WEB BASED MOVIE RECOMMENDATION SYSTEM USING JAVA EE

A Major Project-I Report
Submitted in Partial fulfillment for the award of
Bachelor of Technology in Department of Computer Science & Engineering

Submitted to
RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA
BHOPAL (M.P)

MAJOR PROJECT-I REPORT
Submitted by

Aditya Barodia [0103CS211011] Neelpa Raj [0103CS211115]
Nishant Kumar Nagesh [0103CS211120]

Under the supervision of
Prof. Himanshu Barhaiya
Assistant Professor

Department of CSE

Lakshmi Narain College of Technology, Bhopal (M.P.)

Session
2024-25

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY, BHOPAL

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (CSE)

CERTIFICATE

This is to certify that the work embodied in this project work entitled **”Movie Recommendation System using JAVA EE”** has been satisfactorily completed by the **AdityaBarodia** (0103CS211011), **Neelpa Raj** (0103CS211115), **Nishant Kumar Nagesh** (0103CS211120)**.** It is a bonafide piece of work, carried out under the guidance in **Department of CSE**, **Lakshmi Narain College of Technology, Bhopal** for the partial fulfillment of the **Bachelor of Technology** during the academic year 2024-2025.

Guide By	**Approved By**
Prof. Himanshu Barhaiya	**Dr. Vivek Richhariya**
Assistant Professor (CSE)	**Professor & Head (CSE)**

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY, BHOPAL

DEPARTMENT OF CSE

ACKNOWLEDGEMENT

We express our deep sense of gratitude to Prof. Himanshu Barhaiya department of CSE L.N.C.T., Bhopal. Whose kindness valuable guidance and timely help encouraged me to complete this project.

A special thank goes to Dr. Vivek Richhariya (HOD) who helped me in completing this project work. He exchanged his interesting ideas & thoughts which made this project work successful.

We would also thank our institution and all the faculty members without whom this project work would have been a distant reality.

Aditya Barodia [0103CS211011]   Neelpa Raj [0103CS211115]

Nishant Kumar Nagesh [0103CS211120]

INDEX

CHAPTER 1

PROBLEM DOMAIN DESCRIPTION

In today's era of abundant digital content, the world of cinema stands as a vast and sprawling landscape of artistic expression, storytelling, and entertainment. With an ever-expanding catalog of movies spanning genres, languages, and cultures, navigating this cinematic universe to find content tailored to individual preferences can be a daunting task. This challenge of movie discovery is where the Movie Recommendation System, powered by Java Enterprise Edition (JAVA EE), emerges as a transformative solution.

A recommendation system is an intelligent software application that analyzes user data, preferences, and behaviors to provide personalized suggestions or recommendations. In the realm of movies, a recommendation system becomes indispensable as it addresses the inherent complexity and abundance of choices faced by users. Rather than relying solely on generic browsing methods or word-of-mouth recommendations, a movie recommendation system harnesses the power of data analytics and machine learning algorithms to deliver curated suggestions tailored to each user's unique tastes and preferences.

The need for a movie recommendation system stems from several factors. Firstly, the sheer volume of available movies makes manual browsing and selection a time-consuming and overwhelming task for users. With thousands of titles released annually across various platforms and genres, users can easily feel. Additionally, traditional methods of movie discovery, such as genre-based searches or top-rated lists, often fail to capture the nuanced preferences of individual users, resulting in missed opportunities for discovering hidden gems or niche content.

Moreover, as streaming platforms continue to proliferate and diversify, the problem of content overload exacerbates, further complicating the task of finding relevant and enjoyable movies. Users may find themselves spending more time searching for content than actually watching it, leading to frustration and dissatisfaction. In this context, a movie recommendation system becomes not just a convenience but a necessity, offering users a guided and personalized path through the vast array of available movies.

CHAPTER 2

LITERATURE SURVEY

2.1 Background and Researches

• Introduction: The Movie Recommendation System represents a critical application in the domain of information filtering and recommendation systems. Its primary goal is to assist users in discovering relevant and enjoyable movies from a vast selection based on their preferences and historical behavior. In this project, we focus on developing a recommendation system using Java Enterprise Edition (JAVA EE) that recommends movies based on the similarity score with other raters' movie ratings, obtained by asking ratings from the users themselves.

• Collaborative Filtering Techniques: It is a widely used approach in recommendation systems that leverage the collective wisdom of a group of users to generate personalized recommendations [1]. It operates under the assumption that users who have similar tastes and preferences in the past are likely to have similar tastes in the future. Collaborative filtering can be categorized into user-based and item-based methods, both of which have been extensively studied and applied in recommendation systems.

• User-Based Collaborative Filtering: It computes the similarity between users based on their past interactions with items (movies) [2]. It identifies users with similar rating patterns and recommends items that these similar users have liked or rated highly.

• Item-Based Collaborative Filtering: It filtering computes the similarity between items based on the ratings given by users [2]. It identifies items that are similar to the ones a user has liked or rated highly and recommends those similar items to the user.

• Research Studies and Techniques: Numerous research studies and techniques have been proposed and implemented in the field of movie recommendation systems [3]. These studies explore various aspects of collaborative filtering algorithms, data preprocessing techniques, similarity metrics, and evaluation methods to improve the performance and effectiveness of recommendation systems.

• Implementation Using JAVA EE: Implementing a Movie Recommendation System using JAVA EE involves integrating collaborative filtering techniques into the system architecture [4]. JAVA EE provides a robust platform for building scalable and efficient web applications capable of handling user interactions and data management.

2.2 Existing System

The current Movie Recommendation System is a web-based application developed using Java Enterprise Edition (JAVA EE). It serves as a platform for users to explore and discover movies based on predefined criteria such as genre, popularity, or release date. However, the system lacks personalized recommendations tailored to individual user preferences. Its user interface is a basic web application accessible through standard web browsers. Users can browse through movie categories, view popular titles, and search for specific movies based on predefined filters. The backend architecture relies on JAVA EE technologies like Servlets, JSP, and JDBC to handle user requests, retrieve data from the database, and render dynamic web pages. While the system utilizes a relational database management system (RDBMS) to store movie information and user profiles, it lacks a comprehensive data model for capturing user preferences and ratings. Additionally, the existing system does not incorporate a sophisticated recommendation engine. Instead, it relies on predefined rules or simple algorithms to recommend movies based on generic criteria such as genre or popularity. User interaction with the system is limited to browsing available movie categories, viewing movie details, and adding movies to their watch list, with few options for providing feedback or indicating preferences.

2.3 Proposed System

The proposed Movie Recommendation System aims to enhance the existing system by introducing personalized movie recommendations based on collaborative filtering techniques. Specifically, the system will recommend movies based on the similarity score with other raters' movie ratings, obtained by asking ratings from the users themselves. In this proposed system, the user interface will undergo significant enhancements to accommodate features for collecting user ratings, displaying personalized recommendations, and providing interactive feedback mechanisms. The backend architecture will be extended to incorporate additional components for data processingand recommendation generation. This comprehensive data model will enable the system to capture and analyze user behavior effectively. Users will have the opportunity to provide ratings for movies they have watched, which will serve as input to the recommendation engine to generate personalized recommendations. Additionally, users will have options to provide feedback, update preferences, and explore recommended movies in more detail. The proposed system aims to significantly enhance the movie discovery experience for users, leading to increased engagement and satisfaction.

2.4 Feasibility Study

2.4.1 Technical Feasibility

• Hardware and Software Requirements: The technical feasibility involves evaluating whether the required hardware and software resources are available or can be acquired within budget constraints. Since JAVA EE development primarily requires standard computing hardware and software tools, such as a development environment and database management system the project is technically feasible.

• Expertise and Skillset: Assessing whether the development team possesses the necessary expertise and skillset to implement the project is crucial. JAVA EE development requires proficiency in Java programming, web development, database management, and familiarity with relevant frameworks and libraries. If the team lacks expertise in these areas, additional training or hiring skilled personnel may be required.

2.4.2 Economic Feasibility

• Cost-Benefit Analysis: Conducting a cost-benefit analysis helps determine whether the expected benefits of the project outweigh the costs involved. The costs associated with developing the Movie Recommendation System include hardware and software expenses, development costs (e.g., personnel salaries, training), and operational costs On the other hand, the benefits include improved user experience, increased user engagement, and potentially higher revenue through enhanced movie recommendations. If the projected benefits outweigh the costs, the project is economically feasible.

2.4.3 Operational Feasibility

• User Acceptance: Assessing whether the intended users will accept and adopt the Movie Recommendation System is crucial for operational feasibility. Conducting user surveys, interviews, or focus groups can provide insights into user preferences, expectations, and potential challenges. If users express a strong desire for personalized movie recommendations and show willingness to use the system, it indicates operational feasibility.

• Integration with Existing Systems: Evaluating the compatibility and integration of the proposed system with existing infrastructure, platforms, and databases is essential. If the system can seamlessly integrate with other systems without significant disruptions or modifications, it enhances operational feasibility.

2.5 Related Concepts

2.5.1 Web Application

A web application is a software application that is accessed and interacted with through a web browser over the internet. Unlike traditional desktop applications that are installed locally on a user's device, web applications are hosted on remote servers and accessed via a web browser such as Google Chrome, Mozilla Firefox, or Microsoft Edge.

Components of a Web Application:

• Client-Side Interface: The client-side interface of a web application comprises the user interface elements that users interact with in their web browser. This includes HTML (Hypertext Markup Language) for structuring web pages, CSS (Cascading Style Sheets) for styling and layout, and JavaScript for adding interactivity and dynamic behavior to the interface.
• Server-Side Logic: The server-side logic of a web application handles processing and manipulation of data, as well as business logic and application functionality. This typically involves programming languages such as Java, Python, PHP, Ruby, or JavaScript (Node.js) running on the server-side. Frameworks and libraries like Java EE, Spring Boot, Django, Flask, and Express.js are commonly used to streamline server-side development.
• Database: Web applications often require a database to store and manage persistent data. Relational database management systems (RDBMS) like MySQL, PostgreSQL, Oracle, and SQL Server are commonly used for web application development. NoSQLdatabases such as MongoDB and Redis are also popular choices for applications with non-relational data storage requirements.
• Networking and Communication: Networking and communication protocols such as HTTP (Hypertext Transfer Protocol) and WebSocket facilitate communication between the client-side interface and the server-side logic. AJAX (Asynchronous JavaScript and XML) enables asynchronous data exchange between the client and server without requiring a full page refresh.

2.5.2 User Interface

User Interface, often abbreviated as UI, refers to the graphical user interface that enables users to interact with software or hardware. The goal of UI is to create a visual representation of a program or system that is intuitive and user-friendly, allowing users to easily accomplish their goals and complete tasks.

Here are some key elements of User Interface:

• Layout: The layout of a UI refers to how the various elements of the interface are arranged on the screen. This includes the placement of buttons, menus, and other visual elements that the user interacts with.

