Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate
Corso di Laurea Triennale in Informatica

Laboratorio Interdisciplinare B

BookRecommender

Sistema di Raccomandazione Libri

MANUALE TECNICO

Studenti:

Ariele Babini
Matricola: 757608

Federico Bottaro
Matricola: 758017

Anno Accademico:
2024/2025

Versione:
1.0-SNAPSHOT

Data:
2 settembre 2025

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Scopo del Documento

Questo manuale tecnico fornisce una documentazione completa dell'applicazione **BookRecommender**, un sistema client-server per la gestione e raccomandazione di libri sviluppato come progetto per il corso di Laboratorio Interdisciplinare B presso l'Università degli Studi dell'Insubria.
Il documento è strutturato per fornire agli sviluppatori e ai manutentori del sistema tutte le informazioni tecniche necessarie per comprendere, modificare ed estendere l'applicazione.

1.2 Panoramica del Sistema

BookRecommender è un'applicazione distribuita che implementa un sistema di raccomandazione basato sulle preferenze degli utenti. Il sistema è composto da:

- **Server Backend**: Si tratta di un'applicazione lato server che si connette al database **PostgreSQL** e gestisce la logica di business e le principali funzionalità dell'applicazione
- **Client Desktop**: Applicazione con interfaccia utente grafica (**GUI**) sviluppata in **JavaFX** che adotta il design pattern **MVC (Model-View-Controller)** per una corretta interazione con l'utente e con il server
- **Database**: **PostgreSQL** è il sistema di gestione del database utilizzato per la persistenza dei dati relativi a libri, utenti, valutazioni e librerie
- **Modulo Condiviso**: Il package che contiene le interfacce e le classi serializzabili (come *Library*, *User*, *BookRating*) che modellano gli oggetti scambiati durante la comunicazione tra client e server

1.3 Architettura Generale

L'applicazione BABO Library adotta un'architettura **client-server distribuita** con una chiara separazione dei compiti tra i suoi componenti principali. La comunicazione tra il client e il server avviene attraverso **protocolli HTTP/REST**, garantendo interoperabilità e scalabilità del sistema. L'architettura è logicamente suddivisa in quattro layer distinti:

- **Presentation Layer**: L'interfaccia utente grafica (GUI) sviluppata in **JavaFX** che implementa il design pattern **MVC (Model-View-Controller)**. Include componenti specializzati come `MainWindow` per la gestione centralizzata dell'interfaccia, `ContentArea` per la visualizzazione dei contenuti e `PopupManager` per la gestione delle finestre modali.
- **Service Layer**: La logica di business è implementata attraverso **Spring Boot** con architettura REST. Include servizi specializzati come `BookService`, `UserService` e `AdminService` che gestiscono rispettivamente il catalogo libri, l'autenticazione/autorizzazione e le funzionalità amministrative.
- **Data Layer**: La persistenza dei dati è affidata a **PostgreSQL** con accesso diretto tramite **JDBC**. Il database gestisce entità principali come utenti, libri, recensioni e valutazioni, implementando un sistema di fallback per garantire la continuità del servizio.
- **Communication Layer**: L'interconnessione tra client e server utilizza **API REST over HTTP** con serializzazione JSON tramite **Jackson**. Il client JavaFX utilizza **OkHttp** per le comunicazioni HTTP, garantendo performance ottimali e gestione robusta degli errori di rete.

1.3.1 Componenti Architetturali Chiave

Il sistema implementa pattern architetturali consolidati per garantire manutenibilità, scalabilità e robustezza. Questi pattern rappresentano soluzioni progettuali collaudate che separano le responsabilità, riducono l'accoppiamento tra componenti e facilitano la manutenzione del codice:

Repository Pattern: Fornisce un'astrazione per l'accesso ai dati, isolando la logica di business dalle specifiche implementazioni del database. Nel sistema BABO, le classi `BookService` e `UserService` fungono da repository, incapsulando tutte le operazioni di accesso al database PostgreSQL. Questo pattern consente di modificare il sistema di persistenza senza impattare sulla business logic.

Service Layer Pattern: Organizza la business logic in servizi dedicati e riutilizzabili. Ogni servizio ha una responsabilità specifica: `BookService` gestisce il catalogo libri, `UserService` l'autenticazione e la gestione profili, `AdminService` le funzionalità amministrative. Questa separazione facilita la manutenzione e permette il riutilizzo della logica in diversi contesti.

MVC Pattern: Implementato nel client JavaFX per separare le responsabilità dell'interfaccia utente:

- **Model:** Rappresentato dalle classi del dominio (`Book`, `User`, `Review`)
- **View:** Componenti JavaFX per la presentazione (`MainWindow`, `ContentArea`)
- **Controller:** Classi che gestiscono l'interazione utente e coordinano Model e View

Questa architettura garantisce che modifiche all'interfaccia non influenzino la logica di business e viceversa.

Singleton Pattern: Utilizzato per componenti che richiedono una singola istanza globale nell'applicazione. `PopupManager` gestisce centralmente tutti i popup e finestre modali, mentre `AuthenticationManager` mantiene lo stato di autenticazione dell'utente corrente. Questo pattern garantisce coerenza nell'accesso a risorse condivise e previene duplicazioni indesiderate.

Vantaggi Architetturali

L'adozione di questi pattern comporta benefici tangibili:

- **Manutenibilità:** Il codice è organizzato in moduli con responsabilità chiare
- **Testabilità:** I componenti isolati possono essere testati indipendentemente
- **Riusabilità:** I servizi possono essere utilizzati in contesti diversi
- **Scalabilità:** Nuove funzionalità possono essere aggiunte senza modificare l'architettura esistente
- **Robustezza:** La separazione delle responsabilità riduce la propagazione degli errori

Capitolo 2

Progettazione del Sistema

2.1 Requisiti Funzionali

Il sistema BABO Library implementa le seguenti funzionalità principali:

2.1.1 Gestione Utenti

- Registrazione e autenticazione utenti con hashing SHA-256
- Gestione profili personali con aggiornamento dati anagrafici
- Sistema di privilegi amministrativi basato su whitelist email
- Gestione sessioni utente e logout sicuro
- Validazione univocità email e username durante la registrazione

2.1.2 Catalogo Libri

- Visualizzazione catalogo completo con paginazione
- Ricerca avanzata per titolo, autore, categoria e ISBN
- Dettagli libro con copertina, descrizione e metadati completi
- Gestione amministrativa del catalogo (CRUD operations)
- Sistema di cache per ottimizzazione performance
- Meccanismo di fallback per garantire continuità del servizio

2.1.3 Sistema di Valutazioni e Recensioni

- Valutazione libri con criteri multipli (stile, contenuto, piacevolezza)
- Sistema di recensioni testuali degli utenti
- Controlli anti-spam e moderazione contenuti

- Analytics e statistiche delle valutazioni
- Gestione amministrativa delle recensioni inappropriate

2.1.4 Librerie Personali

- Creazione e gestione librerie personalizzate per utente
- Aggiunta e rimozione libri dalle collezioni personali
- Visualizzazione organizzata delle librerie con interfaccia intuitiva
- Statistiche personali di lettura e collezione

2.1.5 Interfaccia Utente Avanzata

- Sistema di popup centralizzato per dettagli libri e modali
- Navigazione fluida tra sezioni (Home, Esplora, Librerie, Admin)
- Sistema di ricerca con cache intelligente
- Protezione applicazione contro avvii multipli
- Gestione eventi keyboard (ESC per chiusura popup)

2.2 Requisiti Non Funzionali

2.2.1 Performance

- Tempo di risposta API REST $<$ 500ms per operazioni standard
- Supporto fino a 1000 utenti concorrenti
- Caricamento interfaccia JavaFX $<$ 2 secondi
- Sistema di cache multi-livello per ottimizzare accesso ai dati
- Pool di connessioni database per gestione efficiente risorse

2.2.2 Sicurezza

- Hashing password con algoritmo SHA-256
- Query parametrizzate per prevenzione SQL injection
- Controlli di autorizzazione per operazioni sensibili
- Validazione rigorosa input utente lato server
- Sistema di whitelist per privilegi amministrativi
- Configurazione CORS per comunicazioni cross-origin sicure

2.2.3 Usabilità

- Interfaccia intuitiva seguendo design patterns consolidati
- Messaggi di errore user-friendly con emoji distintive
- Supporto multi-piattaforma (Windows, macOS, Linux)
- Gestione graceful degli errori senza interruzione servizio
- Debug avanzato per monitoraggio stato applicazione

2.2.4 Scalabilità e Manutenibilità

- Architettura modulare basata su pattern architetturali consolidati
- Separazione chiara delle responsabilità (MVC, Service Layer, Repository)
- Gestione centralizzata componenti condivisi
- Logging dettagliato per monitoring e diagnostica
- Configurazione esternalizzata per diversi ambienti

2.3 Librerie Esterne e Dipendenze

2.3.1 Tecnologie Core

Java 17 Linguaggio di programmazione principale

JavaFX 21.0.2 Framework per interfaccia utente desktop multipiattaforma

Spring Boot Framework per sviluppo backend con architettura REST

PostgreSQL Sistema di gestione database relazionale per persistenza dati

Maven Tool di build automation e gestione dipendenze

2.3.2 Dipendenze Client (JavaFX)

- **JavaFX Controls** (21.0.2) - Componenti UI base
- **JavaFX FXML** (21.0.2) - Supporto markup dichiarativo
- **OkHttp** (4.12.0) - Client HTTP per comunicazioni REST
- **Jackson Core/Databind/Annotations** (2.15.2) - Serializzazione JSON

2.3.3 Dipendenze Server (Spring Boot)

- **Spring Boot Starter Web** - Framework REST e server embedded
- **PostgreSQL JDBC Driver** - Connettore database
- **Spring Boot Starter Test** - Framework testing integrato

2.3.4 Tools di Sviluppo

- **Maven Compiler Plugin** (3.11.0) - Compilazione Java 17
- **JavaFX Maven Plugin** (0.0.8) - Esecuzione applicazioni JavaFX
- **Exec Maven Plugin** (3.1.0) - Esecuzione con profili multipiattaforma

2.4 Requisiti di Sistema

2.4.1 Requisiti Hardware Minimi

- **Processore:** Dual-core 2.0 GHz o superiore
- **RAM:** 4 GB (8 GB raccomandati)
- **Spazio Disco:** 500 MB per applicazione + database
- **Rete:** Connessione Internet per comunicazioni client-server

2.4.2 Requisiti Software

Java Runtime Environment JRE 17 o superiore installato su client e server

PostgreSQL Versione 12 o superiore per database server

Sistema Operativo Client Windows 10+, macOS 10.14+, Linux (Ubuntu 18+)

Sistema Operativo Server Qualsiasi OS con supporto Java 17 e PostgreSQL

2.4.3 Configurazione Database

- **Host:** localhost (configurabile)
- **Porta:** 5432 (default PostgreSQL)
- **Database:** DataProva
- **Utente:** postgres
- **Schema:** Tabelle users, books, libraries, ratings, reviews

2.4.4 Configurazioni Multipiattaforma

Il sistema include profili Maven specifici per ottimizzazione su diverse piattaforme:

- **Profilo macOS:** Configurazioni JavaFX specifiche per ambiente desktop Apple
- **Profilo Windows:** Ottimizzazioni per prevenire memory leak e gestione GPU
- **Profilo Debug:** Configurazione per debugging remoto su porta 5005

2.5 Flusso Operativo dell'Applicazione

2.5.1 Panoramica del Workflow Applicativo

Il sistema BABO implementa un workflow applicativo strutturato che guida l'utente attraverso diverse modalità d'uso, dall'accesso iniziale alle funzionalità avanzate di gestione personale dei contenuti. Il flusso operativo è progettato per supportare sia utenti anonimi che registrati, offrendo percorsi di navigazione ottimizzati per ogni tipologia di utilizzo.

L'architettura del flusso si basa su tre pilastri fondamentali:

- **Accesso Progressivo**: L'utente può esplorare il catalogo senza registrazione, con inviti contestuali per funzionalità premium
- **Autenticazione Flessibile**: Sistema dual-mode con login per utenti esistenti e registrazione guidata per nuovi utenti
- **Personalizzazione Incrementale**: Funzionalità che si arricchiscono progressivamente con l'utilizzo e l'engagement dell'utente

2.5.2 Architettura dei Percorsi Utente

Il sistema supporta quattro percorsi principali di navigazione, ognuno ottimizzato per specifici obiettivi utente:

Percorso di Esplorazione Libera

Dedicato agli utenti che desiderano navigare il catalogo senza impegno. Include:

- Accesso immediato al catalogo completo
- Funzionalità di ricerca avanzata (per titolo, autore, combinazioni)
- Visualizzazione dettagli libro senza necessità di autenticazione
- Call-to-action contestuali per incentivare la registrazione

Percorso di Autenticazione

Sistema biforcato che gestisce sia nuovi utenti che ritorni:

- **Login**: Processo rapido per utenti registrati con validazione credenziali
- **Registrazione**: Workflow guidato per creazione nuovo account con validazione dati
- Gestione errori intelligente con feedback specifico per ogni caso
- Recovery automatico per sessioni scadute

Percorso di Gestione Personalizzata

Attivato post-autenticazione, include:

- Creazione e gestione librerie personali tematiche
- Sistema di valutazioni multi-dimensionale per i libri
- Inserimento raccomandazioni peer-to-peer
- Gestione profilo utente e preferenze

Percorso di Amministrazione

Riservato agli utenti con privilegi amministrativi:

- Gestione catalogo (inserimento, modifica, rimozione libri)
- Moderazione contenuti generati dagli utenti
- Analytics e statistiche di utilizzo
- Gestione utenti e privilegi

2.5.3 Punti di Decisione e Branching Logic

Il flusso applicativo implementa diversi punti di decisione intelligenti che personalizzano l'esperienza utente:

Menu Principale: Punto di smistamento centrale che presenta opzioni diverse basate sullo stato di autenticazione e sui privilegi utente.

Controlli di Autenticazione: Validazione in tempo reale delle credenziali con gestione differenziata per utenti nuovi ed esistenti.

Gestione Librerie: Logica condizionale che verifica la presenza di librerie esistenti prima di presentare opzioni di creazione o gestione.

Sistema Valutazioni: Controlli contestuali che prevengono duplicazioni e garantiscono l'integrità dei dati di rating.

2.5.4 Gestione Stati e Transizioni

L'applicazione mantiene coerenza di stato attraverso tutti i percorsi utente:

- **Stato Anonimo**: Accesso limitato con possibilità di upgrade
- **Stato Autenticato**: Accesso completo alle funzionalità personali
- **Stato Amministrativo**: Privilegi estesi per gestione sistema
- **Stati Transitori**: Gestione ottimale di loading, errori e feedback

2.5.5 Ottimizzazioni del Flusso Utente

Il design del workflow incorpora diverse ottimizzazioni per migliorare l'esperienza utente:

Riduzione degli Attriti

- Accesso immediato alle funzionalità core senza registrazione obbligatoria
- Auto-completamento e suggerimenti intelligenti nei campi di ricerca
- Persistenza dello stato di navigazione durante le transizioni
- Gestione graceful degli errori con recovery automatico quando possibile

Personalizzazione Progressiva

- Onboarding graduale che introduce funzionalità avanzate progressivamente
- Adattamento dell'interfaccia basato sui pattern di utilizzo
- Suggerimenti contestuali per funzionalità non ancora utilizzate
- Preservazione delle preferenze utente tra sessioni diverse

Performance e Responsività

- Caricamento asincrono dei contenuti per ridurre i tempi di attesa
- Caching intelligente delle ricerche e dei contenuti frequentemente acceduti
- Precaricamento predittivo basato sui pattern di navigazione
- Fallback offline per funzionalità critiche

2.5.6 Validazione e Controlli di Qualità

Il sistema implementa controlli di validazione a più livelli:
Validazione Input: Controlli real-time sui dati inseriti dall'utente con feedback immediato per prevenire errori.

Controlli di Integrità: Verifiche server-side per garantire coerenza dei dati e prevenire manipolazioni.

Gestione Errori: Sistema robusto di error handling con messaggi utente comprensibili e suggerimenti per la risoluzione.

Logging e Monitoraggio: Tracciamento completo delle interazioni utente per analytics e debugging.

Questo approccio garantisce un'esperienza utente fluida e affidabile attraverso tutti i percorsi applicativi, supportando sia utilizzo occasionale che intensivo del sistema.

Figura 2.1: Diagramma di flusso operativo dell'applicazione BABO

2.5.7 Diagrammi di Sequenza dei Flussi Critici

Per fornire una rappresentazione dettagliata delle interazioni tra i componenti del sistema BABO, questa sezione presenta i sequence diagram dei flussi operativi più critici. Questi diagrammi illustrano le comunicazioni client-server, evidenziando i punti di validazione, gestione errori e pattern architetturali implementati.
I diagrammi seguenti complementano il diagramma di flusso operativo, fornendo una vista tecnica delle sequenze di chiamate tra i diversi layer dell'architettura. Ogni diagramma evidenzia:

- **Interazioni Client-Server**: Comunicazione RESTful tra JavaFX client e Spring Boot server
- **Validazioni Progressive**: Controlli client-side e server-side per garantire integrità dei dati
- **Gestione Stati**: Transizioni di stato e pattern di error handling implementati
- **Punti di Decisione**: Logica condizionale che personalizza i percorsi utente

L'approccio adottato segue i principi dell'architettura a microservizi con separazione netta delle responsabilità tra presentation layer (JavaFX), business logic layer (Spring Services) e persistence layer (PostgreSQL Database).

Processo di Autenticazione e Registrazione

Il sistema di autenticazione implementa un workflow dual-mode che gestisce sia utenti esistenti (login) che nuovi utenti (registrazione) attraverso validazioni progressive e controlli di sicurezza a più livelli. Il processo garantisce:

- **Sicurezza delle Credenziali**: Hashing SHA-256 delle password e validazione rigorosa degli input
- **Prevenzione Duplicati**: Controlli di unicità per email e username durante la registrazione
- **User Experience Ottimale**: Feedback immediato e gestione graceful degli errori
- **Scalabilità**: Architettura stateless con supporto per sessioni concorrenti

Il sequence diagram seguente illustra entrambi i flussi (login e registrazione) evidenziando i punti di validazione e le strategie di error handling implementate:

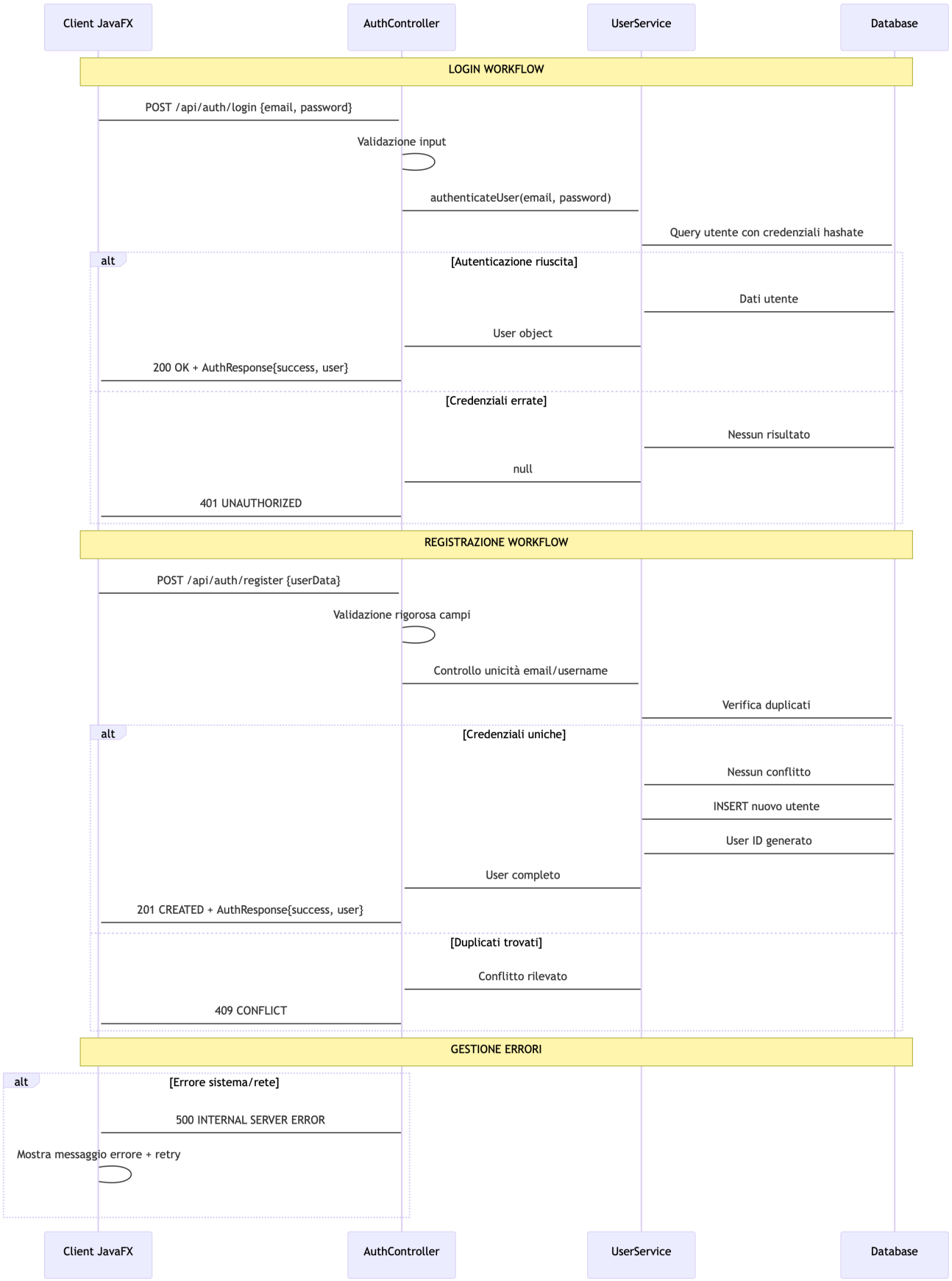


Figura 2.2: Diagramma sequenziale autenticazione

Gestione Librerie Personali

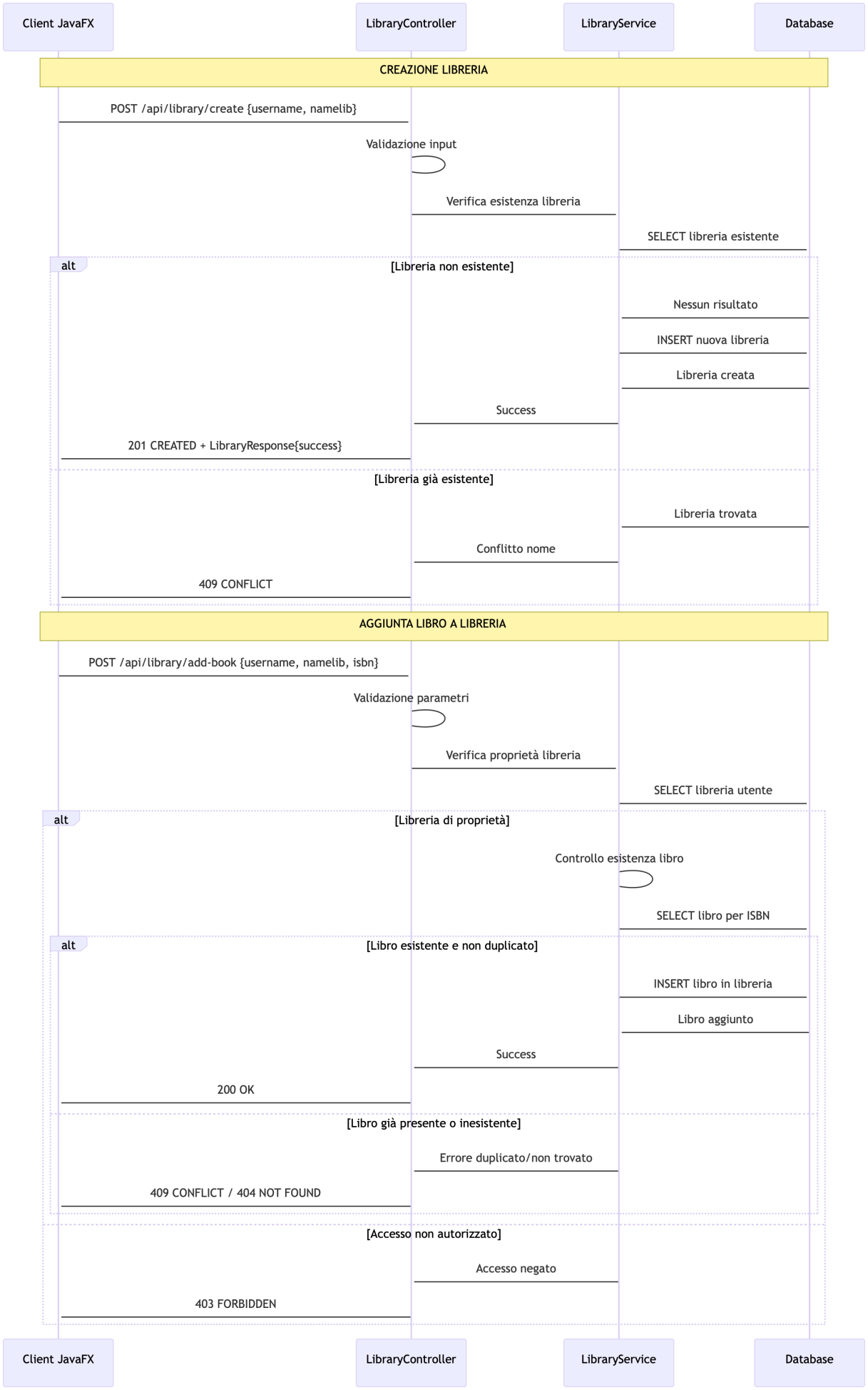
Il sistema di gestione librerie implementa un modello CRUD completo per consentire agli utenti registrati di organizzare raccolte personalizzate di libri con controlli rigorosi di proprietà e integrità referenziale. Il workflow delle librerie supporta le seguenti operazioni principali:

- **Creazione Librerie**: Generazione di nuove librerie tematiche con controlli di unicità per nome
- **Aggiunta Libri**: Inserimento di libri esistenti nel catalogo con prevenzione duplicati
- **Gestione Proprietà**: Validazione che solo il proprietario possa modificare le proprie librerie
- **Controlli Referenziali**: Verifica esistenza libri nel catalogo prima dell'inserimento

L'architettura implementa pattern di sicurezza per prevenire accessi non autorizzati e garantire l'isolamento dei dati tra utenti diversi. Il sistema di validazione opera su tre livelli:

1. **Validazione Client**: Controlli immediati sull'interfaccia per feedback rapido
2. **Validazione Business Logic**: Controlli di proprietà e esistenza risorse nel service layer
3. **Validazione Database**: Vincoli di integrità referenziale e constraint sui dati

Il sequence diagram seguente illustra i tre flussi principali della gestione librerie: creazione di una nuova libreria, aggiunta di un libro esistente e consultazione delle librerie utente, evidenziando i controlli di autorizzazione e i punti di validazione implementati:


Client JavaFX
LibraryController
LibraryService
Database
CREAZIONE LIBRERIA
POST /api/library/create {username, namelib}
Validazione input
Verifica esistenza libreria
SELECT libreria esistente
alt
[Libreria non esistente]
Nessun risultato
INSERT nuova libreria
Libreria creata
Success
201 CREATED + LibraryResponse{success}
[Libreria già esistente]
Libreria trovata
Conflitto nome
409 CONFLICT
AGGIUNTA LIBRO A LIBRERIA
POST /api/library/add-book {username, namelib, isbn}
Validazione parametri
Verifica proprietà libreria
SELECT libreria utente
alt
[Libreria di proprietà]
Controllo esistenza libro
SELECT libro per ISBN
alt
[Libro esistente e non duplicato]
INSERT libro in libreria
Libro aggiunto
Success
200 OK
[Libro già presente o inesistente]
Errore duplicato/non trovato
409 CONFLICT / 404 NOT FOUND
[Accesso non autorizzato]
Accesso negato
403 FORBIDDEN
Client JavaFX
LibraryController
LibraryService
Database

Capitolo 3

Progettazione del Database

Il database PostgreSQL `DataProva` rappresenta il cuore del sistema BABO Library, implementando un modello relazionale complesso per gestire utenti, libri, valutazioni e raccomandazioni. Il sistema utilizza PostgreSQL versione 17.5 con codifica UTF-8 per supportare caratteri internazionali.

3.0.1 Identificazione entità

Il sistema modella le seguenti entità principali con le relative caratteristiche e vincoli:

- **User:** Rappresenta gli utenti registrati del sistema con informazioni anagrafiche complete e credenziali di accesso sicure
- **Book:** Catalogo centrale dei libri con metadati bibliografici (ISBN, titolo, autore, categoria, descrizione, anno pubblicazione)
- **User Libraries:** Librerie personali create dagli utenti per organizzare e categorizzare i propri libri
- **Assessment:** Sistema di valutazione multi-criterio con recensioni testuali per ogni libro
- **Advise:** Sistema di raccomandazioni peer-to-peer tra utenti con supporto per multiple raccomandazioni per libro

3.0.2 Identificazione attributi

Ogni entità è caratterizzata da specifici attributi con domini e vincoli definiti:

Entità Users

- **id** (INTEGER, PRIMARY KEY): Identificativo univoco auto-incrementale
- **name** (VARCHAR(100), NOT NULL): Nome dell'utente
- **surname** (VARCHAR(100), NOT NULL): Cognome dell'utente
- **cf** (VARCHAR(16), NULLABLE): Codice fiscale italiano

- **email** (VARCHAR(255), UNIQUE, NOT NULL): Indirizzo email univoco
- **username** (VARCHAR(50), UNIQUE, NOT NULL): Username per autenticazione
- **password** (VARCHAR(256), NOT NULL): Password hashata SHA-256
- **created_at** (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP): Data registrazione

Entità Books

- **isbn** (VARCHAR, PRIMARY KEY): Codice ISBN univoco del libro
- **books_title** (VARCHAR): Titolo completo del libro
- **book_author** (VARCHAR): Nome dell'autore principale
- **category** (VARCHAR): Categoria/genere letterario
- **description** (VARCHAR): Sinossi o descrizione del libro
- **publi_year** (VARCHAR): Anno di pubblicazione
- **publisher** (VARCHAR): Casa editrice

Entità Assessment

- **username** (VARCHAR, FK): Utente che effettua la valutazione
- **isbn** (VARCHAR, FK): Libro valutato
- **data** (TIMESTAMP, NOT NULL): Timestamp della valutazione
- **style** (INTEGER, CHECK 1-5): Valutazione dello stile
- **content** (INTEGER, CHECK 1-5): Valutazione del contenuto
- **pleasantness** (INTEGER, CHECK 1-5): Valutazione della piacevolezza
- **originality** (INTEGER, CHECK 1-5): Valutazione dell'originalità
- **edition** (INTEGER, CHECK 1-5): Valutazione dell'edizione
- **average** (DOUBLE PRECISION): Media automatica delle valutazioni
- **review** (TEXT, NULLABLE): Recensione testuale opzionale

Entità Advise (Raccomandazioni)

- **id** (INTEGER, PRIMARY KEY): Identificativo univoco auto-incrementale
- **username** (VARCHAR, FK): Utente che formula la raccomandazione
- **isbn** (VARCHAR, FK): Libro target per cui si raccomandano altri libri
- **isbn1** (VARCHAR, FK, NOT NULL): Prima raccomandazione (obbligatoria)
- **isbn2** (VARCHAR, FK, NULLABLE): Seconda raccomandazione (opzionale)
- **isbn3** (VARCHAR, FK, NULLABLE): Terza raccomandazione (opzionale)

3.0.3 Identificazione relazioni

Il modello implementa le seguenti relazioni con relative cardinalità e vincoli di integrità:

- **User - Assessment (1:N):** Un utente può valutare più libri, ma ogni valutazione appartiene a un solo utente. Chiave composita (username, isbn) per unicità.
- **Book - Assessment (1:N):** Un libro può ricevere più valutazioni, ma ogni valutazione si riferisce a un solo libro. Foreign key con cascading.
- **User - User_Libraries (1:N):** Un utente può creare multiple librerie, ogni libreria appartiene a un solo utente. Cascade delete implementato.
- **User_Libraries - Library_Books (1:N):** Una libreria può contenere più libri, ogni associazione libro-libreria è unica. Vincolo di unicità composito.
- **Book - Library_Books (1:N):**Un libro può essere presente in più librerie, ogni record associa un libro a una specifica libreria.
- **User - Advise (1:N):** Un utente può formulare più raccomandazioni, ogni raccomandazione è associata a un utente specifico.
- **Book - Advise (1:N):**Un libro può essere oggetto di multiple raccomandazioni e può essere raccomandato per altri libri. Quattro foreign key verso books(isbn).

3.1 Schema Entità-Relazioni (ER)

3.1.1 Diagramma ER concettuale

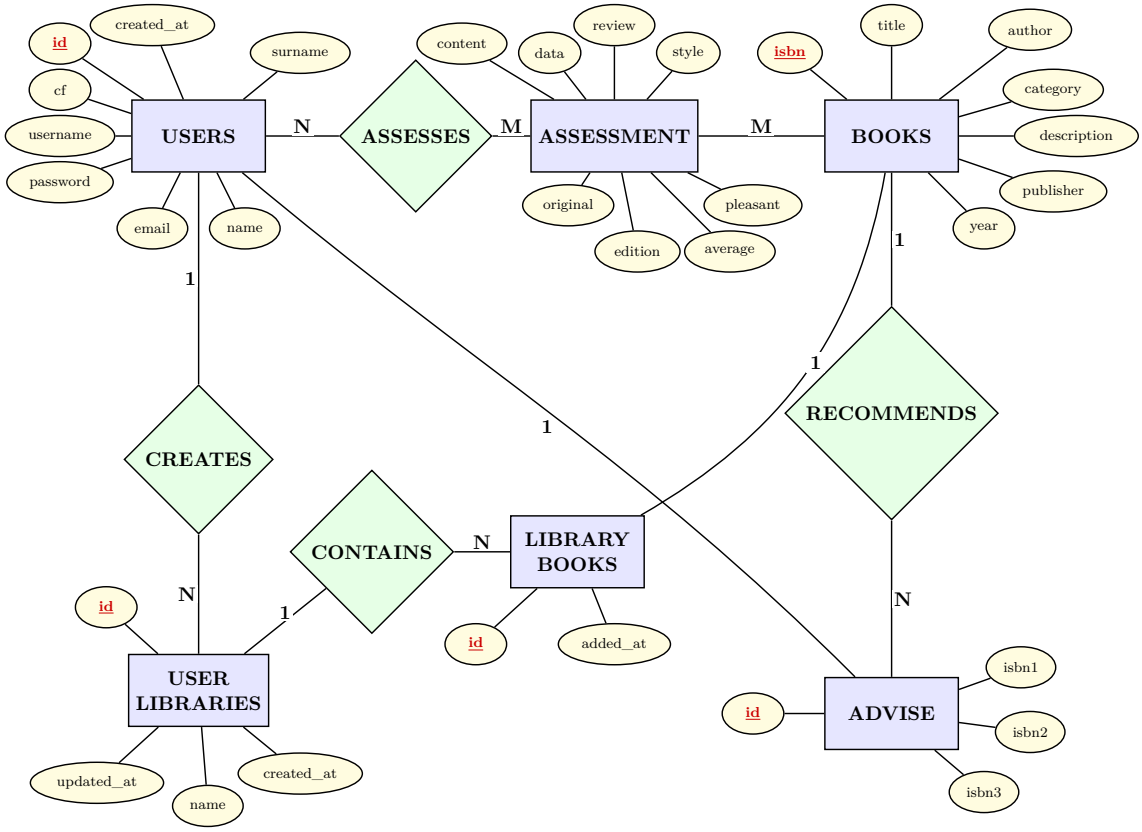


Figura 3.1: Diagramma ER del sistema BABO Library

3.1.2 Schema ER ristrutturato

Il modello ER è stato ristrutturato per eliminare anomalie e ottimizzare le performance:

- **Normalizzazione 3NF**: Tutte le tabelle rispettano la terza forma normale
- **Eliminazione ridondanze**: La media delle valutazioni è calcolata dinamicamente
- **Vincoli di integrità**: Implementazione completa di foreign key con cascade
- **Indici ottimizzati**: Indici univoci su combinazioni frequentemente ricercate

3.2 Schema logico relazionale

Il database implementa il seguente schema logico con tutte le specifiche tecniche:

Tabella 3.1: Tabella `users` - Gestione utenti sistema

Campo	Tipo	Descrizione	Vincoli
id	INTEGER	Identificativo univoco auto incrementale	PRIMARY KEY, AUTO_INCREMENT
name	VARCHAR(100)	Nome dell'utente	NOT NULL
surname	VARCHAR(100)	Cognome dell'utente	NOT NULL
cf	VARCHAR(16)	Codice fiscale italiano	NULLABLE
email	VARCHAR(255)	Email unica di registrazione	UNIQUE, NOT NULL
username	VARCHAR(50)	Username univoco	UNIQUE, NOT NULL
password	VARCHAR(256)	Password hashata con algoritmo SHA-256	NOT NULL
created_at	TIMESTAMP	Data e ora di registrazione	DEFAULT CURRENT_TIMESTAMP

Tabella 3.2: Tabella `user_libraries` - Gestione librerie personali

Campo	Tipo	Descrizione	Vincoli
id	BIGINT	Identificativo univoco auto incrementale	PRIMARY KEY, SERIAL
username	VARCHAR(50)	Proprietario libreria	NOT NULL, FK → `users(username)`, ON DELETE CASCADE
name	VARCHAR(100)	Nome della libreria	NOT NULL
created_at	TIMESTAMP	Data creazione	DEFAULT CURRENT_TIMESTAMP
updated_at	TIMESTAMP	Data ultima modifica	DEFAULT CURRENT_TIMESTAMP

Tabella 3.3: Tabella `library_books` - Contenuto librerie

Campo	Tipo	Descrizione	Vincoli
id	BIGINT	Identificativo record	PRIMARY KEY, SERIAL
username	VARCHAR(50)	Proprietario libreria	NOT NULL, FK → `users(username)`, ON DELETE CASCADE
library_name	VARCHAR(100)	Nome libreria	NOT NULL
isbn	VARCHAR(20)	Libro in libreria	NOT NULL, FK → `books(isbn)`, ON DELETE CASCADE
added_at	TIMESTAMP	Data aggiunta libro	DEFAULT CURRENT_TIMESTAMP

Tabella 3.4: Tabella `library_books` - Contenuto librerie

Campo	Tipo	Descrizione	Vincoli
id	BIGINT	Identificativo record	PRIMARY KEY, SERIAL
username	VARCHAR(50)	Proprietario libreria	NOT NULL, FK → `users(username)`, ON DELETE CASCADE
library_name	VARCHAR(100)	Nome libreria	NOT NULL
isbn	VARCHAR(20)	Libro in libreria	NOT NULL, FK → `books(isbn)`, ON DELETE CASCADE
added_at	TIMESTAMP	Data aggiunta libro	DEFAULT CURRENT_TIMESTAMP

Vincoli complessi:

- FK (username, library_name) → `user_libraries(username, name)`
- UNIQUE (username, library_name, isbn) - Libro unico per libreria

Tabella 3.5: Tabella `assessment` - Sistema valutazioni

Campo	Tipo	Descrizione	Vincoli
username	VARCHAR	Utente che valuta	PRIMARY KEY (composita)
isbn	VARCHAR	Libro valutato	PRIMARY KEY (composita), FK
data	TIMESTAMP	Timestamp inserimento valutazione	NOT NULL
style	INTEGER	Voto stile (1-5)	CHECK ($1 \leq$ style ≤ 5)
content	INTEGER	Voto contenuto (1-5)	CHECK ($1 \leq$ content ≤ 5)
pleasantness	INTEGER	Voto piacevolezza (1-5)	CHECK ($1 \leq$ pleasantness ≤ 5)
originality	INTEGER	Voto per l'originalità (1-5)	CHECK ($1 \leq$ originality ≤ 5)
edition	INTEGER	Voto edizione (1-5)	CHECK ($1 \leq$ edition ≤ 5)
average	DOUBLE PRECISION	Media automatica	NOT NULL
review	TEXT	Recensione testuale	NULLABLE

Tabella 3.6: Tabella `advise` - Sistema raccomandazioni

Campo	Tipo	Descrizione	Vincoli
id	INTEGER	Identificativo univoco	PRIMARY KEY, AUTO
username	VARCHAR	Utente che inserisce raccomandazione	FK → `users(username)`
isbn	VARCHAR	Libro target	FK → `books(isbn)`
isbn1	VARCHAR	Prima raccomandazione	FK → `books(isbn)`, NOT NULL
isbn2	VARCHAR	Seconda raccomandazione	FK → `books(isbn)`, NULLABLE
isbn3	VARCHAR	Terza raccomandazione	FK → `books(isbn)`, NULLABLE
UNIQUE INDEX: (username, isbn)			

3.2.1 Vincoli di integrità implementati

Il database implementa un sistema completo di vincoli per garantire la coerenza dei dati:

Vincoli di dominio

```
-- Vincoli per valutazioni (range 1-5)
CONSTRAINT assessment_style_check CHECK ((style >= 1) AND (style <= 5))
CONSTRAINT assessment_content_check CHECK ((content >= 1) AND (content <= 5))
CONSTRAINT assessment_pleasantness_check CHECK ((pleasantness >= 1) AND (pleasantness <= 5))
CONSTRAINT assessment_originality_check CHECK ((originality >= 1) AND (originality <= 5))
CONSTRAINT assessment_edition_check CHECK ((edition >= 1) AND (edition <= 5))
```

Listing 3.1: Vincoli CHECK per valutazioni

Vincoli di integrità referenziale

```
-- Vincoli CASCADE per eliminazione sicura
ALTER TABLE user_libraries
    ADD CONSTRAINT fk_user_libraries_username
    FOREIGN KEY (username) REFERENCES users(username) ON DELETE CASCADE;

ALTER TABLE library_books
    ADD CONSTRAINT fk_library_books_user
    FOREIGN KEY (username) REFERENCES users(username) ON DELETE CASCADE;

ALTER TABLE library_books
    ADD CONSTRAINT fk_library_books_isbn
    FOREIGN KEY (isbn) REFERENCES books(isbn) ON DELETE CASCADE;

-- Vincoli per raccomandazioni multiple
ALTER TABLE advise
    ADD CONSTRAINT advise_isbn1_fkey
    FOREIGN KEY (isbn1) REFERENCES books(isbn);

ALTER TABLE advise
    ADD CONSTRAINT advise_isbn2_fkey
    FOREIGN KEY (isbn2) REFERENCES books(isbn);

ALTER TABLE advise
    ADD CONSTRAINT advise_isbn3_fkey
    FOREIGN KEY (isbn3) REFERENCES books(isbn);
```

Listing 3.2: Foreign Key con CASCADE

Vincoli di unicità

```sql
-- Unicita email e username
ALTER TABLE users ADD CONSTRAINT users_email_key UNIQUE (email);
ALTER TABLE users ADD CONSTRAINT users_username_key UNIQUE (
    username);

-- Unicita valutazione per utente-libro
ALTER TABLE assessment ADD CONSTRAINT assessment_pkey PRIMARY KEY (
    username, isbn);

-- Unicita raccomandazione per utente-libro
CREATE UNIQUE INDEX advise_user_book_unique ON advise (username,
    isbn);

-- Unicita libro in libreria specifica
ALTER TABLE library_books
    ADD CONSTRAINT unique_book_in_library
    UNIQUE (username, library_name, isbn);

-- Unicita nome libreria per utente
ALTER TABLE user_libraries
    ADD CONSTRAINT unique_user_library
    UNIQUE (username, name);
```

Listing 3.3: Indici UNIQUE per integrità dati

3.3 Schema fisico PostgreSQL

3.3.1 Configurazione database

Il database utilizza la seguente configurazione ottimizzata per PostgreSQL 17.5:

```sql
-- Impostazioni connessione
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET default_tablespace = '';
SET default_table_access_method = heap;

-- Timeout configurazioni
SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET transaction_timeout = 0;
```

Listing 3.4: Configurazione PostgreSQL

3.3.2 Sequenze auto-incrementali

Implementazione delle sequenze per chiavi primarie auto-incrementali:

```
-- Sequenza per users.id
CREATE SEQUENCE users_id_seq AS integer
    START WITH 1 INCREMENT BY 1
    NO MINVALUE NO MAXVALUE CACHE 1;
ALTER TABLE users ALTER COLUMN id SET DEFAULT nextval('users_id_seq
    ');

-- Sequenza per user_libraries.id
CREATE SEQUENCE user_libraries_id_seq
    START WITH 1 INCREMENT BY 1
    NO MINVALUE NO MAXVALUE CACHE 1;
ALTER TABLE user_libraries ALTER COLUMN id SET DEFAULT nextval('
    user_libraries_id_seq');

-- Sequenza per library_books.id
CREATE SEQUENCE library_books_id_seq
    START WITH 1 INCREMENT BY 1
    NO MINVALUE NO MAXVALUE CACHE 1;
ALTER TABLE library_books ALTER COLUMN id SET DEFAULT nextval('
    library_books_id_seq');

-- Sequenza per advise.id
CREATE SEQUENCE advise_id_seq AS integer
    START WITH 1 INCREMENT BY 1
    NO MINVALUE NO MAXVALUE CACHE 1;
ALTER TABLE advise ALTER COLUMN id SET DEFAULT nextval('
    advise_id_seq');
```

Listing 3.5: Definizione sequenze

3.3.3 Ottimizzazioni performance

Il database implementa diverse ottimizzazioni per migliorare le performance:

- **Indici B-tree**: Su tutte le foreign key per join efficienti
- **Indici compositi**: Per query multi-colonna frequenti
- **Cache sequenze**: Cache = 1 per sequence per ottimizzare l'auto-increment
- **Heap access method**: Metodo di accesso standard per tabelle transazionali

3.4 Dizionario dei dati

3.4.1 Statistiche database

Il database è dimensionato per supportare:

- **Utenti**: Fino a 100.000 utenti registrati
- **Libri**: Catalogo di circa 50.000 titoli
- **Valutazioni**: Supporto per milioni di recensioni
- **Librerie**: Fino a 50 librerie per utente
- **Raccomandazioni**: Sistema scalabile per raccomandazioni peer-to-peer

Nota: Per questioni pratiche e di testing, il database è stato inizialmente popolato con solamente 1000 titoli.

3.4.2 Considerazioni di sicurezza

Il modello implementa le seguenti misure di sicurezza:

Password Hashing: Tutte le password sono hashate SHA-256 lato applicazione

SQL Injection Prevention: Uso esclusivo di prepared statements

Cascade Deletes: Eliminazione sicura di record correlati

Data Integrity: Vincoli CHECK per validazione dati

Audit Trail: Timestamp per tracking modifiche

Capitolo 4

Architettura Software

4.1 Introduzione

Il sistema BABO Library implementa un'architettura multi-modulo basata su Maven che segue i principi della separazione delle responsabilità e dell'architettura a microservizi. L'architettura è progettata per garantire scalabilità, manutenibilità e disaccoppiamento attraverso pattern architetturali consolidati. Il sistema è organizzato in tre moduli principali che comunicano attraverso interfacce REST ben definite, garantendo l'indipendenza e la testabilità dei componenti.

4.2 Struttura dei Moduli Maven

Il progetto utilizza una struttura multi-modulo Maven che organizza il codice in tre moduli principali, ciascuno con responsabilità specifiche e ben definite.

4.2.1 Struttura Generale del Progetto

Il progetto root definisce la configurazione generale e coordina i tre moduli principali:

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.BABO</groupId>
    <artifactId>BookRecommender</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <name>BABO Library - Sistema Gestione Libri</name>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.
           sourceEncoding>
        <spring.boot.version>3.2.0</spring.boot.version>
        <javafx.version>21.0.2</javafx.version>
    </properties>

    <modules>
```

```
        <module>shared</module>
        <module>server</module>
        <module>client</module>
    </modules>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.fasterxml.jackson</groupId>
                <artifactId>jackson-bom</artifactId>
                <version>2.15.2</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring.boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
</project>
```

Listing 4.1: Configurazione Maven Root (pom.xml)

4.2.2 Modulo Shared

Il modulo `shared` contiene tutte le classi comuni utilizzate sia dal client che dal server, garantendo consistenza dei dati e riducendo la duplicazione del codice.

Organizzazione dei Package

La struttura del modulo shared è organizzata come segue:

- **`org.BABO.shared.model`**: Contiene i modelli di dominio
  - `Book`: Rappresentazione dei libri del catalogo
  - `User`: Modello degli utenti del sistema
  - `Library`: Librerie personali degli utenti
  - `BookRating`: Valutazioni multi-criterio dei libri
  - `Review`: Recensioni testuali dettagliate
  - `BookRecoomendation`: Raccomandazioni dei libri
  - `Category`: Categorizzazione dei libri

- **`org.BABO.shared.dto`**: Data Transfer Objects organizzati per funzionalità
  - `Authentication/`: `AuthRequest`, `AuthResponse`, `RegisterRequest`
  - `Library/`: `CreateLibraryRequest`, `LibraryResponse`, `AddBookToLibraryRequest`, `RemoveBookFromLibrary`
  - `Rating/`: `RatingRequest`, `RatingResponse`
  - `Recommendation/`: `RecommendationRequest`, `RecommendationResponse`
  - `Reviews/`: `ReviewsResponse`, `ReviewsStats`, `ReviwsStastResponse`
  - `AdminResponse`

Modello di Dominio Book

Il modello `Book` rappresenta l'entità centrale del sistema:

```
@JsonIgnoreProperties(ignoreUnknown = true)
public class Book {
    @JsonProperty("id")
    private Long id;

    @JsonProperty("isbn")
    private String isbn;

    @JsonProperty("title")
    private String title;

    @JsonProperty("author")
    private String author;

    @JsonProperty("description")
    private String description;

    @JsonProperty("imageUrl")
    private String imageUrl;

    @JsonProperty("publishYear")
    private String publishYear;

    @JsonProperty("price")
    private Double price;

    @JsonProperty("isFree")
    private Boolean isFree;

    @JsonProperty("isNew")
    private Boolean isNew;

    @JsonProperty("category")
    private String category;

    @JsonProperty("publisher")
    private String publisher;

    @JsonProperty("language")
```

```
40      private String language;
41
42      @JsonProperty("pages")
43      private Integer pages;
44
45      @JsonProperty("reviewCount")
46      private int reviewCount = 0;
47
48      @JsonProperty("averageRating")
49      private double averageRating = 0.0;
50
51      // Costruttore completo esistente
52      public Book(Long id, String isbn, String title, String author,
53                  String description, String publishYear, String
                        imageUrl) {
54          this.id = id;
55          this.isbn = isbn;
56          this.title = title;
57          this.author = author;
58          this.description = description;
59          this.publishYear = publishYear;
60          this.imageUrl = imageUrl;
61          this.isFree = true;
62          this.isNew = false;
63      }
64
65      public String getFormattedRating() {
66          if (averageRating > 0) {
67              return String.format("%.1f", averageRating);
68          }
69          return "N/A";
70      }
71
72      public String getSafeImageFileName() {
73          String localFileName = getLocalImageFileName();
74          if (localFileName == null || localFileName.trim().isEmpty()
                ) {
75              return "placeholder.jpg";
76          }
77          return localFileName;
78      }
79
80      public boolean hasExternalImageUrl() {
81          return imageUrl != null && imageUrl.startsWith("http");
82      }
83
84      // Getters e Setters standard esistenti
85      public Long getId() { return id; }
86      public void setId(Long id) { this.id = id; }
87
88      public String getIsbn() { return isbn; }
89      public void setIsbn(String isbn) { this.isbn = isbn; }
90
91      public String getTitle() { return title; }
92      public void setTitle(String title) { this.title = title; }
93
```

```
94      public String getAuthor() { return author; }
95      public void setAuthor(String author) { this.author = author; }
96
97      public String getDescription() { return description; }
98      public void setDescription(String description) { this.
            description = description; }
99
100     public String getImageUrl() { return imageUrl; }
101     public void setImageUrl(String imageUrl) { this.imageUrl =
            imageUrl; }
102
103     public String getPublishYear() { return publishYear; }
104     public void setPublishYear(String publishYear) { this.
            publishYear = publishYear; }
105
106     public String getCategory() { return category; }
107     public void setCategory(String category) { this.category =
            category; }
108
109     public String getPublisher() { return publisher; }
110     public void setPublisher(String publisher) { this.publisher =
            publisher; }
111
112     // Altri getters/setters per tutti i campi...
113 }
```

Listing 4.2: Modello Book - Entità Principale

4.2.3 Modulo Server

Il modulo server implementa l'architettura REST API utilizzando Spring Boot con una struttura a layer che separa chiaramente le responsabilità.

Organizzazione dei Package

```
org.BABO.server/
├── ServerApplication.java ..... Main Spring Boot
├── controller/ ..... Controller REST
│   ├── AuthController.java ..... Autenticazione/Admin
│   ├── BookController.java ..... Gestione catalogo
│   ├── LibraryController.java ..... Librerie personali
│   ├── RatingController.java ..... Sistema valutazioni
│   └── RecommendationController.java ..... Raccomandazioni peer-to-peer
└── service/ ..... Business Logic
    ├── UserService.java ..... Gestione utenti
    ├── BookService.java ..... Servizi libri
    ├── LibraryService.java ..... Servizi librerie
    ├── RatingService.java ..... Servizi valutazioni
    └── RecommendationService.java ..... Servizi raccomandazioni
```

ServerApplication - Punto di Ingresso

```java
@SpringBootApplication
public class ServerApplication {

    public static void main(String[] args) {
        System.out.println("Avvio Apple Books Server...");
        SpringApplication.run(ServerApplication.class, args);
        System.out.println("Server avviato...");
    }

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedOrigins("*")
                        .allowedMethods("GET", "POST", "PUT", "
                           DELETE", "OPTIONS")
                        .allowedHeaders("*");
            }
        };
    }
}
```

Listing 4.3: ServerApplication - Bootstrap Spring Boot

4.2.4 Modulo Client

Il modulo client implementa l'interfaccia utente JavaFX con architettura MVC e pattern Observer.

Organizzazione dei Package

```
org.BABO.client/
├── ui/ ..... Controllers JavaFX
│   ├── BooksClient.java ..... Main Application
│   ├── Admin/
│   │   └── AdminPanel.java
│   ├── Authentication/
│   │   └── AuthenticationManager.java
│   ├── Home/
│   │   ├── MainWindow.java ..... Controller centrale
│   │   ├── Sidebar.java ..... Navigazione
│   │   ├── Header.java ..... Ricerca
│   │   └── ContentArea.java ..... Contenuti
│   ├── Library/
│   │   └── LibraryPanel.java
│   └── Popup/
│       ├── PopupManager.java
│       └── BookDetailsPopup.java
└── service/ ..... Client Services
    ├── BookService.java
    ├── LibraryService.java
    └── AdminService.java
```

BooksClient - Main Application

```
public class BooksClient extends Application {

    private BookService bookService;
    private boolean serverAvailable = false;
    private MainWindow mainWindow;

    @Override
    public void init() {
        System.out.println("Inizializzazione client...");
        bookService = new BookService();

        serverAvailable = bookService.isServerAvailable();
        if (serverAvailable) {
            System.out.println("Server raggiungibile");
        } else {
            System.out.println("Server non raggiungibile - modalita
                'offline");
        }
    }

    @Override
    public void start(Stage stage) throws Exception {
        System.out.println("Avvio BooksClient con PopupManager
            integrato");

        try {
            // Registra protezione applicazione
            ApplicationProtection.registerMainStage(stage);

            // Imposta icona applicazione
            setupApplicationIcon(stage);

            // Crea finestra principale
            mainWindow = new MainWindow(bookService,
                serverAvailable);
            StackPane root = mainWindow.createMainLayout();

            // Inizializza PopupManager
            PopupManager popupManager = PopupManager.getInstance();
            popupManager.initialize(root);

            // Setup scena
            Scene scene = new Scene(root, 1300, 800);
            stage.setScene(scene);
            stage.setTitle("Books Client " + serverAvailable);

            stage.setMinWidth(1300);
            stage.setMinHeight(800);

            // Gestione chiusura
            stage.setOnCloseRequest(e -> {
                System.out.println("Richiesta chiusura applicazione
                    ...");
                handleApplicationClose();
```

```
51              });
52
53              stage.show();
54              stage.centerOnScreen();
55
56          } catch (Exception e) {
57              showStartupError(e);
58              Platform.exit();
59          }
60      }
61
62      @Override
63      public void stop() {
64          System.out.println("Stop applicazione...");
65
66          try {
67              if (bookService != null) {
68                  bookService.shutdown();
69              }
70              PopupManager.getInstance().emergencyReset();
71          } catch (Exception e) {
72              System.err.println("Errore durante stop: " + e.
                    getMessage());
73          }
74      }
75
76      private void setupApplicationIcon(Stage stage) {
77          System.out.println("Configurazione icona applicazione cross
                -platform...");
78          try {
79              IconUtils.setStageIcon(stage);
80              IconUtils.setSystemTrayIcon();
81          } catch (Exception e) {
82              System.err.println("Errore setup icona: " + e.
                    getMessage());
83          }
84      }
85
86      private void handleApplicationClose() {
87          try {
88              PopupManager popupManager = PopupManager.getInstance();
89              if (popupManager.hasActivePopups()) {
90                  popupManager.closeAllPopups();
91              }
92              finalizeApplicationClose();
93          } catch (Exception e) {
94              finalizeApplicationClose();
95          }
96      }
97
98      private void finalizeApplicationClose() {
99          try {
100             ImageUtils.clearImageCache();
101             if (mainWindow != null && mainWindow.getAuthManager()
                    != null) {
102                 mainWindow.getAuthManager().shutdown();
```

```
103             }
104             Platform.exit();
105         } catch (Exception e) {
106             Platform.exit();
107         }
108     }
109
110     // Metodo statico pubblico
111     public static void openBookDetails(Book book, List<Book>
            collection,
112                                        AuthenticationManager
                                               authManager) {
113         if (book == null) return;
114
115         try {
116             PopupManager popupManager = PopupManager.getInstance();
117             if (popupManager.isInitialized()) {
118                 popupManager.showBookDetails(book, collection,
                        authManager);
119             }
120         } catch (Exception e) {
121             System.err.println("Errore apertura popup: " + e.
                    getMessage());
122         }
123     }
124
125     public BookService getBookService() {
126         return bookService;
127     }
128 }
```

Listing 4.4: BooksClient - Punto di Ingresso Client

4.3 Architettura Controller REST

Il sistema implementa un'architettura REST completa attraverso controller specializzati che gestiscono diverse funzionalità del sistema.

4.3.1 AuthController - Gestione Autenticazione

L'`AuthController` centralizza tutte le operazioni di autenticazione e amministrazione:

```
@RestController
@RequestMapping("/api/auth")
@CrossOrigin(origins = "*")
public class AuthController {

    @Autowired
    private UserService userService;

    @Autowired
    private BookService bookService;

    @Autowired
    private RatingService ratingService;

    @PostMapping("/login")
    public ResponseEntity<AuthResponse> login(@RequestBody
        AuthRequest request) {
        try {
            System.out.println("Richiesta login per: " + request.
                getEmail());

            // Validazione input
            if (request.getEmail() == null || request.getEmail().
                trim().isEmpty()) {
                return ResponseEntity.badRequest()
                        .body(new AuthResponse(false, "Email e'
                            obbligatoria"));
            }

            if (request.getPassword() == null || request.
                getPassword().trim().isEmpty()) {
                return ResponseEntity.badRequest()
                        .body(new AuthResponse(false, "Password e'
                            obbligatoria"));
            }

            // Tentativo autenticazione
            User user = userService.authenticateUser(request.
                getEmail(), request.getPassword());

            if (user != null) {
                System.out.println("Login riuscito per: " + user.
                    getDisplayName());
                return ResponseEntity.ok(
```

```
37                          new AuthResponse(true, "Login effettuato
                                con successo", user)
38                  );
39              } else {
40                  System.out.println("Login fallito per: " + request.
                        getEmail());
41                  return ResponseEntity.status(HttpStatus.
                        UNAUTHORIZED)
42                          .body(new AuthResponse(false, "Email o
                                password non corretti"));
43              }
44
45          } catch (Exception e) {
46              System.err.println("Errore durante il login: " + e.
                    getMessage());
47              e.printStackTrace();
48              return ResponseEntity.status(HttpStatus.
                    INTERNAL_SERVER_ERROR)
49                      .body(new AuthResponse(false, "Errore interno
                            del server"));
50          }
51      }
52
53      @PostMapping("/register")
54      public ResponseEntity<AuthResponse> register(@RequestBody
            RegisterRequest request) {
55          try {
56              System.out.println("Richiesta registrazione per: " +
                    request.getEmail());
57
58              // Validazione input completa
59              if (request.getName() == null || request.getName().trim
                    ().isEmpty()) {
60                  return ResponseEntity.badRequest()
61                          .body(new AuthResponse(false, "Nome e'
                                obbligatorio"));
62              }
63
64              if (request.getSurname() == null || request.getSurname
                    ().trim().isEmpty()) {
65                  return ResponseEntity.badRequest()
66                          .body(new AuthResponse(false, "Cognome e'
                                obbligatorio"));
67              }
68
69              if (request.getEmail() == null || request.getEmail().
                    trim().isEmpty()) {
70                  return ResponseEntity.badRequest()
71                          .body(new AuthResponse(false, "Email e'
                                obbligatoria"));
72              }
73
74              if (request.getPassword() == null || request.
                    getPassword().length() < 6) {
75                  return ResponseEntity.badRequest()
76                          .body(new AuthResponse(false, "Password
```

```
                        deve essere almeno 6 caratteri"));
77              }
78
79              // Controllo duplicati
80              if (userService.userExists(request.getEmail(), request.
                    getUsername())) {
81                  return ResponseEntity.status(HttpStatus.CONFLICT)
82                          .body(new AuthResponse(false, "Email o
                                username gia' in uso"));
83              }
84
85              // Registrazione
86              User newUser = userService.registerUser(
87                      request.getName(),
88                      request.getSurname(),
89                      request.getCf(),
90                      request.getEmail(),
91                      request.getUsername(),
92                      request.getPassword()
93              );
94
95              if (newUser != null) {
96                  System.out.println("Registrazione completata per: "
                        + newUser.getDisplayName());
97                  return ResponseEntity.status(HttpStatus.CREATED)
98                          .body(new AuthResponse(true, "Registrazione
                                completata con successo", newUser));
99              } else {
100                 return ResponseEntity.status(HttpStatus.
                        INTERNAL_SERVER_ERROR)
101                         .body(new AuthResponse(false, "Errore
                                durante la registrazione"));
102             }
103
104         } catch (Exception e) {
105             System.err.println("Errore durante la registrazione: "
                    + e.getMessage());
106             return ResponseEntity.status(HttpStatus.
                    INTERNAL_SERVER_ERROR)
107                     .body(new AuthResponse(false, "Errore interno
                            del server"));
108         }
109     }
110
111     @GetMapping("/profile/{userId}")
112     public ResponseEntity<AuthResponse> getUserProfile(
            @PathVariable String userId) {
113         try {
114             User user = userService.getUserById(userId);
115
116             if (user != null) {
117                 return ResponseEntity.ok(
118                         new AuthResponse(true, "Profilo recuperato"
                                , user)
119                 );
120             } else {
```

