 # Università degli studi Insubria

"BOOK RECOMMENDER" –

MANUALE TECNICO

Autori:

- Mouhammad Toure
- Daniel Viny Kamdem Tagne
- Agnes Balkaire Makouwe

Progetto Laboratorio B: BookRecommender**

Anno accademico: 2024/2025**

Temi

---

1. Introduzione

BookRecommender è un progetto sviluppato nell'ambito del progetto di Laboratorio B per il corso di laurea in Informatica dell'Università degli Studi dell'Insubria. Il progetto è stato realizzato utilizzando il linguaggio di programmazione Java e testato su sistemi operativi Windows 11, macOS Big Sur e Linux Ubuntu.

Il progetto si articola in tre componenti principali:

1. ****Sistema Client-Server**:** Architettura distribuita basata su Java RMI (Remote Methode Invocation) che permette a più utenti di utilizzare il sistema contemporaneamente.

2. ****Database PostgreSQL**:** Sistema di persistenza dei dati relazionale che sostituisce i file di testo del progetto originale, garantendo maggiore sicurezza e scalabilità.

3.Gestione delle sessioni**:** Sistema multi-utente con sessioni isolate per ogni client, garantendo l'isolamento dei dati tra utenti diversi.

Caratteristiche principali del sistema:

- **Architettura distribuita**: Client e server separati con comunicazione RMI

- **Database relazionale:** PostgreSQL per la persistenza dei dati

- **Sistema multi-utente:** Sessioni isolate per ogni client

- **Setup automatico:** DatabaseInitializer per la configurazione automatica del database

- **Gestione password dinamica:** Password del database richiesta all'avvio del programma per la sicurezza

- **Validazione dati:** Controlli di integrità sui dati inseriti

---

2. Struttura del progetto

Struttura delle cartelle:

BookRecommender/

```
├── src/
│   └── bookrecommender/
│       ├── BookRecommender.java          # Logica di business principale
│       ├── Server.java                   # Server RMI
│       ├── Client.java                   # Client RMI
│       ├── DatabaseInitializer.java      # Inizializzazione database
│       ├── InterfaceBook.java            # Interfaccia RMI
│       ├── InterfaceImpl.java            # Implementazione RMI
│       ├── UserID.java                   # Classe per gestione utenti
│       ├── Valutazione.java              # Classe per gestione valutazioni
│       ├── Trasferimento.java            # Permette di popolare il database di libri
│       ├── Libreria.java                 # Classe per la gestione delle librerie
│       └── Consiglio.java                # Classe per la gestione consigli
├── bin/                                  # File compilati
├── lib/                                  # Driver JDBC PostgreSQL
├── doc/                                  # Documentazione
├── javadoc/                              # File di javadoc
├── clean.bat                             # Per la pulizia del programma
├── compile.bat                           # Per la compilazione del programma
├── setup_database.bat                    # Script setup Windows
├── populate.bat                          # Per la popolazione del database
├── setup_database.ps1                    # Script setup PowerShell
├── start_server.bat                      # Per avviare il server
├── start_client.bat / start_gui.bat      # Per avviare il cliente oppure l'interfaccia utente
├── DATABASE_SETUP_README.md              # Documentazione setup
├── USER_MANUAL.md                        # Manuale utente
├── README.md                             # Documentazione principale
└── TECHNICAL_MANUAL.md                   # Manuale Tecnico (questo manuale)
```

File principali:

- **BookRecommender.java:**

Contiene la logica di business principale e la gestione del database

- **Server.java:**

Avvia il server RMI e gestisce la connessione al database

- **Client.java:**

Interfaccia utente e gestione delle sessioni client

- **DatabaseInitializer.java:**

Configurazione automatica del database

-Trasferimento.java**:**

Permette il popolamento del database

- **InterfaceBook.java:**

Interfaccia RMI per la comunicazione client-server

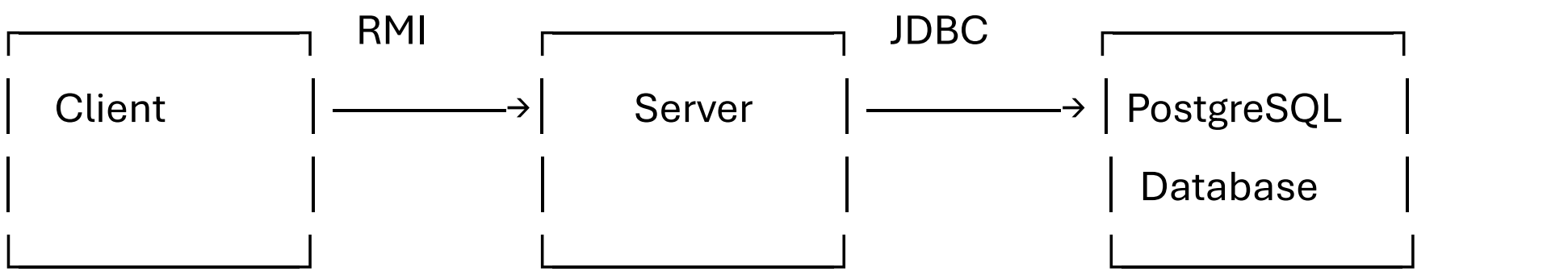
- **InterfaceImpl.java:**

Implementazione dei metodi della classe interface RMI

3. Architettura del sistema

3.1 Architettura Client-Server

Il sistema utilizza un'architettura client-server basata su Java RMI:



3.2 Componenti principali:

Client

- **Funzione:**

Interfaccia utente e gestione delle sessioni

- **Caratteristiche:**

- Session ID univoco per ogni client
- Menu interattivo per le operazioni
- Gestione delle connessioni RMI

Server

- **Funzione:**

gestione delle connessioni di RMI e del database

- **Caratteristiche:**

- Registry RMI per la registrazione dei servizi
- Gestione delle connessioni database
- Validazione e processamento dei dati

Database

- **Funzione:**

Persistenza dei dati

- **Caratteristiche:**

- PostgreSQL come database relazionale
- Tabelle normalizzate per i dati
- Controlli di integrità referenziale

3.3 Comunicazione RMI

Il sistema utilizza Java RMI per la comunicazione tra client e server:

```
```

```java
// Registrazione del servizio
Registry registry = LocateRegistry.createRegistry(PORTA);
InterfaceImpl interfaceImpl = new InterfaceImpl();
registry.rebind("BookRecommender", interfaceImpl);

// Connessione client
Registry registry = LocateRegistry.getRegistry("localhost", PORTA);
InterfaceBook interfaceBook = (InterfaceBook)
registry.lookup("BookRecommender");
```

4. Struttura del programma

4.1 BookRecommender.java

Metodi principali:

```java
// Gestione database
public static synchronized boolean initializeDatabasePassword()
public static synchronized boolean isDatabasePasswordSet()
private synchronized Connection getConnection()

// Operazioni sui libri
public synchronized String visualizzareLibri()
public synchronized String cercaLibroConTitolo(String title)
public synchronized String cercaLibroConAutore(String author)
public synchronized String cercaLibroConAutoreAnno(String author, int year)
```

// Gestione utenti

public synchronized String registrazione(String sessionId, String name, String cf, String email, String userid, String password)

public synchronized String login(String sessionId, String username, String password)

public synchronized String logout(String sessionId)

// Recupero password

public synchronized String recuperaPassword(String username)

public synchronized String generaPasswordTemporanea(String username)

private String generaPasswordCasuale()

public String cambiaPassword(String sessionId, String userid, String attualePassword, String nuovaPassword);

// Gestione librerie

public synchronized String creareLibreria(String sessionId, String nomeLibreria)

public synchronized String aggiungiLibroLibreria(String sessionId, String nomeLibreria, String titoloLibro)

public synchronized String rimuoviLibroLibreria(String sessionId, String nomeLibreria, String titoloLibro)

public synchronized String visualizzaLibreria(String sessionId, String nomeLibreria)

// Valutazioni e consigli

public synchronized String inserisciValutazioneLibro(String sessionId, String title, String style, String content, String pleasantness, String originality, String edition)

public synchronized String inserisciConsiglioLibro(String sessionId, String referenceBook, String recommendedBooks)

// Utility

private synchronized UserID getUserFromSession(String sessionId)

private synchronized String hashedPassword(String password)

Gestione delle sessioni:

```java
private static ConcurrentHashMap<String, UserID> userSessions = new ConcurrentHashMap<>();
```

4.2 Server.java

Funzionalità principali:

```java
public static void main(String[] args) {
    // Richiesta password database
    boolean passwordSet = BookRecommender.initializeDatabasePassword();
// Creazione registry RMI
    Registry registry = LocateRegistry.createRegistry(PORT);
    InterfaceImpl interfaceImpl = new InterfaceImpl();
    registry.rebind("BookRecommender", interfaceImpl);
}
```

