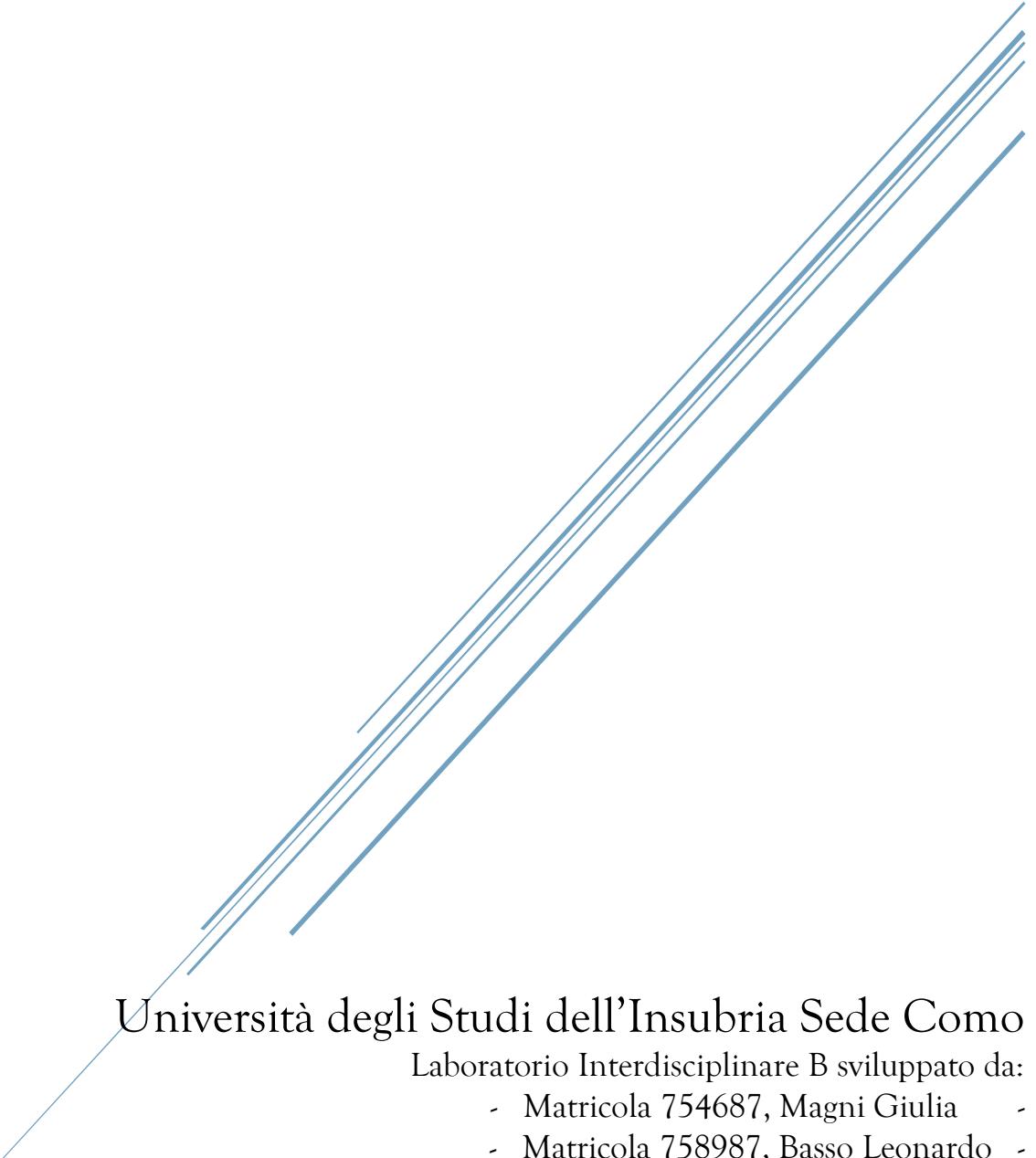


# BOOK RECOMMENDER

MANUALE TECNICO

versione 2.0.B | Data: luglio 2025



Università degli Studi dell'Insubria Sede Como

Laboratorio Interdisciplinare B sviluppato da:

- Matricola 754687, Magni Giulia -
- Matricola 758987, Basso Leonardo -
- Matricola 754625, Beretta Lorenzo -

# Indice

---

<b>1. Introduzione al progetto .....</b>	<b>2</b>
<b>2. Requisiti di sistema e ambientali .....</b>	<b>2</b>
2.1 Requisiti hardware .....	2
2.2 Requisiti software .....	2
2.3 Dipendenze esterne principali.....	3
<b>3. Installazione del sistema.....</b>	<b>3</b>
3.1 Preparazione dell'ambiente .....	3
3.2 Installazione programma .....	4
❖ Frontend .....	4
❖ Backend & DB.....	5
3.3 Risoluzione possibili errori.....	5
<b>4. Esecuzione e utilizzo del sistema.....</b>	<b>6</b>
4.1 Navigazione utente .....	6
4.3 Interfaccia utente.....	6
<b>5. Limiti della soluzione sviluppata .....</b>	<b>7</b>
5.1 Limiti funzionali.....	7
5.2 Limiti tecnici .....	7
<b>6. Progettazione tecnica e architetturale.....</b>	<b>8</b>
6.1 Architettura generale.....	8
6.2 Diagrammi UML.....	10
Diagramma delle classi (Class Diagram).....	10
Diagramma di sequenza .....	11
Diagramma di stato (semplificato).....	11
6.3 Pattern progettuali utilizzati .....	12
<b>7. Strutture dati e scelte algoritmiche.....</b>	<b>13</b>
7.1 Strutture dati principali.....	13
<b>8. Progettazione e documentazione del database .....</b>	<b>13</b>
8.1 Modello concettuale ER.....	13
8.2 Schema relazionale .....	14
8.3 Relazioni tra tabelle .....	14
<b>9. Sitografia e bibliografia .....</b>	<b>15</b>
9.1 Documentazione Interna .....	15

# 1. Introduzione al progetto

Book Recommender è un'applicazione client/server pensata per offrire un sistema di consultazione, valutazione e raccomandazione di libri in ambiente desktop.