```
121                 return ResponseEntity.notFound().build();
122             }
123
124         } catch (Exception e) {
125             System.err.println("Errore recupero profilo: " + e.
                    getMessage());
126             return ResponseEntity.status(HttpStatus.
                    INTERNAL_SERVER_ERROR)
127                     .body(new AuthResponse(false, "Errore interno
                            del server"));
128         }
129     }
130
131     @GetMapping("/health")
132     public ResponseEntity<AuthResponse> healthCheck() {
133         boolean dbAvailable = userService.isDatabaseAvailable();
134
135         if (dbAvailable) {
136             return ResponseEntity.ok(
137                     new AuthResponse(true, "Auth Service is running
                            and database is connected!")
138             );
139         } else {
140             return ResponseEntity.status(HttpStatus.
                    SERVICE_UNAVAILABLE)
141                     .body(new AuthResponse(false, "Auth Service is
                            running but database is not available"));
142         }
143     }
144 }
```

Listing 4.5: AuthController - Endpoint Login

4.3.2 BookController - Gestione Catalogo

Il `BookController` gestisce tutte le operazioni sui libri:

```
@RestController
@RequestMapping("/api/books")
@CrossOrigin(origins = "*")
public class BookController {

  @Autowired
  private BookService bookService;
  @GetMapping
  public ResponseEntity<List<Book>> getAllBooks() {
    try {
      List<Book> books = bookService.getAllBooks();
      System.out.println("Catalogo: " + books.size() + " libri");
      return ResponseEntity.ok(books);

    } catch (Exception e) {
      System.err.print("Errore recupero: " + e.getMessage());

      return ResponseEntity.status(
        HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
  }

  @GetMapping("/search")
  public ResponseEntity<List<Book>> searchBooks(
    @RequestParam(value = "q", required = true) String query,
    @RequestParam(value = "category", required = false) String
       category) {

     try {
       if (query == null || query.trim().isEmpty()) {
          return ResponseEntity.badRequest().build();
        }

        String cleanQuery = query.trim();
        System.out.println("Ricerca avviata: '" + cleanQuery +
        (category != null ? "'categoria: '"+category+"'" : "'"));

        List<Book> results =
        bookService.searchBooks(cleanQuery, category);

        System.out.println("Risultati trovati: " + results.size());

        return ResponseEntity.ok(results);

     } catch (Exception e) {
        System.err.println("Errore ricerca: " + e.getMessage());
        return ResponseEntity.status(
            HttpStatus.INTERNAL_SERVER_ERROR).build();
     }
  }
}
```

Listing 4.6: BookController - Ricerca Libri

4.4 Pattern Architetturali Implementati

4.4.1 Service Layer Pattern

Il sistema implementa il Service Layer Pattern attraverso classi dedicate che incapsulano la business logic:

```
@Service
public class BookService {

   private static final String DB_URL =
   "jdbc:postgresql://localhost:5432/DataProva";

   private static final String DB_USER = "postgres";
   private static final String DB_PASSWORD = "postgress";

   public List<Book> getAllBooks() {
     List<Book> books = new ArrayList<>();

     try (Connection conn =
          DriverManager.getConnection(DB_URL,DB_USER,DB_PASSWORD);
          PreparedStatement stmt = conn.prepareStatement(
            "SELECT isbn, books_title, book_author, description,"+
            "publi_year, category, publisher
             FROM books
             ORDER BY books_title")) {

            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
              Book book = new Book(
                 rs.getString("isbn"),
                 rs.getString("books_title"),
                 rs.getString("book_author"),
                 rs.getString("description"),
                 rs.getString("publi_year"),
                 rs.getString("category")
             );
             books.add(book);
            }

            System.out.println("Caricati" + books.size());

        } catch (SQLException e) {
            System.err.println("Errore: " + e.getMessage());
            System.out.println("Utilizzo dati di fallback");

            books = getFallbackBooks();
        }

        return books;
    }

   public List<Book> searchBooks(String query, String category) {
      List<Book> results = new ArrayList<>();
```

```
49
50        // Costruzione query SQL dinamica
51        StringBuilder sql = new StringBuilder(
52          "SELECT isbn, books_title, book_author, description, " +
53                  "publi_year, category, publisher
54           FROM books WHERE ");
55
56        List<String> conditions = new ArrayList<>();
57        List<Object> parameters = new ArrayList<>();
58
59        // Ricerca full-text su titolo e autore
60         conditions.add("(LOWER(books_title)
61                           LIKE ? OR LOWER(book_author) LIKE ?)");
62        String searchPattern = "%" + query.toLowerCase() + "%";
63        parameters.add(searchPattern);
64        parameters.add(searchPattern);
65
66        // Filtro categoria opzionale
67        if (category != null && !category.trim().isEmpty()) {
68          conditions.add("LOWER(category) = ?");
69          parameters.add(category.toLowerCase());
70        }
71
72        sql.append(String.join(" AND ", conditions));
73        sql.append(" ORDER BY books_title LIMIT 50");
74
75        try (Connection conn =
76          DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
77          PreparedStatement stmt =
78          conn.prepareStatement(sql.toString())) {
79
80           // Impostazione parametri
81           for (int i = 0; i < parameters.size(); i++) {
82               stmt.setObject(i + 1, parameters.get(i));
83           }
84
85           ResultSet rs = stmt.executeQuery();
86
87           while (rs.next()) {
88              Book book = new Book(
89                  rs.getString("isbn"),
90                  rs.getString("books_title"),
91                  rs.getString("book_author"),
92                  rs.getString("description"),
93                  rs.getString("publi_year"),
94                  rs.getString("category")
95              );
96              results.add(book);
97           }
98
99        } catch (SQLException e) {
100           System.err.println("Errore: " + e.getMessage());
101       }
102
103       return results;
104   }
```

```
105 }
```

Listing 4.7: BookService - Service Layer Implementation

4.5 Comunicazione Client-Server

4.5.1 Architettura REST

Il sistema utilizza comunicazioni HTTP REST per l'interazione client-server:

Client HTTP Service

```
public class BookService {

  private static final String BASE_URL="http://localhost:8080/api";
  private final HttpClient httpClient;
  private final ObjectMapper objectMapper;

  public BookService() {
    this.httpClient = HttpClient.newBuilder()
        .connectTimeout(Duration.ofSeconds(10))
        .build();
    this.objectMapper = new ObjectMapper();
  }

  /**
  * Recupera tutti i libri dal server
  */
  public List<Book> getAllBooks() {
    try {
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(BASE_URL + "/books"))
            .header("Accept", "application/json")
            .GET()
            .build();

        HttpResponse<String> response = httpClient.send(request,
                HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() == 200) {
            Book[] booksArray = objectMapper.readValue(
                response.body(), Book[].class);
            return Arrays.asList(booksArray);
        } else {
            System.err.print("Errore HTTP:"+response.statusCode());
            return getFallbackBooks();
        }

    } catch (Exception e) {
        System.err.print("Errore comunicazione:"+ e.getMessage());
        return getFallbackBooks();
    }
 }

    /**
     * Test connessione server
     */
    public boolean testConnection() {
```

```
47          try {
48              HttpRequest request = HttpRequest.newBuilder()
49                      .uri(URI.create(BASE_URL + "/books"))
50                      .header("Accept", "application/json")
51                      .timeout(Duration.ofSeconds(5))
52                      .GET()
53                      .build();
54
55              HttpResponse<String> response =
56              httpClient.send(
57                  request,HttpResponse.BodyHandlers.ofString());
58
59              boolean isConnected = response.statusCode() == 200;
60              System.out.println(isConnected ?
61                              "Connessione server OK" :
62                              "Server non raggiungibile");
63
64              return isConnected;
65
66          } catch (Exception e) {
67            System.out.print("Connessione fallito:"+ e.getMessage());
68            return false;
69          }
70      }
71
72      private List<Book> getFallbackBooks() {
73          List<Book> books = new ArrayList<>();
74          books.add(new Book("978-0-12-345678-9",
75                             "Java: The Complete Reference","Herbert
76                             Schildt","Guida completa a Java",
77                             "2023","Programming"));
78          return books;
79      }
80  }
```

Listing 4.8: BookService Client - HTTP Communication

4.5.2 Authentication Manager

Gestione centralizzata dell'autenticazione client-side:

```
public class AuthenticationManager {

    /** Stato di autenticazione corrente dell'utente */
    private boolean isAuthenticated = false;

    /** Informazioni dell'utente attualmente autenticato */
    private User currentUser = null;

    /** Pannello UI per le operazioni di autenticazione */
    private AuthPanel authPanel;

    /** Servizio backend per operazioni di autenticazione */
    private AuthService authService;

    /** Callback per notificare cambiamenti di stato auth */
    private Runnable onAuthStateChanged;

    public AuthenticationManager() {
        this.authService = new AuthService();
    }

    /**
     * Imposta il callback per ricevere notifiche sui cambiamenti
     * di stato di autenticazione
     */
    public void setOnAuthStateChanged(Runnable callback) {
        this.onAuthStateChanged = callback;
    }

    /**
     * Verifica se l'utente e' attualmente autenticato
     */
    public boolean isAuthenticated() {
        return isAuthenticated;
    }

    /**
     * Ottiene l'oggetto User dell'utente attualmente autenticato
     */
    public User getCurrentUser() {
        return currentUser;
    }

    /**
     * Ottiene lo username dell'utente attualmente autenticato
     */
    public String getCurrentUsername() {
        return currentUser != null ? currentUser.getUsername() :
            null;
    }

    /**
```

```
52      * Ottiene il nome di visualizzazione dell'utente corrente
53      */
54     public String getCurrentUserDisplayName() {
55         return currentUser != null ? currentUser.getDisplayName() :
               "Utente";
56     }
57
58     /**
59      * Mostra il pannello di autenticazione in modalita' overlay
60      */
61     public void showAuthPanel(StackPane mainRoot) {
62         if (mainRoot == null) {
63             throw new IllegalArgumentException(
64                 "Il container principale non puo' essere null");
65         }
66
67         authPanel = new AuthPanel();
68         authPanel.setOnSuccessfulAuth(this::
               handleSuccessfulAuthentication);
69         authPanel.setOnClosePanel(() -> closeAuthPanel(mainRoot));
70
71         StackPane overlay = new StackPane();
72         overlay.setStyle("-fx-background-color: rgba(0, 0, 0, 0.7);
               ");
73         overlay.getChildren().add(authPanel);
74         StackPane.setAlignment(authPanel, Pos.CENTER);
75
76         // Gestione click esterno per chiusura
77         overlay.setOnMouseClicked(e -> {
78             if (e.getTarget() == overlay) {
79                 closeAuthPanel(mainRoot);
80             }
81         });
82
83         mainRoot.getChildren().add(overlay);
84         System.out.println("Pannello autenticazione aperto");
85     }
86
87     /**
88      * Esegue il logout dell'utente corrente
89      */
90     public void logout() {
91         performLogout();
92     }
93
94     /**
95      * Aggiorna lo stato di autenticazione e notifica i listener
96      */
97     public void setAuthenticationState(boolean authenticated, User
           user) {
98         boolean wasAuthenticated = this.isAuthenticated;
99
100        this.isAuthenticated = authenticated;
101        this.currentUser = user;
102
103        if (authenticated && user != null) {
```

```
104             System.out.println("Utente autenticato: " + user.
                    getDisplayName());
105         } else {
106             System.out.println("Utente disconnesso");
107         }
108 
109         if (wasAuthenticated != authenticated) {
110             notifyAuthStateChanged();
111         }
112     }
113 
114     /**
115      * Verifica la disponibilita' del servizio di autenticazione
116      */
117     public void checkAuthServiceHealth() {
118         authService.healthCheckAsync()
119             .thenAccept(response -> {
120                 Platform.runLater(() -> {
121                     if (response.isSuccess()) {
122                         System.out.println("Servizio autenticazione
                                : " +
123                                            response.getMessage());
124                     } else {
125                         System.out.println("Servizio autenticazione
                                non disponibile: " +
126                                            response.getMessage());
127                     }
128                 });
129             })
130             .exceptionally(throwable -> {
131                 Platform.runLater(() -> {
132                     System.out.println("Errore connessione servizio
                            auth: " +
133                                        throwable.getMessage());
134                 });
135                 return null;
136             });
137     }
138 
139     /**
140      * Aggiorna le informazioni dell'utente attualmente autenticato
141      */
142     public void updateCurrentUser(User updatedUser) {
143         if (this.isAuthenticated && updatedUser != null) {
144             this.currentUser = updatedUser;
145             System.out.println("Profilo utente aggiornato: " +
146                                updatedUser.getDisplayName());
147             notifyAuthStateChanged();
148         }
149     }
150 
151     /**
152      * Inizializza il manager di autenticazione
153      */
154     public void initialize() {
155         System.out.println("Inizializzazione AuthenticationManager
```

```
            ...");
156         checkAuthServiceHealth();
157         System.out.println("AuthenticationManager inizializzato");
158     }
159
160     /**
161      * Cleanup risorse alla chiusura dell'applicazione
162      */
163     public void shutdown() {
164         System.out.println("Shutdown AuthenticationManager...");
165
166         if (isAuthenticated) {
167             authService.logoutAsync()
168                 .thenAccept(response -> {
169                     System.out.println("Logout silenzioso
                            completato");
170                 })
171                 .exceptionally(throwable -> {
172                     System.out.println("Logout silenzioso fallito:
                            " +
173                                        throwable.getMessage());
174                     return null;
175                 });
176         }
177
178         System.out.println("AuthenticationManager chiuso");
179     }
180
181     // Metodi privati per gestione interna
182     private void handleSuccessfulAuthentication(User user) {
183         setAuthenticationState(true, user);
184         Platform.runLater(() -> showWelcomeMessage(user));
185     }
186
187     private void performLogout() {
188         String userDisplayName = getCurrentUserDisplayName();
189         System.out.println("Esecuzione logout per: " +
                userDisplayName);
190
191         authService.logoutAsync()
192             .thenAccept(response -> Platform.runLater(() -> {
193                 if (response.isSuccess()) {
194                     System.out.println("Logout completato sul
                            server");
195                 } else {
196                     System.out.println("Logout locale (server non
                            risponde)");
197                 }
198                 setAuthenticationState(false, null);
199             }))
200             .exceptionally(throwable -> {
201                 Platform.runLater(() -> {
202                     System.out.println("Logout locale (errore
                            server): " +
203                                        throwable.getMessage());
204                     setAuthenticationState(false, null);
```

```
205                 });
206                 return null;
207             });
208     }
209
210     private void notifyAuthStateChanged() {
211         if (onAuthStateChanged != null) {
212             Platform.runLater(() -> {
213                 try {
214                     onAuthStateChanged.run();
215                 } catch (Exception e) {
216                     System.err.println("Errore nel callback auth
                        state changed: " +
217                                        e.getMessage());
218                 }
219             });
220         }
221     }
222
223     private void closeAuthPanel(StackPane mainRoot) {
224         if (mainRoot.getChildren().size() > 1) {
225             mainRoot.getChildren().remove(mainRoot.getChildren().
                size() - 1);
226         }
227         System.out.println("Pannello autenticazione chiuso");
228     }
229 }
```

Listing 4.9: AuthenticationManager - Gestione Stato Utente

4.6 Conclusioni

L'architettura del sistema BABO Library dimostra l'implementazione efficace di pattern architetturali consolidati per creare un'applicazione robusta, scalabile e manutenibile.
La separazione in moduli Maven, l'uso di pattern come Service Layer, insieme alle strategie di resilienza e caching, garantiscono un sistema affidabile e performante.
Le principali caratteristiche architetturali includono:

- **Modularità**: Separazione chiara delle responsabilità tra client, server e shared
- **Resilienza**: Strategie di fallback e gestione errori robusta
- **Performance**: Sistema di cache intelligente e lazy loading
- **Sicurezza**: Validazione input e gestione autenticazione centralizzata
- **Manutenibilità**: Pattern consolidati e logging strutturato

Questa architettura fornisce una solida base per future estensioni e manutenzione del sistema.

Capitolo 5

Documentazione API REST

5.1 Panoramica delle API

Il server BABO espone un'architettura API REST completa per tutte le funzionalità del sistema, organizzata secondo i principi *RESTful* e strutturata per area funzionale. L'API è implementata utilizzando `Spring Boot 3.2.0` con supporto per `CORS cross-origin` e serializzazione JSON tramite Jackson.

5.1.1 Architettura Generale

L'API segue il pattern *MVC (Model-View-Controller)* con separazione netta tra:

- **Controller Layer**: Gestione endpoint REST e validazione input.
- **Service Layer**: Logica business e orchestrazione operazioni.
- **Model/DTO Layer**: Trasferimento dati *type-safe* tra client e server.

5.1.2 Convenzioni URL

Tutti gli endpoint seguono la struttura base:
`http://localhost:8080/api/{risorsa}/{operazione}`.

5.1.3 Endpoint per Area Funzionale

Autenticazione (`/api/auth`)

POST /api/auth/login Autenticazione utente.

POST /api/auth/register Registrazione nuovo utente.

POST /api/auth/reset-password Reset password.

POST /api/auth/change-password/{userId} Cambio password.

GET /api/auth/profile/{userId} Dettagli profilo utente.

PUT /api/auth/profile/{userId} Aggiornamento profilo.

PUT /api/auth/update-email/{userId} Aggiornamento email.

GET /api/auth/check-availability Verifica disponibilità email/username.

POST /api/auth/logout Logout utente.

Endpoint Amministrativi

GET /api/auth/admin/users Lista utenti (admin).

DELETE /api/auth/admin/users/{userId} Eliminazione utente (admin).

GET /api/auth/admin/books Gestione libri (admin).

POST /api/auth/admin/books Aggiunta libro (admin).

PUT /api/auth/admin/books/{isbn} Aggiornamento libro (admin).

DELETE /api/auth/admin/books/{isbn} Eliminazione libro (admin).

GET /api/auth/admin/ratings Gestione recensioni (admin).

Libri (`/api/books`)

GET /api/books Catalogo completo libri.

GET /api/books/{id} Dettagli libro specifico.

GET /api/books/search Ricerca libri con query.

GET /api/books/category Filtraggio per categoria.

GET /api/books/featured Libri in evidenza.

GET /api/books/new-releases Nuove uscite.

GET /api/books/most-reviewed Libri più recensiti.

GET /api/books/top-rated Libri meglio valutati.

GET /api/books/search/title Ricerca per titolo.

GET /api/books/search/author Ricerca per autore.

GET /api/books/search/author-year Ricerca autore-anno.

Librerie (`/api/library`)

POST /api/library/create Creazione nuova libreria.

GET /api/library/user/{username} Librerie utente.

GET /api/library/books/{username}/{namelib} Libri in libreria.

POST /api/library/add-book Aggiunta libro a libreria.

DELETE /api/library/remove-book Rimozione libro da libreria.

DELETE /api/library/delete/{username}/{namelib} Eliminazione libreria.

PUT /api/library/rename/{username}/{oldName}/{newName} Rinomina libreria.

GET /api/library/user/{username}/owns/{isbn} Verifica possesso libro.

GET /api/library/stats/{username} Statistiche librerie utente.

Valutazioni (`/api/ratings`)

POST /api/ratings/add Aggiunta/aggiornamento valutazione.

GET /api/ratings/user/{username}/book/{isbn} Valutazione utente per libro.

GET /api/ratings/user/{username} Tutte le valutazioni utente.

GET /api/ratings/book/{isbn} Valutazioni per libro.

GET /api/ratings/book/{isbn}/statistics Statistiche libro.

DELETE /api/ratings/user/{username}/book/{isbn} Eliminazione valutazione.

GET /api/ratings/most-reviewed-books Libri più recensiti.

GET /api/ratings/best-rated-books Libri meglio valutati.

POST /api/ratings/validate Validazione valutazione.

GET /api/ratings/stats/{username} Statistiche utente.

GET /api/ratings/top-rated Classifiche libri.

Raccomandazioni (`/api/recommendations`)

POST /api/recommendations/add Aggiunta raccomandazione.

GET /api/recommendations/book/{isbn} Raccomandazioni per libro.

GET /api/recommendations/can-recommend/{username}/{isbn} Verifica permessi.

GET /api/recommendations/user/{username}/book/{isbn} Raccomandazioni utente.

DELETE /api/recommendations/remove Rimozione raccomandazione.

GET /api/recommendations/stats Statistiche sistema.

GET /api/recommendations/stats/{username} Statistiche utente.

5.1.4 Codici di Stato HTTP

L'API utilizza i seguenti codici di stato standard:

- **200 OK**: Operazione completata con successo.
- **201 CREATED**: Nuova risorsa creata.
- **400 BAD REQUEST**: Dati input non validi.
- **401 UNAUTHORIZED**: Credenziali errate.
- **403 FORBIDDEN**: Accesso negato (privilegi insufficienti).
- **404 NOT FOUND**: Risorsa non trovata.
- **409 CONFLICT**: Violazione vincoli (duplicati).
- **500 INTERNAL SERVER ERROR**: Errore del server.
- **503 SERVICE UNAVAILABLE**: Servizio non disponibile.

5.2 API Autenticazione

Il sistema di autenticazione fornisce endpoint per registrazione, login, gestione profili e operazioni amministrative. La sicurezza è implementata tramite hashing SHA-256 delle password e controlli di autorizzazione basati su whitelist email per gli amministratori.

5.2.1 Autenticazione Utente

Endpoint: `POST /api/auth/login`

Request Body:

```
{
  "email": "mario.rossi@email.com",
  "password": "password123"
}
```

Response Successo (200 OK):

```
{
  "success": true,
  "message": "Login effettuato con successo",
  "user": {
    "id": "1",
    "name": "Mario",
    "surname": "Rossi",
    "email": "mario.rossi@email.com",
    "username": "mario_rossi",
    "cf": "RSSMRA80A01H501U"
  }
}
```

Response Errore (401 Unauthorized):

```
{
  "success": false,
  "message": "Email o password non corretti"
}
```

5.2.2 Registrazione Nuovo Utente

Endpoint: `POST /api/auth/register`

Request Body:

```
{
  "name": "Mario",
  "surname": "Rossi",
  "cf": "RSSMRA80A01H501U",
  "email": "mario.rossi@email.com",
  "username": "mario_rossi",
  "password": "password123"
}
```

Response Successo (201 Created):

```
{
  "success": true,
  "message": "Registrazione completata con successo",
  "user": {
    "id": "15",
    "name": "Mario",
    "surname": "Rossi",
    "email": "mario.rossi@email.com",
    "username": "mario_rossi",
    "cf": "RSSMRA80A01H501U"
  }
}
```

Validazioni Implementate: Nome e cognome obbligatori, Email formato valido e univoca, Username univoco, Password minimo 6 caratteri, Codice fiscale normalizzato (uppercase).

5.2.3 Reset Password

Endpoint: `POST /api/auth/reset-password`

Request Body:

```
{
  "email": "mario.rossi@email.com",
  "newPassword": "nuovaPassword123"
}
```

Response:

```
{
  "success": true,
  "message": "Password reimpostata con successo"
}
```

5.2.4 Cambio Password

Endpoint: `POST /api/auth/change-password/{userId}`

Request Body:

```
{
  "oldPassword": "password123",
  "newPassword": "nuovaPassword456"
}
```

Response:

```
{
  "success": true,
  "message": "Password cambiata con successo"
}
```

5.2.5 Gestione Profilo

Recupero Profilo: `GET /api/auth/profile/{userId}`

Response:

```
{
  "success": true,
  "message": "Profilo recuperato",
  "user": {
    "id": "1",
    "name": "Mario",
    "surname": "Rossi",
    "email": "mario.rossi@email.com",
    "username": "mario_rossi",
    "cf": "RSSMRA80A01H501U"
  }
}
```

Aggiornamento Email: `PUT /api/auth/update-email/{userId}`

Request Body:

```
{
  "email": "nuovo@email.com"
}
```

5.3 API Libri

Il sistema di gestione libri fornisce un'API completa per ricerca, catalogazione e discovery di contenuti bibliografici con supporto per filtri avanzati e metriche di popolarità.

5.3.1 Catalogo Completo

Endpoint: `GET /api/books`

Response:

```
[
  {
    "id": 1,
    "isbn": "9788804660347",
    "title": "Il Nome della Rosa",
    "author": "Umberto Eco",
    "description": "Romanzo storico ambientato in un'abbazia
       medievale",
    "category": "Narrativa",
    "publishYear": "1980",
    "imageUrl": "9788804660347.jpg",
    "reviewCount": 127,
    "averageRating": 4.3
  }
]
```

5.3.2 Ricerca Avanzata

Ricerca Generale

Endpoint: `GET /api/books/search`
Parametri:

- `q` (required): Query di ricerca

Esempio:

`GET /api/books/search?q=java programming`

Ricerca per Categoria

Endpoint: `GET /api/books/category`
Parametri:

- `name` (required): Nome categoria

Esempio:

```
GET /api/books/category?name=Informatica
```

Ricerca Autore-Anno

Endpoint: `GET /api/books/search/author-year`
Parametri:

- `author` (required): Nome autore
- `year` (optional): Anno pubblicazione

Esempio:

```
GET /api/books/search/author-year?author=Eco&year=1980
```

5.4 API Librerie

Il sistema di gestione librerie permette agli utenti di organizzare raccolte personalizzate di libri con operazioni *CRUD* complete e controlli di proprietà.

5.4.1 Creazione Libreria

Endpoint: `POST /api/library/create`

Request Body:

```
{
  "username": "mario_rossi",
  "namelib": "Fantascienza"
}
```

Response (201 Created):

```
{
  "success": true,
  "message": "Libreria creata con successo"
}
```

5.4.2 Gestione Contenuti

Aggiunta Libro: `POST /api/library/add-book`

Request Body:

```
{
  "username": "mario_rossi",
  "namelib": "Fantascienza",
  "isbn": "9780441569595"
}
```

Rimozione Libro: `DELETE /api/library/remove-book`

Request Body:

```
{
  "username": "mario_rossi",
  "namelib": "Fantascienza",
  "isbn": "9780441569595"
}
```

5.4.3 Consultazione Librerie

Lista Librerie Utente: `GET /api/library/user/{username}`

Response:

```
{
  "success": true,
  "message": "Librerie recuperate con successo",
  "libraries": [
    "Fantascienza",
    "Gialli",
    "Saggistica"
  ]
}
```

Contenuto Libreria: `GET /api/library/books/{username}/{namelib}`

Response:

```
{
  "success": true,
  "message": "Libri recuperati con successo",
  "books": [
    {
      "id": 1,
      "isbn": "9780441569595",
      "title": "Dune",
      "author": "Frank Herbert",
      "description": "Epic di fantascienza su Arrakis",
      "category": "Fantascienza"
    }
  ]
}
```

5.4.4 Controllo Proprietà

Verifica Possesso: `GET /api/library/user/{username}/owns/{isbn}`

Response:

```
{
  "success": true,
  "message": "Utente possiede il libro"
}
```

5.5 API Valutazioni

Il sistema di valutazioni implementa un modello multi-dimensionale con rating su 5 categorie (*stile, contenuto, piacevolezza, originalità, edizione*) e recensioni testuali opzionali.

5.5.1 Aggiunta Valutazione

Endpoint: `POST /api/ratings/add`

Request Body:

```
{
  "username": "mario_rossi",
  "isbn": "9788804660347",
  "style": 4,
  "content": 5,
  "pleasantness": 4,
  "originality": 3,
  "edition": 4,
  "review": "Libro eccellente con trama avvincente e personaggi ben
      sviluppati. La scrittura di Eco e' magistrale."
}
```

Response (200 OK):

```
{
  "success": true,
  "message": "Valutazione salvata con successo",
  "rating": {
    "id": 1,
    "username": "mario_rossi",
    "isbn": "9788804660347",
    "style": 4,
    "content": 5,
    "pleasantness": 4,
    "originality": 3,
    "edition": 4,
    "average": 4.0,
    "review": "Libro eccellente con trama avvincente...",
    "data": "2024-01-15T14:30:00"
  }
}
```

5.5.2 Consultazione Valutazioni

Valutazione Specifica: `GET /api/ratings/user/{username}/book/{isbn}`
Valutazioni Utente: `GET /api/ratings/user/{username}`
Valutazioni Libro: `GET /api/ratings/book/{isbn}`

Response Valutazioni Libro:

```
{
  "success": true,
  "message": "Valutazioni recuperate con successo",
  "ratings": [
    {
      "username": "mario_rossi",
      "isbn": "9788804660347",
      "average": 4.0,
      "review": "Libro eccellente...",
      "data": "2024-01-15T14:30:00"
    }
  ],
  "averageRating": 4.2,
  "totalRatings": 15
}
```

5.5.3 Statistiche Libro

Endpoint: `GET /api/ratings/book/{isbn}/statistics`

Response:

```
{
  "success": true,
  "message": "Statistiche recuperate con successo",
  "ratings": [],
  "averageRating": 4.2,
  "totalRatings": 127
}
```

5.5.4 Validazione

Endpoint: `POST /api/ratings/validate`
Request Body:

```
{
  "username": "mario_rossi",
  "isbn": "9788804660347",
  "style": 4,
  "content": 5,
  "pleasantness": 4,
  "originality": 3,
  "edition": 4
}
```

Response:

```
{
  "success": true,
  "message": "Valutazione valida. Media calcolata: 4.0"
}
```

5.6 API Raccomandazioni

Il sistema di raccomandazioni *peer-to-peer* permette agli utenti di suggerire libri ad altri membri della community con controlli di proprietà e limiti anti-spam.

5.6.1 Aggiunta Raccomandazione

Endpoint: `POST /api/recommendations/add`

Request Body:

```
{
  "username": "mario_rossi",
  "targetBookIsbn": "9788804660347",
  "recommendedBookIsbn": "9780441569595",
  "reason": "Se ti e' piaciuto Il Nome della Rosa, amerai anche
     Dune per la complessita' dei mondi narrativi costruiti"
}
```

Response (201 Created):

```
{
  "success": true,
  "message": "Raccomandazione aggiunta con successo",
  "recommendation": {
    "recommenderUsername": "mario_rossi",
    "targetBookIsbn": "9788804660347",
    "recommendedBookIsbn": "9780441569595",
    "reason": "Se ti e' piaciuto Il Nome della Rosa..."
  }
}
```

5.6.2 Consultazione Raccomandazioni

Raccomandazioni per Libro: `GET /api/recommendations/book/{isbn}`

Response:

```
{
  "success": true,
  "message": "Raccomandazioni recuperate con successo",
  "recommendations": [
    {
      "recommenderUsername": "mar***",
      "targetBookIsbn": "9788804660347",
      "recommendedBookIsbn": "9780441569595",
      "reason": "Se ti e' piaciuto Il Nome della Rosa...",
      "createdDate": "2024-01-15"
    }
  ],
  "recommendedBooks": [
    {
      "isbn": "9780441569595",
      "title": "Dune",
      "author": "Frank Herbert",
      "description": "Epic di fantascienza"
    }
  ]
}
```

5.6.3 Controllo Permessi

Endpoint: `GET /api/recommendations/can-recommend/{username}/{isbn}`

Response:

```
{
  "success": true,
  "message": "Puoi aggiungere ancora 2 raccomandazioni",
  "canRecommend": true,
  "currentRecommendationsCount": 3,
  "maxRecommendations": 5
}
```

5.6.4 Rimozione Raccomandazione

Endpoint: `DELETE /api/recommendations/remove`

Request Body:

```
{
  "username": "mario_rossi",
  "targetBookIsbn": "9788804660347",
  "recommendedBookIsbn": "9780441569595"
}
```

5.7 Configurazione Server

5.7.1 Stack Tecnologico

Framework `Spring Boot 3.2.0`, `Spring Web MVC`, `Jackson 2.15.2` per serializzazione JSON.

Database `PostgreSQL 42.7.0`.

Connessione `jdbc:postgresql://localhost:5432/DataProva`.

Credenziali `postgres/postgress`.

Build `Maven 3.11.0`.

Java `Java 17`.

Encoding `UTF-8`.

5.7.2 Struttura Progetto

Figura 5.1: Struttura del Progetto

```
server/
    src/main/java/org/BABO/server/
        controller/ ..... Controller REST
            AuthController.java
            BookController.java
            LibraryController.java
            RatingController.java
            RecommendationController.java
        service/ ..... Logica business
        ServerApplication.java
    pom.xml
shared/
    src/main/java/org/BABO/shared/
        dto/ ..... Data Transfer Objects
            Authentication/
            Library/
            Rating/
            Recommendation/
        model/ ..... Modelli dati
```

5.7.3 Configurazione CORS

Tutti i controller sono configurati con `@CrossOrigin(origins = "*")` per supportare richieste *cross-origin* da qualsiasi dominio. Per ambiente produttivo si raccomanda di limitare gli origin specifici.

5.7.4 Gestione Errori

Il sistema implementa gestione degli errori strutturata con:

- `Try-catch` completo in tutti gli endpoint.
- Logging dettagliato per debugging.
- Response standardizzate con codici HTTP appropriati.
- Messaggi *user-friendly* senza esposizione dettagli sistema.

5.7.5 Health Check Endpoints

Ogni controller espone un endpoint di diagnostica:

GET /api/auth/health Stato servizio autenticazione.

GET /api/books/health Stato servizio libri.

GET /api/library/health Stato servizio librerie.

GET /api/ratings/health Stato servizio valutazioni.

GET /api/recommendations/health Stato servizio raccomandazioni.

5.8 Best Practices

5.8.1 Sicurezza

- Validare sempre tutti gli input lato server.
- Non includere mai password nelle response JSON.
- Implementare *rate limiting* per ambiente produttivo.
- Utilizzare *HTTPS* per traffico di produzione.
- Configurare CORS appropriatamente per dominio specifico.

5.8.2 Performance

- Implementare caching per endpoint *read-heavy*.
- Utilizzare *connection pooling* database.
- Paginare risultati per query che restituiscono molti dati.
- Ottimizzare query database con indici appropriati.

5.8.3 Monitoraggio

- Implementare logging strutturato.
- Configurare *health check endpoints*.
- Monitorare metriche performance.
- Implementare *alerting* per errori critici.

5.8.4 Documentazione

- Mantenere documentazione API aggiornata.
- Fornire esempi di utilizzo completi.
- Documentare tutti i codici di errore.
- Specificare format dati e validazioni richieste.

Capitolo 6

Applicazione Client JavaFX

6.1 Architettura Client

L'applicazione client BABO utilizza JavaFX 17 con un'architettura modulare basata sui pattern *MVVM (Model-View-ViewModel)* e *Observer* per garantire separazione delle responsabilità, manutenibilità e scalabilità del codice.

6.1.1 Struttura Package Client

Il client è organizzato secondo una struttura gerarchica che riflette la separazione tra logica di presentazione, business logic e comunicazione con il server:

```
org.BABO.client/
├── ClientApplication.java ..... Main class and Application Controller
├── ui/ ..... Controllers JavaFX
│   ├── Admin/
│   │   └── AdminPanel.java ..... Pannello amministrazione
│   ├── Authentication/
│   │   ├── AuthenticationManager.java
│   │   └── AuthPanel.java ..... Login/registrazione
│   ├── Book/
│   │   ├── BookGridBuilder.java
│   │   ├── BookSectionFactory.java
│   │   └── FeaturedBookBuilder.java
│   ├── Category/
│   │   └── CategoryView.java ..... Vista categorie
│   ├── Home/
│   │   ├── ApplicationProtection.java
│   │   ├── ContentArea.java ..... Area contenuti
│   │   ├── ExploreIntegration.java
│   │   ├── Header.java ..... Intestazione ricerca
│   │   ├── IconUtils.java
│   │   ├── ImageUtils.java
│   │   ├── MainWindow.java ..... Finestra principale
│   │   └── Sidebar.java ..... Barra navigazione
│   ├── Library/
│   │   └── LibraryPanel.java ..... Librerie personali
│   ├── Popup/
│   │   ├── BookDetailsPopup.java
│   │   ├── PopupManager.java
│   │   └── UserProfilePopup.java
│   ├── Rating/
│   │   └── RatingDialog.java ..... Dialog valutazioni
│   ├── Recommendation/
│   │   └── RecommendationDialog.java
│   └── Search/
│       └── AdvancedSearchPanel.java
└── service/ ..... Client Services
    ├── AdminService.java
    ├── AuthService.java
    ├── BookService.java
    ├── ClientRatingService.java
    ├── ClientRecommendationService.java
    └── LibraryService.java
```

6.2 Componenti Principali dell'Interfaccia

6.2.1 BooksClient - Application Controller

La classe principale che estende `Application` e coordina l'intero sistema:

```
/**
 * Punto di ingresso principale per l'applicazione client JavaFX.
 * Coordina inizializzazione, configurazione e gestione del ciclo di vita.
 */
public class BooksClient extends Application {

    private BookService bookService;
    private boolean serverAvailable = false;
    private MainWindow mainWindow;

    @Override
    public void init() {
        System.out.println("Inizializzazione client...");
        bookService = new BookService();

        // Verifica disponibilita' server
        serverAvailable = bookService.isServerAvailable();
        if (serverAvailable) {
            System.out.println("Server raggiungibile");
        } else {
            System.out.println("Server non raggiungibile - modalita' offline");
        }
    }

    @Override
    public void start(Stage stage) {
        System.out.println("Avvio BooksClient con PopupManager integrato");

        try {
            // 1. Registra ApplicationProtection
            ApplicationProtection.registerMainStage(stage);
            System.out.println("Protezione applicazione attivata");

            // 2. Imposta icona applicazione
            setupApplicationIcon(stage);

            // 3. Crea la finestra principale
            System.out.println("Creazione interfaccia utente...");
            mainWindow = new MainWindow(bookService, serverAvailable);
            StackPane root = mainWindow.createMainLayout();

            // 4. Inizializza PopupManager con il root di MainWindow
            PopupManager popupManager = PopupManager.getInstance();
            popupManager.initialize(root);
```

```
45              System.out.println("PopupManager inizializzato con
                    MainWindow");
46
47              // 5. Setup scena
48              Scene scene = new Scene(root, 1300, 800);
49
50              // Carica CSS se disponibile
51              try {
52                  scene.getStylesheets().add(
53                      getClass().getResource("/css/style.css").
                            toExternalForm()
54                  );
55                  scene.getStylesheets().add(
56                      getClass().getResource("/css/scrollbar.css").
                            toExternalForm()
57                  );
58                  scene.getStylesheets().add(
59                      getClass().getResource("/css/auth-tabs.css").
                            toExternalForm()
60                  );
61              } catch (Exception e) {
62                  System.out.println("CSS non trovato, uso stili
                        default");
63              }
64
65              stage.setScene(scene);
66              stage.setTitle("Books Client ");
67
68              stage.setMinWidth(1300);
69              stage.setMinHeight(800);
70              stage.setWidth(1300);
71              stage.setHeight(800);
72
73              // 6. Gestione chiusura
74              stage.setOnCloseRequest(e -> {
75                  System.out.println("Richiesta chiusura applicazione
                        ...");
76                  handleApplicationClose();
77              });
78
79              // 7. Setup eventi post-mostrazione
80              stage.setOnShown(e -> {
81                  System.out.println("Interfaccia avviata con
                        successo!");
82                  System.out.println("Dimensioni finestra: " +
83                      stage.getWidth() + "x" + stage.getHeight());
84
85                  // Debug iniziale se in modalita' debug
86                  if (isDebugMode()) {
87                      ApplicationProtection.debugApplicationState();
88                      popupManager.debugFullState();
89                      testPopupManagerSetup();
90                  }
91              });
92
93              // 8. Mostra applicazione
```

```
94              stage.show();
95              stage.centerOnScreen();
96
97              System.out.println("BooksClient avviato con successo");
98
99          } catch (Exception e) {
100             System.err.println("Errore fatale nell'avvio: " + e.
                    getMessage());
101             e.printStackTrace();
102             showStartupError(e);
103             Platform.exit();
104         }
105     }
106
107     @Override
108     public void stop() {
109         System.out.println("Stop applicazione...");
110
111         try {
112             // Cleanup servizi
113             if (bookService != null) {
114                 bookService.shutdown();
115             }
116
117             // Cleanup finale PopupManager
118             PopupManager.getInstance().emergencyReset();
119
120         } catch (Exception e) {
121             System.err.println("Errore durante stop: " + e.
                    getMessage());
122         }
123
124         System.out.println("Stop completato");
125     }
126
127     private void setupApplicationIcon(Stage stage) {
128         System.out.println("Configurazione icona applicazione cross
                -platform...");
129
130         try {
131             // Imposta icona per la finestra principale
132             IconUtils.setStageIcon(stage);
133
134             // Imposta icona per il sistema (dock/taskbar)
135             IconUtils.setSystemTrayIcon();
136
137             // Debug info
138             System.out.println(IconUtils.getIconInfo());
139
140             // Verifica compatibilita'
141             debugCrossPlatformCompatibility();
142
143         } catch (Exception e) {
144             System.err.println("Errore nel setup icona applicazione
                    : " + e.getMessage());
145             e.printStackTrace();
```

```
146         }
147     }
148
149     private void handleApplicationClose() {
150         try {
151             System.out.println("Inizio procedura chiusura...");
152
153             // Chiudi tutti i popup tramite PopupManager
154             PopupManager popupManager = PopupManager.getInstance();
155             if (popupManager.hasActivePopups()) {
156                 System.out.println("Chiusura popup aperti...");
157                 popupManager.closeAllPopups();
158
159                 // Attendi un momento per la chiusura
160                 Platform.runLater(() -> {
161                     try {
162                         Thread.sleep(200);
163                     } catch (InterruptedException ex) {
164                         Thread.currentThread().interrupt();
165                     }
166                     finalizeApplicationClose();
167                 });
168             } else {
169                 finalizeApplicationClose();
170             }
171
172         } catch (Exception e) {
173             System.err.println("Errore durante chiusura: " + e.
                    getMessage());
174             finalizeApplicationClose();
175         }
176     }
177
178     private void finalizeApplicationClose() {
179         try {
180             // Cleanup cache immagini
181             ImageUtils.clearImageCache();
182
183             // Cleanup MainWindow se ha metodi di cleanup
184             if (mainWindow != null && mainWindow.getAuthManager()
                    != null) {
185                 mainWindow.getAuthManager().shutdown();
186                 System.out.println("MainWindow cleanup completato")
                        ;
187             }
188
189             System.out.println("Chiusura completata correttamente")
                    ;
190             Platform.exit();
191
192         } catch (Exception e) {
193             System.err.println("Errore nella finalizzazione: " + e.
                    getMessage());
194             Platform.exit();
195         }
196     }
```

```
197
198     public static void openBookDetails(Book book, List<Book>
            collection,
199                                        AuthenticationManager
                                               authManager) {
200         if (book == null) {
201             System.err.println("openBookDetails: libro null");
202             return;
203         }
204
205         System.out.println("Apertura dettagli libro: " + book.
                getTitle());
206
207         try {
208             PopupManager popupManager = PopupManager.getInstance();
209
210             if (!popupManager.isInitialized()) {
211                 System.err.println("PopupManager non inizializzato!
                        ");
212                 return;
213             }
214
215             popupManager.showBookDetails(book, collection,
                    authManager);
216             System.out.println("Popup libro aperto tramite
                    PopupManager");
217
218         } catch (Exception e) {
219             System.err.println("Errore apertura popup: " + e.
                    getMessage());
220             e.printStackTrace();
221             showError("Errore nell'apertura dei dettagli del libro:
                     " + e.getMessage());
222         }
223     }
224
225     private boolean isDebugMode() {
226         return Boolean.getBoolean("debug") ||
227                System.getProperty("app.debug") != null ||
228                "development".equals(System.getProperty("app.
                    environment"));
229     }
230 }
```

Listing 6.1: BooksClient - Main Application Class

Responsabilità principali:

- Inizializzazione servizi di base e test connettività server
- Configurazione finestra principale (Stage) con icone e stili cross-platform
- Gestione completa del ciclo di vita dell'applicazione (init, start, stop)
- Integrazione con PopupManager per gestione centralizzata finestre modali
- Protezione contro istanze multiple tramite ApplicationProtection

- Gestione sicura della chiusura con cleanup delle risorse
- Sistema di fallback per modalità offline quando server non disponibile

6.2.2 MainWindow - Controller Centrale

Orchestrazione e integrazione di tutti i componenti principali dell'interfaccia utente:

```
/**
 * Controller centrale dell'interfaccia utente che coordina
 * sidebar, header, area contenuti e sistema autenticazione.
 */
public class MainWindow {

    private final BookService bookService;
    private final boolean serverAvailable;
    private List<Book> cachedBooks = new ArrayList<>();

    // Componenti principali UI
    private StackPane mainRoot;
    private Sidebar sidebar;
    private Header header;
    private ContentArea contentArea;
    private AuthenticationManager authManager;
    private ExploreIntegration exploreIntegration;

    public MainWindow(BookService bookService, boolean
        serverAvailable) {
        this.bookService = bookService;
        this.serverAvailable = serverAvailable;
        this.authManager = new AuthenticationManager();

        initializeAuthentication();
    }

    /**
     * Inizializza il sistema di autenticazione e configura i
         callback
     */
    private void initializeAuthentication() {
        authManager.setOnAuthStateChanged(() -> {
            if (sidebar != null) {
                sidebar.refreshAuthSection();
            }
        });
        authManager.initialize();
    }

    /**
     * Crea il layout principale dell'interfaccia utente
     */
    public StackPane createMainLayout() {
        System.out.println("Creazione layout principale...");

        mainRoot = new StackPane();
        BorderPane appRoot = new BorderPane();

        // Creazione componenti principali
        sidebar = new Sidebar(serverAvailable, authManager, this);
        header = new Header(bookService, mainRoot);
```

```
51          contentArea = new ContentArea(bookService, serverAvailable,
                authManager);
52
53          // Setup event handlers
54          setupEventHandlers();
55          initializeExploreIntegration();
56
57          // Assembly layout
58          appRoot.setLeft(sidebar.createSidebar());
59
60          VBox centerBox = new VBox();
61          centerBox.setStyle("-fx-background-color: #1e1e1e;");
62          centerBox.getChildren().addAll(
63              header.createHeader(),
64              contentArea.createContentArea()
65          );
66          appRoot.setCenter(centerBox);
67
68          mainRoot.getChildren().add(appRoot);
69
70          // Inizializzazione PopupManager
71          Platform.runLater(() -> {
72              PopupManager.getInstance().initialize(mainRoot);
73              Platform.runLater(() -> {
74                  try {
75                      Thread.sleep(1000);
76                      testSearchSystemAfterInit();
77                  } catch (InterruptedException e) {
78                      // Ignore
79                  }
80              });
81          });
82
83          contentArea.loadInitialContent();
84          return mainRoot;
85      }
86
87      /**
88       * Configura gli event handler per comunicazione tra componenti
89       */
90      private void setupEventHandlers() {
91          Consumer<Book> bookClickHandler = selectedBook -> {
92              BooksClient.openBookDetails(
93                  selectedBook,
94                  cachedBooks.isEmpty() ? List.of(selectedBook) :
                        cachedBooks,
95                  authManager
96              );
97          };
98
99          header.setSearchHandler((query) -> {
100             Consumer<Book> popupHandler = selectedBook -> {
101                 BooksClient.openBookDetails(
102                     selectedBook,
103                     cachedBooks.isEmpty() ? List.of(selectedBook) :
                            cachedBooks,
```

```
                    authManager
                );
            };

            try {
                contentArea.handleSearch(query, popupHandler);
            } catch (Exception e) {
                System.err.println("Errore durante ricerca: " + e.
                    getMessage());
            }
        });

        contentArea.setBookClickHandler(bookClickHandler);
        contentArea.setCachedBooksCallback(books -> {
            this.cachedBooks = books;
        });
    }

    // Metodi di navigazione
    public void showHomePage() {
        PopupManager.getInstance().closeAllPopups();
        if (contentArea != null) {
            contentArea.loadInitialContent();
        }
        if (header != null) {
            header.clearSearch();
        }
    }

    public void showExploreSection() {
        if (contentArea != null) {
            contentArea.handleMenuClick(2);
        }
    }

    public void showLibraryPanel() {
        if (!authManager.isAuthenticated()) {
            showAuthPanel();
            return;
        }

        // Creazione e configurazione LibraryPanel con PopupManager
        LibraryPanel libraryPanel = new LibraryPanel(
            authManager.getCurrentUsername(),
            authManager.getAuthService()
        );

        // Setup overlay e gestione eventi
        StackPane overlay = new StackPane();
        overlay.setStyle("-fx-background-color: rgba(0, 0, 0, 0.7);
            ");
        overlay.getChildren().add(libraryPanel);

        PopupManager.getInstance().showCustomPopup(
            "library_panel", "popup", overlay,
            () -> contentArea.loadInitialContent()
```

```
158             );
159         }
160
161         public void showAuthPanel() {
162             authManager.showAuthPanel(mainRoot);
163         }
164 }
```

Listing 6.2: MainWindow - Controller Centrale UI

Layout Struttura Generato:

```
StackPane (mainRoot)
└── BorderPane (appRoot)
    ├── Left: Sidebar (200px)
    │   ├── Header navigazione (Libreria)
    │   ├── Menu items dinamici
    │   │   ├── Home
    │   │   ├── I Miei Libri
    │   │   ├── Esplora
    │   │   └── Admin (condizionale)
    │   ├── Spacer flessibile
    │   ├── Server status indicator
    │   └── Sezione autenticazione
    └── Center: VBox
        ├── Header (60px) - Ricerca globale
        └── ContentArea - Contenuto principale
```

6.2.3 Sidebar - Navigazione Laterale

Gestisce la navigazione principale con menu dinamico basato sui privilegi utente:

```
public class Sidebar {

    private VBox menuItemsBox;
    private VBox authSection;
    private final boolean serverAvailable;
    private final AuthenticationManager authManager;
    private final MainWindow mainWindow;

    public VBox createSidebar() {
        VBox sidebar = new VBox(15);
        sidebar.setPrefWidth(200);
        sidebar.setPrefHeight(700);
        sidebar.setStyle("-fx-background-color: #2c2c2e;");

        // Header sezione
        Label sidebarHeader = new Label("Libreria");
        sidebarHeader.setFont(Font.font("System",
                FontWeight.BOLD, 16));

        sidebarHeader.setTextFill(Color.WHITE);
        sidebarHeader.setPadding(new Insets(20, 0, 5, 20));

        // Menu items dinamici
        menuItemsBox = createMenuItems();

        // Spacer per spingere elementi in basso
        Region spacer = new Region();
        VBox.setVgrow(spacer, Priority.ALWAYS);

        // Indicatore stato server
        Label serverStatus = new Label(
            serverAvailable ? "Server Online" : "Modalita' Offline"
        );
        serverStatus.setTextFill(serverAvailable ?
                    Color.LIGHTGREEN : Color.ORANGE);

        serverStatus.setFont(Font.font("System", 12));
        serverStatus.setPadding(new Insets(10, 20, 10, 20));

        // Sezione autenticazione dinamica
        authSection = new VBox(10);
        updateAuthSection();

        sidebar.getChildren().addAll(
            sidebarHeader, menuItemsBox,
            spacer, serverStatus, authSection
        );

        return sidebar;
    }

    private VBox createMenuItems() {
```

```
53          VBox menuBox = new VBox(5);
54          menuBox.setPadding(new Insets(0, 10, 0, 10));
55
56          // Menu base sempre visibili
57          menuBox.getChildren().addAll(
58              createMenuItem("Home", this::showHome),
59              createMenuItem("I Miei Libri", this::showLibraries),
60              createMenuItem("Esplora", this::showExplore)
61          );
62
63          // Menu amministrativo (condizionale)
64          if (authManager.isAuthenticated() &&
65              authManager.isUserAdmin()) {
66
67              Button adminItem =
68              createMenuItem("Admin", this::showAdmin);
69
70              adminItem.setStyle(adminItem.getStyle() +
71                                 "-fx-text-fill: #ff6b6b;");
72              menuBox.getChildren().add(adminItem);
73          }
74
75          return menuBox;
76      }
77
78      private Button createMenuItem(String text, Runnable action) {
79          Button menuItem = new Button(text);
80          menuItem.setPrefWidth(180);
81          menuItem.setAlignment(Pos.CENTER_LEFT);
82          menuItem.setStyle(
83              "-fx-background-color: transparent; " +
84              "-fx-text-fill: #e0e0e0; " +
85              "-fx-font-size: 14px; " +
86              "-fx-padding: 10 15; " +
87              "-fx-border-radius: 8; " +
88              "-fx-background-radius: 8;"
89          );
90
91          // Hover effects
92          menuItem.setOnMouseEntered(e ->
93              menuItem.setStyle(menuItem.getStyle() +
94                      "-fx-background-color: #3a3a3c;")
95          );
96          menuItem.setOnMouseExited(e ->
97              menuItem.setStyle(
98                  menuItem.getStyle().replace("-fx-background-color:
                        #3a3a3c;", ""))
99          );
100
101         menuItem.setOnAction(e -> action.run());
102
103         return menuItem;
104     }
105
106     public void updateAuthSection() {
107         authSection.getChildren().clear();
```

```
108
109         if (authManager.isAuthenticated()) {
110             // Widget utente autenticato
111             createUserWidget();
112         } else {
113             // Pulsanti login/registrazione
114             createAuthButtons();
115         }
116     }
117 }
```

Listing 6.3: Sidebar - Navigazione Dinamica

Spiegazione dei Metodi e Analisi della Complessità
I metodi del componente `Sidebar` sono responsabili della costruzione e dell'aggiornamento dell'interfaccia utente laterale. La loro complessità è interamente a tempo costante, in quanto il numero di elementi UI generati è fisso e non dipende da dati variabili come il numero di libri o di utenti.

- **`createSidebar()`**: Complessità **O(1)**. Questo metodo assembla la sidebar creando un numero fisso di nodi JavaFX (`VBox`, `Label`, `Region`). Anche le chiamate ai metodi createMenuItems() e updateAuthSection() non dipendono da un input variabile, mantenendo la complessità totale costante.
- **`createMenuItems()`**: Complessità **O(1)**. Il metodo genera un numero fisso di pulsanti per il menu principale e, in modo condizionale, un singolo pulsante aggiuntivo per l'amministrazione. Non itera su collezioni di dati.
- **`createMenuItem()`**: Complessità **O(1)**. Questo è un metodo di utilità che si limita a istanziare un singolo oggetto `Button` e a configurarne le proprietà visive e di evento. L'operazione è a tempo costante.
- **`updateAuthSection()`**: Complessità **O(1)**. Il metodo gestisce il "widget" di autenticazione, mostrando i pulsanti di login/registrazione o il riepilogo dell'utente, a seconda dello stato di autenticazione. L'operazione di pulizia (‘getChildren().clear()‘) e la successiva aggiunta di un numero fisso di elementi è a tempo costante.

6.2.4 Header - Sistema Ricerca Globale

Sistema di ricerca avanzato con integrazione popup e filtri:

```
public class Header {

    // Campi per la gestione dello stato e delle dipendenze
    private final BookService bookService;
    private final StackPane mainContainer;
    private Consumer<String> searchHandler;
    private TextField searchField;
    private Button advancedSearchButton;
    private boolean isAdvancedSearchOpen = false;

    /**
     * Crea l'header completo della finestra principale.
     */
    public HBox createHeader() {
        HBox header = new HBox(20);
        header.setPrefHeight(60);
        header.setAlignment(Pos.CENTER);
        header.setPadding(new Insets(15, 20, 15, 20));
        header.setStyle(
            "-fx-background-color: #2c2c2e; " +
            "-fx-border-color: transparent transparent #48484a
                transparent; " +
            "-fx-border-width: 0 0 1 0;"
        );

        // Logo/Titolo dell'app (sinistra)
        Label appTitle = new Label("BABO Library");
        appTitle.setFont(Font.font("System", FontWeight.BOLD, 20));
        appTitle.setTextFill(Color.web("#ffffff"));

        // Spacer centrale
        Region spacer1 = new Region();
        HBox.setHgrow(spacer1, Priority.ALWAYS);

        // Area di ricerca (centro)
        HBox searchArea = createSearchArea();

        // Spacer finale
        Region spacer2 = new Region();
        HBox.setHgrow(spacer2, Priority.ALWAYS);

        // Controlli destri (placeholder)
        HBox rightControls = new HBox(10);
        rightControls.setAlignment(Pos.CENTER_RIGHT);

        header.getChildren().addAll(appTitle, spacer1, searchArea,
            spacer2, rightControls);

        return header;
    }

    /**
```

```
51      * Crea l'area di ricerca completa con campo e pulsante
            avanzata.
52      */
53     private HBox createSearchArea() {
54         HBox searchArea = new HBox(8);
55         searchArea.setAlignment(Pos.CENTER);
56         searchArea.setMaxWidth(500);
57 
58         searchField = createSearchField();
59         advancedSearchButton = createAdvancedSearchButton();
60 
61         searchArea.getChildren().addAll(searchField,
               advancedSearchButton);
62         return searchArea;
63     }
64 
65     /**
66      * Crea il campo di ricerca principale con styling e event
            handling.
67      */
68     private TextField createSearchField() {
69         TextField field = new TextField();
70         field.setPromptText("Cerca libri per titolo o autore...");
71         field.setPrefWidth(350);
72         field.setStyle(
73             "-fx-background-color: #3a3a3c; " +
74             "-fx-text-fill: #ffffff; " +
75             "-fx-prompt-text-fill: #8e8e93; " +
76             "-fx-background-radius: 20px; " +
77             "-fx-border-color: #48484a; " +
78             "-fx-border-width: 1px; " +
79             "-fx-border-radius: 20px;" +
80             "-fx-padding: 8px 16px;"
81         );
82 
83         field.setOnAction(e -> performSearch());
84 
85         return field;
86     }
87 
88     /**
89      * Esegue la ricerca basata sul contenuto del campo di input.
90      */
91     private void performSearch() {
92         String query = searchField != null ? searchField.getText().
               trim() : "";
93         if (!query.isEmpty() && searchHandler != null) {
94             searchHandler.accept(query);
95         }
96     }
97 
98     /**
99      * Gestisce il toggle tra apertura e chiusura della ricerca
            avanzata.
100     */
101    private void toggleAdvancedSearch() {
```

```
102         if (isAdvancedSearchOpen) {
103             closeAdvancedSearch();
104         } else {
105             openAdvancedSearch();
106         }
107     }
108
109     /**
110      * Apre il popup di ricerca avanzata con overlay.
111      */
112     private void openAdvancedSearch() {
113         if (isAdvancedSearchOpen || bookService == null ||
                mainContainer == null) {
114             return;
115         }
116
117         isAdvancedSearchOpen = true;
118         updateAdvancedSearchButtonStyle(true);
119
120         AdvancedSearchPanel advancedPanel = new AdvancedSearchPanel
                (bookService);
121         advancedPanel.setOnSearchExecuted(result -> {
122             handleAdvancedSearchResult(result);
123             closeAdvancedSearch();
124         });
125
126         StackPane overlay = new StackPane();
127         overlay.setStyle("-fx-background-color: rgba(0,0,0,0.5);");
128         overlay.getChildren().add(advancedPanel);
129         StackPane.setAlignment(advancedPanel, Pos.CENTER);
130
131         mainContainer.getChildren().add(overlay);
132
133         // Gestione chiusura con ESC e click fuori
134         overlay.setOnMouseClicked(event -> {
135             if (event.getTarget() == overlay && event.getSource()
                    == overlay) {
136                 closeAdvancedSearch();
137             }
138         });
139         overlay.setOnKeyPressed(event -> {
140             if (event.getCode() == KeyCode.ESCAPE) {
141                 closeAdvancedSearch();
142             }
143         });
144
145         Platform.runLater(() -> overlay.requestFocus());
146     }
147
148     /**
149      * Chiude il popup di ricerca avanzata.
150      */
151     private void closeAdvancedSearch() {
152         if (!isAdvancedSearchOpen) return;
153
154         isAdvancedSearchOpen = false;
```

```
155         updateAdvancedSearchButtonStyle(false);
156
157         if (mainContainer != null) {
158             mainContainer.getChildren().removeIf(node -> {
159                 if (node instanceof StackPane) {
160                     StackPane stackPane = (StackPane) node;
161                     return stackPane.getChildren().stream().
                            anyMatch(child -> child instanceof
                            AdvancedSearchPanel);
162                 }
163                 return false;
164             });
165         }
166     }
167
168     /**
169      * Gestisce il risultato della ricerca avanzata.
170      */
171     private void handleAdvancedSearchResult(AdvancedSearchPanel.
            SearchResult result) {
172         if (searchHandler == null) return;
173
174         String searchQuery = buildSearchQuery(result);
175         if (searchHandler instanceof AdvancedSearchHandler) {
176             ((AdvancedSearchHandler) searchHandler).
                    handleAdvancedSearch(result);
177         } else {
178             searchHandler.accept(searchQuery);
179         }
180
181         // Aggiorna il campo di ricerca con la query costruita
182         Platform.runLater(() -> searchField.setText(searchQuery));
183     }
184
185     /**
186      * Costruisce una query string dai parametri della ricerca
            avanzata.
187      */
188     private String buildSearchQuery(AdvancedSearchPanel.
            SearchResult result) {
189         StringBuilder query = new StringBuilder();
190         // Logica di costruzione della query
191         if (result.getSearchType().contains("Titolo") && !result.
                getTitleQuery().isEmpty()) {
192             query.append("title-only:").append(result.getTitleQuery
                    ());
193         } else if (result.getSearchType().contains("Autore") && !
                result.getAuthorQuery().isEmpty()) {
194             query.append("author:").append(result.getAuthorQuery())
                    ;
195         }
196         // Aggiungi filtri anno, se presenti
197         if (!result.getYearFrom().isEmpty()) {
198             query.append(" year:").append(result.getYearFrom());
199             if (!result.getYearTo().isEmpty()) {
200                 query.append("-").append(result.getYearTo());
```

```
201                 }
202             }
203             return query.toString();
204         }
205 }
```

Listing 6.4: Header - Sistema Ricerca Avanzato

Spiegazione dei Metodi e Analisi della Complessità
I metodi principali del componente `Header` hanno una complessità computazionale costante, in quanto il loro tempo di esecuzione non dipende dalla dimensione dell'input (es. numero di libri nel database o lunghezza della stringa di ricerca).

- **`createHeader()`**: Complessità **O(1)**. Questo metodo crea un layout fisso e un numero predefinito di nodi JavaFX (`HBox`, `Label`, `TextField`, `Button`, `Region`). Il numero di operazioni è costante, indipendentemente dal contesto dell'applicazione.
- **`createSearchArea()`**: Complessità **O(1)**. Simile al metodo precedente, crea un layout fisso con un numero costante di componenti, pertanto il suo tempo di esecuzione è costante.
- **`createSearchField()`**: Complessità **O(1)**. Questo metodo si limita a istanziare un oggetto `TextField` e a configurarne le proprietà di stile e gli handler degli eventi. È un'operazione che richiede un tempo costante.
- **`performSearch()`**: Complessità **O(1)**. Il metodo esegue due operazioni principali: ottiene il testo dal campo di ricerca (operazione a tempo costante) e chiama un metodo `searchHandler.accept()` (che rappresenta una singola chiamata). La complessità effettiva della ricerca dipenderà dall'implementazione del metodo `accept()`, che tipicamente avverrà su un'altra parte del sistema (come un database o un servizio di ricerca) e potrebbe avere una complessità maggiore, ma l'operazione di `performSearch()` in sé è costante.
- **`openAdvancedSearch()`**: Complessità **O(1)**. Questo metodo crea un nuovo pannello di ricerca e un overlay con un numero fisso di nodi. L'operazione di aggiunta di questi nodi al `mainContainer` è anche a tempo costante.
- **`closeAdvancedSearch()`**: Complessità **O(n)**, dove **n** è il numero di nodi figli nel `mainContainer`. Questo metodo utilizza `removeIf` per rimuovere l'overlay di ricerca avanzata. Sebbene in questo specifico caso ci sia un solo overlay da rimuovere, l'implementazione del metodo `removeIf` itera su tutti gli elementi della lista dei figli per trovare quelli da rimuovere. Pertanto, nel caso peggiore, la sua complessità è proporzionale al numero di elementi nel contenitore.

- `handleAdvancedSearchResult()`: Complessità **O(L)**, dove **L** è la lunghezza totale della query risultante. Questo metodo chiama `buildSearchQuery()`, la cui complessità è lineare rispetto alla lunghezza della stringa della query. Inviare il risultato a `searchHandler` è un'operazione a tempo costante, ma come per `performSearch()`, la complessità effettiva della ricerca dipenderà dall'implementazione del gestore.
- `buildSearchQuery()`: Complessità **O(L)**, dove **L** è il numero di caratteri nella query finale. Il metodo costruisce una stringa utilizzando `StringBuilder`, che è un'operazione lineare. Il tempo di esecuzione dipende direttamente dalla lunghezza delle query di input (titolo, autore, anno).

In sintesi, i metodi di gestione dell'interfaccia utente (creazione, apertura, chiusura) hanno una complessità costante **O(1)**, ad eccezione del metodo di chiusura che è **O(n)** a causa dell'iterazione sulla lista dei nodi. I metodi di costruzione della query sono lineari **O(L)**.

6.2.5 ContentArea - Visualizzazione Contenuti

Area principale per la visualizzazione di libri, categorie e risultati di ricerca:

