'interazione tra le componenti avviene attraverso connessioni socket basate sul protocollo di trasporto TCP/IP, garantendo affidabilità nella trasmissione. Al livello applicativo, è stato implementato un protocollo proprietario (custom) progettato specificamente per questo dominio, il quale si basa sullo scambio diretto di oggetti Java tramite il meccanismo nativo della serializzazione, permettendo il trasferimento di strutture dati complesse in modo trasparente.

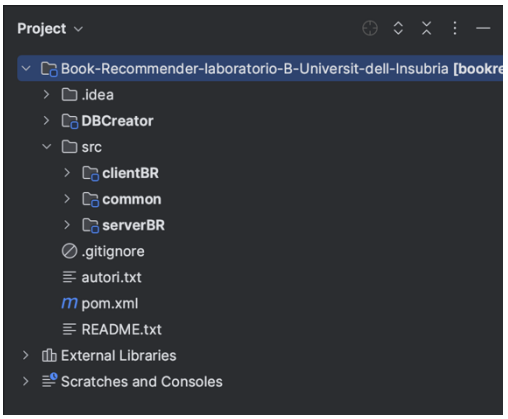
L'intero stack tecnologico è stato realizzato utilizzando il linguaggio Java, adottando la versione JDK 17 (Long Term Support) per garantire stabilità e moderne funzionalità. Questa scelta ha permesso di mantenere uniformità di linguaggio sia per il backend lato server, sia per il frontend lato client, la cui interfaccia grafica è stata costruita mediante la libreria JavaFX.

La gestione della persistenza è affidata al RDBMS PostgreSQL, scelto per la sua robustezza e conformità agli standard SQL. L'interazione con la base di dati è gestita tramite l'utilizzo diretto del driver JDBC (Java Database Connectivity). L'architettura privilegia l'uso delle API standard Java per la manipolazione dei dati, evitando l'introduzione di layer di astrazione esterni (ORM). Questa impostazione consente di gestire la logica di mapping SQL-Oggetto in modo esplicito all'interno delle classi Repository, mantenendo il codice aderente alle specifiche fondamentali del protocollo JDBC.

3. Software Design

3.1 Struttura dell'applicazione

La gestione del ciclo di vita del software e delle dipendenze esterne è stata affidata ad Apache Maven. Al fine di garantire un'architettura scalabile, manutenibile ed estendibile, si è optato per una suddivisione del progetto in moduli logici distinti. Questa strategia di progettazione favorisce un netto disaccoppiamento tra le componenti, isolando le responsabilità e facilitando la navigazione all'interno della codebase.



Struttura gerarchica dei moduli Maven.

L'ecosistema del progetto si articola nei seguenti moduli principali:

- common (Libreria Condivisa): Rappresenta il nucleo trasversale dell'applicazione, contenente le risorse fondamentali condivise tra le entità client e server. Questo modulo definisce il "contratto" di comunicazione e la struttura dati unificata. Al suo interno risiedono:

    - I Modelli di Dominio (POJO): Le classi che mappano le entità del sistema, quali User, Book, Review, Library e Suggestion.
    - Protocollo di Rete: Le classi necessarie alla serializzazione e allo scambio dei messaggi, nello specifico gli oggetti wrapper Request e Response, unitamente all'enumerativo RequestType che tipizza le operazioni consentite.
    - Networking: La classe BRProxy, che funge da intermediario, gestendo l'incapsulamento di basso livello della connessione socket e rendendo trasparente la comunicazione di rete.
- serverBR (Core Backend): Costituisce il motore computazionale dell'architettura. Questo modulo accentra l'intera logica di business e la gestione della persistenza. Le sue componenti chiave includono:
    - Data Access Layer: I Repository dedicati all'interazione con il database e la classe Db per la gestione delle connessioni JDBC.
    - Service Layer: I servizi applicativi (come AuthService e SearchService) che implementano le regole di business.
    - Entry Point: La classe MainServer, responsabile dell'avvio del servizio e dell'accettazione delle connessioni in ingresso.
- clientBR (Frontend & Presentation): Modulo dedicato all'interazione con l'utente finale. Esso implementa l'interfaccia grafica basata sulla tecnologia JavaFX e la logica di presentazione. Include i Controller grafici che gestiscono gli eventi della UI e i servizi lato client (speculari a quelli server, es. AuthService client) che hanno il compito di tradurre le azioni dell'utente in richieste di rete, inoltrandole tramite il proxy.
- DBCreator (Database Utility): Un modulo ausiliario e indipendente, progettato per l'inizializzazione dell'ambiente dati. Contiene gli script DDL (1-schema.sql) per la creazione dello schema relazionale e le routine Java necessarie per il popolamento iniziale o il ripristino del database, garantendo la riproducibilità dell'ambiente di esecuzione.

3.2 Connessione al database

La responsabilità di stabilire e gestire l'interfaccia di comunicazione con il sistema di persistenza è centralizzata all'interno della classe bookrecommender.db.Db, situata nel core del modulo server.

