****

**Indice:**

**da approfondire e sistemare il db, come inserire la concorrenza (conclusioni)**

* Introduzione…………………………………………………………………………………
* Struttura del Codice( suddivisione delle Classi)……………………………………………..
* Librerie Usate………………………………………………………………………………..
* Strutture Dati Utilizzate……………………………………………………………………..
* Scelte Tecniche (JavaFX vs CLI)…………………………………………………………….
* Esempio di Complessità del Codice ………………………………………………………….
* Grafica del Programma………………………………………………………………………..
* Conclusioni …………………………………………………………………………………..
* **Introduzione**

Il progetto "Book Recommender" è un'applicazione desktop basata su JavaFX che permette agli utenti di gestire librerie personali di libri, valutarli e ricevere raccomandazioni basate sulle preferenze. Il sistema implementa un'architettura MVC (Model-View-Controller) con interfaccia grafica moderna e utilizza un database PostreSQL per la persistenza dei dati.

* **Struttura del Codice (suddivisione delle Classi)**

Il progetto è organizzato in diverse classi controller, modelli e servizi, ciascuno con responsabilità specifiche:

* **Homepage.java**
* **Responsabilità**: Punto di ingresso dell'applicazione JavaFX
* **Funzionalità principali**:
  + Inizializza e avvia l'interfaccia grafica
  + Carica la scena principale da homepage.fxml
  + Configura dimensioni iniziali e minime della finestra (800x600, min 600x400)

@Override

public void start(Stage primaryStage) throws Exception {

Parent root = FXMLLoader.load(getClass().getResource("/book\_recommender/lab\_b/homepage.fxml"));

primaryStage.setTitle("Book Recommender");

Scene scene = new Scene(root, INITIAL\_WIDTH, INITIAL\_HEIGHT);

primaryStage.setScene(scene);

primaryStage.setMinWidth(MIN\_WIDTH);

primaryStage.setMinHeight(MIN\_HEIGHT);

primaryStage.setResizable(true);

primaryStage.show();