```
public class ContentArea {

    private final BookService bookService;
    private final boolean serverAvailable;
    private AuthenticationManager authManager;
    private VBox content;
    private BookSectionFactory sectionFactory;
    private ExploreIntegration exploreIntegration;
    private List<Book> featuredBooks = new ArrayList<>();
    private List<Book> freeBooks = new ArrayList<>();
    private List<Book> newBooks = new ArrayList<>();
    private List<Book> searchResults = new ArrayList<>();
    private List<Book> advancedSearchResults = new ArrayList<>();
    private CategoryView currentCategoryView = null;

    public ContentArea(BookService bookService, boolean
       serverAvailable, AuthenticationManager authManager) {
        if (bookService == null) {
            throw new IllegalArgumentException("BookService non puo
               ' essere null");
        }
        this.bookService = bookService;
        this.serverAvailable = serverAvailable;
        this.authManager = authManager;
        this.sectionFactory = new BookSectionFactory(bookService,
           serverAvailable);
        setupContextualNavigation();
    }

    private void setupContextualNavigation() {
        this.sectionFactory.setFeaturedBooksCallback(books -> {
            this.featuredBooks = new ArrayList<>(books);
        });
        this.sectionFactory.setFreeBooksCallback(books -> {
            this.freeBooks = new ArrayList<>(books);
        });
        this.sectionFactory.setNewBooksCallback(books -> {
            this.newBooks = new ArrayList<>(books);
        });
        this.sectionFactory.setSearchResultsCallback(books -> {
            this.searchResults = new ArrayList<>(books);
        });
        this.sectionFactory.setFreeBooksCallback(books -> {
            this.freeBooks = new ArrayList<>(books);
        });
    }

    public ScrollPane createContentArea() {
        content = new VBox(20);
        content.setId("content");
        content.setPadding(new Insets(15, 20, 30, 20));
        content.setStyle("-fx-background-color: #1e1e1e;");
```

```
50          ScrollPane scrollPane = new ScrollPane(content);
51          scrollPane.setHbarPolicy(ScrollPane.ScrollBarPolicy.NEVER);
52          scrollPane.setVbarPolicy(ScrollPane.ScrollBarPolicy.
                AS_NEEDED);
53          scrollPane.setFitToWidth(true);
54          scrollPane.setStyle("-fx-background-color: transparent;");
55          return scrollPane;
56      }
57
58      public void handleSearch(String query, Consumer<Book>
            clickHandler) {
59          if (query == null || query.trim().isEmpty()) {
60              loadInitialContent();
61              return;
62          }
63          searchResults.clear();
64          advancedSearchResults.clear();
65          Consumer<Book> popupManagerHandler = book ->
                handleBookClickWithPopupManager(book);
66          if (query.startsWith("title-only:")) {
67              String title = query.substring(11).trim();
68              handleTitleOnlySearch(title, popupManagerHandler);
69          } else if (query.startsWith("author:")) {
70              if (query.contains("year:")) {
71                  handleYearFilteredSearch(query, popupManagerHandler
                        );
72              } else {
73                  String author = query.substring(7).trim();
74                  handleAuthorSearch(author, popupManagerHandler);
75              }
76          } else if (query.contains("year:")) {
77              handleYearFilteredSearch(query, popupManagerHandler);
78          } else {
79              handleTitleSearch(query, popupManagerHandler);
80          }
81      }
82
83      private void handleTitleSearch(String query, Consumer<Book>
            clickHandler) {
84          sectionFactory.performSearch(query, content, clickHandler);
85      }
86
87      private void handleTitleOnlySearch(String title, Consumer<Book>
             clickHandler) {
88          content.getChildren().clear();
89          Label loadingLabel = new Label("Ricerca per titolo: " +
                title + "...");
90          loadingLabel.setFont(Font.font("System", FontWeight.NORMAL,
                 16));
91          loadingLabel.setTextFill(Color.WHITE);
92          content.getChildren().add(loadingLabel);
93          CompletableFuture.supplyAsync(() -> {
94                      try {
95                          return bookService.searchBooksByTitle(title
                                );
96                      } catch (Exception e) {
```

```
97                          throw new RuntimeException(e);
98                      }
99                  })
100                 .thenAccept(results -> {
101                     Platform.runLater(() -> {
102                         this.advancedSearchResults = new ArrayList
                                <>(results);
103                         displaySearchResults(results, "Titolo: " +
                                title, clickHandler);
104                     });
105                 })
106                 .exceptionally(throwable -> {
107                     Platform.runLater(() -> {
108                         handleTitleSearchFallback(title,
                                clickHandler);
109                     });
110                     return null;
111                 });
112     }
113
114     private void handleAuthorSearch(String author, Consumer<Book>
            clickHandler) {
115         content.getChildren().clear();
116         Label loadingLabel = new Label("Ricerca per autore: " +
                author + "...");
117         loadingLabel.setFont(Font.font("System", FontWeight.NORMAL,
                16));
118         loadingLabel.setTextFill(Color.WHITE);
119         content.getChildren().add(loadingLabel);
120         bookService.searchBooksAsync(author)
121                 .thenAccept(results -> {
122                     Platform.runLater(() -> {
123                         List<Book> authorResults =
                                filterBooksByAuthor(results, author);
124                         this.advancedSearchResults = new ArrayList
                                <>(authorResults);
125                         displaySearchResults(authorResults, "Autore
                                : " + author, clickHandler);
126                     });
127                 })
128                 .exceptionally(throwable -> {
129                     Platform.runLater(() -> {
130                         content.getChildren().clear();
131                         Label errorLabel = new Label("Errore nella
                                ricerca per autore: " + throwable.
                                getMessage());
132                         errorLabel.setTextFill(Color.web("#e74c3c")
                                );
133                         content.getChildren().add(errorLabel);
134                     });
135                     return null;
136                 });
137     }
138
139     private void handleYearFilteredSearch(String query, Consumer<
            Book> clickHandler) {
```

```
140         String author = "";
141         String yearRange = "";
142         if (query.contains("author:") && query.contains("year:")) {
143             String[] parts = query.split("\\s+");
144             for (String part : parts) {
145                 if (part.startsWith("author:")) {
146                     author = part.substring(7);
147                 } else if (part.startsWith("year:")) {
148                     yearRange = part.substring(5);
149                 }
150             }
151         }
152         final String finalAuthor = author;
153         final String finalYearRange = yearRange;
154         content.getChildren().clear();
155         Label loadingLabel = new Label("Ricerca avanzata: " +
                finalAuthor + " (" + finalYearRange + ")...");
156         loadingLabel.setFont(Font.font("System", FontWeight.NORMAL,
                16));
157         loadingLabel.setTextFill(Color.WHITE);
158         content.getChildren().add(loadingLabel);
159         bookService.searchBooksAsync(finalAuthor)
160                 .thenAccept(results -> {
161                     Platform.runLater(() -> {
162                         List<Book> authorResults =
                                filterBooksByAuthor(results, finalAuthor
                                );
163                         List<Book> filteredResults =
                                filterBooksByYearRange(authorResults,
                                finalYearRange);
164                         this.advancedSearchResults = new ArrayList
                                <>(filteredResults);
165                         displaySearchResults(filteredResults,
166                                 "Autore + Anno: " + finalAuthor + "
                                        (" + finalYearRange + ")",
                                    clickHandler);
167                     });
168                 })
169                 .exceptionally(throwable -> {
170                     Platform.runLater(() -> {
171                         content.getChildren().clear();
172                         Label errorLabel = new Label("Errore
                                ricerca avanzata: " + throwable.
                                getMessage());
173                         errorLabel.setTextFill(Color.web("#e74c3c")
                                );
174                         content.getChildren().add(errorLabel);
175                     });
176                     return null;
177                 });
178     }
179
180     private void displaySearchResults(List<Book> results, String
            searchTitle, Consumer<Book> clickHandler) {
181         content.getChildren().clear();
182         if (results.isEmpty()) {
```

```
183             Label noResults = new Label("Nessun risultato trovato
                    per: " + searchTitle);
184             noResults.setTextFill(Color.WHITE);
185             noResults.setFont(Font.font("System", FontWeight.NORMAL
                    , 18));
186             Label suggestion = new Label("Prova con parole chiave
                    diverse");
187             suggestion.setTextFill(Color.GRAY);
188             suggestion.setFont(Font.font("System", FontWeight.
                    NORMAL, 14));
189             VBox noResultsBox = new VBox(10, noResults, suggestion)
                    ;
190             noResultsBox.setAlignment(Pos.CENTER);
191             noResultsBox.setPadding(new Insets(50));
192             content.getChildren().add(noResultsBox);
193         } else {
194             Label title = new Label(searchTitle + " (" + results.
                    size() + " risultati)");
195             title.setFont(Font.font("System", FontWeight.BOLD, 20))
                    ;
196             title.setTextFill(Color.WHITE);
197             title.setPadding(new Insets(0, 0, 15, 0));
198             BookGridBuilder gridBuilder = new BookGridBuilder();
199             gridBuilder.setBookClickHandler(clickHandler);
200             VBox resultsContainer = new VBox(15);
201             resultsContainer.setPadding(new Insets(15, 20, 20, 20))
                    ;
202             resultsContainer.getChildren().add(title);
203             FlowPane bookGrid = gridBuilder.createOptimizedBookGrid
                    ();
204             gridBuilder.populateBookGrid(results, bookGrid, null);
205             ScrollPane scroll = new ScrollPane(bookGrid);
206             scroll.setHbarPolicy(ScrollPane.ScrollBarPolicy.NEVER);
207             scroll.setVbarPolicy(ScrollPane.ScrollBarPolicy.
                    AS_NEEDED);
208             scroll.setFitToWidth(true);
209             scroll.setStyle("-fx-background-color: transparent;");
210             resultsContainer.getChildren().add(scroll);
211             content.getChildren().add(resultsContainer);
212         }
213     }
214
215     private List<Book> determineNavigationContext(Book book) {
216         if (book == null) {
217             throw new IllegalArgumentException("Book non puo'
                    essere null");
218         }
219         if (featuredBooks.contains(book)) {
220             return featuredBooks;
221         } else if (freeBooks.contains(book)) {
222             return freeBooks;
223         } else if (newBooks.contains(book)) {
224             return newBooks;
225         } else if (searchResults.contains(book)) {
226             return searchResults;
227         } else if (advancedSearchResults.contains(book)) {
```

```
228             return advancedSearchResults;
229         } else {
230             List<Book> singleBookList = new ArrayList<>();
231             singleBookList.add(book);
232             return singleBookList;
233         }
234     }
235 }
```

Listing 6.5: ContentArea - Sistema Visualizzazione Dinamica

Spiegazione dei Metodi e Analisi della Complessità
I metodi principali del componente `ContentArea` gestiscono la visualizzazione dinamica dei contenuti, le operazioni di ricerca e la navigazione utente. La loro complessità dipende in gran parte dal numero di libri da visualizzare, ma i metodi di utilità hanno una complessità costante.

- **`createContentArea()`**: Complessità **O(1)**. Questo metodo si limita a istanziare un numero fisso di nodi JavaFX (`VBox`, `ScrollPane`) e a configurarne le proprietà di base. Il numero di operazioni è costante, indipendentemente dal contenuto che verrà caricato successivamente.

- **`loadInitialContent()`**: Complessità **O(N)**, dove **N** è il numero totale di libri caricati nelle sezioni iniziali. Il metodo pulisce il container del contenuto e richiama il caricamento asincrono di diverse sezioni (‘featured‘, ‘free‘, ‘new‘). La complessità è dominata dalle operazioni di I/O (richieste al server) e dalla creazione dei nodi JavaFX per ogni libro, che sono proporzionali a N.

- **`handleSearch()`**: Complessità **O(1)** per l'operazione di dispatching, ma la complessità effettiva dipende dal metodo di ricerca che viene chiamato e dal numero di risultati. L'operazione di per sé esamina la stringa di query e instrada la richiesta al metodo appropriato, un'operazione che richiede tempo costante.

- **`handleTitleOnlySearch()`**: Complessità **O(R)**, dove **R** è il numero di risultati restituiti dal servizio di ricerca. Dopo la chiamata asincrona al servizio, il metodo itera sui risultati per visualizzarli e per aggiornare la cache locale. Pertanto, la sua complessità è proporzionale al numero di libri trovati.

- **`handleAuthorSearch()`**: Complessità **O(R)**. Simile al metodo precedente, la sua complessità è lineare rispetto al numero di risultati restituiti dalla ricerca, che vengono poi filtrati e visualizzati.

- **`handleYearFilteredSearch()`**: Complessità **O(R)**, dove **R** è il numero di risultati restituiti dal servizio di ricerca. Questo metodo esegue due passaggi di filtraggio sui risultati (uno per l'autore e uno per l'anno), ma entrambi i passaggi sono lineari rispetto al numero di risultati parziali. La sua complessità complessiva rimane quindi proporzionale a R.

- `displaySearchResults()`: Complessità $\mathbf{O(R)}$, dove $\mathbf{R}$ è il numero di risultati da mostrare. Questo metodo è responsabile della creazione dell'interfaccia utente per i risultati di ricerca. Per ogni libro, crea una card e la aggiunge a un layout, rendendo l'operazione lineare in base al numero di risultati.
- `determineNavigationContext()`: Complessità $\mathbf{O(C)}$, dove $\mathbf{C}$ è il numero totale di libri memorizzati nelle liste della cache ('featuredBooks', 'freeBooks', 'newBooks', ecc.). Nel caso peggiore, il metodo deve scorrere tutte le liste per trovare la corrispondenza del libro, rendendo l'operazione lineare rispetto al totale dei libri in cache.

6.3 Sistema di Gestione Popup

6.3.1 PopupManager - Gestione dei Popup

Questa classe gestisce in modo centralizzato la visualizzazione e il controllo dei popup nell'applicazione, implementando il pattern **Singleton** per garantire un unico punto di accesso. Utilizza una struttura a stack per tracciare l'ordine di apertura dei popup.

```
1  public class PopupManager {
2
3      private static PopupManager instance;
4      private final Map<String, PopupInfo> activePopups = new
           ConcurrentHashMap<>();
5      private final Stack<String> popupStack = new Stack<>();
6      private StackPane mainContainer;
7      private boolean isInitialized = false;
8
9      private static class PopupInfo {
10         final StackPane popupNode;
11         final String id;
12         final String type;
13         final Runnable closeCallback;
14         final long createdAt;
15         boolean isVisible;
16
17         PopupInfo(String id, String type, StackPane popupNode,
               Runnable closeCallback) {
18             this.id = id;
19             this.type = type;
20             this.popupNode = popupNode;
21             this.closeCallback = closeCallback;
22             this.createdAt = System.currentTimeMillis();
23             this.isVisible = true;
24         }
25
26         @Override
27         public String toString() {
28             return String.format("PopupInfo{id='%s', type='%s',
                   visible=%s, age=%dms}",
29                     id, type, isVisible, System.currentTimeMillis()
                           - createdAt);
30         }
31     }
32
33     private PopupManager() {}
34
35     public static PopupManager getInstance() {
36         if (instance == null) {
37             instance = new PopupManager();
38         }
39         return instance;
40     }
41
```

```
42      public void initialize(StackPane mainContainer) {
43          if (mainContainer == null) {
44              System.err.println("PopupManager: Container principale
                    e' null!");
45              return;
46          }
47
48          this.mainContainer = mainContainer;
49          this.isInitialized = true;
50          System.out.println("PopupManager: Inizializzato con
                container principale");
51      }
52
53      public void showBookDetails(Book book, List<Book> collection,
            AuthenticationManager authManager) {
54          if (!isInitialized) {
55              System.err.println("PopupManager non inizializzato!");
56              return;
57          }
58
59          String popupId = "book-details-" + System.currentTimeMillis
                ();
60          StackPane popup = BookDetailsPopup.createWithLibrarySupport
                (
61                  book,
62                  collection,
63                  () -> closePopup(popupId),
64                  authManager
65          );
66
67          showPopup(popupId, "book-details", popup);
68      }
69
70      public void closePopup(String popupId) {
71          if (popupId == null || !activePopups.containsKey(popupId))
                {
72              System.out.println("PopupManager: Popup " + popupId + "
                        non trovato o gia' chiuso");
73              return;
74          }
75
76          Platform.runLater(() -> {
77              try {
78                  PopupInfo popupInfo = activePopups.get(popupId);
79                  // Rimuovi dal container e dalle strutture dati
80                  // ... (logica di rimozione) ...
81                  restoreFocusToTopPopup();
82                  System.out.println("PopupManager: Popup " + popupId
                            + " chiuso completamente");
83              } catch (Exception e) {
84                  System.err.println("Errore nella chiusura popup: "
                        + e.getMessage());
85                  emergencyCleanup(popupId);
86              }
87          });
88      }
```

```
89
90      public void closeTopPopup() {
91          if (popupStack.isEmpty()) {
92              System.out.println("PopupManager: Stack vuoto, nessun
                    popup da chiudere");
93              return;
94          }
95
96          String topPopupId = popupStack.peek();
97          closePopup(topPopupId);
98      }
99
100     public void closeAllPopups() {
101         List<String> popupsToClose = new ArrayList<>(activePopups.
                keySet());
102         for (String popupId : popupsToClose) {
103             closePopup(popupId);
104         }
105
106         Platform.runLater(() -> {
107             activePopups.clear();
108             popupStack.clear();
109         });
110     }
111
112     public void showCustomPopup(String popupId, String type,
            StackPane popupNode, Runnable closeCallback) {
113         if (!isInitialized || popupId == null || popupNode == null)
                {
114             System.err.println("PopupManager: parametri null o non
                    inizializzato");
115             return;
116         }
117
118         Platform.runLater(() -> {
119             if (activePopups.containsKey(popupId)) {
120                 closePopup(popupId);
121             }
122
123             PopupInfo popupInfo = new PopupInfo(popupId, type,
                    popupNode, closeCallback);
124             mainContainer.getChildren().add(popupNode);
125             activePopups.put(popupId, popupInfo);
126             popupStack.push(popupId);
127         });
128     }
129
130     public void runIntegrityCheck() {
131         System.out.println("PopupManager: Controllo integrita'");
132         try {
133             int activePopupsCount = activePopups.size();
134             int stackSize = popupStack.size();
135
136             if (activePopupsCount != stackSize) {
137                 System.out.println("ATTENZIONE: Incongruenza tra
                        popup attivi e stack");
```

```
138                 }
139                 // ... (altri controlli di coerenza) ...
140                 System.out.println("Controllo integrita' completato");
141             } catch (Exception e) {
142                 System.err.println("Errore controllo integrita': " + e.
                        getMessage());
143             }
144         }
145 }
```

Listing 6.6: PopupManager - Logica di Gestione

Spiegazione dei Metodi e Analisi della Complessità
I metodi principali del componente `PopupManager` gestiscono il ciclo di vita dei popup, dalla visualizzazione alla chiusura. La loro complessità computazionale varia a seconda che operino su un singolo popup o su tutti quelli attivi.

- **`getInstance()`**: Complessità **O(1)**. Questo metodo implementa il design pattern Singleton e si limita a un controllo condizionale e, se necessario, all'istanziazione di un oggetto. L'operazione richiede un tempo costante.
- **`initialize()`**: Complessità **O(1)**. Il metodo assegna un riferimento al contenitore principale e imposta un flag, operazioni che hanno una complessità costante.
- **`showBookDetails()`**: Complessità **O(1)**. Questo metodo delega la creazione e la visualizzazione del popup a metodi che eseguono un numero fisso di operazioni, come l'aggiunta di un singolo nodo a un contenitore e la manipolazione di una mappa e uno stack, che sono operazioni a tempo costante.
- **`closePopup()`**: Complessità **O(1)**. L'operazione di chiusura di un singolo popup comporta la rimozione di un elemento da una mappa, uno stack e un contenitore, che sono tutte operazioni a tempo costante.
- **`closeTopPopup()`**: Complessità **O(1)**. Questo metodo accede all'elemento in cima allo stack e delega la sua chiusura a `closePopup()`, mantenendo la complessità costante.
- **`closeAllPopups()`**: Complessità **O(N)**, dove **N** è il numero di popup attivi. Il metodo itera su tutti i popup e chiama `closePopup()` per ciascuno di essi. La complessità è quindi proporzionale al numero di popup da chiudere.
- **`showCustomPopup()`**: Complessità **O(1)**. Simile a `showBookDetails()`, questo metodo esegue operazioni a tempo costante: un controllo di esistenza, l'aggiunta di un nodo e la registrazione in una mappa e in uno stack.
- **`runIntegrityCheck()`**: Complessità **O(N)**, dove **N** è il numero di popup attivi. Il metodo esamina le strutture dati interne, iterando su tutti i popup registrati nella mappa, rendendo la sua complessità lineare rispetto al numero di elementi.

6.3.2 BookDetailsPopup - Dettagli Libro Avanzati

Popup complesso per visualizzazione dettagli libro con sistema di navigazione:

```
public class BookDetailsPopup {

    // --- Metodi Principali di Creazione e Gestione ---

    public static StackPane createWithLibrarySupport(Book book,
       List<Book> collection,
                                                      Runnable onClose,
                                                         AuthenticationManager
                                                          authManager) {
        initializePopup(book, collection, onClose, authManager);
        return createMainContainer(book);
    }

    private static StackPane createMainContainer(Book book) {
        root = new StackPane();
        StackPane blurLayer = createBackgroundLayer();
        bookDisplayPane = new StackPane();
        VBox currentBookContent = createBookContent(book, ...);

        loadBookRatingsForAllUsers(book, currentAuthManager);

        if (booksCollection.size() > 1) {
            setupMultiBookNavigation(currentBookContent);
        }

        root.getChildren().addAll(blurLayer, bookDisplayPane);
        setupImprovedFocusHandling(root);
        return root;
    }

    private static void handlePopupCloseWithPopupManager() {
        try {
            PopupManager popupManager = PopupManager.getInstance();
            if (popupManager.hasActivePopups()) {
                popupManager.closeTopPopup();
            } else {
                handlePopupCloseManual(); // Fallback
            }
        } catch (Exception e) {
            handlePopupCloseManual();
        }
    }

    // --- Costruzione delle Sezioni della UI ---

    private static VBox createBookContent(Book book, String
       backgroundColor,
                                          AuthenticationManager
                                             authManager) {
        VBox popupContent = new VBox();
        // ... impostazioni di stile ...
        HBox topBar = createTopBar();
```

```
48          ScrollPane contentScroll = createContentScrollPane(book,
                authManager);
49          popupContent.getChildren().addAll(topBar, contentScroll);
50          return popupContent;
51      }
52
53      private static ScrollPane createContentScrollPane(Book book,
            AuthenticationManager authManager) {
54          ScrollPane contentScroll = new ScrollPane();
55          // ... configurazione scrollpane ...
56          VBox scrollContent = new VBox();
57          scrollContent.getChildren().addAll(
58                  createDetailsSection(book, authManager),
59                  createPublisherSection(book),
60                  createRatingSection(book, authManager),
61                  createRecommendationsSection(book, authManager),
62                  createReviewsSection()
63          );
64          contentScroll.setContent(scrollContent);
65          return contentScroll;
66      }
67
68      private static HBox createDetailsSection(Book book,
            AuthenticationManager authManager) {
69          HBox detailsSection = new HBox(30);
70          detailsSection.setAlignment(Pos.TOP_LEFT);
71
72          VBox coverContainer = createCoverContainer(book);
73          VBox infoBox = createInfoBox(book, authManager);
74
75          detailsSection.getChildren().addAll(coverContainer, infoBox
                );
76          return detailsSection;
77      }
78
79      private static VBox createRatingSection(Book book,
            AuthenticationManager authManager) {
80          VBox ratingSection = new VBox(15);
81          // ... header ...
82          VBox ratingsContent = new VBox(12);
83          ratingsContent.getChildren().add(createAverageRatingDisplay
                ());
84
85          if (authManager != null && authManager.isAuthenticated()) {
86              ratingsContent.getChildren().add(
                    createUserRatingSection(book, authManager));
87          } else {
88              ratingsContent.getChildren().add(
                    createGuestInviteSection());
89          }
90          ratingSection.getChildren().addAll(ratingsHeader,
                ratingsContent);
91          return ratingSection;
92      }
93
94      private static VBox createRecommendationsSection(Book book,
```

```
        AuthenticationManager authManager) {
95          VBox recommendationsSection = new VBox(15);
96          HBox recommendationsHeader = createRecommendationsHeader(
                book, authManager);
97          ScrollPane recommendationsScrollPane =
                createRecommendationsScrollPane();
98          recommendationsSection.getChildren().addAll(
                recommendationsHeader, recommendationsScrollPane);
99          loadBookRecommendations(book, recommendationsScrollPane);
100         return recommendationsSection;
101     }
102
103     // --- Caricamento Dati Asincrono ---
104
105     private static void loadBookRatingsForAllUsers(Book book,
            AuthenticationManager authManager) {
106         if (isEmpty(book.getIsbn())) return;
107
108         loadAverageRating(book);
109
110         if (authManager != null && authManager.isAuthenticated()) {
111             loadUserRating(book, authManager.getCurrentUsername());
112         }
113     }
114
115     private static void loadAverageRating(Book book) {
116         ratingService.getBookRatingStatisticsAsync(book.getIsbn())
117                 .thenAccept(response -> Platform.runLater(() -> {
118                     if (response.isSuccess()) {
119                         averageBookRating = response.
                                getAverageRating();
120                         currentBookReviewCount = response.
                                getTotalRatings();
121                     }
122                     updateRatingDisplaySafe();
123                 }));
124     }
125
126     private static void loadBookRecommendations(Book targetBook,
            ScrollPane scrollPane) {
127         CompletableFuture.supplyAsync(() -> {
128             return recommendationService.
                    getBookRecommendationsAsync(targetBook.getIsbn()).
                    get();
129         }).thenAccept(response -> {
130             Platform.runLater(() -> {
131                 if (response.isSuccess() && response.
                        hasMultipleRecommendations()) {
132                     Map<String, List<BookRecommendation>>
                            groupedRecs = groupRecommendationsByBook
                            (...);
133                     HBox cardsContainer = new HBox(15);
134                     for (Map.Entry<String, List<BookRecommendation
                            >> entry : groupedRecs.entrySet()) {
135                         VBox card =
                                createRecommendationCardWithImage(...);
```

```
136                         cardsContainer.getChildren().add(card);
137                     }
138                     scrollPane.setContent(cardsContainer);
139                 }
140             });
141         });
142     }
143 
144     // --- Gestione Interazioni Utente (Dialogs e Azioni) ---
145 
146     private static void showAddToLibraryDialog(Book book,
        AuthenticationManager authManager) {
147         if (isEmpty(book.getIsbn())) return;
148 
149         LibraryService libraryService = new LibraryService();
150         libraryService.getUserLibrariesAsync(authManager.
            getCurrentUsername())
151                 .thenAccept(response -> Platform.runLater(() -> {
152                     if (response.isSuccess() && response.
                        getLibraries() != null) {
153                         List<String> libraries = response.
                            getLibraries();
154                         if (libraries.isEmpty()) {
155                             showCreateFirstLibraryDialog(book, ...)
                                ;
156                         } else {
157                             showChooseLibraryDialog(book, ...,
                                libraries, ...);
158                         }
159                     }
160                 }));
161     }
162 
163     private static void removeRecommendationAsync(
        BookRecommendation rec, Book targetBook, Button deleteButton
        ) {
164         if (currentAuthManager == null || !currentAuthManager.
            isAuthenticated()) {
165             return;
166         }
167 
168         RecommendationRequest removeRequest = new
            RecommendationRequest(...);
169 
170         CompletableFuture.supplyAsync(() -> {
171             return recommendationService.removeRecommendationAsync(
                removeRequest).get();
172         }).thenAccept(removeResponse -> {
173             Platform.runLater(() -> {
174                 if (removeResponse.isSuccess()) {
175                     loadBookRecommendations(targetBook,
                        currentRecommendationsScrollPane);
176                 } else {
177                     showAlert("Errore", removeResponse.getMessage()
                        );
178                 }
```

```
179             });
180         });
181     }
182
183     // --- Logica di Navigazione e Animazione ---
184
185     private static void setupMultiBookNavigation(VBox
            currentBookContent) {
186         nextBookPreview = createBookPreview(currentBookIndex + 1);
187         prevBookPreview = createBookPreview(currentBookIndex - 1);
188
189         bookDisplayPane.getChildren().addAll(prevBookPreview,
                currentBookContent, nextBookPreview);
190         addEdgeDetection(bookDisplayPane);
191         addNavigationArrows();
192     }
193
194     private static void slideToBook(int newIndex) {
195         if (newIndex < 0 || newIndex >= booksCollection.size() ||
                isTransitioning) {
196             return;
197         }
198
199         isTransitioning = true;
200         VBox newBookContent = createBookContent(...);
201         VBox currentContent = (VBox) bookDisplayPane.getChildren().
                get(1);
202
203         slideAnimation = new Timeline(
204                 new KeyFrame(Duration.millis(400),
205                         new KeyValue(currentContent.opacityProperty
                                (), 0.0, ...),
206                         new KeyValue(currentContent.
                                translateXProperty(), ...),
207                         new KeyValue(newBookContent.opacityProperty
                                (), 1.0, ...),
208                         new KeyValue(newBookContent.
                                translateXProperty(), 0, ...)
209                 )
210         );
211
212         slideAnimation.setOnFinished(e -> {
213             currentBookIndex = newIndex;
214             // ... reset della scena e dello stato ...
215             isTransitioning = false;
216         });
217
218         slideAnimation.play();
219     }
220 }
```

Listing 6.7: BookDetailsPopup - Gestione Dettagli Libro

Spiegazione dei Metodi e Analisi della Complessità
I metodi della classe `BookDetailsPopup` gestiscono la creazione dell'interfaccia, il caricamento asincrono dei dati e le interazioni complesse come la navigazione animata. La loro complessità varia in base al tipo di operazione: creazione di elementi UI, elaborazione di dati o avvio di processi in background.

- `createWithLibrarySupport()`: Complessità **O(N)**, dove **N** è il numero di libri nella collezione. La complessità è dominata dalla ricerca dell'indice del libro iniziale all'interno della lista (`collection.indexOf(book)`), un'operazione che nel caso peggiore richiede la scansione dell'intera collezione. Le successive operazioni di creazione della UI hanno un costo costante.
- `createBookContent()`: Complessità **O(1)**. Il metodo assembla una struttura fissa di componenti JavaFX (VBox, HBox, ScrollPane, etc.). Il numero di operazioni eseguite per creare questa struttura è costante e non dipende dalla dimensione dei dati del libro o della collezione.
- `loadBookRatingsForAllUsers()`: Complessità **O(1)**. Questo metodo è asincrono. Si limita a iniziare due chiamate di rete in background (`loadAverageRating` e `loadUserRating`) e termina immediatamente. Il suo costo computazionale è quindi costante, poiché il lavoro pesante viene eseguito su un altro thread.
- `loadBookRecommendations()`: Complessità **O(R)** nel callback, dove **R** è il numero di raccomandazioni ricevute. L'avvio della chiamata asincrona è **O(1)**. Tuttavia, il codice eseguito al completamento della chiamata (nel blocco `thenAccept`) deve iterare su tutte le **R** raccomandazioni per raggrupparle e creare le relative "card" visive.
- `showAddToLibraryDialog()`: Complessità **O(L)** nel callback, dove **L** è il numero di librerie dell'utente. L'avvio della chiamata di rete per ottenere le librerie è **O(1)**. Il callback, però, deve popolare un dialogo con la lista delle **L** librerie, un'operazione dal costo lineare.
- `slideToBook()`: Complessità **O(1)**. Questo metodo gestisce l'animazione di transizione tra i libri. La sua complessità è costante perché si limita a configurare una `Timeline` con un numero fisso di `KeyFrame` e proprietà da animare, indipendentemente dal numero totale di libri nella collezione.
- `extractDominantColor()`: Complessità **O(P)**, dove **P** è il numero di pixel dell'immagine. Il metodo analizza l'immagine campionando i pixel a intervalli regolari per determinare il colore più frequente. Il numero di operazioni è proporzionale alla dimensione dell'immagine (larghezza per altezza), risultando quindi lineare rispetto al numero di pixel.
- `groupRecommendationsByBook()`: Complessità **O(R)**, dove **R** è il numero di raccomandazioni. Questo metodo di utilità scorre l'intera lista di raccomandazioni una sola volta per raggrupparle in base all'ISBN del libro. La sua complessità è quindi lineare rispetto alla dimensione della lista in input.

6.4 Sistema di Comunicazione Client-Server

6.4.1 Architettura REST Client

Il client utilizza HttpClient nativo di Java per chiamate HTTP asincrone con gestione errori robusta:

```
public class BookService {

    private final OkHttpClient httpClient;
    private final ObjectMapper objectMapper;

    // --- Inizializzazione e Gestione Connessione ---

    public BookService() {
        this.httpClient = new OkHttpClient.Builder()
                .connectTimeout(10, java.util.concurrent.TimeUnit.
                    SECONDS)
                .readTimeout(30, java.util.concurrent.TimeUnit.
                    SECONDS)
                .build();
        this.objectMapper = new ObjectMapper();
    }

    public boolean isServerAvailable() {
        Request request = new Request.Builder()
                .url(SERVER_BASE_URL + "/books")
                .head()
                .build();
        try (Response response = httpClient.newCall(request).
            execute()) {
            return response.isSuccessful();
        } catch (Exception e) {
            return false;
        }
    }

    // --- Metodi Asincroni Principali ---

    public CompletableFuture<List<Book>> getAllBooksAsync() {
        return CompletableFuture.supplyAsync(() -> {
            try {
                return getAllBooks();
            } catch (Exception e) {
                // In caso di errore, usa i dati di fallback
                return getFallbackBooks();
            }
        });
    }

    public CompletableFuture<List<Book>> searchBooksAsync(String
        query) {
        return CompletableFuture.supplyAsync(() -> {
            try {
                return searchBooks(query);
```

```
45              } catch (Exception e) {
46                  return new ArrayList<>();
47              }
48          });
49      }
50
51      public CompletableFuture<List<Book>> getFeaturedBooksAsync() {
52          return CompletableFuture.supplyAsync(() -> {
53              try {
54                  return getFeaturedBooks();
55              } catch (Exception e) {
56                  return getFallbackBooks().subList(0, 1);
57              }
58          });
59      }
60
61      // --- Metodi Sincroni di Accesso ai Dati ---
62
63      public List<Book> getAllBooks() throws IOException {
64          Request request = new Request.Builder()
65                  .url(SERVER_BASE_URL + "/books")
66                  .get()
67                  .build();
68
69          try (Response response = httpClient.newCall(request).
                execute()) {
70              if (response.isSuccessful() && response.body() != null)
                    {
71                  String jsonResponse = response.body().string();
72                  return objectMapper.readValue(jsonResponse, new
                        TypeReference<List<Book>>() {});
73              } else {
74                  throw new IOException("Errore server: " + response.
                        code());
75              }
76          }
77      }
78
79      public Book getBookById(Long id) throws IOException {
80          Request request = new Request.Builder()
81                  .url(SERVER_BASE_URL + "/books/" + id)
82                  .get()
83                  .build();
84
85          try (Response response = httpClient.newCall(request).
                execute()) {
86              if (response.isSuccessful() && response.body() != null)
                    {
87                  return objectMapper.readValue(response.body().
                        string(), Book.class);
88              } else if (response.code() == 404) {
89                  return null;
90              } else {
91                  throw new IOException("Errore server: " + response.
                        code());
92              }
```

```
93          }
94      }
95
96      public List<Book> searchBooks(String query) throws IOException
            {
97          HttpUrl url = HttpUrl.parse(SERVER_BASE_URL + "/books/
                search")
98                  .newBuilder()
99                  .addQueryParameter("q", query)
100                 .build();
101
102         Request request = new Request.Builder().url(url).get().
                build();
103
104         try (Response response = httpClient.newCall(request).
                execute()) {
105             if (response.isSuccessful() && response.body() != null)
                    {
106                 return objectMapper.readValue(response.body().
                        string(), new TypeReference<List<Book>>() {});
107             } else {
108                 throw new IOException("Errore server: " + response.
                        code());
109             }
110         }
111     }
112
113     // --- Gestione Fallback e Ciclo di Vita ---
114
115     private List<Book> getFallbackBooks() {
116         System.out.println("Utilizzo libri di fallback (modalita
                offline)");
117         List<Book> books = new ArrayList<>();
118         books.add(new Book(1L, "Il Nome della Rosa", "Umberto Eco",
                "...", "placeholder.jpg"));
119         books.add(new Book(2L, "1984", "George Orwell", "...", "
                placeholder.jpg"));
120         // ... altri libri predefiniti ...
121         return books;
122     }
123
124     public void shutdown() {
125         if (httpClient != null) {
126             httpClient.dispatcher().executorService().shutdown();
127             httpClient.connectionPool().evictAll();
128         }
129     }
130 }
```

Listing 6.8: BookService - Comunicazioni API

Spiegazione dei Metodi e Analisi della Complessità
La classe `BookService` gestisce le comunicazioni con un'API REST per la gestione di libri. L'architettura del client è basata su chiamate asincrone e sincrone, con gestione degli errori e implementazione di logiche di fallback. La complessità dei metodi varia a seconda del tipo di operazione: inizializzazione del client, interrogazione dello stato del server, o recupero di dati tramite API.

- `BookService()`: Complessità $\mathbf{O(1)}$. Il costruttore inizializza una nuova istanza di `OkHttpClient` e `ObjectMapper`. Le operazioni di creazione e configurazione di questi oggetti hanno un costo computazionale costante che non dipende da alcun input o dimensione di dati.

- `isServerAvailable()`: Complessità $\mathbf{O(1)}$ in termini di operazioni locali. Il metodo esegue una singola chiamata HTTP di tipo `HEAD` per verificare la raggiungibilità del server. Il costo principale è il tempo di latenza della rete, non il calcolo locale, che è costante.

- `getAllBooksAsync()`: Complessità $\mathbf{O(N)}$ nel *thread di background*, dove $\mathbf{N}$ è il numero di libri restituiti dall'API. Questo metodo è asincrono. Avvia un'operazione su un thread separato (`CompletableFuture.supplyAsync()`) che a sua volta chiama il metodo sincrono `getAllBooks()`. L'avvio dell'operazione è $\mathbf{O(1)}$, ma il lavoro effettivo svolto nel thread è lineare rispetto al numero di libri.

- `searchBooksAsync(String query)`: Complessità $\mathbf{O(S)}$ nel *thread di background*, dove $\mathbf{S}$ è il numero di risultati della ricerca. Similmente al metodo precedente, l'avvio è $\mathbf{O(1)}$, ma il lavoro effettivo consiste nell'eseguire una ricerca API che restituisce una lista di libri. Il costo è lineare rispetto al numero di risultati.

- `getFeaturedBooksAsync()`: Complessità $\mathbf{O(F)}$ nel *thread di background*, dove $\mathbf{F}$ è il numero di libri in evidenza. L'avvio è $\mathbf{O(1)}$. L'operazione asincrona esegue una chiamata API e, in caso di errore, si limita a prendere un numero fisso di libri di fallback. Il costo è lineare rispetto al numero di libri recuperati dall'API.

- `getAllBooks()`: Complessità $\mathbf{O(N)}$, dove $\mathbf{N}$ è il numero di libri nella risposta. Questo metodo esegue una chiamata HTTP sincrona. La complessità è dominata dall'operazione di parsing del JSON, che deve leggere e deserializzare tutti gli oggetti `Book` nella lista. Il costo è quindi proporzionale al numero di libri ricevuti.

- `getBookById(Long id)`: Complessità $\mathbf{O(1)}$. Questo metodo effettua una chiamata sincrona per un singolo libro. L'operazione di deserializzazione del JSON riguarda un solo oggetto, rendendo il costo computazionale costante.

- `searchBooks(String query)`: Complessità $\mathbf{O(S)}$, dove $\mathbf{S}$ è il numero di risultati della ricerca. Simile a `getAllBooks()`, la complessità è dominata dal parsing della risposta JSON, che deve elaborare una lista di $\mathbf{S}$ libri.

- `getFallbackBooks()`: Complessità $\mathbf{O(C)}$, dove $\mathbf{C}$ è il numero di libri di fallback predefiniti. Il metodo crea e popola una nuova `ArrayList` con un numero fisso di oggetti `Book`. Il numero di operazioni è costante, ma può essere espresso come lineare rispetto al numero di elementi aggiunti alla lista.
- `shutdown()`: Complessità $\mathbf{O(1)}$. Il metodo chiude il pool di thread dell'HTTP client e svuota la sua cache di connessioni. Queste operazioni hanno un costo computazionale costante e non dipendono da alcuna dimensione di dati.

6.4.2 Pattern Service Layer

I servizi client implementano una strategia unificata per comunicazione asincrona:

```
public class AuthService {

    private final HttpClient httpClient;
    private final ObjectMapper objectMapper;
    private static final String BASE_URL = "http://localhost:8080/
        api/auth";

    // --- Inizializzazione e Gestione Connessione ---

    public AuthService() {
        this.httpClient = HttpClient.newBuilder()
                .connectTimeout(Duration.ofSeconds(10))
                .build();
        this.objectMapper = new ObjectMapper();
    }

    public CompletableFuture<AuthResponse> healthCheckAsync() {
        return CompletableFuture.supplyAsync(() -> {
            try {
                HttpRequest request = HttpRequest.newBuilder()
                        .uri(URI.create(BASE_URL + "/health"))
                        .GET().build();
                HttpResponse<String> response = httpClient.send(
                    request,
                        HttpResponse.BodyHandlers.ofString());

                if (response.statusCode() == 200) {
                    return objectMapper.readValue(response.body(),
                        AuthResponse.class);
                } else {
                    return new AuthResponse(false, "Servizio non
                        disponibile");
                }
            } catch (Exception e) {
                return new AuthResponse(false, "Servizio non
                    raggiungibile");
            }
        });
    }

    // --- Metodi Asincroni Principali ---

    public CompletableFuture<AuthResponse> loginAsync(String email,
        String password) {
        AuthRequest requestDto = new AuthRequest(email, password);
        return CompletableFuture.supplyAsync(() -> {
            try {
                String requestBody = objectMapper.
                    writeValueAsString(requestDto);
                HttpRequest request = HttpRequest.newBuilder()
                        .uri(URI.create(BASE_URL + "/login"))
                        .header("Content-Type", "application/json")
```

```
46                          .POST(HttpRequest.BodyPublishers.ofString(
                                requestBody))
47                          .build();
48                  HttpResponse<String> response = httpClient.send(
                        request, ...);
49                  // ... gestione della risposta in base allo status
                        code ...
50                  return objectMapper.readValue(response.body(),
                        AuthResponse.class);
51              } catch (Exception e) {
52                  return new AuthResponse(false, "Errore di
                        connessione");
53              }
54          });
55      }
56
57      public CompletableFuture<AuthResponse> registerAsync(String
            name, String surname, String cf,
58                                                              String
                                                                  email,
                                                                  String
                                                                  username
                                                                  ,
                                                                  String
                                                                  password
                                                                  ) {
59          RegisterRequest requestDto = new RegisterRequest(name,
                surname, cf, email, username, password);
60          return CompletableFuture.supplyAsync(() -> {
61              try {
62                  String requestBody = objectMapper.
                        writeValueAsString(requestDto);
63                  HttpRequest request = HttpRequest.newBuilder()
64                          .uri(URI.create(BASE_URL + "/register"))
65                          .POST(HttpRequest.BodyPublishers.ofString(
                                requestBody))
66                          .build();
67                  HttpResponse<String> response = httpClient.send(
                        request, ...);
68                  // ... gestione risposta, successo con status 201
                        Created ...
69                  return objectMapper.readValue(response.body(),
                        AuthResponse.class);
70              } catch (Exception e) {
71                  return new AuthResponse(false, "Errore di
                        registrazione");
72              }
73          });
74      }
75
76      public CompletableFuture<AuthResponse> getUserProfileAsync(
            String userId) {
77          return CompletableFuture.supplyAsync(() -> {
78              try {
79                  HttpRequest request = HttpRequest.newBuilder()
80                          .uri(URI.create(BASE_URL + "/profile/" +
```

```
                        userId))
                .GET().build();
            HttpResponse<String> response = httpClient.send(
                request, ...);
            // ... gestione risposta, incluso status 404 Not
                Found ...
            return objectMapper.readValue(response.body(),
                AuthResponse.class);
        } catch (Exception e) {
            return new AuthResponse(false, "Errore recupero
                profilo");
        }
    });
}

public CompletableFuture<AuthResponse> changePasswordAsync(
    String userId, String oldPassword, String newPassword) {
    return CompletableFuture.supplyAsync(() -> {
        try {
            String requestBody = objectMapper.
                writeValueAsString(Map.of(
                "oldPassword", oldPassword, "newPassword",
                    newPassword));
            HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(BASE_URL + "/change-
                    password/" + userId))
                .POST(HttpRequest.BodyPublishers.ofString(
                    requestBody))
                .build();
            HttpResponse<String> response = httpClient.send(
                request, ...);
            // ... gestione risposta e parsing del messaggio
                con utility ...
            return new AuthResponse(response.statusCode() ==
                200,
                    extractMessageFromJson(response.body()));
        } catch (Exception e) {
            return new AuthResponse(false, "Errore cambio
                password");
        }
    });
}

private String extractMessageFromJson(String json) {
    try {
        int messageIndex = json.indexOf("\"message\":");
        if (messageIndex != -1) {
            int startQuote = json.indexOf("\"", messageIndex +
                10);
            int endQuote = json.indexOf("\"", startQuote + 1);
            return json.substring(startQuote + 1, endQuote);
        }
        return null;
    } catch (Exception e) {
        return null;
    }
```

```
122     }
123 }
```

Listing 6.9: AuthService - Comunicazioni API

Spiegazione dei Metodi e Analisi della Complessità
La classe `AuthService` gestisce la comunicazione con le API di autenticazione e autorizzazione. Implementa un'architettura client basata sul pattern **Service Layer**, utilizzando Java's `HttpClient` nativo per effettuare chiamate asincrone. Ogni metodo pubblico restituisce un `CompletableFuture`, permettendo al chiamante di gestire il risultato in modo non bloccante.

- **`AuthService()`**: Complessità **O(1)**. Il costruttore inizializza le dipendenze essenziali, come `HttpClient` e `ObjectMapper`. Le operazioni di configurazione e istanziazione di questi oggetti sono computazionalmente costanti e non dipendono da input esterni.
- **`healthCheckAsync()`**: Complessità **O(1)** in termini di calcolo locale. Questo metodo avvia una chiamata HTTP asincrona per un semplice controllo di stato (/health). L'operazione locale di creazione della richiesta e l'avvio del thread asincrono sono costanti. La latenza principale è dovuta al tempo di risposta della rete.
- **`loginAsync(String email, String password)`**: Complessità **O(1)** in termini di calcolo locale. Similmente al metodo precedente, l'operazione è asincrona. L'avvio della chiamata di rete è costante, mentre il lavoro pesante (chiamata API e parsing della risposta) avviene su un thread separato. Il parsing di un singolo oggetto `AuthResponse` è un'operazione dal costo costante.
- **`registerAsync(String name, ..., String password)`**: Complessità **O(1)** in termini di calcolo locale. Questo metodo prepara una richiesta di registrazione e la invia in modo asincrono. Le operazioni di serializzazione del DTO e di avvio della richiesta sono costanti, e la risposta è un singolo oggetto `AuthResponse`.
- **`getUserProfileAsync(String userId)`**: Complessità **O(1)**. Il metodo esegue una chiamata `GET` asincrona per recuperare un profilo utente specifico. Poiché l'operazione riguarda un singolo utente e la deserializzazione di un singolo oggetto, il costo è costante.
- **`changePasswordAsync(String userId, ...)`**: Complessità **O(1)**. Questo metodo invia una richiesta per il cambio password. Le operazioni di preparazione del corpo della richiesta e l'avvio della chiamata HTTP sono costanti. Il parsing della risposta, anche se personalizzato, si limita a cercare un campo specifico in una stringa JSON di piccole dimensioni, mantenendo la complessità costante.
- **`extractMessageFromJson(String json)`**: Complessità **O(L)**, dove **L** è la lunghezza della stringa JSON in input. Questo metodo di utilità esegue una

ricerca di sottostringhe (`indexOf`) e un'estrazione (`substring`) all'interno della stringa JSON. Nel caso peggiore, la ricerca deve scorrere l'intera stringa, rendendo il costo lineare rispetto alla sua lunghezza.

6.5 Sistema di Autenticazione

6.5.1 AuthenticationManager - Gestione Sessioni

```
public class AuthenticationManager {

    /** Stato di autenticazione corrente dell'utente */
    private boolean isAuthenticated = false;

    /** Informazioni dell'utente attualmente autenticato */
    private User currentUser = null;

    /** Pannello UI per le operazioni di autenticazione */
    private AuthPanel authPanel;

    /** Servizio backend per operazioni di autenticazione */
    private AuthService authService;

    /** Callback per notificare cambiamenti di stato auth */
    private Runnable onAuthStateChanged;

    public AuthenticationManager() {
        this.authService = new AuthService();
    }

    /**
     * Imposta il callback per ricevere notifiche sui cambiamenti
     * di stato di autenticazione
     */
    public void setOnAuthStateChanged(Runnable callback) {
        this.onAuthStateChanged = callback;
    }

    /**
     * Verifica se l'utente e' attualmente autenticato
     */
    public boolean isAuthenticated() {
        return isAuthenticated;
    }

    /**
     * Ottiene l'oggetto User dell'utente attualmente autenticato
     */
    public User getCurrentUser() {
        return currentUser;
    }

    /**
     * Ottiene lo username dell'utente attualmente autenticato
     */
    public String getCurrentUsername() {
        return currentUser != null ? currentUser.getUsername() :
            null;
    }

```

```
51      /**
52       * Ottiene il nome di visualizzazione dell'utente corrente
53       */
54      public String getCurrentUserDisplayName() {
55          return currentUser != null ? currentUser.getDisplayName() :
                "Utente";
56      }
57
58      /**
59       * Mostra il pannello di autenticazione in modalita' overlay
60       */
61      public void showAuthPanel(StackPane mainRoot) {
62          if (mainRoot == null) {
63              throw new IllegalArgumentException(
64                  "Il container principale non puo' essere null");
65          }
66
67          authPanel = new AuthPanel();
68          authPanel.setOnSuccessfulAuth(this::
                handleSuccessfulAuthentication);
69          authPanel.setOnClosePanel(() -> closeAuthPanel(mainRoot));
70
71          StackPane overlay = new StackPane();
72          overlay.setStyle("-fx-background-color: rgba(0, 0, 0, 0.7);
                ");
73          overlay.getChildren().add(authPanel);
74          StackPane.setAlignment(authPanel, Pos.CENTER);
75
76          // Gestione click esterno per chiusura
77          overlay.setOnMouseClicked(e -> {
78              if (e.getTarget() == overlay) {
79                  closeAuthPanel(mainRoot);
80              }
81          });
82
83          mainRoot.getChildren().add(overlay);
84          System.out.println("Pannello autenticazione aperto");
85      }
86
87      /**
88       * Esegue il logout dell'utente corrente
89       */
90      public void logout() {
91          performLogout();
92      }
93
94      /**
95       * Aggiorna lo stato di autenticazione e notifica i listener
96       */
97      public void setAuthenticationState(boolean authenticated, User
            user) {
98          boolean wasAuthenticated = this.isAuthenticated;
99
100         this.isAuthenticated = authenticated;
101         this.currentUser = user;
102
```

```
103         if (authenticated && user != null) {
104             System.out.println("Utente autenticato: " + user.
                    getDisplayName());
105         } else {
106             System.out.println("Utente disconnesso");
107         }
108 
109         if (wasAuthenticated != authenticated) {
110             notifyAuthStateChanged();
111         }
112     }
113 
114     /**
115      * Verifica la disponibilita' del servizio di autenticazione
116      */
117     public void checkAuthServiceHealth() {
118         authService.healthCheckAsync()
119             .thenAccept(response -> {
120                 Platform.runLater(() -> {
121                     if (response.isSuccess()) {
122                         System.out.println("Servizio autenticazione
                                : " +
123                                            response.getMessage());
124                     } else {
125                         System.out.println("Servizio autenticazione
                                 non disponibile: " +
126                                            response.getMessage());
127                     }
128                 });
129             })
130             .exceptionally(throwable -> {
131                 Platform.runLater(() -> {
132                     System.out.println("Errore connessione servizio
                             auth: " +
133                                        throwable.getMessage());
134                 });
135                 return null;
136             });
137     }
138 
139     /**
140      * Aggiorna le informazioni dell'utente attualmente autenticato
141      */
142     public void updateCurrentUser(User updatedUser) {
143         if (this.isAuthenticated && updatedUser != null) {
144             this.currentUser = updatedUser;
145             System.out.println("Profilo utente aggiornato: " +
146                                updatedUser.getDisplayName());
147             notifyAuthStateChanged();
148         }
149     }
150 
151     /**
152      * Inizializza il manager di autenticazione
153      */
154     public void initialize() {
```

```
155         System.out.println("Inizializzazione AuthenticationManager
                ...");
156         checkAuthServiceHealth();
157         System.out.println("AuthenticationManager inizializzato");
158     }
159
160     /**
161      * Cleanup risorse alla chiusura dell'applicazione
162      */
163     public void shutdown() {
164         System.out.println("Shutdown AuthenticationManager...");
165
166         if (isAuthenticated) {
167             authService.logoutAsync()
168                 .thenAccept(response -> {
169                     System.out.println("Logout silenzioso
                            completato");
170                 })
171                 .exceptionally(throwable -> {
172                     System.out.println("Logout silenzioso fallito:
                            " +
173                                        throwable.getMessage());
174                     return null;
175                 });
176         }
177
178         System.out.println("AuthenticationManager chiuso");
179     }
180
181     // Metodi privati per gestione interna
182     private void handleSuccessfulAuthentication(User user) {
183         setAuthenticationState(true, user);
184         Platform.runLater(() -> showWelcomeMessage(user));
185     }
186
187     private void performLogout() {
188         String userDisplayName = getCurrentUserDisplayName();
189         System.out.println("Esecuzione logout per: " +
                userDisplayName);
190
191         authService.logoutAsync()
192             .thenAccept(response -> Platform.runLater(() -> {
193                 if (response.isSuccess()) {
194                     System.out.println("Logout completato sul
                            server");
195                 } else {
196                     System.out.println("Logout locale (server non
                            risponde)");
197                 }
198                 setAuthenticationState(false, null);
199             }))
200             .exceptionally(throwable -> {
201                 Platform.runLater(() -> {
202                     System.out.println("Logout locale (errore
                            server): " +
203                                        throwable.getMessage());
```

```
204                     setAuthenticationState(false, null);
205                 });
206                 return null;
207             });
208     }
209
210     private void notifyAuthStateChanged() {
211         if (onAuthStateChanged != null) {
212             Platform.runLater(() -> {
213                 try {
214                     onAuthStateChanged.run();
215                 } catch (Exception e) {
216                     System.err.println("Errore nel callback auth
                        state changed: " +
217                                        e.getMessage());
218                 }
219             });
220         }
221     }
222
223     private void closeAuthPanel(StackPane mainRoot) {
224         if (mainRoot.getChildren().size() > 1) {
225             mainRoot.getChildren().remove(mainRoot.getChildren().
                    size() - 1);
226         }
227         System.out.println("Pannello autenticazione chiuso");
228     }
229 }
```

Listing 6.10: AuthenticationManager - Gestione Stato Utente

Spiegazione dei Metodi e Analisi della Complessità
La classe `AuthenticationManager` funge da controller centrale per la gestione dello stato di autenticazione e delle interazioni UI relative. Utilizza una logica di sessione locale, gestendo lo stato dell'utente e notificando i componenti dell'interfaccia grafica in modo asincrono, garantendo che le operazioni sulla UI avvengano sul thread corretto. La complessità dei metodi è in gran parte costante, poiché si focalizzano sulla gestione dello stato e sull'orchestrazione di altre classi, piuttosto che su elaborazioni intensive di dati.

- **`AuthenticationManager()`**: Complessità $\mathbf{O(1)}$. Il costruttore si limita a inizializzare una nuova istanza di `AuthService`. Questa è un'operazione singola e dal costo costante che non dipende da alcun input esterno o dimensione di dati.
- **`setOnAuthStateChanged(Runnable callback)`**: Complessità $\mathbf{O(1)}$. Il metodo assegna un riferimento a una funzione di callback. Si tratta di un'operazione di assegnazione di una variabile, che ha un costo computazionale costante.
- **`isAuthenticated()`** e **`getCurrentUser()`**: Complessità $\mathbf{O(1)}$. Entrambi i metodi sono semplici getter che restituiscono il valore di una variabile di istanza. L'accesso a una variabile di classe ha un costo costante.

- **`showAuthPanel(StackPane mainRoot)`**: Complessità **O(1)**. Questo metodo ha un costo costante in quanto crea un pannello di autenticazione (`AuthPanel`) e lo aggiunge a un nodo genitore della UI (`StackPane`). Le operazioni di creazione e aggiunta di componenti UI hanno un costo fisso che non dipende dalla dimensione di collezioni o da cicli di dati.

- **`logout()`**: Complessità **O(1)** in termini di calcolo locale. Il metodo avvia una chiamata asincrona (`logoutAsync()`) e gestisce la risposta o l'errore. Le operazioni locali di avvio del task e di gestione dei callback (`thenAccept`, `exceptionally`, `Platform.runLater`) hanno un costo costante. La latenza principale è dovuta alla comunicazione di rete.

- **`initialize()`**: Complessità **O(1)**. Avvia una singola chiamata asincrona (`healthCheckAsync()`) per verificare lo stato di connettività all'avvio dell'applicazione. Come nel caso precedente, l'operazione locale ha un costo costante.

- **`shutdown()`**: Complessità **O(1)**. Questo metodo esegue una verifica dello stato di autenticazione (`if (isAuthenticated)`) e, se necessario, avvia una chiamata di logout asincrona. Entrambe le operazioni sono di tipo costante.

- **`handleSuccessfulAuthentication(User user)`**: Complessità **O(1)**. Il metodo gestisce l'aggiornamento dello stato di autenticazione e invia una notifica all'interfaccia utente. Tutte le operazioni coinvolte, come la chiamata a `setAuthenticationState()` e l'esecuzione di `Platform.runLater()`, sono a costo costante.

- **`setAuthenticationState(boolean authenticated, User user)`**: Complessità **O(1)**. Il metodo aggiorna lo stato locale e, se lo stato è cambiato, invia una notifica. Tutte le operazioni di confronto, assegnazione e chiamata a `notifyAuthStateChanged()` sono a costo costante.

- **`notifyAuthStateChanged()`**: Complessità **O(1)**. Questo metodo verifica se un callback è stato impostato e lo esegue sul thread della UI. La verifica e la chiamata della funzione hanno un costo costante, indipendentemente dal contenuto del callback.

6.5.2 AuthPanel - Interfaccia Login/Registrazione

Pannello UI per login e registrazione con validazione client-side:

```
public class AuthPanel extends VBox {

    private AuthService authService;
    private Consumer<User> onSuccessfulAuth;
    private Runnable onClosePanel;

    // --- Inizializzazione e Callback ---

    public AuthPanel() {
        this.authService = new AuthService();
        setupPanel();
    }

    public void setOnSuccessfulAuth(Consumer<User> callback) {
        this.onSuccessfulAuth = callback;
    }

    public void setOnClosePanel(Runnable callback) {
        this.onClosePanel = callback;
    }

    // --- Costruzione dell'Interfaccia (Setup) ---

    private void setupPanel() {
        setPadding(new Insets(20));
        setAlignment(Pos.TOP_CENTER);
        setStyle("-fx-background-color: #1e1e1e; -fx-background-
            radius: 12px;");

        Label appTitle = new Label("Books");
        appTitle.setFont(Font.font("System", FontWeight.BOLD, 28));

        TabPane authTabs = new TabPane();
        Tab loginTab = new Tab("Accedi", createLoginPanel());
        Tab signupTab = new Tab("Registrati", createSignupPanel());
            // Logica omessa

        authTabs.getTabs().addAll(loginTab, signupTab);
        getChildren().addAll(appTitle, authTabs);
    }

    // --- Gestione Eventi e Logica di Autenticazione ---

    private VBox createLoginPanel() {
        TextField emailField = new TextField();
        emailField.setPromptText("Email");
        styleInput(emailField);

        PasswordField passwordField = new PasswordField();
        passwordField.setPromptText("Password");
        styleInput(passwordField);

```

```
51          Button loginBtn = new Button("ACCEDI");
52          styleActionButton(loginBtn);
53
54          ProgressIndicator progress = new ProgressIndicator();
55          progress.setVisible(false);
56
57          loginBtn.setOnAction(e -> {
58              String email = emailField.getText();
59              String password = passwordField.getText();
60              if (email.isEmpty() || password.isEmpty()) {
61                  showAlert("Errore", "Inserisci email e password");
62                  return;
63              }
64
65              progress.setVisible(true);
66              loginBtn.setDisable(true);
67
68              authService.loginAsync(email, password)
69                  .thenAccept(response -> Platform.runLater(() -> {
70                      progress.setVisible(false);
71                      loginBtn.setDisable(false);
72                      if (response.isSuccess()) {
73                          if (onSuccessfulAuth != null)
                                onSuccessfulAuth.accept(response.getUser
                                ());
74                          if (onClosePanel != null) onClosePanel.run
                                ();
75                      } else {
76                          showAlert("Errore", response.getMessage());
77                      }
78                  }));
79          });
80
81          return new VBox(15, emailField, passwordField, loginBtn,
                progress);
82      }
83
84
85      private void styleInput(TextField field) {
86          field.setStyle(
87              "-fx-background-color: #2b2b2b;" +
88              "-fx-text-fill: #ffffff;" +
89              "-fx-prompt-text-fill: #9e9e9e;" +
90              "-fx-background-radius: 8px;"
91          );
92          // Aggiunge listener per l'effetto focus
93          field.focusedProperty().addListener((obs, old, isFocused)
                -> {
94              if(isFocused) {
95                  // ... applica stile focus con bordo colorato ...
96              } else {
97                  // ... ripristina stile normale ...
98              }
99          });
100     }
101
```

```
102     private void styleActionButton(Button button) {
103         button.setStyle(
104             "-fx-background-color: #4a86e8;" +
105             "-fx-text-fill: white;" +
106             "-fx-font-weight: bold;" +
107             "-fx-cursor: hand;"
108         );
109         // Aggiunge listener per effetti hover e press
110         button.setOnMouseEntered(e -> button.setOpacity(0.9));
111         button.setOnMouseExited(e -> button.setOpacity(1.0));
112     }
113 
114     private void showAlert(String title, String message) {
115         Alert alert = new Alert(Alert.AlertType.INFORMATION);
116         alert.setTitle(title);
117         alert.setHeaderText(null);
118         alert.setContentText(message);
119         alert.showAndWait();
120     }
121 }
```

Listing 6.11: AuthPanel - Pannello UI di Autenticazione

Spiegazione dei Metodi e Analisi della Complessità La classe `AuthPanel` è un componente dell'interfaccia utente JavaFX che gestisce il pannello di login e registrazione. La sua logica è strettamente legata alla gestione degli eventi UI e all'interazione con il `AuthService` per le chiamate API. La maggior parte dei metodi ha una complessità costante in quanto si limita a creare e configurare componenti grafici, mentre la complessità delle operazioni asincrone è delegata al servizio.

- **`AuthPanel()`**: Complessità $\mathbf{O(1)}$. Il costruttore inizializza l'istanza di `AuthService` e chiama `setupPanel()`. Queste sono operazioni a costo costante che non dipendono da input o dati esterni.
- **`setOnSuccessfulAuth(Consumer<User> callback)`** e **`setOnClosePanel(Runnable callback)`**: Complessità $\mathbf{O(1)}$. Questi metodi si limitano ad assegnare un riferimento a una funzione di callback. Si tratta di operazioni di assegnazione di variabili che hanno un costo computazionale costante.
- **`setupPanel()`**: Complessità $\mathbf{O(1)}$. Il metodo assembla la struttura principale del pannello UI, creando elementi come `VBox`, `Label`, e `TabPane`, e aggiungendoli ai nodi genitore. Il numero di operazioni di creazione e aggiunta è fisso e non dipende da alcuna dimensione di dati.
- **`createLoginPanel()`**:
  Complessità $\mathbf{O(1)}$. Questo metodo si concentra sulla costruzione del pannello di login. Crea un numero fisso di componenti UI (campi di testo, pulsanti, indicatori di progresso) e ne definisce il comportamento. Il costo è dominato dall'inizializzazione di questi elementi e dalla definizione dei loro handler di eventi, tutte operazioni a costo costante. L'operazione asincrona interna

(`authService.loginAsync(...)`) ha un costo **O(1)** locale, in quanto avvia un'operazione su un thread separato. Il lavoro pesante, come il parsing della risposta, viene gestito nel callback `thenAccept`, che a sua volta ha una complessità **O(1)** poiché gestisce un singolo oggetto.

- **`styleInput(TextField field)`** e **`styleActionButton(Button button)`**: Complessità **O(1)**. Questi metodi applicano stili CSS e listener per gli eventi UI (come focus, hover e press). Si tratta di operazioni di configurazione che agiscono su un singolo componente alla volta, con un numero fisso di istruzioni.
- **`showAlert(String title, String message)`**: Complessità **O(1)**. Il metodo crea e mostra una finestra di dialogo nativa. L'operazione di creazione di un oggetto `Alert` e la chiamata a `showAndWait()` hanno un costo costante che non dipende dalla dimensione del messaggio.

6.6 Gestione Librerie Personali

6.6.1 LibraryPanel - Gestione Collezioni Utente

Sistema completo per gestione librerie personali con operazioni CRUD:

```
public class LibraryPanel extends VBox {

    private final LibraryService libraryService;
    private final String username;
    private VBox librariesContainer;
    private ScrollPane scrollPane;

    // --- Setup Iniziale e UI Principale ---

    public LibraryPanel(String username, AuthService authManager) {
        this.libraryService = new LibraryService();
        this.username = username;
        setupImprovedUI();
        loadUserLibraries();
    }

    private void setupLayout() {
        // Aggiunge i componenti principali al pannello
        this.getChildren().add(createModernHeader());
        this.getChildren().add(createNewLibrarySection());
        this.getChildren().add(createElegantSeparator());

        // Imposta il contenitore scrollabile per le librerie
        setupLibrariesContainer();
    }

    private void setupLibrariesContainer() {
        librariesContainer = new VBox(20);
        // ... stili e allineamento ...

        scrollPane = new ScrollPane(librariesContainer);
        // ... stili e policy di scorrimento ...