Il sistema consente agli utenti di:

- ❖ Ricercare libri in base a criteri come titolo, autore e anno.
- ❖ Visualizzare informazioni dettagliate e valutazioni associate ai titoli, con copertine prese automaticamente dall'API di Google Books.
- ❖ Registrarsi come utenti per gestire una libreria personale.
- ❖ Inserire valutazioni secondo criteri multipli (stile, contenuto, originalità, ecc.).
- ❖ Suggerire libri correlati fino a un massimo di tre per titolo.

L'applicazione è stata realizzata come progetto accademico per il corso “Laboratorio Interdisciplinare B” presso l’Università degli Studi dell’Insubria (Sede Como), con l’obiettivo di combinare competenze di ingegneria del software, progettazione database e sviluppo frontend.

Il software è distribuito come applicazione desktop multipiattaforma, con un’architettura che prevede:

- ❖ Backend Java basato su Javalin, con accesso ai dati tramite JDBC/PostgreSQL.
- ❖ Frontend costruito con Vue.js e compilato tramite Tauri.

---

## 2. Requisiti di sistema e ambientali

### 2.1 Requisiti hardware

- ❖ **RAM:** almeno 4 GB (consigliati 8 GB per ambienti di sviluppo)
- ❖ **Processore:** x86\_64 Dual-Core o superiore
- ❖ **Spazio disco:** minimo 500 MB per codice, librerie e dataset
- ❖ **Risoluzione schermo:** 1280x720 o superiore per la UI desktop

### 2.2 Requisiti software

- ❖ **Sistema operativo:** Windows 10+, Linux (kernel  $\geq$  4.15)
- ❖ **Java Development Kit (JDK):** versione 17 o compatibile
- ❖ **Database Management System (DBMS):** PostgreSQL 13 o superiore
- ❖ **Strumenti di build:** Apache Maven  $\geq$  3.6 o Apache Ant
- ❖ **Node.js:**  $\geq$  18 (necessario per compilazione frontend Vue/Tauri)
- ❖ **Rust:**  $\geq$  1.70 per il backend Tauri
- ❖ **Docker**

## 2.3 Dipendenze esterne principali

Componente	Versione minima	Scopo
PostgreSQL	13	Archiviazione dati persistenti
Apache Maven	3.6	Build ed esecuzione backend
Javalin	5.0	Webserver Java leggero
Vue.js	3	Interfaccia utente frontend
Tauri	2	Wrapper desktop multipiattaforma
JDBC Driver	PostgreSQL 42.6+	Connessione Java ↔ DB
Argon2	2	Criptaggio delle password

---

## 3. Installazione del sistema

### 3.1 Preparazione dell'ambiente

Per installare e avviare correttamente Book Recommender, assicurarsi che tutti gli strumenti richiesti siano installati:

1. **Installare Java JDK 17**

Scaricabile da <https://jdk.java.net/17/>

2. **Installare Apache Maven**

Scaricabile da <https://maven.apache.org/download.cgi>

Verificare l'installazione con: mvn -v

3. **Installare Node.js e Rust**

Per Node.js ≥ 18: [Node.js – Run JavaScript Everywhere](#)

Per Rust ≥ 1.70: [Rust Programming Language](#)

4. **Installare Docker** per eseguire tutto in contenitori isolati

Docker Desktop (Windows)

Docker Engine (Linux)

5. (Consigliato) Installare un IDE Java come **IntelliJ IDEA**, **WebStorm**, **Eclipse**, **VS Code** o altri.

Maggiori informazioni sul processo di installazione sono riportate sul file README.md nella repository GitHub del progetto oppure proseguire nel paragrafo successivo:

#### 3.2 Installazione programma.

## 3.2 Installazione programma

### ❖ Frontend

Il programma è in Developer Mode ed all'interno del progetto si possono trovare già presenti gli eseguibili nella cartella /bin ove l'eseguibile .deb non è stato testato, fare attenzione e provare ad utilizzare sia install.sh che .deb.

Installare le dipendenze del programma:

#### ♦ Linux:

##### ◊ Debian/Ubuntu:

```
sudo apt update  
sudo apt install libwebkit2gtk-4.0-dev \  
    build-essential \  
    curl \  
    wget \  
    file \  
    libssl-dev \  
    libgtk-3-dev \  
    libayatana-appindicator3-dev \  
    librsvg2-dev
```

##### ◊ Arch:

```
sudo pacman -Syu  
sudo pacman -S --needed \  
    webkit2gtk \  
    base-devel \  
    curl \  
    wget \  
    file \  
    openssl \  
    apmenu-gtk-module \  
    gtk3 \  
    libappindicator-gtk3 \  
    librsvg \  
    libvips
```

##### ◊ Fedora:

```
sudo dnf check-update  
sudo dnf install webkit2gtk4.0-devel \  
    openssl-devel \  
    curl \  
    wget \  
    file \  
    libappindicator-gtk3-devel \  
    librsvg2-devel  
sudo dnf group install "C Development  
Tools and Libraries"
```

##### ◊ OpenSuse:

```
sudo zypper up  
sudo zypper in webkit2gtk3-soup2-devel \  
    libopenssl-devel \  
    curl \  
    wget \  
    file \  
    libappindicator3-1 \  
    librsvg-devel  
sudo zypper in -t pattern devel_basis
```

È possibile installare Rust con:

```
curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh
```

#### ♦ Windows:

È necessario installare i Build Tools di C++ e le webview2 di rust con:

```
winget install --id RustLang.Rustup  
rustup default stable-msvc
```

Per far funzionare la prima volta il programma usare:

```
cd frontend  
npm install
```

## || PRIMO AVVIO ||

#### ❑ Importante:

È necessario avere Node installato sul proprio computer.