• Color: Color is an important element of UI, as it can be used to convey meaning, create visual interest, and establish brand identity. Colors can be used to differentiate between different elements, highlight important information, or create a certain mood or atmosphere.

• Typography: Typography refers to the font styles, sizes, and layouts used in the UI. The right typography can improve readability, create hierarchy and visual interest, and reinforce the brand identity.

• Icons: Icons are visual representations of actions or concepts, and they are used extensively in UI design. They can make interfaces more intuitive and easy to use by providing visual cues for users.

• Interactivity: Interactivity is an important element of UI, as it allows users to interact with the interface and accomplish tasks. This includes features such as buttons, forms, and menus, as well as animations and other visual feedback that lets users know what actions they are taking.

• Consistency: Consistency is essential in UI design, as it helps to create a sense of familiarity and predictability for users. Consistency means using the same design patterns, styles, and language throughout the UI, making it easy for users to understand and use.

2.5.3 User Experience

User Experience, often abbreviated as UX, refers to the overall experience that a user has while interacting with a product or service. It encompasses all aspects of the user's interaction, including the design, usability, and functionality of the product or service. The goal of UX is to create a positive experience for the user, one that is easy, intuitive, and enjoyable.

Here are some key elements of User Experience:

• User Research: User research is the process of gathering information about users, their needs, and their behaviors. This information is used to inform the design of the product or service and to ensure that it meets the needs of the target audience.

• User Testing: User testing is the process of evaluating the product or service by having users interact with it and provide feedback. This feedback is used to improve the design and ensure that it is meeting the needs of the target audience.

• Usability: Usability refers to how easy it is for users to use the product or service. This includes elements such as navigation, user interface design, and the overall user flow.

• Accessibility: Accessibility refers to how easy it is for users with disabilities to use the product or service. This includes elements such as visual design, audio design, and interaction design.

• Visual Design: Visual design refers to the overall look and feel of the product or service. This includes elements such as color, typography, and layout.

• Interaction Design: Interaction design refers to how users interact with the product or service. This includes elements such as buttons, forms, and menus.

• Content: Content refers to the text, images, and other media used in the product or service. The content should be clear, concise, and engaging, and should support the overall goals of the product or service.

2.5.4 Object Oriented Programming

User Object-oriented programming (OOP) is a programming paradigm that emphasizes the use of objects and classes to design and build software applications. OOP is based on the concept of objects, which are instances of classes that contain data and methods (functions) that operate on that data.

Here are some key features of OOP:

• Abstraction: Abstraction refers to the process of creating abstract representations of real-world objects or concepts. This allows developers to model complex systems in a simplified manner.

• Encapsulation: Encapsulation is the process of hiding the implementation details of an object from the outside world. This prevents unauthorized access to the object's internal state and provides a well-defined interface for interacting with the object.

• Inheritance: Inheritance is a mechanism for creating new classes from existing classes. Inheritance allows developers to reuse code and build class hierarchies that model complex relationships between objects.

• Polymorphism: Polymorphism is the ability of objects to take on multiple forms. Polymorphism allows developers to write code that works with objects of different types, without having to know the specific type at compile-time.

• Classes and Objects: Classes are blueprints that define the structure and behavior of objects. Objects are instances of classes that contain data and methods that operate on that data.

• Modularity: Modularity is the process of breaking down complex systems into smaller, more manageable components. OOP encourages modularity by allowing developers to build classes that encapsulate specific functionality.

2.6 Technology & Libraries Used

2.6.1 Java

Java [5] is a general-purpose, object-oriented programming language that was developed by Sun Microsystems (now owned by Oracle Corporation) in the mid-1990s. It is a high-level language that is known for its simplicity, portability, and security features. Java is widely used for developing desktop applications, web applications, mobile applications, and enterprise software.

Here are some key features of Java:

• Object-Oriented Programming: Java is an object-oriented programming (OOP) language, which means that it is designed to represent real-world objects and their interactions. It allows developers to write code that is organized into classes and objects, making it easy to understand, maintain, and reuse.

• Platform Independence: One of the most important features of Java is its platform independence. Java code can be written once and run on any platform that supports Java, without the need for any modifications. This is achieved through the use of the Java Virtual Machine (JVM), which translates the Java code into byte code that can be executed on any platform.

• Security: Java is designed with built-in security features to protect against common security threats such as viruses, trojans, and other malicious code. Java's security model includes features such as sandboxing, class loaders, and access control.

• Multithreading: Java provides built-in support for multithreading, which allows programs to perform multiple tasks concurrently. This can help to improve performance and responsiveness in applications that need to perform multiple tasks at the same time.

• Rich API: Java comes with a rich set of libraries and APIs (Application Programming Interfaces) that provide developers with a wide range of tools and functions to build powerful and complex applications. These APIs include everything from basic I/O functions to advanced graphics and networking libraries.

2.6.2 Eclipse IDE

Eclipse is a popular open-source integrated development environment (IDE) used for Java development, but it has expanded to support other programming languages and technologies through a system of plugins. Here are the key details about the Eclipse IDE:

Some of its features include:

• Java Development Tools (JDT): Eclipse is well-known for its robust set of tools for Java development. This includes a powerful code editor, debugger, compiler, and various wizards to assist with project creation and management.

• Plugin Architecture: Eclipse is highly extensible due to its plugin architecture. Developers can add plugins to extend functionality for different programming languages, frameworks, and tools.

• Code Assistance: Offers features like code completion, code navigation, and quick fixes to enhance productivity.

• Integrated Debugger: Provides a built-in debugger for identifying and fixing bugs in the code.

• Version Control Integration: Eclipse supports version control systems such as Git, CVS, and SVN, allowing developers to manage source code changes effectively.

• Refactoring Tools: Includes tools for code refactoring, making it easier to restructure and optimize code.

• User Interface Designer: Eclipse includes graphical editors for creating user interfaces, making it easier to design and layout GUI components.

• Build and Deployment Tools: Supports various build systems and deployment configurations.

• Task Management: Eclipse includes a task management system to help developers keep track of their work and priorities.

• Rich Ecosystem: The Eclipse Marketplace allows developers to discover, install, and manage a wide range of plugins, tools, and extensions.

2.6.3 JavaEE

Java EE, which stands for Java Enterprise Edition, is a set of specifications, APIs (Application Programming Interfaces), and runtime environments that provide a platform for developing and deploying enterprise-scale Java applications. Originally developed by Sun Microsystems (now owned by Oracle Corporation), Java EE is designed to simplify and standardize the development of distributed, multi-tiered, and scalable enterprise applications.

Key features and components of Java EE include:

• Servlets and JSP (JavaServer Pages): Servlets are Java classes that handle HTTP requests and generate dynamic web content. JSP is a technology that allows embedding Java code within HTML pages to generate dynamic web content. Together, Servlets and JSP enable the development of dynamic web applications.

• Enterprise JavaBeans (EJB): Enterprise JavaBeans (EJB) is a server-side component model for building scalable, distributed, and transactional enterprise applications. EJB components encapsulate business logic and can be deployed on Java EE application servers to provide services such as persistence, messaging, and security.

• JPA (Java Persistence API): JPA is a Java API for accessing, managing, and persisting data in relational databases. It provides a standard way for Java applications to interact with databases using object-relational mapping (ORM) techniques, allowing developers to work with Java objects rather than SQL queries.

• JMS (Java Message Service): JMS is an API for asynchronous messaging between distributed components in a Java application. It enables reliable communication and integration between different parts of an enterprise application by facilitating the exchange of messages via message queues or topics.

• JTA (Java Transaction API): JTA is an API for managing distributed transactions in Java applications. It provides a standard mechanism for coordinating transactions across multiple resources, ensuring data consistency and integrity in distributed environments.

• JSF (JavaServer Faces): JSF is a component-based web framework for building user interfaces in Java web applications. It provides a rich set of UI components, event handling mechanisms, and lifecycle management features for developing interactive and visually appealing web applications.

2.6.4 Apache Tomcat

Apache Tomcat, often referred to simply as Tomcat, is an open-source web server and servlet container developed by the Apache Software Foundation. It is one of the most widely used Java-based web servers and is particularly popular for hosting Java web applications, including those built using JavaServer Pages (JSP) and Servlets.

Key Features of Apache Tomcat:

• Servlet Container: Apache Tomcat serves as a servlet container, providing an environment for running Java Servlets and JavaServer Pages (JSP). Servlets and JSPs are Java-based technologies used for developing dynamic web applications that generate content dynamically in response to client requests.

• HTTP Server: Tomcat functions as a web server capable of serving static content (HTML, CSS, JavaScript, etc.) over the HTTP protocol. It handles HTTP requests from clients (typically web browsers) and responds with the appropriate content or redirects requests to servlets or other resources.

• Java EE Compatibility: While Tomcat is not a full Java EE application server like Oracle WebLogic or IBM WebSphere, it supports many Java EE specifications and APIs, including Servlet, JSP, JDBC (Java Database Connectivity), JNDI (Java Naming and Directory Interface), and JTA (Java Transaction API). This makes Tomcat suitable for hosting a wide range of Java web applications.

• Embeddable: Tomcat is lightweight and embeddable, meaning it can be easily integrated into other applications or frameworks. Developers can embed Tomcat within their Java applications to provide web server functionality without the need for a standalone server installation.

• Modularity: Tomcat is modular in design, allowing administrators to configure and extend its functionality through the use of various components and modules. Features such as security, logging, and performance tuning can be customized according to the specific requirements of the application.

• Open Source and Community Support: Tomcat is open-source software released under the Apache License, which means it is freely available for download, use, and modification. The Apache Tomcat community actively maintains and supports the project, providing regular updates, bug fixes, and enhancements.

2.6.5 Package edu.duke

The edu.duke package[6] provides a comprehensive set of resource classes designed to facilitate file and data manipulation tasks within the context of the Duke/Coursera course environment. These classes offer functionality for interacting with various types of resources, including directories, files, images, URLs, and data sequences, enabling users to access, manipulate, and analyze data efficiently.

The DirectoryResource class allows users to select one or more files from a directory using a file selection dialog box, providing flexibility in handling multiple files within a directory structure. Similarly, the FileResource class represents individual files and provides methods for accessing their contents line by line or word by word, facilitating easy reading and processing of file data.

For image manipulation tasks, the ImageResource class represents images as grids of Pixel objects, allowing users to access and manipulate individual pixels and their color components. Additionally, the Point class provides functionality for representing two-dimensional locations and calculating distances between points.

The RangeResource class offers support for working with ranges of integer numbers, providing methods for accessing and iterating over sequences of numbers within a specified range. Meanwhile, the Shape class enables the modeling of polygonal shapes using collections of Point objects, allowing for easy representation and manipulation of geometric shapes.