4.3 Client.java

Caratteristiche principali:

```java
public class Client {
    private String sessionId;
    public Client() {
        this.sessionId = UUID.randomUUID().toString();
    }
```

public void start() {

// Connessione al server RMI

Registry registry = LocateRegistry.getRegistry("localhost", 1099);

InterfaceBook interfaceBook = (InterfaceBook) registry.lookup("BookRecommender");

// Menu interattivo

while (true) {

// Gestione delle opzioni del menu

}

}

}

` ` `

4.4 DatabaseInitializer.java

**Funzionalità principali:**

` ` `java

// Richiesta password

private static boolean requestDatabasePassword()

// Creazione tabelle

private void createUseridTable(Connection conn)

private void createLibriTable(Connection conn)

private void createValutazioniTable(Connection conn)

private void createConsigliTable(Connection conn)

private void createLibrerieTable(Connection conn)

---

5. Le scelte architetturali

5.1 Architettura distribuita

****Motivazione**:**

Il passaggio da un'architettura monolitica a un'architettura client-server permette:

- ****Scalabilità**:** Più utenti possono utilizzare il sistema contemporaneamente
- ****Isolamento**:** Ogni client ha una sessione indipendente
- ****Manutenibilità**:** Separazione delle responsabilità tra client e server
- ****Estensibilità**:** Facile aggiunta di nuove funzionalità

5.2 Database relazionale

****Motivazione**:** Sostituzione dei file di testo con PostgreSQL:

- ****Integrità dei dati**:** Controlli di integrità referenziale
- ****Sicurezza**:** Gestione sicura delle credenziali
- ****Performance**:** Query ottimizzate e indici
- ****Scalabilità**:** Gestione di grandi volumi di dati

5.3 Gestione delle sessioni

****Motivazione**:** Implementazione di sessioni isolate:

private static ConcurrentHashMap<String, UserID> userSessions = new ConcurrentHashMap<>();

- ****Isolamento**:**

Ogni client ha una sessione univoca

- ****Concorrenza**:**

Gestione thread-safe delle sessioni

- ****Sicurezza**:**

Impossibilità di accesso ai dati di un utente da altri utenti

5.4 Setup automatico

Motivazione:

DatabaseInitializer per la configurazione automatica:

- **Semplicità**:

Setup guidato per l'utente

- **Affidabilità**:

Creazione automatica delle tabelle

- **Completo**:

Inserimento automatico di dati dei libri

---

6. Scelte algoritmiche

6.1 Gestione delle connessioni

```java
private synchronized Connection getConnection() throws SQLException {
    if (connection == null || connection.isClosed()) {
        connection = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD);
    }
    return connection;
}
```

Motivazione:

Pooling delle connessioni per ottimizzare le performance.

6.2 Validazione dei dati

```**java**

public static synchronized boolean verificaCodiceFiscale(String codice)

public static synchronized boolean verificaPassword(String password)

public static synchronized boolean verificaEmail(String email)

public static synchronized Boolean verificaUserIDEsistente(String user)

**Motivazione**:

Controlli di validazione per garantire l'integrità dei dati.

### 6.3 Hashing delle password

    **java**

private synchronized String hashedPassword(String password) {
   MessageDigest md = MessageDigest.getInstance("SHA-256");
   byte[] hash = md.digest(password.getBytes(StandardCharsets.UTF_8));
   return Base64.getEncoder().encodeToString(hash);
}

**Motivazione**:

Sicurezza delle credenziali utente.

```

6.4 Gestione delle sessioni

private synchronized UserID getUserFromSession(String sessionId) {
   return userSessions.get(sessionId);
}

Motivazione:

Accesso thread-safe alle sessioni utente.