Per far partire il programma usare: npx tauri dev

## ❖ Backend & DB

||PRIMO AVVIO||

□ Importante:

È necessario avere Docker installato sul proprio computer.

Prima di utilizzare Docker è necessario generare il jar andando nella cartella di backend ed eseguendo: `mvn clean package` oppure si può fare anche graficamente da un IDE. Su Linux (SystemD) è necessario avviare Docker con: `sudo systemctl start docker`

Su Windows tutti i comandi citati vanno eseguiti da amministratore.

Per avviare il progetto usare nel terminale:

◆ **Linux:**

```
sudo docker compose up --build
```

◆ **Windows (Powershell):**

```
docker-compose up --build
```

Per interrompere Docker è necessario scrivere:

◆ **Linux:**

```
sudo docker compose down
```

◆ **Windows (Powershell):**

```
docker-compose down
```

### 3.3 Risoluzione possibili errori

❖ Se vengono apportati cambiamenti al progetto non rilevati da docker, usare:

◆ **Linux:**

```
sudo docker system prune -a  
sudo docker compose up --build
```

◆ **Windows (Powershell):**

```
docker system prune -a  
docker-compose up --build
```

❖ Se si hanno problemi con le dipendenze di Tauri con errore:

`%1 non è un'applicazione di Win32 valida. (os error 193)`

Per risolvere seguire passo-passo i seguenti passaggi:

1. Verifica toolchain attiva `rustup show` ti dirà il target predefinito.

2. Se non è `x86_64-pc-windows-msvc`, esegui:

```
rustup default stable-x86_64-pc-windows-msvc
```

3. Spostati nella cartella `src-tauri`:

```
cd .\frontend\src-tauri
```

4. Esegui il clean di Cargo:

```
cargo clean
```

5. Torna alla cartella principale del frontend e rilancia Tauri:

```
cd ..\..  
npx tauri dev
```

❖ Se non cambia nulla provare a spegnere e riaccendere il computer.

# 4. Esecuzione e utilizzo del sistema

## 4.1 Navigazione utente

All'avvio del programma, viene mostrata una schermata iniziale (1) con:

- ❖ Barra di ricerca per titoli, autori e anno
- ❖ Elenco di libri presenti nell'applicazione
- ❖ Accesso alla registrazione e al login

Book Recommender				
Titolo				
Autore	Anno			
Il Grande Gatsby	F. Scott Fitzgerald	1925	Roman...	
1984	George Orwell	1949	Fantasc...	
Orgoglio e pregiudizio	Jane Austen	1813	Roman...	
Il Signore degli Anelli	J.R.R. Tolkien	1954	Fantasy	
Cime tempestose	Emily Brontë	1847	Roman...	
Il vecchio e il mare	Ernest Hemingway	1952	Roman...	
Guerra e Pace	Lev Tolstoj	1869	Storico	
Il codice Da Vinci	Dan Brown	2003	Thriller	
Lo Hobbit	J.R.R. Tolkien	1937	Fantasy	
Il nome della rosa	Umberto Eco	1980	Roman...	
La Divine Comédie				
Dante Alighieri				
1320 - Parigi				

Dalla schermata home si potrà procedere come:

1. **Utente generico non registrato o non autenticato** che può:
  - ❖ Cercare libri per **titolo, autore o anno**
  - ❖ Visualizzare informazioni di dettaglio su un libro a piacere
  - ❖ Consultare le **valutazioni aggregate** (media multiparametrica)
  - ❖ Esplorare i **consigli di lettura** proposti da altri utenti
  - ❖ Registrarsi tramite l'apposita funzione per ottenere funzionalità avanzate
2. **Utente registrato** viene richiesto il **login** e dopo aver inserito le credenziali valide (userid + password), l'utente ha accesso a:
  - ❖ Gestione **librerie personali**
    - ◊ Creazione nuove librerie
    - ◊ Aggiunta libri
  - ❖ Inserimento **valutazioni dettagliate**
    - ◊ Valutare un libro secondo 5 criteri (stile, contenuto, gradevolezza, originalità, edizione)
    - ◊ Inserire commenti testuali
  - ❖ Inserimento **consigli di lettura**
    - ◊ Associare fino a 3 libri suggeriti a ogni libro letto

## 4.3 Interfaccia utente

L'interfaccia desktop è basata su Vue.js e comprende:

- ❖ Schermata menu principale (ricerca, elenco libri)
- ❖ Schermata registrazione o login
- ❖ Schermata utente con informazioni, bottone di logout e librerie
- ❖ Schermata singolo libro con possibilità di valutazione, inserimento in libreria e consigli con copertine prese automaticamente dall'API di Google Books.

## 5. Limiti della soluzione sviluppata

Durante lo sviluppo del sistema Book Recommender, alcune scelte progettuali e vincoli tecnici hanno portato alla definizione di limiti funzionali e architetturali, elencati di seguito.

### 5.1 Limiti funzionali

- ❖ **Assenza di recupero password**

Non è previsto un meccanismo automatico per la reimpostazione delle credenziali utente in caso di smarrimento.

- ❖ **Nessuna gestione avanzata dei ruoli**

Il sistema distingue solo tra utente registrato e non registrato, senza livelli di accesso differenziati (es. admin, moderatore).