Durante la fase di bootstrap (avvio) dell'applicazione, è implementata una procedura di configurazione interattiva che richiede all'amministratore di sistema di fornire esplicitamente i parametri essenziali per l'accesso al database: indirizzo host, porta di ascolto, nome del database e credenziali di

autenticazione (username e password). Questi dati vengono elaborati in tempo reale per costruire dinamicamente la *Connection String* JDBC, seguendo il formato standard:

```
jdbc:postgresql://[host]:[port]/[dbName]?currentSchema=br
```

Un dettaglio architetturale rilevante è l'impostazione del parametro currentSchema=br, che vincola tutte le query successive ad operare all'interno dello schema logico denominato br, garantendo un corretto isolamento delle tabelle.

Per quanto concerne la strategia di gestione delle risorse, il sistema adotta un approccio *on-demand* semplificato: non viene utilizzato un meccanismo di *Connection Pooling* (pool di connessioni). Al contrario, l'architettura prevede che venga istanziata una nuova connessione fisica dedicata per ogni singola operazione atomica o transazione richiesta dai Repository, assicurando che la connessione venga aperta e chiusa puntualmente al termine dell'esecuzione.

3.3 Traduzione entità relazionali in entità Java

L'architettura del sistema organizza la gestione della persistenza applicando il Pattern DAO (Data Access Object). Questa struttura consente di incapsulare l'accesso ai dati, separando nettamente la logica applicativa dalle operazioni di basso livello sul database.

Di conseguenza, il meccanismo di mappatura tra le tuple delle tabelle relazionali e gli oggetti del dominio Java è stato implementato esplicitamente all'interno delle classi Repository. Sfruttando le funzionalità native e standard del protocollo JDBC, il codice gestisce direttamente la traduzione dei tipi di dati SQL nei corrispondenti attributi Java, assicurando il corretto popolamento delle entità e il controllo completo sul flusso dei dati.

Il processo di conversione dei dati è gestito attraverso due fasi distinte:

- Fase di Lettura (Data Retrieval): Durante le operazioni di interrogazione, i risultati restituiti dal database sotto forma di ResultSet JDBC vengono iterati sequenzialmente. I dati primitivi estratti dalle singole colonne vengono quindi utilizzati per istanziare e popolare le classi del modello di dominio (quali User o Book), invocando esplicitamente i rispettivi costruttori parametrizzati.
- Fase di Scrittura (Data Persistence): Nelle operazioni di inserimento o aggiornamento, avviene il processo inverso: gli oggetti Java complessi vengono decomposti nei loro attributi fondamentali. Tali valori vengono successivamente iniettati come parametri all'interno

dei PreparedStatement. Questa pratica non solo assicura il corretto mapping dei tipi, ma garantisce anche la sicurezza del sistema prevenendo vulnerabilità legate alla SQL Injection.

Questa precisa scelta progettuale offre vantaggi tangibili in termini di ottimizzazione delle performance, eliminando l'overhead computazionale e di memoria spesso introdotto dagli strati di astrazione degli ORM. Inoltre, l'approccio manuale garantisce la massima flessibilità nell'utilizzo di funzionalità native e specifiche del motore PostgreSQL, permettendo l'uso di operatori avanzati come ILIKE per le ricerche testuali o la clausola ON CONFLICT per la gestione efficiente degli inserimenti condizionali (upsert).

3.4 Architettura

L'architettura logica del sistema è stata definita conformemente al pattern Multilayer, articolandosi in quattro livelli gerarchici distinti. Questa organizzazione strutturale risponde al principio della separazione delle responsabilità (Separation of Concerns), garantendo un elevato grado di modularità e facilitando la manutenibilità e l'evoluzione futura dell'intero ecosistema software.

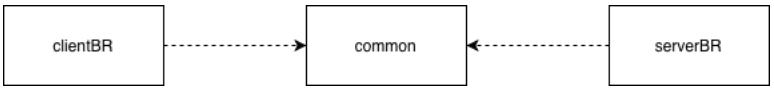
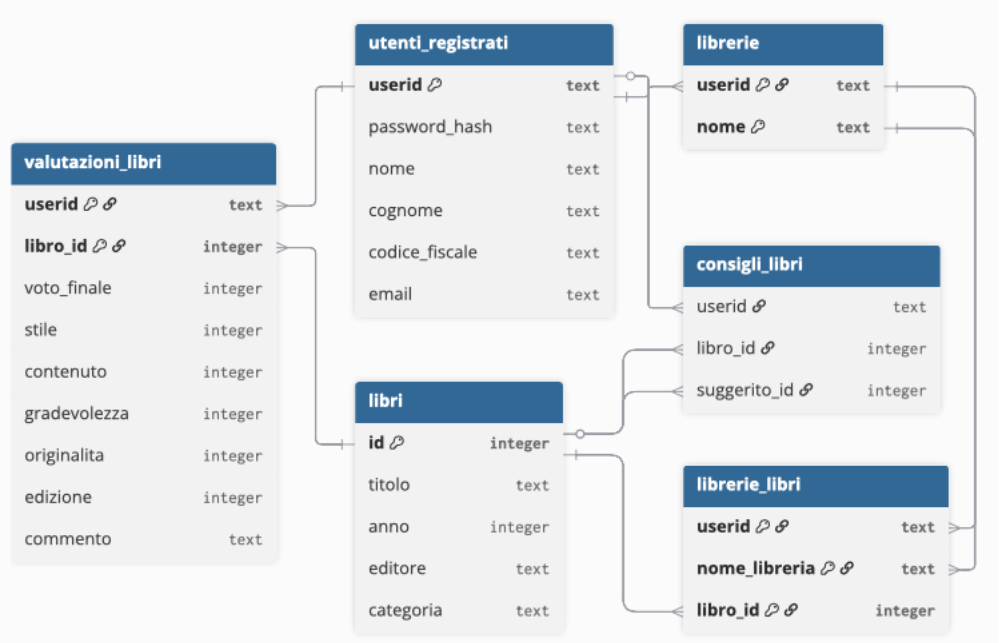


