# BookRecommender

*Manuale Tecnico*
*Versione documento: 1.0*

---

*Autrici:*
*Giulia Kalemi – Matricola 756143*
*Chiara Leone – Matricola 759095*

*Progetto Laboratorio B-Anno Accademico 2025/2026 ,Sede di Como*

## *Indice*

**Struttura dell'applicazione**
**Suddivisione moduli**
**Scelte progettuali**
**Documentazione script SQL**
**Query SQL a supporto dei servizi di interfaccia**
**Documentazione ER**
**Scelte architetturali**
**Strutture dati utilizzate**
**Formato file e gestione risorse**
**Bibliografia e Sitografie**

# Struttura dell'applicazione

L'applicazione BookRecommender è basata su un'architettura client-server. Durante lo sviluppo, sono stati adottati diversi design pattern e una struttura progettuale ben definita.
Il progetto utilizza Apache Maven per la gestione delle dipendenze; pertanto, è organizzato secondo la struttura standard prevista da Maven, inclusi tre file pom.xml: uno per il client, uno per il server e un pom padre per il progetto generale

```
BookRecommender/
├── .vscode/
├── client/
├── src/
│   └── main/
│       ├── java/bookrecommender/
│       ├── resources/
│       └── target/
├── pom.xml
├── doc/
├── launcher/
├── META-INF/
├── server/
├── .gitignore
├── autori.txt
├── pom.xml
├── README.txt
```

## Organizzazione dei package

Tutte le classi che compongono il progetto BookRecommender sono contenute all'interno del package principale:
**bookrecommender**.
Per una migliore organizzazione del codice e una separazione logica delle responsabilità, le classi dedicate all'accesso ai dati sono state collocate in un package secondario specifico:
**bookrecommender.dao**.

## Suddivisione moduli

Il **lato server** è responsabile della gestione della logica applicativa, dell'elaborazione delle richieste provenienti dal client e dell'accesso al database.

- L'architettura prevede l'ascolto delle richieste tramite socket, utilizzando il metodo accept(), con cui il server resta in attesa di una connessione da parte del client.

La comunicazione avviene attraverso una porta dedicata, configurata all'avvio del server.

- Il database utilizzato è PostgreSQL, scelto per ll suo standard SQL avanzato, flessibilità.
- Il server si occupa di eseguire le operazioni CRUD sulle entità principali dell'applicazione.

Le principali classi che compongono il modulo server sono:

- **ServerMain.java**: avvia il server, apre il socket sulla porta configurata e rimane in ascolto delle connessioni client tramite il metodo accept().
- ClientHandler.java:crea e avvia un nuovo thread dedicato per ogni client che si connette al server, permettendo così la gestione simultanea di più connessioni
- DBManager.java: si occupa della connessione al database

Classi DAO:

- Le seguenti classi DAO si occupano dell'accesso ai dati e dell'esecuzione delle query specifiche per ogni funzionalità:
- *Registra.java*: gestione della registrazione di nuovi utenti e librerie.
- *Ricerca.java*: esecuzione delle ricerche in base a titolo, autore o anno.
- *Valutazione.java*: inserimento e gestione delle valutazioni utente.
- *Visualizza.java*: recupero delle informazioni da visualizzare, come librerie, suggerimenti o riepiloghi.

Il **lato client** è responsabile dell’interazione con l’utente.
Le sue componenti principali sono:

- GUI (View): generata dinamicamente tramite i controller e realizzata con JavaFX, comprende le schermate che permettono all’utente di interfacciarsi con le principali funzionalità dell'applicazione, come:
  - eseguire ricerche sui libri,
  - accedere all'area riservata,
  - registrarsi,
  - inserire nuove valutazioni,
  - suggerire nuovi libri,
  - creare e gestire librerie personali.

Di seguito si riporta un elenco completo delle classi controller che gestiscono queste funzionalità:
- ARController.java
- **BookRecommender.java**
- ClientConnection.java
- HomeController.java
- LibController.java
- LibroController.java
- LoginController.java
- MainController.java
- MainStart.java
- NuovaLib2Controller.java
- NuovaLibController.java
- RegController.java
- SuggController.java
- TrovatoController.java
- ValutaController.java

