

# **Book Recommender**

## **Manuale Tecnico**

Ionut Puiu, Hamdi Kebeli, Matteo Ferrario

## Sommario

<b>1. Introduzione</b>	4
<b>1.1 Obiettivi del documento</b>	4
<b>1.2 Ambito del progetto</b>	4
<b>2. Architettura generale dell'applicazione</b>	5
<b>2.1 Panoramica dell'architettura</b>	5
<b>2.2 Componenti principali del sistema</b>	5
<b>2.3 Flusso generale di esecuzione</b>	6
<b>3. Struttura del progetto e organizzazione del codice</b>	7
<b>3.1 Organizzazione generale del progetto</b>	7
<b>3.2 Package principali</b>	8
<b>3.3 Classe BookRecommender e metodo main</b>	8
<b>3.4 Dipendenze tra i componenti</b>	9
<b>4. Gestione della persistenza dei dati</b>	9
<b>4.1 Obiettivo e strategia di persistenza</b>	9
<b>4.2 Organizzazione della cartella data</b>	10
<b>4.3 File Libri.dat</b>	10
<b>4.4 File UtentiRegistrati.dat</b>	11
<b>4.5 File Librerie.dat</b>	11
<b>4.6 File ValutazioniLibri.dat</b>	11
<b>4.7 File ConsigliLibri.dat</b>	12
<b>4.8 Coerenza e gestione dei riferimenti tra file</b>	12
<b>5. Strutture dati utilizzate</b>	12
<b>5.1 Principi generali</b>	12
<b>5.2 Gestione del repository dei libri</b>	13
<b>5.3 Gestione degli utenti registrati</b>	13
<b>5.4 Gestione delle librerie personali</b>	14
<b>5.5 Gestione delle valutazioni</b>	15
<b>5.6 Gestione dei suggerimenti di libri</b>	15
<b>5.7 Coerenza tra le strutture dati</b>	16
<b>6. Scelte algoritmiche</b>	16
<b>6.1 Algoritmi di ricerca dei libri</b>	16
<b>6.2 Calcolo e aggregazione delle valutazioni</b>	17

<b>6.3 Gestione dei suggerimenti di libri</b> .....	18
<b>6.4 Considerazioni sulle prestazioni</b> .....	19
<b>7. Interfaccia utente e integrazione con la logica applicativa</b> .....	19
<b>7.1 Struttura dell'interfaccia utente</b> .....	19
<b>7.2 Separazione tra interfaccia e logica applicativa</b> .....	19
<b>7.3 Gestione degli eventi e delle azioni utente</b> .....	20
<b>7.4 Gestione dello stato dell'utente</b> .....	21
<b>7.5 Benefici dell'approccio adottato</b> .....	21
<b>8. Documentazione del codice (JavaDoc)</b> .....	21
<b>8.1 Utilizzo del JavaDoc</b> .....	21
<b>8.2 Documentazione dei package</b> .....	22
<b>8.3 Documentazione delle classi</b> .....	22
<b>8.4 Documentazione di attributi e metodi</b> .....	22
<b>8.5 Generazione della documentazione</b> .....	23
<b>9. Limiti della soluzione sviluppata</b> .....	23
<b>9.1 Assenza di un database</b> .....	23
<b>9.2 Assenza di accesso concorrente</b> .....	24
<b>9.3 Architettura monolitica</b> .....	24
<b>9.4 Prestazioni legate alla dimensione del dataset</b> .....	24
<b>9.5 Estensioni future</b> .....	25

# 1. Introduzione

## 1.1 Obiettivi del documento

Il presente documento costituisce il **Manuale Tecnico** del progetto **Book Recommender**, sviluppato nell'ambito del corso di **Laboratorio Interdisciplinare A**.

L'obiettivo del manuale è fornire una **descrizione tecnica dettagliata** dell'applicazione, illustrandone:

- l'architettura generale;
- l'organizzazione del codice sorgente;
- le strutture dati adottate;
- le scelte algoritmiche principali;
- la gestione della persistenza dei dati su file;
- le modalità di documentazione del codice tramite JavaDoc.