Diagramma dei package e dipendenze tra i moduli.

3.4.1 Persistence Layer

La persistenza delle informazioni è affidata al RDBMS PostgreSQL. La struttura del database è definita formalmente nello script DDL 1-schema.sql, che contiene le istruzioni necessarie per la creazione dello schema fisico, delle tabelle e dei relativi vincoli di integrità.

Schema E-R del database.

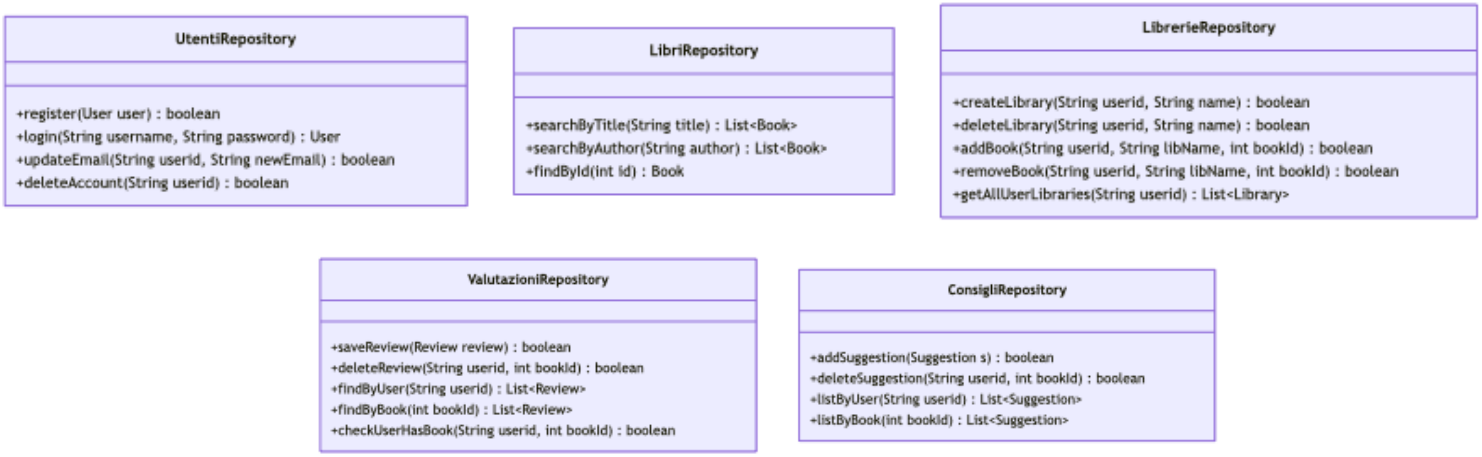


Il modello dati è strutturato attorno a un nucleo di tabelle relazionali progettate per garantire l'integrità referenziale e l'efficienza delle query. Le principali entità che compongono lo schema sono:

- utenti_registrati: Rappresenta l'anagrafica centralizzata degli utenti del sistema. Oltre ai dati identificativi, la tabella gestisce la sicurezza delle credenziali memorizzando esclusivamente l'hash della password (generato tramite algoritmo crittografico SHA-256), evitando così il salvataggio in chiaro di informazioni sensibili.
- valutazioni_libri: È una tabella associativa che risolve la relazione molti-a-molti tra utenti e libri. Essa non si limita a un singolo voto, ma storicizza una valutazione multidimensionale composta da 5 metriche specifiche (Stile, Contenuto, Gradevolezza, Originalità, Edizione), arricchita da un campo testuale per la recensione discorsiva.
- librerie: Definisce i contenitori logici (o "scaffali virtuali") creati dinamicamente dagli utenti per organizzare la propria collezione (es. "Preferiti", "Da Leggere").
- librerie_libri: Tabella di relazione che mappa l'appartenenza dei singoli volumi alle specifiche librerie personali, permettendo a un libro di essere presente in molteplici collezioni (relazione one-to-many rispetto alla libreria).
- consigli_libri: Implementa la logica sociale del sistema, mappando una tripla relazione: l'utente che effettua il suggerimento, il libro di origine (letto o apprezzato) e il libro target suggerito, creando una rete di raccomandazioni user-generated.