### 5.2 Limiti tecnici

- ❖ **Assenza di caching**

Nessun layer di cache è previsto tra database e backend: tutte le query vengono eseguite live.

- ❖ **Backup manuale**

Non è previsto un sistema automatizzato di backup o recovery.

- ❖ Il Software utilizza una versione di node-sass che supporta il tag @import, che in futuro verrà rimosso.

Controllare di star usando una versione compatibile della versione node-sass.

---

# 6. Progettazione tecnica e architetturale

## 6.1 Architettura generale

Book Recommender adotta un'architettura **client/server**, dove i due moduli principali interagiscono tramite chiamate HTTP RESTful.

### ❖ Client: (2)

Applicazione desktop Vue.js impacchettata tramite Tauri.

Si occupa dell'interazione con l'utente, gestione interfaccia grafica e invio richieste al backend.

Tauri garantisce un eseguibile performante usando una webview di Rust per renderizzare la UI.

Mentre, Vue.js garantisce un sistema reattivo tramite il pattern observer implementato di default dal framework, permettendo una gestione dinamica della UI senza necessità di fare reload per mostrare cambiamenti.

### ❖ Server:

Applicazione backend Java basata su Javalin.

Espone endpoint dell'API per tutte le operazioni (file Main.java (3)), gestisce le operazioni SQL tramite JDBC.

Il server è ospitato su Docker (la piattaforma che gestisce e distribuisce il server), il quale legge il file app.jar generato da Maven.

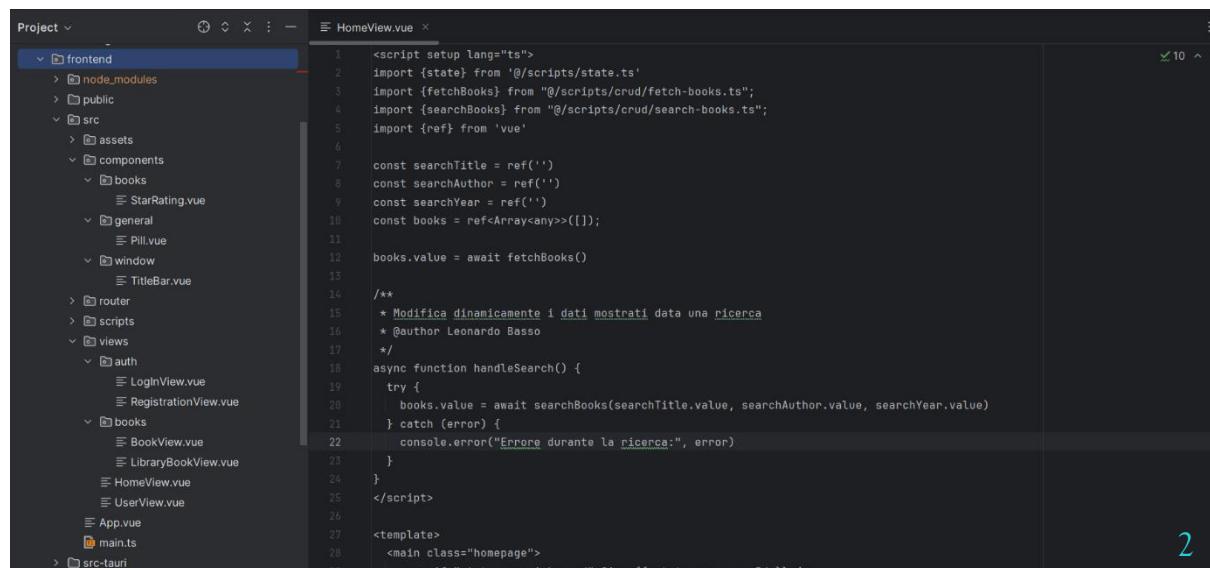
### ❖ Database:

Istanza PostgreSQL locale (4), contenente tutte le informazioni persistenti su libri, utenti, librerie, valutazioni e suggerimenti.

La comunicazione tra client e server avviene tramite protocollo HTTP, e la persistenza dei dati è garantita tramite connessione JDBC.

---

Esempio di come i dati e le responsabilità vengono suddivise da diverse classi:



The screenshot shows a code editor with the file 'HomeView.vue' open. The left sidebar displays the project structure under the 'frontend' folder, including 'node\_modules', 'public', 'src' (with 'assets', 'components' containing 'StarRating.vue', 'books' containing 'StarRating.vue', 'general' containing 'Pill.vue', 'window' containing 'TitleBar.vue'), 'router', 'scripts', 'views' (with 'auth' containing 'LoginView.vue', 'RegistrationView.vue', 'books' containing 'BookView.vue', 'LibraryBookView.vue', 'HomeView.vue', 'UserView.vue', 'App.vue'), and 'main.ts'. The right pane shows the code for 'HomeView.vue':

```
<script setup lang="ts">
import {state} from '@scripts/state.ts'
import {fetchBooks} from "@scripts/crud/fetch-books.ts";
import {searchBooks} from "@scripts/crud/search-books.ts";
import {ref} from 'vue'

const searchTitle = ref('')
const searchAuthor = ref('')
const searchYear = ref('')
const books = ref<Array<any>>([]);

books.value = await fetchBooks()

/**
 * Modifica dinamicamente i dati mostrati data una ricerca
 * @author Leonardo Basso
 */
async function handleSearch() {
  try {
    books.value = await searchBooks(searchTitle.value, searchAuthor.value, searchYear.value)
  } catch (error) {
    console.error("Errore durante la ricerca:", error)
  }
}

</script>

<template>
<main class="homepage">
  <n v-if="state.user.isLoggedIn">Ciao {{ state.user.userId }}</n>
</main>
</template>
```

```

1 public static void main(String[] args) {
2     var app = Javalin.create(config -> {
3         config.bundledPlugins.enableCors(cors -> {
4             cors.addRule(CorsPluginConfig.CorsRule::anyHost);
5         });
6     }).start(7070);
7
8     // api/book
9     app.get("/api/book/all", BookController::getAllBooks);
10    app.get("/api/book/search", BookController::searchBook);
11    app.get("/api/book/{id}", BookController::getSingleBook);
12
13    // api/book/suggestion
14    app.post("/api/book/suggestion/create", BookController::insertSuggestion);
15    app.get("/api/book/suggestion/get/all/{id}", BookController::getLibriConsigliatiLibri);
16    app.post("/api/book/suggestion/get/user", BookController::getLibriConsigliatiUtenteLibri);
17
18    // api/user
19    app.post("/api/user/register", UserController::register);
20    app.post("/api/user/login", UserController::login);
21
22    // api/library
23    app.post("/api/library/create", LibraryController::createLibrary);
24    app.post("/api/library/add/book", LibraryController::addBook);
25    app.get("/api/library/details/{id}", LibraryController::details);
26    app.get("/api/library/user/{id}", LibraryController::getLibrariesByUser);
27    app.get("/api/library/books-user/{id}", LibraryController::getAllBooksByUser);
28
29    // api/review
30    app.get("/api/review/book/{id}", ReviewController::getReviewByBook);
31    app.post("/api/review/create", ReviewController::insertReview);
32 }

```

3

```

Project ▾
  bookrecommender sources root, C:\Users\giulio...
    > idea
    > backend
      > idea
      > META-INF
      > src.main.java.com.bookrecommender
        > controller
        > jdbc
        > model
        > utils
          Main
          package-info.java
      > target
      .gitignore
      dependency-reduced-pom.xml
      Dockerfile
      main
      pom.xml
      user
    > db.sql
      CreateDB.sql
      PopulateDB.sql
    > doc.diagrams
    > frontend
    .gitattributes
    .gitignore
    docker-compose.yml
    LICENSE
    README.md

```

```

CREATE TABLE Libro
(
    id           INT PRIMARY KEY,
    Nome         VARCHAR(255) NOT NULL,
    Autore       VARCHAR(255) NOT NULL,
    Descrizione  VARCHAR(255),
    Categoria   VARCHAR(255),
    Publisher   VARCHAR(255) NOT NULL,
    Prezzo       DECIMAL(10, 4) NOT NULL,
    MesePub     VARCHAR(20) NOT NULL,
    AnnoPub     INT          NOT NULL
);

CREATE TABLE Utente
(
    UserId      VARCHAR(255) PRIMARY KEY,
    Password    VARCHAR(255) NOT NULL,
    Nome        VARCHAR(255) NOT NULL,
    Cognome     VARCHAR(255) NOT NULL,
    Taxcode     VARCHAR(255) UNIQUE CHECK ( Utente.Taxcode ~ '^[A-Z]{6}[0-9]{2}[A-Z]{2}[0-9]{2}[A-Z]{2}[0-9]{3}[A-Z]$' ),
    Email       VARCHAR(255) UNIQUE CHECK ( Utente.Email ~ '^+[A-Za-z0-9._+%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$' )
);

CREATE TABLE Libreria
(
    LibreriaId  SERIAL PRIMARY KEY,
    NomeLibreria VARCHAR(255) UNIQUE NOT NULL,
    UserId       VARCHAR(255) REFERENCES Utente (UserId)
);

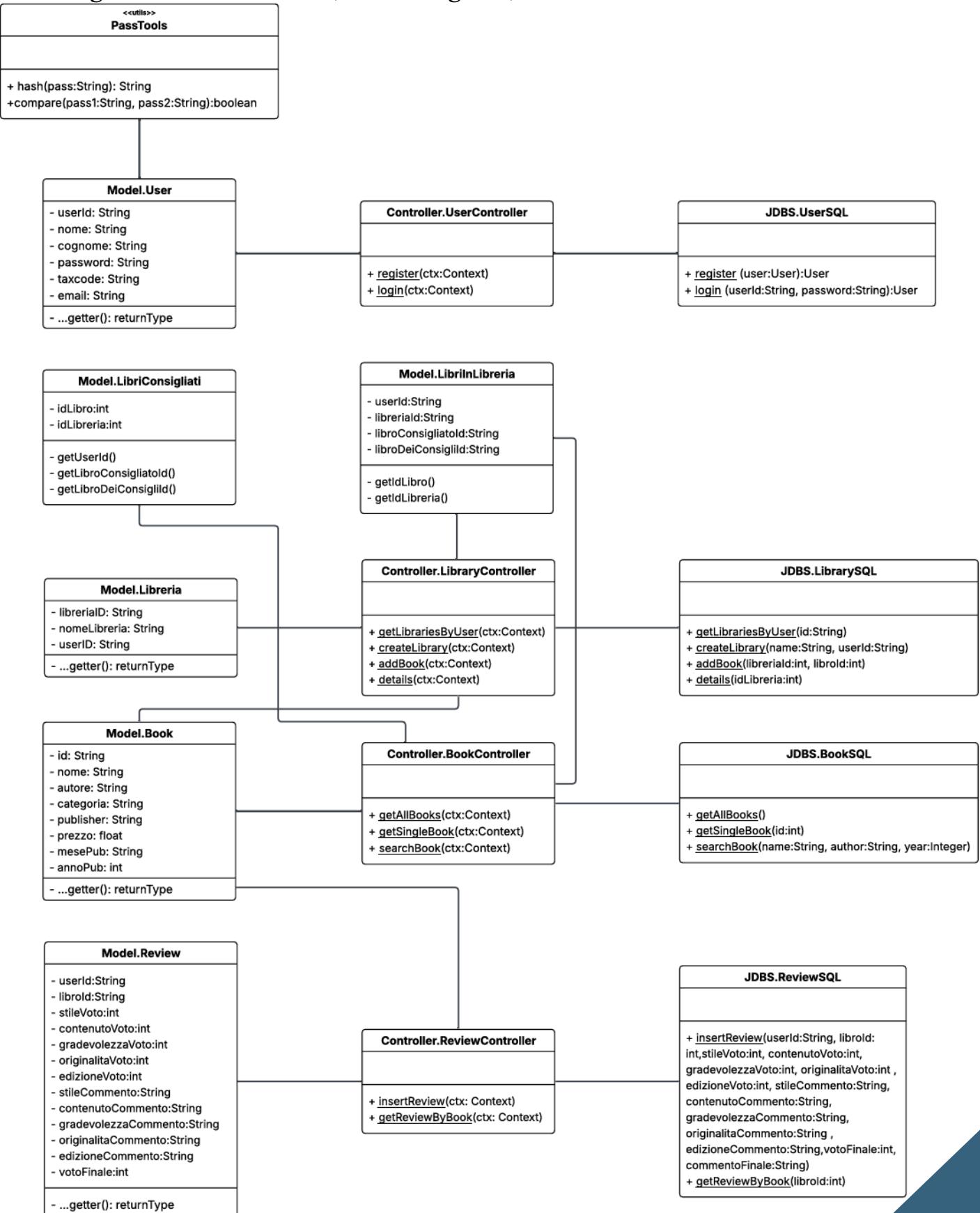
CREATE TABLE LibroInLibreria