# Scelte progettuali

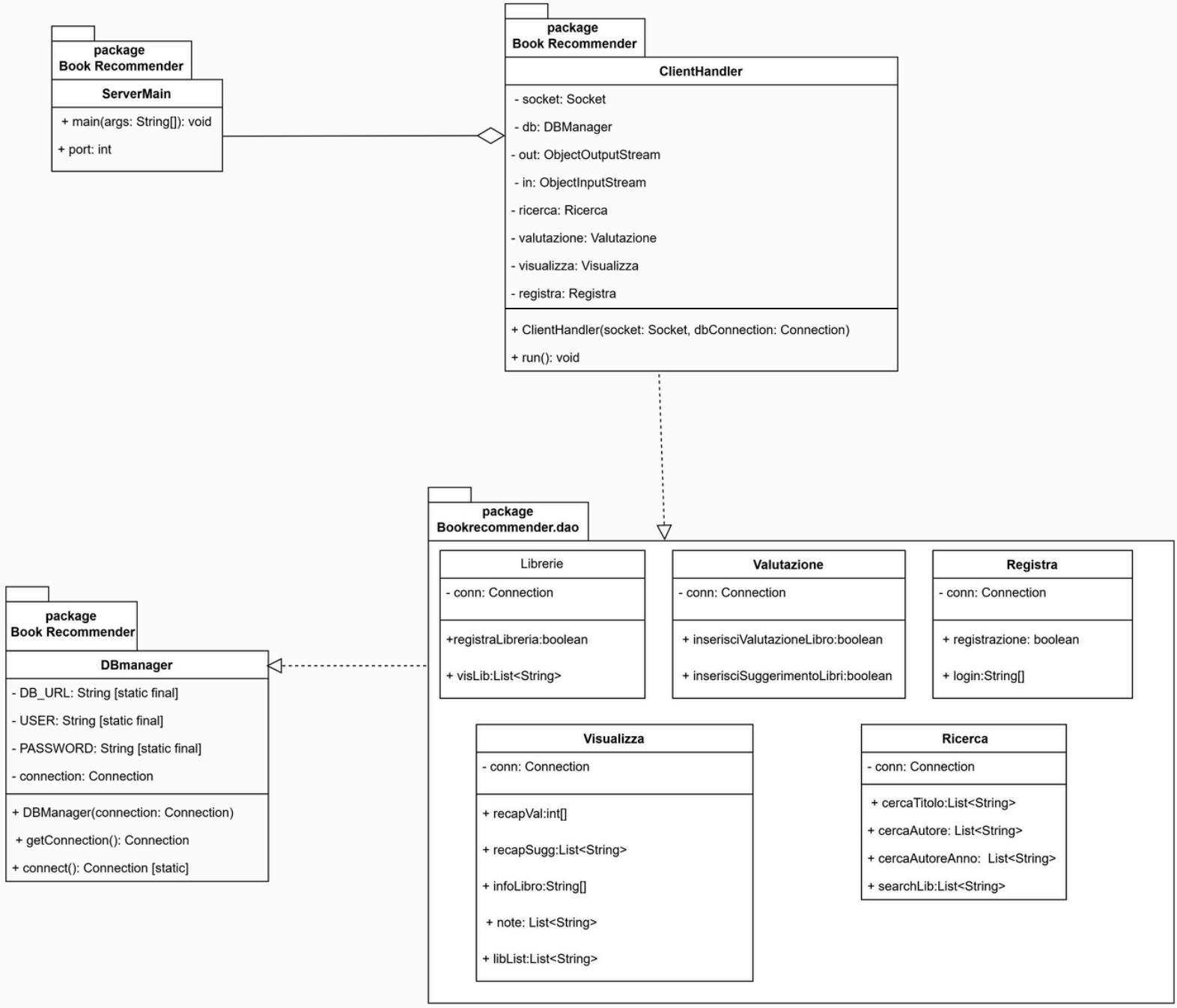
## *Class diagram: componenti server*

### Componenti attivi

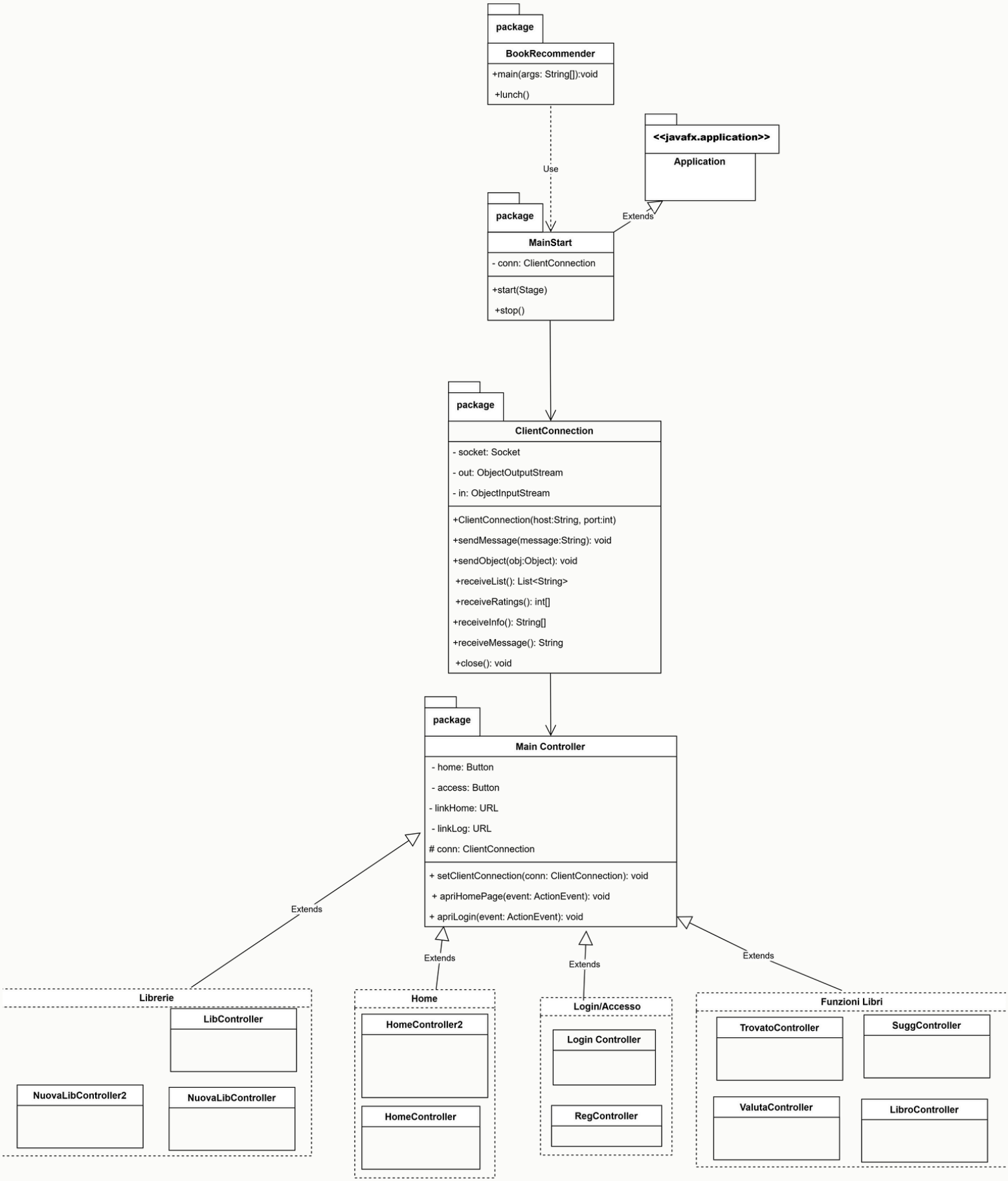
- ServerMain