3.4.2 Data Access Layer

La gestione della persistenza e l'interazione diretta con il database sono rigorosamente incapsulate all'interno del modulo serverBR. L'architettura adotta il Pattern Repository, il quale fornisce un'astrazione orientata agli oggetti sui dati relazionali. Le classi Repository espongono un'interfaccia pubblica di alto livello per la manipolazione delle entità di dominio, nascondendo completamente ai livelli superiori (Service Layer) la complessità sintattica delle query SQL sottostanti e i dettagli implementativi del driver JDBC.



Classi del Data Access Layer.

Le componenti implementate in questo livello sono:

- LibriRepository: Responsabile dell'interrogazione del catalogo letterario. Fornisce metodi ottimizzati per la ricerca testuale (per titolo o autore) e per il recupero puntuale dei metadati associati alle singole opere.
- UtentiRepository: Centralizza la logica di gestione dell'identità. Si occupa delle operazioni CRUD relative agli account utente, gestendo in sicurezza la registrazione, la verifica delle credenziali in fase di login e l'aggiornamento dei dati di profilo.
- ValutazioniRepository: Amministra la persistenza dei feedback utente. Gestisce sia la scrittura delle nuove recensioni (comprese le valutazioni metriche dimensionali), sia le query analitiche per il recupero dello storico valutazioni per utente o per libro.
- LibrerieRepository: Governa la logica delle collezioni personali. Permette la creazione e la modifica delle librerie virtuali customizzate, gestendo le relazioni tra l'utente proprietario e i volumi inseriti.
- ConsigliRepository: Gestisce le entità relazionali legate al sistema di raccomandazione, occupandosi dell'inserimento dei nuovi suggerimenti e del recupero delle liste di lettura proposte dalla community.

3.4.3 Service Layer

Questo livello rappresenta il nucleo decisionale del sistema, fungendo da intermediario tra l'interfaccia utente (o il livello di rete) e lo strato di persistenza dei dati. La sua implementazione è distribuita e specializzata a seconda del modulo di appartenenza:

- Lato Server (serverBR - Core Business Logic): In questo contesto, le classi di servizio (quali AuthService, SearchService) assumono il ruolo di orchestratori. Esse incapsulano la logica di business vera e propria, coordinando le chiamate verso i vari Repository per soddisfare le richieste pervenute. È in questo livello che vengono applicate le regole di dominio (es. validazione delle credenziali, verifica della disponibilità di un libro, calcoli statistici sulle valutazioni) prima di rendere permanenti le modifiche nel database.
- Lato Client (clientBR - Service Adapters): I servizi presenti nel modulo client ricoprono un ruolo strutturale differente: non implementano logica di business complessa (che rimane centralizzata sul server per sicurezza), ma agiscono come adattatori. Il loro compito è disaccoppiare i Controller grafici dalla complessità della comunicazione di rete. Essi prendono i dati inseriti dall'utente nell'interfaccia, li incapsulano all'interno degli oggetti standard Request e delegano la trasmissione al layer di rete (BRProxy), attendendo e restituendo poi la Response elaborata al controller chiamante.

Al fine di ridurre il numero di richieste ridondanti verso il server e migliorare le prestazioni percepite dall'utente, il client implementa una cache locale dei libri già recuperati. Tale struttura consente il riutilizzo dei dati precedentemente ottenuti, evitando interrogazioni ripetute a parità di contenuto.