The StorageResource class serves as a versatile storage container for storing and accessing multiple string objects, offering methods for storing and retrieving data values one at a time. Lastly, the URLResource class facilitates the retrieval and processing of web page contents from URLs, providing methods for accessing data line by line or word by word.

2.6.6 Package org.apache.commons.csv

The org.apache.commons.csv package[7] in Apache Commons CSV provides comprehensive support for handling CSV (Comma Separated Values) files, a widely used format for data interchange and importation, particularly in scenarios involving legacy systems or manual data imports. CSV files consist of records separated by newlines, with each record comprising a list of values.

The package offers classes such as CSVParser and CSVRecord to parse CSV files according to specified formats and work with individual records parsed from CSV files. This support extends to various aspects of CSV files, including separators, delimiters, comments, encapsulation, handling of simple and complex values, and options for skipping empty lines.

Key aspects supported by the Commons CSV parser include:

• Separators for Lines: The record separators, which are hardcoded and cannot be changed, must be '\r', '\n', or '\r\n'.

• Delimiter for Values: The delimiter for values within CSV files is configurable and defaults to ','.

• Comments: Certain CSV dialects support a simple comment syntax, where a record starting with a designated character (the commentStarter) is treated as a comment and removed from the input.

• Encapsulator for Complex Values: Complex values, such as those containing the delimiter character, are enclosed within a pair of encapsulator characters, which default to '"'. The encapsulator character itself must be escaped or doubled when used inside complex values.

• Handling of Simple Values: Simple values consist of all characters (excluding the delimiter) until the next delimiter or record terminator. Optionally, surrounding whitespace of simple values can be ignored.

• Handling of Complex Values: Complex values, encapsulated within the defined encapsulator characters, preserve all formatting, including newlines for multiline values.

• Empty Line Skipping: Optionally, empty lines in CSV files can be skipped. Otherwise, empty lines will return a record with a single empty value.

Additionally, the package provides support for predefined dialects such as strict-csv and excel-csv, along with the flexibility to define custom dialects as needed.

2.7 Conclusion

The literature survey conducted for the Movie Recommendation System project using JAVA EE has provided valuable insights into the various approaches, techniques, and methodologies employed in the field of recommendation systems. The survey encompassed a wide range of research papers, articles, and resources related to movie recommendation systems, focusing on methods that recommend movies based on similarity scores derived from user ratings.

In conclusion, the literature survey provided a comprehensive understanding of the state-of-the-art techniques and best practices in movie recommendation systems. By leveraging collaborative filtering methods and user ratings, coupled with the capabilities of Java EE for web application development, the proposed Movie Recommendation System project holds promise for delivering personalized and accurate movie recommendations to users. However, further research and experimentation are warranted to address existing challenges and refine the system's performance in real-world scenarios.

CHAPTER 3

PROJECT OBJECTIVE AND SCOPE

3.1 Objective

The primary objective of the Movie Recommendation System project using JAVA EE is to develop a robust and user-friendly web application capable of providing personalized movie recommendations to users based on similarity scores derived from the ratings of other users. The project aims to address the following specific objectives:

• User-Centric Movie Recommendations: Design and implement a user-centric recommendation system that takes into account the preferences and ratings of individual users to generate personalized movie recommendations. The system will analyze user ratings and behaviors to identify similarities with other users and recommend movies that align with their tastes and preferences.

• Collaborative Filtering Algorithm Implementation: Implement collaborative filtering algorithms, such as user-based or item-based collaborative filtering, to calculate similarity scores between users or items (movies) based on their ratings. These algorithms will form the backbone of the recommendation system, enabling it to generate accurate and relevant movie recommendations.

• User Interaction and Feedback Mechanism: Implement user interaction mechanisms within the web application to gather movie ratings and feedback from users. Provide intuitive user interfaces for rating movies, providing feedback, and viewing personalized recommendations to enhance user engagement and satisfaction.

• Performance Optimization and Scalability: Optimize the performance of the recommendation system to ensure fast response times and efficient processing of user requests, even with a large number of users and movie ratings. Employ techniques such as caching, database optimization, and distributed computing to enhance scalability and handle increased user load.

• Evaluation and Validation: Evaluate the effectiveness and accuracy of the recommendation system through rigorous testing and validation using real-world movie datasets. Use metrics such as precision, recall, and user satisfaction scores to assess the quality of recommendations and fine-tune the system parameters for optimal performance.

3.2 Scope

The scope of the Movie Recommendation System project using JAVA EE encompasses various aspects related to the development, implementation, and deployment of a robust and user-friendly web application for recommending movies based on similarity scores derived from user ratings. The project's scope includes, but is not limited to, the following key components:

• User Interface Development: Design and develop an intuitive and interactive user interface for the web application, allowing users to browse movies, rate them, and receive personalized recommendations based on their preferences.

• Backend Development: Implement the backend logic and functionality of the recommendation system using Java EE technologies such as Servlets, JSP, and EJB. Develop algorithms for calculating similarity scores between users and movies based on their ratings.

• Database Management: Set up and manage a database to store movie data, user ratings, and other relevant information. Utilize relational database management systems (RDBMS) such as MySQL or PostgreSQL to efficiently store and retrieve data.

• Performance Optimization: Optimize the performance of the recommendation system to ensure fast response times and efficient processing of user requests. Employ techniques such as caching, indexing, and database optimization to enhance system performance and scalability.

• Testing and Validation:Conduct thorough testing and validation of the recommendation system to ensure accuracy, reliability, and usability. Test the system with real-world movie datasets and evaluate its performance using metrics such as precision, recall, and user satisfaction scores.

• Documentation and Deployment: Document the design, implementation, and deployment process of the web application comprehensively. Provide clear instructions for setting up, configuring, and deploying the application in different environments. Document the system architecture, data flow, and algorithms used in the recommendation system.

• Future Enhancements: Identify potential areas for future enhancements and improvements to the recommendation system. Consider features such as social integration, content-based filtering, and machine learning techniques for further enhancing the accuracy and relevance of movie recommendations.

CHAPTER 4

PROBLEM ANALYSIS & REQUIREMENT SPECIFICATION

4.1 Problem Analysis

The development of a Movie Recommendation System using JAVA EE involves a thorough analysis of various challenges and considerations associated with building an effective and user-friendly recommendation system. The following key aspects need to be analyzed in detail:

4.1.1 Data Acquisition and Preprocessing

Gathering movie data from reliable sources and preprocessing it to extract relevant information such as movie titles, genres, and ratings. Handling missing or incomplete data and ensuring data consistency and quality are crucial steps in the preprocessing phase.

4.1.2 User Ratings Collection

Designing mechanisms for collecting movie ratings from users in an efficient and user-friendly manner. Providing intuitive interfaces for users to rate movies and submit feedback is essential for encouraging user engagement and participation.

4.1.3 Algorithm Selection and Implementation

Choosing appropriate collaborative filtering algorithms, such as user-based or item-based collaborative filtering, based on the nature of the movie recommendation task. Implementing these algorithms effectively to calculate similarity scores between users and movies and generate personalized recommendations.

4.1.4 Scalability and Performance

Ensuring that the recommendation system is capable of handling a large volume of users and movie ratings while maintaining optimal performance. Employing techniques such as caching, indexing, and database optimization to improve system scalability and response times.

4.1.5 User Experience and Interface Design

Designing an intuitive and visually appealing user interface that allows users to easily browse movies, view recommendations, and provide ratings and feedback. Ensuring seamless navigation and responsiveness across different devices and screen sizes is essential for enhancing the user experience.

4.1.6 Privacy and Security

Implementing measures to protect user privacy and ensure the security of sensitive data such as user profiles and movie ratings. Employing encryption techniques, access controls, and user authentication mechanisms to safeguard user information from unauthorized access and misuse.

4.1.7 Evaluation and Validation

Developing methods for evaluating the effectiveness and accuracy of the recommendation system. Using metrics such as precision, recall, and user satisfaction scores to assess the quality of recommendations and identify areas for improvement.

4.1.8 Integration with JAVA EE Framework

Leveraging the features and components of the JAVA EE framework, such as Servlets, JSP, and EJB, to build a scalable and efficient web application. Integrating the recommendation system seamlessly into the JAVA EE architecture to ensure compatibility and interoperability with other components.

4.2 System Requirement Specification

The System Requirement Specification (SRS) is a comprehensive document that outlines the requirements for a software system. In this case of a food delivery app for a brand, the SRS defines the functional and non-functional requirements of the app, as well as the constraints and assumptions that need to be taken into consideration during the development process.

The following is a detailed System Requirement Specification for this project:

4.2.1 Functional Requirements

A) User Registration and Authentication

• Users should be able to register for an account and authenticate themselves securely.

• Registered users should have access to personalized recommendations based on their ratings.

B) Movie Rating and Feedback

• Users should be able to rate movies on a scale or provide feedback on their viewing experience.

• The system should store and update user ratings and feedback for future recommendations.

C) Recommendation Generation

• The system should generate movie recommendations for users based on similarity scores with other users.

• Recommendations should be personalized and reflect user preferences and viewing history.

D) User Interface

• The web application should have an intuitive and user-friendly interface for browsing movies, viewing recommendations, and providing ratings.

• The interface should be responsive and accessible across different devices and screen sizes.

E) Administration Panel

• Admins should have access to an administration panel for managing user accounts, movie data, and system settings.

• Admins should be able to view reports and analytics related to user activity and recommendation performance

4.2.2 Non - Functional Requirements

A) Performance

• The system should be capable of handling a large number of users and movie ratings while maintaining optimal performance.

• Response times for generating recommendations and loading movie data should be within acceptable limits.

B) Scalability

• The system architecture should be scalable to accommodate growth in user base and movie database size.

• Scalability should be achieved through techniques such as load balancing, caching, and horizontal scaling.

C) Security

• User data, including personal information and movie ratings, should be stored securely and protected from unauthorized access.

• The system should implement encryption, access controls, and user authentication mechanisms to ensure data security.

D) Reliability

• The system should be reliable and available for use without frequent interruptions or downtime.

• Measures should be in place to handle system failures gracefully and ensure data integrity.

E) Compatibility

• The web application should be compatible with popular web browsers such as Chrome, Firefox, and Safari.

• Compatibility should be maintained across different operating systems and devices.

F) Documentation

• Comprehensive documentation should be provided for installation, configuration, and usage of the system.

• Documentation should include user guides, administrator manuals, and developer documentation.

4.2.3 System Constraints

• The system should be developed using JAVA EE technologies such as Servlets, JSP, and EJB. Database management should be implemented using relational database management systems (RDBMS) such as MySQL or PostgreSQL

• The web application should be hosted on a reliable and scalable hosting environment capable of supporting JAVA EE applications.

• The hosting environment should provide adequate resources for handling user traffic and data storage requirements

• The system should comply with relevant data privacy regulations and standards, such as GDPR (General Data Protection Regulation)

CHAPTER 5

DETAILED DESIGN

5.1 Data Flow Diagram

5.1.1 Introduction to DFD

A Data Flow Diagram (DFD) is a visual representation of the flow of data within a system. It shows how data is input, processed, stored, and outputted in a system. DFDs are an important tool for systems analysis and design because they help to model the system's data flow and identify its components.