```

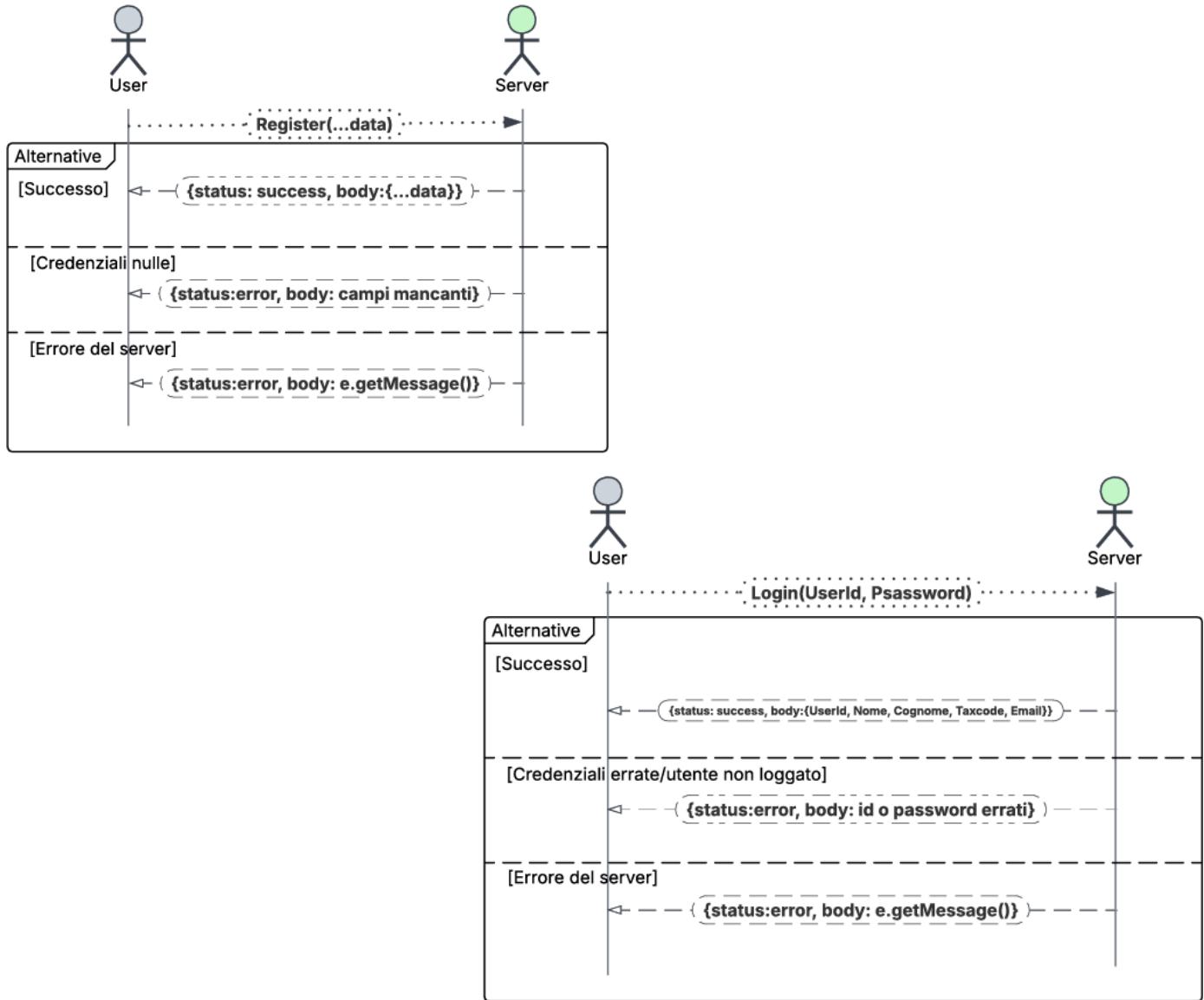
4

## 6.2 Diagrammi UML

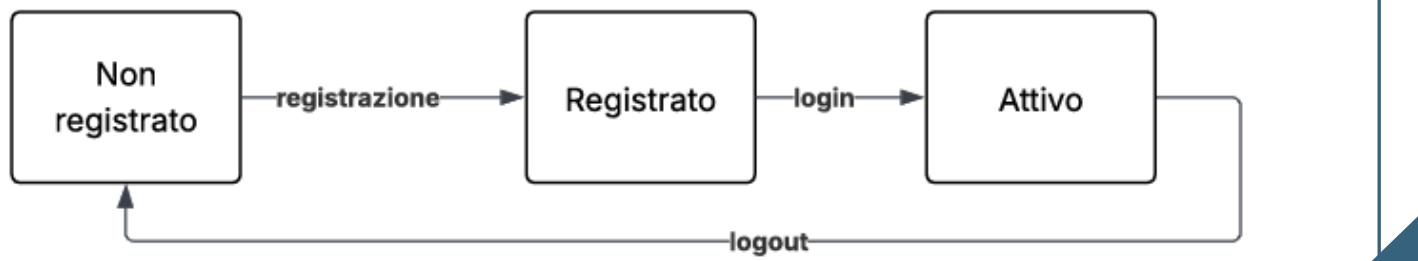
### Diagramma delle classi (Class Diagram)



## Diagramma di sequenza



## Diagramma di stato (semplificato)



## 6.3 Pattern progettuali utilizzati

Per garantire una struttura scalabile e manutenibile, il sistema impiega i seguenti pattern architetturali:

### ❖ MVC: (5)

- **MODEL:** (package: /model & /jdbcs) crea oggetti che sono repliche delle tavole del database, l'accesso e la modifica ai dati viene eseguita dalle classi nel package JDBS.
- **VIEW:** (frontend) le vue di Vue.js che mostrano i dati all'utente.
- **CONTROLLER:** (package: /controller) esegue operazioni che si interfacciano con il server, spesso modellando richieste/risposte usando i model. (6)

### ❖ VUE:

Il frontend è basato sulle convenzioni di Vue.js:

- **VIEWS:** contiene le pagine mostrate all'utente. (7)
- **COMPONENTS:** contiene componenti grafici riutilizzabili.
- **ROUTER:** contiene il router dell'applicazione.
- **ASSETS:** contiene immagini e stili.
- **SCRIPTS:** contiene scripts typescript riutilizzabili.
  - ◊ suddivisione non standardizzata:
    - /crud: script che si interfacciano con le API.
    - state.js: contiene un oggetto globale che rappresenta lo stato dell'applicazione.

---

Esempio di come i dati e le responsabilità vengono suddivise da diverse classi:

```
1 // JDBC di una libreria                                     5
2
3     public static List<Libreria> getLibrariesByUser(String id) {
4         try {
5             Connection conn = DriverManager.getConnection(DbInfo.url, DbInfo.user, DbInfo.pass);
6             PreparedStatement statement = conn.prepareStatement("SELECT * FROM Libreria WHERE UserId = ?");
7         }
8
9             statement.setString(1, id);
10            ResultSet rs = statement.executeQuery();
11            List<Libreria> librerias = new LinkedList<>();
12            while (rs.next()) {
13                librerias.add(new Libreria(
14                    rs.getString("LibreriaId"),