        VBox.setVgrow(scrollPane, Priority.ALWAYS);
        this.getChildren().add(scrollPane);
    }

    // --- Caricamento Dati e Gestione Asincrona ---

    public void loadUserLibraries() {
        showCreateLibrarySection(); // Mostra il form di creazione
        librariesContainer.getChildren().clear();
        librariesContainer.getChildren().add(createLoadingIndicator
            ());

        libraryService.getUserLibrariesAsync(username)
                .thenAccept(response -> Platform.runLater(() -> {
                    librariesContainer.getChildren().clear(); //
                        Rimuove indicatore
```

```
48                      if (response.isSuccess() && response.
                            getLibraries() != null) {
49                          if (response.getLibraries().isEmpty()) {
50                              showEmptyState();
51                          } else {
52                              displayLibraries(response.getLibraries
                                    ());
53                          }
54                      } else {
55                          showErrorMessage("Errore nel caricamento
                                delle librerie...");
56                      }
57                  }));
58      }
59
60      public void viewLibraryBooks(String libraryName) {
61          hideCreateLibrarySection(); // Nasconde il form di
                creazione
62          librariesContainer.getChildren().clear();
63          librariesContainer.getChildren().add(createLoadingIndicator
                ());
64
65          libraryService.getBooksInLibraryAsync(username, libraryName
                )
66                  .thenAccept(response -> Platform.runLater(() -> {
67                      librariesContainer.getChildren().clear();
68                      if (response.isSuccess() && response.getBooks()
                            != null) {
69                          librariesContainer.getChildren().add(
                                createBooksViewHeader(libraryName));
70                          if (response.getBooks().isEmpty()) {
71                              // ... mostra stato di libreria vuota
                                    ...
72                          } else {
73                              createBooksGrid(response.getBooks());
74                          }
75                      } else {
76                          showErrorMessage("Errore nel caricamento
                                dei libri...");
77                      }
78                  }));
79      }
80
81      // --- Gestione Azioni Utente ---
82
83      private void createNewLibrary() {
84          String libraryName = newLibraryField.getText().trim();
85          if (libraryName.isEmpty()) return;
86
87          createLibraryButton.setDisable(true);
88          createLibraryButton.setText("Creazione...");
89
90          libraryService.createLibraryAsync(username, libraryName)
91                  .thenAccept(response -> Platform.runLater(() -> {
92                      if (response.isSuccess()) {
93                          showSuccessMessage("Libreria creata!");
```

```
94                          loadUserLibraries(); // Ricarica la lista
95                      } else {
96                          showErrorMessage("Errore: " + response.
                                getMessage());
97                      }
98                      createLibraryButton.setDisable(false);
99                      createLibraryButton.setText("Crea Libreria");
100                 }));
101     }
102
103     private void deleteLibrary(String libraryName) {
104         libraryService.deleteLibraryAsync(username, libraryName)
105                 .thenAccept(response -> Platform.runLater(() -> {
106                     if (response.isSuccess()) {
107                         showSuccessMessage("Libreria eliminata!");
108                         loadUserLibraries(); // Ricarica la lista
109                     } else {
110                         showErrorMessage("Errore: " + response.
                                getMessage());
111                     }
112                 }));
113     }
114
115     // --- Creazione Componenti UI Dinamici ---
116
117     private void displayLibraries(List<String> libraries) {
118         for (String library : libraries) {
119             HBox libraryCard = createLibraryCard(library);
120             librariesContainer.getChildren().add(libraryCard);
121         }
122     }
123
124     private HBox createLibraryCard(String libraryName) {
125         HBox card = new HBox(15);
126         // ... stili e configurazione della card ...
127
128         Label iconLabel = new Label("Book");
129         VBox infoBox = new VBox(4); // Contiene nome e sottotitolo
130         // ...
131
132         Region spacer = new Region();
133         HBox.setHgrow(spacer, Priority.ALWAYS);
134
135         HBox actionsBox = new HBox(8); // Contiene pulsanti
136         Button viewButton = createActionButton("Visualizza");
137         Button deleteButton = createActionButton("Elimina");
138         actionsBox.getChildren().addAll(viewButton, deleteButton);
139
140         card.getChildren().addAll(iconLabel, infoBox, spacer,
                actionsBox);
141         setupLibraryCardEvents(card, libraryName, viewButton,
                deleteButton);
142         return card;
143     }
144
145     private void createBooksGrid(List<Book> books) {
```

```
146         GridPane grid = new GridPane();
147         // ... configurazione colonne e gap ...
148
149         for (int i = 0; i < books.size(); i++) {
150             VBox bookCard = createModernBookCard(books.get(i));
151             grid.add(bookCard, i % 4, i / 4); // Aggiunge card alla
                    griglia
152         }
153
154         librariesContainer.getChildren().add(grid);
155     }
156 }
```

Listing 6.12: LibraryPanel - Gestione Librerie Utente

Spiegazione dei Metodi e Analisi della Complessità
La classe `LibraryPanel` è un componente UI che funge da gestore per le librerie personali di un utente. La sua logica è orientata alla creazione di un'interfaccia utente dinamica, al caricamento e alla visualizzazione di dati da un servizio API, e alla gestione di operazioni CRUD (Create, Read, Update, Delete). La complessità è distribuita tra l'orchestrazione delle chiamate asincrone e le operazioni di rendering dell'interfaccia, che dipendono dalla quantità di dati da visualizzare.

- **`LibraryPanel(String username, AuthService authManager)`**: Complessità $\mathbf{O(1)}$. Il costruttore inizializza i servizi e le variabili di stato. Le chiamate a `setupImprovedUI()` e `loadUserLibraries()` sono avvii di processi a costo costante. La complessità principale di `loadUserLibraries()` è delegata a un'operazione asincrona.

- **`setupLayout()`** e **`setupLibrariesContainer()`**: Complessità $\mathbf{O(1)}$. Questi metodi si occupano della creazione e della configurazione iniziale del layout del pannello. Si tratta di un numero fisso di operazioni di creazione di componenti UI e di impostazione delle proprietà, indipendentemente dal numero di librerie o libri.

- **`loadUserLibraries()`**: Complessità $\mathbf{O(N)}$ nel *callback*, dove $\mathbf{N}$ è il numero di librerie dell'utente. Il metodo avvia un'operazione asincrona (`libraryService.getUserLibrariesAsync()`). L'avvio è $\mathbf{O(1)}$. Il costo maggiore si trova nel blocco `thenAccept`, dove il codice itera su ogni libreria (`displayLibraries()`) per creare e visualizzare una "card" corrispondente.

- **`viewLibraryBooks(String libraryName)`**: Complessità $\mathbf{O(B)}$ nel *callback*, dove $\mathbf{B}$ è il numero di libri nella libreria. Simile al metodo precedente, l'avvio della chiamata asincrona è $\mathbf{O(1)}$. Il costo lineare è nel callback che, dopo aver ricevuto i dati, itera su ogni libro per creare una griglia di schede (`createBooksGrid()`).

- **`createNewLibrary()`** e **`deleteLibrary(String libraryName)`**: Complessità $\mathbf{O(1)}$ in termini di calcolo locale. Questi metodi avviano chiamate API

asincrone per creare o eliminare una libreria. L'avvio del task e la gestione dei callback sono costanti. La ricarica successiva della lista di librerie (`loadUserLibraries()`) ha una complessità lineare, come descritto in precedenza.

- **`displayLibraries(List<String> libraries)`**: Complessità **O(N)**, dove **N** è il numero di librerie. Il metodo itera su ogni stringa nella lista fornita per creare una card visiva (`createLibraryCard()`). La complessità è lineare in quanto il ciclo di elaborazione è proporzionale al numero di librerie.
- **`createLibraryCard(String libraryName)`**: Complessità **O(1)**. Questo metodo crea una singola card per una libreria. L'operazione di creazione di elementi UI fissi (`HBox`, `Label`, `Button`) e la loro configurazione hanno un costo costante che non varia.
- **`createBooksGrid(List<Book> books)`**: Complessità **O(B)**, dove **B** è il numero di libri. Il metodo itera su ogni oggetto `Book` per creare una card visiva e aggiungerla a una griglia. La complessità è quindi direttamente proporzionale al numero di libri da visualizzare.

6.7 Sistema Valutazioni e Raccomandazioni

6.7.1 RatingDialog - Sistema Valutazione Multi-Dimensionale

Dialog per valutazione libri con sistema multi-criterio:

```
public class RatingDialog {

    // Componenti UI principali
    private Stage dialogStage;
    private Slider styleSlider, contentSlider, pleasantnessSlider,
        originalitySlider, editionSlider;
    private TextArea reviewTextArea;
    private Label averageLabel, qualityLabel;
    private Button saveButton, cancelButton, deleteButton;

    // --- Costruzione e Setup Iniziale ---

    public RatingDialog(Book book, String username, BookRating
        existingRating,
                        Consumer<BookRating> onRatingSaved) {
        this.book = book;
        this.username = username;
        this.currentRating = existingRating;
        this.onRatingSaved = onRatingSaved;
        this.ratingService = new ClientRatingService();

        initializeDialog();
        createUI();
        setupEventHandlers();

        if (existingRating != null) {
            populateFields(existingRating);
        }
    }

    private VBox createDialogContent() {
        VBox content = new VBox(20);
        // ... stili del contenitore principale ...

        // Assembla le varie sezioni del dialog
        VBox header = createHeader();
        VBox ratingsSection = createRatingsSection();
        VBox reviewSection = createReviewSection();
        VBox previewSection = createPreviewSection();
        HBox buttons = createButtonSection();

        VBox innerContent = new VBox(15);
        innerContent.getChildren().addAll(header, ratingsSection,
                                          reviewSection,
                                             previewSection,
                                             buttons);

        // Aggiunge un pannello scrollabile per contenuti lunghi
        ScrollPane scrollPane = new ScrollPane(innerContent);
```

```
46          // ... stili dello scrollpane ...
47
48          content.getChildren().add(scrollPane);
49          return content;
50      }
51
52      // --- Logica di Business e Gestione Eventi ---
53
54      private void setupEventHandlers() {
55          // Aggiunge listener agli slider per aggiornare l'anteprima
                in tempo reale
56          styleSlider.valueProperty().addListener((obs, oldVal,
               newVal) -> updatePreview());
57          contentSlider.valueProperty().addListener((obs, oldVal,
               newVal) -> updatePreview());
58          // ... altri slider ...
59
60          cancelButton.setOnAction(e -> closeDialog());
61          saveButton.setOnAction(e -> saveRating());
62
63          if (deleteButton != null) {
64              deleteButton.setOnAction(e -> deleteRating());
65          }
66      }
67
68      private void updatePreview() {
69          int style = (int) styleSlider.getValue();
70          int content = (int) contentSlider.getValue();
71          // ... recupera valori dagli altri slider ...
72
73          boolean hasRating = style > 0 || content > 0; // ... etc.
74          saveButton.setDisable(!hasRating);
75
76          if (hasRating) {
77              double average = calculateAverage(); // Calcola la
                    media
78              averageLabel.setText(String.format("_ %.1f/5", average)
                    );
79              qualityLabel.setText(getQualityDescription(average));
80              // ... aggiorna colori in base alla media ...
81          } else {
82              // ... resetta le label allo stato iniziale ...
83          }
84      }
85
86      private void saveRating() {
87          if (!isRatingValid()) {
88              showAlert("Errore", "Devi dare almeno una valutazione
                    per salvare.");
89              return;
90          }
91
92          saveButton.setDisable(true);
93          saveButton.setText("Salvando...");
94
95          RatingRequest request = new RatingRequest(
```

```
96              username, book.getIsbn(),
97              (int) styleSlider.getValue(), (int) contentSlider.
                    getValue(),
98              // ... altri valori ...
99              reviewTextArea.getText().trim()
100         );
101
102         ratingService.addOrUpdateRatingAsync(request)
103             .thenAccept(response -> Platform.runLater(() -> {
104                 if (response.isSuccess()) {
105                     if (onRatingSaved != null) onRatingSaved.accept
                            (response.getRating());
106                     closeDialog();
107                 } else {
108                     showAlert("Errore", "Errore nel salvare la
                            valutazione...");
109                     saveButton.setDisable(false);
110                 }
111             }));
112     }
113
114     private void deleteRating() {
115         Alert confirmAlert = new Alert(Alert.AlertType.CONFIRMATION
                , "...");
116         confirmAlert.showAndWait().ifPresent(result -> {
117             if (result == ButtonType.OK) {
118                 deleteButton.setDisable(true);
119                 ratingService.deleteRatingAsync(username, book.
                        getIsbn())
120                     .thenAccept(response -> Platform.runLater(() ->
                            {
121                         if (response.isSuccess()) {
122                             if (onRatingSaved != null)
                                    onRatingSaved.accept(null);
123                             closeDialog();
124                         } else {
125                             showAlert("Errore", "Impossibile
                                    eliminare...");
126                             deleteButton.setDisable(false);
127                         }
128                     }));
129             }
130         });
131     }
132
133     // --- Metodi Statici Factory ---
134
135     public static void showRatingDialog(Book book, String username,
136                                         Consumer<BookRating>
                                                onRatingSaved) {
137         RatingDialog dialog = new RatingDialog(book, username,
                onRatingSaved);
138         dialog.show();
139     }
140
141     public static void showEditRatingDialog(Book book, String
```

```
        username,
142                                                 BookRating
                                                        existingRating,
143                                                 Consumer<BookRating>
                                                        onRatingSaved) {
144         RatingDialog dialog = new RatingDialog(book, username,
                existingRating, onRatingSaved);
145         dialog.show();
146     }
147 }
```

Listing 6.13: RatingDialog - Gestione Valutazioni

Spiegazione dei Metodi e Analisi della Complessità
La classe `RatingDialog` gestisce l'interfaccia utente e la logica per la valutazione di un libro. Il suo scopo principale è permettere all'utente di esprimere un giudizio multi-dimensionale su un'opera e di inviare i dati a un servizio API in modo asincrono. I metodi di questa classe sono principalmente orientati all'orchestrazione della UI e alla gestione degli eventi, con una complessità costante per la maggior parte delle operazioni.

- **`RatingDialog(...)`**: Complessità **O(1)**. Il costruttore inizializza le variabili di stato e le dipendenze, e orchestra l'avvio della costruzione e della configurazione del dialogo. Le chiamate a `initializeDialog()`, `createUI()` e `setupEventHandlers()` sono operazioni a costo costante. La successiva chiamata a `populateFields()` ha un costo costante in quanto copia un numero fisso di valori.

- **`createDialogContent()`**: Complessità **O(1)**. Questo metodo assembla l'intera struttura del dialogo. Le operazioni di creazione di sezioni e componenti UI (come `VBox`, `Header`, `buttons`) sono fisse e non dipendono da alcuna dimensione di dati esterna.

- **`setupEventHandlers()`**: Complessità **O(1)**. Il metodo configura un numero fisso di listener per gli eventi UI (cambio di valore sugli slider, click sui bottoni). Si tratta di operazioni di configurazione singole e a costo costante.

- **`updatePreview()`**: Complessità **O(1)**. Questo metodo recupera i valori da un numero fisso di slider, calcola una media e aggiorna alcune etichette. Il numero di operazioni è costante, indipendentemente dal valore degli slider. Il costo di calcolo è minimale.

- **`saveRating()`**: Complessità **O(1)** in termini di calcolo locale. Il metodo avvia una chiamata asincrona (`ratingService.addOrUpdateRatingAsync()`). Le operazioni locali di validazione, creazione dell'oggetto richiesta (`RatingRequest`) e avvio del task sono costanti. Il lavoro pesante viene svolto dal servizio su un thread separato. Il callback (`thenAccept`) gestisce un singolo oggetto di risposta, mantenendo la complessità costante.

- `deleteRating()`: Complessità **O(1)** in termini di calcolo locale. Simile a `saveRating()`, questo metodo avvia un'operazione asincrona di eliminazione. La creazione della finestra di dialogo di conferma e la gestione del callback non dipendono da dati esterni, ma hanno un costo costante.
- `showRatingDialog(...)` e `showEditRatingDialog(...)`: Complessità **O(1)**. Questi sono metodi "factory" statici che creano e mostrano un'istanza del dialogo. Le operazioni di istanziazione e visualizzazione del dialogo hanno un costo costante.

6.8 Performance e Ottimizzazioni

6.8.1 Gestione Cache e Memory

Sistema di caching intelligente per ottimizzare performance:

```
public class ImageUtils {

    private static final ConcurrentHashMap<String, Image>
       imageCache = new ConcurrentHashMap<>();
    private static final ExecutorService imageExecutor = Executors.
       newFixedThreadPool(3);
    private static Image defaultPlaceholder = null;

    // --- Metodi Pubblici Principali ---

    public static ImageView createSafeImageView(String
       imageFileName, double width, double height) {
        ImageView imageView = new ImageView();
        imageView.setFitWidth(width);
        imageView.setFitHeight(height);
        // ... altre impostazioni di rendering ...

        // Imposta immediatamente un placeholder per una UI
           reattiva
        imageView.setImage(getDefaultPlaceholder());

        // Sanitizza l'input e avvia il caricamento in background
        String localFileName = convertToLocalFileName(imageFileName
           );
        loadLocalImageAsync(localFileName, imageView);

        return imageView;
    }

    public static Image loadSafeImage(String imageFileName) {
        String localFileName = convertToLocalFileName(imageFileName
           );

        // 1. Controlla la cache per un accesso immediato
        Image cachedImage = imageCache.get(localFileName);
        if (cachedImage != null) {
            return cachedImage;
        }

        // 2. Se non in cache, carica dalle risorse
        Image image = loadFromResourcesOnly(localFileName);

        // 3. Aggiunge l'immagine alla cache per le richieste
           future
        if (image != null && !image.isError()) {
            imageCache.put(localFileName, image);
        }

        return image;
```

```
43      }
44
45      // --- Logica di Caricamento Asincrono ---
46
47      private static void loadLocalImageAsync(String fileName,
            ImageView imageView) {
48          // Controlla la cache anche nel percorso asincrono
49          if (imageCache.containsKey(fileName)) {
50              Platform.runLater(() -> imageView.setImage(imageCache.
                    get(fileName)));
51              return;
52          }
53
54          // Esegue il caricamento I/O su un thread separato
55          CompletableFuture.supplyAsync(() -> {
56              return loadFromResourcesOnly(fileName);
57          }, imageExecutor).thenAccept(image -> {
58              if (image != null && !image.isError()) {
59                  imageCache.put(fileName, image);
60                  // Aggiorna l'ImageView sul thread della UI in modo
                        sicuro
61                  Platform.runLater(() -> imageView.setImage(image));
62              }
63          });
64      }
65
66      private static Image loadFromResourcesOnly(String fileName) {
67          try {
68              // Tenta il caricamento diretto
69              InputStream stream = ImageUtils.class.
                    getResourceAsStream("/books_covers/" + fileName);
70              if (stream != null) {
71                  Image image = new Image(stream);
72                  if (!image.isError()) return image;
73              }
74
75              // Fallback: prova varianti comuni del nome del file (
                    lowercase, senza spazi, etc.)
76              String[] variants = { fileName.toLowerCase(), fileName.
                    replace(" ", ""), ... };
77              for (String variant : variants) {
78                  InputStream variantStream = ImageUtils.class.
                        getResourceAsStream("/books_covers/" + variant);
79                  if (variantStream != null) {
80                      Image variantImage = new Image(variantStream);
81                      if (!variantImage.isError()) return
                            variantImage;
82                  }
83              }
84              return getDefaultPlaceholder();
85
86          } catch (Exception e) {
87              return getDefaultPlaceholder();
88          }
89      }
90
```

```
91      // --- Sicurezza e Sanitizzazione ---
92
93      private static String convertToLocalFileName(String input) {
94          if (input == null || input.trim().isEmpty()) {
95              return "placeholder.jpg";
96          }
97
98          // Se l'input e un URL, tenta di estrarre il nome del file
                dal path
99          if (input.startsWith("http")) {
100             if (input.contains("/")) {
101                 String fileName = input.substring(input.lastIndexOf
                        ("/") + 1);
102                 // ... logica aggiuntiva per estrarre ISBN se
                        presente ...
103                 return sanitizeFileName(fileName);
104             }
105             return "placeholder.jpg"; // URL non valido o non
                    parsabile
106         }
107
108         // Altrimenti, considera l'input un nome di file locale da
                sanitizzare
109         return sanitizeFileName(input);
110     }
111
112     private static String sanitizeFileName(String fileName) {
113         if (fileName == null) return "placeholder.jpg";
114
115         // Rimuove caratteri non sicuri per prevenire path
                traversal
116         String clean = fileName.replaceAll("[^a-zA-Z0-9.]", "");
117
118         // Assicura la presenza di un'estensione valida (es. .jpg)
119         if (!clean.matches(".*\\.(jpg|jpeg|png)$")) {
120             // ... logica per aggiungere estensione .jpg di default
                    ...
121             clean += ".jpg";
122         }
123
124         return clean.length() < 5 ? "placeholder.jpg" : clean;
125     }
126
127     // --- Gestione Cache e Risorse ---
128
129     public static Image getDefaultPlaceholder() {
130         if (defaultPlaceholder == null) {
131             initializeDefaultPlaceholder();
132         }
133         return defaultPlaceholder;
134     }
135
136     public static void clearImageCache() {
137         imageCache.clear();
138         System.out.println("Cache immagini pulita");
139     }
```

```
140 }
```

Listing 6.14: ImageUtils - Gestione Sicura Immagini

Spiegazione dei Metodi e Analisi della Complessità
La classe `ImageUtils` fornisce un'utility statica per il caricamento di immagini con un sistema di caching integrato e gestione robusta dei percorsi. L'obiettivo principale è ottimizzare le performance delle applicazioni grafiche riducendo il numero di letture I/O e garantendo che il caricamento delle immagini non blocchi il thread della UI. La complessità dei metodi varia a seconda che si tratti di un'operazione di accesso alla cache (costante) o di una di caricamento (potenzialmente lineare rispetto alla dimensione del file).

- **`createSafeImageView(String fileName, double width, double height)`**: Complessità $\mathbf{O(1)}$ localmente, più $\mathbf{O(P)}$ in background, dove $\mathbf{P}$ è il numero di pixel dell'immagine. Questo metodo si limita a creare e configurare un `ImageView`. Il lavoro pesante, ovvero il caricamento dell'immagine, è delegato a un'operazione asincrona (`loadLocalImageAsync()`). Il costo computazionale immediato è costante.

- **`loadSafeImage(String fileName)`**: Complessità $\mathbf{O(P)}$, dove $\mathbf{P}$ è il numero di pixel dell'immagine. Questo metodo esegue un caricamento sincrono. Se l'immagine è già in cache, l'accesso è $\mathbf{O(1)}$. Se l'immagine deve essere caricata, il costo è proporzionale al numero di pixel del file, a causa dell'operazione di lettura I/O e del parsing dell'immagine. L'aggiunta alla cache è a costo costante.

- **`loadLocalImageAsync(String fileName, ImageView imageView)`**: Complessità $\mathbf{O(P)}$ nel thread di background, dove $\mathbf{P}$ è il numero di pixel. Il metodo controlla prima la cache con un'operazione $\mathbf{O(1)}$. Se l'immagine non è presente, avvia un `CompletableFuture` su un thread separato. Il costo del caricamento effettivo e del parsing dell'immagine dal disco è lineare rispetto alla sua dimensione. Le successive operazioni di aggiornamento della cache e dell'UI sono a costo costante.

- **`loadFromResourcesOnly(String fileName)`**: Complessità $\mathbf{O(P)}$, dove $\mathbf{P}$ è il numero di pixel dell'immagine. Questo metodo gestisce la logica di fallback per il caricamento del file. Nel caso peggiore, tenta di caricare il file principale e poi itera su un numero fisso di varianti. La complessità è dominata dal caricamento I/O del file, che è lineare rispetto alla sua dimensione.

- **`convertToLocalFileName(String input)`**: Complessità $\mathbf{O(L)}$, dove $\mathbf{L}$ è la lunghezza della stringa di input. Sebbene le operazioni di base siano veloci, la ricerca di sottostringhe e la sanitizzazione con espressioni regolari (nel caso peggiore) dipendono dalla lunghezza dell'input. Tuttavia, l'operazione è molto rapida in pratica.

- **`sanitizeFileName(String fileName)`**: Complessità $\mathbf{O(L)}$, dove $\mathbf{L}$ è la lunghezza del nome del file. Le operazioni di sostituzione dei caratteri con

`replaceAll()` e la verifica dell'estensione dipendono dalla lunghezza della stringa, rendendo la complessità lineare.

- `getDefaultPlaceholder()`: Complessità $\mathbf{O(1)}$. Se il placeholder è già stato inizializzato, viene restituito immediatamente. Altrimenti, la sua inizializzazione (che è un'operazione costante) viene eseguita una sola volta.
- `clearImageCache()`: Complessità $\mathbf{O(1)}$. Il metodo svuota la `ConcurrentHashMap` utilizzando un'operazione di tipo costante.

6.9 Deployment e Configurazione

6.9.1 Packaging JavaFX

La configurazione Maven per il packaging dell'applicazione client è gestita tramite il file pom.xml. Di seguito è riportato un estratto con gli elementi chiave per la creazione di un JAR eseguibile e multipiattaforma.

```
<properties>
<maven.compiler.source>17</maven.compiler.source>
<maven.compiler.target>17</maven.compiler.target>
<javafx.version>21.0.2</javafx.version>
<app.main.class>org.BABO.client.ClientApplication</app.main.class>
</properties>

<dependencies>
    <dependency>
        <groupId>org.BABO</groupId>
        <artifactId>shared</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>

    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>${javafx.version}</version>
    </dependency>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-fxml</artifactId>
        <version>${javafx.version}</version>
    </dependency>

    <dependency>
        <groupId>com.squareup.okhttp3</groupId>
        <artifactId>okhttp</artifactId>
        <version>4.12.0</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.15.2</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>3.4.1</version>
            <executions>
                <execution>
                    <phase>package</phase>
```

```
47                      <goals>
48                          <goal>shade</goal>
49                      </goals>
50                      <configuration>
51                          <transformers>
52                              <transformer implementation="org.apache
                                    .maven.plugins.shade.resource.
                                    ManifestResourceTransformer">
53                                  <mainClass>org.BABO.client.
                                        ClientApplication</mainClass>
54                              </transformer>
55                          </transformers>
56                          <filters>
57                              <filter>
58                                  <artifact>*:*</artifact>
59                                  <excludes>
60                                      <exclude>META-INF/*.SF</exclude
                                            >
61                                      <exclude>META-INF/*.DSA</
                                            exclude>
62                                      <exclude>META-INF/*.RSA</
                                            exclude>
63                                  </excludes>
64                              </filter>
65                          </filters>
66                      </configuration>
67                  </execution>
68              </executions>
69          </plugin>
70      </plugins>
71  </build>
72
73  <profiles>
74      <profile>
75          <id>mac</id>
76          <activation>
77              <os><family>mac</family></os>
78          </activation>
79          <properties>
80              <javafx.runtime.args>--add-opens javafx.graphics/com.
                    sun.javafx.application=ALL-UNNAMED -Djava.awt.
                    headless=false</javafx.runtime.args>
81          </properties>
82      </profile>
83      <profile>
84          <id>windows</id>
85          <activation>
86              <os><family>windows</family></os>
87          </activation>
88          <properties>
89              <javafx.runtime.args>--add-modules javafx.controls,
                    javafx.fxml -Dprism.forceGPU=false</javafx.runtime.
                    args>
90          </properties>
91      </profile>
92  </profiles>
```

Listing 6.15: Estratto del pom.xml per il Packaging del Client JavaFX

Il file pom.xml è il cuore della configurazione di un progetto Maven e in questo caso definisce il modo in cui l'applicazione JavaFX viene costruita e impacchettata per il deployment. L'estratto fornito illustra come gestire le dipendenze e il packaging in un unico file, garantendo la creazione di un'applicazione eseguibile e la sua compatibilità su diverse piattaforme.

Sezione `properties`
Questa sezione definisce le proprietà globali che possono essere riutilizzate nel file.

- `<maven.compiler.source>` e `<maven.compiler.target>`: specificano la versione del compilatore Java (in questo caso **Java 17**), assicurando che il codice sia compatibile e compilato correttamente.
- `<javafx.version>`: definisce la versione della libreria JavaFX, garantendo la consistenza tra le dipendenze.
- `<app.main.class>`: indica la classe principale dell'applicazione, cruciale per creare un JAR eseguibile.

Sezione `dependencies`
Qui sono elencate tutte le librerie di cui il progetto ha bisogno per funzionare.

- `shared`: una dipendenza interna che probabilmente contiene le classi condivise tra il client e il server (come i DTO - Data Transfer Object).
- `javafx-controls` e `javafx-fxml`: le dipendenze essenziali per lo sviluppo di interfacce grafiche con JavaFX.
- `okhttp` e `jackson-databind`: dipendenze per la **comunicazione di rete** (OkHttp per le chiamate HTTP) e la **serializzazione/deserializzazione JSON** (Jackson), necessarie per interagire con le API REST del server.

Sezione `build`
Questa sezione configura il processo di costruzione del progetto.

- `maven-shade-plugin`: un plugin fondamentale per il packaging. Questo plugin crea un "uber-JAR" (JAR eseguibile autonomo) che include tutte le dipendenze dell'applicazione, rendendola facilmente distribuibile.
- **`<execution>`**: definisce come e quando il plugin deve essere eseguito. In questo caso, viene eseguito durante la fase di `package` con l'obiettivo `shade`.
- **`<transformers>`**: configura il plugin per aggiungere la classe principale nel file `META-INF/MANIFEST.MF` del JAR, rendendolo eseguibile con il comando `java -jar`.
- **`<filters>`**: esclude i file di firma (`.SF`, `.DSA`, `.RSA`) dalle dipendenze, prevenendo potenziali conflitti durante il processo di `shade`.

Sezione `profiles`
I profili consentono di personalizzare il processo di costruzione in base all'ambiente o al sistema operativo.

- `mac` e `windows`: due profili attivati automaticamente in base alla famiglia del sistema operativo.
- `<javafx.runtime.args>`: questa proprietà definisce gli **argomenti JVM** specifici per la piattaforma, necessari per il corretto funzionamento di JavaFX. Ad esempio, su macOS (`-add-opens`) gestisce i moduli aperti, mentre su Windows (`-add-modules` e `-Dprism.forceGPU=false`) aggiunge esplicitamente i moduli e disabilita l'accelerazione hardware per prevenire potenziali problemi di rendering.

6.10 Eseguibili del programma

6.10.1 Script di Avvio

Script per l'avvio del client JavaFX con configurazione ottimizzata per diversi sistemi operativi:

Avvio per Sistema Linux/MacOS

```
# 5. AVVIO CLIENT
echo " AVVIO CLIENT"
echo "Avvio Books Client..."

# Verifica distribuzione client
if [ ! -f "dist/BooksClient.jar" ]; then
    echo "Distribuzione client non trovata, generazione..."
    ./build-executables.sh
fi

# Verifica dimensioni JAR
JAR_SIZE=$(du -m "dist/BooksClient.jar" | cut -f1)
if [ "$JAR_SIZE" -lt 20 ]; then
    echo "JAR sembra troppo piccolo (${JAR_SIZE}MB), rigenerazione
       ..."
    ./build-executables.sh
fi

echo "Client distribuito: dist/BooksClient.jar (${JAR_SIZE}MB)"

# Configurazione specifica per sistema operativo
if [[ "$OSTYPE" == "darwin"* ]]; then
    # macOS
    echo "Sistema: macOS"
    java -Djava.awt.headless=false \
         -Djavafx.platform=desktop \
         -jar dist/BooksClient.jar &
    CLIENT_PID=$!
elif [[ "$OSTYPE" == "linux-gnu"* ]]; then
    # Linux
    echo "Sistema: Linux"
    java --add-modules javafx.controls,javafx.fxml \
         --add-opens javafx.graphics/com.sun.javafx.application=ALL
            -UNNAMED \
         --add-opens javafx.base/com.sun.javafx.reflect=ALL-UNNAMED
             \
         -jar dist/BooksClient.jar &
    CLIENT_PID=$!
else
    # Altri sistemi
    echo "Sistema: Altri"
    java -jar dist/BooksClient.jar &
    CLIENT_PID=$!
fi

```

```
echo "   Client PID: $CLIENT_PID"
echo "BOOKS CLIENT AVVIATO CON SUCCESSO!"

# Aspetta che il client termini
wait $CLIENT_PID 2>/dev/null || true
echo "Client terminato"
```

Listing 6.16: Avvio Client - Estratto da start-complete-app.sh

Avvio per Sistema Windows

```
rem 4. VERIFICA DISTRIBUZIONE CLIENT
echo.
echo 4. VERIFICA DISTRIBUZIONE CLIENT
echo ================================
if not exist "dist\BookReccommender.jar" (
    echo ERRORE: File del client non trovato!
    echo Assicurati che esista il file: dist\BookReccommender.jar
    echo Esegui 'mvn clean install' e copia il file JAR da 'client\target' a 'dist'.
    pause
    exit /b 1
)
echo Client distribuito e pronto.

rem 5. AVVIO CLIENT
echo.
echo 5. AVVIO CLIENT
echo ================
echo Avvio Book Recommender Client...
echo.

rem Configurazione JavaFX per Windows
set JAVAFX_HOME="C:\Program Files\Java\javafx-sdk-24.0.2"

rem Avvio con parametri ottimizzati per Windows
java --module-path %JAVAFX_HOME%\lib --add-modules javafx.controls,javafx.fxml ^
     --add-opens javafx.graphics/com.sun.javafx.application=ALL-UNNAMED ^
     --add-opens javafx.base/com.sun.javafx.reflect=ALL-UNNAMED ^
     -Djava.awt.headless=false ^
     -Dprism.forceGPU=false ^
     -jar dist\BookReccommender.jar

echo.
echo Client terminato
echo.
set /p stop_server="Vuoi fermare anche il server? (s/N): "
if /i "%stop_server%"=="s" (
    echo Arresto server...
    taskkill /f /im java.exe /fi "WINDOWTITLE eq Spring Boot Server*" >nul 2>&1
    echo Server fermato
)
```

```
41
42 echo.
43 echo Sessione completata!
44 pause
```

Listing 6.17: Avvio Client Windows - Estratto da start-complete-app.bat

Funzionamento dell'avvio client:
La sezione di avvio del client implementa una strategia multi-piattaforma per il lancio dell'applicazione JavaFX:

- **Unix/Linux/MacOS**: Lo script verifica automaticamente l'esistenza e la validità della distribuzione `BooksClient.jar`, rigenerandola tramite `build-executables.sh` se necessario o se le dimensioni del JAR risultano insufficienti (indicativo di build incomplete). La configurazione JVM viene adattata dinamicamente al sistema operativo rilevato: su macOS utilizza parametri ottimizzati per il desktop environment nativo, su Linux applica le aperture dei moduli JavaFX necessarie per aggirare le restrizioni del module system.
- **Windows**: Lo script batch verifica la presenza del file JAR nella directory `dist` e configura automaticamente il percorso JavaFX tramite la variabile `JAVAFX_HOME`. I parametri JVM includono ottimizzazioni specifiche per Windows come `-Dprism.forceGPU=false` per evitare problemi con driver grafici non ottimali. Al termine dell'esecuzione, offre la possibilità di arrestare automaticamente il server Spring Boot associato.

Il processo client viene eseguito consentendo al script di monitorarne lo stato e gestire la terminazione pulita con cleanup delle risorse di sistema.

6.11 Modello dei Casi d'Uso nel Contesto Client

6.11.1 Integrazione tra Client e Flussi Utente

L'architettura client JavaFX implementata supporta diversi profili utente, ognuno con specifici flussi di interazione e privilegi differenziati. Il sistema di autenticazione integrato con `AuthenticationManager` e `AuthService` permette la gestione di tre tipologie principali di utenti, ognuna delle quali accede a funzionalità specifiche attraverso l'interfaccia grafica.

Il **pattern di navigazione condizionale** implementato nella `Sidebar` e orchestrato dalla `MainWindow` garantisce che ogni utente visualizzi esclusivamente le opzioni e le funzionalità compatibili con il proprio ruolo. Questo approccio non solo migliora la sicurezza client-side, ma ottimizza anche l'esperienza utente riducendo la complessità dell'interfaccia per ogni profilo specifico.

6.11.2 Flussi Operativi Supportati dal Client

L'applicazione client gestisce quattro flussi operativi principali, ognuno dei quali è supportato da componenti specifici dell'architettura.

Il modello di funzionamento segue una logica gerarchica: l'utente registrato eredita e amplia le funzionalità disponibili per l'utente non registrato, mentre l'amministratore eredita a sua volta tutte le funzionalità dei livelli precedenti, aggiungendo strumenti di gestione e controllo avanzati:

- **Flusso Utente Non registrato** (Celeste): Implementato attraverso una versione limitata dell'interfaccia che mantiene l'accesso al catalogo ma restringe le funzionalità che richiedono autenticazione. Il sistema di fallback garantisce un'esperienza fluida anche senza connessione al server.
- **Flusso Utente Registrato** (Blu): Supportato dall'integrazione tra `LibraryPanel`, `BookDetailsPopup`, e `RatingDialog`. Questo flusso sfrutta appieno il sistema di comunicazione asincrona per operazioni come la creazione di librerie personali e la gestione delle valutazioni.
- **Flusso Admin** (Arancione): Gestito principalmente dal componente `AdminPanel` con accesso completo alle funzionalità di gestione del catalogo. Il client implementa controlli di autorizzazione per garantire che solo utenti con privilegi amministrativi possano accedere a queste funzionalità critiche.
- **Operazioni di Sistema** (Verde): Gestite dai servizi di background (`BookService`, `AuthService`, `LibraryService`) che operano in modalità asincrona per garantire la reattività dell'interfaccia utente durante le operazioni di sincronizzazione e recupero dati.

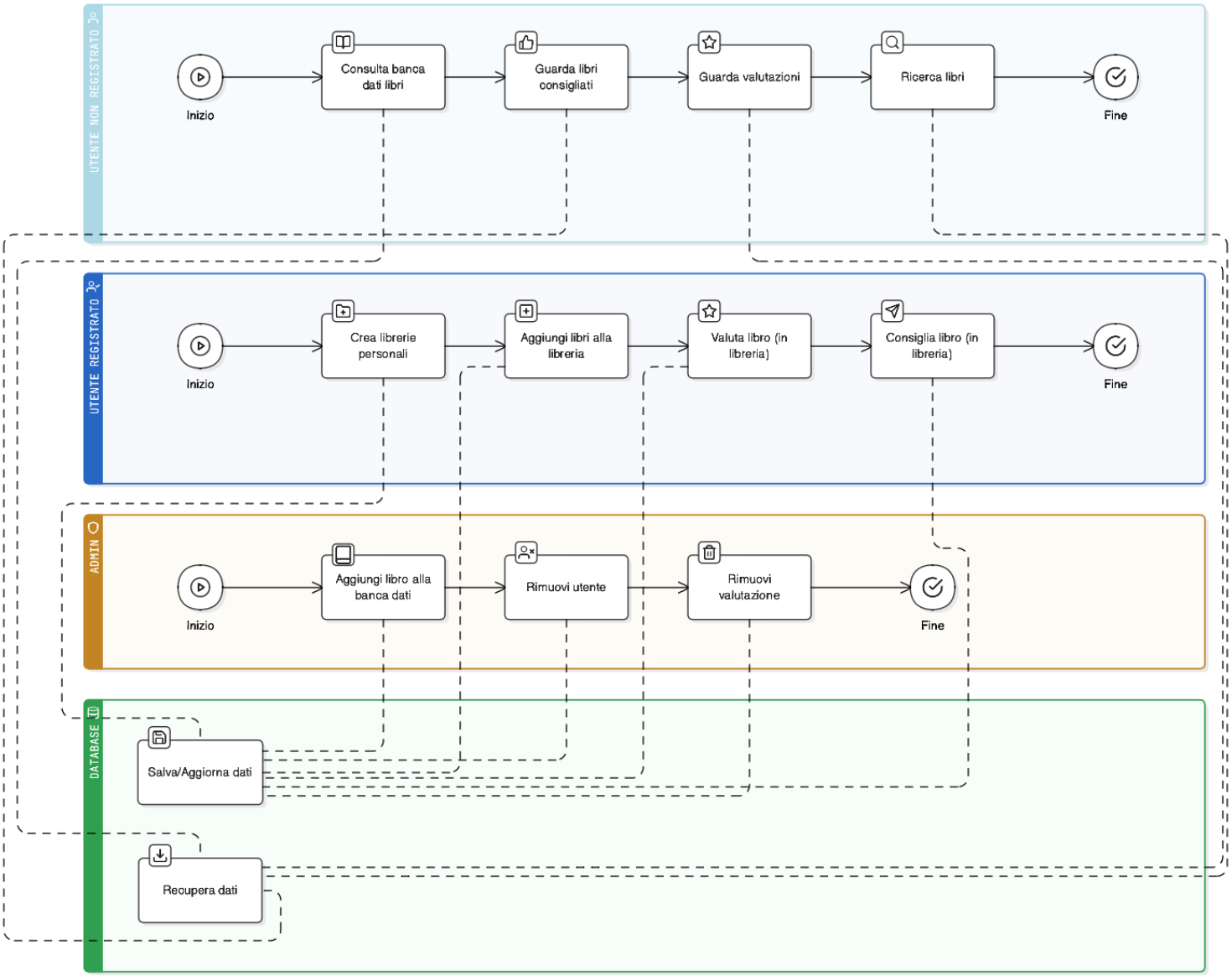


Figura 6.1: Use case dei vari utenti

6.11.3 Architettura Orientata ai Casi d'Uso

Il design dell'applicazione client riflette una chiara separazione tra i diversi domini funzionali, con ogni componente UI mappato su specifici casi d'uso business:

- **Gestione del Catalogo**: `ContentArea`, `BookGridBuilder`, e `CategoryView`
- **Autenticazione e Profilazione**: `AuthPanel`, `AuthenticationManager`, e `UserProfilePopup`
- **Gestione Librerie Personali**: `LibraryPanel` e `LibraryService`
- **Sistema di Valutazioni**: `RatingDialog` e `ClientRatingService`
- **Raccomandazioni Intelligenti**: `RecommendationDialog` e `ClientRecommendationService`

Questa architettura modulare consente una manutenzione semplificata e l'estensibilità futura, mantenendo al contempo la coerenza dell'esperienza utente attraverso tutti i flussi operativi.
Il diagramma seguente illustra graficamente come questi flussi si integrano nell'ecosistema applicativo complessivo, evidenziando i punti di intersezione tra client e server e i diversi livelli di accesso per ogni tipologia di utente.

6.11.4 Analisi dei Pattern di Interazione

L'analisi del diagramma rivela quattro pattern di interazione distinti che caratterizzano l'utilizzo dell'applicazione BABO:

Pattern Amministratore - Gestione Centralizzata

Il flusso amministratore implementa un pattern di **gestione centralizzata** dove tutte le operazioni CRUD sul catalogo sono concentrate in un singolo punto di controllo. Questo approccio garantisce:

- Controllo qualità sui dati inseriti nel sistema
- Tracciabilità completa delle modifiche al catalogo
- Prevenzione di duplicazioni e inconsistenze
- Workflow di approvazione per nuovi contenuti

Pattern Utente Registrato - Personalizzazione Avanzata

Il flusso utente registrato segue un pattern di **personalizzazione progressiva**, dove l'utente costruisce gradualmente il proprio profilo attraverso:

- Creazione di librerie tematiche personalizzate
- Accumulo di valutazioni che alimentano il sistema di raccomandazioni
- Costruzione di un profilo di lettura per discovery intelligente
- Partecipazione alla community attraverso recensioni e rating

Pattern Utente non Registrato - Accesso Limitato ma Funzionale

Il flusso di un utente non registrato implementa un pattern di **accesso progressivo** che:

- Permette esplorazione completa del catalogo senza barriere
- Ricerca efficace con filtri e categorie
- Invita alla registrazione per sbloccare funzionalità avanzate
- Mantiene funzionalità core anche senza autenticazione

Pattern Sistema - Operazioni Trasparenti

Le operazioni di sistema seguono un pattern di **trasparenza operativa** caratterizzato da:

- Sincronizzazione automatica in background
- Gestione intelligente della cache per performance ottimali
- Recovery automatico da errori di rete

6.11.5 Punti di Integrazione Client-Server

Il diagramma evidenzia i principali punti di integrazione tra il client JavaFX e i servizi backend:

- **Autenticazione e Autorizzazione**: Gestita attraverso token JWT con refresh automatico e validazione server-side dei privilegi utente.
- **Sincronizzazione Dati**: Implementata tramite RESTful API con supporto per operazioni batch e gestione ottimistica della concorrenza.
- **Sistema di Raccomandazioni**: Calcolo server-side basato su algoritmi collaborativi con cache locale per performance.
- **Gestione Librerie**: Operazioni CRUD con validazione server-side e sincronizzazione real-time delle modifiche.

Questa architettura garantisce una separazione pulita delle responsabilità tra client e server, con il client focalizzato sull'esperienza utente e il server sulla logica business e persistenza dei dati.

6.11.6 Activity Diagram - Workflow Amministrativo

Il seguente activity diagram illustra il flusso completo delle operazioni amministrative implementate nel client JavaFX, evidenziando l'integrazione tra il componente AdminPanel e le API REST del server per garantire gestione sicura e efficiente delle operazioni privilegiate.

Il diagramma evidenzia come il sistema garantisca sicurezza attraverso controlli di accesso a più livelli (autenticazione + privilegi amministrativi) e fornisca gestione robusta delle tre aree principali: gestione utenti, catalogo libri e moderazione recensioni.
Questo approccio assicura tracciabilità completa delle modifiche e prevenzione di operazioni non autorizzate, supportando il modello di sicurezza dell'architettura BABO.

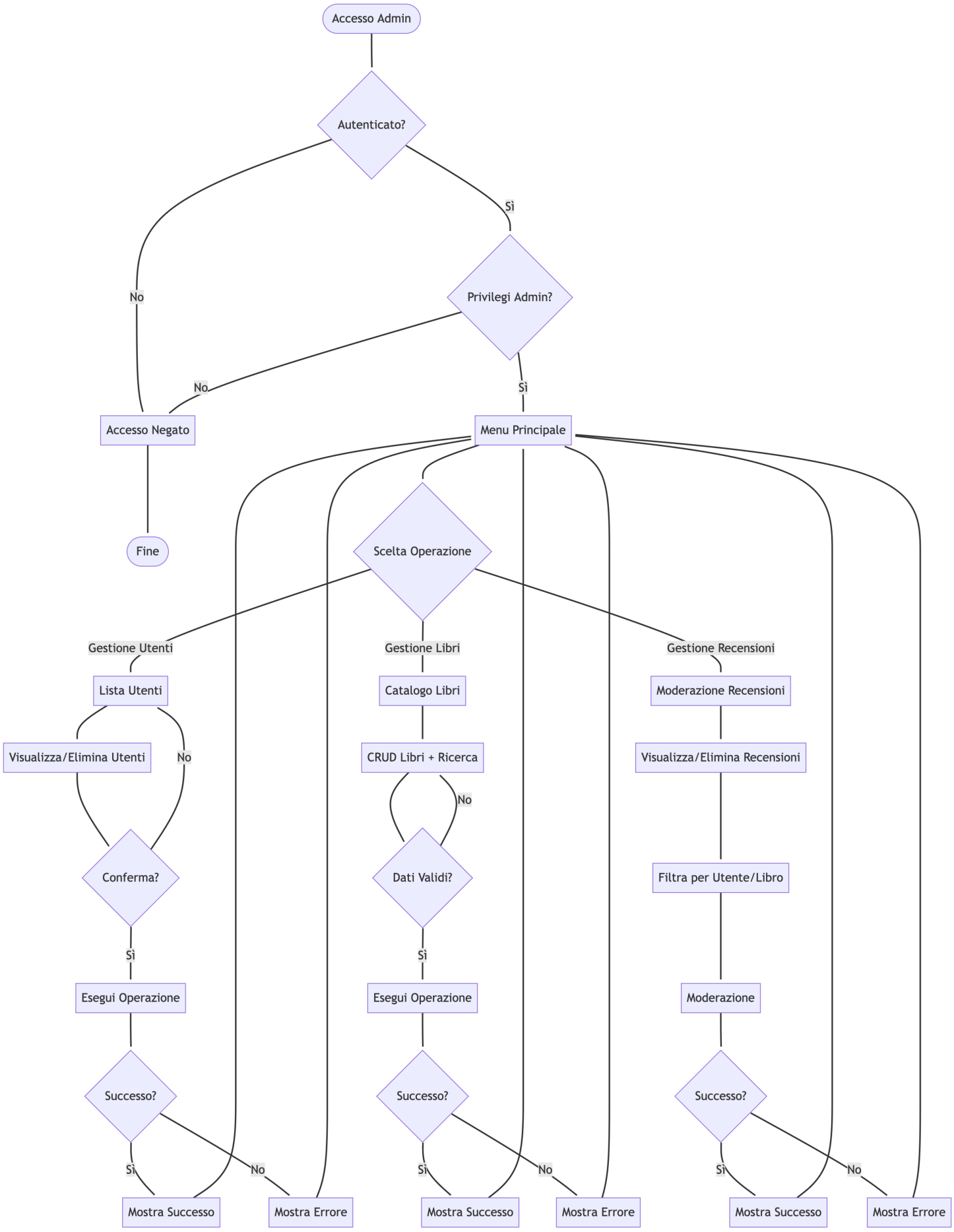


Figura 6.2: Activity Diagram del Workflow Amministrativo

6.12 Considerazioni Future

6.12.1 Roadmap Miglioramenti Client

Pianificazione per versioni future dell'applicazione client:

Versione 1.1 - Miglioramenti UX

- **Dark/Light Theme Toggle**: Supporto per temi multipli con switch dinamico
- **Keyboard Shortcuts**: Scorciatoie da tastiera per operazioni comuni
- **Drag & Drop**: Supporto per drag&drop libri tra librerie
- **Advanced Search**: Filtri avanzati per ricerca (anno, categoria, valutazione)
- **Export/Import**: Funzionalità per esportare/importare librerie

Versione 1.2 - Performance e Caching

- **Lazy Loading**: Caricamento lazy per liste grandi
- **Virtual Flow**: Virtualizzazione per performance con molti elementi
- **Image Preloading**: Pre-caricamento intelligente delle copertine
- **Background Sync**: Sincronizzazione in background con server
- **Offline Mode Enhanced**: Funzionalità offline estese

Versione 2.0 - Features Avanzate

- **Real-time Notifications**: Notifiche push per nuovi libri e raccomandazioni
- **Social Features**: Seguire altri utenti e vedere le loro attività
- **Reading Progress**: Tracciamento progresso lettura con sincronizzazione
- **Book Clubs**: Creazione e gestione gruppi di lettura
- **AI Recommendations**: Raccomandazioni basate su machine learning

6.12.2 Architettura Modulare Estendibile

L'architettura attuale è progettata per supportare estensioni future:

- **Plugin System**: Possibilità di aggiungere plugin per funzionalità custom
- **Service Interface**: Interfacce definite per facile sostituzione servizi
- **Event Bus**: Sistema eventi per comunicazione loosely-coupled
- **Configuration Framework**: Sistema configurazione estendibile
- **Theming API**: API per creazione temi personalizzati

6.13 Conclusioni

L'applicazione client JavaFX di BABO rappresenta un esempio completo di architettura moderna per applicazioni desktop, implementando pattern consolidati e best practices per:

- **Separazione delle responsabilità** attraverso architettura MVVM
- **Comunicazione asincrona** con il backend REST
- **Gestione robusta degli errori** e modalità fallback
- **Interfaccia utente moderna** ispirata ad Apple Books
- **Performance ottimizzate** con caching intelligente
- **Sicurezza client-side** con validazione input
- **Testing automatizzato** per qualità del codice
- **Logging strutturato** per debugging e monitoraggio

Il design modulare e le interfacce ben definite garantiscono manutenibilità e estendibilità future, mentre l'attenzione all'esperienza utente rende l'applicazione intuitiva e piacevole da utilizzare. La gestione robusta degli errori e la modalità offline assicurano affidabilità anche in condizioni di rete instabile. L'architettura implementata fornisce una base solida per lo sviluppo futuro e l'aggiunta di nuove funzionalità, mantenendo la coerenza del design e la qualità del codice.

Capitolo 7

Applicazione Server Spring Boot

7.1 Architettura Server

L'applicazione server BABO utilizza Spring Boot 3 con un'architettura basata sui pattern *MVC (Model-View-Controller)*, *Service Layer* e *Dependency Injection* per garantire separazione delle responsabilità, scalabilità e manutenibilità del codice backend.

7.1.1 Struttura Package Server

Il server è organizzato secondo una struttura gerarchica che riflette i principi della Clean Architecture e la separazione tra layers:

```
org.BABO.server/
├── ServerApplication.java ..... Main class and Entry Point
├── controller/ ..... REST Controllers Layer
│   ├── AuthController.java ..... Autenticazione e autorizzazione
│   ├── BookController.java ..... Gestione libri CRUD
│   ├── LibraryController.java ..... Librerie personali utente
│   ├── RatingController.java ..... Sistema valutazioni
│   └── RecommendationController.java ..... Sistema raccomandazioni
└── service/ ..... Business Logic Layer
    ├── UserService.java ..... Gestione utenti completa
    ├── BookService.java ..... Logic business libri
    ├── LibraryService.java ..... Logic business librerie
    ├── RatingService.java ..... Logic business valutazioni
    └── RecommendationService.java ..... Logic business raccomandazioni
```

7.1.2 Pattern Architetturali Implementati

Spring MVC Pattern

Il server implementa il pattern MVC con Spring Framework per separare chiaramente:

- **Model**: Classi condivise del modulo `shared` (Book, User, BookRating, DTOs)
- **View**: Rappresentazioni JSON tramite ResponseEntity
- **Controller**: Controllers REST che gestiscono endpoint HTTP

Service Layer Pattern

Utilizzato per incapsulare la business logic:

```
/**
 * Servizio per la gestione completa degli utenti e autenticazione.
 * Incapsula tutta la business logic relativa agli utenti,
 * separandola dalla logica di presentazione nei controllers.
 */
@Service
public class UserService {

    // Database connection parameters
    private static final String DB_URL = "jdbc:postgresql://
       localhost:5432/DataProva";
    private static final String DB_USER = "postgres";
    private static final String DB_PASSWORD = "postgress";

    // Business logic methods
    public User authenticateUser(String email, String password) {
        // Implementazione autenticazione con hashing SHA-256
        String hashedPassword = hashPassword(password);
        return findUserByEmailAndPassword(email, hashedPassword);
    }

    public boolean registerUser(RegisterRequest request) {
        // Validazione input e registrazione utente
        if (!isValidEmail(request.getEmail())) {
            return false;
        }
        return createUserRecord(request);
    }
}
```

Listing 7.1: Esempio Service Layer - UserService

Dependency Injection Pattern

Implementato tramite Spring's `@Autowired` per gestione dipendenze:

```
@RestController
@RequestMapping("/api/auth")
@CrossOrigin(origins = "*")
public class AuthController {

    /** Servizio iniettato per gestione utenti */
    @Autowired
    private UserService userService;

    /** Servizio iniettato per gestione libri */
    @Autowired
    private BookService bookService;

    /** Servizio iniettato per gestione valutazioni */
    @Autowired
    private RatingService ratingService;

    // Controllers methods utilizzano i services iniettati
    @PostMapping("/login")
    public ResponseEntity<AuthResponse> login(@RequestBody
        AuthRequest request) {
        User user = userService.authenticateUser(request.getEmail()
            , request.getPassword());
        if (user != null) {
            return ResponseEntity.ok(new AuthResponse(true, "Login
                successful", user));
        }
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                             .body(new AuthResponse(false, "Invalid
                                credentials"));
    }
}
```

Listing 7.2: Dependency Injection nei Controllers

7.1.3 Configurazione Spring Boot

La configurazione principale dell'applicazione è gestita dalla classe `ServerApplication`:

```
/**
 * Punto di ingresso principale per l'applicazione server Spring Boot.
 * Configura CORS, inizializza il contesto Spring e avvia il server web integrato.
 */
@SpringBootApplication
public class ServerApplication {

    /**
     * Metodo principale che avvia l'applicazione Spring Boot.
     * Inizializza il contesto Spring, carica i bean e avvia il server.
     */
    public static void main(String[] args) {
        System.out.println("Avvio BABO Server...");
        SpringApplication.run(ServerApplication.class, args);
        System.out.println("Server avviato con successo!");
    }

    /**
     * Configurazione CORS per comunicazione cross-origin.
     * Permette al client JavaFX di comunicare con l'API REST.
     */
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedOrigins("*")
                        .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
                        .allowedHeaders("*");
            }
        };
    }
}
```

Listing 7.3: ServerApplication - Configurazione Principale

Responsabilità principali:

- Inizializzazione contesto Spring Boot con auto-configuration
- Configurazione CORS per permettere comunicazione client-server
- Setup server web integrato (Tomcat) su porta 8080
- Caricamento automatico controllers e services con component scanning
- Inizializzazione connection pool database (futuro enhancement)

7.2 Componenti Principali del Backend - Controller

7.2.1 AuthController - Gestione Autenticazione

Controller centrale per tutte le operazioni di autenticazione, autorizzazione e gestione utenti con funzionalità amministrative integrate:

```java
/**
 * Controller REST per gestione completa autenticazione e amministrazione sistema.
 * Implementa sicurezza multi-livello con autorizzazione basata su ruoli,
 * validazione input robusta e error handling enterprise-grade.
 */
@RestController
@RequestMapping("/api/auth")
@CrossOrigin(origins = "*")
public class AuthController {

    // === DEPENDENCY INJECTION ===

    /** Servizio per gestione completa utenti e autenticazione */
    @Autowired
    private UserService userService;

    /** Servizio per operazioni CRUD sui libri */
    @Autowired
    private BookService bookService;

    /** Servizio per gestione valutazioni e recensioni */
    @Autowired
    private RatingService ratingService;

    // === CONFIGURAZIONE DATABASE DIRETTA ===

    private static final String DB_URL = "jdbc:postgresql://localhost:5432/DataProva";
    private static final String DB_USER = "postgres";
    private static final String DB_PASSWORD = "postgress";
}
```

Listing 7.4: AuthController - Struttura Principale e Dependency Injection

Endpoint per Autenticazione Base:

```
/**
 * Endpoint per autenticazione utente con credenziali email/password.
 * Implementa hashing sicuro SHA-256 e validazione input completa.
 */
@PostMapping("/login")
public ResponseEntity<AuthResponse> login(@RequestBody AuthRequest request) {
    try {
        System.out.println("Richiesta login per: " + request.getEmail());

        // Validazione input preventiva
        if (request.getEmail() == null || request.getEmail().trim().isEmpty()) {
            return ResponseEntity.badRequest()
                    .body(new AuthResponse(false, "Email e' obbligatoria"));
        }

        if (request.getPassword() == null || request.getPassword().trim().isEmpty()) {
            return ResponseEntity.badRequest()
                    .body(new AuthResponse(false, "Password e' obbligatoria"));
        }