Il documento è pensato come riferimento per **sviluppatori, manutentori o valutatori tecnici**, e presuppone una conoscenza di base del linguaggio **Java** e dei concetti fondamentali di programmazione orientata agli oggetti.

---

## 1.2 Ambito del progetto

Il progetto **Book Recommender** è un'applicazione **standalone** sviluppata in linguaggio Java, che consente la gestione e la consultazione di un catalogo di libri, offrendo funzionalità di:

- ricerca dei libri;
- gestione di utenti registrati;
- creazione e gestione di librerie personali;
- inserimento e aggregazione di valutazioni;
- suggerimento di libri correlati.

L'applicazione **non utilizza database** né architetture client–server: la persistenza dei dati è realizzata tramite **file di testo strutturati**, caricati all'avvio del programma e aggiornati durante l'esecuzione.

Il Manuale Tecnico si concentra esclusivamente sugli aspetti **progettuali e implementativi** della soluzione adottata, mentre le modalità di utilizzo dell'applicazione dal punto di vista dell'utente finale sono descritte separatamente nel **Manuale Utente**.

## 2. Architettura generale dell'applicazione

### 2.1 Panoramica dell'architettura

L'applicazione **Book Recommender** è stata progettata come **applicazione standalone monolitica**, sviluppata interamente in linguaggio **Java**, senza l'uso di database o architetture distribuite.

L'architettura segue una **separazione logica a livelli**, che distingue chiaramente:

- **la logica di dominio** (libri, utenti, librerie, valutazioni, consigli);
- **la gestione della persistenza dei dati** su file;
- **la logica applicativa** che coordina le operazioni;
- **l'interfaccia utente**, responsabile dell'interazione con l'utente finale.

Questa scelta progettuale consente di:

- mantenere il codice leggibile e manutenibile;
- ridurre l'accoppiamento tra le diverse parti del sistema;
- rispettare i vincoli del progetto, che non prevedono l'utilizzo di database o servizi remoti.

---

### 2.2 Componenti principali del sistema

L'applicazione è organizzata attorno ai seguenti componenti logici principali:

- **Gestione dei libri**  
Responsabile del caricamento del catalogo dei libri dal file di persistenza e

dell'esecuzione delle operazioni di ricerca (per titolo, autore, anno).

- **Gestione degli utenti**

Si occupa della registrazione, autenticazione e gestione degli utenti registrati, inclusa la modifica dei dati dell'account e l'eliminazione dell'utente.

- **Gestione delle librerie personali**

Permette a ciascun utente di creare, rinominare ed eliminare librerie, nonché di aggiungere o rimuovere libri da esse.

- **Gestione delle valutazioni**

Consente agli utenti di inserire valutazioni sui libri presenti nelle proprie librerie e di calcolare le valutazioni aggregate visualizzate agli altri utenti.

- **Gestione dei suggerimenti**

Gestisce l'associazione tra un libro e un insieme limitato di libri consigliati, inseriti dagli utenti.

- **Persistenza dei dati**

Tutti i dati dell'applicazione vengono salvati e recuperati tramite file strutturati, che rappresentano lo stato persistente del sistema.

Ciascun componente è progettato per occuparsi di una responsabilità ben definita, riducendo la duplicazione di logica e facilitando eventuali estensioni future.

---

## 2.3 Flusso generale di esecuzione

All'avvio dell'applicazione, il flusso di esecuzione segue le seguenti fasi principali:

1. **Inizializzazione del sistema**

- caricamento dei file di persistenza dalla cartella **data**;
- creazione delle strutture dati in memoria;
- inizializzazione dei componenti applicativi.

2. **Avvio dell'interfaccia utente**

- visualizzazione della schermata principale;
- messa a disposizione delle funzionalità di ricerca per utenti non autenticati.

3. **Interazione con l'utente**

- gestione delle operazioni di ricerca e consultazione;
- eventuale autenticazione dell'utente;
- accesso alle funzionalità avanzate (librerie, valutazioni, consigli).