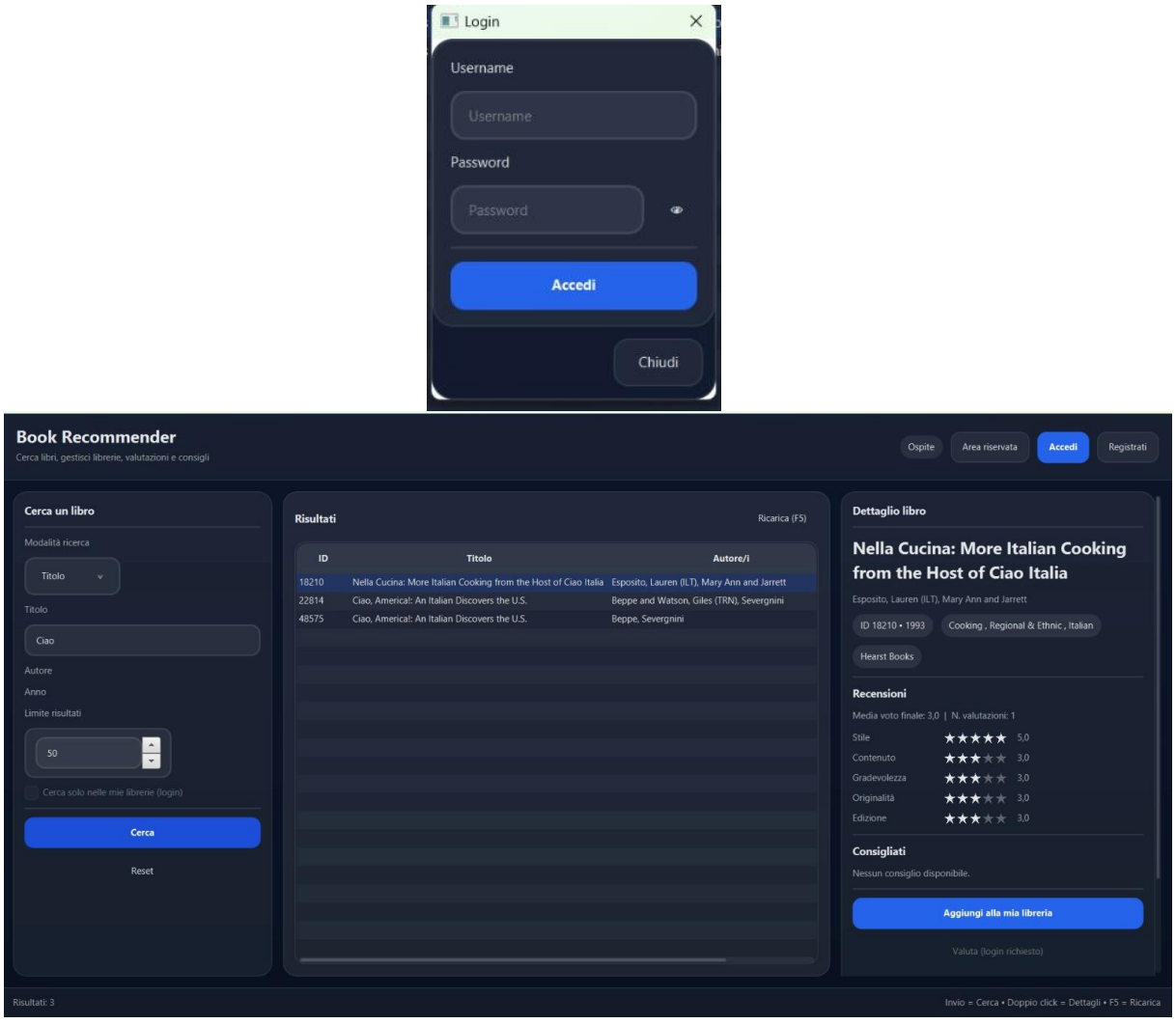
Nel contesto del sistema di raccomandazione, i suggerimenti vengono inizialmente gestiti tramite identificativi; il recupero dei metadati mancanti (come il titolo del libro consigliato) avviene esclusivamente quando necessario, e solo se l'informazione non risulta già presente in cache.

Questo approccio permette di minimizzare il traffico di rete, preservando la coerenza dei dati e garantendo una navigazione fluida dell'interfaccia, senza introdurre dipendenze logiche tra il livello di presentazione e quello di persistenza.

3.4.4 Presentation Layer

L'interfaccia grafica dell'applicazione è stata realizzata interamente mediante il framework JavaFX, residente all'interno del modulo clientBR.

Per la costruzione delle schermate, si è optato per un approccio programmatico (Java-based code) anziché dichiarativo. Le singole viste non sono file FXML statici, bensì classi Java che estendono dinamicamente i container grafici standard della libreria (quali BorderPane, VBox, GridPane). Questa strategia garantisce un controllo granulare sulla creazione dei componenti a runtime. Tuttavia, per garantire una netta separazione tra struttura e stile, l'aspetto estetico (colori, font, spaziature) è stato disaccoppiato dalla logica e delegato a un foglio di stile esterno CSS (app.css), facilitando la manutenzione del look & feel.



Interfaccia utente dell'applicazione.

Il punto nevralgico dell'interazione utente è la classe BookRecommenderFX. Questo componente agisce come Main Controller e orchestratore della navigazione: esso gestisce il routing tra le diverse

viste funzionali (Schermata di Login, Ricerca Catalogo, Profilo Utente, Gestione Librerie) scambiando i contenuti nel container principale. Inoltre, è responsabile del mantenimento dello stato della sessione (Session State), conservando in memoria i dati dell'utente autenticato per garantire la persistenza dell'identità durante l'intera sessione di lavoro.

L'interfaccia grafica del client adotta un modello *state-driven*, in cui la visualizzazione dei componenti dipende dallo stato corrente dell'applicazione. In assenza di una selezione attiva, il pannello di dettaglio mostra un messaggio informativo che invita l'utente a selezionare un libro dal catalogo.

Al variare dello stato (login/logout, selezione o deselezione di un libro), la UI viene aggiornata dinamicamente senza ricostruzioni distruttive delle viste. Il pannello dei dettagli visualizza informazioni, valutazioni e suggerimenti solo quando i dati risultano effettivamente disponibili, evitando contenuti duplicati o incoerenti.

Le interazioni con i libri consigliati sono integrate con la tabella del catalogo: la selezione di un elemento suggerito apre il relativo dettaglio, sincronizzando la navigazione tra le componenti dell'interfaccia.

3.5 Strutture dati

Il meccanismo di interscambio informativo tra Client e Server è stato ingegnerizzato attraverso un protocollo proprietario basato sulla serializzazione nativa Java. Le strutture dati fondamentali (DTO - Data Transfer Objects) sono definite all'interno del modulo condiviso common, garantendo la perfetta interoperabilità semantica tra i nodi della rete.

