DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PSG COLLEGE OF TECHNOLOGY

COIMBATORE – 641 004

B.E COMPUTER SCIENCE AND ENGINEERING (G2) 2023-27

23Z310 OBJECT ORIENTED PROGRAMMING

MOVIE RECOMMENDATION APPLICATION REPORT

NAME: Neelesh V

ROLL NO: 23Z350

Language: Java

Tools: Netbeans IDE, sqlite

Abstract

The Movie Recommendation Application is a robust platform designed to enhance user engagement by providing personalized movie suggestions, allowing users to create watchlists, mark movies as watched, and maintain a diary of movie experiences. Leveraging Object-Oriented Programming (OOP) principles and Java features, the application integrates efficient backend functionalities with an intuitive user interface to deliver a seamless experience.

Introduction

This application addresses the growing need for personalized content in the entertainment industry. By enabling users to search for movies, maintain a diary, and create a watchlist, the application enhances user convenience and engagement. Developed using Java, the platform emphasizes modular design, code reusability, and robust database interactions. The application's ability to export data and integrate secure authentication highlights its suitability for modern-day users seeking customized entertainment solutions.

Chapter 1: Application Design and Structure

The design of the Movie Recommendation Application is modular, adhering to OOP principles for scalability and maintainability. The architecture comprises the following key components:

1.1 OOP Concepts Used

1. **Encapsulation**: Encapsulation is achieved through private class fields and public getter and setter methods, ensuring controlled access to data. By exposing only necessary data and hiding implementation details, it promotes data security and simplifies debugging.
2. **Inheritance**: Abstract classes like UserAbstract serve as templates for derived classes such as User. This enables code reuse, reduces redundancy, and ensures consistency in derived classes by enforcing common behaviors.
3. **Polymorphism**: Interfaces like MovieInterface allow for flexible implementation in the Movie class. Polymorphism supports method overriding, enabling the application to handle diverse objects uniformly, thus improving extensibility.
4. **Abstraction**: Abstract classes and interfaces abstract away implementation details, focusing on essential functionalities. This reduces complexity, enhances maintainability, and simplifies collaboration among developers.

1.2 Java Features Used

1. **Collections Framework**: The use of lists for managing movie data ensures efficient storage, retrieval, and manipulation of large datasets. This framework provides robust data structures like ArrayList and HashMap to support dynamic data handling.
2. **JDBC**: Java Database Connectivity is used for seamless interaction with a relational database, enabling CRUD operations. JDBC ensures secure and efficient data transactions through prepared statements and connection pooling.
3. **Exception Handling**: Try-catch blocks are employed to handle runtime errors gracefully. By catching and managing exceptions effectively, the application minimizes crashes and ensures smooth user experiences.
4. **File Handling**: File handling is used to export data such as watchlists and diaries as text files. This feature improves data portability, allowing users to access their information offline or share it across devices.
5. **Multithreading**: Although not fully implemented, the design allows for extending functionality with asynchronous operations. Multithreading can optimize tasks such as database queries and file exports by running processes concurrently.

1.3 Application Features

1. Secure user login and authentication using password hashing.
2. A search feature to find movies by title, ensuring user convenience.
3. Ability to maintain a diary of watched movies with personalized ratings and viewing dates.
4. Tools to create and manage a movie watchlist.
5. Functionality to export the diary, watchlist, and other data as text files for offline access.

1.4 Technical Implementation

Core Classes

1. **User**: Responsible for user data, authentication, and validation.
2. **Movie**: Implements MovieInterface to encapsulate movie details such as title, genre, and director.
3. **Diary**: Tracks watched movies and associates them with user reviews and ratings.
4. **MovieSearch**: Handles search queries, fetching relevant data from the database based on user input.

Database Interaction

The application leverages JDBC for database operations, ensuring reliable data management. SQL queries are executed to manage movie and user data effectively. The properties class encapsulates database connection parameters, enhancing security and reusability.

User Authentication

The SHA-256 algorithm ensures secure password hashing, protecting user credentials. This feature minimizes risks associated with data breaches.

Sample Workflow

1. User registers or logs in securely.
2. Searches for movies using the intuitive search functionality.
3. Adds movies to their watchlist or marks them as watched.
4. Reviews, updates, and exports data as text files.

Chapter 2: User Interface

The user interface is designed for simplicity and usability, featuring distinct views for various functionalities:

1. **Default View**: Displays an overview of available features and quick navigation options.
2. **Login View**: Facilitates secure user authentication with clear input prompts.
3. **Movie Display View**: Lists movies based on search results with detailed information.
4. **Diary View**: Presents watched movies with ratings and viewing dates in an organized manner.
5. **Watchlist View**: Allows users to manage and prioritize movies they intend to watch.

Chapter 3: Limitations and Areas for Improvement

3.1 Limitations

- Dependency on a stable internet connection for database operations.
- Limited optimization for handling large datasets, affecting performance as data scales.
- The recommendation algorithm is rudimentary, relying solely on keyword matching.

3.2 Areas for Improvement

1. **Advanced Recommendations**: Implement machine learning algorithms to provide personalized movie suggestions based on viewing history.
2. **Scalability**: Transition to a cloud-based database to handle larger datasets and improve accessibility.
3. **Enhanced UI/UX**: Upgrade the user interface with modern frameworks like JavaFX or React for an improved visual experience.
4. **Offline Functionality**: Introduce offline modes where users can access their data without an internet connection.

Conclusion

The Movie Recommendation Application demonstrates the effective use of OOP principles and Java features to build a functional and user-friendly platform. Its modular design ensures scalability, while its intuitive interface simplifies user interactions. By incorporating advanced recommendation systems and addressing current limitations, the application holds promise as a comprehensive solution in the personalized entertainment domain.