- ClientHandler

### Componenti passivi

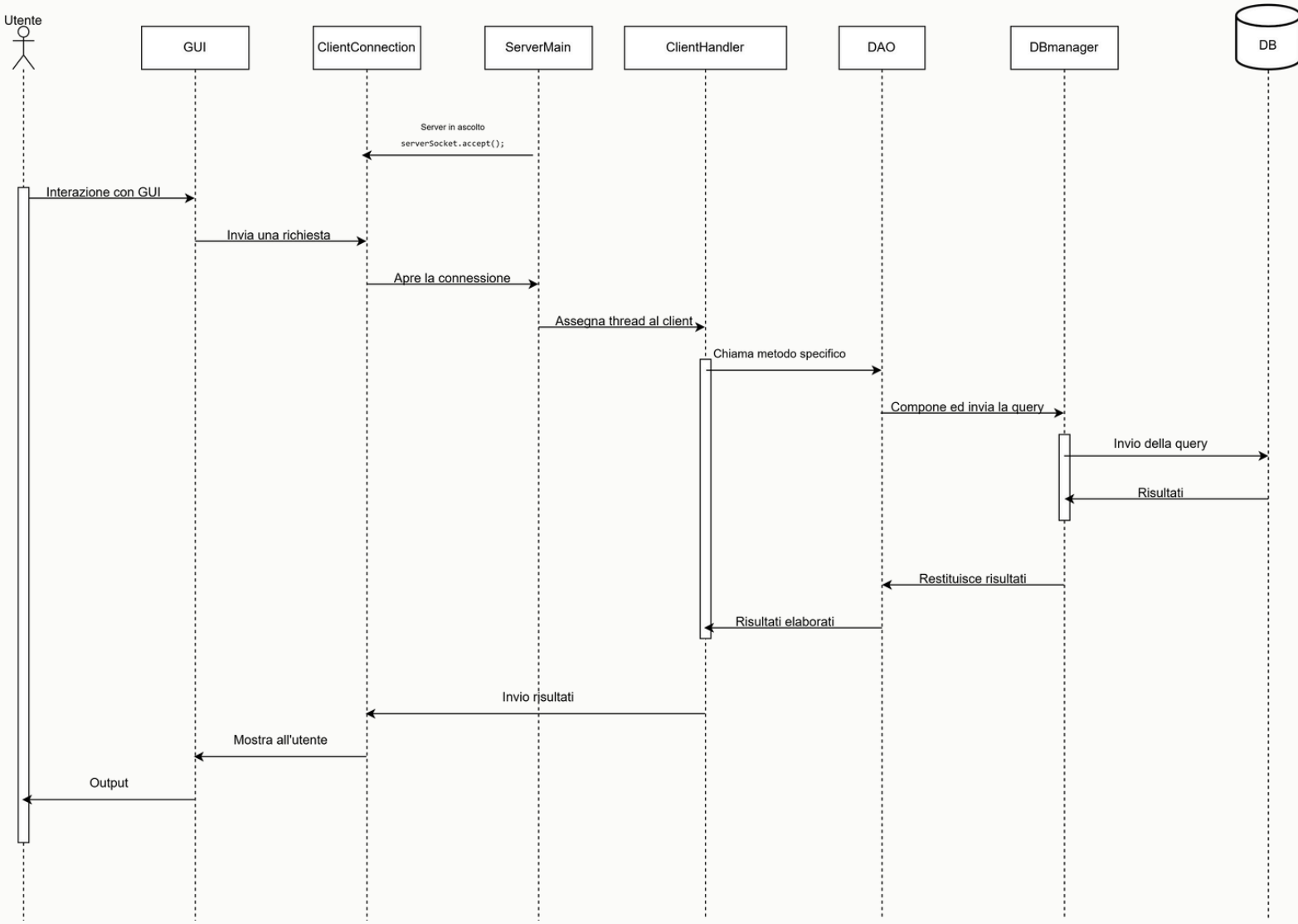
- DAO (LibrerieDAO, RegistraDAO, RicercaDAO, ValutazioneDAO, VisualizzaDAO
- DBManager