DFDs consist of several components, including:

• Processes: Processes are represented by circles and show the operations that are performed on data within the system. A process can be an activity, transformation, or calculation that takes place within the system.

• Data Stores: Data stores are represented by rectangles with two parallel lines on the left side and show where data is stored in the system. Data stores can be databases, files, or any other storage medium used to hold data.

• Data Flows: Data flows are represented by arrows and show the movement of data between processes, data stores, and external entities. They represent the direction in which data moves through the system.

• External Entities: External entities are represented by rectangles with one or more lines extending from them and show the sources or destinations of data outside the system. External entities can be users, systems, or organizations that interact with the system.

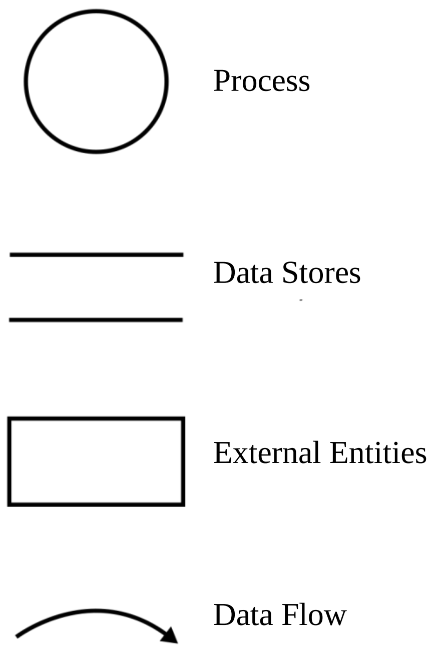


Fig 5.1 DFD Notations

DFDs are typically created in three levels, with each level providing more detail than the previous one. The levels are:

• Level 0 DFD: This is the highest-level DFD, which provides an overview of the entire system. It shows the main processes and data stores and how they are interconnected.

• Level 1 DFD: This level breaks down the processes in the level 0 DFD into sub-processes and provides more detail about the system. It shows how the sub-processes interact with each other and the data stores.

• Level 2 DFD: This level provides even more detail about the system and breaks down the sub-processes in the level 1 DFD into further detail. It shows how the data flows between the processes and data stores.

5.1.2 Level 0 DFD

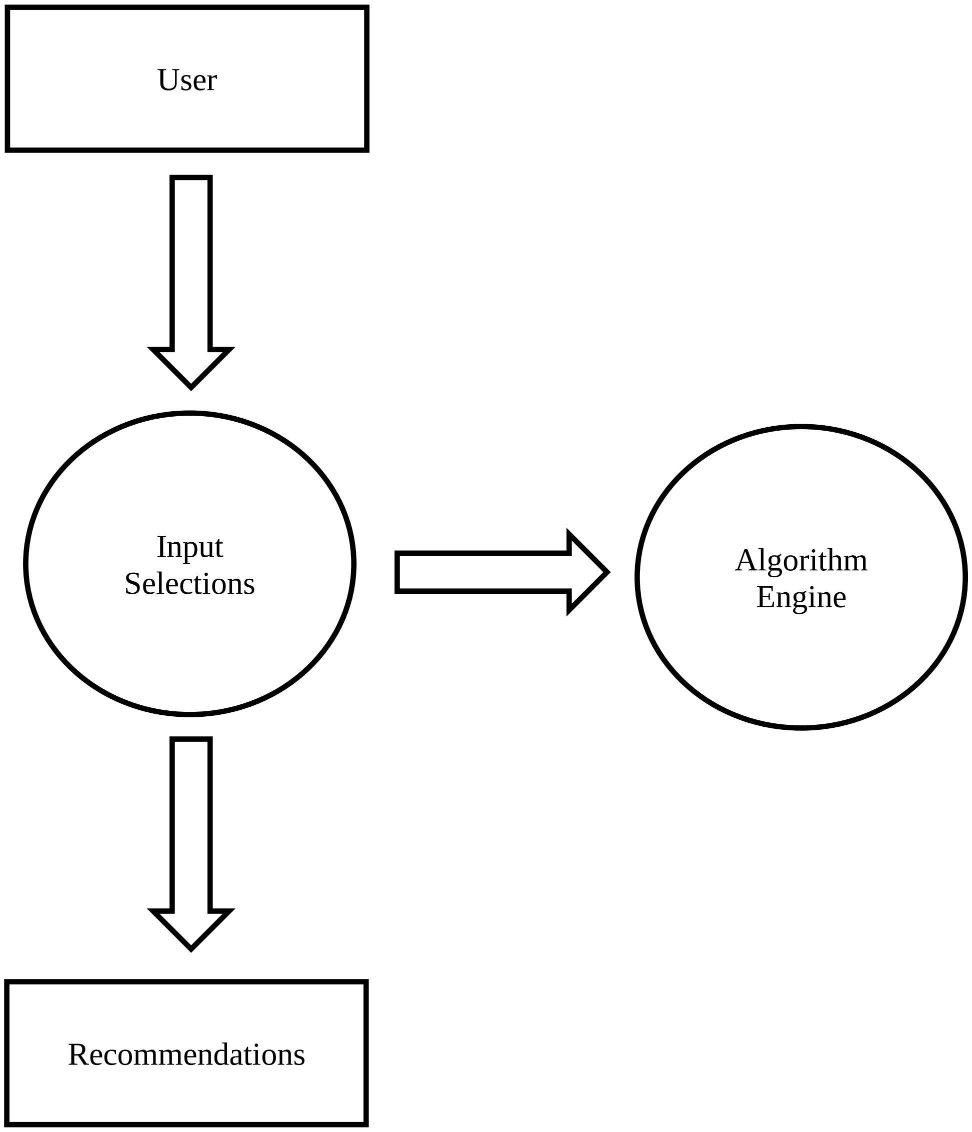
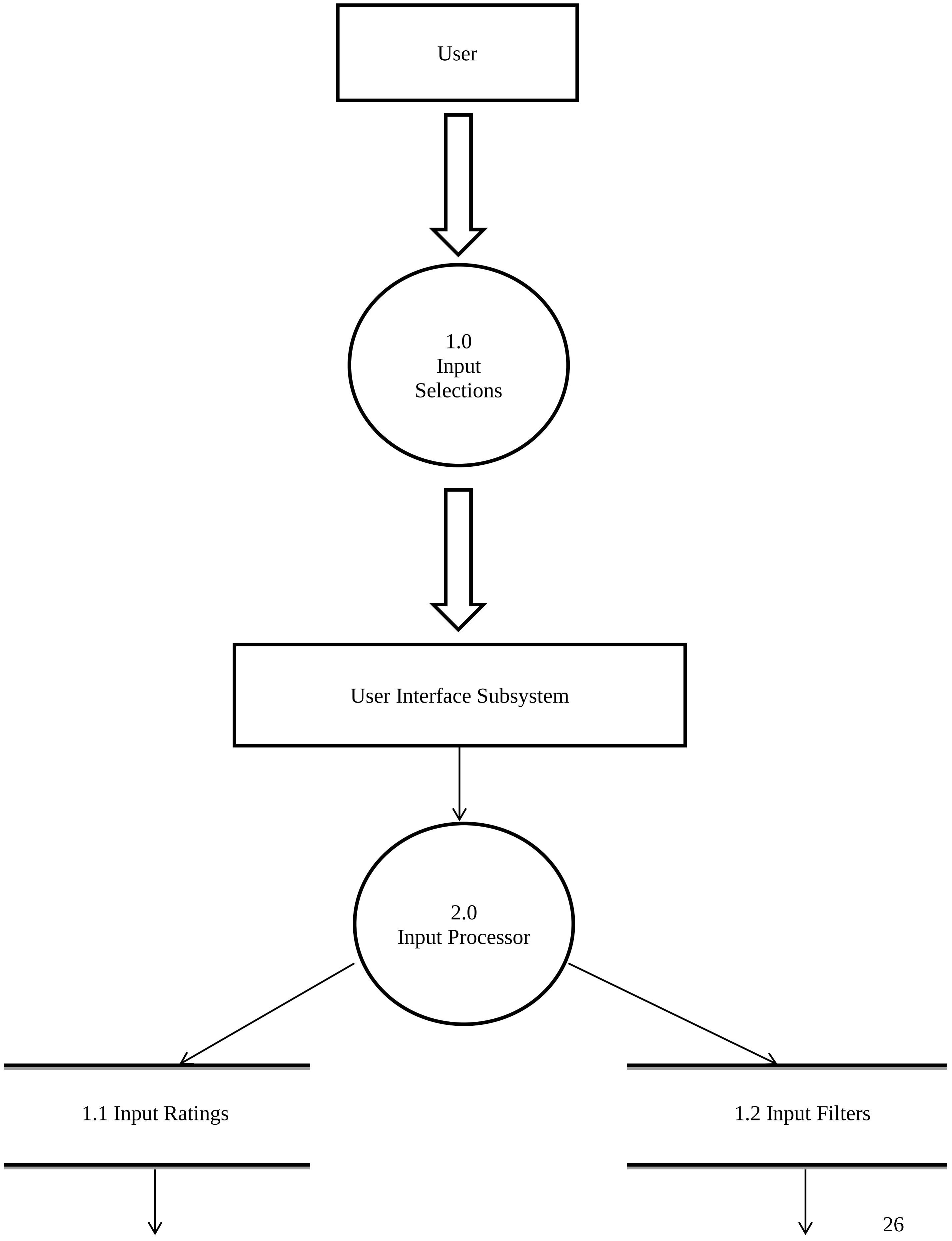