Il protocollo si fonda su due entità primarie che standardizzano il flusso di comunicazione:

- La Classe Request (Richiesta): Agisce come envelope (busta) per tutte le invocazioni remote verso il server. La sua architettura interna incapsula due elementi chiave:
    - RequestType: Un enumerativo che definisce l'intento semantico dell'operazione (es. LOGIN, SEARCH_BY_TITLE), fungendo da discriminante per il dispatching della richiesta al servizio corretto.
    - Payload: Un campo generico (Object) progettato per trasportare il contenuto informativo variabile necessario all'esecuzione del comando (sia esso una stringa di ricerca, un intero ID o un'entità complessa come un oggetto User).

- La Classe Response (Risposta): Standardizza l'output del server, fornendo al client un contratto di risposta uniforme indipendentemente dall'operazione eseguita. Ogni oggetto di risposta aggrega lo stato dell'esecuzione (tramite un flag booleano di successo/errore) e il risultato effettivo dell'elaborazione (o l'eventuale messaggio di eccezione), semplificando la logica di gestione degli errori lato client.

3.6 Design Pattern

Al fine di garantire un'architettura software robusta, modulare e facilmente manutenibile, lo sviluppo del sistema è stato guidato dall'applicazione di consolidati Design Patterns (GoF). Le principali soluzioni adottate includono:

- Repository Pattern: Impiegato nel livello di accesso ai dati per realizzare un netto disaccoppiamento tra la logica di business e i meccanismi di persistenza. Grazie a questo pattern, il resto dell'applicazione interagisce con il database attraverso un'interfaccia orientata agli oggetti pulita, ignorando completamente la complessità delle query SQL e le specifiche del driver JDBC sottostante.
- Proxy Pattern: Implementato attraverso la classe BRProxy, questo pattern funge da intermediario locale (Surrogate) per le risorse remote. Il suo ruolo è cruciale per garantire la trasparenza di rete: i servizi lato client invocano le operazioni su questo oggetto locale, il quale si fa carico, "dietro le quinte", di gestire l'apertura delle connessioni socket e la trasmissione dei dati, rendendo la distribuzione fisica del server invisibile ai livelli superiori del client.
- Command Pattern: Realizzato mediante la strutturazione degli oggetti Request e l'uso dell'enumerativo RequestType. Questo approccio permette di incapsulare ogni richiesta operativa (es. login, ricerca, voto) all'interno di un oggetto autonomo. Ciò consente di trattare l'azione da compiere come un parametro trasferibile, facilitando il dispatching e l'esecuzione flessibile delle operazioni sul lato server senza dover creare endpoint dedicati rigidi.

4. Algoritmi

4.1 Ricerca per titolo o autore

La logica di Information Retrieval (recupero delle informazioni) implementata per la ricerca dei libri non effettua elaborazioni in memoria lato applicativo, bensì delega interamente l'onere computazionale al motore di database PostgreSQL.

L'operazione si basa sull'utilizzo dell'operatore SQL nativo ILIKE, il quale permette di effettuare un pattern matchingflessibile e case-insensitive (insensibile alle maiuscole/minuscole) sulle colonne target (titolo e autori).

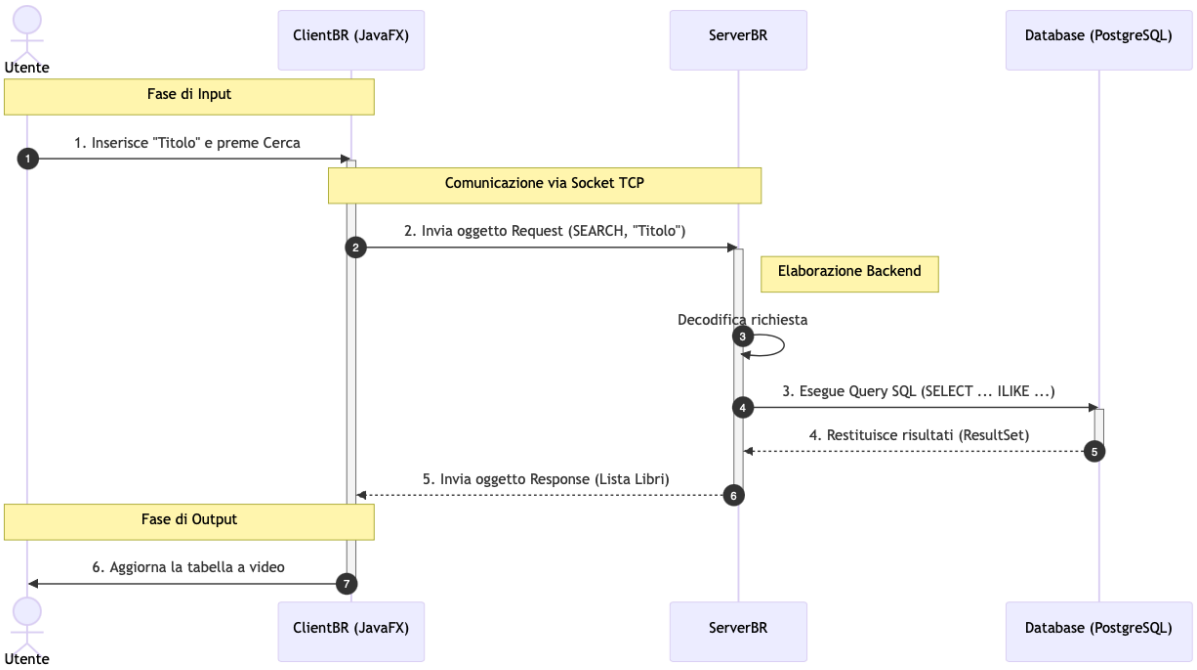
Sotto il profilo delle performance, è necessario analizzare il piano di esecuzione (Query Plan). In assenza di indici specifici per la ricerca Full-Text (quali indici invertiti GIN o GiST), il Query Planner del database è costretto ad adottare una strategia di Scansione Sequenziale (Sequential Scan). Ciò implica che il motore deve scorrere ed esaminare ogni singola tupla presente nella tabella per verificare la corrispondenza con la stringa di input.