}

* **HomepageController.java**
* **Responsabilità**: Controller per la schermata principale
* **Funzionalità principali**:
  + Gestisce la ricerca libri per titolo, autore e autore+anno
  + Visualizza i top 3 libri più votati
  + Gestisce la navigazione verso login e registrazione
  + Controlla i tab di ricerca e aggiorna l'interfaccia
* **LoginController.java**
* **Responsabilità**: Gestione dell'autenticazione utenti
* **Funzionalità principali**:
  + Valida credenziali (userID e password)
  + Legge dal file UtentiRegistrati.csv
  + Gestisce la navigazione al menu utente dopo login
  + Supporta login con tasto Enter
* **RegistrationController.java**
* **Responsabilità**: Gestione registrazione nuovi utenti
* **Funzionalità principali**:
  + Valida i dati di registrazione
  + Implementa regex per validazione email, codice fiscale e password
  + Verifica duplicazione userID
  + Salva nuovi utenti in UtentiRegistrati.csv
* **UserMenuController.java**
* **Responsabilità**: Gestione menu principale utente autenticato
* **Funzionalità principali**:
  + Visualizza opzioni: crea libreria, aggiungi libri, valuta, consiglia
  + Gestisce il logout
  + Mostra messaggi di stato
  + Naviga verso le diverse funzionalità
* **CreateLibraryController.java**
* **Responsabilità**: Creazione nuove librerie
* **Funzionalità principali**:
  + Valida nome libreria
  + Salva in Librerie.dati.csv
  + Naviga automaticamente alla schermata di aggiunta libri
* **LibrarySelectionController.java**
* **Responsabilità**: Selezione librerie esistenti
* **Funzionalità principali**:
  + Carica librerie utente da CSV
  + Gestisce navigazione in base all'operazione (add, rate, recommend)
  + Supporta selezione con click e Enter
* **AddBooksToLibraryController.java**
* **Responsabilità**: Gestione aggiunta libri alle librerie
* **Funzionalità principali**:
  + Ricerca libri con tre modalità (titolo, autore, autore+anno)
  + ListView con libri selezionati e pulsanti di rimozione
  + Salvataggio in Librerie.dati.csv
  + Gestione eliminazione libreria vuota con conferma
* **BookSelectionController.java**
* **Responsabilità**: Selezione libro da una libreria specifica
* **Funzionalità principali**:
  + Carica libri della libreria selezionata
  + Naviga verso valutazione o raccomandazione
  + Supporta diversi tipi di operazione
* **RateBookController.java**
* **Responsabilità**: Sistema di valutazione libri
* **Funzionalità principali**:
  + Valutazione 5 caratteristiche con stelle cliccabili
  + TextArea con limite caratteri (256) per commenti
  + Calcolo e visualizzazione media valutazioni
  + Salvataggio in ValutazioniLibri.csv
  + Carica valutazioni esistenti
* **RecommendBookController.java**
* **Responsabilità**: Sistema raccomandazione libri
* **Funzionalità principali**:
  + Limite massimo 3 libri consigliati
  + Ricerca e selezione libri da consigliare
  + ListView con libri consigliati e rimozione
  + Salvataggio in ConsigliLibri.dati.csv
* **BookDetailsController.java**
* **Responsabilità**: Visualizzazione dettagli completi libro
* **Funzionalità principali**:
  + Mostra informazioni bibliografiche
  + Visualizza valutazioni con stelle
  + Recensioni organizzate per caratteristica
  + Sistema raccomandazioni basato su categoria
  + Carica consigli personalizzati
* **Book.java**
* **Modello**: Rappresenta un libro
* **Attributi**: title, authors, category, publisher, publishYear
* **Costruttori**: Standard e da array CSV
* **BookService.java**
* **Servizio**: Gestione centralizzata libri
* **Funzionalità**:
  + Caricamento libri da CSV
  + Ricerca per titolo, autore, autore+anno
  + Parsing CSV con gestione virgolette
  + Calcolo libri più votati
* **RecommendedBook.java**
* **Modello**: Rappresenta un libro consigliato
* **Attributi**: userId, bookTitle, authorName, category
* **Librerie Usate**
* **JavaFX Core**
* **javafx.controls**: Componenti UI (Button, TextField, ListView, etc.)
* **javafx.fxml**: Binding FXML-Controller
* **javafx.base**: Classi base JavaFX
* **Librerie di Supporto**
* **ControlsFX (11.2.1)**: Componenti UI avanzati
* **FormsFX (11.6.0)**: Gestione form complessi
* **ValidatorFX (0.5.0)**: Validazione input
* **JUnit Jupiter (5.10.2)**: Testing unitario

**Java Standard Library**

* **java.io**: I/O file (BufferedReader, FileWriter)
* **java.util**: Collections e utilities
* **java.util.regex**: Pattern matching per validazioni
* **java.nio.file**: Operazioni file system moderne

**4. Strutture Dati Utilizzate**

**ArrayList<String>**

* **Utilizzo**: Liste di libri, librerie, risultati ricerca
* **Vantaggi**: Accesso indicizzato O(1), ridimensionamento dinamico
* **Esempio**: recommendedBooks in RecommendBookController

**HashSet<String>**

* **Utilizzo**: Libri selezionati in AddBooksToLibraryController
* **Vantaggi**: Nessun duplicato, ricerca O(1)
* **Esempio**: selectedBooks per evitare duplicati

**HashMap<String, String>**

* **Utilizzo**: Colori utenti in BookDetailsController
* **Vantaggi**: Associazione rapida chiave-valore
* **Esempio**: userColors per colorare recensioni

**List<Book>**

* **Utilizzo**: Risultati ricerca, libri caricati
* **Vantaggi**: Interfaccia flessibile, ordinamento
* **Esempio**: searchResults in controllers di ricerca

**Pattern/Matcher**

* **Utilizzo**: Validazione input in RegistrationController
* **Vantaggi**: Validazioni regex potenti
* **Esempio**: EMAIL\_PATTERN, CF\_PATTERN, PASSWORD\_PATTERN

**5. Scelte Tecniche (JavaFX vs CLI)**

**JavaFX - Vantaggi**

* **GUI Moderna**: Interfaccia grafica user-friendly
* **FXML**: Separazione netta UI/logica
* **Event Handling**: Gestione eventi intuitiva
* **CSS Styling**: Personalizzazione aspetto
* **Componenti Rich**: ListView, TableView, etc.
* **Data Binding**: Aggiornamento automatico UI