15                    rs.getString("NomeLibreria"),
16                    rs.getString("UserId")
17                ));
18            }
19
20            return librerias;
21        } catch (SQLException e) {
22            throw new RuntimeException(e);
23        }
24    }
```

```
1 // metodo del controller delle librerie                      6
2     public static void getLibrariesByUser(Context ctx) {
3         String id = ctx.pathParam("id");
4         List<Libreria> libraries = LibrarySQL.getLibrariesByUser(id);
5         try {
6             ctx.status(200).json(Map.of(
7                 "status", "success",
8                 "body", libraries
9             ));
10        } catch (Exception e) {
11            ctx.status(500).json(Map.of(
12                "status", "error",
13                "body", e.getMessage()
14            ));
15            throw new RuntimeException(e);
16        }
17    }
18 // Model delle librerie
19 public class Libreria {
20     private String LibreriaId;
21     private String nomeLibreria;
22     private String userId;
23
24     /**
25      * Model della libreria
26      * @param libreriaID L'id della libreria
27      * @param userID L'id dell'utente che ha creato la libreria
28      * @author Leonardo Basso
29      */
30     public Libreria(String libreriaID, String nomeLibreria, String userID) {
31         this.LibreriaId = libreriaID;
32         this.nomeLibreria = nomeLibreria;
33         this.userId = userID;
34     }
35 // getters...
```

```
1 <!--View su VueJS -->                                         7
2     <div class="user_view_elements" v-for="library in libraries">
3         <RouterLink :to="/library/books/${library.id}">
4             <p class="user_view_el collapsed-text">{{ library.title }}</p>
5         </RouterLink>
6     </div>
```

# 7. Strutture dati e scelte algoritmiche

## 7.1 Strutture dati principali

Il backend Java utilizza una rappresentazione orientata agli oggetti, che mappa le entità del dominio su tabelle relazionali PostgreSQL .