#### 4. Aggiornamento dei dati

- ogni modifica effettuata dall'utente (creazione librerie, valutazioni, suggerimenti, modifica account) comporta l'aggiornamento delle strutture dati in memoria;
- i dati vengono successivamente salvati sui file di persistenza.

#### 5. Chiusura dell'applicazione

- il sistema termina mantenendo la coerenza tra lo stato in memoria e quello salvato su file.

Questo flusso garantisce che i dati siano sempre sincronizzati tra memoria e file, senza la necessità di un database o di meccanismi di caching complessi.

## 3. Struttura del progetto e organizzazione del codice

### 3.1 Organizzazione generale del progetto

Il progetto **Book Recommender** è organizzato seguendo una struttura modulare, con l'obiettivo di separare le diverse responsabilità dell'applicazione e rendere il codice più leggibile e manutenibile.

Il codice sorgente è contenuto nella directory `src` ed è organizzato in package Java, mentre i dati persistenti sono collocati nella directory `data`.

Questa separazione consente di distinguere chiaramente:

- **il codice applicativo;**
  - **i dati persistenti;**
  - l'eventuale codice compilato.
-

## 3.2 Package principali

Il progetto utilizza come package principale:

- **bookrecommender**

All'interno di questo package sono definite le classi fondamentali dell'applicazione.

Eventuali sotto-package sono utilizzati per raggruppare classi con responsabilità simili, ad esempio:

- classi di **modello** (libri, utenti, librerie);
- classi di **gestione della persistenza**;
- classi di **supporto alla logica applicativa**;
- classi dell'**interfaccia utente**.

Questa organizzazione favorisce una chiara separazione dei ruoli e riduce l'accoppiamento tra le diverse parti del sistema.

---

## 3.3 Classe **BookRecommender** e metodo **main**

La classe **BookRecommender**, contenuta nel package **book recommender**, rappresenta il **punto di ingresso dell'applicazione**.

All'interno di questa classe è definito il metodo **main**, che ha il compito di:

- inizializzare le strutture dati principali;
- caricare i dati persistenti dai file;
- avviare l'interfaccia utente;
- coordinare l'esecuzione generale dell'applicazione.

La presenza di un unico punto di avvio consente di mantenere il controllo centralizzato del ciclo di vita dell'applicazione.

---

### 3.4 Dipendenze tra i componenti

I componenti dell'applicazione comunicano tra loro attraverso riferimenti diretti alle strutture dati e ai gestori logici, evitando dipendenze circolari.

In particolare:

- l'interfaccia utente invoca i metodi della logica applicativa;
- la logica applicativa utilizza le strutture dati in memoria;
- i moduli di persistenza si occupano esclusivamente della lettura e scrittura dei file.

Questa organizzazione permette di:

- isolare le modifiche a una singola parte del sistema;
- facilitare il debugging;
- migliorare la comprensione del flusso applicativo.

## 4. Gestione della persistenza dei dati

### 4.1 Obiettivo e strategia di persistenza

L'applicazione **Book Recommender** non utilizza un database: la persistenza è realizzata tramite **file locali**.

Questa scelta è coerente con i vincoli del progetto, che richiedono l'uso di file per memorizzare i dati e non prevede componenti client-server o accesso concorrente.

La strategia adottata è la seguente:

- **Caricamento iniziale**: all'avvio vengono letti tutti i file `.dati` e i dati vengono caricati in strutture dati in memoria.
- **Aggiornamento**: durante l'utilizzo, le operazioni dell'utente modificano le strutture in memoria.
- **Salvataggio**: dopo operazioni che alterano lo stato (registrazione utente, creazione libreria, inserimento valutazione/consiglio, ecc.) i dati vengono scritti nuovamente sui file per mantenerne la consistenza.

Questa modalità garantisce:

- semplicità di implementazione;
- coerenza tra stato in memoria e su file;
- portabilità su sistemi diversi.