Fig 5.2 Level 0 DFD

5.1.3 Level 1 DFD



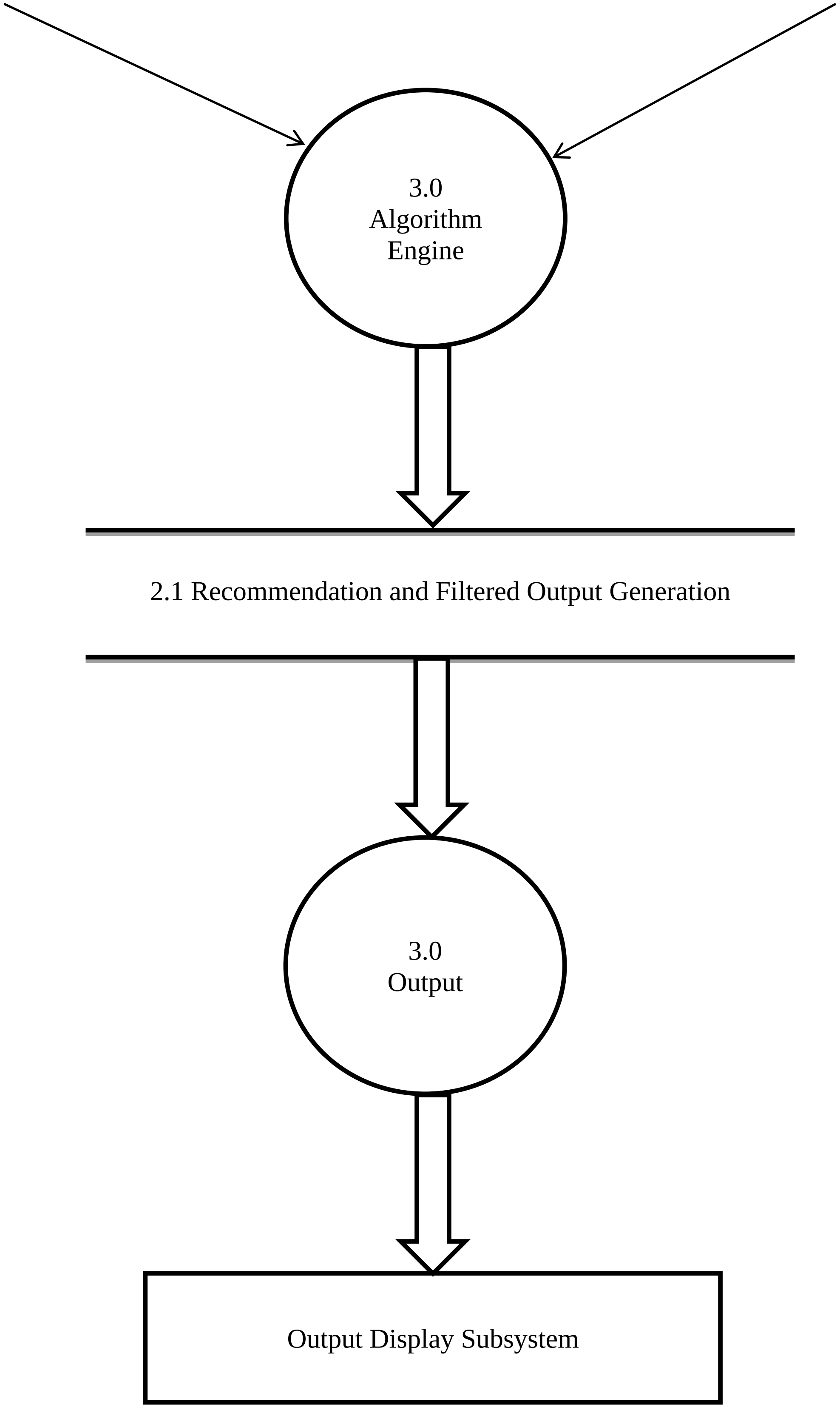
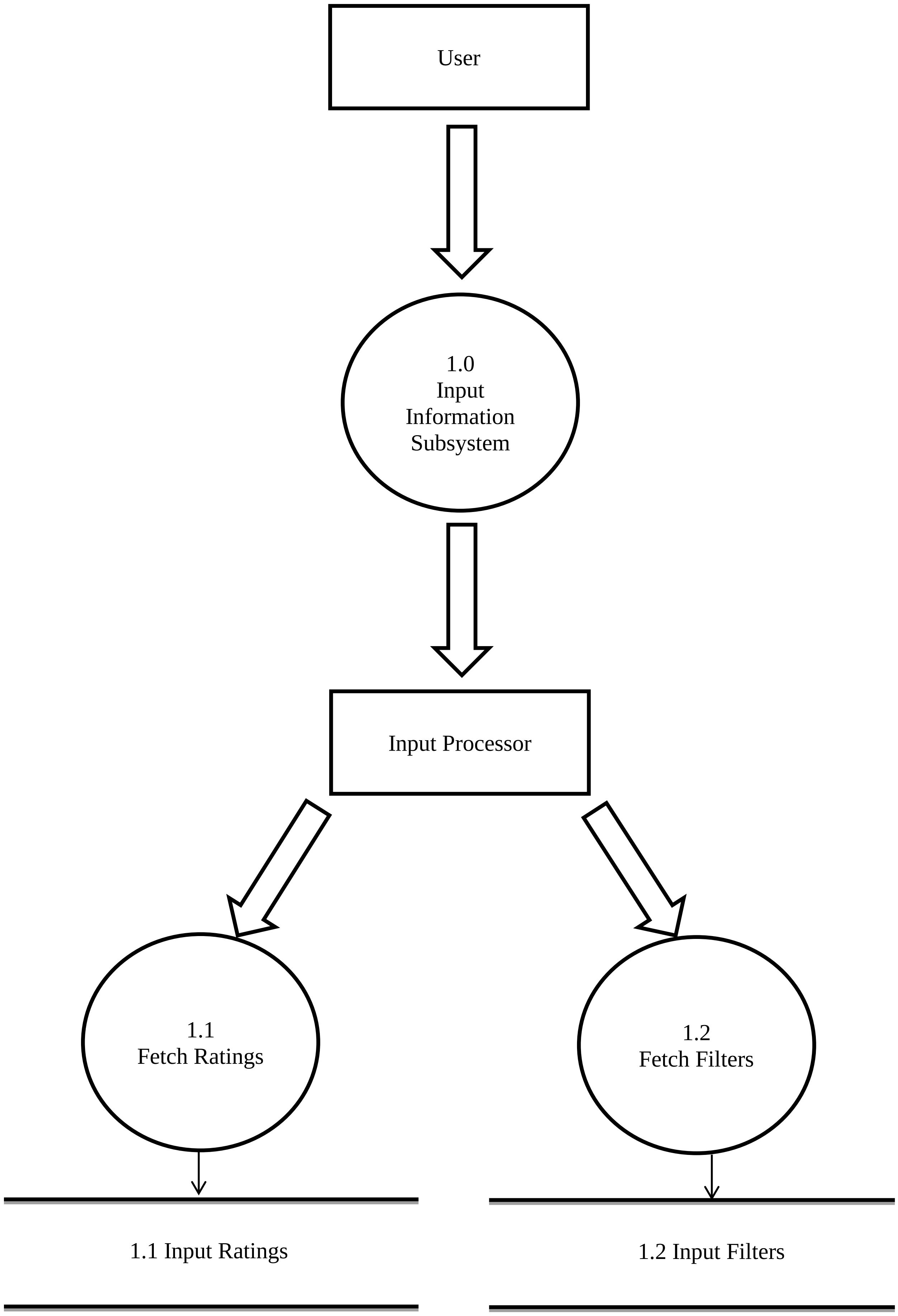
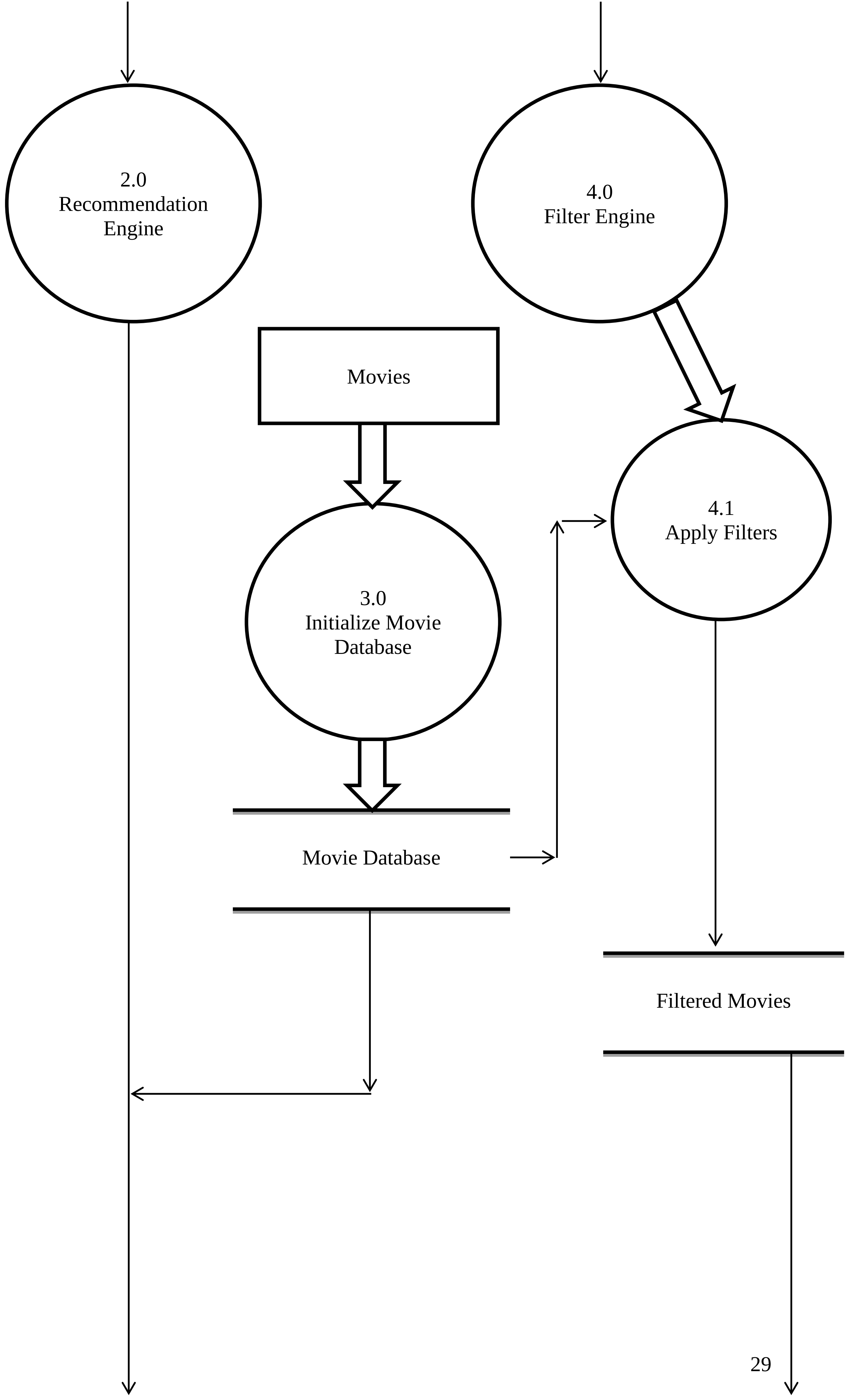
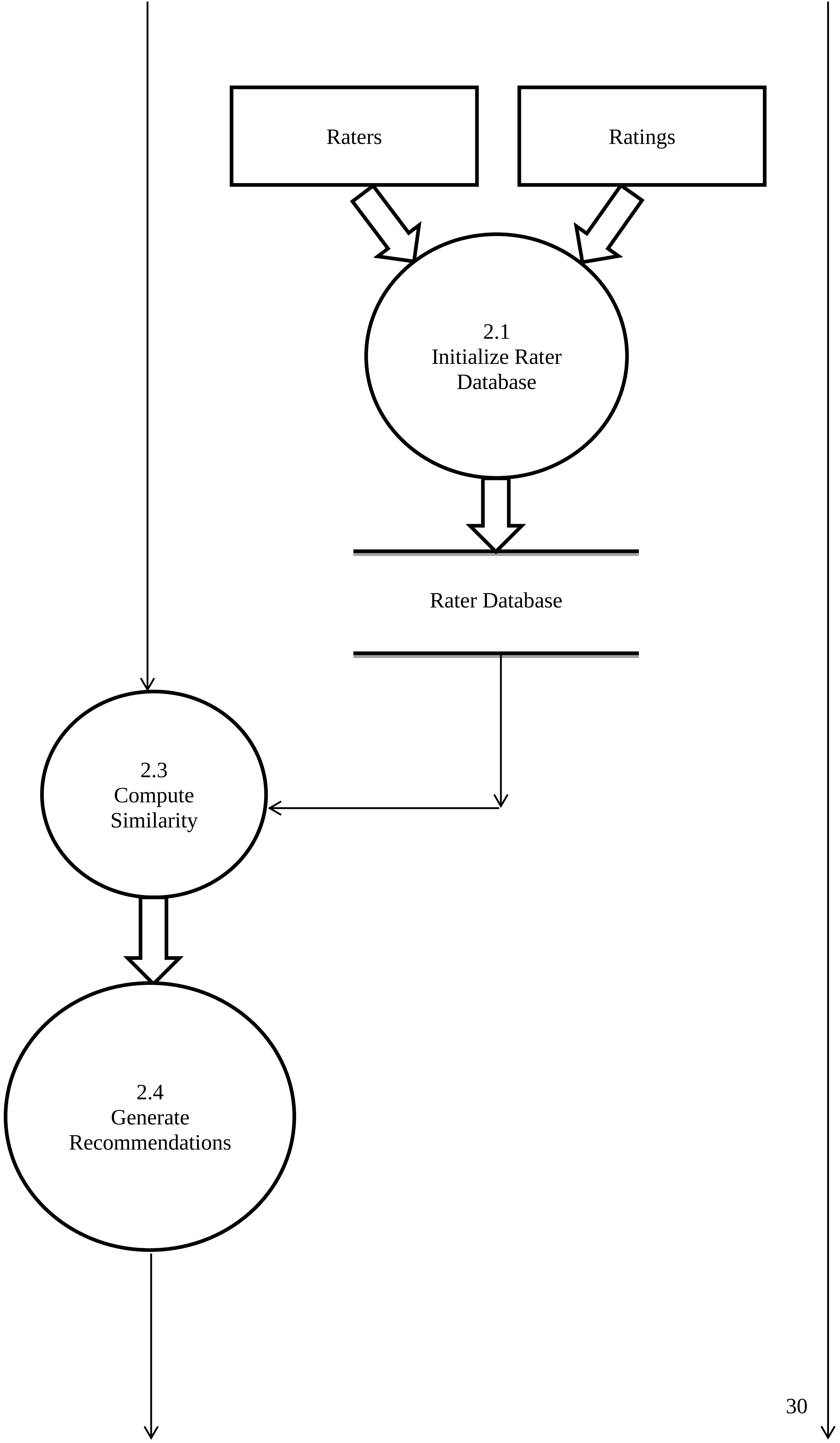