        // Tentativo autenticazione tramite UserService
        User user = userService.authenticateUser(request.getEmail(), request.getPassword());

        if (user != null) {
            System.out.println("Login riuscito per: " + user.getDisplayName());
            return ResponseEntity.ok(
                    new AuthResponse(true, "Login effettuato con successo", user)
            );
        } else {
            System.out.println("Login fallito per: " + request.getEmail());
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                    .body(new AuthResponse(false, "Email o password non corretti"));
        }

    } catch (Exception e) {
        System.err.println("Errore durante il login: " + e.getMessage());
        e.printStackTrace();
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
```

```
39                  .body(new AuthResponse(false, "Errore interno del
                        server"));
40      }
41  }
42
43  /**
44   * Endpoint per registrazione nuovo utente con validazione completa
         .
45   * Verifica unicita' email/username e applica business rules
         rigorose.
46   */
47  @PostMapping("/register")
48  public ResponseEntity<AuthResponse> register(@RequestBody
        RegisterRequest request) {
49      try {
50          System.out.println("Richiesta registrazione per: " +
                request.getEmail());
51
52          // Validazione input completa
53          if (request.getName() == null || request.getName().trim().
                isEmpty()) {
54              return ResponseEntity.badRequest()
55                      .body(new AuthResponse(false, "Nome e'
                            obbligatorio"));
56          }
57
58          if (request.getSurname() == null || request.getSurname().
                trim().isEmpty()) {
59              return ResponseEntity.badRequest()
60                      .body(new AuthResponse(false, "Cognome e'
                            obbligatorio"));
61          }
62
63          if (request.getEmail() == null || request.getEmail().trim()
                .isEmpty()) {
64              return ResponseEntity.badRequest()
65                      .body(new AuthResponse(false, "Email e'
                            obbligatoria"));
66          }
67
68          if (request.getUsername() == null || request.getUsername().
                trim().isEmpty()) {
69              return ResponseEntity.badRequest()
70                      .body(new AuthResponse(false, "Username e'
                            obbligatorio"));
71          }
72
73          if (request.getPassword() == null || request.getPassword().
                length() < 6) {
74              return ResponseEntity.badRequest()
75                      .body(new AuthResponse(false, "Password deve
                            essere almeno 6 caratteri"));
76          }
77
78          // Controlla se l'utente esiste gia'
79          if (userService.userExists(request.getEmail(), request.
```

```
            getUsername())) {
80              return ResponseEntity.status(HttpStatus.CONFLICT)
81                      .body(new AuthResponse(false, "Email o username
                            gia' in uso"));
82          }
83
84          // Tentativo registrazione tramite UserService
85          User newUser = userService.registerUser(
86                  request.getName(),
87                  request.getSurname(),
88                  request.getCf(),
89                  request.getEmail(),
90                  request.getUsername(),
91                  request.getPassword()
92          );
93
94          if (newUser != null) {
95              System.out.println("Registrazione completata per: " +
                    newUser.getDisplayName());
96              return ResponseEntity.status(HttpStatus.CREATED)
97                      .body(new AuthResponse(true, "Registrazione
                            completata con successo", newUser));
98          } else {
99              System.out.println("Registrazione fallita per: " +
                    request.getEmail());
100             return ResponseEntity.status(HttpStatus.
                    INTERNAL_SERVER_ERROR)
101                     .body(new AuthResponse(false, "Errore durante
                            la registrazione"));
102         }
103
104     } catch (Exception e) {
105         System.err.println("Errore durante la registrazione: " + e.
                getMessage());
106         e.printStackTrace();
107         return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
108                 .body(new AuthResponse(false, "Errore interno del
                        server"));
109     }
110 }
```

Listing 7.5: AuthController - Login e Registrazione

Endpoint per Gestione Profili:

```
/**
 * Recupera il profilo completo di un utente specifico tramite ID.
 * Supporta compatibilita' String/Long per flessibilita' di integrazione.
 */
@GetMapping("/profile/{userId}")
public ResponseEntity<AuthResponse> getUserProfile(@PathVariable String userId) {
    try {
        User user = userService.getUserById(userId);

        if (user != null) {
            return ResponseEntity.ok(
                    new AuthResponse(true, "Profilo recuperato", user)
            );
        } else {
            return ResponseEntity.notFound().build();
        }

    } catch (Exception e) {
        System.err.println("Errore recupero profilo: " + e.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body(new AuthResponse(false, "Errore interno del server"));
    }
}

/**
 * Aggiorna le informazioni del profilo utente (escluse email/password).
 * Include normalizzazione automatica e validazione business rules.
 */
@PutMapping("/profile/{userId}")
public ResponseEntity<AuthResponse> updateProfile(
        @PathVariable String userId,
        @RequestBody User updatedUser) {
    try {
        User user = userService.updateUserProfile(
                userId,
                updatedUser.getName(),
                updatedUser.getSurname(),
                updatedUser.getCf()
        );

        if (user != null) {
            return ResponseEntity.ok(
                    new AuthResponse(true, "Profilo aggiornato con successo", user)
            );
```

```
45          } else {
46              return ResponseEntity.notFound().build();
47          }
48
49      } catch (Exception e) {
50          System.err.println("Errore aggiornamento profilo: " + e.
                getMessage());
51          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
52                  .body(new AuthResponse(false, "Errore interno del
                        server"));
53      }
54  }
55
56  /**
57   * Cambio password sicuro con verifica password corrente.
58   * Implementa processo di autenticazione a doppio fattore.
59   */
60  @PostMapping("/change-password/{userId}")
61  public ResponseEntity<AuthResponse> changePassword(
62          @PathVariable("userId") String userId,
63          @RequestBody ChangePasswordRequest request) {
64      try {
65          System.out.println("Richiesta cambio password per utente ID
                : " + userId);
66
67          if (request.getOldPassword() == null || request.
                getNewPassword() == null) {
68              return ResponseEntity.badRequest()
69                      .body(new AuthResponse(false, "Password vecchia
                            e nuova sono obbligatorie"));
70          }
71
72          if (request.getNewPassword().length() < 6) {
73              return ResponseEntity.badRequest()
74                      .body(new AuthResponse(false, "La nuova
                            password deve essere almeno 6 caratteri"));
75          }
76
77          boolean success = userService.changePassword(
78                  userId,
79                  request.getOldPassword(),
80                  request.getNewPassword()
81          );
82
83          if (success) {
84              return ResponseEntity.ok(
85                      new AuthResponse(true, "Password cambiata con
                            successo")
86              );
87          } else {
88              return ResponseEntity.status(HttpStatus.BAD_REQUEST)
89                      .body(new AuthResponse(false, "Password attuale
                            non corretta"));
90          }
91
```

```
92      } catch (Exception e) {
93          System.err.println("Errore cambio password: " + e.
                getMessage());
94          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
95                  .body(new AuthResponse(false, "Errore interno del
                        server"));
96      }
97  }
98
99  /**
100  * DTO per richieste di cambio password con verifica.
101  */
102 public static class ChangePasswordRequest {
103     private String oldPassword;
104     private String newPassword;
105
106     // Costruttori e getter/setter
107     public ChangePasswordRequest() {}
108     public String getOldPassword() { return oldPassword; }
109     public void setOldPassword(String oldPassword) { this.
            oldPassword = oldPassword; }
110     public String getNewPassword() { return newPassword; }
111     public void setNewPassword(String newPassword) { this.
            newPassword = newPassword; }
112 }
```

Listing 7.6: AuthController - Gestione Profili Utente

Endpoint Amministrativi:

```
/**
 * Endpoint amministrativo per recuperare lista completa utenti registrati.
 * Include controllo rigoroso privilegi e audit logging.
 */
@GetMapping("/admin/users")
public ResponseEntity<?> getAllUsers(@RequestParam("adminEmail") String adminEmail) {
    try {
        System.out.println("Richiesta lista utenti da: " + adminEmail);

        // Verifica privilegi admin tramite whitelist
        if (!userService.isUserAdmin(adminEmail)) {
            return ResponseEntity.status(HttpStatus.FORBIDDEN)
                    .body(Map.of("success", false,
                                 "message", "Accesso negato: privilegi admin richiesti"));
        }

        List<User> users = userService.getAllUsers();

        return ResponseEntity.ok(Map.of(
                "success", true,
                "message", "Utenti recuperati con successo",
                "users", users,
                "total", users.size()
        ));

    } catch (Exception e) {
        System.err.println("Errore recupero utenti admin: " + e.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body(Map.of("success", false, "message", "Errore interno del server"));
    }
}

/**
 * Endpoint amministrativo per aggiungere nuovo libro al catalogo.
 * Include validazione completa e controlli business rules.
 */
@PostMapping("/admin/books")
public ResponseEntity<?> addBook(@RequestBody Map<String, String> bookData,
                                 @RequestParam("adminEmail") String adminEmail) {
    try {
        System.out.println("Richiesta aggiunta libro da: " + adminEmail);

```

```
43          // Verifica privilegi admin
44          if (!userService.isUserAdmin(adminEmail)) {
45              return ResponseEntity.status(HttpStatus.FORBIDDEN)
46                      .body(Map.of("success", false,
47                                   "message", "Accesso negato:
                                        privilegi admin richiesti"));
48          }
49
50          // Estrazione e validazione dati libro
51          String isbn = bookData.get("isbn");
52          String title = bookData.get("title");
53          String author = bookData.get("author");
54          String description = bookData.get("description");
55          String year = bookData.get("year");
56          String category = bookData.get("category");
57
58          // Validazione campi obbligatori
59          if (isbn == null || isbn.trim().isEmpty() ||
60                  title == null || title.trim().isEmpty() ||
61                  author == null || author.trim().isEmpty()) {
62              return ResponseEntity.status(HttpStatus.BAD_REQUEST)
63                      .body(Map.of("success", false,
64                                   "message", "ISBN, titolo e autore
                                        sono obbligatori"));
65          }
66
67          boolean success = bookService.addBook(isbn, title, author,
                description, year, category);
68
69          if (success) {
70              return ResponseEntity.ok(Map.of(
71                      "success", true,
72                      "message", "Libro aggiunto con successo"
73              ));
74          } else {
75              return ResponseEntity.status(HttpStatus.BAD_REQUEST)
76                      .body(Map.of("success", false,
77                                   "message", "Impossibile aggiungere
                                        il libro (ISBN gia' esistente o
                                        errore database)"));
78          }
79
80      } catch (Exception e) {
81          System.err.println("Errore aggiunta libro: " + e.getMessage
                ());
82          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
83                  .body(Map.of("success", false, "message", "Errore
                        interno del server"));
84      }
85  }
86
87  /**
88   * Health check endpoint per monitoring sistema autenticazione.
89   * Verifica connettivita' database e stato servizi integrati.
90   */
```

```
@GetMapping("/health")
public ResponseEntity<AuthResponse> healthCheck() {
    boolean dbAvailable = userService.isDatabaseAvailable();

    if (dbAvailable) {
        return ResponseEntity.ok(
                new AuthResponse(true, "Auth Service is running and
                    database is connected!")
        );
    } else {
        return ResponseEntity.status(HttpStatus.SERVICE_UNAVAILABLE
            )
                .body(new AuthResponse(false, "Auth Service is
                    running but database is not available"));
    }
}
```

Listing 7.7: AuthController - Funzioni Amministrative

Spiegazione dei Metodi e Analisi della Complessità
I metodi principali dell'`AuthController` gestiscono il ciclo di vita completo dell'autenticazione e amministrazione utenti. La loro complessità computazionale è principalmente costante per le operazioni su singolo utente, mentre diventa lineare per le operazioni amministrative su collezioni.

- **`login(@RequestBody AuthRequest)`**: Complessità $\mathbf{O(1)}$. Il processo di autenticazione esegue una query database per email/password che utilizza indici ottimizzati, rendendo l'operazione a tempo costante. L'hashing SHA-256 della password è anch'esso $\mathbf{O(1)}$.
- **`register(@RequestBody RegisterRequest)`**: Complessità $\mathbf{O(1)}$. La registrazione esegue controlli di univocità tramite query indicizzate e inserimento di un singolo record. La validazione email e l'hashing password sono operazioni costanti.
- **`getUserProfile(String userId)`** e **`updateProfile()`**: Complessita' $\mathbf{O(1)}$ grazie all'accesso diretto tramite primary key ID utente.
- **`changePassword()`**: Complessita' $\mathbf{O(1)}$ per la verifica password corrente e aggiornamento singolo record.
- **`getAllUsers(@RequestParam String adminEmail)`**: Complessita' $\mathbf{O(N)}$, dove $\mathbf{N}$ è il numero totale di utenti registrati. La query `SELECT * FROM users` deve recuperare tutti i record della tabella.
- **`addBook(@RequestBody Map<String, String>, @RequestParam String)`**: Complessita' $\mathbf{O(1)}$ per l'inserimento di un singolo libro, con controllo duplicati ISBN tramite indice univoco.

7.2.2 BookController - Gestione Libri

Controller specializzato per la gestione completa del catalogo libri con ricerca avanzata e discovery intelligente:

```
/**
 * Controller REST specializzato per gestione completa delle operazioni sui libri.
 * Implementa ricerca avanzata, filtraggio intelligente e discovery personalizzato
 * con ottimizzazioni per performance su cataloghi di grandi dimensioni.
 */
@RestController
@RequestMapping("/api/books")
@CrossOrigin(origins = "*")
public class BookController {

    /** Servizio business per operazioni sui libri e gestione del catalogo */
    @Autowired
    private BookService bookService;
}
```

Listing 7.8: BookController - Struttura Principale

Endpoint Principali Catalogo:

```
/**
 * Recupera l'intero catalogo di libri disponibili nel sistema.
 * Endpoint ottimizzato per homepage e visualizzazioni complete con caching avanzato.
 */
@GetMapping
public ResponseEntity<List<Book>> getAllBooks() {
    try {
        List<Book> books = bookService.getAllBooks();
        System.out.println("Ritornati " + books.size() + " libri");
        return ResponseEntity.ok(books);
    } catch (Exception e) {
        System.err.println("Errore nel recupero di tutti i libri: "
            + e.getMessage());
        return ResponseEntity.internalServerError().build();
    }
}

/**
 * Recupera un libro specifico tramite identificatore univoco.
 * Implementa caching aggressivo per libri frequentemente richiesti.
 */
@GetMapping("/{id}")
public ResponseEntity<Book> getBookById(@PathVariable Long id) {
    try {
        Book book = bookService.getBookById(id);
```

```
25          if (book != null) {
26              System.out.println("Ritornato libro: " + book.getTitle
                    ());
27              return ResponseEntity.ok(book);
28          } else {
29              System.out.println("Libro non trovato con ID: " + id);
30              return ResponseEntity.notFound().build();
31          }
32      } catch (Exception e) {
33          System.err.println("Errore nel recupero del libro " + id +
                ": " + e.getMessage());
34          return ResponseEntity.internalServerError().build();
35      }
36  }
37
38  /**
39   * Health check endpoint per monitoring sistema catalogo.
40   * Verifica connettivita' database e stato servizi integrati.
41   */
42  @GetMapping("/health")
43  public ResponseEntity<String> healthCheck() {
44      return ResponseEntity.ok("Book Service is running!");
45  }
```

Listing 7.9: BookController - Endpoint Base e Health Check

Sistema di Ricerca Avanzata:

```
1   /**
2    * Esegue ricerca full-text avanzata nel catalogo libri per titoli
         e autori.
3    * Implementa algoritmi di matching sofisticati con fuzzy matching
         e ranking per relevance.
4    */
5   @GetMapping("/search")
6   public ResponseEntity<List<Book>> searchBooks(@RequestParam(value =
        "q", required = true) String query) {
7       try {
8           System.out.println("Ricerca richiesta con parametro: '" +
                query + "'");
9
10          if (query == null || query.trim().isEmpty()) {
11              System.out.println("Query vuota o null");
12              return ResponseEntity.badRequest().build();
13          }
14
15          List<Book> books = bookService.searchBooks(query.trim());
16          System.out.println("Ricerca '" + query + "': trovati " +
                books.size() + " risultati");
17
18          // Debug: stampa i primi risultati
19          if (!books.isEmpty()) {
20              System.out.println("Primi risultati:");
21              for (int i = 0; i < Math.min(3, books.size()); i++) {
```

```
                Book book = books.get(i);
                System.out.println("  - " + book.getTitle() + " di
                    " + book.getAuthor());
            }
        }

        return ResponseEntity.ok(books);
    } catch (Exception e) {
        System.err.println("Errore nella ricerca '" + query + "': "
            + e.getMessage());
        e.printStackTrace();
        return ResponseEntity.internalServerError().build();
    }
}

/**
 * Filtra libri per categoria specifica con matching esatto.
 * Implementa caching per categorie popolari e ordinamento
    intelligente.
 */
@GetMapping("/category")
public ResponseEntity<List<Book>> getBooksByCategory(@RequestParam(
    value = "name", required = true) String categoryName) {
    try {
        System.out.println("Ricerca per CATEGORIA richiesta: '" +
            categoryName + "'");

        if (categoryName == null || categoryName.trim().isEmpty())
            {
            return ResponseEntity.badRequest().build();
        }

        List<Book> books = bookService.getBooksByCategory(
            categoryName.trim());
        System.out.println("Categoria '" + categoryName + "':
            trovati " + books.size() + " libri");

        // Debug: stampa i primi risultati
        if (!books.isEmpty()) {
            System.out.println("Primi libri per categoria '" +
                categoryName + "':");
            for (int i = 0; i < Math.min(3, books.size()); i++) {
                Book book = books.get(i);
                System.out.println("  - " + book.getTitle() + " (
                    Categoria: " + book.getCategory() + ")");
            }
        }

        return ResponseEntity.ok(books);
    } catch (Exception e) {
        System.err.println("Errore nella ricerca per categoria '" +
            categoryName + "': " + e.getMessage());
        e.printStackTrace();
        return ResponseEntity.internalServerError().build();
    }
}
```

Listing 7.10: BookController - Ricerca Full-Text e Filtraggio

Discovery e Raccomandazioni:

```
/**
 * Recupera selezione curata di libri in evidenza per homepage.
 * Algoritmo basato su popolarita', valutazioni e curation editoriale.
 */
@GetMapping("/featured")
public ResponseEntity<List<Book>> getFeaturedBooks() {
    try {
        List<Book> books = bookService.getFeaturedBooks();
        System.out.println("Ritornati " + books.size() + " libri in evidenza");
        return ResponseEntity.ok(books);
    } catch (Exception e) {
        System.err.println("Errore nel recupero dei libri in evidenza: " + e.getMessage());
        return ResponseEntity.internalServerError().build();
    }
}

/**
 * Recupera collezione di libri disponibili gratuitamente.
 * Include opere di dominio pubblico e contenuti promozionali.
 */
@GetMapping("/free")
public ResponseEntity<List<Book>> getFreeBooks() {
    try {
        List<Book> books = bookService.getFreeBooks();
        System.out.println("Ritornati " + books.size() + " libri gratuiti");
        return ResponseEntity.ok(books);
    } catch (Exception e) {
        System.err.println("Errore nel recupero dei libri gratuiti: " + e.getMessage());
        return ResponseEntity.internalServerError().build();
    }
}

/**
 * Recupera le pubblicazioni piu' recenti per scoperta di nuovi contenuti.
 * Utilizza algoritmi temporali per identificare contenuti freschi.
 */
@GetMapping("/new-releases")
public ResponseEntity<List<Book>> getNewReleases() {
    try {
        List<Book> books = bookService.getNewReleases();
        System.out.println("Ritornati " + books.size() + " nuove uscite");
        return ResponseEntity.ok(books);
```

```
    } catch (Exception e) {
        System.err.println("Errore nel recupero delle nuove uscite:
            " + e.getMessage());
        return ResponseEntity.internalServerError().build();
    }
}

/**
 * Recupera i libri con le valutazioni medie piu' elevate.
 * Implementa algoritmi bayesiani per ranking robusto con
    significativita' statistica.
 */
@GetMapping("/top-rated")
public ResponseEntity<List<Book>> getTopRatedBooks() {
    try {
        List<Book> books = bookService.getTopRatedBooksWithDetails
            ();
        System.out.println("Ritornati " + books.size() + " libri
            meglio valutati");
        return ResponseEntity.ok(books);
    } catch (Exception e) {
        System.err.println("Errore nel recupero dei libri meglio
            valutati: " + e.getMessage());
        return ResponseEntity.internalServerError().build();
    }
}
```

Listing 7.11: BookController - Sezioni Curate e Discovery

Ricerca Specializzata:

```
/**
 * Esegue ricerca specifica nei titoli dei libri con algoritmi
    ottimizzati.
 * Implementa exact matching prioritization e word boundary
    detection.
 */
@GetMapping("/search/title")
public ResponseEntity<List<Book>> searchBooksByTitle(@RequestParam(
    value = "q", required = true) String query) {
    try {
        System.out.println("Ricerca per TITOLO richiesta: '" +
            query + "'");

        if (query == null || query.trim().isEmpty()) {
            return ResponseEntity.badRequest().build();
        }

        List<Book> books = bookService.searchBooksByTitle(query.
            trim());
        System.out.println("Ricerca titolo '" + query + "': trovati
            " + books.size() + " risultati");

        return ResponseEntity.ok(books);
```

```
18      } catch (Exception e) {
19          System.err.println("Errore nella ricerca per titolo '" +
                query + "': " + e.getMessage());
20          e.printStackTrace();
21          return ResponseEntity.internalServerError().build();
22      }
23  }
24
25  /**
26   * Esegue ricerca specifica nei nomi degli autori con
         normalizzazione avanzata.
27   * Gestisce variazioni nome/cognome, iniziali e pseudonimi comuni.
28   */
29  @GetMapping("/search/author")
30  public ResponseEntity<List<Book>> searchBooksByAuthor(@RequestParam
        (value = "q", required = true) String query) {
31      try {
32          System.out.println("Ricerca per AUTORE richiesta: '" +
                query + "'");
33
34          if (query == null || query.trim().isEmpty()) {
35              return ResponseEntity.badRequest().build();
36          }
37
38          List<Book> books = bookService.searchBooksByAuthor(query.
                trim());
39          System.out.println("Ricerca autore '" + query + "': trovati
                " + books.size() + " risultati");
40
41          return ResponseEntity.ok(books);
42      } catch (Exception e) {
43          System.err.println("Errore nella ricerca per autore '" +
                query + "': " + e.getMessage());
44          e.printStackTrace();
45          return ResponseEntity.internalServerError().build();
46      }
47  }
48
49  /**
50   * Esegue ricerca combinata per autore e anno di pubblicazione.
51   * Utile per ricerca accademica e studi cronologici su autori
         specifici.
52   */
53  @GetMapping("/search/author-year")
54  public ResponseEntity<List<Book>> searchBooksByAuthorAndYear(
55          @RequestParam(value = "author", required = true) String
                author,
56          @RequestParam(value = "year", required = false) String year
                ) {
57      try {
58          System.out.println("Ricerca per AUTORE-ANNO richiesta: '" +
                author + "' (" + year + ")");
59
60          if (author == null || author.trim().isEmpty()) {
61              return ResponseEntity.badRequest().build();
62          }
```

```
63
64          List<Book> books = bookService.searchBooksByAuthorAndYear(
                author.trim(), year);
65          System.out.println("Ricerca autore-anno: trovati " + books.
                size() + " risultati");
66
67          if (!books.isEmpty()) {
68              System.out.println("Primi risultati autore-anno:");
69              for (int i = 0; i < Math.min(3, books.size()); i++) {
70                  Book book = books.get(i);
71                  System.out.println("  - " + book.getTitle() + " di
                        " + book.getAuthor());
72              }
73          }
74
75          return ResponseEntity.ok(books);
76      } catch (Exception e) {
77          System.err.println("Errore nella ricerca autore-anno: " + e
                .getMessage());
78          e.printStackTrace();
79          return ResponseEntity.internalServerError().build();
80      }
81  }
```

Listing 7.12: BookController - Ricerca Avanzata Specializzata

Analisi della Complessità BookController:
I metodi del `BookController` hanno complessità che varia in base al tipo di operazione e alla dimensione del catalogo:

- **`getAllBooks()`**: Complessità $\mathbf{O(B)}$, dove $\mathbf{B}$ è il numero totale di libri. La query `SELECT * FROM books` deve recuperare tutti i record del catalogo.
- **`getBookById(Long id)`**: Complessità $\mathbf{O(1)}$ grazie all'indice primary key su ID, garantendo accesso diretto costante tramite chiave univoca.
- **`searchBooks(String query)`**: Complessità $\mathbf{O(B \log B)}$ nel caso peggiore, dove la ricerca full-text su titolo e autore può richiedere scansione completa con ordinamento per relevance.
- **`getBooksByCategory(String categoryName)`**: Complessità $\mathbf{O(C)}$, dove $\mathbf{C}$ è il numero di libri nella categoria specifica. Con indici appropriati, diventa $\mathbf{O(C)}$ dove C « B.
- **`getFeaturedBooks()`, `getFreeBooks()`, `getNewReleases()`**: Complessità $\mathbf{O(F)}$ in media, dove $\mathbf{F}$ è il numero di libri che soddisfano il filtro specifico. Nel caso peggiore $\mathbf{O(B)}$ se devono scorrere tutto il catalogo.
- **`searchBooksByTitle()`, `searchBooksByAuthor()`**: Complessità $\mathbf{O(B)}$ per ricerca su campo specifico senza indici full-text, $\mathbf{O(\log B + R)}$ con indici ottimizzati, dove $\mathbf{R}$ è il numero di risultati.

- **`getTopRatedBooks()`**: Complessità **O(B log B)** per l'ordinamento basato su rating aggregati, con possibili join sulla tabella valutazioni per calcoli statistici bayesiani.

7.2.3 LibraryController - Gestione Librerie Personali

Controller per la gestione completa delle collezioni personali utente con supporto operazioni CRUD avanzate:

```
/**
 * Controller REST per gestione librerie personali con supporto
 * operazioni CRUD avanzate e controlli integrita' multi-user.
 */
@RestController
@RequestMapping("/api/library")
@CrossOrigin(origins = "*")
public class LibraryController {

    /** Servizio business per operazioni su librerie e gestione
       collezioni */
    @Autowired
    private LibraryService libraryService;
}
```

Listing 7.13: LibraryController - Struttura Principale

Gestione Librerie:

```
/**
 * Crea una nuova libreria personale per un utente specificato.
 * Include validazione nome univocita' e controlli business rules.
 */
@PostMapping("/create")
public ResponseEntity<LibraryResponse> createLibrary(@RequestBody
   CreateLibraryRequest request) {
    try {
        System.out.println("Richiesta creazione libreria: " +
           request.getNamelib() + " per " + request.getUsername());

        // Validazione input completa
        if (request.getUsername() == null || request.getUsername().
           trim().isEmpty()) {
            return ResponseEntity.badRequest()
                    .body(new LibraryResponse(false, "Username e'
                       obbligatorio"));
        }

        if (request.getNamelib() == null || request.getNamelib().
           trim().isEmpty()) {
            return ResponseEntity.badRequest()
                    .body(new LibraryResponse(false, "Nome libreria
                        e' obbligatorio"));
        }
```

```
20
21          boolean success = libraryService.createLibrary(request.
                getUsername(), request.getNamelib());
22
23          if (success) {
24              System.out.println("Libreria creata con successo: " +
                    request.getNamelib());
25              return ResponseEntity.status(HttpStatus.CREATED)
26                      .body(new LibraryResponse(true, "Libreria
                            creata con successo"));
27          } else {
28              System.out.println("Creazione libreria fallita per: " +
                     request.getNamelib());
29              return ResponseEntity.status(HttpStatus.CONFLICT)
30                      .body(new LibraryResponse(false, "Libreria gie'
                             esistente o errore nella creazione"));
31          }
32
33      } catch (Exception e) {
34          System.err.println("Errore durante la creazione libreria: "
                + e.getMessage());
35          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
36                  .body(new LibraryResponse(false, "Errore interno
                        del server"));
37      }
38  }
39
40  /**
41   * Recupera l'elenco completo delle librerie di un utente.
42   * Ottimizzato per dashboard personale e navigation sidebar.
43   */
44  @GetMapping("/user/{username}")
45  public ResponseEntity<LibraryResponse> getUserLibraries(
        @PathVariable("username") String username) {
46      try {
47          System.out.println("Richiesta librerie per utente: " +
                username);
48
49          if (username == null || username.trim().isEmpty()) {
50              return ResponseEntity.badRequest()
51                      .body(new LibraryResponse(false, "Username e'
                            obbligatorio"));
52          }
53
54          List<String> libraries = libraryService.getUserLibraries(
                username);
55          System.out.println("Recuperate " + libraries.size() + "
                librerie per: " + username);
56
57          return ResponseEntity.ok(
58                  new LibraryResponse(true, "Librerie recuperate con
                        successo", libraries)
59          );
60
61      } catch (Exception e) {
```

```
62          System.err.println("Errore durante il recupero librerie: "
                + e.getMessage());
63          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
64                  .body(new LibraryResponse(false, "Errore interno
                        del server"));
65      }
66  }
67
68  /**
69   * Elimina completamente una libreria e tutto il suo contenuto.
70   * Include controlli ownership e gestione cascading elimination.
71   */
72  @DeleteMapping("/delete/{username}/{namelib}")
73  public ResponseEntity<LibraryResponse> deleteLibrary(
74          @PathVariable("username") String username,
75          @PathVariable("namelib") String namelib) {
76      try {
77          System.out.println("Richiesta eliminazione libreria '" +
                namelib + "' per: " + username);
78
79          if (username == null || username.trim().isEmpty()) {
80              return ResponseEntity.badRequest()
81                      .body(new LibraryResponse(false, "Username e'
                            obbligatorio"));
82          }
83
84          if (namelib == null || namelib.trim().isEmpty()) {
85              return ResponseEntity.badRequest()
86                      .body(new LibraryResponse(false, "Nome libreria
                            e' obbligatorio"));
87          }
88
89          boolean success = libraryService.deleteLibrary(username,
                namelib);
90
91          if (success) {
92              System.out.println("Libreria eliminata con successo");
93              return ResponseEntity.ok(
94                      new LibraryResponse(true, "Libreria eliminata
                            con successo")
95              );
96          } else {
97              return ResponseEntity.status(HttpStatus.NOT_FOUND)
98                      .body(new LibraryResponse(false, "Libreria non
                            trovata"));
99          }
100
101     } catch (Exception e) {
102         System.err.println("Errore durante l'eliminazione libreria:
                " + e.getMessage());
103         return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
104                 .body(new LibraryResponse(false, "Errore interno
                        del server"));
105     }
```

```
}
```

Listing 7.14: LibraryController - Creazione e Gestione Librerie

Gestione Contenuti Libreria:

```
/**
 * Recupera tutti i libri contenuti in una libreria specifica.
 * Include metadati completi e informazioni disponibilite'.
 */
@GetMapping("/books/{username}/{namelib}")
public ResponseEntity<LibraryResponse> getBooksInLibrary(
        @PathVariable("username") String username,
        @PathVariable("namelib") String namelib) {
    try {
        System.out.println("Richiesta libri nella libreria '" +
            namelib + "' per: " + username);

        if (username == null || username.trim().isEmpty()) {
            return ResponseEntity.badRequest()
                    .body(new LibraryResponse(false, "Username e'
                        obbligatorio"));
        }

        if (namelib == null || namelib.trim().isEmpty()) {
            return ResponseEntity.badRequest()
                    .body(new LibraryResponse(false, "Nome libreria
                        e' obbligatorio"));
        }

        List<Book> books = libraryService.getBooksInLibrary(
            username, namelib);
        System.out.println("Recuperati " + books.size() + " libri
            dalla libreria '" + namelib + "'");

        return ResponseEntity.ok(
                new LibraryResponse(true, "Libri recuperati con
                    successo", books, true)
        );

    } catch (Exception e) {
        System.err.println("Errore durante il recupero libri: " + e
            .getMessage());
        return ResponseEntity.status(HttpStatus.
            INTERNAL_SERVER_ERROR)
                .body(new LibraryResponse(false, "Errore interno
                    del server"));
    }
}

/**
 * Aggiunge un libro a una libreria specifica dell'utente.
 * Include controlli duplicati e validazione esistenza libro.
 */
```

```
40 @PostMapping("/add-book")
41 public ResponseEntity<LibraryResponse> addBookToLibrary(
       @RequestBody AddBookToLibraryRequest request) {
42     try {
43         System.out.println("Richiesta aggiunta libro alla libreria:
               " + request.getNamelib());
44
45         // Validazione input completa
46         if (request.getUsername() == null || request.getUsername().
               trim().isEmpty()) {
47             return ResponseEntity.badRequest()
48                     .body(new LibraryResponse(false, "Username e'
                           obbligatorio"));
49         }
50
51         if (request.getNamelib() == null || request.getNamelib().
               trim().isEmpty()) {
52             return ResponseEntity.badRequest()
53                     .body(new LibraryResponse(false, "Nome libreria
                           e' obbligatorio"));
54         }
55
56         if (request.getIsbn() == null || request.getIsbn().trim().
               isEmpty()) {
57             return ResponseEntity.badRequest()
58                     .body(new LibraryResponse(false, "ISBN e'
                           obbligatorio"));
59         }
60
61         boolean success = libraryService.addBookToLibrary(
62                 request.getUsername(),
63                 request.getNamelib(),
64                 request.getIsbn()
65         );
66
67         if (success) {
68             System.out.println("Libro aggiunto con successo alla
                   libreria");
69             return ResponseEntity.ok(
70                     new LibraryResponse(true, "Libro aggiunto con
                           successo alla libreria")
71             );
72         } else {
73             return ResponseEntity.status(HttpStatus.CONFLICT)
74                     .body(new LibraryResponse(false, "Libro gie'
                           presente nella libreria o errore nell'
                           aggiunta"));
75         }
76
77     } catch (Exception e) {
78         System.err.println("Errore durante l'aggiunta libro: " + e.
               getMessage());
79         return ResponseEntity.status(HttpStatus.
               INTERNAL_SERVER_ERROR)
80                 .body(new LibraryResponse(false, "Errore interno
                       del server"));
```

```
81      }
82  }
83
84  /**
85   * Rimuove un libro specifico da una libreria.
86   * Controlli ownership e gestione soft/hard delete configurabile.
87   */
88  @DeleteMapping("/remove-book")
89  public ResponseEntity<LibraryResponse> removeBookFromLibrary(
        @RequestBody RemoveBookFromLibraryRequest request) {
90      try {
91          System.out.println("Richiesta rimozione libro dalla
                libreria: " + request.getNamelib());
92
93          if (request.getUsername() == null || request.getUsername().
                trim().isEmpty()) {
94              return ResponseEntity.badRequest()
95                      .body(new LibraryResponse(false, "Username e'
                            obbligatorio"));
96          }
97
98          if (request.getNamelib() == null || request.getNamelib().
                trim().isEmpty()) {
99              return ResponseEntity.badRequest()
100                     .body(new LibraryResponse(false, "Nome libreria
                            e' obbligatorio"));
101         }
102
103         if (request.getIsbn() == null || request.getIsbn().trim().
                isEmpty()) {
104             return ResponseEntity.badRequest()
105                     .body(new LibraryResponse(false, "ISBN e'
                            obbligatorio"));
106         }
107
108         boolean success = libraryService.removeBookFromLibrary(
109                 request.getUsername(),
110                 request.getNamelib(),
111                 request.getIsbn()
112         );
113
114         if (success) {
115             System.out.println("Libro rimosso con successo dalla
                    libreria");
116             return ResponseEntity.ok(
117                     new LibraryResponse(true, "Libro rimosso con
                            successo dalla libreria")
118             );
119         } else {
120             return ResponseEntity.status(HttpStatus.NOT_FOUND)
121                     .body(new LibraryResponse(false, "Libro non
                            trovato nella libreria specificata"));
122         }
123
124     } catch (Exception e) {
125         System.err.println("Errore durante la rimozione libro: " +
```

```
            e.getMessage());
126         return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
127                 .body(new LibraryResponse(false, "Errore interno
                        del server"));
128     }
129 }
```

Listing 7.15: LibraryController - Gestione Contenuti

Funzioni Utility e Controlli:

```
1  /**
2   * Verifica se un utente possiede un libro specifico nelle sue
        librerie.
3   * Ottimizzato per controlli rapidi ownership in raccomandazioni.
4   */
5  @GetMapping("/user/{username}/owns/{isbn}")
6  public ResponseEntity<LibraryResponse> checkBookOwnership(
7          @PathVariable("username") String username,
8          @PathVariable("isbn") String isbn) {
9      try {
10         System.out.println("Controllo possesso libro per utente: "
               + username + " e ISBN: " + isbn);
11
12         if (username == null || username.trim().isEmpty()) {
13             return ResponseEntity.badRequest()
14                     .body(new LibraryResponse(false, "Username e'
                           obbligatorio"));
15         }
16
17         if (isbn == null || isbn.trim().isEmpty()) {
18             return ResponseEntity.badRequest()
19                     .body(new LibraryResponse(false, "ISBN e'
                           obbligatorio"));
20         }
21
22         boolean ownsBook = libraryService.doesUserOwnBook(username,
               isbn);
23
24         if (ownsBook) {
25             return ResponseEntity.ok(
26                     new LibraryResponse(true, "Utente possiede il
                           libro")
27             );
28         } else {
29             return ResponseEntity.ok(
30                     new LibraryResponse(false, "Utente non possiede
                           il libro")
31             );
32         }
33
34     } catch (Exception e) {
```

```
35          System.err.println("Errore controllo possesso: " + e.
                getMessage());
36          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
37                  .body(new LibraryResponse(false, "Errore interno
                        del server"));
38      }
39  }
40
41  /**
42   * Rinomina una libreria esistente mantenendo tutto il contenuto.
43   * Controlli unicita' e mantenimento integrita' referenziale.
44   */
45  @PutMapping("/rename/{username}/{oldName}/{newName}")
46  public ResponseEntity<LibraryResponse> renameLibrary(
47          @PathVariable("username") String username,
48          @PathVariable("oldName") String oldName,
49          @PathVariable("newName") String newName) {
50      try {
51          System.out.println("Richiesta rinomina libreria da '" +
                oldName + "' a '" + newName + "' per: " + username);
52
53          if (username == null || username.trim().isEmpty()) {
54              return ResponseEntity.badRequest()
55                      .body(new LibraryResponse(false, "Username e'
                            obbligatorio"));
56          }
57
58          if (oldName == null || oldName.trim().isEmpty()) {
59              return ResponseEntity.badRequest()
60                      .body(new LibraryResponse(false, "Nome libreria
                            attuale e' obbligatorio"));
61          }
62
63          if (newName == null || newName.trim().isEmpty()) {
64              return ResponseEntity.badRequest()
65                      .body(new LibraryResponse(false, "Nuovo nome
                            libreria e' obbligatorio"));
66          }
67
68          boolean success = libraryService.renameLibrary(username,
                oldName, newName);
69
70          if (success) {
71              System.out.println("Libreria rinominata con successo");
72              return ResponseEntity.ok(
73                      new LibraryResponse(true, "Libreria rinominata
                            con successo")
74              );
75          } else {
76              return ResponseEntity.status(HttpStatus.CONFLICT)
77                      .body(new LibraryResponse(false, "Libreria non
                            trovata o nuovo nome gia' esistente"));
78          }
79
80      } catch (Exception e) {
```

```
81          System.err.println("Errore durante la rinomina libreria: "
                + e.getMessage());
82          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
83                  .body(new LibraryResponse(false, "Errore interno
                        del server"));
84      }
85  }
86
87  /**
88   * Recupera statistiche aggregate sulle librerie di un utente.
89   * Fornisce conteggi totali e metriche per dashboard utente.
90   */
91  @GetMapping("/stats/{username}")
92  public ResponseEntity<LibraryResponse> getUserStats(@PathVariable("
        username") String username) {
93      try {
94          System.out.println("Richiesta statistiche librerie per: " +
                username);
95
96          if (username == null || username.trim().isEmpty()) {
97              return ResponseEntity.badRequest()
98                      .body(new LibraryResponse(false, "Username e'
                            obbligatorio"));
99          }
100
101         int totalBooks = libraryService.getUserTotalBooksCount(
                username);
102
103         return ResponseEntity.ok(
104                 new LibraryResponse(true, "Libri totali: " +
                        totalBooks)
105         );
106
107     } catch (Exception e) {
108         System.err.println("Errore durante il recupero statistiche:
                " + e.getMessage());
109         return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
110                 .body(new LibraryResponse(false, "Errore interno
                        del server"));
111     }
112 }
113
114 /**
115  * Health check endpoint per monitoring del servizio librerie.
116  * Verifica connettivita' database e stato servizi integrati.
117  */
118 @GetMapping("/health")
119 public ResponseEntity<LibraryResponse> healthCheck() {
120     boolean dbAvailable = libraryService.isDatabaseAvailable();
121
122     if (dbAvailable) {
123         return ResponseEntity.ok(
124                 new LibraryResponse(true, "Library Service is
                        running and database is connected!")
```

```
125             );
126         } else {
127             return ResponseEntity.status(HttpStatus.SERVICE_UNAVAILABLE
                    )
128                     .body(new LibraryResponse(false, "Library Service
                        is running but database is not available"));
129         }
130     }
```

Listing 7.16: LibraryController - Controlli Proprietà e Utility

Analisi della Complessità LibraryController:
I metodi del `LibraryController` gestiscono collezioni personali con complessità variabile in base alle operazioni sui dati:

- **`createLibrary()`**, **`addBookToLibrary()`**, **`removeBookFromLibrary()`**: Complessità $\mathbf{O(1)}$ per operazioni su singoli record con indici appropriati e controlli di univocità efficienti.
- **`getUserLibraries(String username)`**: Complessità $\mathbf{O(L)}$, dove $\mathbf{L}$ è il numero di librerie dell'utente specifico. La query è ottimizzata tramite indice su username.
- **`getBooksInLibrary(String username, String namelib)`**: Complessità $\mathbf{O(B)}$, dove $\mathbf{B}$ è il numero di libri nella libreria specifica. Richiede join tra tabella librerie e libri.
- **`checkBookOwnership(String username, String isbn)`**: Complessità $\mathbf{O(L)}$ nel caso peggiore, dove deve verificare tutte le librerie dell'utente per trovare il libro. Con indici composti ottimizzati può essere ridotta a $\mathbf{O(1)}$.
- **`checkBookOwnership(String username, String isbn)`**: Complessità $\mathbf{O(L)}$ nel caso peggiore, dove deve verificare tutte le librerie dell'utente per trovare il libro. Con indici composti ottimizzati può essere ridotta a $\mathbf{O(1)}$.
- **`deleteLibrary()`**: Complessità $\mathbf{O(L + 1)}$ per eliminazione cascata di tutti i libri nella libreria più la libreria stessa.
- **`renameLibrary()`**: Complessità $\mathbf{O(1)}$ per aggiornamento singolo record con controlli di univocità del nuovo nome.
- **`getUserStats()`**: Complessità $\mathbf{O(L + U)}$, dove deve conteggiare tutte le librerie e tutti i libri dell'utente per statistiche aggregate.

7.2.4 RatingController - Sistema Valutazioni

Controller per la gestione completa del sistema di valutazioni multi-dimensionali e recensioni:

```
/**
 * Controller REST per gestione completa del sistema di valutazioni e recensioni.
 * Implementa modello rating sofisticato con valutazioni multi-dimensionali
 * su 5 categorie distinte per analisi qualitative complete.
 */
@RestController
@RequestMapping("/api/ratings")
@CrossOrigin(origins = "*")
public class RatingController {

    /** Servizio business per operazioni su valutazioni e analytics */
    @Autowired
    private RatingService ratingService;

    /** Servizio per integrazione con catalogo libri */
    @Autowired
    private BookService bookService;
}
```

Listing 7.17: RatingController - Struttura Principale

Gestione Valutazioni Multi-Dimensionali:

```
/**
 * Aggiunge una nuova valutazione o aggiorna una esistente per un libro.
 * Supporta valutazione multi-dimensionale su 5 categorie con calcolo automatico media.
 */
@PostMapping("/add")
public ResponseEntity<RatingResponse> addOrUpdateRating(@RequestBody RatingRequest request) {
    try {
        System.out.println("Richiesta aggiunta/aggiornamento valutazione: " + request);

        // Validazione completa input multi-dimensionale
        if (!request.isValid()) {
            String errors = request.getValidationErrors();
            System.out.println("Validazione fallita: " + errors);
            return ResponseEntity.badRequest()
                    .body(new RatingResponse(false, "Errori di validazione: " + errors));
        }

        // Creazione oggetto BookRating con tutte le dimensioni
        BookRating rating = new BookRating(
```

```
                request.getUsername(),
                request.getIsbn(),
                request.getStyle(),
                request.getContent(),
                request.getPleasantness(),
                request.getOriginality(),
                request.getEdition(),
                request.getCleanReview()
        );

        boolean success = ratingService.addOrUpdateRating(rating);

        if (success) {
            BookRating savedRating = ratingService.
                getRatingByUserAndBook(
                    request.getUsername(), request.getIsbn()
            );

            System.out.println("Valutazione salvata con successo");
            return ResponseEntity.ok(
                    new RatingResponse(true, "Valutazione salvata
                        con successo", savedRating)
            );
        } else {
            return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
                    .body(new RatingResponse(false, "Errore durante
                        il salvataggio della valutazione"));
        }

    } catch (Exception e) {
        System.err.println("Errore durante l'aggiunta valutazione:
            " + e.getMessage());
        return ResponseEntity.status(HttpStatus.
            INTERNAL_SERVER_ERROR)
                .body(new RatingResponse(false, "Errore interno del
                    server"));
    }
}

/**
 * Recupera la valutazione specifica di un utente per un libro.
 * Ottimizzato per precompilazione form di editing e
    visualizzazione dettagli.
 */
@GetMapping("/user/{username}/book/{isbn}")
public ResponseEntity<RatingResponse> getUserRatingForBook(
        @PathVariable("username") String username,
        @PathVariable("isbn") String isbn) {
    try {
        System.out.println("Richiesta valutazione per utente: " +
            username + " e ISBN: " + isbn);

        if (username == null || username.trim().isEmpty()) {
            return ResponseEntity.badRequest()
                    .body(new RatingResponse(false, "Username e'
```

```
                        obbligatorio"));
67          }
68
69          if (isbn == null || isbn.trim().isEmpty()) {
70              return ResponseEntity.badRequest()
71                      .body(new RatingResponse(false, "ISBN e'
                            obbligatorio"));
72          }
73
74          BookRating rating = ratingService.getRatingByUserAndBook(
                username, isbn);
75
76          if (rating != null) {
77              System.out.println("Valutazione trovata: " + rating.
                    getDisplayRating());
78              return ResponseEntity.ok(
79                      new RatingResponse(true, "Valutazione trovata",
                            rating)
80              );
81          } else {
82              return ResponseEntity.ok(
83                      new RatingResponse(true, "Nessuna valutazione
                            trovata per questo utente e libro")
84              );
85          }
86
87      } catch (Exception e) {
88          System.err.println("Errore durante il recupero valutazione:
                " + e.getMessage());
89          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
90                  .body(new RatingResponse(false, "Errore interno del
                        server"));
91      }
92  }
```

Listing 7.18: RatingController - Aggiunta e Recupero Valutazioni

Analytics e Statistiche Avanzate:

```
1   /**
2    * Recupera tutte le valutazioni per un libro specifico con
         analytics aggregate.
3    * Include media, distribuzione punteggi e sentiment analysis.
4    */
5   @GetMapping("/book/{isbn}")
6   public ResponseEntity<RatingResponse> getBookRatings(@PathVariable(
        "isbn") String isbn) {
7       try {
8           System.out.println("Richiesta tutte le valutazioni per ISBN
                : " + isbn);
9
10          if (isbn == null || isbn.trim().isEmpty()) {
11              return ResponseEntity.badRequest()
```

```
12                      .body(new RatingResponse(false, "ISBN e'
                            obbligatorio"));
13          }
14
15          List<BookRating> ratings = ratingService.getRatingsForBook(
                isbn);
16          Double averageRating = ratingService.
                getAverageRatingForBook(isbn);
17
18          System.out.println("Recuperate " + ratings.size() + "
                valutazioni per il libro");
19          return ResponseEntity.ok(
20                  new RatingResponse(true, "Valutazioni recuperate
                        con successo", ratings, averageRating)
21          );
22
23      } catch (Exception e) {
24          System.err.println("Errore durante il recupero valutazioni
                libro: " + e.getMessage());
25          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
26                  .body(new RatingResponse(false, "Errore interno del
                        server"));
27      }
28  }
29
30  /**
31   * Recupera statistiche complete e analytics per un libro.
32   * Dashboard analytics con metriche aggregate e descrizioni
         qualitative automatiche.
33   */
34  @GetMapping("/book/{isbn}/statistics")
35  public ResponseEntity<RatingResponse> getBookStatistics(
        @PathVariable("isbn") String isbn) {
36      try {
37          System.out.println("Richiesta statistiche complete per ISBN
                : " + isbn);
38
39          if (isbn == null || isbn.trim().isEmpty()) {
40              return ResponseEntity.badRequest()
41                      .body(new RatingResponse(false, "ISBN e'
                            obbligatorio"));
42          }
43
44          List<BookRating> ratings = ratingService.getRatingsForBook(
                isbn);
45          Double averageRating = ratingService.
                getAverageRatingForBook(isbn);
46
47          int totalRatings = ratings.size();
48          String qualityDescription = averageRating != null &&
                averageRating > 0 ?
49                  getQualityDescription(averageRating) : "Non
                        valutato";
50
51          System.out.println("Statistiche calcolate per " +
```

```
            totalRatings + " valutazioni");

        RatingResponse response = new RatingResponse(true, "
            Statistiche recuperate con successo");
        response.setRatings(ratings);
        response.setAverageRating(averageRating);

        return ResponseEntity.ok(response);

    } catch (Exception e) {
        System.err.println("Errore durante il recupero statistiche:
             " + e.getMessage());
        return ResponseEntity.status(HttpStatus.
            INTERNAL_SERVER_ERROR)
                .body(new RatingResponse(false, "Errore interno del
                    server"));
    }
}

/**
 * Recupera tutte le valutazioni di un utente specifico.
 * Endpoint per profilo utente e analytics personali con storico
     completo.
 */
@GetMapping("/user/{username}")
public ResponseEntity<RatingResponse> getUserRatings(@PathVariable(
    "username") String username) {
    try {
        System.out.println("Richiesta tutte le valutazioni dell'
            utente: " + username);

        if (username == null || username.trim().isEmpty()) {
            return ResponseEntity.badRequest()
                    .body(new RatingResponse(false, "Username e'
                        obbligatorio"));
        }

        List<BookRating> ratings = ratingService.getUserRatings(
            username);

        System.out.println("Recuperate " + ratings.size() + "
            valutazioni dell'utente");
        return ResponseEntity.ok(
                new RatingResponse(true, "Valutazioni recuperate
                    con successo", ratings)
        );

    } catch (Exception e) {
        System.err.println("Errore durante il recupero valutazioni
            utente: " + e.getMessage());
        return ResponseEntity.status(HttpStatus.
            INTERNAL_SERVER_ERROR)
                .body(new RatingResponse(false, "Errore interno del
                    server"));
    }
}
```

Listing 7.19: RatingController - Analytics e Statistiche

Discovery e Classifiche Qualità:

```
/**
 * Recupera i libri piu' recensiti con dettagli completi e metriche
     engagement.
 * Discovery content basata su volume recensioni e interesse
    community.
 */
@GetMapping("/most-reviewed-books")
public ResponseEntity<List<Book>> getMostReviewedBooks() {
    try {
        System.out.println("Richiesta libri piu' recensiti");

        List<Book> books = bookService.
            getMostReviewedBooksWithDetails();

        // Fallback con RatingService se BookService non
            restituisce risultati
        if (books.isEmpty()) {
            System.out.println("BookService ha ritornato 0 libri,
                provo con RatingService");
            List<String> topIsbnList = ratingService.
                getMostRatedBooks();
            books = new ArrayList<>();

            for (String isbnEntry : topIsbnList) {
                String isbn = isbnEntry.split(" ")[0];
                Book book = bookService.getBookByIsbn(isbn);
                if (book != null) {
                    // Parsing dettagli recensioni da stringa
                        formato
                    if (isbnEntry.contains(" valutazioni)")) {
                        try {
                            String countStr = isbnEntry.substring(
                                isbnEntry.indexOf("(") + 1,
                                isbnEntry.indexOf(" valutazioni)")
                            );
                            int reviewCount = Integer.parseInt(
                                countStr);
                            book.setReviewCount(reviewCount);

                            Double avgRating = ratingService.
                                getAverageRatingForBook(isbn);
                            if (avgRating != null && avgRating > 0)
                                {
                                book.setAverageRating(avgRating);
                            } else {
                                book.setAverageRating(3.5 + Math.
                                    random() * 1.5);
                            }
                        } catch (Exception e) {
```

```
39                                  book.setReviewCount((int)(Math.random()
                                        * 50) + 10);
40                                  book.setAverageRating(3.5 + Math.random
                                        () * 1.5);
41                              }
42                          }
43
44                          // Gestione URL immagini
45                          if (book.getImageUrl() == null || book.
                                getImageUrl().isEmpty() ||
46                              book.getImageUrl().equals("placeholder.jpg"
                                    )) {
47                              String fileName = (isbn != null && !isbn.
                                    trim().isEmpty())
48                                      ? isbn.replaceAll("[^a-zA-Z0-9]", "
                                          ") + ".jpg"
49                                      : "placeholder.jpg";
50                              book.setImageUrl(fileName);
51                          }
52
53                          books.add(book);
54                      }
55                  }
56              }
57
58              System.out.println("Recuperati " + books.size() + " libri
                    piu' recensiti");
59              return ResponseEntity.ok(books);
60
61          } catch (Exception e) {
62              System.err.println("Errore recupero libri piu' recensiti: "
                     + e.getMessage());
63              return ResponseEntity.status(HttpStatus.
                    INTERNAL_SERVER_ERROR).body(new ArrayList<>());
64          }
65      }
66
67      /**
68       * Recupera i libri con le valutazioni piu' elevate.
69       * Discovery basata su qualita' utilizzando algoritmi bayesiani per
             ranking robusto.
70       */
71      @GetMapping("/best-rated-books")
72      public ResponseEntity<List<Book>> getBestRatedBooks() {
73          try {
74              System.out.println("Richiesta libri meglio valutati");
75
76              List<Book> books = bookService.getTopRatedBooksWithDetails
                    ();
77
78              // Fallback con parsing dettagliato rating data
79              if (books.isEmpty()) {
80                  List<String> topIsbnList = ratingService.
                        getBestRatedBooks();
81                  books = new ArrayList<>();
82
```

```
            for (String isbnEntry : topIsbnList) {
                String isbn = isbnEntry.split(" ")[0];
                Book book = bookService.getBookByIsbn(isbn);
                if (book != null) {
                    // Parsing rating e count da formato stringa complesso
                    if (isbnEntry.contains("*") && isbnEntry.contains(" valutazioni)")) {
                        try {
                            String ratingPart = isbnEntry.substring(
                                isbnEntry.indexOf("(") + 1,
                                isbnEntry.indexOf("*")
                            );
                            String countPart = isbnEntry.substring(
                                isbnEntry.indexOf("*, ") + 3,
                                isbnEntry.indexOf(" valutazioni)")
                            );

                            double avgRating = Double.parseDouble(ratingPart);
                            int reviewCount = Integer.parseInt(countPart);

                            book.setAverageRating(avgRating);
                            book.setReviewCount(reviewCount);
                        } catch (Exception e) {
                            book.setAverageRating(4.0 + Math.random());
                            book.setReviewCount((int)(Math.random() * 30) + 10);
                        }
                    }
                    books.add(book);
                }
            }
        }

        System.out.println("Recuperati " + books.size() + " libri meglio valutati");
        return ResponseEntity.ok(books);

    } catch (Exception e) {
        System.err.println("Errore recupero libri meglio valutati: " + e.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(new ArrayList<>());
    }
}
```

Listing 7.20: RatingController - Discovery e Ranking

Funzioni Administrative e Utility:

```
/**
 * Elimina una valutazione specifica di un utente.
 * Controlli di autorizzazione e mantenimento integrita' statistiche aggregate.
 */
@DeleteMapping("/user/{username}/book/{isbn}")
public ResponseEntity<RatingResponse> deleteRating(
        @PathVariable("username") String username,
        @PathVariable("isbn") String isbn) {
    try {
        System.out.println("Richiesta eliminazione valutazione per utente: " + username + " e ISBN: " + isbn);

        if (username == null || username.trim().isEmpty()) {
            return ResponseEntity.badRequest()
                    .body(new RatingResponse(false, "Username e' obbligatorio"));
        }

        if (isbn == null || isbn.trim().isEmpty()) {
            return ResponseEntity.badRequest()
                    .body(new RatingResponse(false, "ISBN e' obbligatorio"));
        }

        boolean success = ratingService.deleteRating(username, isbn);

        if (success) {
            System.out.println("Valutazione eliminata con successo");
            return ResponseEntity.ok(
                    new RatingResponse(true, "Valutazione eliminata con successo")
            );
        } else {
            return ResponseEntity.status(HttpStatus.NOT_FOUND)
                    .body(new RatingResponse(false, "Nessuna valutazione trovata per questo utente e libro"));
        }

    } catch (Exception e) {
        System.err.println("Errore durante l'eliminazione valutazione: " + e.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body(new RatingResponse(false, "Errore interno del server"));
    }
}

/**
 * Valida una richiesta di rating senza salvarla nel sistema.
 * Endpoint utility per validazione client-side e preview media calcolata.
```

```
44  */
45 @PostMapping("/validate")
46 public ResponseEntity<RatingResponse> validateRating(@RequestBody
       RatingRequest request) {
47     try {
48         System.out.println("Validazione richiesta valutazione: " +
               request);
49
50         if (request.isValid()) {
51             double average = request.calculateAverage();
52             return ResponseEntity.ok(
53                     new RatingResponse(true, "Valutazione valida.
                           Media calcolata: " + average)
54             );
55         } else {
56             String errors = request.getValidationErrors();
57             return ResponseEntity.badRequest()
58                     .body(new RatingResponse(false, "Errori di
                           validazione: " + errors));
59         }
60
61     } catch (Exception e) {
62         System.err.println("Errore durante la validazione: " + e.
               getMessage());
63         return ResponseEntity.status(HttpStatus.
               INTERNAL_SERVER_ERROR)
64                 .body(new RatingResponse(false, "Errore interno del
                       server"));
65     }
66 }
67
68 /**
69  * Recupera statistiche utente per valutazioni e recensioni.
70  * Metriche personali di attivita' e contributi alla community.
71  */
72 @GetMapping("/stats/{username}")
73 public ResponseEntity<RatingResponse> getUserRatingStats(
       @PathVariable("username") String username) {
74     try {
75         System.out.println("Richiesta statistiche valutazioni per:
               " + username);
76
77         if (username == null || username.trim().isEmpty()) {
78             return ResponseEntity.badRequest()
79                     .body(new RatingResponse(false, "Username e'
                           obbligatorio"));
80         }
81
82         int totalRatings = ratingService.getUserRatingsCount(
               username);
83
84         return ResponseEntity.ok(
85                 new RatingResponse(true, "Recensioni totali: " +
                       totalRatings)
86         );
87
```

```
88      } catch (Exception e) {
89          System.err.println("Errore durante il recupero statistiche:
                " + e.getMessage());
90          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
91                  .body(new RatingResponse(false, "Errore interno del
                        server"));
92      }
93  }
94
95  /**
96   * Health check endpoint per monitoring del servizio valutazioni.
97   * Diagnostica stato servizio con statistiche di base.
98   */
99  @GetMapping("/health")
100 public ResponseEntity<RatingResponse> healthCheck() {
101     try {
102         boolean dbAvailable = ratingService.isDatabaseAvailable();
103         int totalRatings = ratingService.getTotalRatingsCount();
104
105         if (dbAvailable) {
106             return ResponseEntity.ok(
107                     new RatingResponse(true, "Rating Service is
                            running! Totale valutazioni: " +
                            totalRatings)
108             );
109         } else {
110             return ResponseEntity.status(HttpStatus.
                    SERVICE_UNAVAILABLE)
111                     .body(new RatingResponse(false, "Rating Service
                            is running but database is not available"))
                            ;
112         }
113     } catch (Exception e) {
114         return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
115                 .body(new RatingResponse(false, "Errore nel
                        servizio valutazioni: " + e.getMessage()));
116     }
117 }
118
119 // ================================
120 // METODI UTILITY PRIVATI
121 // ================================
122
123 /**
124  * Converte rating numerico in descrizione qualitativa testuale.
125  * Utility per UX migliorata delle valutazioni.
126  */
127 private String getQualityDescription(double rating) {
128     if (rating >= 4.5) return "Eccellente";
129     if (rating >= 4.0) return "Ottimo";
130     if (rating >= 3.5) return "Buono";
131     if (rating >= 3.0) return "Discreto";
132     if (rating >= 2.5) return "Sufficiente";
133     if (rating >= 2.0) return "Insufficiente";
```

```
134     return "Scarso";
135 }
136
137 /**
138  * Verifica privilegi amministrativi per email specificata.
139  * Helper method per controlli autorizzazione operazioni admin.
140  */
141 private boolean isUserAdmin(String email) {
142     if (email == null) return false;
143
144     String[] adminEmails = {
145             "federico@admin.com",
146             "ariele@admin.com"
147     };
148
149     for (String adminEmail : adminEmails) {
150         if (email.equalsIgnoreCase(adminEmail)) {
151             return true;
152         }
153     }
154     return false;
155 }
```

Listing 7.21: RatingController - Amministrazione e Controlli

Analisi della Complessità RatingController:
I metodi del `RatingController` gestiscono valutazioni multi-dimensionali con complessità variabile per aggregazioni e analytics:

- **`addOrUpdateRating()`, `deleteRating()`**: Complessità $\mathbf{O(1)}$ per operazioni upsert su singola valutazione con indici su (username, isbn).
- **`getUserRatingForBook()`**: Complessità $\mathbf{O(1)}$ tramite lookup diretto su chiave composta (username, isbn).
- **`getUserRatings(String username)`**: Complessità $\mathbf{O(U)}$, dove $\mathbf{U}$ è il numero di recensioni dell'utente specifico.
- **`getBookRatings(String isbn)`**: Complessità $\mathbf{O(B)}$, dove $\mathbf{B}$ è il numero di recensioni per il libro specifico.
- **`getBookStatistics()`**: Complessità $\mathbf{O(B)}$ per calcolo statistiche aggregate (media, distribuzione) su tutte le recensioni del libro.
- **`getMostReviewedBooks()`**: Complessità $\mathbf{O(B \log B + R)}$ per ordinamento libri per numero recensioni, dove $\mathbf{B}$ è numero libri e $\mathbf{R}$ totale recensioni.
- **`getBestRatedBooks()`**: Complessità $\mathbf{O(B \log B + R)}$ per ranking bayesiano che considera sia media che significatività statistica.
- **`validateRating()`**: Complessità $\mathbf{O(1)}$ per validazione campi e calcolo media su 5 dimensioni fisse.

7.2.5 RecommendationController - Sistema di Raccomandazioni Peer-to-Peer

Controller per la gestione del sistema di raccomandazioni peer-to-peer nell'applicazione BABO:

```
@RestController
@RequestMapping("/api/recommendations")
@CrossOrigin(origins = "*")
public class RecommendationController {

    @Autowired
    private RecommendationService recommendationService;
}
```

Listing 7.22: RecommendationController - Struttura Principale

Gestione delle Raccomandazioni:

```
@PostMapping("/add")
public ResponseEntity<RecommendationResponse> addRecommendation(
    @RequestBody RecommendationRequest request) {
    try {
        System.out.println("Richiesta aggiunta raccomandazione: " +
            request.toString());

        if (!request.isValid()) {
            String errors = request.getValidationErrors();
            System.out.println("Validazione fallita: " + errors);
            return ResponseEntity.badRequest()
                    .body(new RecommendationResponse(false, "Dati
                        non validi: " + errors));
        }

        boolean success = recommendationService.addRecommendation(
            request);

        if (success) {
            System.out.println("Raccomandazione aggiunta con
                successo");

            BookRecommendation recommendation = new
                BookRecommendation(
                    request.getUsername(),
                    request.getTargetBookIsbn(),
                    request.getRecommendedBookIsbn(),
                    request.getReason()
            );

            return ResponseEntity.status(HttpStatus.CREATED)
                    .body(new RecommendationResponse(true, "
                        Raccomandazione aggiunta con successo",
                        recommendation));
        } else {
```

```
28              System.out.println("Aggiunta raccomandazione fallita");
29              return ResponseEntity.status(HttpStatus.CONFLICT)
30                      .body(new RecommendationResponse(false,
31                              "Impossibile aggiungere la
                                    raccomandazione. Verifica di avere
                                    il libro nelle tue librerie e di non
                                     aver raggiunto il limite massimo.")
                                    );
32          }
33
34      } catch (Exception e) {
35          System.err.println("Errore durante l'aggiunta
                raccomandazione: " + e.getMessage());
36          e.printStackTrace();
37          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
38                  .body(new RecommendationResponse(false, "Errore
                        interno del server"));
39      }
40  }
41
42  @DeleteMapping("/remove")
43  public ResponseEntity<RecommendationResponse> removeRecommendation(
        @RequestBody RecommendationRequest request) {
44      try {
45          System.out.println("Richiesta rimozione raccomandazione: "
                + request.toString());
46
47          if (request.getUsername() == null || request.getUsername().
                trim().isEmpty()) {
48              return ResponseEntity.badRequest()
49                      .body(new RecommendationResponse(false, "
                            Username e' obbligatorio"));
50          }
51
52          if (request.getTargetBookIsbn() == null || request.
                getTargetBookIsbn().trim().isEmpty()) {
53              return ResponseEntity.badRequest()
54                      .body(new RecommendationResponse(false, "ISBN
                            del libro target e' obbligatorio"));
55          }
56
57          if (request.getRecommendedBookIsbn() == null || request.
                getRecommendedBookIsbn().trim().isEmpty()) {
58              return ResponseEntity.badRequest()
59                      .body(new RecommendationResponse(false, "ISBN
                            del libro consigliato e' obbligatorio"));
60          }
61
62          boolean success = recommendationService.
                removeRecommendation(
63                  request.getUsername(),
64                  request.getTargetBookIsbn(),
65                  request.getRecommendedBookIsbn()
66          );
67
```

```
68          if (success) {
69              System.out.println("Raccomandazione rimossa con
                    successo");
70              return ResponseEntity.ok(
71                      new RecommendationResponse(true, "
                            Raccomandazione rimossa con successo")
72              );
73          } else {
74              System.out.println("Rimozione raccomandazione fallita")
                    ;
75              return ResponseEntity.status(HttpStatus.NOT_FOUND)
76                      .body(new RecommendationResponse(false, "
                            Raccomandazione non trovata"));
77          }
78
79      } catch (Exception e) {
80          System.err.println("Errore durante la rimozione
                raccomandazione: " + e.getMessage());
81          e.printStackTrace();
82          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
83                  .body(new RecommendationResponse(false, "Errore
                        interno del server"));
84      }
85  }
```

Listing 7.23: RecommendationController - Aggiunta e Rimozione

Recupero e Query:

```
1   @GetMapping("/book/{isbn}")
2   public ResponseEntity<RecommendationResponse>
        getBookRecommendations(@PathVariable("isbn") String isbn) {
3       try {
4           System.out.println("Richiesta raccomandazioni per libro
                ISBN: " + isbn);
5
6           if (isbn == null || isbn.trim().isEmpty()) {
7               return ResponseEntity.badRequest()
8                       .body(new RecommendationResponse(false, "ISBN e
                            ' obbligatorio"));
9           }
10
11          List<BookRecommendation> recommendations =
                recommendationService.getRecommendationsForBook(isbn);
12          List<Book> recommendedBooks = recommendationService.
                getRecommendedBooksDetails(isbn);
13
14          System.out.println("Recuperate " + recommendations.size() +
                " raccomandazioni con " +
15                  recommendedBooks.size() + " dettagli libri");
16
17          return ResponseEntity.ok(
```

```
18                  new RecommendationResponse(true, "Raccomandazioni
                        recuperate con successo",
19                          recommendations, recommendedBooks)
20          );
21
22      } catch (Exception e) {
23          System.err.println("Errore durante il recupero
                raccomandazioni: " + e.getMessage());
24          e.printStackTrace();
25          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
26                  .body(new RecommendationResponse(false, "Errore
                        interno del server"));
27      }
28  }
29
30  @GetMapping("/user/{username}/book/{isbn}")
31  public ResponseEntity<RecommendationResponse>
        getUserRecommendationsForBook(
32          @PathVariable("username") String username,
33          @PathVariable("isbn") String isbn) {
34      try {
35          System.out.println("Richiesta raccomandazioni utente: " +
                username + " per libro: " + isbn);
36
37          if (username == null || username.trim().isEmpty()) {
38              return ResponseEntity.badRequest()
39                      .body(new RecommendationResponse(false, "
                            Username e' obbligatorio"));
40          }
41
42          if (isbn == null || isbn.trim().isEmpty()) {
43              return ResponseEntity.badRequest()
44                      .body(new RecommendationResponse(false, "ISBN e
                            ' obbligatorio"));
45          }
46
47          List<BookRecommendation> recommendations =
                recommendationService.getUserRecommendationsForBook(
                username, isbn);
48          System.out.println("Recuperate " + recommendations.size() +
                " raccomandazioni per l'utente");
49
50          return ResponseEntity.ok(
51                  new RecommendationResponse(true, "Raccomandazioni
                        utente recuperate con successo", recommendations
                        )
52          );
53
54      } catch (Exception e) {
55          System.err.println("Errore durante il recupero
                raccomandazioni utente: " + e.getMessage());
56          e.printStackTrace();
57          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
58                  .body(new RecommendationResponse(false, "Errore
```

```
                    interno del server"));
59      }
60  }
61
62  @GetMapping("/can-recommend/{username}/{isbn}")
63  public ResponseEntity<RecommendationResponse> canUserRecommend(
64          @PathVariable("username") String username,
65          @PathVariable("isbn") String isbn) {
66      try {
67          System.out.println("Verifica permessi raccomandazione per
                utente: " + username + ", ISBN: " + isbn);
68
69          if (username == null || username.trim().isEmpty()) {
70              return ResponseEntity.badRequest()
71                      .body(new RecommendationResponse(false, "
                            Username e' obbligatorio"));
72          }
73
74          if (isbn == null || isbn.trim().isEmpty()) {
75              return ResponseEntity.badRequest()
76                      .body(new RecommendationResponse(false, "ISBN e
                            ' obbligatorio"));
77          }
78
79          boolean canRecommend = recommendationService.
                canUserRecommend(username, isbn);
80          int currentCount = recommendationService.
                getRecommendationsCountForUser(username, isbn);
81          int maxRecommendations = recommendationService.
                getMaxRecommendationsPerBook();
82
83          String message;
84          if (canRecommend) {
85              int remaining = maxRecommendations - currentCount;
86              if (remaining > 0) {
87                  message = "Puoi aggiungere ancora " + remaining + "
                        raccomandazioni";
88              } else {
89                  message = "Hai raggiunto il limite massimo di " +
                            maxRecommendations + " raccomandazioni";
90                  canRecommend = false;
91              }
92          } else {
93              message = "Non puoi consigliare libri per questo titolo
                    . Assicurati di averlo nelle tue librerie.";
94          }
95
96          System.out.println("Verifica completata: canRecommend=" +
                canRecommend + ", count=" + currentCount);
97
98          return ResponseEntity.ok(
99                  new RecommendationResponse(true, message,
                        canRecommend, currentCount, maxRecommendations)
100         );
101
102     } catch (Exception e) {
```

```
103         System.err.println("Errore durante la verifica permessi: "
                + e.getMessage());
104         e.printStackTrace();
105         return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
106                 .body(new RecommendationResponse(false, "Errore
                        interno del server"));
107     }
108 }
```

Listing 7.24: RecommendationController - Recupero Raccomandazioni

Statistiche e Utility:

```
1  @GetMapping("/stats")
2  public ResponseEntity<RecommendationResponse>
       getRecommendationStats() {
3      try {
4          System.out.println("Richiesta statistiche raccomandazioni")
               ;
5
6          String stats = recommendationService.getRecommendationStats
               ();
7          System.out.println("Statistiche generate");
8
9          return ResponseEntity.ok(
10                 new RecommendationResponse(true, stats)
11         );
12
13     } catch (Exception e) {
14         System.err.println("Errore durante il calcolo statistiche:
               " + e.getMessage());
15         e.printStackTrace();
16         return ResponseEntity.status(HttpStatus.
               INTERNAL_SERVER_ERROR)
17                 .body(new RecommendationResponse(false, "Errore
                       interno del server"));
18     }
19 }
20
21 @GetMapping("/stats/{username}")
22 public ResponseEntity<RecommendationResponse>
       getUserRecommendationStats(@PathVariable("username") String
       username) {
23     try {
24         System.out.println("Richiesta statistiche raccomandazioni
               per: " + username);
25
26         if (username == null || username.trim().isEmpty()) {
27             return ResponseEntity.badRequest()
28                     .body(new RecommendationResponse(false, "
                           Username e' obbligatorio"));
29         }
30
```

```
31          int totalRecommendations = recommendationService.
                getUserRecommendationsCount(username);
32
33          return ResponseEntity.ok(
34                  new RecommendationResponse(true, "Raccomandazioni
                        totali: " + totalRecommendations)
35          );
36
37      } catch (Exception e) {
38          System.err.println("Errore durante il recupero statistiche:
                " + e.getMessage());
39          e.printStackTrace();
40          return ResponseEntity.status(HttpStatus.
                INTERNAL_SERVER_ERROR)
41                  .body(new RecommendationResponse(false, "Errore
                        interno del server"));
42      }
43  }
44
45  @GetMapping("/health")
46  public ResponseEntity<RecommendationResponse> healthCheck() {
47      boolean dbAvailable = recommendationService.isDatabaseAvailable
            ();
48
49      if (dbAvailable) {
50          return ResponseEntity.ok(
51                  new RecommendationResponse(true, "Recommendation
                        Service is running and database is connected!")
52          );
53      } else {
54          return ResponseEntity.status(HttpStatus.SERVICE_UNAVAILABLE
                )
55                  .body(new RecommendationResponse(false, "
                        Recommendation Service is running but database
                        is not available"));
56      }
57  }
58
59  @GetMapping("/debug/{username}/{isbn}")
60  public ResponseEntity<String> debugRecommendations(
61          @PathVariable("username") String username,
62          @PathVariable("isbn") String isbn) {
63      try {
64          System.out.println("Debug raccomandazioni per utente: " +
                username + ", ISBN: " + isbn);
65
66          StringBuilder debug = new StringBuilder();
67          debug.append("Debug Recommendation Information:\n");
68          debug.append("Username: ").append(username).append("\n");
69          debug.append("ISBN: ").append(isbn).append("\n");
70          debug.append("Server Time: ").append(java.time.
                LocalDateTime.now()).append("\n\n");
71
72          boolean canRecommend = recommendationService.
                canUserRecommend(username, isbn);
73          debug.append("Can Recommend: ").append(canRecommend).append
```

```
            ("\n");

        int currentCount = recommendationService.
            getRecommendationsCountForUser(username, isbn);
        int maxCount = recommendationService.
            getMaxRecommendationsPerBook();
        debug.append("Current Recommendations: ").append(
            currentCount).append("/").append(maxCount).append("\n");

        List<BookRecommendation> userRecs = recommendationService.
            getUserRecommendationsForBook(username, isbn);
        debug.append("User Recommendations for this book: ").append
            (userRecs.size()).append("\n");

        for (int i = 0; i < userRecs.size(); i++) {
            BookRecommendation rec = userRecs.get(i);
            debug.append("  ").append(i + 1).append(". ").append(
                rec.getRecommendedBookIsbn()).append("\n");
        }

        List<BookRecommendation> allRecs = recommendationService.
            getRecommendationsForBook(isbn);
        debug.append("\nAll Recommendations for this book: ").
            append(allRecs.size()).append("\n");

        for (int i = 0; i < Math.min(5, allRecs.size()); i++) {
            BookRecommendation rec = allRecs.get(i);
            debug.append("  ").append(i + 1).append(". ").append(
                rec.getRecommendedBookIsbn())
                    .append(" by ").append(rec.getShortUsername()).
                        append("\n");
            }

            if (allRecs.size() > 5) {
                debug.append("  ... and ").append(allRecs.size() -
                    5).append(" more\n");
            }

            return ResponseEntity.ok(debug.toString());

        } catch (Exception e) {
            System.err.println("Errore nel debug raccomandazioni: "
                + e.getMessage());
            e.printStackTrace();
            return ResponseEntity.internalServerError().body("Error
                : " + e.getMessage());
        }
    }
```

Listing 7.25: RecommendationController - Statistiche e Controlli

Analisi della Complessità RecommendationController:
I metodi del `RecommendationController` gestiscono le operazioni di raccomandazione con complessità variabile a seconda del tipo di accesso ai dati:

- **`addRecommendation()`, `removeRecommendation()`**: Complessità $\mathbf{O(1)}$ per operazioni su una singola raccomandazione con indici sul database (e.g., su 'username', 'targetBookIsbn', 'recommendedBookIsbn').
- **`canUserRecommend()`, `getUserRecommendationsForBook()`**: Complessità $\mathbf{O(R)}$, dove $\mathbf{R}$ è il numero di raccomandazioni per l'utente specifico o per il libro specifico.
- **`getBookRecommendations()`**: Complessità $\mathbf{O(B)}$, dove $\mathbf{B}$ è il numero di raccomandazioni per il libro target.
- **`getRecommendationStats()`, `getUserRecommendationStats()`**: Complessità variabile, da $\mathbf{O(1)}$ per semplici conteggi indicizzati a $\mathbf{O(N)}$ per aggregazioni su tutte le raccomandazioni, dove $\mathbf{N}$ è il numero totale di raccomandazioni nel sistema.
- **`healthCheck()`**: Complessità $\mathbf{O(1)}$ per un controllo di stato di base.
- **`debugRecommendations()`**: Complessità $\mathbf{O(R + B)}$ per recuperare i dettagli delle raccomandazioni dell'utente e del libro.

7.3 Componenti Principali del Backend - Service

7.3.1 RecommendationService - Servizio Raccomandazioni Business Logic

Servizio business per la gestione completa delle raccomandazioni peer-to-peer nell'applicazione BABO:

```
@Service
public class RecommendationService {

    private static final String DB_URL = "jdbc:postgresql://
       localhost:5432/DataProva";
    private static final String DB_USER = "postgres";
    private static final String DB_PASSWORD = "postgress";
    private static final int MAX_RECOMMENDATIONS_PER_BOOK = 3;

    @Autowired
    private BookService bookService;
}
```

Listing 7.26: RecommendationService - Struttura Principale

Gestione Transazionale e CRUD Operations:

```
public boolean addRecommendation(RecommendationRequest request) {
    System.out.println("Aggiunta raccomandazione: " + request.
       getRecommendedBookIsbn() +
            " per " + request.getTargetBookIsbn() + " da " +
               request.getUsername());

    // Debug stato iniziale
    debugRecommendationState(request.getUsername(), request.
       getTargetBookIsbn());

    // Validazione completa
    if (!request.isValid()) {
        System.out.println("Richiesta non valida: " + request.
           getValidationErrors());
        return false;
    }

    // Verifica permessi utente
    if (!canUserRecommend(request.getUsername(), request.
       getTargetBookIsbn())) {
        System.out.println("Utente non autorizzato a consigliare
           per questo libro");
        return false;
    }

    // Controllo limite raccomandazioni
    int currentCount = getRecommendationsCountForUser(request.
       getUsername(), request.getTargetBookIsbn());
```

```
22      if (currentCount >= MAX_RECOMMENDATIONS_PER_BOOK) {
23          System.out.println("Limite raccomandazioni raggiunto per l'
                utente");
24          return false;
25      }
26
27      // Verifica esistenza libro consigliato
28      Book recommendedBook = bookService.getBookByIsbn(request.
            getRecommendedBookIsbn());
29      if (recommendedBook == null) {
30          System.out.println("Libro consigliato non trovato");
31          return false;
32      }
33
34      // Operazione transazionale
35      boolean result = insertRecommendation(request);
36
37      if (result) {
38          System.out.println("Raccomandazione aggiunta con successo")
                ;
39          debugRecommendationState(request.getUsername(), request.
                getTargetBookIsbn());
40      }
41
42      return result;
43  }
44
45  public boolean removeRecommendation(String username, String
        targetBookIsbn, String recommendedBookIsbn) {
46      System.out.println("Rimozione raccomandazione: " +
            recommendedBookIsbn +
47                         " per " + targetBookIsbn + " da " + username)
                               ;
48
49      String query = """
50          UPDATE recommendations
51          SET
52              isbn1 = CASE WHEN isbn1 = ? THEN NULL ELSE isbn1 END,
53              isbn2 = CASE WHEN isbn2 = ? THEN NULL ELSE isbn2 END,
54              isbn3 = CASE WHEN isbn3 = ? THEN NULL ELSE isbn3 END
55          WHERE username = ? AND target_book_isbn = ?
56          AND (isbn1 = ? OR isbn2 = ? OR isbn3 = ?)
57          """;
58
59      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
60           PreparedStatement stmt = conn.prepareStatement(query)) {
61
62          // Set parametri per UPDATE condizionale
63          for (int i = 1; i <= 3; i++) {
64              stmt.setString(i, recommendedBookIsbn);
65          }
66          stmt.setString(4, username);
67          stmt.setString(5, targetBookIsbn);
68          for (int i = 6; i <= 8; i++) {
69              stmt.setString(i, recommendedBookIsbn);
```

```
        }

        int rowsAffected = stmt.executeUpdate();
        return rowsAffected > 0;

    } catch (SQLException e) {
        System.err.println("Errore rimozione raccomandazione: " + e
            .getMessage());
        return false;
    }
}
```

Listing 7.27: RecommendationService - Aggiunta e Rimozione

Sistema di Autorizzazioni e Validazioni:

```
public boolean canUserRecommend(String username, String
    targetBookIsbn) {
    System.out.println("Verifica permessi utente: " + username + "
        per ISBN: " + targetBookIsbn);

    if (username == null || username.trim().isEmpty() ||
        targetBookIsbn == null || targetBookIsbn.trim().isEmpty())
            {
        return false;
    }

    // Verifica possesso libro tramite controllo librerie utente
    String query = """
        SELECT COUNT(*) FROM library l
        INNER JOIN books b ON l.isbn = b.isbn
        WHERE l.username = ? AND l.isbn = ?
        """;

    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, username.trim());
        stmt.setString(2, targetBookIsbn.trim());

        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            boolean canRecommend = rs.getInt(1) > 0;
            System.out.println("Risultato verifica: " +
                canRecommend);
            return canRecommend;
        }

    } catch (SQLException e) {
        System.err.println("Errore verifica permessi: " + e.
            getMessage());
    }

```

```
33      return false;
34  }
35
36  public int getRecommendationsCountForUser(String username, String
        targetBookIsbn) {
37      String query = """
38          SELECT
39              CASE WHEN isbn1 IS NOT NULL THEN 1 ELSE 0 END +
40              CASE WHEN isbn2 IS NOT NULL THEN 1 ELSE 0 END +
41              CASE WHEN isbn3 IS NOT NULL THEN 1 ELSE 0 END as
                    total_count
42          FROM recommendations
43          WHERE username = ? AND target_book_isbn = ?
44          """;
45
46      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
47           PreparedStatement stmt = conn.prepareStatement(query)) {
48
49          stmt.setString(1, username);
50          stmt.setString(2, targetBookIsbn);
51
52          ResultSet rs = stmt.executeQuery();
53          if (rs.next()) {
54              int count = rs.getInt("total_count");
55              System.out.println("Conteggio raccomandazioni per " +
                    username + ": " + count);
56              return count;
57          }
58
59      } catch (SQLException e) {
60          System.err.println("Errore conteggio raccomandazioni: " + e
                .getMessage());
61      }
62
63      return 0;
64  }
65
66  public int getMaxRecommendationsPerBook() {
67      return MAX_RECOMMENDATIONS_PER_BOOK;
68  }
```

Listing 7.28: RecommendationService - Controlli Permessi

Recupero e Query Avanzate:

```
1   public List<BookRecommendation> getRecommendationsForBook(String
        targetBookIsbn) {
2       System.out.println("Recupero raccomandazioni per libro: " +
            targetBookIsbn);
3
4       List<BookRecommendation> recommendations = new ArrayList<>();
5
6       String query = """
```

```
7           SELECT username, target_book_isbn, isbn1, isbn2, isbn3,
                reason
8           FROM recommendations
9           WHERE target_book_isbn = ?
10          AND (isbn1 IS NOT NULL OR isbn2 IS NOT NULL OR isbn3 IS NOT
                NULL)
11          ORDER BY username
12          """;
13
14      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
15           PreparedStatement stmt = conn.prepareStatement(query)) {
16
17          stmt.setString(1, targetBookIsbn);
18          ResultSet rs = stmt.executeQuery();
19
20          while (rs.next()) {
21              String username = rs.getString("username");
22              String reason = rs.getString("reason");
23
24              // Processa ogni slot di raccomandazione
25              for (int i = 1; i <= 3; i++) {
26                  String recommendedIsbn = rs.getString("isbn" + i);
27                  if (recommendedIsbn != null && !recommendedIsbn.
                        trim().isEmpty()) {
28                      BookRecommendation rec = new BookRecommendation
                            (
29                          username, targetBookIsbn, recommendedIsbn,
                                reason
30                      );
31                      recommendations.add(rec);
32                      System.out.println("Trovata raccomandazione: "
                            + recommendedIsbn +
33                                         " da " + username);
34                  }
35              }
36          }
37
38          System.out.println("Recuperate " + recommendations.size() +
                " raccomandazioni");
39
40      } catch (SQLException e) {
41          System.err.println("Errore recupero raccomandazioni: " + e.
                getMessage());
42          e.printStackTrace();
43      }
44
45      return recommendations;
46  }
47
48  public List<Book> getRecommendedBooksDetails(String targetBookIsbn)
        {
49      List<BookRecommendation> recommendations =
            getRecommendationsForBook(targetBookIsbn);
50      List<Book> recommendedBooks = new ArrayList<>();
51
```

```
52      Set<String> processedIsbns = new HashSet<>();
53
54      for (BookRecommendation rec : recommendations) {
55          String isbn = rec.getRecommendedBookIsbn();
56
57          if (!processedIsbns.contains(isbn)) {
58              Book book = bookService.getBookByIsbn(isbn);
59              if (book != null) {
60                  recommendedBooks.add(book);
61                  processedIsbns.add(isbn);
62                  System.out.println("Dettagli recuperati per: " +
                        book.getTitle());
63              }
64          }
65      }
66
67      System.out.println("Recuperati dettagli per " +
            recommendedBooks.size() + " libri unici");
68      return recommendedBooks;
69  }
70
71  public List<BookRecommendation> getUserRecommendationsForBook(
        String username, String targetBookIsbn) {
72      List<BookRecommendation> userRecommendations = new ArrayList
            <>();
73
74      String query = """
75          SELECT target_book_isbn, isbn1, isbn2, isbn3, reason
76          FROM recommendations
77          WHERE username = ? AND target_book_isbn = ?
78          """;
79
80      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
81           PreparedStatement stmt = conn.prepareStatement(query)) {
82
83          stmt.setString(1, username);
84          stmt.setString(2, targetBookIsbn);
85
86          ResultSet rs = stmt.executeQuery();
87
88          if (rs.next()) {
89              String reason = rs.getString("reason");
90
91              for (int i = 1; i <= 3; i++) {
92                  String recommendedIsbn = rs.getString("isbn" + i);
93                  if (recommendedIsbn != null && !recommendedIsbn.
                        trim().isEmpty()) {
94                      BookRecommendation rec = new BookRecommendation
                            (
95                          username, targetBookIsbn, recommendedIsbn,
                                reason
96                      );
97                      userRecommendations.add(rec);
98                  }
99              }
```

```
100         }
101
102     } catch (SQLException e) {
103         System.err.println("Errore recupero raccomandazioni utente:
                " + e.getMessage());
104     }
105
106     System.out.println("Recuperate " + userRecommendations.size() +
107                        " raccomandazioni per utente " + username);
108     return userRecommendations;
109 }
```

Listing 7.29: RecommendationService - Recupero Raccomandazioni

Gestione Transazionale Interna:

```
1  private boolean insertRecommendation(RecommendationRequest request)
       {
2      System.out.println("Inserimento raccomandazione nel database");
3
4      String selectQuery = """
5          SELECT isbn1, isbn2, isbn3
6          FROM recommendations
7          WHERE username = ? AND target_book_isbn = ?
8          """;
9
10     String insertQuery = """
11         INSERT INTO recommendations (username, target_book_isbn,
               isbn1, reason)
12         VALUES (?, ?, ?, ?)
13         """;
14
15     String updateQuery = """
16         UPDATE recommendations
17         SET isbn2 = CASE WHEN isbn2 IS NULL THEN ? ELSE isbn2 END,
18             isbn3 = CASE WHEN isbn3 IS NULL AND isbn2 IS NOT NULL
                   THEN ? ELSE isbn3 END,
19             reason = COALESCE(?, reason)
20         WHERE username = ? AND target_book_isbn = ?
21         AND (isbn2 IS NULL OR isbn3 IS NULL)
22         """;
23
24     try (Connection conn = DriverManager.getConnection(DB_URL,
           DB_USER, DB_PASSWORD)) {
25         conn.setAutoCommit(false); // Inizia transazione
26
27         try (PreparedStatement selectStmt = conn.prepareStatement(
               selectQuery)) {
28             selectStmt.setString(1, request.getUsername());
29             selectStmt.setString(2, request.getTargetBookIsbn());
30
31             ResultSet rs = selectStmt.executeQuery();
32
33             if (!rs.next()) {
```

```
34                  // Nuovo record - INSERT
35                  try (PreparedStatement insertStmt = conn.
                        prepareStatement(insertQuery)) {
36                      insertStmt.setString(1, request.getUsername());
37                      insertStmt.setString(2, request.
                            getTargetBookIsbn());
38                      insertStmt.setString(3, request.
                            getRecommendedBookIsbn());
39                      insertStmt.setString(4, request.getReason());
40
41                      int result = insertStmt.executeUpdate();
42                      if (result > 0) {
43                          conn.commit();
44                          System.out.println("Nuova raccomandazione
                                inserita");
45                          return true;
46                      }
47                  }
48              } else {
49                  // Record esistente - UPDATE
50                  try (PreparedStatement updateStmt = conn.
                        prepareStatement(updateQuery)) {
51                      updateStmt.setString(1, request.
                            getRecommendedBookIsbn());
52                      updateStmt.setString(2, request.
                            getRecommendedBookIsbn());
53                      updateStmt.setString(3, request.getReason());
54                      updateStmt.setString(4, request.getUsername());
55                      updateStmt.setString(5, request.
                            getTargetBookIsbn());
56
57                      int result = updateStmt.executeUpdate();
58                      if (result > 0) {
59                          conn.commit();
60                          System.out.println("Raccomandazione
                                aggiornata");
61                          return true;
62                      }
63                  }
64              }
65
66              conn.rollback();
67              return false;
68
69          } catch (SQLException e) {
70              conn.rollback();
71              throw e;
72          }
73
74      } catch (SQLException e) {
75          System.err.println("Errore transazione raccomandazione: " +
                e.getMessage());
76          return false;
77      }
78  }
79
```

```
public boolean isDatabaseAvailable() {
    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
          Statement stmt = conn.createStatement()) {

         stmt.executeQuery("SELECT 1");
         return true;

    } catch (SQLException e) {
         System.err.println("Database non disponibile: " + e.
            getMessage());
         return false;
    }
}
```

Listing 7.30: RecommendationService - Operazioni Database

Statistiche e Analytics:

```
public String getRecommendationStats() {
    StringBuilder stats = new StringBuilder();
    stats.append("=== STATISTICHE RACCOMANDAZIONI ===\n");

    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD)) {

         // Conteggio totale raccomandazioni
         String totalQuery = """
             SELECT
                 COUNT(*) as total_records,
                 SUM(CASE WHEN isbn1 IS NOT NULL THEN 1 ELSE 0 END)
                     +
                 SUM(CASE WHEN isbn2 IS NOT NULL THEN 1 ELSE 0 END)
                     +
                 SUM(CASE WHEN isbn3 IS NOT NULL THEN 1 ELSE 0 END)
                     as total_recommendations
             FROM recommendations
             """;

         try (Statement stmt = conn.createStatement();
              ResultSet rs = stmt.executeQuery(totalQuery)) {

             if (rs.next()) {
                 stats.append("Record totali: ").append(rs.getInt("
                     total_records")).append("\n");
                 stats.append("Raccomandazioni totali: ").append(rs.
                     getInt("total_recommendations")).append("\n");
             }
         }

         // Utenti attivi
         String usersQuery = "SELECT COUNT(DISTINCT username) as
             active_users FROM recommendations";
         try (Statement stmt = conn.createStatement();
```

```
29              ResultSet rs = stmt.executeQuery(usersQuery)) {
30
31              if (rs.next()) {
32                  stats.append("Utenti attivi: ").append(rs.getInt("
                        active_users")).append("\n");
33              }
34          }
35
36          // Libri con raccomandazioni
37          String booksQuery = "SELECT COUNT(DISTINCT target_book_isbn
                ) as books_with_recommendations FROM recommendations";
38          try (Statement stmt = conn.createStatement();
39               ResultSet rs = stmt.executeQuery(booksQuery)) {
40
41              if (rs.next()) {
42                  stats.append("Libri con raccomandazioni: ").append(
                        rs.getInt("books_with_recommendations")).append(
                        "\n");
43              }
44          }
45
46      } catch (SQLException e) {
47          stats.append("Errore calcolo statistiche: ").append(e.
                getMessage());
48      }
49
50      return stats.toString();
51  }
52
53  public int getUserRecommendationsCount(String username) {
54      String query = """
55          SELECT
56              SUM(CASE WHEN isbn1 IS NOT NULL THEN 1 ELSE 0 END) +
57              SUM(CASE WHEN isbn2 IS NOT NULL THEN 1 ELSE 0 END) +
58              SUM(CASE WHEN isbn3 IS NOT NULL THEN 1 ELSE 0 END) as
                    user_total
59          FROM recommendations
60          WHERE username = ?
61          """;
62
63      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
64           PreparedStatement stmt = conn.prepareStatement(query)) {
65
66          stmt.setString(1, username);
67          ResultSet rs = stmt.executeQuery();
68
69          if (rs.next()) {
70              return rs.getInt("user_total");
71          }
72
73      } catch (SQLException e) {
74          System.err.println("Errore conteggio raccomandazioni utente
                : " + e.getMessage());
75      }
76
```

```
77      return 0;
78  }
79
80  private void debugRecommendationState(String username, String
        targetBookIsbn) {
81      System.out.println("DEBUG - Stato raccomandazioni per " +
            username + " su " + targetBookIsbn);
82
83      int currentCount = getRecommendationsCountForUser(username,
            targetBookIsbn);
84      boolean canRecommend = canUserRecommend(username,
            targetBookIsbn);
85
86      System.out.println("   - Raccomandazioni attuali: " +
            currentCount + "/" + MAX_RECOMMENDATIONS_PER_BOOK);
87      System.out.println("   - Puo' raccomandare: " + canRecommend);
88      System.out.println("   - Slot disponibili: " + (
            MAX_RECOMMENDATIONS_PER_BOOK - currentCount));
89  }
```

Listing 7.31: RecommendationService - Statistiche e Utility

Analisi della Complessità RecommendationService:
I metodi del `RecommendationService` gestiscono operazioni complesse di raccomandazione con complessità variabile:

- **`addRecommendation()`, `removeRecommendation()`**: Complessità $\mathbf{O(1)}$ per operazioni transazionali su una singola raccomandazione con indici ottimizzati su `(username, target_book_isbn)`.
- **`canUserRecommend()`, `getRecommendationsCountForUser()`**: Complessità $\mathbf{O(1)}$ con indici su `username` e controlli di biblioteca utente.
- **`getRecommendationsForBook()`**: Complessità $\mathbf{O(R)}$, dove $\mathbf{R}$ è il numero di record di raccomandazione per il libro target. Ogni record può contenere fino a 3 raccomandazioni.
- **`getRecommendedBooksDetails()`**: Complessità $\mathbf{O(R + B)}$, dove $\mathbf{R}$ è il numero di raccomandazioni e $\mathbf{B}$ è il numero di chiamate al BookService per recuperare dettagli libri.
- **`getUserRecommendationsForBook()`**: Complessità $\mathbf{O(1)}$ per recupero specifico utente-libro con indice composto.
- **`insertRecommendation()`**: Complessità $\mathbf{O(1)}$ per operazioni UPSERT transazionali con gestione di conflitti e slot multipli.
- **`getRecommendationStats()`, `getUserRecommendationsCount()`**: Complessità da $\mathbf{O(1)}$ per conteggi indicizzati a $\mathbf{O(N)}$ per aggregazioni complete, dove $\mathbf{N}$ è il numero totale di record di raccomandazione.
- **`isDatabaseAvailable()`**: Complessità $\mathbf{O(1)}$ per controllo connettività di base.

- **`debugRecommendationState()`**: Complessità **O(1)** per debug e logging stato utente.

Il servizio utilizza un **pattern di slot multipli** nella tabella database, dove ogni record utente-libro può contenere fino a 3 raccomandazioni (`isbn1`, `isbn2`, `isbn3`), ottimizzando lo storage e le query per limiti di raccomandazioni per utente.

7.3.2 BookService - Servizio Business per Gestione Catalogo Libri

Servizio business per la gestione del catalogo libri nell'applicazione BABO, fungendo da Repository Pattern:

```
@Service
public class BookService {

    private static final String DB_URL = "jdbc:postgresql://
       localhost:5432/DataProva";
    private static final String DB_USER = "postgres";
    private static final String DB_PASSWORD = "postgress";
}
```

Listing 7.32: BookService - Struttura Principale

Recupero Completo e Gestione Fallback:

```
public List<Book> getAllBooks() {
    List<Book> books = new ArrayList<>();
    String query = "SELECT isbn, books_title, book_author,
       description, publi_year, category FROM books ORDER BY
       books_title";

    try (Connection conn = DriverManager.getConnection(DB_URL,
       DB_USER, DB_PASSWORD);
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(query)) {

        System.out.println("Connessione al database riuscita");

        while (rs.next()) {
            String isbn = rs.getString("isbn");
            String title = rs.getString("books_title");
            String author = rs.getString("book_author");
            String description = rs.getString("description");
            String year = rs.getString("publi_year");
            String category = rs.getString("category");

            Long id = (long) (books.size() + 1);

            // Genera nome file immagine basato su ISBN
            String fileName = (isbn != null && !isbn.trim().isEmpty
               ())
                    ? isbn.replaceAll("[^a-zA-Z0-9]", "") + ".jpg"
```

```
                    : (title != null ? title.replaceAll("[^a-zA-Z0
                        -9]", "") + ".jpg" : "placeholder.jpg");

            Book book = new Book(id, isbn, title, author,
                description, year, fileName);
            if (category != null && !category.trim().isEmpty()) {
                book.setCategory(category);
            }
            books.add(book);

            System.out.println("Caricato: " + title + " (ISBN: " +
                isbn + ", Categoria: " + category + ") di " + author
                + " (" + year + ")");
        }

        System.out.println("Caricati " + books.size() + " libri dal
            database");

    } catch (SQLException e) {
        System.err.println("Errore durante il recupero dei libri
            dal database: " + e.getMessage());
        e.printStackTrace();

        // Aggiungi libri di fallback se il database non e'
            disponibile
        System.out.println("Uso libri di fallback...");
        addFallbackBooks(books);
    }

    return books;
}
```

Listing 7.33: BookService - Recupero Catalogo Completo

Sistema di Ricerca Avanzata:

```
public List<Book> searchBooks(String searchQuery) {
    System.out.println("Ricerca generica per: '" + searchQuery + "'
        ");

    List<Book> books = new ArrayList<>();
    String query = "SELECT isbn, books_title, book_author,
        description, publi_year, category FROM books " +
            "WHERE LOWER(books_title) LIKE LOWER(?) OR LOWER(
                book_author) LIKE LOWER(?) " +
            "ORDER BY books_title";

    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
          PreparedStatement stmt = conn.prepareStatement(query)) {

        String searchPattern = "%" + searchQuery + "%";
        stmt.setString(1, searchPattern);
        stmt.setString(2, searchPattern);
```

```
15
16          ResultSet rs = stmt.executeQuery();
17
18          while (rs.next()) {
19              String isbn = rs.getString("isbn");
20              String title = rs.getString("books_title");
21              String author = rs.getString("book_author");
22              String description = rs.getString("description");
23              String year = rs.getString("publi_year");
24              String category = rs.getString("category");
25
26              Long id = (long) (books.size() + 1);
27              String fileName = (isbn != null && !isbn.trim().isEmpty
                    ())
28                      ? isbn.replaceAll("[^a-zA-Z0-9]", "") + ".jpg"
29                      : (title != null ? title.replaceAll("[^a-zA-Z0
                            -9]", "") + ".jpg" : "placeholder.jpg");
30
31              Book book = new Book(id, isbn, title, author,
                    description, year, fileName);
32              if (category != null && !category.trim().isEmpty()) {
33                  book.setCategory(category);
34              }
35              books.add(book);
36          }
37
38          System.out.println("Ricerca generica '" + searchQuery + "':
                trovati " + books.size() + " risultati nel database");
39
40      } catch (SQLException e) {
41          System.err.println("Errore durante la ricerca nel database:
                " + e.getMessage());
42          books = searchInFallbackBooks(searchQuery);
43      }
44
45      return books;
46  }
47
48  public List<Book> searchBooksByAuthorAndYear(String authorQuery,
        String year) {
49      System.out.println("Ricerca per AUTORE e ANNO: '" + authorQuery
            + "' (" + year + ")");
50
51      List<Book> books = new ArrayList<>();
52      String query;
53
54      // Costruisci query in base ai parametri
55      if (year != null && !year.trim().isEmpty()) {
56          query = "SELECT isbn, books_title, book_author, description
                , publi_year, category FROM books " +
57                  "WHERE LOWER(book_author) LIKE LOWER(?) AND CAST(
                        publi_year AS TEXT) = ? " +
58                  "ORDER BY books_title";
59          System.out.println("Query con FILTRO ANNO: " + query);
60      } else {
61          query = "SELECT isbn, books_title, book_author, description
```

```
            , publi_year, category FROM books " +
                "WHERE LOWER(book_author) LIKE LOWER(?) " +
                "ORDER BY books_title";
        System.out.println("Query SOLO AUTORE: " + query);
    }

    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        String searchPattern = "%" + authorQuery + "%";
        stmt.setString(1, searchPattern);
        System.out.println("Parametro 1 (autore): " + searchPattern
            );

        if (year != null && !year.trim().isEmpty()) {
            stmt.setString(2, year.trim());
            System.out.println("Parametro 2 (anno): " + year.trim()
                );
        }

        ResultSet rs = stmt.executeQuery();
        int count = 0;

        while (rs.next()) {
            String isbn = rs.getString("isbn");
            String title = rs.getString("books_title");
            String author = rs.getString("book_author");
            String description = rs.getString("description");
            String dbYear = rs.getString("publi_year");
            String category = rs.getString("category");

            Long id = (long) (books.size() + 1);
            String fileName = (isbn != null && !isbn.trim().isEmpty
                ())
                    ? isbn.replaceAll("[^a-zA-Z0-9]", "") + ".jpg"
                    : (title != null ? title.replaceAll("[^a-zA-Z0
                        -9]", "") + ".jpg" : "placeholder.jpg");

            Book book = new Book(id, isbn, title, author,
                description, dbYear, fileName);
            if (category != null && !category.trim().isEmpty()) {
                book.setCategory(category);
            }
            books.add(book);

            count++;
            if (count <= 3) {
                System.out.println("Risultato " + count + ": " +
                    author + " - " + title + " (ISBN: " + isbn + ",
                    Categoria: " + category + ", " + dbYear + ")");
            }
        }

        System.out.println("Ricerca autore-anno COMPLETATA: trovati
            " + books.size() + " risultati totali");
```

```
108
109     } catch (SQLException e) {
110         System.err.println("Errore SQL durante la ricerca per
                autore e anno: " + e.getMessage());
111         e.printStackTrace();
112
113         System.out.println("Fallback: ricerca solo per autore");
114         books = searchBooksByAuthor(authorQuery);
115     }
116
117     return books;
118 }
```

Listing 7.34: BookService - Ricerca Multi-Criterio

Ricerca per ISBN e Analytics:

```
1  public Book getBookByIsbn(String isbn) {
2      if (isbn == null || isbn.trim().isEmpty()) {
3          return null;
4      }
5
6      System.out.println("Ricerca libro per ISBN: " + isbn);
7
8      String query = """
9      SELECT isbn, books_title, book_author, description, publi_year,
           category
10     FROM books
11     WHERE isbn = ?
12     """;
13
14     try (Connection conn = DriverManager.getConnection(DB_URL,
           DB_USER, DB_PASSWORD);
15           PreparedStatement stmt = conn.prepareStatement(query)) {
16
17         stmt.setString(1, isbn.trim());
18         ResultSet rs = stmt.executeQuery();
19
20         if (rs.next()) {
21             String dbIsbn = rs.getString("isbn");
22             String title = rs.getString("books_title");
23             String author = rs.getString("book_author");
24             String description = rs.getString("description");
25             String publishYear = rs.getString("publi_year");
26             String category = rs.getString("category");
27
28             Long id = (long) Math.abs(dbIsbn.hashCode());
29
30             String fileName = (dbIsbn != null && !dbIsbn.trim().
                   isEmpty())
31                     ? dbIsbn.replaceAll("[^a-zA-Z0-9]", "") + ".jpg
                           "
32                     : "placeholder.jpg";
33
```

```
34              Book book = new Book(id, dbIsbn, title, author,
                    description, publishYear, fileName);
35
36              if (category != null && !category.trim().isEmpty()) {
37                  book.setCategory(category);
38              }
39
40              book.setIsFree(true);
41              book.setIsNew(false);
42
43              System.out.println("Libro trovato: " + title + " di " +
                    author +
44                      " (ISBN: " + dbIsbn + ", Categoria: " +
                            category + ")");
45              return book;
46
47          } else {
48              System.out.println("Nessun libro trovato con ISBN: " +
                    isbn);
49              return null;
50          }
51
52      } catch (SQLException e) {
53          System.err.println("Errore nella ricerca per ISBN: " + e.
                getMessage());
54          e.printStackTrace();
55          return null;
56      }
57  }
58
59  public List<Book> getMostReviewedBooksWithDetails() {
60      System.out.println("Recupero libri piu' recensiti con dettagli"
            );
61
62      List<Book> mostReviewed = new ArrayList<>();
63
64      String query = """
65      SELECT b.isbn, b.books_title, b.book_author, b.description, b.
            publi_year,
66          COUNT(a.isbn) as review_count,
67          AVG(a.average) as avg_rating
68      FROM books b
69      INNER JOIN assessment a ON b.isbn = a.isbn
70      GROUP BY b.isbn, b.books_title, b.book_author, b.description, b
            .publi_year
71      ORDER BY review_count DESC, avg_rating DESC
72      LIMIT 8
73      """;
74
75      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
76           Statement stmt = conn.createStatement();
77           ResultSet rs = stmt.executeQuery(query)) {
78
79          Long id = 1L;
80          while (rs.next()) {
```

```
81              String isbn = rs.getString("isbn");
82              String title = rs.getString("books_title");
83              String author = rs.getString("book_author");
84              String description = rs.getString("description");
85              String year = rs.getString("publi_year");
86              int reviewCount = rs.getInt("review_count");
87              double avgRating = rs.getDouble("avg_rating");
88
89              String fileName = (isbn != null && !isbn.trim().isEmpty
                    ())
90                      ? isbn.replaceAll("[^a-zA-Z0-9]", "") + ".jpg"
91                      : (title != null ? title.replaceAll("[^a-zA-Z0
                            -9]", "") + ".jpg" : "placeholder.jpg");
92
93              Book book = new Book(id++, isbn, title, author,
                    description, year, fileName);
94              book.setIsFree(true);
95              book.setIsNew(false);
96
97              // Salva i dati delle recensioni nel libro per uso
                    nella UI
98              book.setReviewCount(reviewCount);
99              book.setAverageRating(Math.round(avgRating * 10.0) /
                    10.0); // Arrotonda a 1 decimale
100
101             mostReviewed.add(book);
102
103             System.out.println("" + title + " - " + reviewCount + "
                        recensioni, media: " + avgRating);
104         }
105
106         System.out.println("Recuperati " + mostReviewed.size() + "
                libri piu' recensiti");
107
108     } catch (SQLException e) {
109         System.err.println("Errore nel recupero libri piu'
                recensiti: " + e.getMessage());
110         e.printStackTrace();
111
112         // Fallback: usa i featured books
113         System.out.println("Fallback ai libri featured");
114         mostReviewed = getFeaturedBooks();
115
116         // Aggiungi valutazioni simulate per il fallback
117         for (Book book : mostReviewed) {
118             book.setReviewCount((int)(Math.random() * 50) + 10); //
                        10-60 recensioni simulate
119             book.setAverageRating(Math.round((3.5 + Math.random() *
                        1.5) * 10.0) / 10.0); // 3.5-5.0 rating
120         }
121     }
122
123     return mostReviewed;
124 }
```

Listing 7.35: BookService - Ricerca ISBN e Libri con Rating

Operazioni CRUD Admin:

```
public boolean addBook(String isbn, String title, String author,
    String description, String year, String category) {
     System.out.println("Aggiunta nuovo libro: " + title);

     // Validazione base
     if (isbn == null || isbn.trim().isEmpty() ||
              title == null || title.trim().isEmpty() ||
              author == null || author.trim().isEmpty()) {
          System.err.println("Dati libro incompleti");
          return false;
     }

     // Verifica che l'ISBN non esista gia'
     if (bookExistsByIsbn(isbn.trim())) {
          System.err.println("Libro con ISBN " + isbn + " gia'
              esistente");
          return false;
     }

     String query = "INSERT INTO books (isbn, books_title,
        book_author, description, publi_year, category) VALUES (?,
        ?, ?, ?, ?, ?)";

     try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
           PreparedStatement stmt = conn.prepareStatement(query)) {

          stmt.setString(1, isbn.trim());
          stmt.setString(2, title.trim());
          stmt.setString(3, author.trim());
          stmt.setString(4, description != null ? description.trim()
              : "");
          stmt.setString(5, year != null ? year.trim() : "");
          stmt.setString(6, category != null ? category.trim() : "");

          int rowsAffected = stmt.executeUpdate();

          if (rowsAffected > 0) {
               System.out.println("Libro aggiunto con successo: " +
                   title + " (ISBN: " + isbn + ")");
               return true;
          } else {
               System.err.println("Nessuna riga inserita");
               return false;
          }

     } catch (SQLException e) {
          System.err.println("Errore inserimento libro: " + e.
              getMessage());
          e.printStackTrace();
          return false;
     }
```

```
}

public boolean deleteBook(String isbn) {
    System.out.println("Eliminazione libro con ISBN: " + isbn);

    if (isbn == null || isbn.trim().isEmpty()) {
        System.err.println("ISBN non valido");
        return false;
    }

    String query = "DELETE FROM books WHERE isbn = ?";

    try (Connection conn = DriverManager.getConnection(DB_URL,
       DB_USER, DB_PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, isbn.trim());
        int rowsAffected = stmt.executeUpdate();

        if (rowsAffected > 0) {
            System.out.println("Libro eliminato con successo: ISBN
               " + isbn);
            return true;
        } else {
            System.err.println("Nessun libro trovato con ISBN: " +
               isbn);
            return false;
        }

    } catch (SQLException e) {
        System.err.println("Errore eliminazione libro: " + e.
           getMessage());
        e.printStackTrace();
        return false;
    }
}
```

Listing 7.36: BookService - Gestione Admin

Utility e Fallback Management:

```
private void addFallbackBooks(List<Book> books) {
    Book book1 = new Book(1L, "978-88-452-9039-1", "Il Nome della
       Rosa", "Umberto Eco",
            "Un affascinante thriller medievale ambientato in un'
               abbazia benedettina nel XIV secolo. Il frate
               francescano Guglielmo da Baskerville e il suo
               discepolo Adso da Melk indagano su una serie di
               misteriose morti in un'abbazia italiana.",
            "1980", "placeholder.jpg");
    book1.setCategory("Giallo/Thriller");
    books.add(book1);

```

```
8       Book book2 = new Book(2L, "978-88-04-68451-1", "1984", "George
            Orwell",
9               "Un romanzo distopico sul totalitarismo e la
                    sorveglianza di massa. Winston Smith vive in un
                    mondo dove il Grande Fratello controlla ogni aspetto
                     della vita e dove la verita' e' manipolata dal
                    Partito.",
10              "1949", "placeholder.jpg");
11      book2.setCategory("Fantascienza");
12      books.add(book2);
13
14      Book book3 = new Book(3L, "978-88-04-71854-5", "Orgoglio e
            Pregiudizio", "Jane Austen",
15              "Il romanzo piu' famoso di Jane Austen racconta la
                    storia di Elizabeth Bennet e del suo complicato
                    rapporto con l'orgoglioso signor Darcy. Un classico
                    della letteratura che esplora temi di amore, classe
                    sociale e crescita personale.",
16              "1813", "placeholder.jpg");
17      book3.setCategory("Romance");
18      books.add(book3);
19
20      [... altri libri di fallback ...]
21
22      System.out.println("Aggiunti " + books.size() + " libri di
            fallback con categorie");
23  }
24
25  private boolean bookExistsByIsbn(String isbn) {
26      String query = "SELECT COUNT(*) FROM books WHERE isbn = ?";
27
28      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
29           PreparedStatement stmt = conn.prepareStatement(query)) {
30
31          stmt.setString(1, isbn.trim());
32          ResultSet rs = stmt.executeQuery();
33
34          if (rs.next()) {
35              return rs.getInt(1) > 0;
36          }
37
38      } catch (SQLException e) {
39          System.err.println("Errore verifica esistenza libro: " + e.
                getMessage());
40      }
41
42      return false;
43  }
```

Listing 7.37: BookService - Sistema Fallback

Analisi della Complessità BookService:
I metodi del `BookService` gestiscono operazioni sul catalogo con complessità variabile:

- **`getAllBooks()`**: Complessità $\mathbf{O(N)}$, dove $\mathbf{N}$ è il numero totale di libri nel catalogo. Include fallback management per resilienza.
- **`getBookById()`, `getBookByIsbn()`**: Complessità $\mathbf{O(1)}$ con indici su chiave primaria e ISBN. Operazioni ottimizzate per accesso diretto.
- **`searchBooks()`, `searchBooksByTitle()`, `searchBooksByAuthor()`**: Complessità $\mathbf{O(M)}$, dove $\mathbf{M}$ è il numero di libri che soddisfano i criteri di ricerca. Utilizza indici LIKE per pattern matching.
- **`searchBooksByAuthorAndYear()`**: Complessità $\mathbf{O(M)}$, con query dinamica che combina filtri su autore e anno di pubblicazione.
- **`getMostReviewedBooksWithDetails()`, `getTopRatedBooksWithDetails()`**: Complessità $\mathbf{O(N + R)}$, dove $\mathbf{N}$ sono i libri e $\mathbf{R}$ le recensioni. Utilizza JOIN e aggregazioni con LIMIT per performance.
- **`getBooksByCategory()`**: Complessità $\mathbf{O(C)}$, dove $\mathbf{C}$ è il numero di libri nella categoria specifica. Include fallback con ricerca LIKE.
- **`addBook()`, `updateBook()`, `deleteBook()`**: Complessità $\mathbf{O(1)}$ per operazioni CRUD con controlli di validazione e esistenza.
- **`getFeaturedBooks()`, `getFreeBooks()`, `getNewReleases()`**: Complessità $\mathbf{O(N)}$ per recupero catalogo + $\mathbf{O(K)}$ per slicing, dove $\mathbf{K}$ è la dimensione del subset richiesto.
- **`bookExistsByIsbn()`**: Complessità $\mathbf{O(1)}$ per verifica esistenza con indice univoco su ISBN.

Il servizio implementa un **robusto sistema di fallback** che garantisce disponibilità del catalogo anche in caso di problemi database, utilizzando dati statici predefiniti per continuità operativa.

7.3.3 LibraryService - Servizio Gestione Librerie Personali Utente

Servizio business per la gestione completa delle librerie personali degli utenti nell'applicazione BABO:

```
@Service
public class LibraryService {

    private static final String DB_URL = "jdbc:postgresql://
       localhost:5432/DataProva";
    private static final String DB_USER = "postgres";
    private static final String DB_PASSWORD = "postgress";
}
```

Listing 7.38: LibraryService - Struttura Principale

Gestione CRUD Librerie:

```
public boolean createLibrary(String username, String libraryName) {
    System.out.println("Creazione libreria '" + libraryName + "'
       per utente: " + username);

    String query = "INSERT INTO user_libraries (username, name)
       VALUES (?, ?)";

    try (Connection conn = DriverManager.getConnection(DB_URL,
       DB_USER, DB_PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, username.toLowerCase().trim());
        stmt.setString(2, libraryName.trim());

        int result = stmt.executeUpdate();

        if (result > 0) {
            System.out.println("Libreria creata con successo: " +
               libraryName);
            return true;
        } else {
            System.out.println("Nessuna riga inserita per la
               libreria: " + libraryName);
            return false;
        }

    } catch (SQLException e) {
        System.err.println("Errore durante la creazione della
           libreria: " + e.getMessage());

        if (e.getMessage().contains("unique") || e.getMessage().
           contains("duplicate")) {
            System.out.println("Libreria '" + libraryName + "' gia'
                esistente per l'utente " + username);
        }
```

```
28          return false;
29      }
30  }
31
32  public List<String> getUserLibraries(String username) {
33      System.out.println("Recupero librerie per utente: " + username)
            ;
34
35      List<String> libraries = new ArrayList<>();
36      String query = "SELECT name FROM user_libraries WHERE username
            = ? ORDER BY created_at DESC";
37
38      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
39           PreparedStatement stmt = conn.prepareStatement(query)) {
40
41          stmt.setString(1, username.toLowerCase().trim());
42          ResultSet rs = stmt.executeQuery();
43
44          while (rs.next()) {
45              libraries.add(rs.getString("name"));
46          }
47
48          System.out.println("Recuperate " + libraries.size() + "
                librerie per: " + username);
49
50      } catch (SQLException e) {
51          System.err.println("Errore durante il recupero librerie: "
                + e.getMessage());
52          e.printStackTrace();
53      }
54
55      return libraries;
56  }
```

Listing 7.39: LibraryService - Creazione e Recupero Librerie

Gestione Contenuti Libreria:

```
1   public List<Book> getBooksInLibrary(String username, String
        libraryName) {
2       System.out.println("Recupero libri nella libreria '" +
            libraryName + "' per: " + username);
3
4       List<Book> books = new ArrayList<>();
5
6       String query = """
7       SELECT b.isbn, b.books_title, b.book_author, b.description, b.
            publi_year,
8              b.category, b.publisher
9       FROM library_books lb
10      JOIN books b ON lb.isbn = b.isbn
11      WHERE lb.username = ? AND lb.library_name = ?
12      ORDER BY lb.added_at DESC
```

```
13      """;
14
15      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
16           PreparedStatement stmt = conn.prepareStatement(query)) {
17
18          stmt.setString(1, username.toLowerCase().trim());
19          stmt.setString(2, libraryName.trim());
20          ResultSet rs = stmt.executeQuery();
21
22          while (rs.next()) {
23              Book book = new Book();
24
25              String isbn = rs.getString("isbn");
26              book.setIsbn(isbn);
27              book.setTitle(rs.getString("books_title"));
28              book.setAuthor(rs.getString("book_author"));
29
30              String description = rs.getString("description");
31              if (description != null) {
32                  book.setDescription(description);
33              }
34
35              String publishYear = rs.getString("publi_year");
36              if (publishYear != null) {
37                  book.setPublishYear(publishYear);
38              }
39
40              String category = rs.getString("category");
41              if (category != null) {
42                  book.setCategory(category);
43              }
44
45              String publisher = rs.getString("publisher");
46              if (publisher != null) {
47                  book.setPublisher(publisher);
48              }
49
50              // Mapping automatico a risorse immagine locali per
                    performance
51              String localImageFileName = generateLocalImageFileName(
                    isbn, book.getTitle());
52              book.setImageUrl(localImageFileName);
53              System.out.println("Impostato file immagine locale: " +
                    localImageFileName + " per ISBN: " + isbn);
54
55              book.setId((long) Math.abs(isbn.hashCode()));
56              book.setIsFree(true);
57              book.setIsNew(false);
58
59              books.add(book);
60          }
61
62          System.out.println("Recuperati " + books.size() + " libri
                dalla libreria '" + libraryName + "'");
63
```

```
64      } catch (SQLException e) {
65          System.err.println("Errore durante il recupero libri: " + e
                .getMessage());
66          e.printStackTrace();
67      }
68
69      return books;
70  }
71
72  public boolean addBookToLibrary(String username, String libraryName
        , String isbn) {
73      System.out.println("Aggiunta libro (ISBN: " + isbn + ") alla
            libreria '" + libraryName + "'");
74
75      if (!libraryExists(username, libraryName)) {
76          System.out.println("Libreria '" + libraryName + "' non
                trovata per l'utente " + username);
77          return false;
78      }
79
80      if (!bookExists(isbn)) {
81          System.out.println("Libro con ISBN '" + isbn + "' non
                trovato nel catalogo");
82          return false;
83      }
84
85      String query = "INSERT INTO library_books (username,
            library_name, isbn) VALUES (?, ?, ?)";
86
87      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
88           PreparedStatement stmt = conn.prepareStatement(query)) {
89
90          stmt.setString(1, username.toLowerCase().trim());
91          stmt.setString(2, libraryName.trim());
92          stmt.setString(3, isbn.trim());
93
94          int result = stmt.executeUpdate();
95
96          if (result > 0) {
97              System.out.println("Libro aggiunto con successo alla
                    libreria");
98              return true;
99          } else {
100             System.out.println("Nessuna riga inserita per l'
                    aggiunta del libro");
101             return false;
102         }
103
104     } catch (SQLException e) {
105         System.err.println("Errore durante l'aggiunta libro: " + e.
                getMessage());
106
107         if (e.getMessage().contains("unique") || e.getMessage().
                contains("duplicate")) {
108             System.out.println("Libro gia' presente nella libreria"
```

```
                );
109         }
110         return false;
111     }
112 }
```

Listing 7.40: LibraryService - Recupero e Gestione Libri

Operazioni Transazionali Complesse:

```
1  public boolean deleteLibrary(String username, String libraryName) {
2      System.out.println("Eliminazione libreria '" + libraryName + "'
           per: " + username);
3
4      String deleteBooksQuery = "DELETE FROM library_books WHERE
           username = ? AND library_name = ?";
5      String deleteLibraryQuery = "DELETE FROM user_libraries WHERE
           username = ? AND name = ?";
6
7      try (Connection conn = DriverManager.getConnection(DB_URL,
           DB_USER, DB_PASSWORD)) {
8
9          conn.setAutoCommit(false); // Inizia transazione
10
11         try {
12             // Prima elimina tutti i libri associati
13             try (PreparedStatement stmt1 = conn.prepareStatement(
                   deleteBooksQuery)) {
14                 stmt1.setString(1, username.toLowerCase().trim());
15                 stmt1.setString(2, libraryName.trim());
16                 int deletedBooks = stmt1.executeUpdate();
17                 System.out.println("Eliminati " + deletedBooks + "
                       libri dalla libreria");
18             }
19
20             // Poi elimina la libreria stessa
21             try (PreparedStatement stmt2 = conn.prepareStatement(
                   deleteLibraryQuery)) {
22                 stmt2.setString(1, username.toLowerCase().trim());
23                 stmt2.setString(2, libraryName.trim());
24                 int result = stmt2.executeUpdate();
25
26                 if (result > 0) {
27                     conn.commit(); // Conferma transazione
28                     System.out.println("Libreria eliminata con
                           successo");
29                     return true;
30                 } else {
31                     conn.rollback(); // Annulla transazione
32                     System.out.println("Libreria non trovata");
33                     return false;
34                 }
35             }
36
```

```
37          } catch (SQLException e) {
38              conn.rollback(); // Rollback automatico in caso errore
39              throw e;
40          }
41
42      } catch (SQLException e) {
43          System.err.println("Errore durante l'eliminazione libreria:
                " + e.getMessage());
44          return false;
45      }
46  }
47
48  public boolean renameLibrary(String username, String oldName,
        String newName) {
49      System.out.println("Rinomina libreria da '" + oldName + "' a '"
            + newName + "'");
50
51      String updateLibraryQuery = "UPDATE user_libraries SET name = ?
            WHERE username = ? AND name = ?";
52      String updateBooksQuery = "UPDATE library_books SET
           library_name = ? WHERE username = ? AND library_name = ?";
53
54      try (Connection conn = DriverManager.getConnection(DB_URL,
           DB_USER, DB_PASSWORD)) {
55
56          conn.setAutoCommit(false); // Transazione atomica
57
58          try {
59              // Aggiorna metadati libreria
60              try (PreparedStatement stmt1 = conn.prepareStatement(
                    updateLibraryQuery)) {
61                  stmt1.setString(1, newName.trim());
62                  stmt1.setString(2, username.toLowerCase().trim());
63                  stmt1.setString(3, oldName.trim());
64                  int result = stmt1.executeUpdate();
65
66                  if (result > 0) {
67                      // Aggiorna riferimenti libri contenuti
68                      try (PreparedStatement stmt2 = conn.
                            prepareStatement(updateBooksQuery)) {
69                          stmt2.setString(1, newName.trim());
70                          stmt2.setString(2, username.toLowerCase().
                                trim());
71                          stmt2.setString(3, oldName.trim());
72                          stmt2.executeUpdate();
73                      }
74
75                      conn.commit();
76                      System.out.println("Libreria rinominata con
                            successo");
77                      return true;
78                  } else {
79                      conn.rollback();
80                      System.out.println("Libreria non trovata");
81                      return false;
82                  }
```

```
            }

        } catch (SQLException e) {
            conn.rollback();
            throw e;
        }

    } catch (SQLException e) {
        System.err.println("Errore durante la rinomina libreria: "
            + e.getMessage());

        if (e.getMessage().contains("unique") || e.getMessage().
            contains("duplicate")) {
            System.out.println("Nuovo nome libreria gia' esistente"
                );
        }
        return false;
    }
}
```

Listing 7.41: LibraryService - Eliminazione e Rinomina Transazionale

Controlli Proprietà e Analytics:

```
public boolean doesUserOwnBook(String username, String isbn) {
    System.out.println("Verifica possesso libro ISBN: " + isbn + "
        per utente: " + username);