# Class diagram: componenti Client



# Sequence diagram



## *Documentazione Script SQL*

**1. Tabella UtentiRegistrati**

```
CREATE TABLE UtentiRegistrati (
    UserID SERIAL PRIMARY KEY,
    Name_Surname VARCHAR(100) NOT NULL,
    CF VARCHAR(16) UNIQUE NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Password VARCHAR(100) NOT NULL
);
```

**2. Tabella Librerie**

```
CREATE TABLE Librerie (
    ID  PRIMARY KEY,
    Lib_Name VARCHAR(100) NOT NULL,
    UserID INT NOT NULL,
    FOREIGN KEY (UserID) REFERENCES UtentiRegistrati(UserID)
);
```

**3. Tabella Libri**

```
CREATE TABLE Libri (
    ID  PRIMARY KEY,
    Title VARCHAR(200) NOT NULL,
    Authors VARCHAR(200),
    Category VARCHAR(100),
    Publisher VARCHAR(100),
    Pub_Year INT
);
```

**4. Tabella Libri.Librerie**

```
CREATE TABLE Libri_Librerie (
    LibID INT NOT NULL,
    BookID INT NOT NULL,
    PRIMARY KEY (LibID, BookID),
    FOREIGN KEY (LibID) REFERENCES Librerie(ID),
    FOREIGN KEY (BookID) REFERENCES Libri(ID)
);
```

**5. Tabella ConsigliLibri**

```
CREATE TABLE ConsigliLibri (
    ID SERIAL PRIMARY KEY,
    UserID INT NOT NULL,
    BookID INT NOT NULL,
    SuggID INT,
    FOREIGN KEY (UserID) REFERENCES UtentiRegistrati(UserID)

);
```

**6. Tabella ValutazioniLibri**

```
CREATE TABLE ValutazioniLibri (
 ID PRIMARY KEY,
 UserID INT NOT NULL,
 BookID INT NOT NULL,
 Style INT,
 Content INT,
 Pleasantness INT,
 Originality INT,
 Edition INT,
 FinalVote INT,
 Note_Style VARCHAR(256),
 Note_Content VARCHAR(256),
 Note_Pleasantness VARCHAR(256),
 Note_Originality VARCHAR(256),
 Note_Edition VARCHAR(256),
 FOREIGN KEY (UserID) REFERENCES UtentiRegistrati(UserID),
```

## *Query SQL a supporto dei servizi di interfaccia*

Le query utilizzate per supportare la gestione delle librerie sono contenute nella classe Librerie.java
Di seguito si riportano le principali query utilizzate per la **classe Librerie.java**

**Metodo registraLibrerie()**

Le seguenti query sono utilizzate per registrare una nuova libreria e per associare un libro ad essa:

// Recupera l'ID del libro a partire dal titolo

- String getBookIdSql = "SELECT \"id\" FROM \"Libri\" WHERE \"Title\" = ?";

// Verifica se esiste già una libreria con lo stesso nome per lo stesso utente

- String checkSql = "SELECT \"id\" FROM \"Librerie\" WHERE \"UserID\" = ? AND \"Lib_Name\" = ?";

// Inserisce una nuova libreria

- String insertSql = "INSERT INTO \"Librerie\" (\"Lib_Name\", \"UserID\") VALUES (?, ?)";

// Verifica se il libro è già presente nella libreria

- String checkBookSql = "SELECT COUNT(*) FROM \"Libri.Librerie\" WHERE \"LibID\" = ? AND \"BookID\" = ?";

// Inserisce un libro nella libreria

- String insertBookSql = "INSERT INTO \"Libri.Librerie\" (\"LibID\", \"BookID\") VALUES (?, ?)";

**Metodo deleteLib()**

Questo metodo elimina una libreria di un utente e tutti i libri associati ad essa.
Le query utilizzate sono le seguenti:

// Recupera l'ID della libreria da eliminare

- String getLibIdSql = "SELECT \"id\" FROM \"Librerie\" WHERE \"UserID\" = ? AND \"Lib_Name\" = ?";

// Elimina i libri associati alla libreria

- String deleteBooksSql = "DELETE FROM \"Libri.Librerie\" WHERE \"LibID\" = ?";

Elimina la libreria

- String deleteLibSql = "DELETE FROM \"Librerie\" WHERE \"id\" = ?";

Le query utilizzate per supportare la gestione della registrazione sono contenute nella classe Registra.java
Di seguito le query principali associate ai metodi di **Registra.Java**

**Metodo registrazione()**

Questa query permette di inserire un nuovo utente nella tabella UtentiRegistrati, salvando i dati personali e le credenziali necessarie per l'accesso.