---

## 4.2 Organizzazione della cartella **data**

Tutti i file di persistenza sono collocati nella directory:

- **data/**

All'interno di questa cartella risiedono i file richiesti dalle specifiche del progetto:

- **Libri.dati**
- **UtentiRegistrati.dati**
- **Librerie.dati**
- **ValutazioniLibri.dati**
- **ConsigliLibri.dati**

La separazione in file distinti consente di:

- isolare le informazioni per tipologia (catalogo, utenti, librerie, valutazioni, consigli);
  - semplificare lettura/scrittura dei dati;
  - ridurre la complessità della gestione delle dipendenze tra entità.
- 

## 4.3 File **Libri.dati**

**Scopo:** contiene il repository iniziale dei libri consultabili dall'applicazione.

Secondo le specifiche, ogni record deve contenere almeno:

- titolo,
- autori,
- anno di pubblicazione,  
con eventuali campi opzionali come editore e categoria

**Utilizzo nel sistema:**

- viene letto all'avvio e caricato in memoria;
  - viene usato come base per tutte le ricerche (titolo/autore/anno);
  - rappresenta la sorgente dati “principale” per la consultazione.
-

## 4.4 File UtentiRegistrati.dat

**Scopo:** memorizza gli utenti registrati al sistema.

Le specifiche richiedono che la registrazione salvi:

- nome e cognome,
- codice fiscale,
- email,
- userid,
- password

**Utilizzo nel sistema:**

- validazione credenziali in fase di login;
  - gestione dati dell'account (es. aggiornamento email/password);
  - supporto alle operazioni che richiedono autenticazione.
- 

## 4.5 File Librerie.dat

**Scopo:** memorizza le librerie personali degli utenti e i libri associati ad esse.

Ogni libreria è collegata a un utente e contiene:

- nome libreria;
- elenco dei libri presenti (riferiti tramite identificativo o chiave del libro).

**Utilizzo nel sistema:**

- permette la creazione e gestione di più librerie per utente (come da specifica)
  - determina l'insieme di libri su cui l'utente può operare (valutare e consigliare);
  - abilita la ricerca filtrata “solo nelle mie librerie”.
- 

## 4.6 File ValutazioniLibri.dat

**Scopo:** memorizza le valutazioni inserite dagli utenti sui libri presenti nelle loro librerie.

Secondo la specifica, ogni valutazione include:

- punteggi (1–5) per i criteri: stile, contenuto, gradevolezza, originalità, edizione;
- recensione testuale opzionale (max 256 caratteri);
- voto finale calcolato come media arrotondata degli altri criteri

#### **Utilizzo nel sistema:**

- consente la visualizzazione delle valutazioni individuali dell'utente (“le mie valutazioni”);
  - permette la costruzione delle statistiche aggregate per ciascun libro (medie, conteggi, commenti).
- 

#### **4.7 File `ConsigliLibri.dat`**

**Scopo:** memorizza i suggerimenti di lettura inseriti dagli utenti.

Per ciascun libro, un utente può associare fino a **3 libri consigliati**

#### **Utilizzo nel sistema:**

- visualizzazione dei consigli in pagina dettaglio libro;
  - aggregazione dei suggerimenti (es. “quanti utenti hanno consigliato quel libro”);
  - gestione dei consigli personali dell'utente (“i miei consigli”).
- 

#### **4.8 Coerenza e gestione dei riferimenti tra file**

Poiché i dati sono distribuiti su più file, è necessario mantenere coerenza tra le entità. In particolare:

- le librerie fanno riferimento ai libri del catalogo;
- valutazioni e consigli sono associati a un utente e a un libro;
- eliminazioni o modifiche dell'account richiedono la gestione dei dati collegati (librerie, valutazioni, consigli).

Per questo motivo l'applicazione gestisce i dati tramite strutture in memoria che collegano le entità tra loro (utente ↔ librerie ↔ libri ↔ valutazioni/consigli), e sincronizza tali strutture sui file di persistenza quando necessario.