**CSV per Persistenza - Vantaggi**

* **Semplicità**: File testo facilmente editabili
* **Portabilità**: Nessuna dipendenza database
* **Leggibilità**: Debug e ispezione facili
* **Compatibilità**: Importabile in Excel/Sheets

**Architettura MVC**

* **Separazione Concerns**: Modelli, Viste, Controller separati
* **Manutenibilità**: Codice organizzato e modulare
* **Estensibilità**: Facile aggiungere nuove funzionalità

**6. Esempio di Complessità del Codice**

**saveRecommendationsToFile() in RecommendBookController**

private boolean saveRecommendationsToFile(String filePath, String separator) {

try {

File dataDir = new File("data");

if (!dataDir.exists()) {

dataDir.mkdirs();

}

List<String> lines = new ArrayList<>();

boolean found = false;

File file = new File(filePath);

if (file.exists()) {

try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {

String line;

String header = br.readLine();

if (header != null) {

lines.add(header);

}

while ((line = br.readLine()) != null) {

String[] data = line.split(separator.equals(",") ? "," : "\\s+");

if (data.length >= 2 && data[0].trim().equals(userId.trim())

&& data[1].trim().equals(selectedBook.trim())) {

// Sostituisci riga esistente

StringBuilder sb = new StringBuilder();

sb.append(userId).append(separator).append(selectedBook);

for (int i = 0; i < 3; i++) {

sb.append(separator);

if (i < recommendedBooks.size()) {

String bookTitle = recommendedBooks.get(i);

if (separator.equals(" ")) {

bookTitle = bookTitle.replace(" ", "\_");

}

sb.append(bookTitle);

} else {

sb.append("null");

}

}

lines.add(sb.toString());

found = true;

} else {

lines.add(line);

}

}

}

}

// Se non trovato, aggiungi nuova riga

if (!found) {

// ... codice per aggiungere nuova riga

}

Files.write(Paths.get(filePath), lines,

StandardOpenOption.CREATE,

StandardOpenOption.TRUNCATE\_EXISTING);

return true;

} catch (IOException e) {

errorLabel.setText("Errore: " + e.getMessage());

errorLabel.setVisible(true);

return false;

}

}

**Complessità di tempo**: O(n) dove n è il numero di righe nel file

**Complessità di spazio**: O(n) per memorizzare tutte le righe in memoria

**7. Grafica del Programma**

**Layout e Stile**

* **Colori Principali**:
  + Header: #4054B2 (blu)
  + Pulsanti azione: #75B965(lightgreen) o green
  + Pulsanti cancella: red
  + Stelle: #f2e485 (giallo)

**Componenti UI Principali**

1. **Homepage**: Tab per ricerca (Titolo/Autore/Anno)
2. **Login/Registrazione**: Form con validazione
3. **Menu Utente**: 4 pulsanti principali
4. **Valutazione**: Stelle cliccabili + commenti
5. **Raccomandazione**: ListView con limite 3 libri

**Responsive Design**

* Finestre ridimensionabili
* Layout flessibili con HBox/VBox
* ScrollPane per contenuti lunghi

**8. Conclusioni**

**Punti di Forza**

1. **Interfaccia User-Friendly**: GUI intuitiva con JavaFX
2. **Architettura Modulare**: MVC ben implementato
3. **Persistenza Semplice**: CSV efficaci per questo caso d'uso
4. **Validazioni Robuste**: Regex per input utente
5. **Feedback Visivo**: Messaggi errore/successo chiari

**Possibili Miglioramenti**

1. **Sicurezza**: Hash password, SSL per comunicazioni
2. **Caching**: Cache per libri frequentemente acceduti
3. **Internazionalizzazione**: Supporto multilingua
4. **Testing**: Aumentare copertura test unitari
5. **Concorrenza**: Gestione accessi simultanei ai file

**Conclusione Finale**

Il progetto "Book Recommender " rappresenta un'implementazione solida di un sistema di raccomandazione libri con interfaccia grafica. L'architettura MVC, l'uso di JavaFX e la persistenza su PostreSQL lo rendono perfetto per una consultazione veloce.