- String query = """
  INSERT INTO "UtentiRegistrati" ("Name_Surname", "CF", "Email", "UserID", "Password")
  VALUES (?, ?, ?, ?, ?)
""";

**Metodo checkReg()**

Questa query viene utilizzata per verificare se esiste già un utente registrato con lo stesso UserID, codice fiscale (CF) o email, evitando così duplicazioni durante la fase di registrazione.

- String query = """
  SELECT * FROM "UtentiRegistrati"
  WHERE "UserID" = ? OR "CF" = ? OR "Email" = ?
""";

**Metodo login()**

Questa query consente di autenticare un utente verificando che la combinazione di UserID e Password corrisponda a un record presente nella tabella UtentiRegistrati.

- String query = """
  SELECT * FROM "UtentiRegistrati"
  WHERE "UserID" = ? AND "Password" = ?
""";

Le query utilizzate per le funzionalità di ricerca dei libri sono contenute nella classe DAO Ricerca.
Di seguito le query principali associate ai metodi di **Ricerca.java:**

**Metodo cercaTitolo()**

Questa query restituisce i titoli dei libri che corrispondono al parametro di ricerca (case-insensitive).

- String query = "SELECT \"Title\" FROM \"Libri\" WHERE LOWER(\"Title\") LIKE ?";

**Metodo cercaAutore()**

Questa query restituisce i titoli dei libri il cui autore corrisponde al parametro di ricerca (case-insensitive)

- String query = "SELECT \"Title\" FROM \"Libri\" WHERE LOWER(\"Authors\") LIKE ?";

**Metodo cercaAutoreAnno()**

Questa query filtra i libri per autore (ricerca case-insensitive) e anno di pubblicazione esatto**.**

- String query = "SELECT \"Title\" FROM \"Libri\" WHERE LOWER(\"Authors\") LIKE ? AND \"Pub_Year\" = ?";

**Metodo searchLib()**

Questa query recupera i titoli dei libri appartenenti a una libreria specifica di un utente, filtrando per titolo (case-insensitive), UserID e nome della libreria

- String query = """
  SELECT l."Title"
  FROM "Libri" l
  JOIN "Libri.Librerie" ll ON l."id" = ll."BookID"
  JOIN "Librerie" lib ON ll."LibID" = lib."id"
  WHERE LOWER(l."Title") LIKE ?
    AND lib."UserID" = ?
    AND lib."Lib_Name" = ?
""";

Le query utilizzate per gestire le valutazioni e i suggerimenti di libri sono contenute nella classe DAO Valutazione.Java
Di seguito le query principali associate ai metodi di Valutazione.java

**Metodo inserisciValutazioneLibro()**
Questa query si occupa di recuperare l'ID del libro a partire dal titolo

- String getBookIdSql = "SELECT \"id\" FROM \"Libri\" WHERE \"Title\" = ?";

Questa query si occupa di verificare se l'utente ha già inserito una valutazione per il libro

- String checkSql = "SELECT COUNT(*) FROM \"ValutazioniLibri\" WHERE \"UserID\" = ? AND \"BookID\" = ?";

Questa query si occupa di inserire una nuova valutazione per il libro

- String insertSql = """
  INSERT INTO "ValutazioniLibri" (
  "UserID", "BookID", "Style", "Content", "Pleasantness",
  "Originality", "Edition", "FinalVote",
  "Note_Style", "Note_Content", "Note_Pleasantness",
  "Note_Originality", "Note_Edition"
  ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
  """;

**Metodo inserisciSuggerimentoLibri**
Questa query si occupa di recuperare l'ID del libro suggerito a partire dal titolo

- String getBookIdSql = "SELECT \"id\" FROM \"Libri\" WHERE \"Title\" = ?";

Questa query si occupa di recuperare l'ID del libro suggeritore a partire dal titolo

- String getSuggIdSql = "SELECT \"id\" FROM \"Libri\" WHERE \"Title\" = ?";

Questa query si occupa di verificare se l'utente ha già suggerito quel libro

- String checkSql = "SELECT COUNT(*) FROM \"ConsigliLibri\" WHERE \"UserID\" = ? AND \"BookID\" = ?";

Questa query si occupa di verificare se esiste già un suggerimento duplicato per lo stesso libro e utente

- String checkDuplicateSql = "SELECT COUNT(*) FROM \"ConsigliLibri\" WHERE \"UserID\" = ? AND \"BookID\" = ? AND \"SuggID\" = ?";

Questa query si occupa di inserire un nuovo suggerimento di libro

- String insertSql = "INSERT INTO \"ConsigliLibri\" (\"UserID\", \"BookID\", \"SuggID\") VALUES (?, ?, ?)";

Le query utilizzate per gestire le valutazioni e i suggerimenti di libri sono contenute nella classe DAO

**Visualizza.Java**

Di seguito le query principali associate ai metodi di Visualizza.Java:

**Metodo recapVal()**

Recupera le valutazioni aggregate di un libro, ottenendo i voti per stile, contenuto, gradevolezza, originalità, edizione e voto finale in base al titolo (case-insensitive).