### **5. Strutture dati utilizzate**

#### **5.1 Principi generali**

I dati persistenti dell'applicazione **Book Recommender** sono memorizzati su file testuali strutturati in **formato CSV**.

All'avvio dell'applicazione, tali dati vengono **letti dai file** e caricati in **strutture dati in memoria**, che vengono poi utilizzate per l'intera durata dell'esecuzione.

Questa scelta consente di:

- separare la logica applicativa dalla persistenza;
  - lavorare in memoria per migliorare le prestazioni;
  - semplificare la gestione delle operazioni di ricerca, aggiornamento e aggregazione.
- 

## 5.2 Gestione del repository dei libri

Il contenuto del file `Libri.dati` viene caricato in una struttura dati di tipo **lista**, che contiene tutti i libri disponibili nel catalogo.

Tipicamente viene utilizzata una struttura del tipo:

- `List<Libro>`

Ogni elemento rappresenta un singolo libro e contiene le informazioni principali (titolo, autori, anno, eventuali campi opzionali).

Per supportare in modo efficiente le operazioni di ricerca:

- la lista viene attraversata per le ricerche basate su **sottostringhe** (titolo e autore);
- le ricerche sono implementate in modo **case-insensitive**.

Questa scelta è adeguata considerando la dimensione prevista del dataset e i vincoli del progetto.

---

## 5.3 Gestione degli utenti registrati

I dati contenuti nel file `UtentiRegistrati.dati` vengono caricati in una struttura dati che consente un accesso rapido agli utenti in fase di autenticazione.

Tipicamente viene utilizzata:

- una `Map<String, Utente>`, dove la chiave è lo **username**

Questa struttura permette:

- verifica rapida delle credenziali durante il login;
  - accesso diretto ai dati dell'utente autenticato;
  - gestione efficiente delle operazioni di modifica ed eliminazione dell'account.
- 

## 5.4 Gestione delle librerie personali

Le librerie personali degli utenti, memorizzate nel file `Librerie.dat`, vengono rappresentate in memoria tramite strutture che associano:

- un utente;
- un insieme di librerie;
- l'elenco dei libri contenuti in ciascuna libreria.

Una possibile organizzazione logica è:

- una `Map<Utente, List<Libreria>>`
- oppure una `Map<String, List<Libreria>>`, indicizzata per username

Ogni libreria contiene:

- un nome identificativo;
- una collezione di riferimenti ai libri presenti nel catalogo.

Questa struttura consente di:

- gestire più librerie per ciascun utente;
  - limitare le operazioni di valutazione e suggerimento ai soli libri presenti nelle librerie personali.
-

## 5.5 Gestione delle valutazioni

Le valutazioni dei libri, lette dal file `ValutazioniLibri.dat`, vengono caricate in strutture che associano:

- un utente;
- un libro;
- l'insieme dei punteggi assegnati ai criteri di valutazione.

In memoria, le valutazioni possono essere gestite tramite:

- una lista di valutazioni (`List<Valutazione>`);
- oppure una mappa indicizzata per libro (`Map<Libro, List<Valutazione>>`) per facilitare il calcolo delle statistiche aggregate.

Questa organizzazione permette:

- il calcolo delle medie per ciascun criterio;
  - la costruzione delle valutazioni aggregate visualizzate agli utenti;
  - la visualizzazione delle valutazioni personali di un singolo utente.
- 

## 5.6 Gestione dei suggerimenti di libri

I suggerimenti memorizzati nel file `ConsigliLibri.dat` vengono caricati in strutture che associano:

- un libro principale;
- un insieme limitato di libri consigliati;
- l'utente che ha inserito il suggerimento.

Una possibile struttura logica è:

- una `Map<Libro, List<Consiglio>>`

Ogni suggerimento contiene fino a **tre libri consigliati**, come richiesto dalle specifiche del progetto.