Di conseguenza, la Complessità Asintotica dell'algoritmo è classificabile come Lineare, ovvero di ordine:

- Dove rappresenta la cardinalità totale (numero di righe) della relazione libri O(n), $T(n) = k \cdot n + c$ dove: n = dimensione input, k = costo per elemento, c = costo fisso.
- Questo comporta che il tempo di esecuzione cresce in maniera direttamente proporzionale all'aumentare delle dimensioni del catalogo gestito.

4.2 Ricerca per autore e anno

Questa funzionalità implementa una strategia di filtraggio multicriterio dinamico. La costruzione dell'istruzione SQL avviene a runtime: tramite una logica condizionale, il vincolo sull'anno di pubblicazione viene annesso alla clausola WHERE esclusivamente se il parametro è stato valorizzato. A livello di esecuzione, l'assenza di indici compositi dedicati costringe il motore di database a ricorrere a una scansione sequenziale delle tuple. Pertanto, la complessità asintotica permane, mantenendo un costo computazionale lineare rispetto al volume dei dati.



Flusso di esecuzione per la ricerca di un libro.

4.3 Gestione Operazioni Autenticate

Prima di commettere qualsiasi operazione di scrittura sul database, il server esegue una rigorosa validazione preventiva dei permessi per preservare la coerenza logica dei dati.

Nel caso specifico del salvataggio di una recensione, il sistema lancia una query di controllo (userHasBook) sulle tabelle associative per verificare l'effettivo possesso del libro. Poiché questa operazione sfrutta le chiavi primarie ed esterne indicizzate, il database evita scansioni costose, puntando direttamente ai record interessati.

Sotto il profilo algoritmico, la complessità asintotica è pari a O(log M) (o tendente a O(1) in caso di ottimizzazioni hash/cache). Questo ordine di grandezza riflette la rapidità dell'attraversamento degli indici B-Tree, dove indica la dimensione della tabella di relazione, rendendo il controllo praticamente istantaneo anche su grandi volumi di dati.

4.4 Recupero dettagli libro

Il recupero dei dettagli puntuali di un'opera avviene mediante accesso diretto tramite Chiave Primaria (ID). Questa operazione rappresenta lo scenario di utilizzo ottimale per un database relazionale.

Il sistema sfrutta l'indicizzazione automatica della Primary Key, gestita internamente tramite strutture dati B-Tree (alberi bilanciati). Ciò consente al motore di database di navigare gerarchicamente l'indice per localizzare il record, evitando costose scansioni sequenziali.

Di conseguenza, la complessità asintotica è pari a O(log N) (Logaritmica). Questo modello garantisce prestazioni eccellenti e tempi di risposta immediati, assicurando la scalabilità del sistema anche a fronte di una crescita esponenziale del catalogo (N).

5. Gestione della concorrenza

Sotto il profilo esecutivo, il server adotta un modello architetturale di tipo Thread-per-Request. Il componente MainServer opera in un ciclo continuo di ascolto (loop) sulla porta TCP configurata. Al momento della ricezione di una richiesta di connessione tramite la primitiva bloccante accept(), il sistema istanzia immediatamente un Thread dedicato(handleClient). Questo thread assume la responsabilità esclusiva della gestione dell'intero ciclo di vita della sessione con quello specifico client, garantendo che le operazioni di un utente non blocchino quelle degli altri.

Per quanto concerne l'integrità dei dati in ambiente multi-utente, la concorrenza è governata a livello di persistenza sfruttando le proprietà ACID (Atomicità, Coerenza, Isolamento, Durabilità) native di PostgreSQL. Tuttavia, per operazioni che richiedono modifiche multiple e interdipendenti (come la cancellazione a cascata di un account e dei relativi dati associati), la logica applicativa Java interviene manualmente sui confini della transazione. Disabilitando la modalità autoCommit, il sistema assicura l'atomicità del processo: le modifiche vengono rese permanenti (commit) solo al successo dell'intera

sequenza, oppure interamente annullate (rollback) in caso di errore, preservando così la rigorosa integrità referenziale del database.