- String query = """
  SELECT VL."Style", VL."Content", VL."Pleasantness", VL."Originality", VL."Edition", VL."FinalVote"
  FROM "ValutazioniLibri" VL
  JOIN "Libri" L ON VL."BookID" = L."id"
  WHERE LOWER(L."Title") = LOWER(?)
""";

**Metodo recapSugg()**

Restituisce la lista di titoli suggeriti per un dato libro, con il conteggio degli utenti che hanno fatto ciascun suggerimento.

- String query = """
  SELECT L2."Title", COUNT(DISTINCT CL."UserID") as nSugg
  FROM "ConsigliLibri" CL
  JOIN "Libri" L1 ON CL."BookID" = L1."id"
  JOIN "Libri" L2 ON CL."SuggID" = L2."id"
  WHERE LOWER(L1."Title") = LOWER(?)
  GROUP BY L2."Title"
""";

**Metodo infoLib()**

Recupera le informazioni di base di un libro (autore, categoria, casa editrice, anno di pubblicazione) a partire da titolo.

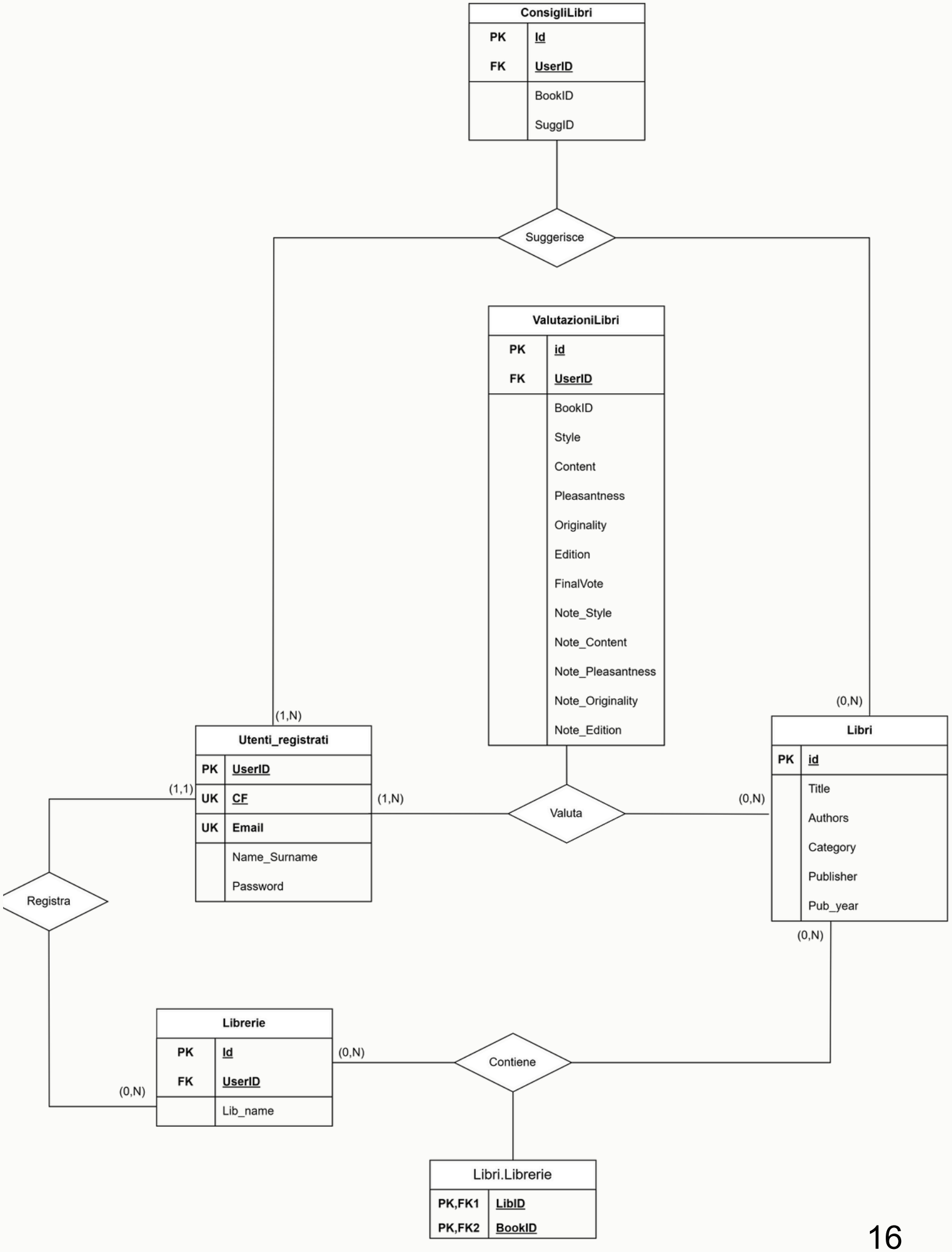
- String query = """
  SELECT "Authors", "Category", "Publisher", "Pub_Year"
  FROM "Libri"
  WHERE LOWER("Title") = LOWER(?)
""";

**Metodo libList()**

Restituisce la lista delle librerie associate a un utente.