Questa struttura consente di:

- visualizzare rapidamente i consigli associati a un libro;
  - calcolare il numero di utenti che hanno suggerito un determinato libro;
  - gestire l'elenco dei consigli inseriti da un singolo utente.
- 

## 5.7 Coerenza tra le strutture dati

Le strutture dati in memoria sono progettate per mantenere la coerenza tra:

- utenti;
- librerie;
- libri;
- valutazioni;
- suggerimenti.

Le associazioni tra le entità vengono mantenute tramite riferimenti logici (ad esempio identificativi o riferimenti a oggetti), evitando duplicazioni di dati e garantendo consistenza durante le operazioni di aggiornamento e salvataggio su file.

# 6. Scelte algoritmiche

## 6.1 Algoritmi di ricerca dei libri

L'applicazione **Book Recommender** supporta diverse modalità di ricerca dei libri, come richiesto dalle specifiche di progetto:

- ricerca per titolo;
- ricerca per autore;

- ricerca per autore e anno di pubblicazione.

Le ricerche vengono eseguite sul catalogo dei libri caricato in memoria a partire dal file **Libri.dat**.

L'algoritmo di ricerca segue i seguenti principi:

- **ricerca lineare** sull'elenco dei libri;
- confronto basato su **sottestringhe**;
- confronto **non case-sensitive**, ottenuto normalizzando le stringhe (ad esempio convertendo in minuscolo).

Per ogni libro del catalogo:

- il campo di interesse (titolo, autore o entrambi) viene confrontato con il testo inserito dall'utente;
- il libro viene incluso nei risultati se la sottostringa ricercata è contenuta nel campo considerato.

Questa soluzione, pur avendo complessità lineare, è adeguata alle dimensioni del dataset previste e garantisce semplicità e chiarezza implementativa.

---

## 6.2 Calcolo e aggregazione delle valutazioni

Le valutazioni dei libri sono basate su cinque criteri:

- stile;
- contenuto;
- gradevolezza;
- originalità;
- edizione.

Per ogni valutazione inserita:

- i punteggi assegnati ai singoli criteri sono valori interi compresi tra 1 e 5;

- il **voto finale** viene calcolato come **media aritmetica arrotondata** dei punteggi dei criteri.

Per la visualizzazione delle valutazioni aggregate di un libro, l'algoritmo:

1. recupera tutte le valutazioni associate al libro selezionato;
2. calcola la media dei punteggi per ciascun criterio;
3. calcola il voto finale aggregato;
4. raccoglie le eventuali recensioni testuali.

Nel caso in cui un libro non presenti valutazioni, il sistema segnala l'assenza di dati, evitando la visualizzazione di informazioni incomplete o fuorvianti.

---

### **6.3 Gestione dei suggerimenti di libri**

Ogni utente registrato può suggerire fino a **tre libri correlati** per un libro presente nelle proprie librerie personali.

L'algoritmo di gestione dei suggerimenti prevede che:

- i libri suggeriti siano selezionati esclusivamente tra quelli presenti nelle librerie dell'utente;
- non sia possibile superare il limite massimo di tre suggerimenti per ciascun libro e utente.

Per la visualizzazione dei suggerimenti associati a un libro, il sistema:

1. recupera tutti i suggerimenti inseriti dagli utenti per il libro selezionato;
2. raggruppa i libri consigliati;
3. calcola il numero di utenti che hanno suggerito ciascun libro.

Questo approccio consente di fornire all'utente una visione aggregata dei consigli di lettura più rilevanti.

---

## 6.4 Considerazioni sulle prestazioni

Le scelte algoritmiche adottate privilegiano:

- semplicità;
- chiarezza;
- coerenza con i vincoli del progetto.

Poiché l'applicazione opera su dataset di dimensioni limitate e non prevede accessi concorrenti, algoritmi di complessità lineare risultano adeguati e non impattano negativamente sull'esperienza d'uso.

Eventuali ottimizzazioni (indicizzazione avanzata, caching, strutture dati più complesse) sono rimandate a possibili evoluzioni future del progetto.