Fig 5.3 Level 1 DFD

5.1.4 Level 2 DFD




2.0
Recommendation
Engine
4.0
Filter Engine
Movies
4.1
Apply Filters
3.0
Initialize Movie
Database
Movie Database
Filtered Movies


Raters
Ratings
2.1
Initialize Rater Database
Rater Database
2.3
Compute Similarity
2.4
Generate Recommendations

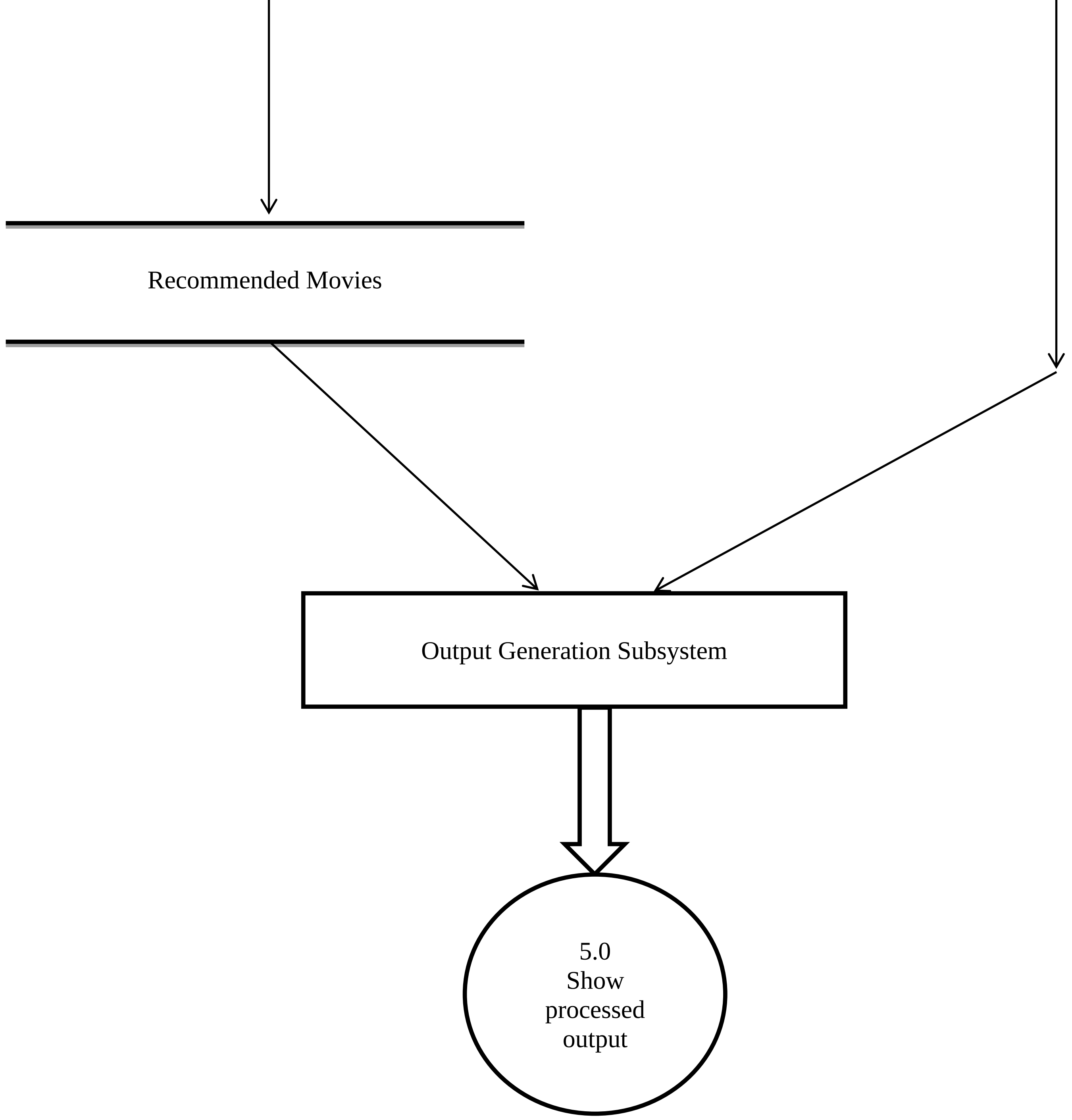


Fig 5.4 Level 2 DFD

5.2 Entity Relationship Diagram

5.2.1 Introduction to ERD

An Entity-Relationship Diagram (ERD) is a graphical representation of the relationships between entities in a system. It is used to model the data elements and their relationships within a database system. An ERD consists of entities, attributes, and relationships.

• Entities: An entity is a real-world object or concept that is represented by a rectangle in the ERD. Examples of entities include customers, products, orders, and employees.

• Attributes: An attribute is a characteristic or property of an entity. Examples of attributes include the customer's name, product price, order date, and employee ID. Attributes are represented by ovals connected to the entity rectangle.

• Relationships: Relationships describe the associations between entities in a system. A relationship can be one-to-one, one-to-many, or many-to-many. Relationships are represented by diamonds connected to the entity rectangles. The lines connecting the diamonds to the entity rectangles show the cardinality of the relationship.

ERDs are typically created in three levels, with each level providing more detail than the previous one. The levels are:

• Conceptual ERD: This is the highest-level ERD, which provides an overview of the entire system. It shows the main entities and their relationships.

• Logical ERD: This level breaks down the entities in the conceptual ERD into more detail and provides more information about their attributes and relationships.

• Physical ERD: This level provides the implementation details for the database system, such as data types, primary keys, and foreign keys.