- String query = """
  SELECT "Lib_Name"
  FROM "Librerie"
  WHERE "UserID" = ?
""";

# Diagramma ER



# Documentazione diagramma ER

## Dizionario dei dati –entità

| Entità | Attributi | PK | Descrizione |
|---|---|---|---|
| **ConsigliLibri** | id,UserID,BookID,SuggID | id | Gestisce i suggerimenti di lettura |
| **Librerie** | id,Lib_Name,UserID | id | Contiene informazioni su librerie registrate nel sistema. |
| **Libri** | id,Title, Authors, Category, Publisher, Pub_year | id | Rappresenta un libro presente nel sistema. |
| **ValutazioniLibri** | ID, UserID, Style, Content, Pleasantness, Originality, Edition, Final Vote, Note_Style, Note_Content, Note_Pleasantness, Note_Originality | ID | Contiene le valutazioni assegnate ai libri da parte degli utenti. |
| **Libri.Librerie** | LibID,BookID | LibID,BookID | indica che un certo libro è presente in una determinata libreria. |
| **Utenti_Registrati** | Name_Surname, CF, Email, UserID, Password | UserID | Rappresenta un utente registrato nel sistema. |

## Dizionario dei dati –relazioni

| Relazione | Descrizione | Entità coinvolte | Collgamenti |
|---|---|---|---|
| **Suggerisce** | Associa un suggerimento riguardante un libro a un utente | Utenti_Registrati(1,N) Libri(0,N) | - |
| **Registra** | Associa la registrazione/creazione di una nuova libreria a un utente | Utenti_Registrati(1,1), Librerie(0,N) | - |
| **Contiene** | Indica quali libri contiene una libreria | Librerie(0,N), Libri(0,N) | Libri.Librerie |
| **Valuta** | Associa a un libro la valutazione fornita da un utente | Utenti_Registrati(1,N), Libri(0,N) | Valutazioni.libri |

## Vincoli d’integrità

| Vincolo | Descrizione |
|---|---|
| 1 | I campi note nella tabella ValutazioniLibri possono contenere un massimo di 256 caratteri |
| 2 | Se l’utente non inserisce una valutazione in una categoria nella tabella ValutazioniLibri, verrà assegnato automaticamente un valore di 5 stelle. |
| 3 | Il campo user_id nella tabella UtentiRegistrati può contenere al massimo 250 caratteri. |

## Analisi dei requisiti

Sono stati considerati i seguenti requisiti funzionali del sistema:
**Registrazione utenti**: gli utenti possono registrarsi fornendo CF, email, nome, cognome, user_id, password.
**Gestione librerie personali:** ogni utente può creare una o più librerie personalizzate.
**Gestione dei libri:** il sistema memorizza informazioni bibliografiche su ogni libro (titolo, autori, categoria, editore, anno).
**Valutazione dei libri**: gli utenti possono valutare i libri secondo criteri multipli (stile, contenuto, piacevolezza, originalità, edizione), con note testuali. La valutazione complessiva genera un voto finale
**Suggerimento di libri:** gli utenti possono suggerire un libro partendo da un altro libro.
**Associazione libri–librerie:** un libro può appartenere a più librerie, e ogni libreria può contenere più libri.

## Scelte progettuali effettuate durante lo sviluppo

Organizzazione in package separati

- bookrecommender
- bookrecommender.dao

Questa organizzazione consente di separare la logica applicativa dalla logica di accesso ai dati.
L'obiettivo di questa scelta è migliorare la leggibilità del codice.

Principio di Responsabilità Singola

Ogni classe è stata progettata per avere una sola responsabilità ben definita.
In particolare:

- Ricerca.java: si occupa delle funzionalità di ricerca dei libri nel database.
- Valutazione.java: gestisce l'inserimento di valutazioni e suggerimenti.
- Visualizza.java: recupera e restituisce informazioni per la visualizzazione da parte del client.
- Registra.java: gestisce la registrazione di nuovi utenti e l'autenticazione.

L'obiettivo è garantire una chiara separazione dei compiti.

Interfaccia Grafica con JavaFX

L'interfaccia utente lato client è realizzata utilizzando JavaFX.