Oltre al modello *thread-per-request* adottato dal server, anche il client implementa una gestione esplicita della concorrenza al fine di preservare la reattività dell'interfaccia grafica.

Le operazioni che coinvolgono comunicazioni di rete o elaborazioni potenzialmente onerose, come la ricerca dei libri e il recupero dei dettagli di un'opera selezionata, vengono eseguite su thread in tramite l'utilizzo di ExecutorService e Task.

L'interazione con i componenti grafici avviene esclusivamente sul *JavaFX Application Thread*, garantendo il rispetto del modello di threading di JavaFX ed evitando blocchi dell'interfaccia utente.

6. Logging

6.1 Client

La stabilità del client è assicurata da una strategia di gestione degli errori progettata per prevenire interruzioni improvvise dell'applicazione. Il sistema intercetta puntualmente le eccezioni a runtime adottando un duplice approccio:

1. Feedback Utente (UX): Al verificarsi di un'anomalia, l'interfaccia grafica notifica immediatamente l'utente tramite finestre di dialogo modali (classe Alert di JavaFX). Questo meccanismo garantisce che l'errore venga comunicato in modo chiaro e leggibile, migliorando l'usabilità complessiva.
2. Diagnostica (Debug): Parallelamente, per supportare le fasi di sviluppo e manutenzione, i dettagli tecnici completi dell'errore (Stack Trace) vengono reindirizzati sul flusso di errore standard (System.err). Questa pratica consente agli sviluppatori di analizzare la catena di chiamate che ha portato al guasto senza esporre dati incomprensibili all'utente finale.

6.2 Server

Al fine di garantire la completa osservabilità del sistema durante l'esecuzione, il server implementa un meccanismo di logging sincronizzato che instrada i flussi informativi direttamente sullo Standard Output (System.out). Questa scelta permette un monitoraggio in tempo reale dello stato dell'applicazione tramite console.

Il tracciamento copre le seguenti categorie critiche di eventi:

- Ciclo di Vita del Servizio: Log dettagliati relativi alla fase di bootstrap, confermando l'avvenuta inizializzazione dei componenti core.
- Diagnostica di Persistenza: Registrazione dei parametri di configurazione e dell'esito della negoziazione della connessione JDBC con il database PostgreSQL.
- Analisi delle Performance: Tracciamento selettivo delle query SQL a maggiore complessità computazionale, utile per l'ottimizzazione e il debug delle interazioni con i dati.
- Gestione delle Sessioni: Monitoraggio puntuale degli eventi di networking, inclusi l'accettazione di nuove connessioni client e la chiusura delle sessioni esistenti.

7. Strumenti, librerie e linguaggi utilizzati

- Linguaggio di programmazione: Java 17.
- Gestione dipendenze: Apache Maven.
- Database: PostgreSQL (Driver JDBC 42.7.4).
- Interfaccia Grafica: JavaFX 21.0.4 (org.openjfx).
- Ambiente di Sviluppo: IntelliJ IDEA + vsCode.
- Controllo di Versione: Git.

8. Limiti dell'applicazione e conclusioni

Sebbene l'architettura attuale garantisca stabilità e coerenza funzionale rispetto agli obiettivi didattici prefissati, un'analisi critica orientata all'industrializzazione del software evidenzia specifici margini di miglioramento:

- Introduzione del Connection Pooling: L'implementazione corrente prevede l'apertura di una connessione fisica dedicata per ogni singola operazione atomica dei Repository. Seppur sicura, questa strategia introduce un overheadsignificativo dovuto ai tempi di handshake col database. L'integrazione di un middleware di pooling (come la libreria HikariCP) permetterebbe il riutilizzo delle connessioni attive, abbattendo drasticamente la latenza e migliorando il throughput del sistema sotto carichi di lavoro elevati.
- Sicurezza del Canale di Comunicazione: Attualmente, lo scambio dati su socket TCP avviene in chiaro. Per un impiego in ambienti di produzione, è mandatorio implementare protocolli crittografici SSL/TLS. Questo garantirebbe la confidenzialità e l'integrità dei dati sensibili (in

particolare le credenziali di autenticazione), mitigando vulnerabilità quali attacchi Man-in-the-Middle o sniffing di rete.

Conclusioni

In sintesi, il progetto BookRecommender ha soddisfatto pienamente i requisiti funzionali previsti per la realizzazione di un sistema di gestione libraria collaborativa. L'applicazione ha permesso di validare empiricamente l'efficacia del pattern architetturale Client-Server modulare in ambiente Java. La netta separazione delle responsabilità tra i livelli (Presentation, Service, Data Access) e l'adozione di protocolli di comunicazione personalizzati hanno prodotto un ecosistema software estendibile, manutenibile e tecnicamente coerente con gli standard dell'Ingegneria del Software.

9. Bibliografia

1. Java Platform, Standard Edition 17 API Specification - Oracle.
2. PostgreSQL 14 Documentation - The PostgreSQL Global Development Group.
3. JavaFX 21 API Documentation - OpenJFX.