5.2.2 ERD Diagram

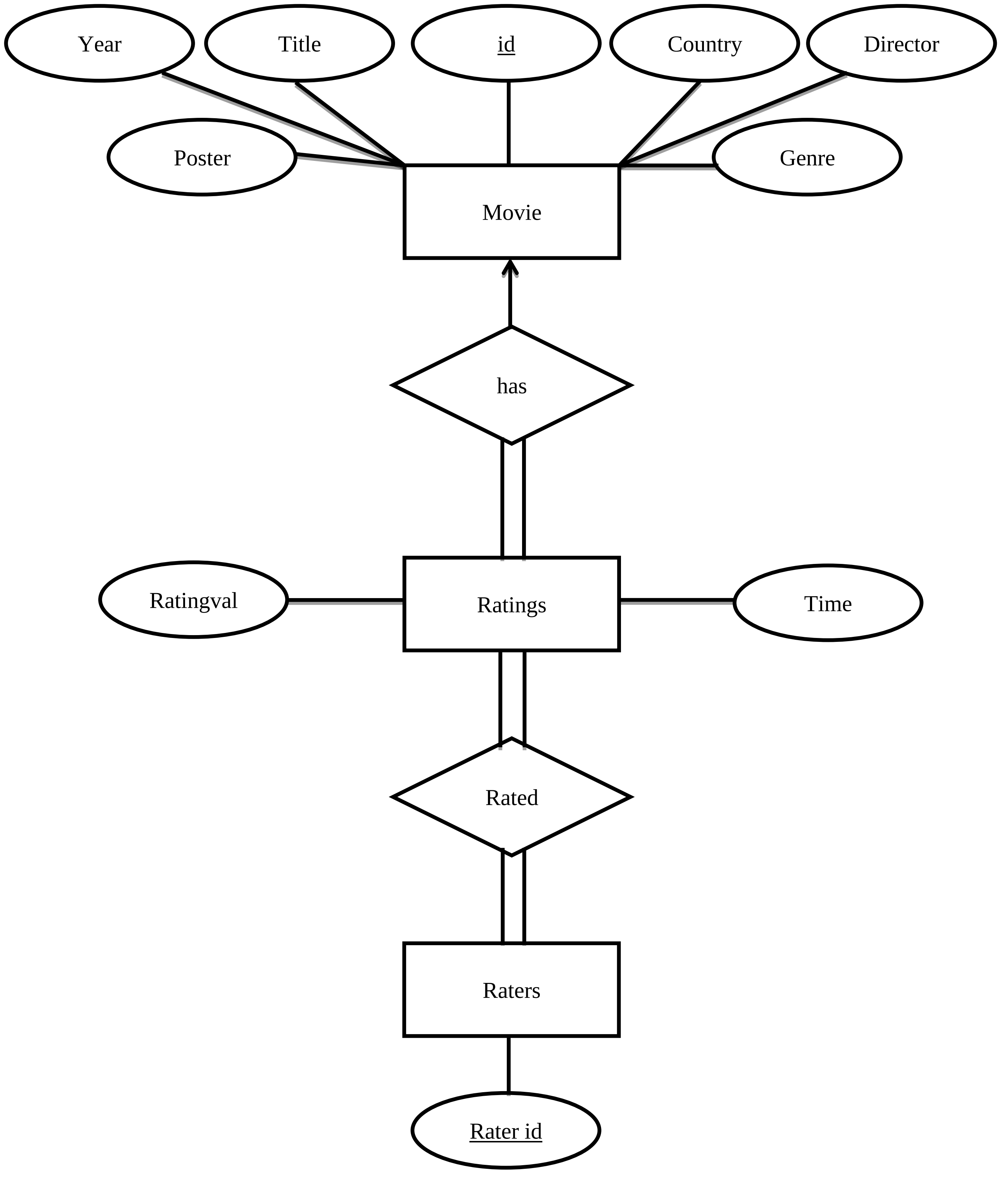


Fig 5.5 ER Diagram

5.3 Waterfall Model

The Waterfall model is a linear sequential approach to software development that follows a sequential, phased approach in which the output of each phase becomes the input for the next phase. This model was first introduced by Winston Royce in 1970 and has since been widely used in software development.

The Waterfall model consists of the following phases:

- Requirements gathering and analysis: In this phase, the requirements are gathered and analyzed to understand what needs to be done. This phase is crucial as it sets the foundation for the entire project.

- Design: In this phase, the system design is created based on the requirements gathered in the previous phase. This includes creating the architecture of the system and specifying how different components of the system will interact with each other.

- Implementation: In this phase, the actual coding of the system is done based on the design created in the previous phase. This phase involves writing code, testing the code, and fixing any issues that arise during testing.

- Testing: In this phase, the system is tested to ensure that it meets the requirements that were specified in the first phase. This phase involves testing the system at different levels, including unit testing, integration testing, system testing, and acceptance testing.

- Deployment: In this phase, the system is deployed to the production environment. This phase involves installing the system, configuring it, and ensuring that it is running as expected.

- Maintenance: In this phase, the system is maintained and updated to ensure that it continues to meet the requirements and operates as expected. This involves fixing bugs, adding new features, and addressing any issues that arise during operation.

CHAPTER 6

HARDWARE & SOFTWARE ENVIRONMENT

6.1 Hardware Requirements

• Processor: Intel Core i5 or equivalent

• RAM: 8GB or higher

• Storage: 256GB SSD or higher

• Network Connectivity: Ethernet or Wi-Fi for internet access

• Display: Minimum resolution of 1366x768 pixels

• Input Devices: Keyboard and Mouse or Touchpad

6.2 Software Requirements

• Operating System: Windows 10, macOS, or Linux (Ubuntu, CentOS, etc.)

• Web Browser: Latest versions of Google Chrome, Mozilla Firefox, Safari, or Microsoft Edge for accessing the web application

• Java Development Kit (JDK): JDK 8 or later for compiling and running Java code

• Integrated Development Environment (IDE): Eclipse, IntelliJ IDEA, or NetBeans for Java EE development

• Apache Tomcat: Tomcat 8 or later as the Servlet container for deploying the web application

• Version Control System: Git for managing source code and collaborating with team members

• Dependency Management: Apache Maven for managing project dependencies and building the application

• Text Editor: Optional for editing configuration files, scripts, and documentation (e.g., Sublime Text, Visual Studio Code)

6.3 Java EE Technologies

• Servlets: Java Servlet technology for handling HTTP requests and generating dynamic web content

• JavaServer Pages (JSP): JSP technology for embedding Java code in HTML pages and creating dynamic user interfaces

• Java EE Application Server: Apache Tomcat as the Java EE application server for deploying and running the web application

6.4 Development Environment

• Development Language: Java for backend logic, HTML, CSS, and JavaScript for frontend user interface

• Frameworks and Libraries: jQuery, Bootstrap, or other frontend frameworks for enhancing user experience and styling

• Testing Framework: JUnit or TestNG for unit testing Java code, Selenium for automated web browser testing

• Continuous Integration: Jenkins, Travis CI, or CircleCI for automating build, test, and deployment processes

• Project Documentation: Apache Maven or Gradle for generating project documentation, Javadoc for documenting Java code

• Collaboration Tools: GitHub, GitLab, or Bitbucket for version control, issue tracking, and code review (GitHub Repository [8])

CHAPTER 7

SNAPSHOTS

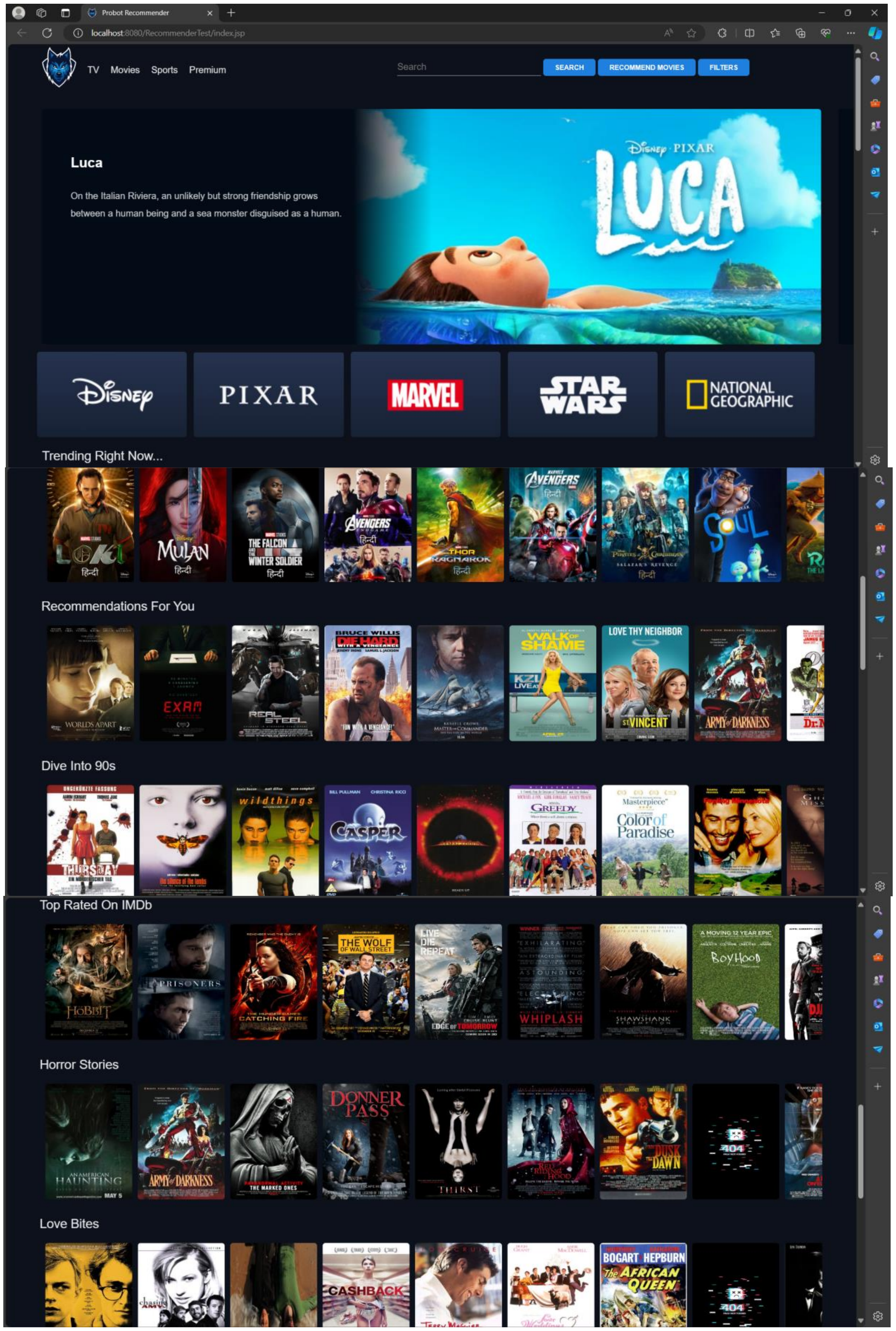


Fig. 7.1 Home Page

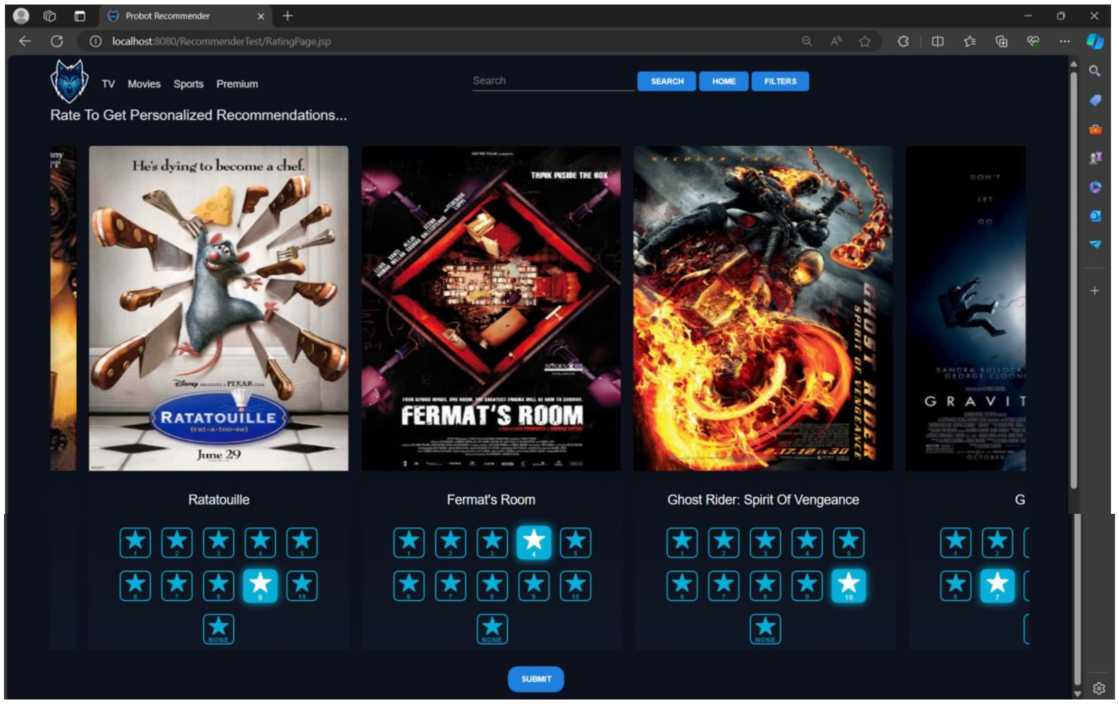

Probot Recommender
localhost:8080/RecommenderTest/RatingPage.jsp
TV Movies Sports Premium
Search
SEARCH
HOME
FILTERS
Rate To Get Personalized Recommendations...
Ratatouille
Fermat's Room
Ghost Rider: Spirit Of Vengeance
NONE
SUBMIT