Utilizzo di Apache Maven

L'intero progetto è gestito tramite Apache Maven.
Sono presenti tre file pom.xml: uno per il client, uno per il server e uno principale (padre).
Maven si occupa della gestione delle dipendenze, della compilazione e dell'esecuzione del progetto in modo strutturato e modulare.
L'obiettivo è focalizzarsi sull'organizzazione del progetto, favorendo una struttura chiara e la gestione efficiente dei diversi moduli

# Scelte architetturali

Nel progettare il sistema BookRecommender, sono state effettuate alcune scelte architetturali:

- MVC (Model-View-Controller)
- DAO (Data Access Object)-Design Pattern
- Singelton-Design Pattern

Il **MVC** ci ha permesso di avere una chiara separazione tra logica, interfaccia e gestione dei dati

- Il modello (Model), pur non essendo rappresentato come entità separata, si trova nella logica di accesso ai dati presente nella classe DAO (Data Access Object) e nel database sottostante.
- View (V): è rappresentata dalla GUI generata dinamicamente dai controller
- Controller (C): le classi che compongono la logica del progetto BookRecommender e MainStart...ecc, agiscono come controller.

Ognuna è responsabile di una specifica funzionalità

**DAO**

Il pattern DAO è stato utilizzato per isolare la logica di accesso ai dati dal resto dell'applicazione. Tutte le operazioni di lettura e scrittura sul database vengono gestite attraverso oggetti DAO.

**Singleton**

è stata implementata una singola istanza di Connection conn che rappresenta la connessione tra client e server.

- Questa connessione viene creata una sola volta all'avvio dell'applicazione client e poi riutilizzata in tutti i metodi che devono comunicare con il server.

# Strutture dati utilizzate

## Client Connection

Nella classe ClientConnection vengono utilizzate diverse strutture dati per gestire la comunicazione client-server:

- public **List<String>** receiveList() throws IOException

Per ricevere e restituire una lista di stringhe dal server

- public **int []** receiveRatings() throws IOException

Per ricevere un array di valutazioni dal server

- public **String []** receiveInfo() throws IOException

Per ricevere un array con informazioni testuali

## ClientHandler

List<String>

- List<String> titoli
- List<String> note
- List<String> suggerimenti
- List<String> libriLibreria

Usata in più metodi per restituire elenchi di risultati al client

**int[]**

- int[] val

**String[]**

- String[] paramAA = parts[1].split(";", 2);
- String[] info = visualizza.infoLibro(parts[1]);
- String[] noteParams = parts[1].split(",");
- String[] paramLib = parts[1].split(",");
- String[] paramVisLib = parts[1].split(",");
- String[] paramDelLib = parts[1].split(",");
- String[] paramLogin = parts[1].split(",");
- String[] paramReg = parts[1].split(",");
- String[] paramCheck = parts[1].split(",");
- String[] paramSugg = parts[1].split(",");
- String[] idAndTit = (String[]) in.readObject();

## ServerMain

**Map<String, String>**

- Map<String, String> credentials = DBLoginWindow.showLoginDialog()

# Formato file e gestione

Nel modulo client del progetto BookRecommender, i file sono organizzati all'interno della struttura src/main/resources, suddivisi in base al loro tipo e utilizzo:

**File CSS (.css)**

Utilizzati per definire lo stile grafico delle interfacce JavaFX.

- Percorso: src/main/resources/css/Stile.css

**File FXML (.fxml)**

Utilizzati per descrivere la struttura dell'interfaccia grafica in JavaFX.

- Percorso: src/main/resources/fxml/

**File Immagine (.png)**

Utilizzati per icone e elementi visivi dell'interfaccia utente.

- Percorso: src/main/resources/png/

Nel modulo server del progetto BookRecommender, i file sono organizzati all'interno della struttura src/main/resources:

**pom.xml**

- server/pom.xml

**ClientHandler.java**

server/src/main/java/bookrecommender/dao/ClientHandler.java

**DBLoginWindow.java**

- server/src/main/java/bookrecommender/dao/DBLoginWindow.java

**DBManager.java**

server/src/main/java/bookrecommender/dao/DBManager.java

**ServerMain.java**

server/src/main/java/bookrecommender/dao/ServerMain.java

# Bibliografia/Sitografia

Durante lo sviluppo dell'applicazione e la stesura del manuale utente, sono state consultate le seguenti fonti per ottenere riferimenti tecnici, linee guida e librerie utilizzate:


- Pighizzini, G., & Ferrari, M. (2015). Dai fondamenti agli oggetti. Corso di programmazione Java
- Materiale didattico del corso di Laboratorio B, a.a. 2024/2025.
- Materiale del corso di "Basi di Dati" a cura del Prof. Davide Spoladore – Dispense, slide e risorse didattiche fornite durante il corso universitario.
- Materiale del corso di "Progettazione del Software" a cura del Prof. Davide Albertini – Appunti, esempi pratici e concetti teorici utilizzati per la realizzazione del progetto.

- Java Documentation (Oracle)
- JavaFX 22 – OpenJFX Official Site
- Apache Maven – Project Management Tool
- Visual Studio Code - Java Extension Pack
- Stack Overflow – Risoluzione di dubbi e problemi tecnici
- GitHub – Progetti open-source consultati per struttura e ispirazione
- OpenJDK – Java Development Kit
- JavaFX Documentation – GitHub
- Draw.io – Strumento online per la creazione di diagrammi e schemi interattivi.: https://www.draw.io