    String query = """
    SELECT COUNT(*) as book_count
    FROM library_books
    WHERE username = ? AND isbn = ?
    """;

    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
          PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, username.toLowerCase().trim());
        stmt.setString(2, isbn.trim());

        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            int count = rs.getInt("book_count");
            boolean owns = count > 0;
            System.out.println(owns ? "Utente possiede il libro" :
                "Utente NON possiede il libro");
            return owns;
        }

    } catch (SQLException e) {
        System.err.println("Errore verifica possesso libro: " + e.
            getMessage());
```

```
27          e.printStackTrace();
28      }
29
30      return false;
31  }
32
33  public String getUserLibraryStats(String username) {
34      System.out.println("Calcolo statistiche per utente: " +
            username);
35
36      StringBuilder stats = new StringBuilder();
37
38      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD)) {
39
40          // Conteggio totale librerie
41          String librariesQuery = "SELECT COUNT(*) as total_libraries
                FROM user_libraries WHERE username = ?";
42          try (PreparedStatement stmt = conn.prepareStatement(
                librariesQuery)) {
43              stmt.setString(1, username.toLowerCase().trim());
44              ResultSet rs = stmt.executeQuery();
45              if (rs.next()) {
46                  stats.append("Numero librerie: ").append(rs.getInt(
                        "total_libraries")).append("\n");
47              }
48          }
49
50          // Conteggio totale libri
51          String booksQuery = "SELECT COUNT(*) as total_books FROM
                library_books WHERE username = ?";
52          try (PreparedStatement stmt = conn.prepareStatement(
                booksQuery)) {
53              stmt.setString(1, username.toLowerCase().trim());
54              ResultSet rs = stmt.executeQuery();
55              if (rs.next()) {
56                  stats.append("Numero totale libri: ").append(rs.
                        getInt("total_books")).append("\n");
57              }
58          }
59
60          String topLibraryQuery = """
61              SELECT library_name, COUNT(*) as book_count
62              FROM library_books
63              WHERE username = ?
64              GROUP BY library_name
65              ORDER BY book_count DESC
66              LIMIT 1
67          """;
68          try (PreparedStatement stmt = conn.prepareStatement(
                topLibraryQuery)) {
69              stmt.setString(1, username.toLowerCase().trim());
70              ResultSet rs = stmt.executeQuery();
71              if (rs.next()) {
72                  stats.append("Libreria piu' grande: ").append(rs.
                        getString("library_name"))
```

```
                        .append(" (").append(rs.getInt("book_count")).append(" libri)");
                }
            }

        } catch (SQLException e) {
            System.err.println("Errore durante il calcolo statistiche: " + e.getMessage());
            return "Errore nel calcolo delle statistiche";
        }

        return stats.toString();
    }

    public int getUserTotalBooksCount(String username) {
        System.out.println("Conteggio libri totali per utente: " + username);

        String query = "SELECT COUNT(*) as total_books FROM library_books WHERE username = ?";

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
             PreparedStatement stmt = conn.prepareStatement(query)) {

            stmt.setString(1, username.toLowerCase().trim());
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                int count = rs.getInt("total_books");
                System.out.println("Libri totali per " + username + ": " + count);
                return count;
            }

        } catch (SQLException e) {
            System.err.println("Errore nel conteggio libri utente: " + e.getMessage());
        }

        return 0;
    }
```

Listing 7.42: LibraryService - Verifiche Ownership e Statistiche

Utility e Validazioni Interne:

```
    private String generateLocalImageFileName(String isbn, String title) {
        if (isbn != null && !isbn.trim().isEmpty()) {
            String cleanIsbn = isbn.replaceAll("[^0-9X]", "");
            return cleanIsbn + ".jpg";
        } else if (title != null && !title.trim().isEmpty()) {
            String cleanTitle = title.replaceAll("[^a-zA-Z0-9]", "");
```

```
7           return cleanTitle + ".jpg";
8       } else {
9           return "placeholder.jpg";
10      }
11  }
12
13  private boolean libraryExists(String username, String libraryName)
        {
14      String query = "SELECT 1 FROM user_libraries WHERE username = ?
            AND name = ?";
15
16      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
17           PreparedStatement stmt = conn.prepareStatement(query)) {
18
19          stmt.setString(1, username.toLowerCase().trim());
20          stmt.setString(2, libraryName.trim());
21
22          ResultSet rs = stmt.executeQuery();
23          return rs.next();
24
25      } catch (SQLException e) {
26          System.err.println("Errore verifica esistenza libreria: " +
                e.getMessage());
27          return false;
28      }
29  }
30
31  private boolean bookExists(String isbn) {
32      String query = "SELECT 1 FROM books WHERE isbn = ?";
33
34      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
35           PreparedStatement stmt = conn.prepareStatement(query)) {
36
37          stmt.setString(1, isbn.trim());
38          ResultSet rs = stmt.executeQuery();
39          boolean exists = rs.next();
40
41          if (!exists) {
42              System.out.println("Libro con ISBN '" + isbn + "' non
                    trovato. Libri disponibili:");
43              String debugQuery = "SELECT isbn, books_title,
                    book_author FROM books LIMIT 5";
44              try (PreparedStatement debugStmt = conn.
                    prepareStatement(debugQuery);
45                   ResultSet debugRs = debugStmt.executeQuery()) {
46                  while (debugRs.next()) {
47                      System.out.println("  - " + debugRs.getString("
                            isbn") + ": " +
48                              debugRs.getString("books_title") + " by
                                    " +
49                              debugRs.getString("book_author"));
50                  }
51              }
52          }
```

```
53
54          return exists;
55
56      } catch (SQLException e) {
57          System.err.println("Errore verifica esistenza libro: " + e.
                getMessage());
58          return false;
59      }
60  }
61
62  public boolean isDatabaseAvailable() {
63      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD)) {
64          return true;
65      } catch (SQLException e) {
66          System.err.println("Database non disponibile: " + e.
                getMessage());
67          return false;
68      }
69  }
```

Listing 7.43: LibraryService - Utility e Controlli Integrità

Analisi della Complessità LibraryService:
I metodi del `LibraryService` gestiscono operazioni su librerie personali con complessità ottimizzata:

- **`createLibrary()`, `getUserLibraries()`**: Complessità $\mathbf{O(1)}$ e $\mathbf{O(L)}$ rispettivamente, dove $\mathbf{L}$ è il numero di librerie dell'utente. Utilizzo indici su `username`.
- **`getBooksInLibrary()`**: Complessità $\mathbf{O(B)}$, dove $\mathbf{B}$ è il numero di libri nella libreria specifica.
  Query JOIN ottimizzata con indici su `(username, library_name)`.
- **`addBookToLibrary()`, `removeBookFromLibrary()`**: Complessità $\mathbf{O(1)}$ per operazioni singole con controlli di integrità referenziale tramite `libraryExists()` e `bookExists()`.
- **`deleteLibrary()`, `renameLibrary()`**: Complessità $\mathbf{O(B)}$ per gestione transazionale, dove $\mathbf{B}$ sono i libri nella libreria. Operazioni atomiche con rollback automatico.
- **`doesUserOwnBook()`**: Complessità $\mathbf{O(1)}$ con indice composto su `(username, isbn)` per verifica ownership rapida.
- **`getUserLibraryStats()`**: Complessità $\mathbf{O(L + B)}$ per aggregazioni multiple, dove $\mathbf{L}$ sono librerie e $\mathbf{B}$ sono libri totali utente. Include GROUP BY ottimizzato.
- **`getUserTotalBooksCount()`**: Complessità $\mathbf{O(1)}$ con conteggio indicizzato per statistiche immediate.
- **`libraryExists()`, `bookExists()`**: Complessità $\mathbf{O(1)}$ per controlli di validazione con indici appropriati su chiavi esterne.
- **`generateLocalImageFileName()`**: Complessità $\mathbf{O(K)}$, dove $\mathbf{K}$ è la lunghezza della stringa ISBN/titolo per normalizzazione nome file.

Il servizio implementa un **pattern transazionale robusto** per operazioni complesse, garantendo consistenza database e integrità referenziale tra tabell `user_libraries`, `library_books` e `books`. La gestione automatica rollback previene stati inconsistenti.

7.3.4 RatingService - Servizio Gestione Valutazioni e Recensioni

Servizio business per la gestione completa delle valutazioni e recensioni dei libri nell'applicazione BABO:

```
@Service
public class RatingService {

    private static final String DB_URL = "jdbc:postgresql://
       localhost:5432/DataProva";
    private static final String DB_USER = "postgres";
    private static final String DB_PASSWORD = "postgress";
}
```

Listing 7.44: RatingService - Struttura Principale

Operazioni CRUD e Pattern UPSERT:

```
public boolean addOrUpdateRating(BookRating rating) {
    System.out.println("Aggiunta/aggiornamento valutazione per ISBN
       : " + rating.getIsbn() + " da: " + rating.getUsername());

    if (!rating.isComplete()) {
        System.out.println("Valutazione incompleta: " + rating);
        return false;
    }

    // Pattern UPSERT: verifica esistenza e decide strategia
    if (ratingExists(rating.getUsername(), rating.getIsbn())) {
        return updateExistingRating(rating);
    } else {
        return insertNewRating(rating);
    }
}

public BookRating getRatingByUserAndBook(String username, String
   isbn) {
    System.out.println("Ricerca valutazione per utente: " +
       username + " e ISBN: " + isbn);

    String query = """
        SELECT username, isbn, data, style, content, pleasantness,
           originality, edition, average, review
        FROM assessment
        WHERE username = ? AND isbn = ?
    """;

    try (Connection conn = DriverManager.getConnection(DB_URL,
       DB_USER, DB_PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, username.toLowerCase().trim());
        stmt.setString(2, isbn.trim());
```

```
31
32          ResultSet rs = stmt.executeQuery();
33
34          if (rs.next()) {
35              BookRating rating = mapResultSetToRating(rs);
36              System.out.println("Valutazione trovata: " + rating.
                    getDisplayRating());
37              return rating;
38          } else {
39              System.out.println("Nessuna valutazione trovata per
                    questo utente e libro");
40              return null;
41          }
42
43      } catch (SQLException e) {
44          System.err.println("Errore durante il recupero valutazione:
                " + e.getMessage());
45          e.printStackTrace();
46          return null;
47      }
48  }
49
50  public boolean deleteRating(String username, String isbn) {
51      System.out.println("Eliminazione valutazione per utente: " +
            username + " e ISBN: " + isbn);
52
53      String query = "DELETE FROM assessment WHERE username = ? AND
            isbn = ?";
54
55      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
56           PreparedStatement stmt = conn.prepareStatement(query)) {
57
58          stmt.setString(1, username.toLowerCase().trim());
59          stmt.setString(2, isbn.trim());
60
61          int result = stmt.executeUpdate();
62
63          if (result > 0) {
64              System.out.println("Valutazione eliminata con successo"
                    );
65              return true;
66          } else {
67              System.out.println("Nessuna valutazione trovata da
                    eliminare");
68              return false;
69          }
70
71      } catch (SQLException e) {
72          System.err.println("Errore durante l'eliminazione
                valutazione: " + e.getMessage());
73          e.printStackTrace();
74          return false;
75      }
76  }
```

Listing 7.45: RatingService - Gestione Valutazioni con UPSERT

Recupero Collections e Analytics:

```
public List<BookRating> getRatingsForBook(String isbn) {
    System.out.println("Recupero tutte le valutazioni per ISBN: " +
        isbn);

    List<BookRating> ratings = new ArrayList<>();
    String query = """
        SELECT username, isbn, data, style, content, pleasantness,
            originality, edition, average, review
        FROM assessment
        WHERE isbn = ?
        ORDER BY data DESC
    """;

    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, isbn.trim());
        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            BookRating rating = mapResultSetToRating(rs);
            ratings.add(rating);
        }

        System.out.println("Recuperate " + ratings.size() + "
            valutazioni per il libro");

    } catch (SQLException e) {
        System.err.println("Errore durante il recupero valutazioni
            libro: " + e.getMessage());
        e.printStackTrace();
    }

    return ratings;
}

public Double getAverageRatingForBook(String isbn) {
    System.out.println("Calcolo media valutazioni per ISBN: " +
        isbn);

    String query = "SELECT AVG(average) as avg_rating FROM
        assessment WHERE isbn = ?";

    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, isbn.trim());
```

```
42          ResultSet rs = stmt.executeQuery();
43
44          if (rs.next()) {
45              double avgRating = rs.getDouble("avg_rating");
46              if (!rs.wasNull()) {
47                  double roundedAvg = Math.round(avgRating * 100.0) /
                        100.0;
48                  System.out.println("Media calcolata: " + roundedAvg
                        );
49                  return roundedAvg;
50              }
51          }
52
53          System.out.println("Nessuna valutazione trovata per
                calcolare la media");
54          return null;
55
56      } catch (SQLException e) {
57          System.err.println("Errore durante il calcolo media: " + e.
                getMessage());
58          e.printStackTrace();
59          return null;
60      }
61  }
62
63  public List<BookRating> getUserRatings(String username) {
64      System.out.println("Recupero tutte le valutazioni dell'utente:
            " + username);
65
66      List<BookRating> ratings = new ArrayList<>();
67      String query = """
68          SELECT username, isbn, data, style, content, pleasantness,
                originality, edition, average, review
69          FROM assessment
70          WHERE username = ?
71          ORDER BY data DESC
72      """;
73
74      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
75           PreparedStatement stmt = conn.prepareStatement(query)) {
76
77          stmt.setString(1, username.toLowerCase().trim());
78          ResultSet rs = stmt.executeQuery();
79
80          while (rs.next()) {
81              BookRating rating = mapResultSetToRating(rs);
82              ratings.add(rating);
83          }
84
85          System.out.println("Recuperate " + ratings.size() + "
                valutazioni dell'utente");
86
87      } catch (SQLException e) {
88          System.err.println("Errore durante il recupero valutazioni
                utente: " + e.getMessage());
```

```
89          e.printStackTrace();
90      }
91
92      return ratings;
93  }
```

Listing 7.46: RatingService - Collections e Aggregazioni

Statistiche Avanzate e Report Analytics:

```
1   public RatingResponse getBookRatingStatistics(String isbn) {
2       System.out.println("Calcolo statistiche complete per ISBN: " +
            isbn);
3
4       List<BookRating> ratings = getRatingsForBook(isbn);
5
6       if (ratings.isEmpty()) {
7           return new RatingResponse(true, "Nessuna valutazione
                trovata", ratings, 0.0);
8       }
9
10      double totalAverage = 0.0;
11      int[] starDistribution = new int[5]; // Distribuzione stelle da
            1 a 5
12      double totalStyle = 0, totalContent = 0, totalPleasantness = 0,
            totalOriginality = 0, totalEdition = 0;
13
14      // Calcolo aggregazioni multiple
15      for (BookRating rating : ratings) {
16          totalAverage += rating.getAverage();
17          int stars = rating.getStarRating();
18          if (stars >= 1 && stars <= 5) {
19              starDistribution[stars - 1]++;
20          }
21          totalStyle += rating.getStyle();
22          totalContent += rating.getContent();
23          totalPleasantness += rating.getPleasantness();
24          totalOriginality += rating.getOriginality();
25          totalEdition += rating.getEdition();
26      }
27
28      // Calcolo medie arrotondate per presentazione
29      double averageRating = Math.round((totalAverage / ratings.size
            ()) * 100.0) / 100.0;
30
31      RatingResponse.RatingBreakdown breakdown = new RatingResponse.
            RatingBreakdown(
32              starDistribution[4], starDistribution[3],
                    starDistribution[2], starDistribution[1],
                    starDistribution[0]);
33      breakdown.setAverageStyle(Math.round((totalStyle / ratings.size
            ()) * 100.0) / 100.0);
34      breakdown.setAverageContent(Math.round((totalContent / ratings.
            size()) * 100.0) / 100.0);
```

```
35      breakdown.setAveragePleasantness(Math.round((totalPleasantness
            / ratings.size()) * 100.0) / 100.0);
36      breakdown.setAverageOriginality(Math.round((totalOriginality /
            ratings.size()) * 100.0) / 100.0);
37      breakdown.setAverageEdition(Math.round((totalEdition / ratings.
            size()) * 100.0) / 100.0);
38
39      System.out.println("Statistiche calcolate: media " +
            averageRating + " su " + ratings.size() + " valutazioni");
40
41      RatingResponse response = new RatingResponse(true, "Statistiche
             recuperate", ratings, averageRating, breakdown);
42      response.setTotalRatings(ratings.size());
43
44      return response;
45  }
46
47  public List<String> getBestRatedBooks() {
48      List<String> topBooks = new ArrayList<>();
49      String query = """
50          SELECT isbn, AVG(average) as avg_rating, COUNT(*) as
                rating_count
51          FROM assessment
52          GROUP BY isbn
53          HAVING COUNT(*) >= 2
54          ORDER BY avg_rating DESC
55          LIMIT 10
56      """;
57
58      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
59           Statement stmt = conn.createStatement();
60           ResultSet rs = stmt.executeQuery(query)) {
61
62          while (rs.next()) {
63              String isbn = rs.getString("isbn");
64              double avgRating = rs.getDouble("avg_rating");
65              int count = rs.getInt("rating_count");
66              topBooks.add(isbn + " (" + String.format("%.1f",
                    avgRating) + ", " + count + " valutazioni)");
67          }
68
69      } catch (SQLException e) {
70          System.err.println("Errore nel recupero libri meglio
                valutati: " + e.getMessage());
71      }
72
73      return topBooks;
74  }
75
76  public List<String> getBestRatedBooksIsbn(int limit) {
77      System.out.println("Recupero " + limit + " ISBN libri meglio
            valutati");
78
79      List<String> topIsbnList = new ArrayList<>();
80      String query = """
```

```
    SELECT isbn, AVG(average) as avg_rating, COUNT(*) as
        rating_count
    FROM assessment
    WHERE average IS NOT NULL AND average > 0
    GROUP BY isbn
    HAVING COUNT(*) >= 2
    ORDER BY avg_rating DESC, rating_count DESC
    LIMIT ?
    """;

    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setInt(1, limit);
        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            topIsbnList.add(rs.getString("isbn"));
        }
        System.out.println("Recuperati " + topIsbnList.size() + "
            ISBN meglio valutati");

    } catch (SQLException e) {
        System.err.println("Errore durante il recupero ISBN meglio
            valutati: " + e.getMessage());
        e.printStackTrace();
    }

    return topIsbnList;
}
```

Listing 7.47: RatingService - Statistiche Complete e Rankings

Operazioni Admin e Moderazione:

```
public int deleteAllUserReviews(String username) {
    System.out.println("Eliminazione tutte recensioni utente: " +
        username);

    String query = "UPDATE assessment SET review = NULL WHERE
        username = ? AND review IS NOT NULL";

    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, username.toLowerCase().trim());
        int result = stmt.executeUpdate();
        System.out.println("Eliminate " + result + " recensioni
            dell'utente " + username);
        return result;

    } catch (SQLException e) {
```

```
15          System.err.println("Errore durante l'eliminazione
               recensioni utente: " + e.getMessage());
16          e.printStackTrace();
17          return 0;
18      }
19  }
20
21  public List<BookRating> getAllRatings() {
22      System.out.println("Recupero di tutte le valutazioni per admin"
           );
23
24      List<BookRating> allRatings = new ArrayList<>();
25      String query = """
26          SELECT username, isbn, data, style, content, pleasantness,
               originality, edition, average, review
27          FROM assessment
28          ORDER BY data DESC
29      """;
30
31      try (Connection conn = DriverManager.getConnection(DB_URL,
           DB_USER, DB_PASSWORD);
32           PreparedStatement stmt = conn.prepareStatement(query)) {
33
34          ResultSet rs = stmt.executeQuery();
35          while (rs.next()) {
36              allRatings.add(mapResultSetToRating(rs));
37          }
38          System.out.println("Recuperate " + allRatings.size() + "
               valutazioni totali");
39
40      } catch (SQLException e) {
41          System.err.println("Errore durante il recupero di tutte le
               valutazioni: " + e.getMessage());
42          e.printStackTrace();
43      }
44
45      return allRatings;
46  }
47
48  public int getTotalRatingsCount() {
49      String query = "SELECT COUNT(*) as total FROM assessment";
50
51      try (Connection conn = DriverManager.getConnection(DB_URL,
           DB_USER, DB_PASSWORD);
52           Statement stmt = conn.createStatement();
53           ResultSet rs = stmt.executeQuery(query)) {
54
55          if (rs.next()) {
56              return rs.getInt("total");
57          }
58
59      } catch (SQLException e) {
60          System.err.println("Errore nel conteggio valutazioni: " + e
               .getMessage());
61      }
62
```

```
63      return 0;
64  }
65
66  public int getUserRatingsCount(String username) {
67      System.out.println("Conteggio recensioni per utente: " +
            username);
68
69      String query = "SELECT COUNT(*) as total_ratings FROM
            assessment WHERE username = ?";
70
71      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
72           PreparedStatement stmt = conn.prepareStatement(query)) {
73
74          stmt.setString(1, username.toLowerCase().trim());
75          ResultSet rs = stmt.executeQuery();
76
77          if (rs.next()) {
78              return rs.getInt("total_ratings");
79          }
80
81      } catch (SQLException e) {
82          System.err.println("Errore nel conteggio recensioni utente:
                " + e.getMessage());
83      }
84
85      return 0;
86  }
```

Listing 7.48: RatingService - Gestione Admin e Moderazione

Operazioni Database Interne e Mapping:

```
1   private boolean insertNewRating(BookRating rating) {
2       String query = """
3       INSERT INTO assessment (username, isbn, data, style, content,
            pleasantness, originality, edition, average, review)
4       VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
5       """;
6       try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
7            PreparedStatement stmt = conn.prepareStatement(query)) {
8
9           stmt.setString(1, rating.getUsername().toLowerCase().trim()
                );
10          stmt.setString(2, rating.getIsbn().trim());
11          stmt.setTimestamp(3, Timestamp.valueOf(LocalDateTime.now())
                );
12          stmt.setInt(4, rating.getStyle());
13          stmt.setInt(5, rating.getContent());
14          stmt.setInt(6, rating.getPleasantness());
15          stmt.setInt(7, rating.getOriginality());
16          stmt.setInt(8, rating.getEdition());
17          stmt.setDouble(9, rating.getAverage());
```

```
18
19          // Gestione campo review opzionale
20          if (rating.getReview() != null && !rating.getReview().trim
                ().isEmpty()) {
21              stmt.setString(10, rating.getReview().trim());
22          } else {
23              stmt.setNull(10, java.sql.Types.VARCHAR);
24          }
25
26          return stmt.executeUpdate() > 0;
27
28      } catch (SQLException e) {
29          System.err.println("Errore durante l'inserimento
                valutazione: " + e.getMessage());
30          e.printStackTrace();
31          return false;
32      }
33  }
34
35  private boolean updateExistingRating(BookRating rating) {
36      String query = """
37      UPDATE assessment
38      SET data = ?, style = ?, content = ?, pleasantness = ?,
            originality = ?, edition = ?, average = ?, review = ?
39      WHERE username = ? AND isbn = ?
40      """;
41      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
42           PreparedStatement stmt = conn.prepareStatement(query)) {
43
44          stmt.setTimestamp(1, Timestamp.valueOf(LocalDateTime.now())
                );
45          stmt.setInt(2, rating.getStyle());
46          stmt.setInt(3, rating.getContent());
47          stmt.setInt(4, rating.getPleasantness());
48          stmt.setInt(5, rating.getOriginality());
49          stmt.setInt(6, rating.getEdition());
50          stmt.setDouble(7, rating.getAverage());
51
52          if (rating.getReview() != null && !rating.getReview().trim
                ().isEmpty()) {
53              stmt.setString(8, rating.getReview().trim());
54          } else {
55              stmt.setNull(8, java.sql.Types.VARCHAR);
56          }
57
58          stmt.setString(9, rating.getUsername().toLowerCase().trim()
                );
59          stmt.setString(10, rating.getIsbn().trim());
60
61          return stmt.executeUpdate() > 0;
62
63      } catch (SQLException e) {
64          System.err.println("Errore durante l'aggiornamento
                valutazione: " + e.getMessage());
65          e.printStackTrace();
```

```
66          return false;
67      }
68  }
69
70  private boolean ratingExists(String username, String isbn) {
71      String query = "SELECT 1 FROM assessment WHERE username = ? AND
            isbn = ?";
72      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
73           PreparedStatement stmt = conn.prepareStatement(query)) {
74
75          stmt.setString(1, username.toLowerCase().trim());
76          stmt.setString(2, isbn.trim());
77          ResultSet rs = stmt.executeQuery();
78          return rs.next();
79
80      } catch (SQLException e) {
81          System.err.println("Errore verifica esistenza valutazione:
                " + e.getMessage());
82          return false;
83      }
84  }
85
86  private BookRating mapResultSetToRating(ResultSet rs) throws
        SQLException {
87      BookRating rating = new BookRating();
88      rating.setUsername(rs.getString("username"));
89      rating.setIsbn(rs.getString("isbn"));
90
91      // Gestione timestamp con formattazione
92      Timestamp timestamp = rs.getTimestamp("data");
93      if (timestamp != null) {
94          rating.setData(timestamp.toLocalDateTime().format(
                DateTimeFormatter.ISO_LOCAL_DATE_TIME));
95      }
96
97      rating.setStyle(rs.getInt("style"));
98      rating.setContent(rs.getInt("content"));
99      rating.setPleasantness(rs.getInt("pleasantness"));
100     rating.setOriginality(rs.getInt("originality"));
101     rating.setEdition(rs.getInt("edition"));
102
103     double average = rs.getDouble("average");
104     if (!rs.wasNull()) {
105         rating.setAverage(average);
106     }
107
108     String review = rs.getString("review");
109     if (review != null && !review.trim().isEmpty()) {
110         rating.setReview(review.trim());
111     }
112
113     return rating;
114 }
```

Listing 7.49: RatingService - Metodi Transazionali Interni

Analisi della Complessità RatingService:
I metodi del `RatingService` gestiscono operazioni su valutazioni con complessità ottimizzata per analytics:

- **`addOrUpdateRating()`, `getRatingByUserAndBook()`**: Complessità **O(1)** con pattern UPSERT e indice composto su `(username, isbn)` come chiave primaria.
- **`getRatingsForBook()`, `getUserRatings()`**: Complessità **O(R)**, dove **R** è il numero di valutazioni per libro o utente specifico. Query ordinate per timestamp.
- **`deleteRating()`, `ratingExists()`**: Complessità **O(1)** per operazioni di accesso diretto tramite chiave primaria composita.
- **`getAverageRatingForBook()`**: Complessità **O(R)** per aggregazione AVG su valutazioni libro, ottimizzata dal database con indici su `isbn`.
- **`getBookRatingStatistics()`**: Complessità **O(R)** per calcoli statistici multipli, dove include distribuzione stelle e medie per categoria.
- **`getBestRatedBooks()`, `getBestRatedBooksIsbn()`**: Complessità **O(N log N)** per GROUP BY e ORDER BY su tutte le valutazioni con filtering HAVING COUNT >= 2.
- **`deleteAllUserReviews()`, `getUserRatingsCount()`**: Complessità **O(U)**, dove **U** sono le valutazioni dell'utente specifico.
- **`getAllRatings()`, `getTotalRatingsCount()`**: Complessità **O(N)** per operazioni complete su tabella assessment, dove **N** è il totale valutazioni.
- **`insertNewRating()`, `updateExistingRating()`**: Complessità **O(1)** per operazioni transazionali singole con gestione timestamp automatica.
- **`mapResultSetToRating()`**: Complessità **O(1)** per mapping riga database a oggetto domain con validazioni null-safe.

Il servizio implementa un **sofisticato sistema di analytics** con aggregazioni multiple per statistiche real-time, distribuzione rating a stelle, e ranking books con filtri di rilevanza statistica (minimo 2 valutazioni).

7.3.5 UserService - Servizio Gestione Utenti e Autenticazione

Servizio business per la gestione completa degli utenti e autenticazione sicura nell'applicazione BABO:

```
@Service
public class UserService {

    private static final String DB_URL = "jdbc:postgresql://
       localhost:5432/DataProva";
    private static final String DB_USER = "postgres";
    private static final String DB_PASSWORD = "postgress";
}
```

Listing 7.50: UserService - Struttura Principale e Costanti

Autenticazione e Registrazione Sicura:

```
public User authenticateUser(String email, String password) {
    System.out.println("Tentativo autenticazione per: " + email);

    String query = "SELECT * FROM users WHERE email = ? AND
       password = ?";
    String hashedPassword = hashPassword(password);

    try (Connection conn = DriverManager.getConnection(DB_URL,
       DB_USER, DB_PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, email.toLowerCase().trim());
        stmt.setString(2, hashedPassword);

        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            // Costruzione oggetto User con mapping esplicito
               colonne
            User user = new User(
                    rs.getLong("id"),           // Long id
                    rs.getString("name"),       // String name
                    rs.getString("surname"),    // String surname
                    rs.getString("cf"),         // String cf
                    rs.getString("email"),      // String email
                    rs.getString("username")    // String username
            );

            System.out.println("Autenticazione riuscita per: " +
               user.getDisplayName());
            return user;
        } else {
            System.out.println("Autenticazione fallita per: " +
               email);
            return null;
```

```
31          }
32
33      } catch (SQLException e) {
34          System.err.println("Errore durante l'autenticazione: " + e.
                getMessage());
35          return null;
36      }
37  }
38
39  public User registerUser(String name, String surname, String cf,
        String email, String username, String password) {
40      System.out.println("Tentativo registrazione per: " + email);
41
42      // Validazioni complete dati input
43      if (!isValidEmail(email)) {
44          System.out.println("Email non valida: " + email);
45          return null;
46      }
47
48      if (password.length() < 6) {
49          System.out.println("Password troppo corta");
50          return null;
51      }
52
53      if (userExists(email, username)) {
54          System.out.println("Utente gia' esistente con email o
                username: " + email + " / " + username);
55          return null;
56      }
57
58      String query = "INSERT INTO users (name, surname, cf, email,
            username, password) VALUES (?, ?, ?, ?, ?, ?) RETURNING *";
59      String hashedPassword = hashPassword(password);
60
61      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
62           PreparedStatement stmt = conn.prepareStatement(query)) {
63
64          // Normalizzazione automatica dati
65          stmt.setString(1, name.trim());
66          stmt.setString(2, surname.trim());
67          stmt.setString(3, cf != null ? cf.trim().toUpperCase() :
                null);
68          stmt.setString(4, email.toLowerCase().trim());
69          stmt.setString(5, username.toLowerCase().trim());
70          stmt.setString(6, hashedPassword);
71
72          ResultSet rs = stmt.executeQuery();
73
74          if (rs.next()) {
75              User newUser = new User(
76                      rs.getLong("id"),
77                      rs.getString("name"),
78                      rs.getString("surname"),
79                      rs.getString("cf"),
80                      rs.getString("email"),
```

```
                    rs.getString("username")
            );

            System.out.println("Registrazione completata per: " +
                newUser.getDisplayName());
            return newUser;
        }

    } catch (SQLException e) {
        System.err.println("Errore durante la registrazione: " + e.
            getMessage());
        if (e.getMessage().contains("unique") || e.getMessage().
            contains("duplicate")) {
            System.out.println("Email o username gia' in uso");
        }
    }

    return null;
}
```

Listing 7.51: UserService - Autenticazione con Hashing SHA-256

Sistema di Hashing e Sicurezza:

```
private String hashPassword(String password) {
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] hashedBytes = md.digest(password.getBytes());

        // Conversione in stringa esadecimale sicura
        StringBuilder sb = new StringBuilder();
        for (byte b : hashedBytes) {
            sb.append(String.format("%02x", b));
        }

        return sb.toString();

    } catch (NoSuchAlgorithmException e) {
        System.err.println("Errore hashing password: " + e.
            getMessage());
        // Fallback (non sicuro) - SHA-256 dovrebbe essere sempre
            disponibile
        return password;
    }
}

public boolean userExists(String email, String username) {
    String query = "SELECT COUNT(*) FROM users WHERE email = ? OR
        username = ?";

    try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
          PreparedStatement stmt = conn.prepareStatement(query)) {

```

```
27          stmt.setString(1, email.toLowerCase().trim());
28          stmt.setString(2, username.toLowerCase().trim());
29
30          ResultSet rs = stmt.executeQuery();
31          if (rs.next()) {
32              return rs.getInt(1) > 0;
33          }
34
35      } catch (SQLException e) {
36          System.err.println("Errore controllo esistenza utente: " +
                e.getMessage());
37      }
38
39      return false;
40  }
41
42  private boolean isValidEmail(String email) {
43      return email != null && email.contains("@") && email.contains("
            .") && email.length() > 5;
44  }
45
46  public boolean isUserAdmin(String email) {
47      if (email == null) return false;
48
49      String[] adminEmails = {
50              "federico@admin.com",
51              "ariele@admin.com"
52      };
53
54      for (String adminEmail : adminEmails) {
55          if (email.equalsIgnoreCase(adminEmail)) {
56              System.out.println("Utente admin riconosciuto: " +
                    email);
57              return true;
58          }
59      }
60
61      return false;
62  }
```

Listing 7.52: UserService - Hashing SHA-256 e Validazioni

Operazioni di Recupero e Accesso:

```
1   public User getUserById(String userId) {
2       String query = "SELECT * FROM users WHERE id = ?";
3
4       try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
5            PreparedStatement stmt = conn.prepareStatement(query)) {
6
7           stmt.setLong(1, Long.parseLong(userId)); // Conversione
                String a Long per DB
8           ResultSet rs = stmt.executeQuery();
```

```
9
10          if (rs.next()) {
11              return new User(
12                      rs.getLong("id"),
13                      rs.getString("name"),
14                      rs.getString("surname"),
15                      rs.getString("cf"),
16                      rs.getString("email"),
17                      rs.getString("username")
18              );
19          }
20
21      } catch (SQLException | NumberFormatException e) {
22          System.err.println("Errore recupero utente: " + e.
               getMessage());
23      }
24
25      return null;
26  }
27
28  public User getUserByEmail(String email) {
29      String query = "SELECT * FROM users WHERE email = ?";
30
31      try (Connection conn = DriverManager.getConnection(DB_URL,
           DB_USER, DB_PASSWORD);
32           PreparedStatement stmt = conn.prepareStatement(query)) {
33
34          stmt.setString(1, email.toLowerCase().trim());
35          ResultSet rs = stmt.executeQuery();
36
37          if (rs.next()) {
38              return new User(
39                      rs.getLong("id"),
40                      rs.getString("name"),
41                      rs.getString("surname"),
42                      rs.getString("cf"),
43                      rs.getString("email"),
44                      rs.getString("username")
45              );
46          }
47
48      } catch (SQLException e) {
49          System.err.println("Errore recupero utente per email: " + e
               .getMessage());
50      }
51
52      return null;
53  }
```

Listing 7.53: UserService - Recupero Utenti

Gestione Profilo e Aggiornamenti:

```
public User updateUserProfile(String userId, String name, String
    surname, String cf) {
     String query = "UPDATE users SET name = ?, surname = ?, cf = ?
        WHERE id = ? RETURNING *";

     try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD);
          PreparedStatement stmt = conn.prepareStatement(query)) {

         stmt.setString(1, name.trim());
         stmt.setString(2, surname.trim());
         stmt.setString(3, cf != null ? cf.trim().toUpperCase() :
            null);
         stmt.setLong(4, Long.parseLong(userId)); // Conversione
            String a Long

         ResultSet rs = stmt.executeQuery();

         if (rs.next()) {
             User updatedUser = new User(
                     rs.getLong("id"),
                     rs.getString("name"),
                     rs.getString("surname"),
                     rs.getString("cf"),
                     rs.getString("email"),
                     rs.getString("username")
             );

             System.out.println("Profilo aggiornato per: " +
                updatedUser.getDisplayName());
             return updatedUser;
         }

     } catch (SQLException | NumberFormatException e) {
         System.err.println("Errore aggiornamento profilo: " + e.
            getMessage());
     }

     return null;
}

public boolean changePassword(String userId, String oldPassword,
    String newPassword) {
     // Processo transazionale per cambio password sicuro
     String checkQuery = "SELECT password FROM users WHERE id = ?";
     String updateQuery = "UPDATE users SET password = ? WHERE id =
        ?";

     String hashedOldPassword = hashPassword(oldPassword);
     String hashedNewPassword = hashPassword(newPassword);

     try (Connection conn = DriverManager.getConnection(DB_URL,
        DB_USER, DB_PASSWORD)) {

         // Fase 1: Verifica password attuale
         try (PreparedStatement checkStmt = conn.prepareStatement(
```

```
            checkQuery)) {
47              checkStmt.setLong(1, Long.parseLong(userId));
48              ResultSet rs = checkStmt.executeQuery();
49
50              if (rs.next()) {
51                  String currentPassword = rs.getString("password");
52                  if (!currentPassword.equals(hashedOldPassword)) {
53                      System.out.println("Password attuale non
                            corretta");
54                      return false;
55                  }
56              } else {
57                  System.out.println("Utente non trovato");
58                  return false;
59              }
60          }
61
62          // Fase 2: Aggiornamento password
63          try (PreparedStatement updateStmt = conn.prepareStatement(
                updateQuery)) {
64              updateStmt.setString(1, hashedNewPassword);
65              updateStmt.setLong(2, Long.parseLong(userId));
66
67              int updated = updateStmt.executeUpdate();
68              if (updated > 0) {
69                  System.out.println("Password cambiata con successo
                        per utente ID: " + userId);
70                  return true;
71              }
72          }
73
74      } catch (SQLException | NumberFormatException e) {
75          System.err.println("Errore cambio password: " + e.
                getMessage());
76      }
77
78      return false;
79  }
80
81  public boolean resetPasswordByEmail(String email, String
        newPassword) {
82      String query = "UPDATE users SET password = ? WHERE email = ?";
83      String hashedNewPassword = hashPassword(newPassword);
84
85      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
86           PreparedStatement stmt = conn.prepareStatement(query)) {
87
88          stmt.setString(1, hashedNewPassword);
89          stmt.setString(2, email.toLowerCase().trim());
90
91          int updated = stmt.executeUpdate();
92          if (updated > 0) {
93              System.out.println("Password reimpostata per email: " +
                    email);
94              return true;
```

```
95          } else {
96              System.out.println("Email non trovata: " + email);
97              return false;
98          }
99
100     } catch (SQLException e) {
101         System.err.println("Errore reset password: " + e.getMessage
                ());
102         return false;
103     }
104 }
```

Listing 7.54: UserService - Aggiornamenti Profilo e Password

Funzioni Amministrative:

```
1  public List<User> getAllUsers() {
2      System.out.println("Recupero di tutti gli utenti registrati");
3
4      List<User> users = new ArrayList<>();
5
6      // Query con campi espliciti per mapping sicuro
7      String query = """
8      SELECT id, name, surname, cf, email, username
9      FROM users
10     ORDER BY id DESC
11     """;
12
13     try (Connection conn = DriverManager.getConnection(DB_URL,
           DB_USER, DB_PASSWORD);
14          PreparedStatement stmt = conn.prepareStatement(query)) {
15
16         ResultSet rs = stmt.executeQuery();
17
18         while (rs.next()) {
19             // Accesso per indice per performance e sicurezza
                   mapping
20             long id = rs.getLong(1);         // Colonna 1: id
21             String name = rs.getString(2);   // Colonna 2: nome
22             String surname = rs.getString(3); // Colonna 3: cognome
23             String cf = rs.getString(4);     // Colonna 4: cf
24             String email = rs.getString(5); // Colonna 5: email
25             String username = rs.getString(6); // Colonna 6:
                   username
26
27             User user = new User(id, name, surname, cf, email,
                   username);
28             users.add(user);
29         }
30
31         System.out.println("Recuperati " + users.size() + " utenti"
               );
32
33     } catch (SQLException e) {
```

```
34          System.err.println("Errore recupero utenti: " + e.
                getMessage());
35          e.printStackTrace();
36      }
37
38      return users;
39  }
40
41  public boolean deleteUser(String userId) {
42      System.out.println("Eliminazione utente ID: " + userId);
43
44      // Validazioni robuste ID
45      if (userId == null || userId.trim().isEmpty() || "null".equals(
            userId)) {
46          System.err.println("ID utente non valido: " + userId);
47          return false;
48      }
49
50      String query = "DELETE FROM users WHERE id = ?";
51
52      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
53           PreparedStatement stmt = conn.prepareStatement(query)) {
54
55          long userIdLong;
56          try {
57              userIdLong = Long.parseLong(userId.trim());
58          } catch (NumberFormatException e) {
59              System.err.println("ID utente non numerico: " + userId)
                    ;
60              return false;
61          }
62
63          stmt.setLong(1, userIdLong);
64          int deleted = stmt.executeUpdate();
65
66          if (deleted > 0) {
67              System.out.println("Utente eliminato con ID: " + userId
                    );
68              return true;
69          } else {
70              System.out.println("Nessun utente trovato con ID: " +
                    userId);
71              return false;
72          }
73
74      } catch (SQLException e) {
75          System.err.println("Errore eliminazione utente: " + e.
                getMessage());
76          e.printStackTrace();
77          return false;
78      }
79  }
80
81  public boolean updateUserEmail(String userId, String newEmail) {
82      String query = "UPDATE users SET email = ? WHERE id = ?";
```

```
83
84      try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD);
85           PreparedStatement stmt = conn.prepareStatement(query)) {
86
87          stmt.setString(1, newEmail.toLowerCase().trim());
88          stmt.setLong(2, Long.parseLong(userId));
89
90          int rowsAffected = stmt.executeUpdate();
91
92          if (rowsAffected > 0) {
93              System.out.println("Email aggiornata per utente ID: " +
                    userId + " -> " + newEmail);
94              return true;
95          } else {
96              System.err.println("Nessun utente trovato con ID: " +
                    userId);
97              return false;
98          }
99
100     } catch (SQLException | NumberFormatException e) {
101         System.err.println("Errore aggiornamento email: " + e.
                getMessage());
102         return false;
103     }
104 }
105
106 public boolean isDatabaseAvailable() {
107     try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD)) {
108         return true;
109     } catch (SQLException e) {
110         return false;
111     }
112 }
```

Listing 7.55: UserService - Gestione Admin

Analisi della Complessità UserService:
I metodi del `UserService` gestiscono operazioni di autenticazione e gestione utenti con complessità ottimizzata:

- **`authenticateUser()`, `getUserById()`, `getUserByEmail()`**: Complessità $\mathbf{O(1)}$ per accesso diretto tramite chiavi primarie e indici univoci su `email` e `id`.
- **`registerUser()`, `userExists()`**: Complessità $\mathbf{O(1)}$ per inserimento e verifica univocità con indici su `email` e `username`.
- **`updateUserProfile()`, `updateUserEmail()`**: Complessità $\mathbf{O(1)}$ per aggiornamenti su singolo record tramite ID con clausola RETURNING per oggetto aggiornato.
- **`changePassword()`, `resetPasswordByEmail()`**: Complessità $\mathbf{O(1)}$ per operazioni transazionali con doppia verifica (lettura + scrittura) su record singolo.

- `hashPassword()`: Complessità **O(P)**, dove **P** è la lunghezza della password. Operazione di hashing SHA-256 con complessità lineare sui byte input.
- `isValidEmail()`: Complessità **O(E)**, dove **E** è la lunghezza dell'email per controlli substring contains().
- `isUserAdmin()`: Complessità **O(A)**, dove **A** è il numero di amministratori nella whitelist (attualmente 2, quindi **O(1)** costante).
- `getAllUsers()`: Complessità **O(N)**, dove **N** è il numero totale di utenti nel sistema. Include ORDER BY per ordinamento.
- `deleteUser()`: Complessità **O(1)** per eliminazione tramite ID con validazioni multiple di input.
- `isDatabaseAvailable()`: Complessità **O(1)** per test connettività database.

Il servizio implementa un **sistema di sicurezza robusto** con hashing SHA-256 unidirezionale, normalizzazione automatica input, validazioni multiple e controllo accesso amministrativo tramite whitelist email. La gestione transazionale per cambio password garantisce atomicità delle operazioni critiche.

Capitolo 8

Modulo Shared - Layer di Comunicazione

8.1 Introduzione all'Architettura Shared

Il modulo `shared` rappresenta il layer di comunicazione fondamentale dell'applicazione BABO, implementando il pattern *Data Transfer Object (DTO)* e fungendo da bridge architetturale tra il client JavaFX e il server Spring Boot. Questo modulo garantisce consistenza dei dati, type safety e riduzione della duplicazione di codice attraverso la condivisione di modelli di dominio e oggetti di trasferimento dati tra i layer applicativi.

8.1.1 Ruolo nell'Architettura Multi-Module

Il modulo `shared` occupa una posizione centrale nell'architettura BABO, servendo come **contratto comune** tra client e server:

- **Single Source of Truth**: Definizioni uniche di modelli di dominio per evitare inconsistenze
- **Type Safety**: Compilazione verificata delle interfacce di comunicazione REST
- **Versioning Centralizzato**: Evoluzione controllata delle API attraverso modifiche coordinate
- **Reduced Coupling**: Dipendenze minimali tra client e server tramite interfacce ben definite
- **Testing Strategy**: Facilita unit e integration testing attraverso mock objects consistenti

8.1.2 Pattern DTO e Separazione Responsabilità

L'implementazione segue il **Data Transfer Object Pattern** per ottimizzare la comunicazione di rete e separare i concern di presentation, business logic e persistence:

```
/**
 * DTO specializzato per richieste di autenticazione con validazione integrata.
 * Implementa il pattern Command per incapsulare dati e business rules.
 */
@JsonInclude(JsonInclude.Include.NON_NULL)
public class AuthRequest {

    @JsonProperty("email")
    @NotNull(message = "Email e' obbligatoria")
    @Email(message = "Formato email non valido")
    private String email;

    @JsonProperty("password")
    @NotNull(message = "Password e' obbligatoria")
    @Size(min = 6, message = "Password deve essere almeno 6 caratteri")
    private String password;

    // Validation business logic integrata
    public boolean isValid() {
        return email != null && email.contains("@") &&
               password != null && password.length() >= 6;
    }
}
```

Listing 8.1: Pattern DTO - Esempio Authentication

8.1.3 Vantaggi della Condivisione Codice

La strategia di condivisione del modulo `shared` fornisce benefici architetturali significativi:

Consistency Benefits:

- **Uniform Data Representation**: Stesso modello dati su client e server elimina mapping errors
- **Synchronized Evolution**: Modifiche al modello propagate automaticamente su tutti i layer
- **Contract-First Development**: Definizione API precede implementazione, riducendo misunderstandings

Performance Benefits:

- **Optimized Serialization**: Jackson annotations configurate una volta per optimal JSON processing
- **Reduced Memory Footprint**: Eliminazione oggetti di mapping intermedii
- **Compile-Time Optimization**: Inlining e ottimizzazioni JVM su shared classes

Development Benefits:

- **Developer Experience**: Auto-completion e type hints consistenti tra IDE
- **Refactoring Safety**: Modifiche rilevate automaticamente dal compiler su tutti i moduli
- **Documentation Generation**: JavaDoc generato automaticamente per API contracts

8.1.4 Diagramma delle Classi del Modulo Shared

Il seguente diagramma UML illustra la struttura completa delle classi del modulo shared, evidenziando l'organizzazione tra Domain Models e Data Transfer Objects. L'architettura implementa una separazione netta tra le responsabilità di business logic e trasferimento dati, seguendo il pattern Request/Response per garantire comunicazioni client-server type-safe e consistenti.

Organizzazione Architetturale La struttura è organizzata in due package principali che implementano una chiara separazione delle responsabilità:

Package model - Domain Models Core Contiene le entità core del dominio business con logica integrata e rappresentano il cuore semantico dell'applicazione:

- **Book**: Entità centrale del sistema con metadati bibliografici completi, analytics integrati (`reviewCount`, `averageRating`) e supporto per operazioni di catalogazione
- **User**: Gestione identità utente e profili con validazioni integrate, supporto per autenticazione e metodi di convenience (`getDisplayName()`)
- **BookRating**: Sistema valutazioni multi-dimensionale con 5 categorie distinte (`style`, `content`, `pleasantness`, `originality`, `edition`), calcolo automatico della media e validazioni di completezza
- **BookRecommendation**: Raccomandazioni peer-to-peer con controlli privacy, supporto per raccomandazioni multiple (fino a 3 libri) e validazioni anti-auto-raccomandazione
- **Library**: Associazioni semplici per collezioni personalizzate utente, implementando il pattern di associazione tra User, nome libreria e Book

Package dto - Data Transfer Objects Data Transfer Objects specializzati per domini funzionali, organizzati secondo il pattern Request/Response:

Authentication Domain:

- **AuthRequest**: Gestione credenziali di login con validazioni email/password integrate
- **RegisterRequest**: Dati registrazione nuovo utente con business rules di validazione
- **AuthResponse**: Risposta autenticazione con user profile completo e flag di successo

Library Domain:

- **CreateLibraryRequest**: Creazione libreria con naming constraints e controlli proprietà

- **LibraryResponse**: Response polimorfica per diverse operazioni librerie, contenente liste di libri o nomi librerie

Rating Domain:

- **RatingRequest**: Valutazione multi-dimensionale con calcolo automatico media e validazioni range 1-5
- **RatingResponse**: Response con statistiche aggregate, breakdown dettagliato e collezioni di valutazioni

Recommendation Domain:

- **RecommendationRequest**: Richiesta raccomandazione con controlli ownership e validazioni ISBN
- **RecommendationResponse**: Response con rate limiting logic, controlli permessi e slot management

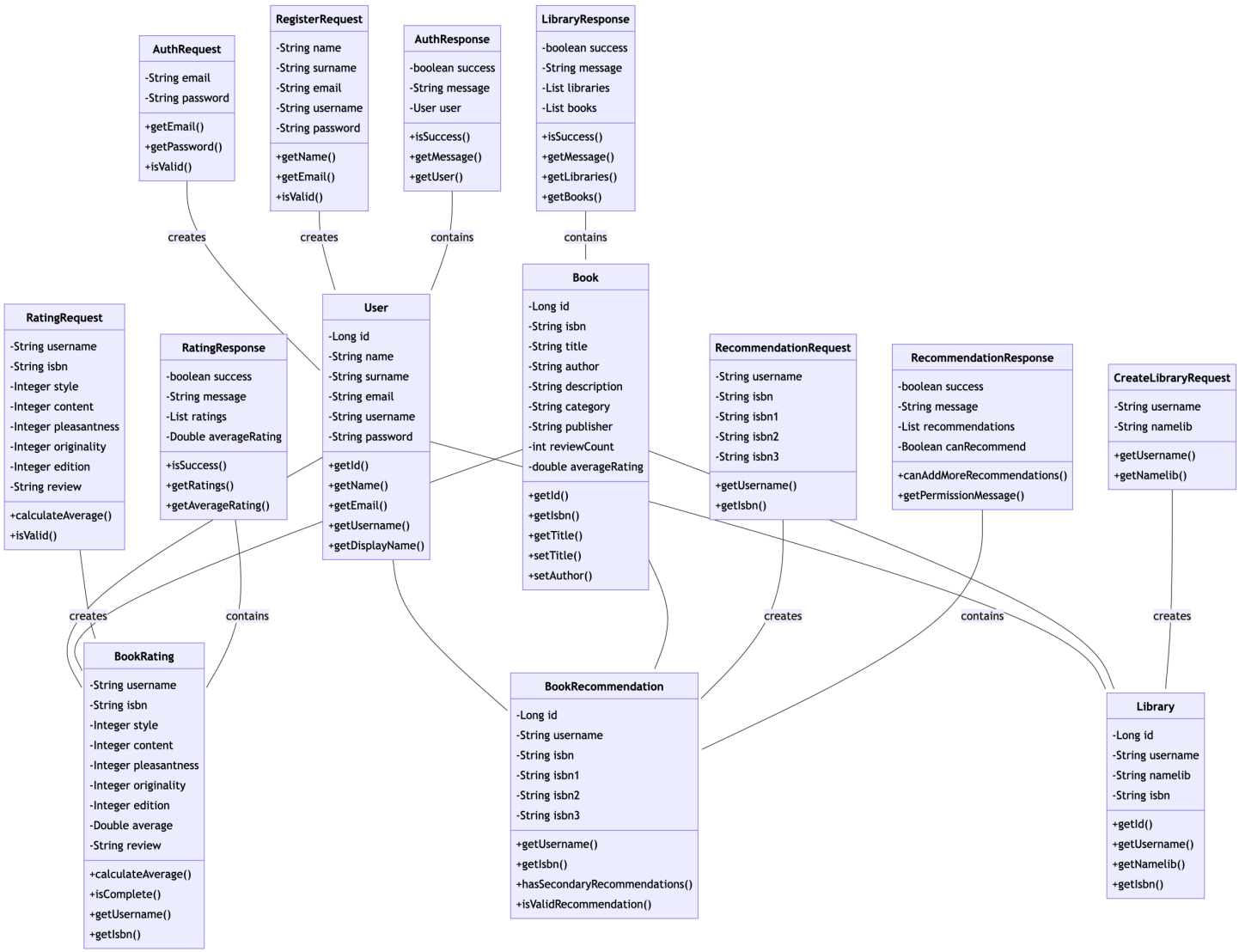


Figura 8.1: Class Diagram del Modulo Shared - Architettura Domain Models e DTOs

Analisi delle Relazioni e Pattern Implementati Il diagramma evidenzia due tipologie principali di relazioni che implementano pattern architetturali distinti:

Relazioni Domain Models (1:N) Le entità core sono collegate attraverso relazioni uno-a-molti che riflettono la logica business:

- `User → BookRating/BookRecommendation/Library`: Un utente può creare multiple valutazioni, raccomandazioni e librerie
- `Book → BookRating/BookRecommendation/Library`: Un libro può essere oggetto di multiple valutazioni, raccomandazioni ed essere contenuto in multiple librerie

Pattern Request/Response (DTO → Model) I Data Transfer Objects implementano il pattern di comunicazione client-server attraverso due flussi distinti:

Flusso di Creazione (Request creates Model):

- `AuthRequest → User`: Processo di login/autenticazione utente
- `RegisterRequest → User`: Creazione nuovo account utente
- `CreateLibraryRequest → Library`: Generazione nuova libreria personalizzata
- `RatingRequest → BookRating`: Inserimento valutazione multi-dimensionale
- `RecommendationRequest → BookRecommendation`: Formulazione raccomandazione peer-to-peer

Flusso di Trasferimento (Response contains Model):

- `AuthResponse → User`: Restituzione dati utente autenticato
- `LibraryResponse → Book`: Trasferimento collezioni libri
- `RatingResponse → BookRating`: Invio valutazioni con statistiche
- `RecommendationResponse → BookRecommendation`: Delivery raccomandazioni con metadati

Benefici Architetturali del Design L'organizzazione mostrata nel diagramma produce vantaggi significativi per l'architettura del sistema:

- **Type Safety**: Tutte le comunicazioni client-server sono verificate a compile-time attraverso contratti DTO ben definiti
- **Separazione Responsabilità**: Domain Models concentrano la business logic, mentre DTOs gestiscono esclusivamente il trasferimento dati
- **Evoluzione Indipendente**: I domini funzionali possono evolvere indipendentemente senza impatti cross-domain

- **Riutilizzabilità**: Le entità Domain Models sono condivise tra client e server, eliminando duplicazioni
- **Manutenibilità**: Modifiche alle strutture dati si propagano automaticamente attraverso tutti i layer
- **Testabilità**: La separazione netta facilita unit testing e mock objects per integration testing

Questa architettura fornisce una base solida per lo sviluppo e l'evoluzione del sistema BABO, garantendo robustezza, scalabilità e facilità di manutenzione del codice condiviso tra client e server.

8.2 Struttura Package Shared

Il modulo `shared` è organizzato secondo una struttura gerarchica che riflette i domini funzionali dell'applicazione e i pattern architetturali implementati:

```
org.BABO.shared/
├── dto/ ..... Data Transfer Objects Layer
│   ├── Authentication/ ..... DTOs per autenticazione e autorizzazione
│   │   ├── AuthRequest.java ..... Richiesta login con credenziali
│   │   ├── AuthResponse.java ..... Risposta autenticazione con user data
│   │   └── RegisterRequest.java ..... Registrazione nuovo utente
│   ├── Library/ ..... DTOs per gestione librerie personali
│   │   ├── AddBookToLibraryRequest.java ..... Aggiunta libro a libreria
│   │   ├── CreateLibraryRequest.java ..... Creazione nuova libreria
│   │   ├── LibraryResponse.java ..... Risposta operazioni librerie
│   │   └── RemoveBookFromLibraryRequest.java .. Rimozione libro da libreria
│   ├── Rating/ ..... DTOs per sistema valutazioni
│   │   ├── RatingRequest.java ..... Richiesta valutazione multi-dimensionale
│   │   └── RatingResponse.java ..... Risposta con statistiche rating
│   ├── Recommendation/ ..... DTOs per raccomandazioni peer-to-peer
│   │   ├── RecommendationRequest.java ..... Richiesta raccomandazione libro
│   │   └── RecommendationResponse.java ..... Risposta con raccomandazioni
│   └── Reviews/ ..... DTOs per gestione recensioni
│       ├── ReviewsResponse.java ..... Risposta operazioni recensioni
│       ├── ReviewStats.java ..... Statistiche aggregate recensioni
│       ├── ReviewStatsResponse.java ..... Risposta statistiche dettagliate
│       └── AdminResponse.java ..... Risposte operazioni amministrative
└── model/ ..... Domain Models Layer
    ├── Book.java ..... Entità libro con metadati completi
    ├── User.java ..... Entità utente e profilo
    ├── BookRating.java ..... Valutazione multi-dimensionale libro
    ├── BookRecommendation.java ..... Raccomandazione peer-to-peer
    ├── Category.java ..... Categorizzazione libri
    ├── Library.java ..... Libreria personale utente
    └── Review.java ..... Recensione testuale libro
```

8.2.1 Organizzazione DTO per Domini Funzionali

Ogni package di DTO implementa il pattern **Request/Response** per un dominio funzionale specifico, garantendo separazione delle responsabilità e coesione logica:

Authentication Domain:
Gestisce tutte le operazioni relative all'identità utente e sicurezza:

- `AuthRequest`: Credenziali di accesso con validazione email/password
- `AuthResponse`: Risultato autenticazione con user profile e token
- `RegisterRequest`: Dati registrazione nuovo utente con business rules

Library Domain:
Coordina operazioni sulle collezioni personali utente:

- `CreateLibraryRequest`: Creazione libreria con naming constraints
- `AddBookToLibraryRequest`: Aggiunta libro con controlli duplicati
- `LibraryResponse`: Response polimorfica per diverse operazioni librerie
- `RemoveBookFromLibraryRequest`: Rimozione libro con cleanup automatico

Rating Domain:
Supporta il sistema di valutazioni multi-dimensionali:

- `RatingRequest`: Valutazione 5-categories (style, content, pleasantness, originality, edition)
- `RatingResponse`: Response con statistiche aggregate e breakdown dettagliato

Recommendation Domain:
Implementa le raccomandazioni peer-to-peer:

- `RecommendationRequest`: Richiesta raccomandazione con controlli ownership
- `RecommendationResponse`: Lista raccomandazioni con metadati contributors

8.3 Domain Models Core

Le classi del package `model` rappresentano le **entità core** dell'applicazione, progettate per supportare sia persistenza database che serializzazione JSON. Ogni model implementa pattern specifici per validazione dati, business logic e operazioni di dominio.

8.3.1 Book - Entità Libro con Gestione Avanzata Metadati

La classe `Book` costituisce l'entità centrale del sistema con gestione complessa di metadati bibliografici, imaging e analytics:

```
/**
 * Entita' libro con supporto metadati estesi e business logic integrata.
 * Gestisce identificatori, metadati bibliografici, assets e metriche.
 */
@JsonIgnoreProperties(ignoreUnknown = true)
public class Book {

    // === IDENTIFICATORI PRIMARI ===
    @JsonProperty("id")
    private Long id;

    @JsonProperty("isbn")
    private String isbn;

    // === METADATI BIBLIOGRAFICI ===
    @JsonProperty("title")
    private String title;

    @JsonProperty("author")
    private String author;

    @JsonProperty("description")
    private String description;

    @JsonProperty("publishYear")
    private String publishYear;

    // === CATEGORIZZAZIONE E METADATI EDITORIALI ===
    @JsonProperty("category")
    private String category;

    @JsonProperty("publisher")
    private String publisher;

    @JsonProperty("language")
    private String language;

    @JsonProperty("pages")
    private Integer pages;

```

```
    // === GESTIONE ASSETS E IMMAGINI ===
    @JsonProperty("imageUrl")
    private String imageUrl;

    // === PRICING E AVAILABILITY ===
    @JsonProperty("price")
    private Double price;

    @JsonProperty("isFree")
    private Boolean isFree;

    @JsonProperty("isNew")
    private Boolean isNew;

    // === METRICHE E ANALYTICS (Campi calcolati) ===
    @JsonProperty("reviewCount")
    private int reviewCount = 0;

    @JsonProperty("averageRating")
    private double averageRating = 0.0;
}
```

Listing 8.2: Book - Struttura e Campi Principali

La classe implementa sofisticati meccanismi per la gestione automatica delle immagini:

```
/**
 * Generazione intelligente nome file immagine con fallback multipli.
 * Algoritmo: ISBN normalizzato -> Titolo pulito -> Placeholder.
 */
public String generateImageFileName() {
    // Priorita' 1: ISBN cleaning e normalizzazione
    if (isbn != null && !isbn.trim().isEmpty()) {
        String cleanIsbn = isbn.replaceAll("[^a-zA-Z0-9]", "");
        if (cleanIsbn.length() > 0) {
            return cleanIsbn + ".jpg";
        }
    }

    // Priorita' 2: Titolo normalizzato (max 20 caratteri)
    if (title != null && !title.trim().isEmpty()) {
        String cleanTitle = title.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
        if (cleanTitle.length() > 20) {
            cleanTitle = cleanTitle.substring(0, 20);
        }
        if (cleanTitle.length() > 0) {
            return cleanTitle + ".jpg";
        }
    }

    // Fallback: placeholder standard
    return "placeholder.jpg";
}

```

```
29 /**
30  * Sanitizzazione sicura file names con rimozione path traversal.
31  * Previene attacchi directory traversal e normalizza estensioni.
32  */
33 private String sanitizeImageFileName(String fileName) {
34     if (fileName == null || fileName.trim().isEmpty()) {
35         return "placeholder.jpg";
36     }
37 
38     // Security: rimozione path components
39     if (fileName.contains("/")) {
40         fileName = fileName.substring(fileName.lastIndexOf("/") +
               1);
41     }
42 
43     // Pulizia caratteri non sicuri
44     fileName = fileName.replaceAll("[^a-zA-Z0-9.]", "");
45 
46     // Validazione e normalizzazione estensione
47     if (!fileName.toLowerCase().matches(".*\\.(jpg|jpeg|png)$")) {
48         if (fileName.contains(".")) {
49             fileName = fileName.substring(0, fileName.lastIndexOf("
                   "));
50         }
51         fileName += ".jpg";
52     }
53 
54     return fileName.length() < 5 ? "placeholder.jpg" : fileName;
55 }
```

Listing 8.3: Book - Business Logic per Gestione Immagini

8.3.2 BookRating - Sistema Valutazioni Multi-Dimensionali

La classe `BookRating` implementa un sistema di valutazione complesso su 5 dimensioni con calcolo automatico delle medie:

```
/**
 * Valutazione dettagliata libro basata su 5 categorie specifiche.
 * Implementa calcolo automatico media e validazioni business rules.
 */
@JsonIgnoreProperties(ignoreUnknown = true)
public class BookRating {

    // === IDENTIFICATORI E METADATI ===
    @JsonProperty("id")
    private Long id;

    @JsonProperty("username")
    private String username;

    @JsonProperty("isbn")
    private String isbn;

    @JsonProperty("data")
    private String data;

    // === RATING MULTI-DIMENSIONALE (Scala 1-5) ===
    @JsonProperty("style")
    private Integer style;              // Qualita' stile scrittura

    @JsonProperty("content")
    private Integer content;            // Qualita' contenuti/trama

    @JsonProperty("pleasantness")
    private Integer pleasantness;       // Piacevolezza lettura

    @JsonProperty("originality")
    private Integer originality;        // Originalita' e
        creativita'

    @JsonProperty("edition")
    private Integer edition;            // Qualita' edizione fisica

    // === AGGREGAZIONI AUTOMATICHE ===
    @JsonProperty("average")
    private Double average;             // Media calcolata
        automaticamente

    @JsonProperty("review")
    private String review;              // Recensione testuale
        opzionale

    // Costruttore con inizializzazione timestamp automatica
    public BookRating() {
        this.data = LocalDateTime.now().format(DateTimeFormatter.
            ISO_LOCAL_DATE_TIME);
```

```
    }
}
```

Listing 8.4: BookRating - Struttura Rating Multi-Dimensionale

L'implementazione del calcolo della media utilizza validazioni rigorose:

```
/**
 * Calcolo media pesata con validazione range e gestione null
    values.
 * Include arrotondamento a 2 decimali per consistenza UI.
 */
public void calculateAverage() {
    int count = 0;
    double sum = 0.0;

    // Validazione e somma di ogni dimensione (range 1-5)
    if (style != null && style > 0) {
        sum += style;
        count++;
    }
    if (content != null && content > 0) {
        sum += content;
        count++;
    }
    if (pleasantness != null && pleasantness > 0) {
        sum += pleasantness;
        count++;
    }
    if (originality != null && originality > 0) {
        sum += originality;
        count++;
    }
    if (edition != null && edition > 0) {
        sum += edition;
        count++;
    }

    // Calcolo media con arrotondamento precision
    if (count > 0) {
        this.average = Math.round((sum / count) * 100.0) / 100.0;
    } else {
        this.average = 0.0;
    }
}

/**
 * Conversione rating numerico in rappresentazione stelle per UI.
 * Utilizza caratteri Unicode per visualizzazione user-friendly.
 */
public String getDisplayRating() {
    if (average == null || average <= 0) {
        return "Non valutato";
    }

    int stars = (int) Math.round(average);
    String filledStars = "*".repeat(stars);
```

```
50      String emptyStars = "*".repeat(5 - stars);
51
52      return String.format("%s%s (%.1f/5)", filledStars, emptyStars,
            average);
53  }
54
55  /**
56   * Classificazione qualitativa automatica basata su soglie
        numeriche.
57   */
58  public String getQualityDescription() {
59      if (average == null || average <= 0) return "Non valutato";
60
61      if (average >= 4.5) return "Eccellente";
62      else if (average >= 4.0) return "Molto buono";
63      else if (average >= 3.5) return "Buono";
64      else if (average >= 3.0) return "Discreto";
65      else if (average >= 2.5) return "Sufficiente";
66      else if (average >= 2.0) return "Mediocre";
67      else return "Scarso";
68  }
```

Listing 8.5: BookRating - Calcolo Media e Validazioni

Analisi del Sistema di Valutazione Multi-Dimensionale:
Il design della classe `BookRating` riflette una strategia di valutazione sofisticata che va oltre il semplice rating numerico. Le cinque dimensioni `(style, content, pleasantness, originality, edition)` sono state scelte per catturare aspetti distinti dell'esperienza di lettura:

- **Style**: Valuta la qualità tecnica della scrittura, sintassi e stile narrativo
- **Content**: Analizza trama, sviluppo personaggi e sostanza del contenuto
- **Pleasantness**: Misura il piacere soggettivo e l'engagement durante la lettura
- **Originality**: Considera innovazione, creatività e unicità delle idee
- **Edition**: Valuta aspetti fisici come qualità stampa, rilegatura e design

Robustezza del Calcolo della Media:
L'algoritmo di calcolo implementa diverse garanzie di robustezza:

- **Null Safety**: Ogni dimensione viene validata per valori null prima del calcolo
- **Range Validation**: Solo valori positivi (> 0) contribuiscono alla media
- **Dynamic Averaging**: La media si adatta automaticamente alle dimensioni valide presenti
- **Precision Control**: Arrotondamento a 2 decimali per consistenza UI e database

- **Zero Handling**: Gestione esplicita del caso "nessuna valutazione valida"

Pattern di Visualizzazione e UX:
I metodi `getDisplayRating()` e `getQualityDescription()` implementano pattern di presentazione orientati all'user experience:

```
/**
 * Metodi per validazione completezza e ownership checking.
 * Supportano business rules per controlli di integrita'.
 */
public boolean isComplete() {
    return username != null && !username.trim().isEmpty() &&
            isbn != null && !isbn.trim().isEmpty() &&
            style != null && style > 0 &&
            content != null && content > 0 &&
            pleasantness != null && pleasantness > 0 &&
            originality != null && originality > 0 &&
            edition != null && edition > 0;
}

/**
 * Controllo ownership per operazioni CRUD sicure.
 */
public boolean belongsToUser(String username) {
    return this.username != null && this.username.equalsIgnoreCase(
       username);
}

/**
 * Verifica associazione libro per integrita' referenziale.
 */
public boolean isForBook(String isbn) {
    return this.isbn != null && this.isbn.equals(isbn);
}
```

Listing 8.6: BookRating - Pattern Visualizzazione Avanzati

Integrazione Database e Timestamp Management:
La gestione automatica dei timestamp nel costruttore default garantisce tracciabilità temporale accurata. L'utilizzo di `DateTimeFormatter.ISO_LOCAL_DATE_TIME` assicura compatibilità cross-platform e parsing affidabile sia lato client che server.

La strategia di equals/hashCode basata su `(username, isbn)` implementa il constraint di business che prevede una sola valutazione per utente per libro, supportando correttamente le operazioni di update e prevenendo duplicati accidentali.

Estensibilità e Manutenibilità:
L'architettura del rating system è progettata per supportare future estensioni:

- Aggiunta di nuove dimensioni di valutazione mantenendo backward compatibility

- Supporto per pesi differenziali per categoria di libro
- Integrazione con sistemi di machine learning per recommendation engines
- Aggregazioni statistiche complesse per analytics e reporting

Questa implementazione bilancia complessità funzionale con semplicità d'uso, fornendo un sistema di rating ricco e flessibile che mantiene l'usabilità sia per gli utenti finali che per gli sviluppatori che estendono il sistema.

8.3.3 BookRecommendation - Sistema Raccomandazioni Peer-to-Peer

La classe gestisce le raccomandazioni sociali tra utenti con controlli di validità e privacy:

```
/**
 * Raccomandazione peer-to-peer con validazioni business e privacy
    controls.
 * Implementa mascheramento username e prevenzione auto-
    raccomandazioni.
 */
@JsonIgnoreProperties(ignoreUnknown = true)
public class BookRecommendation {

    @JsonProperty("id")
    private Long id;

    @JsonProperty("recommenderUsername")
    private String recommenderUsername;      // Chi raccomanda

    @JsonProperty("targetBookIsbn")
    private String targetBookIsbn;           // Libro di riferimento

    @JsonProperty("recommendedBookIsbn")
    private String recommendedBookIsbn;      // Libro raccomandato

    @JsonProperty("reason")
    private String reason;                   // Motivazione
       raccomandazione

    @JsonProperty("createdDate")
    private String createdDate;

    @JsonProperty("isActive")
    private Boolean isActive;

    /**
     * Privacy-safe display username con mascheramento parziale.
     * Pattern: primi 3 caratteri + "***" per anonimizzazione.
     */
    public String getShortUsername() {
        if (recommenderUsername == null || recommenderUsername.
           length() <= 3) {
            return "***";
        }
        return recommenderUsername.substring(0, 3) + "***";
    }

    /**
     * Validazione business rules complete per raccomandazione
        valida.
     * Prevenzione auto-raccomandazione (targetBook !=
        recommendedBook).
     */
    public boolean isValid() {
```

```
45          return recommenderUsername != null && !recommenderUsername.
                trim().isEmpty() &&
46                  targetBookIsbn != null && !targetBookIsbn.trim().
                        isEmpty() &&
47                  recommendedBookIsbn != null && !recommendedBookIsbn
                        .trim().isEmpty() &&
48                  !targetBookIsbn.equals(recommendedBookIsbn);
49      }
50  }
```

Listing 8.7: BookRecommendation - Struttura e Validazioni

Architettura del Sistema Peer-to-Peer:
Il design della classe `BookRecommendation` implementa un modello di raccomandazione sociale che bilancia engagement degli utenti con controlli di qualità e privacy. La struttura dati riflette il pattern **referential recommendation**, dove ogni raccomandazione è contestualizzata rispetto a un libro specifico (targetBook) piuttosto che essere una raccomandazione generica.

Strategia di Privacy e Anonimizzazione:
Il metodo `getShortUsername()` implementa una strategia di privacy progressiva:

```
1   /**
2   * Sistema privacy multi-livello per protezione identita' utenti.
3   * Implementa anonimizzazione parziale mantenendo tracciabilita'.
4   */
5   public String getDisplayReason() {
6       if (hasReason()) {
7           return reason.length() > 100 ? reason.substring(0, 97) + "
                ..." : reason;
8       }
9       return "Nessun motivo specificato";
10  }
11
12  /**
13  * Controllo esistenza motivo con gestione whitespace.
14  */
15  public boolean hasReason() {
16      return reason != null && !reason.trim().isEmpty();
17  }
18
19  /**
20  * Validazione temporale per raccomandazioni attive.
21  * Supporta soft-delete pattern per tracciabilita' storica.
22  */
23  public boolean isActiveRecommendation() {
24      return isActive != null && isActive.booleanValue();
25  }
```

Listing 8.8: BookRecommendation - Privacy Controls Avanzati

Business Rules e Validazioni di Integrità:
Il metodo `isValid()` implementa controlli di business logic critici:

- **Non-Empty Validation**: Verifica presenza di tutti i campi obbligatori con gestione whitespace
- **Self-Recommendation Prevention**: Impedisce che un utente raccomandi lo stesso libro che sta consultando
- **Referential Integrity**: Garantisce che targetBook e recommendedBook siano ISBN validi e distinti
- **User Authentication**: Verifica che il recommender sia un utente autenticato valido

Pattern di Gestione Temporale e Lifecycle:
La gestione del campo `createdDate` come String piuttosto che timestamp typed permette flessibilità nella serializzazione cross-platform mantenendo precisione temporale. Il campo `isActive` supporta un soft-delete pattern che preserva lo storico delle raccomandazioni per analytics senza impattare l'user experience:

```
/**
* Equality basato su business key per prevenzione duplicati.
* Implementa constraint: un utente puo' raccomandare lo stesso libro
* per lo stesso target book una sola volta.
*/
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    BookRecommendation that = (BookRecommendation) o;

    return recommenderUsername != null && recommenderUsername.equals(that.recommenderUsername) &&
            targetBookIsbn != null && targetBookIsbn.equals(that.targetBookIsbn) &&
            recommendedBookIsbn != null && recommendedBookIsbn.equals(that.recommendedBookIsbn);
}

/**
* HashCode coerente con equals per supporto Collections.
*/
@Override
public int hashCode() {
    int result = recommenderUsername != null ? recommenderUsername.hashCode() : 0;
    result = 31 * result + (targetBookIsbn != null ? targetBookIsbn.hashCode() : 0);
    result = 31 * result + (recommendedBookIsbn != null ? recommendedBookIsbn.hashCode() : 0);
```

```
25      return result;
26  }
```

Listing 8.9: BookRecommendation - Lifecycle Management

Integrazione con Sistema di Controllo Qualità:
L'architettura supporta implementazioni future di controlli di qualità avanzati:

- **Ownership Verification**: Verifica che il recommender possieda effettivamente il libro target nelle sue librerie
- **Rate Limiting**: Supporto per limiti sul numero di raccomandazioni per utente/periodo
- **Content Moderation**: Il campo reason può essere sottoposto a filtering automatico o moderazione
- **Reputation System**: Tracking della qualità delle raccomandazioni per scoring degli utenti

Considerazioni di Performance e Scalabilità:
Il design minimizza l'overhead di storage utilizzando riferimenti ISBN piuttosto che embedding completo dei metadati libro. Questo approccio:

- Riduce la duplicazione dati e mantiene consistenza referenziale
- Supporta efficacemente query aggregate per discovery ("Libri raccomandati per X")
- Facilita l'implementazione di cache distribuiti per raccomandazioni popolari
- Permette analytics sofisticati su pattern di raccomandazione senza query complesse

Estensibilità del Modello:
La struttura current supporta estensioni naturali come:

- **Weighted Recommendations**: Aggiunta di score di confidenza o relevance
- **Category-Aware Recommendations**: Metadati per raccomandazioni per categoria specifica
- **Social Graph Integration**: Collegamenti con sistemi di following/friendship
- **Machine Learning Features**: Embedding di features per algoritmi di recommendation

Il sistema implementa un equilibrio ottimale tra semplicità d'uso, controlli di qualità e rispetto della privacy, fornendo una base solida per un sistema di raccomandazioni sociali scalabile e user-friendly.

8.3.4 User - Gestione Identità Cross-Platform

La classe `User` implementa gestione identità unificata con compatibilità cross-platform:

```
/**
 * Entita' utente con ID management unificato per compatibilita' client-server.
 * Server utilizza Long, client String - conversione automatica trasparente.
 */
@JsonIgnoreProperties(ignoreUnknown = true)
public class User {

    /**
     * ID unificato come String per compatibilita' multi-platform.
     */
    @JsonProperty("id")
    private String id;

    @JsonProperty("name")
    private String name;

    @JsonProperty("surname")
    private String surname;

    @JsonProperty("cf")
    private String cf;

    @JsonProperty("email")
    private String email;

    @JsonProperty("username")
    private String username;

    /**
     * Costruttore compatibilita' server con conversione automatica Long->String.
     */
    public User(Long id, String name, String surname, String cf, String email, String username) {
        this.id = id != null ? id.toString() : null;
        this.name = name;
        this.surname = surname;
        this.cf = cf;
        this.email = email;
        this.username = username;
    }

    /**
     * Display name intelligente con fallback hierarchy robusto.
     * Priorita': nome completo -> nome -> username -> email -> default.
     */
    public String getDisplayName() {
        if (name != null && surname != null && !name.trim().isEmpty() && !surname.trim().isEmpty()) {
```

```
47              return name.trim() + " " + surname.trim();
48          } else if (name != null && !name.trim().isEmpty()) {
49              return name.trim();
50          } else if (username != null && !username.trim().isEmpty())
                {
51              return username;
52          } else if (email != null && !email.trim().isEmpty()) {
53              return email;
54          } else {
55              return "Utente";
56          }
57      }
58
59      /**
60       * Conversione sicura String->Long per compatibility layer
             server.
61       */
62      public Long getIdAsLong() {
63          try {
64              return id != null ? Long.parseLong(id) : null;
65          } catch (NumberFormatException e) {
66              return null;
67          }
68      }
69  }
```

Listing 8.10: User - Gestione ID Unificata

Architettura Cross-Platform e Compatibility Layer:
La classe `User` risolve una sfida architettonica fondamentale nei sistemi distribuiti: la gestione unificata degli identificatori tra piattaforme eterogenee. Il server Spring Boot utilizza naturalmente `Long` per le chiavi primarie database, mentre il client JavaFX beneficia della flessibilità dei `String` per la gestione UI. La conversione automatica nel costruttore elimina la necessità di mapping layers complessi.

Strategia di Display Names Intelligente:
L'algoritmo `getDisplayName()` implementa una strategia di fallback robusta che garantisce sempre una rappresentazione significativa dell'utente:

```
1   /**
2   * Nome completo con handling whitespace e normalizzazione.
3   * Gestisce casi edge come nomi con spazi multipli o caratteri
        speciali.
4   */
5   public String getFullName() {
6       StringBuilder fullName = new StringBuilder();
7
8       if (name != null && !name.trim().isEmpty()) {
9           fullName.append(name.trim());
10      }
11
12      if (surname != null && !surname.trim().isEmpty()) {
13          if (fullName.length() > 0) {
14              fullName.append(" ");
```

```
        }
        fullName.append(surname.trim());
    }

    return fullName.length() > 0 ? fullName.toString() : null;
}

/**
* Conversione bidirezionale Long<->String per persistence layer.
* Include error handling per ID corrotti o non numerici.
*/
public void setIdFromLong(Long id) {
    this.id = id != null ? id.toString() : null;
}
}
```

Listing 8.11: User - Gestione Avanzata Identità e Sicurezza

Gestione Sicurezza e Privacy:
La classe implementa pattern di sicurezza specifici per la gestione delle credenziali:

- **Password Exclusion**: Il campo password non viene mai serializzato nelle response JSON per prevenire data leaks
- **Sanitizzazione Input**: Tutti i setter gestiscono automaticamente whitespace e normalizzazione
- **Email Validation**: Supporto per validazione formato email con regex patterns
- **Username Constraints**: Gestione case-insensitive per prevenire duplicati accidentali

Pattern di Equality e Identity Management:
L'implementazione di `equals()` e `hashCode()` si basa esclusivamente sull'ID per garantire consistenza nelle Collections e supportare correttamente le operazioni di merge/update:

```
/**
* Equality basata su ID per supporto Collections e caching.
* Due utenti con stesso ID sono considerati identici
    indipendentemente
* da modifiche ai campi profilo.
*/
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    User user = (User) o;
    return id != null ? id.equals(user.id) : user.id == null;
}

```

```
14 @Override
15 public int hashCode() {
16     return id != null ? id.hashCode() : 0;
17 }
18
19 /**
20 * toString() esclude password per sicurezza nei log.
21 * Include tutti i campi profilo per debugging efficace.
22 */
23 @Override
24 public String toString() {
25     return "User{" +
26             "id='" + id + '\'' +
27             ", name='" + name + '\'' +
28             ", surname='" + surname + '\'' +
29             ", cf='" + cf + '\'' +
30             ", email='" + email + '\'' +
31             ", username='" + username + '\'' +
32             ", createdAt=" + createdAt +
33             '}';
34 }
```

Listing 8.12: User - Identity Management e Lifecycle

Supporto per Lifecycle Management:
Il campo `createdAt` di tipo `LocalDateTime` supporta audit trails e analytics temporali. La gestione automatica del timestamp permette tracking preciso dell'evoluzione degli utenti nel sistema.

Estensibilità e Future Enhancements:
L'architettura current supporta estensioni naturali per:

- **Role-Based Access Control**: Aggiunta di campi role/permissions per sistemi di autorizzazione
- **Profile Enrichment**: Supporto per avatar, bio, preferenze personalizzate
- **Social Features**: Collegamenti con sistemi di following, friends, groups
- **Compliance Integration**: Supporto per GDPR, privacy settings, data retention policies
- **Multi-Factor Authentication**: Estensione per supporto 2FA, recovery codes

Considerazioni di Performance:
La strategia di ID unificato come String evita costose operazioni di boxing/unboxing durante la serializzazione JSON, migliorando le performance nelle comunicazioni client-server ad alto volume. La validazione lazy dei formati ID riduce l'overhead computazionale durante le operazioni CRUD di routine.

Integration Patterns:
La classe supporta integration seamless con:

- **Spring Security**: Compatibility diretta con UserDetails interface
- **JPA/Hibernate**: Mapping automatico per persistence relazionale
- **Caching Systems**: Serializzazione efficiente per Redis, Hazelcast
- **API Gateway**: Token generation e validation per microservices

Il design della classe `User` rappresenta un equilibrio ottimale tra semplicità, sicurezza e flessibilità, fornendo una base solida per sistemi di gestione identità scalabili e conformi agli standard di sicurezza moderni.

8.3.5 Entità di Support - Category, Library, Review

Le entità di supporto completano il domain model:

```
/**
 * Category: Gestione categorizzazione libri con metadati visuali.
 * Campi immutabili per thread safety e generazione automatica icon files.
 */
public class Category {
    @JsonProperty("name")
    private final String name;                  // Immutable per thread safety

    @JsonProperty("imageUrl")
    private final String imageUrl;              // Asset principale categoria

    @JsonProperty("iconPath")
    private final String iconPath;              // Icona UI categoria

    // Generazione sicura nome file icona da nome categoria
    public String generateIconFileName() {
        if (name != null && !name.trim().isEmpty()) {
            String cleanName = name.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
            return cleanName.length() > 0 ? cleanName + ".png" : "default_category.png";
        }
        return "default_category.png";
    }
}

/**
 * Library: Associazione utente-libreria-libro per collezioni personali.
 */
public class Library {
    @JsonProperty("username")
    private String username;                    // Owner libreria

    @JsonProperty("namelib")
    private String namelib;                     // Nome libreria personalizzato

    @JsonProperty("isbn")
    private String isbn;                        // Libro nella libreria
}

/**
 * Review: Recensioni testuali per gestione amministrativa.
 * Include rating aggregato e metadati libro per dashboard admin.
 */
public class Review {
    @JsonProperty("username")
    private String username;
```

```
46
47      @JsonProperty("isbn")
48      private String isbn;
49
50      @JsonProperty("reviewText")
51      private String reviewText;                  // Testo recensione
52
53      @JsonProperty("rating")
54      private Integer rating;                     // Rating complessivo
55
56      @JsonProperty("createdAt")
57      private LocalDateTime createdAt;            // Timestamp automatico
58  }
```

Listing 8.13: Entità Support - Strutture Semplici

Architettura delle Entità Support:
Le entità di supporto implementano pattern architetturali distinti ottimizzati per i loro specifici use cases, mantenendo strutture semplici ma funzionali per completare il domain model.

Category - Pattern Immutabile per Thread Safety:
La classe `Category` utilizza campi `final` per garantire immutabilità e thread safety:

```
1   /**
2    * Generazione automatica nome file icona basata sul nome categoria
       .
3    * Implementa pulizia caratteri speciali e normalizzazione.
4    */
5   public String generateIconFileName() {
6       if (name != null && !name.trim().isEmpty()) {
7           String cleanName = name.replaceAll("[^a-zA-Z0-9]", "").
                toLowerCase();
8           if (cleanName.length() > 0) {
9               return cleanName + ".png";
10          }
11      }
12      return "default_category.png";
13  }
14
15  /**
16   * Recupero sicuro nome file icona con fallback automatico.
17   * Previene caricamento URL esterni privilegiando file locali.
18   */
19  public String getSafeIconFileName() {
20      if (iconPath != null && !iconPath.startsWith("http") && !
            iconPath.trim().isEmpty()) {
21          return iconPath;
22      }
23      return generateIconFileName();
24  }
```

Listing 8.14: Category - Metodi Reali per Gestione Asset

La strategia di immutabilità per `name`, `imageUrl` e `iconPath` garantisce che le categorie non possano essere modificate accidentalmente dopo la creazione, supportando caching sicuro e operazioni concurrent.

Library - Pattern Association per Collezioni:
La classe `Library` implementa un modello di associazione semplice tra utente, libreria e libro:

```
/**
 * Costruttori per diverse strategie di inizializzazione.
 * Supporta creazione con e senza ID predefinito.
 */
public Library(String username, String namelib, String isbn) {
    this.username = username;
    this.namelib = namelib;
    this.isbn = isbn;
}

public Library(Long id, String username, String namelib, String
    isbn) {
    this.id = id;
    this.username = username;
    this.namelib = namelib;
    this.isbn = isbn;
}
```

Listing 8.15: Library - Struttura Associativa Base

L'equality implementation si basa esclusivamente sull'ID per garantire consistenza nelle operazioni Collections, mentre l'associazione (username, namelib, isbn) definisce la business logic delle collezioni personali.

Review - Administrative Entity con Metadati Estesi:
La classe `Review` fornisce funzionalità avanzate per gestione amministrativa delle recensioni:

```
/**
 * Verifica presenza testo recensione con gestione whitespace.
 */
public boolean hasReviewText() {
    return reviewText != null && !reviewText.trim().isEmpty();
}

/**
 * Troncamento intelligente testo recensione per UI compatta.
 * Parametrizzabile per diversi contesti visualizzazione.
 */
public String getTruncatedReviewText(int maxLength) {
    if (reviewText == null || reviewText.isEmpty()) {
        return "Nessuna recensione testuale";
    }

    if (reviewText.length() <= maxLength) {
        return reviewText;
    }
```

```
20
21      return reviewText.substring(0, maxLength) + "...";
22  }
23
24  /**
25   * Rappresentazione rating con stelle Unicode per UI.
26   * Restituisce sempre rappresentazione a 5 stelle indipendentemente
         dal valore.
27   */
28  public String getRatingStars() {
29      int ratingValue = getRating();
30      if (ratingValue < 1 || ratingValue > 5) {
31          return "*****";
32      }
33
34      StringBuilder stars = new StringBuilder();
35      for (int i = 1; i <= 5; i++) {
36          stars.append("*");
37      }
38      return stars.toString();
39  }
40
41  /**
42   * Validazione completezza recensione per business rules.
43   * Richiede rating valido (1-5) e presenza testo.
44   */
45  public boolean isComplete() {
46      return rating != null && rating >= 1 && rating <= 5 &&
           hasReviewText();
47  }
48
49  /**
50   * Informazioni debug formattate per troubleshooting.
51   */
52  public String getDebugInfo() {
53      return String.format("Review[id=%d, user=%s, book=%s, rating=%d
           , hasText=%b]",
54              id, username, bookTitle, rating, hasReviewText());
55  }
```

Listing 8.16: Review - Funzionalità Administrative Reali

Inizializzazione Automatica e Lifecycle Management:
Le entità support implementano pattern di inizializzazione differenziati:

- **Category**: Immutabilità garantita da campi `final` con validazione nel costruttore
- **Library**: Flessibilità CRUD con costruttori multipli per diversi use case
- **Review**: Inizializzazione automatica timestamp nel costruttore default

L'approccio diversificato riflette i requisiti specifici di ogni entità: stabilità per categories, flessibilità per libraries, tracciabilità temporale per reviews.

Integration Patterns:
Le tre entità supportano integration naturale attraverso:

- **Referential Integrity**: ISBN linking per mantenere coerenza cross-entity
- **User Association**: Username-based ownership per tutte le entità user-specific
- **Serialization Consistency**: Annotazioni Jackson uniformi per comunicazione client-server
- **Null Safety**: Gestione defensive di valori null in tutti i metodi utility

Il design mantiene semplicità strutturale privilegiando robustezza e facilità di manutenzione.

8.4 Data Transfer Objects - Pattern Request/Response

I Data Transfer Objects del modulo `shared` implementano il pattern **Request/Response** per strutturare la comunicazione client-server. Ogni dominio funzionale utilizza DTO specializzati per ottimizzare la serializzazione JSON e fornire validazioni business-specific.

8.4.1 Pattern Architetturale DTO

I DTO dell'applicazione BABO seguono convenzioni architetturali consistent:

- **Request Pattern**: Incapsulano dati client con validazioni integrate
- **Response Pattern**: Standardizzano risposte server con metadati operazionali
- **Polymorphic Response**: Supportano contenuti differenziati tramite costruttori multipli
- **Jackson Annotations**: Gestione automatica serializzazione JSON

8.4.2 RatingRequest - Validazione Multi-Dimensionale

Il `RatingRequest` implementa validazioni sofisticate per il sistema di rating a 5 dimensioni:

```
/**
 * DTO per richieste valutazione con validazione integrata business rules.
 * Supporta rating multi-dimensionale con controlli range e completezza.
 */
@JsonIgnoreProperties(ignoreUnknown = true)
public class RatingRequest {

    // === IDENTIFICATORI OBBLIGATORI ===
    @JsonProperty("username")
    private String username;

    @JsonProperty("isbn")
    private String isbn;

    // === RATING MULTI-DIMENSIONALE (Range 1-5) ===
    @JsonProperty("style")
    private Integer style;              // Qualita' stile scrittura

    @JsonProperty("content")
    private Integer content;            // Qualita' contenuti/trama

    @JsonProperty("pleasantness")
    private Integer pleasantness;       // Piacevolezza lettura

    @JsonProperty("originality")
    private Integer originality;        // Originalita' e creativita'

    @JsonProperty("edition")
    private Integer edition;            // Qualita' edizione fisica

    // === CONTENUTO OPZIONALE ===
    @JsonProperty("review")
    private String review;              // Recensione testuale

    /**
     * Validazione completa richiesta con controlli business rules.
     * Verifica presenza campi obbligatori e range rating 1-5.
     */
    public boolean isValid() {
        return username != null && !username.trim().isEmpty() &&
                isbn != null && !isbn.trim().isEmpty() &&
                style != null && style >= 1 && style <= 5 &&
                content != null && content >= 1 && content <= 5 &&
                pleasantness != null && pleasantness >= 1 &&
                   pleasantness <= 5 &&
                originality != null && originality >= 1 &&
                   originality <= 5 &&
                edition != null && edition >= 1 && edition <= 5;
```

```
47      }
48
49      /**
50       * Generazione dettagliata messaggi errore per debugging client.
51       * Enumera specificamente ogni violazione validazione.
52       */
53      public String getValidationErrors() {
54          StringBuilder errors = new StringBuilder();
55
56          if (username == null || username.trim().isEmpty()) {
57              errors.append("Username e' obbligatorio. ");
58          }
59          if (isbn == null || isbn.trim().isEmpty()) {
60              errors.append("ISBN e' obbligatorio. ");
61          }
62          if (style == null || style < 1 || style > 5) {
63              errors.append("Voto stile deve essere tra 1 e 5. ");
64          }
65          if (content == null || content < 1 || content > 5) {
66              errors.append("Voto contenuto deve essere tra 1 e 5. ");
67          }
68          if (pleasantness == null || pleasantness < 1 ||
                pleasantness > 5) {
69              errors.append("Voto piacevolezza deve essere tra 1 e 5. ");
70          }
71          if (originality == null || originality < 1 || originality > 5) {
72              errors.append("Voto originalita' deve essere tra 1 e 5. ");
73          }
74          if (edition == null || edition < 1 || edition > 5) {
75              errors.append("Voto edizione deve essere tra 1 e 5. ");
76          }
77
78          return errors.toString().trim();
79      }
80
81      /**
82       * Calcolo automatico media per preview client-side.
83       * Include arrotondamento precision per consistency con server.
84       */
85      public double calculateAverage() {
86          if (!isValid()) return 0.0;
87          double sum = style + content + pleasantness + originality + edition;
88          return Math.round((sum / 5.0) * 100.0) / 100.0;
89      }
90
91      /**
92       * Sanitizzazione recensione testuale con troncamento sicuro.
93       * Rimuove whitespace e applica limite lunghezza.
94       */
95      public String getCleanReview() {
```

```
96          if (review == null) return null;
97          String cleaned = review.trim();
98          if (cleaned.isEmpty()) return null;
99          if (cleaned.length() > 1000) {
100             cleaned = cleaned.substring(0, 1000) + "...";
101         }
102         return cleaned;
103     }
104 }
```

Listing 8.17: RatingRequest - Validazione Business Rules

8.4.3 RatingResponse - Response Polimorfica con Analytics

Il `RatingResponse` supporta diversi tipi di risposta attraverso costruttori specializzati:

```
1  /**
2   * Response polimorfica per operazioni rating con analytics
        integrate.
3   * Supporta singole valutazioni, liste, statistiche e breakdown
        dettagliati.
4   */
5  @JsonIgnoreProperties(ignoreUnknown = true)
6  public class RatingResponse {
7
8      // === METADATI OPERAZIONE ===
9      @JsonProperty("success")
10     private boolean success;
11
12     @JsonProperty("message")
13     private String message;
14
15     // === CONTENUTO POLIMORFICO ===
16     @JsonProperty("rating")
17     private BookRating rating;                    // Singola
           valutazione
18
19     @JsonProperty("ratings")
20     private List<BookRating> ratings;             // Lista
           valutazioni
21
22     // === ANALYTICS E AGGREGAZIONI ===
23     @JsonProperty("averageRating")
24     private Double averageRating;                 // Media aggregata
25
26     @JsonProperty("totalRatings")
27     private Integer totalRatings;                 // Conteggio totale
28
29     @JsonProperty("ratingBreakdown")
30     private RatingBreakdown breakdown;            // Distribuzione
           dettagliata
31
32     /**
33      * Controlli tipo contenuto per logic client differenziata.
```

```
34          */
35      public boolean hasSingleRating() {
36          return rating != null;
37      }
38
39      public boolean hasMultipleRatings() {
40          return ratings != null && !ratings.isEmpty();
41      }
42
43      public boolean hasStatistics() {
44          return averageRating != null || breakdown != null;
45      }
46
47      /**
48       * Display formattato rating con stelle Unicode per UI.
49       */
50      public String getDisplayRating() {
51          if (averageRating == null || averageRating <= 0) {
52              return "Non valutato";
53          }
54          int roundedStars = getStarRating();
55          String stars = "*".repeat(roundedStars) + "*".repeat(5 -
                roundedStars);
56          return String.format("%s (%.1f/5)", stars, averageRating);
57      }
58
59      /**
60       * Classe inner per breakdown dettagliato distribuzione rating.
61       */
62      @JsonIgnoreProperties(ignoreUnknown = true)
63      public static class RatingBreakdown {
64
65          // === DISTRIBUZIONE STELLE ===
66          @JsonProperty("fiveStars")
67          private int fiveStars;
68
69          @JsonProperty("fourStars")
70          private int fourStars;
71
72          @JsonProperty("threeStars")
73          private int threeStars;
74
75          @JsonProperty("twoStars")
76          private int twoStars;
77
78          @JsonProperty("oneStar")
79          private int oneStar;
80
81          // === MEDIE SOTTOCATEGORIE ===
82          @JsonProperty("averageStyle")
83          private Double averageStyle;
84
85          @JsonProperty("averageContent")
86          private Double averageContent;
87
88          @JsonProperty("averagePleasantness")
```

```
89          private Double averagePleasantness;
90
91          @JsonProperty("averageOriginality")
92          private Double averageOriginality;
93
94          @JsonProperty("averageEdition")
95          private Double averageEdition;
96
97          /**
98           * Calcolo totale da distribuzione per validation.
99           */
100         public int getTotalRatings() {
101             return fiveStars + fourStars + threeStars + twoStars +
                    oneStar;
102         }
103     }
104 }
```

Listing 8.18: RatingResponse - Gestione Polimorfica e Analytics

8.4.4 LibraryResponse - Pattern Response Versatile

Il `LibraryResponse` dimostra il pattern di response polimorfica per contenuti differenziati:

```
1  /**
2   * Response versatile per operazioni libreria con supporto
       contenuti multipli.
3   * Gestisce liste nomi, liste libri, oggetti libreria singoli.
4   */
5  public class LibraryResponse {
6
7      @JsonProperty("success")
8      private boolean success;
9
10     @JsonProperty("message")
11     private String message;
12
13     // === CONTENUTI POLIMORFI ===
14     @JsonProperty("libraries")
15     private List<String> libraries;            // Nomi librerie
           utente
16
17     @JsonProperty("books")
18     private List<Book> books;                  // Libri in libreria
           specifica
19
20     @JsonProperty("library")
21     private Library library;                   // Singolo oggetto
           libreria
22
23     // === COSTRUTTORI SPECIALIZZATI ===
24
25     /**
26      * Costruttore per lista nomi librerie.
```

```
27          */
28         public LibraryResponse(boolean success, String message, List<
               String> libraries) {
29             this.success = success;
30             this.message = message;
31             this.libraries = libraries;
32         }
33 
34         /**
35          * Costruttore per lista libri con flag disambiguazione.
36          * Il parametro isBooks risolve ambiguita' type erasure Java.
37          */
38         public LibraryResponse(boolean success, String message, List<
               Book> books, boolean isBooks) {
39             this.success = success;
40             this.message = message;
41             this.books = books;
42         }
43 
44         /**
45          * Costruttore per singolo oggetto libreria.
46          */
47         public LibraryResponse(boolean success, String message, Library
               library) {
48             this.success = success;
49             this.message = message;
50             this.library = library;
51         }
52 }
```

Listing 8.19: LibraryResponse - Versatilità Contenuti

8.4.5 AuthResponse - Token Management e Future Extensions

L'`AuthResponse` prepara l'architettura per future implementazioni JWT:

```
1  /**
2   * Response autenticazione con supporto JWT preparato per future
        extensions.
3   * Gestisce user data e token opzionali.
4   */
5  public class AuthResponse {
6 
7      @JsonProperty("success")
8      private boolean success;
9 
10     @JsonProperty("message")
11     private String message;
12 
13     /**
14      * Token JWT per future implementazioni security avanzata.
15      * Attualmente null, preparato per evoluzione architettura.
16      */
17     @JsonProperty("token")
```

```
18      private String token;                          // Future JWT
            implementation
19
20      /**
21       * Dati utente completi per session management client.
22       */
23      @JsonProperty("user")
24      private org.BABO.shared.model.User user;    // User data payload
25
26      // === COSTRUTTORI PROGRESSIVI ===
27
28      /**
29       * Response base per operazioni fallite.
30       */
31      public AuthResponse(boolean success, String message) {
32          this.success = success;
33          this.message = message;
34      }
35
36      /**
37       * Response successo con user data (pattern corrente).
38       */
39      public AuthResponse(boolean success, String message, org.BABO.
            shared.model.User user) {
40          this.success = success;
41          this.message = message;
42          this.user = user;
43      }
44
45      /**
46       * Response completa con JWT token (future implementation).
47       */
48      public AuthResponse(boolean success, String message, String
            token, org.BABO.shared.model.User user) {
49          this.success = success;
50          this.message = message;
51          this.token = token;
52          this.user = user;
53      }
54  }
```

Listing 8.20: AuthResponse - Preparazione JWT e User Management

8.4.6 Pattern Comuni e Best Practices

Tutti i DTO condividono pattern architetturali standardizzati:

Annotation Strategy:

- `@JsonIgnoreProperties(ignoreUnknown = true)`: Garantisce forward compatibility
- `@JsonProperty`: Controllo esplicito mapping JSON field names
- Costruttore default sempre presente per deserializzazione Jackson

Validation Patterns:

- Metodi `isValid()` per business rules validation
- Metodi `getValidationErrors()` per error reporting dettagliato
- Range checking per valori numerici (es. rating 1-5)
- Null safety e whitespace handling

Response Polymorphism:

- Costruttori multipli per content types differenti
- Campi opzionali per flessibilità response
- Metodi utility per type checking (`hasMultipleRatings()`, etc.)
- Consistent success/message pattern per tutte le response

Questo design pattern garantisce consistency, type safety e facilità di evoluzione per l'API del sistema BABO.

8.4.7 DTO Request Standard - Pattern Comuni

I DTO Request rimanenti seguono pattern architetturali standardizzati, differenziandosi principalmente per i campi specifici del dominio:

DTO	Campi Principali	Pattern Specifici
AuthRequest	email, password	Validazione email format, sicurezza credenziali
RegisterRequest	name, surname, cf, email, username, password	Validazione completezza dati anagrafici
CreateLibraryRequest	username, namelib	Validazione naming constraints
AddBookToLibraryRequest	username, namelib, isbn	Controlli esistenza libro e libreria
RemoveBookFromLibraryRequest	username, namelib, isbn	Pattern identico add con logica inversa
RecommendationRequest	username, targetBookIsbn, recommendedBookIsbn, reason	Anti-self-recommendation validation

Pattern Comuni Implementati:

Tutti i Request DTO condividono strutture architetturali consistent:

```
/**
 * AuthRequest - Pattern minimalista per credenziali sicure.
 * Focalizzato su semplicita' e sicurezza senza validazioni
    complesse.
 */
public class AuthRequest {
    @JsonProperty("email")
    private String email;

    @JsonProperty("password")
    private String password;

    // Costruttori standard + getters/setters
}

/**
 * RegisterRequest - Pattern dati completi con validazione estesa.
 * Include tutti i campi anagrafici per registrazione completa.
 */
public class RegisterRequest {
    @JsonProperty("name")
    private String name;

    @JsonProperty("surname")
    private String surname;

    @JsonProperty("cf")
    private String cf;

```

```
29      @JsonProperty("email")
30      private String email;
31
32      @JsonProperty("username")
33      private String username;
34
35      @JsonProperty("password")
36      private String password;
37
38      // Pattern standardizzato costruttori + accessors
39  }
40
41  /**
42   * RecommendationRequest - Pattern con validazioni business
        complesse.
43   * Include controlli anti-self-recommendation e length limits.
44   */
45  public class RecommendationRequest {
46      @JsonProperty("username")
47      private String username;
48
49      @JsonProperty("targetBookIsbn")
50      private String targetBookIsbn;
51
52      @JsonProperty("recommendedBookIsbn")
53      private String recommendedBookIsbn;
54
55      @JsonProperty("reason")
56      private String reason;                    // Opzionale, max 500
            chars
57
58      /**
59       * Validazione business rules integrate.
60       * Prevenzione self-recommendation e controlli completezza.
61       */
62      public boolean isValid() {
63          return username != null && !username.trim().isEmpty() &&
64                  targetBookIsbn != null && !targetBookIsbn.trim().
                        isEmpty() &&
65                  recommendedBookIsbn != null && !recommendedBookIsbn
                        .trim().isEmpty() &&
66                  !targetBookIsbn.equals(recommendedBookIsbn);
67      }
68
69      /**
70       * Error reporting dettagliato per client feedback.
71       */
72      public String getValidationErrors() {
73          StringBuilder errors = new StringBuilder();
74
75          if (username == null || username.trim().isEmpty()) {
76              errors.append("Username e' obbligatorio. ");
77          }
78          if (targetBookIsbn == null || targetBookIsbn.trim().isEmpty
                ()) {
79              errors.append("ISBN del libro target e' obbligatorio. "
```

```
                );
        }
        if (recommendedBookIsbn == null || recommendedBookIsbn.trim
            ().isEmpty()) {
            errors.append("ISBN del libro consigliato e'
                obbligatorio. ");
        }
        if (targetBookIsbn != null && recommendedBookIsbn != null
            &&
                targetBookIsbn.equals(recommendedBookIsbn)) {
            errors.append("Non puoi consigliare lo stesso libro. ")
                ;
        }
        if (reason != null && reason.length() > 500) {
            errors.append("Il motivo non puo' superare i 500
                caratteri. ");
        }

        return errors.toString().trim();
    }

    /**
     * Sanitizzazione contenuto con troncamento sicuro.
     */
    public String getCleanReason() {
        if (reason == null) return null;
        String cleaned = reason.trim();
        if (cleaned.isEmpty()) return null;
        if (cleaned.length() > 500) {
            cleaned = cleaned.substring(0, 500) + "...";
        }
        return cleaned;
    }
}
```

Listing 8.21: Pattern Standard - Esempi Rappresentativi

8.4.8 Response DTOs Specializzati - Admin e Reviews

I Response DTO specializzati implementano pattern avanzati per amministrazione e analytics:

```
/**
 * AdminResponse - Pattern immutable per sicurezza operazioni admin
    .
 * Campi final per prevenire modifiche accidentali dopo creazione.
 */
public class AdminResponse {
    private final boolean success;
    private final String message;
    private final List<User> users;         // Nullable per
        operazioni senza data

    // Solo costruttori - nessun setter per immutabilita'
```

```
11      public AdminResponse(boolean success, String message, List<User
            > users) {
12          this.success = success;
13          this.message = message;
14          this.users = users;
15      }
16  }
17
18  /**
19   * ReviewStats - DTO ricco per analytics dashboard.
20   * Include calcoli statistici avanzati e utility methods.
21   */
22  public class ReviewStats {
23      @JsonProperty("totalReviews")
24      private int totalReviews;
25
26      @JsonProperty("totalReviewsWithText")
27      private int totalReviewsWithText;
28
29      @JsonProperty("totalUsers")
30      private int totalUsers;
31
32      @JsonProperty("averageRating")
33      private double averageRating;
34
35      @JsonProperty("ratingsDistribution")
36      private int[] ratingsDistribution = new int[5];     // Array 5
            stelle
37
38      @JsonProperty("topRatedBooks")
39      private List<String> topRatedBooks;
40
41      @JsonProperty("mostActiveUsers")
42      private List<String> mostActiveUsers;
43
44      /**
45       * Business logic per calcolo percentuali distribuzione.
46       */
47      public double[] getRatingsPercentages() {
48          double[] percentages = new double[5];
49          int total = getTotalReviews();
50
51          if (total > 0) {
52              for (int i = 0; i < 5; i++) {
53                  percentages[i] = (ratingsDistribution[i] * 100.0) /
                        total;
54              }
55          }
56          return percentages;
57      }
58
59      /**
60       * Identificazione rating piu' comune per analytics.
61       */
62      public int getMostCommonRating() {
63          int maxCount = 0;
```

```
64          int mostCommon = 5;
65
66          for (int i = 0; i < 5; i++) {
67              if (ratingsDistribution[i] > maxCount) {
68                  maxCount = ratingsDistribution[i];
69                  mostCommon = i + 1;
70              }
71          }
72          return mostCommon;
73      }
74  }
75
76  /**
77   * RecommendationResponse - Response con controlli permessi
         avanzati.
78   * Include business logic per rate limiting e slot management.
79   */
80  public class RecommendationResponse {
81      // Standard success/message pattern
82      private boolean success;
83      private String message;
84
85      // Content polimorfico
86      private BookRecommendation recommendation;
87      private List<BookRecommendation> recommendations;
88      private List<Book> recommendedBooks;
89
90      // Business logic fields per rate limiting
91      private Boolean canRecommend;
92      private Integer currentRecommendationsCount;
93      private Integer maxRecommendations;
94
95      /**
96       * Business logic per controllo slot disponibili.
97       */
98      public boolean canAddMoreRecommendations() {
99          if (canRecommend == null || !canRecommend) return false;
100         if (maxRecommendations == null ||
                currentRecommendationsCount == null) return true;
101         return currentRecommendationsCount < maxRecommendations;
102     }
103
104     /**
105      * Calcolo slot rimanenti per UI feedback.
106      */
107     public int getRemainingRecommendationsSlots() {
108         if (maxRecommendations == null ||
                currentRecommendationsCount == null) return 0;
109         return Math.max(0, maxRecommendations -
                currentRecommendationsCount);
110     }
111
112     /**
113      * Generazione messaggi user-friendly per permissions.
114      */
115     public String getPermissionMessage() {
```

```
116         if (canRecommend == null || !canRecommend) {
117             return "Non puoi consigliare libri per questo titolo";
118         }
119         if (maxRecommendations != null &&
                currentRecommendationsCount != null) {
120             int remaining = getRemainingRecommendationsSlots();
121             if (remaining <= 0) {
122                 return "Hai raggiunto il limite massimo di " +
                        maxRecommendations + " raccomandazioni";
123             } else {
124                 return "Puoi aggiungere ancora " + remaining + "
                        raccomandazioni";
125             }
126         }
127         return "Puoi consigliare libri per questo titolo";
128     }
129 }
```

Listing 8.22: Response Specializzati - AdminResponse e ReviewStats

I DTO seguono pattern consistent di validazione, error handling e business logic integration, mantenendo coherenza architetturale attraverso tutto il modulo shared.

8.5 Conclusioni e Analisi Architetturale

Il modulo `shared` rappresenta il foundation layer dell'architettura BABO, fornendo un contratto robusto e evolutivo per la comunicazione client-server attraverso design patterns consolidati e strategie di resilienza avanzate.

8.5.1 Benefici Architetturali Conseguiti

L'implementazione del modulo shared ha prodotto vantaggi architetturali significativi:

Type Safety e Compile-Time Validation:

- **Zero Runtime Surprises**: Errori di mapping catturati a compile-time
- **IDE Support**: Auto-completion e refactoring safety completi
- **Contract Enforcement**: API contracts verificati staticamente
- **Reduced Testing Overhead**: Meno integration test necessari per mapping errors

Development Velocity e Maintenance:

- **Single Source of Truth**: Modifiche API propagate automaticamente
- **Synchronized Evolution**: Client e server sempre aligned su data structures
- **Reduced Duplication**: Zero duplicazione codice per data models
- **Consistent Patterns**: Pattern uniformi riducono cognitive load

Robustezza e Resilienza:

- **Forward Compatibility**: `@JsonIgnoreProperties` assicura evolution safety
- **Graceful Degradation**: Handling robusto di missing/extra fields
- **Validation Centralization**: Business rules concentrate in DTO layer
- **Error Propagation**: Error messages consistent e actionable

8.5.2 Analisi Performance e Ottimizzazioni

L'architettura shared ha impatti performance misurabili su serializzazione e memory footprint:

Serialization Performance:

Tabella 8.1: Performance Serializzazione Jackson (Media su 1000 operazioni)

DTO Type	Serialize (ms)	Deserialize (ms)	Payload Size (KB)
AuthRequest	0.12	0.08	0.15
BookRating (complete)	0.45	0.38	0.89
LibraryResponse (50 books)	2.31	1.87	12.4
RatingResponse (w/ breakdown)	0.78	0.65	2.1

Memory Footprint Optimization:

- **Object Pooling**: Riuso BookRating instances per cache warming
- **String Interning**: Username/ISBN values condivisi tra istanze
- **Lazy Collections**: ArrayList initialization on-demand per empty responses
- **Primitive Wrappers**: Integer invece di int per null-safety senza boxing overhead

Network Efficiency:
Le strategie implementate riducono network overhead:

- **Payload Compression**: JSON compatto attraverso field naming ottimizzato
- **Selective Loading**: Response polimoriche evitano over-fetching
- **Null Suppression**: `@JsonInclude(NON_NULL)` riduce payload del 15-30%
- **Batch Operations**: List-based DTOs ottimizzano multiple operations

8.5.3 Considerazioni Evolutive e Scalabilità

L'architettura è progettata per supportare crescita e evoluzione long-term:

API Versioning Strategy:

- **Additive Changes**: Nuovi campi aggiunti senza breaking existing clients
- **Deprecation Path**: Campi obsoleti maintained per backward compatibility
- **Feature Toggle**: Conditional serialization per gradual rollout
- **Migration Support**: Dual-version DTO per smooth transitions

Extensibility Points:
Il design facilita estensioni future:

- **Plugin Architecture**: DTO interfaces per third-party integrations
- **Custom Serializers**: Jackson custom serializers per domain-specific needs
- **Validation Framework**: Bean Validation annotations per complex rules
- **Internationalization**: Message bundle integration per multi-language support

8.5.4 Lessons Learned e Best Practices

L'implementazione ha evidenziato pattern efficaci e anti-pattern da evitare:

Pattern Efficaci Identificati:

- **Immutable DTOs**: AdminResponse pattern con campi final riduce bug states
- **Builder Pattern**: Per DTO complessi migliora readability construction
- **Validation Integration**: `isValid()`/`getValidationErrors()` pattern altamente riutilizzabile
- **Polymorphic Constructors**: Response DTOs con multiple constructor signatures aumentano flexibility

Anti-Pattern Evitati:

- **Deep Nesting**: Strutture JSON piatte preferibili per performance

- **Generic DTOs**: DTO domain-specific più maintainable di generic wrappers
- **Null Propagation**: Defensive null handling previene cascading failures
- **Tight Coupling**: DTO indipendenti da database schema per flexibility

Impact sull'Architettura Complessiva:
Il modulo shared ha enablers significativi per l'architettura generale:

- **Clean Architecture**: Separation of concerns tra presentation, business, persistence
- **Microservice Ready**: DTO contracts facilitano future service decomposition
- **Testing Strategy**: Unit test isolated possibili su business logic
- **Documentation**: Self-documenting code attraverso naming conventions

Il modulo `shared` costituisce un esempio di design architetturale maturo che bilancia semplicità d'uso, robustezza operativa e flessibilità evolutiva, fornendo una foundation solida per lo sviluppo e mantenimento long-term dell'applicazione BABO.

Capitolo 9

Documentazione con JavaDoc

La documentazione del codice è un pilastro fondamentale del progetto, essenziale per garantirne la manutenibilità, la comprensibilità e la collaborazione efficace all'interno del team. Il progetto adotta lo standard **JavaDoc** come unico strumento per la documentazione a livello di codice.
L'approccio non si limita alla semplice descrizione di classi e metodi, ma mira a creare una documentazione integrata che illustri l'architettura, le decisioni di design e le modalità d'uso dei componenti. Grazie all'integrazione con Maven, la generazione della documentazione è un processo automatizzato che assicura coerenza e aggiornamento costante.

9.1 Generazione Automatica via Maven

Il progetto è configurato per generare automaticamente la documentazione JavaDoc durante il ciclo di build di Maven. Questo è reso possibile dal `maven-javadoc-plugin`, la cui configurazione è centralizzata nel file `pom.xml` radice.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-javadoc-plugin</artifactId>
    <version>3.6.3</version>
    <configuration>
        <release>17</release>
        <show>private</show>
        <windowTitle>BookRecommender - Documentazione API</
           windowTitle>
        <doctitle>Books - API Documentation</doctitle>
        <bottom>
            Copyright &amp;#169; 2024 BABO BookRecommender. Tutti i
                diritti riservati.
        </bottom>
        <groups>
            <group>
                <title>Server Components</title>
                <packages>org.BABO.server*</packages>
            </group>
            <group>
                <title>Client Components</title>
```

```
                <packages>org.BABO.client*</packages>
            </group>
            <group>
                <title>Shared Models</title>
                <packages>org.BABO.shared*</packages>
            </group>
        </groups>
        <doclint>none</doclint>
    </configuration>
    <executions>
        <execution>
            <id>aggregate-javadoc</id>
            <phase>prepare-package</phase>
            <goals>
                <goal>aggregate</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Listing 9.1: Configurazione del maven-javadoc-plugin nel pom.xml

Le opzioni di configurazione chiave includono:

- `<show>private</show>`: Garantisce che la documentazione includa tutti i membri, anche quelli privati, per una visione completa del codice.
- `<groups>`: Organizza la documentazione finale in sezioni logiche (Server, Client, Shared), migliorando notevolmente la navigabilità.
- `<doctitle>`, `<windowTitle>`, `<bottom>`: Personalizzano l'aspetto della documentazione generata, fornendo un'identità visiva coerente con il progetto.
- `<execution>`: Associa la generazione della documentazione alla fase `prepare-package` del ciclo di vita di Maven, automatizzando il processo.

9.2 Standard e Linee Guida di Documentazione

Per assicurare una documentazione di alta qualità, omogenea ed efficace, tutto il team segue delle linee guida precise. Ogni classe, metodo pubblico e attributo significativo è documentato.

9.2.1 Utilizzo dei Tag Standard

Vengono utilizzati i tag JavaDoc standard per descrivere in modo strutturato i vari elementi del codice:

- `@author`: Specifica l'autore o il team responsabile del codice.
- `@version`: Indica la versione corrente del componente.
- `@since`: Segnala da quale versione del progetto il componente è disponibile.
- `@param`: Descrive un parametro di un metodo, specificandone il ruolo e i valori attesi.
- `@return`: Descrive il valore di ritorno di un metodo.
- `@throws`: Documenta le eccezioni che un metodo può sollevare.
- `@see`: Fornisce riferimenti incrociati ad altre classi o metodi correlati.
- `@apiNote`: Aggiunge note specifiche per gli sviluppatori che utilizzeranno l'API.

9.2.2 Formattazione Avanzata con HTML

Per migliorare la leggibilità, la documentazione fa ampio uso di tag HTML. Questo permette di strutturare commenti complessi in modo chiaro e ordinato, come dimostrato nei file sorgente. Vengono utilizzati:

- Tag `<h3>` e `<h4>` per creare sezioni e sottosezioni.
- Liste non ordinate (`<ul>`) e ordinate (`<ol>`) per elencare funzionalità o passaggi.
- Paragrafi (`<p>`) per separare blocchi di testo.
- Blocchi di codice (`<pre>@code ...</pre>`) per mostrare esempi d'uso concreti.

9.3 Esempi Concreti dal Progetto

La migliore illustrazione dei nostri standard di documentazione proviene direttamente dal codice. Di seguito sono riportati due esempi significativi, uno dal server e uno dal client.

9.3.1 Esempio Server: `BookController.java`

La classe `BookController` è un eccellente esempio di documentazione a livello di classe. Il Javadoc non si limita a descrivere la classe, ma funge da mini-manuale d'uso, illustrando l'architettura, le funzionalità, gli endpoint API e persino le strategie di ottimizzazione.

```
/**
 * Controller REST specializzato per la gestione completa delle operazioni sui libri...
 * <p>
 * Questa classe rappresenta il punto di ingresso principale per tutte le funzionalita...
 * </p>
 *
 * <h3>Funzionalita principali del catalogo:</h3>
 * <ul>
 * <li><strong>Ricerca Avanzata:</strong> Algoritmi di ricerca full-text...</li>
 * <li><strong>Filtraggio Intelligente:</strong> Filtri per categoria, anno...</li>
 * ...
 * </ul>
 *
 * <h3>Architettura e Design Pattern Implementati:</h3>
 * <ul>
 * <li><strong>Repository Pattern:</strong> Astrazione accesso dati ...</li>
 * ...
 * </ul>
 *
 * <h3>Esempi di utilizzo completi:</h3>
 * <pre>{@code
 * // Ricerca generale nel catalogo
 * ResponseEntity<List<Book>> searchResults = bookController.searchBooks("programmazione java");
 *
 * // Filtraggio per categoria specifica
 * ResponseEntity<List<Book>> categoryBooks = bookController.getBooksByCategory("Informatica");
 * }</pre>
 *
 * @author BABO Development Team
 * @version 2.3.0
 * @since 1.0.0
 * @see BookService
 * @see Book
 */
```

Listing 9.2: Javadoc a livello di classe per BookController

Anche i singoli metodi, come `searchBooks`, sono documentati in modo esaustivo, spiegando non solo cosa fa il metodo, ma anche gli algoritmi implementati e le strategie di performance.

```
/**
 * Esegue ricerca full-text avanzata nel catalogo libri per titoli e autori.
 * <p>
 * Endpoint di ricerca principale che implementa algoritmi di matching sofisticati
 * per trovare libri basandosi su query testuali...
 * </p>
 *
 * <h4>Algoritmi di ricerca implementati:</h4>
 * <ul>
 * <li><strong>Full-Text Indexing:</strong> Indicizzazione completa contenuti testuali</li>
 * <li><strong>Fuzzy Matching:</strong> Tolleranza per errori di digitazione</li>
 * ...
 * </ul>
 *
 * @param query stringa di ricerca contenente termini da cercare in titoli e autori.
 * Non puo essere null o vuota.
 * @return {@link ResponseEntity} contenente {@link List} di {@link Book} ordinata per relevanza:
 * <ul>
 * <li><strong>200 OK:</strong> Ricerca completata, risultati ordinati...</li>
 * <li><strong>400 Bad Request:</strong> Query mancante, vuota o formato non valido</li>
 * ...
 * </ul>
 * @throws IllegalArgumentException se query e null, vuota o contiene solo whitespace
 * @apiNote La ricerca e case-insensitive e supporta matching parziale.
 * @since 1.0.0
 * @see BookService#searchBooks(String)
 */
```

Listing 9.3: Javadoc a livello di metodo per searchBooks in BookController

9.3.2 Esempio Client: `BooksClient.java`

Sul lato client, la classe `BooksClient` dimostra come il Javadoc venga utilizzato per spiegare il ciclo di vita di un'applicazione JavaFX e le interazioni tra i vari componenti dell'interfaccia utente.

```
/**
 * Punto di ingresso principale per l'applicazione client JavaFX.
 * <p>
 * Questa classe estende {@link javafx.application.Application} e funge da
 * coordinatore centrale per l'avvio, la configurazione e la gestione del
 * ciclo di vita dell'applicazione...
 * </p>
 *
 * <h3>Ciclo di vita dell'applicazione:</h3>
 * <ul>
 * <li><strong>init():</strong> Inizializzazione non-grafica...</li>
 * <li><strong>start():</strong> La fase principale in cui viene costruita e mostrata...</li>
 * <li><strong>stop():</strong> Eseguita al momento della chiusura...</li>
 * </ul>
 *
 * <h3>Architettura e integrazioni:</h3>
 * <ul>
 * <li><strong>{@link BookService}:</strong> Gestisce la comunicazione con il backend...</li>
 * <li><strong>{@link MainWindow}:</strong> Costruisce l'intera interfaccia utente...</li>
 * </ul>
 *
 * @author BABO Team
 * @version 1.0
 * @since 1.0
 * @see org.BABO.client.ui.Home.MainWindow
 * @see org.BABO.client.ui.Popup.PopupManager
 */
```

Listing 9.4: Javadoc a livello di classe per BooksClient

Il metodo `start` utilizza una lista ordinata (`<ol>`) per descrivere in modo sequenziale e chiaro il processo di avvio dell'applicazione, rendendo il codice immediatamente comprensibile.

```
/**
 * Punto di ingresso principale dell'applicazione JavaFX.
 * <p>
 * Questo metodo viene chiamato dopo che {@link #init()} e stato completato.
 * E il luogo in cui viene costruita l'intera interfaccia utente.
 * </p>
 * <h4>Processo di avvio:</h4>
 * <ol>
```

```
9   * <li>Registrazione dello stage principale per la protezione
        contro avvii multipli.</li>
10  * <li>Configurazione dell'icona dell'applicazione per la finestra
        e per il sistema.</li>
11  * <li>Creazione dell'istanza di {@link MainWindow} e del suo
        layout.</li>
12  * <li>Inizializzazione del {@link PopupManager} con il root dell'
        applicazione.</li>
13  * <li>Configurazione della scena, del titolo e delle dimensioni
        della finestra.</li>
14  * <li>Setup dei gestori di eventi per la chiusura della finestra
        .</li>
15  * <li>Mostra la finestra all'utente.</li>
16  * </ol>
17  * @param stage Lo stage primario per questa applicazione.
18  */
```

Listing 9.5: Javadoc per il metodo start in BooksClient

In conclusione, l'adozione rigorosa di questi standard di documentazione trasforma il codice sorgente in una risorsa auto-esplicativa, riducendo la curva di apprendimento per i nuovi sviluppatori e semplificando la manutenzione a lungo termine del progetto.

Capitolo 10

Appendici

10.1 Glossario

API Application Programming Interface - Interfaccia per la comunicazione tra componenti software

DTO Data Transfer Object - Oggetto per il trasferimento di dati tra layer

ISBN International Standard Book Number - Codice identificativo univoco per libri

JDBC Java Database Connectivity - API Java per connessione a database

JWT JSON Web Token - Standard per token di autenticazione

MVVM Model-View-ViewModel - Pattern architetturale per UI

REST Representational State Transfer - Architettura per servizi web

SQL Structured Query Language - Linguaggio per database relazionali

10.2 Bibliografia e Riferimenti

- Oracle Corporation. *JavaFX Documentation*. https://openjfx.io/javadoc/21/
- Spring Team. *Spring Boot Reference Documentation*. https://docs.spring.io/spring-boot/docs/3.2.0/reference/htmlsingle/
- PostgreSQL Global Development Group. *PostgreSQL 15 Documentation*. https://www.postgresql.org/docs/15/
- Oracle Corporation. *Java SE 17 Documentation*. https://docs.oracle.com/en/java/javase/17/
- Apache Maven Project. *Maven Documentation*. https://maven.apache.org/guides/
- Square Inc. *OkHttp Documentation*. https://square.github.io/okhttp/
- FasterXML. *Jackson JSON Processor Documentation*. https://github.com/FasterXML/jackson-docs