---

7. Strutture dati utilizzate

7.1 Database PostgreSQL

Tabella `userid`

```sql
CREATE TABLE userid (
    nome_cognome VARCHAR(100) NOT NULL,
    codice_fiscale VARCHAR(16) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    userid VARCHAR(50) PRIMARY KEY,
    password VARCHAR(256) NOT NULL
);
```

```````````````````````````````````````````````````````````````````````````

#### **Tabella `libri`**

```sql
CREATE TABLE libri (
    titolo VARCHAR(100) PRIMARY KEY,
    autore VARCHAR(100) NOT NULL,
    genere VARCHAR(50) NOT NULL,
    editore VARCHAR(100),
    anno INT NOT NULL
);
```

```````````````````````````````````````````````````````````````````````````

Tabella `valutazioni`

```sql
CREATE TABLE valutazioni (
    userid VARCHAR(50) NOT NULL,
    titolo_libro VARCHAR(500) NOT NULL,
    stile INT NOT NULL CHECK (stile >= 1 AND stile <= 5),
    contenuto INT NOT NULL CHECK (contenuto >= 1 AND contenuto <= 5),
    gradevolezza INT NOT NULL CHECK (gradevolezza >= 1 AND gradevolezza <= 5),
    originalita INT NOT NULL CHECK (originalita >= 1 AND originalita <= 5),
    edizione INT NOT NULL CHECK (edizione >= 1 AND edizione <= 5),
    voto_finale FLOAT NOT NULL,
    PRIMARY KEY (userid, titolo_libro),
    FOREIGN KEY (userid) REFERENCES userid(userid) ON DELETE CASCADE,
    FOREIGN KEY (titolo_libro) REFERENCES libri(titolo) ON DELETE CASCADE
);
```

Tabella `consigli`

```sql
CREATE TABLE consigli (
    userid VARCHAR(50) NOT NULL,
    libro_referenziale VARCHAR(500) NOT NULL,
    libro_consigliato VARCHAR(500) NOT NULL,
    FOREIGN KEY (userid) REFERENCES userid(userid) ON DELETE CASCADE,
    FOREIGN KEY (libro_referenziale) REFERENCES libri(titolo) ON DELETE CASCADE,
    FOREIGN KEY (libro_consigliato) REFERENCES libri(titolo) ON DELETE CASCADE
);
```

Tabella `librerie`

sql

```
CREATE TABLE librerie (
   id SERIAL PRIMARY KEY,
   userid VARCHAR(50) NOT NULL,
   nome_libreria VARCHAR(100) NOT NULL,
   libro VARCHAR(500),
   UNIQUE(userid, nome_libreria, libro),
   FOREIGN KEY (userid) REFERENCES userid(userid) ON DELETE CASCADE,
   FOREIGN KEY (libro) REFERENCES libri(titolo) ON DELETE CASCADE
);
```

```````````````````````````````````````````````````````````````````````````````

### 7.2 Strutture in memoria

#### **Gestione delle sessioni**

```java
private static ConcurrentHashMap<String, UserID> userSessions = new
ConcurrentHashMap<>();
```

#### **Connessione database**

**java**

```
private static Connection connection = null;
```

---
```

## 8. Gestione delle sessioni

### 8.1 Sistema multi-utente

Il sistema supporta più utenti contemporaneamente attraverso:

**- **Session ID univoco**:**

Ogni client riceve un UUID univoco

**- **Isolamento delle sessioni**:**

Ogni client ha una sessione indipendente

**- **Concorrenza**:**

Gestione thread-safe delle operazioni

### 8.2 Implementazione

**Java**

```
// Generazione Session ID
public Client() {
   this.sessionId = UUID.randomUUID().toString();
}

// Gestione sessioni
private synchronized UserID getUserFromSession(String sessionId) {
   return userSessions.get(sessionId);
}
```

***// Login***

```
public synchronized String login(String sessionId, String username, String password) {
   // Verifica credenziali

   // Inserimento in userSessions

}
```

***// Logout***

```
public synchronized String logout(String sessionId) {
   userSessions.remove(sessionId);
}
```

```
```

### 8.3 Sicurezza delle sessioni

**- **Isolamento**:**

Impossibilità di accesso ai dati di altri utenti

**- **Timeout**:**

Gestione automatica delle sessioni scadute

**- **Validazione**:**

Controlli di validità delle sessioni

---

## 9. Sicurezza e validazione

### 9.1 Hashing delle password

```java
private synchronized String hashedPassword(String password) {
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] hash = md.digest(password.getBytes(StandardCharsets.UTF_8));
        return Base64.getEncoder().encodeToString(hash);
    } catch (NoSuchAlgorithmException e) {
        return String.valueOf(password.hashCode());
    }
}
```

### 9.2 Sistema di recupero password

#### **Recupero informazioni utente**

```java
public synchronized String recuperaPassword(String username) {
    // Verifica esistenza utente
    // Mostra informazioni (senza password originale)
    // Offre opzioni di recupero
}
```