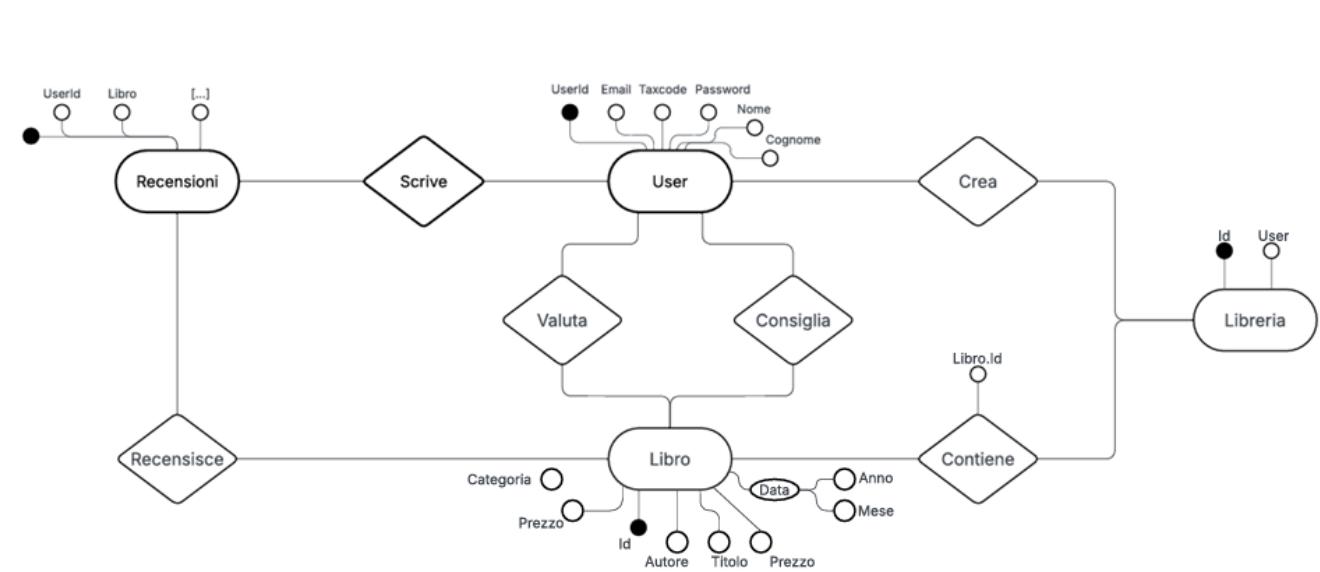
Entità Java	Tabella DB associata	Descrizione
Book	BookSQL	Titolo, autori, anno, editore, categoria
User	UserSQL	Dati anagrafici, credenziali
Libreria	LibrarySQL	Associazione libro-libreria
Review	ReviewSQL	Valutazioni secondo 5 criteri + nota
LibriConsigliati	/* */	Libri suggeriti per ogni titolo
LibriInLibreria	/* */	Libri inseriti nelle librerie

Tutte le entità sono modellate in classi Java con attributi privati, costruttori e metodi getter.

---

# 8. Progettazione e documentazione del database

## 8.1 Modello concettuale ER



## 8.2 Schema relazionale

**Tabella**

	<b>Campi principali</b>
Book	id, nome, autore, categoria, publisher, prezzo, mesePub, annoPub
User	userId, nome, cognome, password, taxcode, email
Libreria	libreriaID, nomeLibreria, userID
Review	userId, libroId, stileVoto, contenutoVoto, gradevolezzaVoto, originalitaVoto, edizioneVoto, stileCommento, contenutoCommento, gradevolezzaCommento, originalitaCommento, edizioneCommento, votoFinale
LibriConsigliati	idLibro, idLibreria
LibriInLibreria	userId, libreriaId, libroConsigliatoId, libroDeiConsiglioId

## 8.3 Relazioni tra tavelle

- ❖ Ogni utente può avere più **librerie**, ognuna contenente più **libri** (N:M).
- ❖ Ogni **valutazione** è associata a un solo libro e a un solo utente (1:1).
- ❖ Ogni **consiglio** associa un libro ad un utente ed ha un massimo di tre libri suggeriti per utente.
- ❖ Tutte le relazioni sono implementate tramite **chiavi esterne** con vincoli di integrità referenziale.

# 9. Sitografia e bibliografia

Risorsa	Link	Descrizione
PostgreSQL Official Site	<a href="https://www.postgresql.org/">https://www.postgresql.org/</a>	Documentazione e download del DBMS
Apache Maven	<a href="https://maven.apache.org/">https://maven.apache.org/</a>	Tool di automazione per progetti Java
Javalin Framework	<a href="https://javalin.io">https://javalin.io</a>	Webserver Java utilizzato nel backend
Vue.js Official Guide	<a href="https://vuejs.org/">https://vuejs.org/</a>	Framework JavaScript per frontend reattivo
Tauri Framework	<a href="https://tauri.app/">https://tauri.app/</a>	Pacchettizzazione e runtime desktop multipiattaforma
Rust Programming Language	<a href="https://www.rust-lang.org/">https://www.rust-lang.org/</a>	Linguaggio di sistema per il backend Tauri
Books Dataset (Kaggle)	<a href="https://www.kaggle.com/datasets/elvinrustam/books-dataset">https://www.kaggle.com/datasets/elvinrustam/books-dataset</a>	Dataset utilizzato nel test iniziale
UML Standard (OMG)	<a href="https://www.omg.org/spec/UML/">https://www.omg.org/spec/UML/</a>	Specifiche formali dei diagrammi UML

## 9.1 Documentazione Interna

Il codice è documentato tramite JavaDoc e JSDoc, esse saranno renderizzate all'interno del progetto nella cartella doc con /javadoc & /jsdoc.

Nel frontend sono presenti funzioni non documentate, tipicamente prefissate con handle+name (es. handleNewReview), che agiscono da wrapper per funzioni documentate. Queste vengono utilizzate all'interno dei componenti per delegare l'esecuzione delle funzioni principali e gestire centralmente la visualizzazione di errori, riducendo così la ridondanza nel codice.