## 7. Interfaccia utente e integrazione con la logica applicativa

### 7.1 Struttura dell'interfaccia utente

L'applicazione **Book Recommender** utilizza un'interfaccia grafica che consente all'utente di interagire con il sistema in modo intuitivo e guidato.

L'interfaccia è responsabile esclusivamente della:

- raccolta degli input dell'utente;
- visualizzazione dei risultati delle operazioni;
- gestione della navigazione tra le diverse funzionalità.

Non contiene logica di business complessa, che è invece demandata ai componenti applicativi sottostanti.

---

### 7.2 Separazione tra interfaccia e logica applicativa

Dal punto di vista architettonico, l'interfaccia utente è progettata come **livello di presentazione**, separato dalla logica applicativa.

In particolare:

- l'interfaccia utente invoca metodi esposti dalla logica applicativa per eseguire operazioni come ricerca, registrazione, login, gestione librerie, valutazioni e suggerimenti;
- la logica applicativa restituisce i risultati sotto forma di dati strutturati, che l'interfaccia si limita a visualizzare;
- l'accesso diretto ai file di persistenza non avviene mai dall'interfaccia utente.

Questa separazione consente di:

- ridurre l'accoppiamento tra UI e logica;
  - semplificare la manutenzione del codice;
  - facilitare eventuali modifiche future dell'interfaccia senza impattare il funzionamento interno del sistema.
- 

### 7.3 Gestione degli eventi e delle azioni utente

Le azioni dell'utente (ad esempio clic su pulsanti, selezione di libri, inserimento di dati) generano eventi che vengono intercettati dall'interfaccia utente.

Per ciascuna azione:

1. l'interfaccia valida i dati di input (ad esempio campi obbligatori o limiti di lunghezza);
2. viene invocato il metodo corrispondente nella logica applicativa;
3. il risultato dell'operazione viene restituito all'interfaccia;
4. l'interfaccia aggiorna la visualizzazione in base all'esito dell'operazione.

Questo flusso garantisce un comportamento coerente dell'applicazione e una gestione centralizzata delle regole di funzionamento.

---

## 7.4 Gestione dello stato dell'utente

Lo stato dell'utente (non autenticato / autenticato) viene mantenuto dalla logica applicativa e utilizzato dall'interfaccia per:

- abilitare o disabilitare determinate funzionalità;
- mostrare o nascondere opzioni riservate agli utenti registrati;
- garantire che operazioni sensibili (valutazioni, suggerimenti, gestione librerie) siano eseguibili solo dopo autenticazione.

L'interfaccia si limita a riflettere lo stato corrente dell'utente, senza gestirne direttamente la persistenza.

---

## 7.5 Benefici dell'approccio adottato

L'integrazione tra interfaccia utente e logica applicativa è stata progettata per ottenere:

- chiarezza nella distribuzione delle responsabilità;
- riduzione degli errori dovuti a logica duplicata;
- maggiore facilità di estensione futura dell'applicazione.

Questo approccio è coerente con i principi della programmazione orientata agli oggetti e con gli obiettivi formativi del Laboratorio Interdisciplinare A.

# 8. Documentazione del codice (JavaDoc)

## 8.1 Utilizzo del JavaDoc

Il progetto **Book Recommender** utilizza lo strumento **JavaDoc** per la documentazione del codice sorgente, come richiesto dalle indicazioni del corso.

La documentazione JavaDoc è considerata **parte integrante del Manuale Tecnico** e accompagna il codice sorgente dell'applicazione

Il JavaDoc è stato utilizzato per documentare:

- i **package** principali del progetto;
- le **classi**;
- i **metodi pubblici**;
- gli **attributi significativi**.

I commenti JavaDoc sono inseriti direttamente nel codice sorgente, utilizzando la sintassi standard `/** ... */`.

---

## 8.2 Documentazione dei package

Ogni package principale dell'applicazione è corredata da una documentazione generale, che descrive:

- il ruolo del package all'interno dell'architettura;
- le responsabilità delle classi in esso contenute.