#### **Generazione password temporanea**

```java
public synchronized String generaPasswordTemporanea(String username) {
    // Genera password sicura di 12 caratteri
    // Mix di lettere maiuscole, minuscole, numeri, caratteri speciali
    // Aggiorna password nel database
    // Restituisce password temporanea
}
```

#### **Generazione password casuale**

```java
private String generaPasswordCasuale() {
    // 12 caratteri con mix di tipi
    // Almeno una lettera maiuscola, minuscola, numero, carattere speciale
    // Mischia posizioni per maggiore sicurezza
}
```

### 9.3 Validazione dei dati

#### **Codice fiscale**

```java
public static synchronized boolean verificaCodiceFiscale(String codice) {
    return codice.matches("^[A-Z]{6}\\d{2}[A-Z]\\d{2}[A-Z]\\d{3}[A-Z]$");
}
```

#### **Password**

```java
public static synchronized boolean verificaPassword(String password) {
   return password.length() >= 8 && password.matches(".*[a-zA-Z].*") &&
password.matches(".*\\d.*");
}
```

#### **Email**

```java
public static synchronized boolean verificaEmail(String email) {
   return email.matches("^[A-Za-z0-9+_.-]+@(.+)$");
}
```

### 9.4 Controlli di integrità

- **Database**: Controlli di integrità referenziale
- **Validazione input**: Controlli sui dati inseriti dall'utente
- **Gestione errori**: Messaggi di errore informativi senza esposizione di dati sensibili
- **Recupero password**: Sistema sicuro senza esposizione password originali

---

## 10. Configurazione del database

### 10.1 Setup automatico

Il sistema include una classe `DatabaseInitializer` che automatizza completamente la configurazione:

```**java**
*// Richiesta password*
private static boolean requestDatabasePassword()

*// Creazione tabelle*
private void createTables()

*// Inserimento dati dei libri*
Classe Trasferimento.java
```

### 10.2 Script di configurazione

#### **setup_database.bat** (Windows Batch)

```batch
@echo off
echo ========================================
echo BookRecommender Database Setup
echo ========================================
# ... configurazione automatica
```

#### **setup_database.ps1** (PowerShell)

```powershell

Write-Host "=======================================" -ForegroundColor Cyan

Write-Host "BookRecommender Database Setup" -ForegroundColor Cyan

*# ... configurazione automatica*

```

### 10.3 Gestione password dinamica

**Caratteristiche:**

- Password richiesta all'avvio del programma
- Test automatico della connessione
- Possibilità di retry in caso di errore
- Nessuna password hardcodata nel codice

**Esempio di utilizzo:**

**=== BookRecommender Database Setup ===**

Inserisci la password del database PostgreSQL:

Password: ****

✅ Password del database verificata correttamente!

---

## Conclusioni

Il progetto BookRecommender rappresenta un'evoluzione significativa rispetto alla versione originale, passando da un'architettura monolitica basata su file di testo a un sistema distribuito client-server con database relazionale.

### Principali miglioramenti:

1. **Architettura distribuita**: Client-server con comunicazione RMI
2. **Database relazionale**: PostgreSQL per la persistenza dei dati
3. **Sistema multi-utente**: Sessioni isolate per ogni client
4. **Setup automatico**: Configurazione semplificata del database
5. **Sicurezza migliorata**: Hashing delle password e validazione dei dati
6. **Scalabilità**: Supporto per più utenti contemporaneamente

### Tecnologie utilizzate:

- **Java 17+**:

Linguaggio di programmazione principale

- **Java RMI**:

Comunicazione client-server

- **PostgreSQL**:

Database relazionale

- **JDBC**:

Connessione al database

- **ConcurrentHashMap**:

Gestione delle sessioni concorrenti

## 6. Sitografia

- [JAR Files: The Basics](https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html)
- [Java recognition problems](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Java-Not-Recognized-Error-Fix)
- [Classes, Methods and Paths in Java](https://stackoverflow.com/)
- [Input, Output in Java](https://www.w3schools.com/java/)
- [PostgreSQL Downloads](https://www.postgresql.org/download/)
- [Oracle Java Downloads](https://www.oracle.com/java/technologies/downloads/)
- [OpenJDK Downloads](https://adoptium.net/)

Il sistema è ora più robusto, scalabile e sicuro, mantenendo al contempo la semplicità d'uso per gli utenti finali.

---

**Versione:** 3.0

**Data:** 08/2025

**Compatibile con:**

BookRecommender v3.0 (Architettura client-server con database)