La documentazione dei package è fornita tramite il file `package-info.java`, come previsto dalle linee guida JavaDoc

---

## 8.3 Documentazione delle classi

Per ogni classe significativa del progetto, il JavaDoc descrive:

- lo **scopo della classe**;
- il suo ruolo nel sistema;
- le principali responsabilità funzionali.

Al termine della descrizione di ciascuna classe è presente il tag:

- `@author`

contenente il nome dell'autore o degli autori del codice, come richiesto dalle specifiche di documentazione.

---

## 8.4 Documentazione di attributi e metodi

Gli attributi principali delle classi sono documentati indicando:

- il significato dell'attributo;
- il tipo di informazione rappresentata;
- il suo utilizzo all'interno della classe.

Per ogni metodo documentato, il JavaDoc include:

- una descrizione sintetica della funzionalità;
- il tag `@param` per ciascun parametro, rispettando l'ordine di dichiarazione;
- il tag `@return` per descrivere il valore restituito, se presente;
- il tag `@throws` per documentare le eventuali eccezioni sollevate.

Questa documentazione consente di comprendere il comportamento dei metodi senza dover analizzare direttamente il codice sorgente.

---

## 8.5 Generazione della documentazione

La documentazione JavaDoc può essere generata automaticamente utilizzando il comando `javadoc`, incluso nella distribuzione del **Java Development Kit (JDK)**.

Il risultato è una documentazione consultabile in formato HTML, utile come supporto alla manutenzione e all'estensione futura del progetto.

# 9. Limiti della soluzione sviluppata

La soluzione sviluppata per il progetto **Book Recommender** rispetta pienamente i requisiti e i vincoli previsti per il **Laboratorio Interdisciplinare A**. Tuttavia, presenta alcuni limiti intrinseci, legati sia alle scelte progettuali adottate sia agli obiettivi formativi del corso.

## 9.1 Assenza di un database

L'applicazione non utilizza un database relazionale o non relazionale.

La persistenza dei dati è realizzata esclusivamente tramite **file CSV**, come richiesto dalle specifiche del progetto.

Questa scelta comporta:

- minore efficienza nella gestione di grandi volumi di dati;
  - necessità di caricare i dati in memoria all'avvio dell'applicazione;
  - assenza di query complesse e di meccanismi avanzati di indicizzazione.
-

## 9.2 Assenza di accesso concorrente

L'applicazione non gestisce accessi concorrenti ai dati.

È progettata per essere utilizzata da **un solo utente alla volta**, in un contesto locale.

Non sono previsti:

- meccanismi di locking;
- sincronizzazione tra thread;
- gestione di conflitti di accesso ai file.

Questo limite è coerente con l'assenza di architetture distribuite o client–server, che saranno oggetto del Laboratorio Interdisciplinare B.

---

## 9.3 Architettura monolitica

Il sistema è realizzato come applicazione **monolitica**, in cui:

- interfaccia utente;
- logica applicativa;
- gestione della persistenza

sono contenute all'interno dello stesso progetto.

Sebbene questa scelta semplifichi lo sviluppo e la comprensione del codice, limita la scalabilità e la possibilità di riutilizzo dei componenti in contesti più complessi.

---

## 9.4 Prestazioni legate alla dimensione del dataset

Le operazioni di ricerca e aggregazione sono basate su algoritmi a complessità lineare, che risultano adeguati per dataset di dimensioni contenute.

Con un numero molto elevato di libri, utenti o valutazioni:

- i tempi di risposta potrebbero aumentare;
- sarebbe necessario introdurre strutture dati più complesse o meccanismi di indicizzazione.

---

## 9.5 Estensioni future

I limiti descritti rappresentano anche **potenziali punti di estensione futura**, tra cui:

- introduzione di un database per la persistenza dei dati;
- supporto a un'architettura client–server;
- gestione di accessi concorrenti;
- ottimizzazione degli algoritmi di ricerca e aggregazione.

Tali evoluzioni esulano dagli obiettivi del Laboratorio Interdisciplinare A e sono coerenti con i contenuti previsti nei corsi successivi.