Fig. 7.2 Recommendation Page

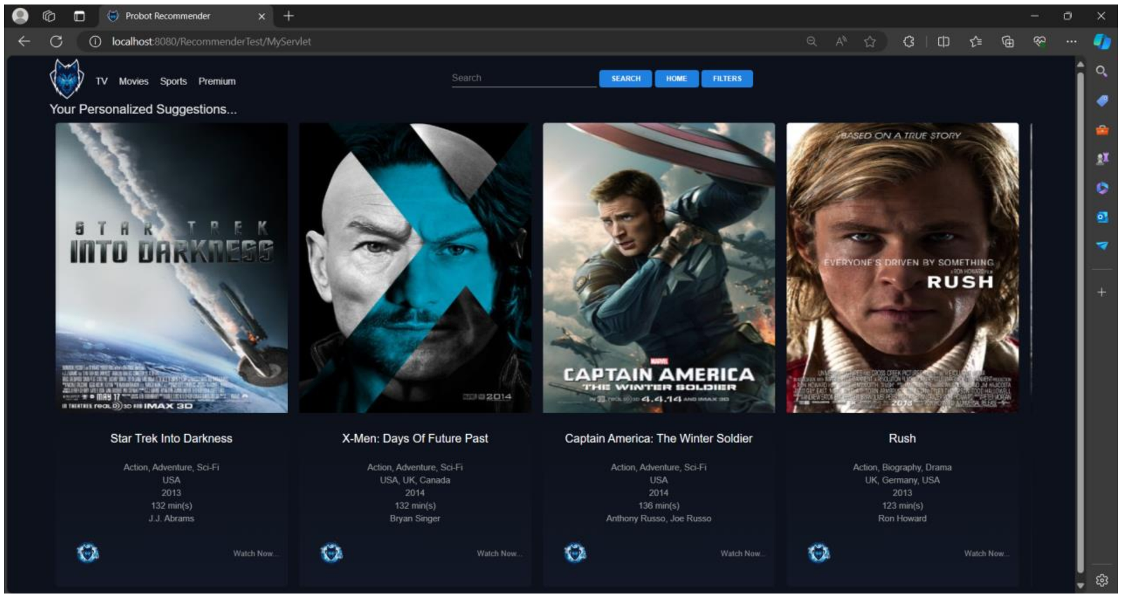


Fig. 7.3 Recommendation Output

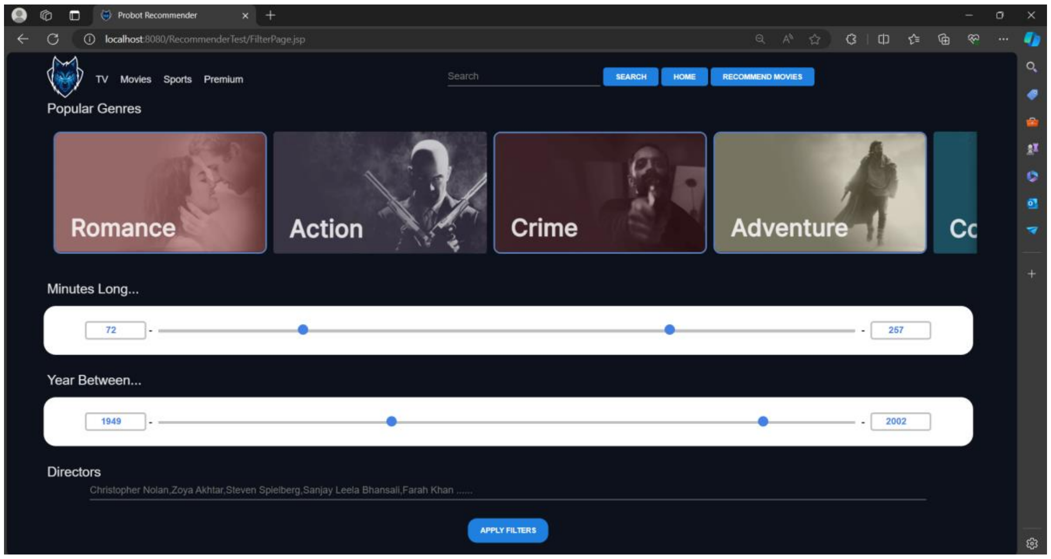
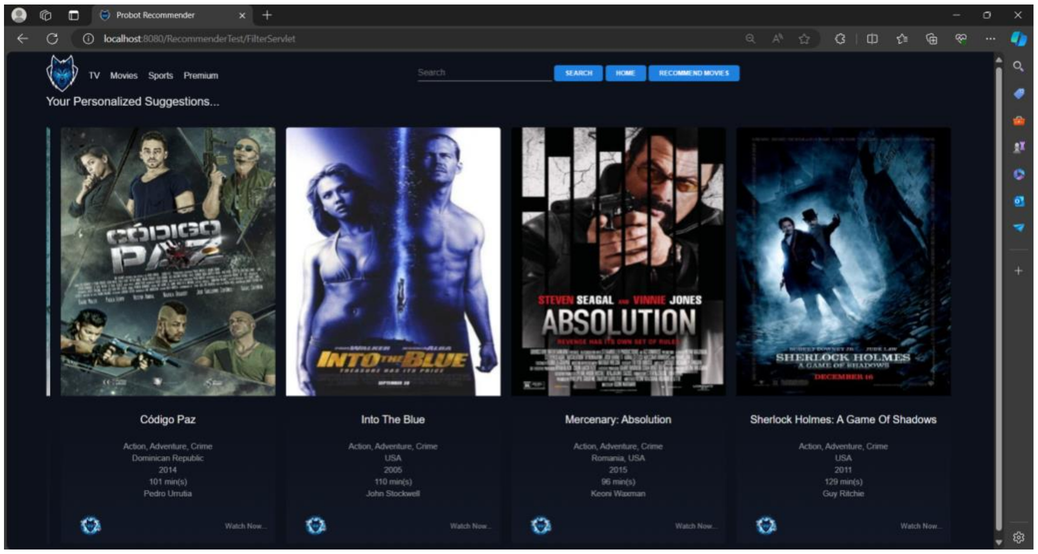


Fig. 7.4 Filter Page



Código Paz
Action, Adventure, Crime
Dominican Republic
2014
101 min(s)
Pedro Urrutia

Into The Blue
Action, Adventure, Crime
USA
2005
110 min(s)
John Stockwell

Mercenary: Absolution
Action, Adventure, Crime
Romania, USA
2015
96 min(s)
Keoni Waxman

Sherlock Holmes: A Game Of Shadows
Action, Adventure, Crime
USA
2011
129 min(s)
Guy Ritchie

Fig. 7.5 Filter Output

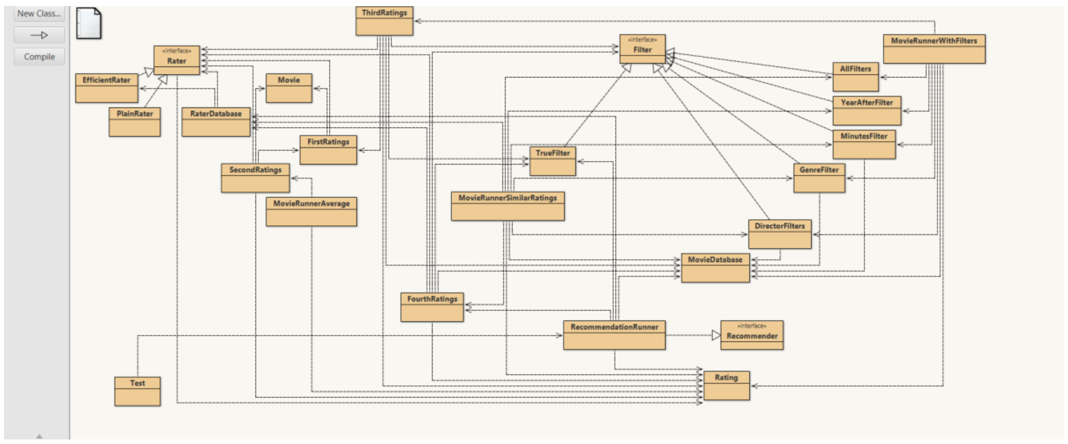


Fig. 7.6 Java Code Flow

CHAPTER 8

CODING

Java Runner:

```
public class RecommendationRunner implements Recommender{
   public ArrayList<String> getItemsToRate(){
      ArrayList<String> chosenMovies = new ArrayList<String>();
      ArrayList<String> movies = MovieDatabase.filterBy(new TrueFilter());
      for(int i=0; chosenMovies.size()<20; i++){
         Random seed = new Random();
         int randomInt = seed.nextInt(movies.size());
         String movieID = movies.get(randomInt);
         if(!chosenMovies.contains(movieID)){
            chosenMovies.add(movieID);
         }  }
      return chosenMovies;  }
   public ArrayList<String> getActionMovies(){
      ArrayList<String> actionMovies = new ArrayList<String>();
      ArrayList<String> movies = MovieDatabase.filterBy(new GenreFilter("Action"));
      for(int i=0; actionMovies.size()<20; i++){
         Random seed = new Random();
         int randomInt = seed.nextInt(movies.size());
         String movieID = movies.get(randomInt);
         if(!actionMovies.contains(movieID)){
            actionMovies.add(movieID);
         }  }
      return actionMovies; }
```

```
public ArrayList<String> getRomanceMovies(){
    ArrayList<String> romanceMovies = new ArrayList<String>();
    ArrayList<String> movies = MovieDatabase.filterBy(new GenreFilter("Romance"));
    for(int i=0; romanceMovies.size()<20; i++){
        Random seed = new Random();
        int randomInt = seed.nextInt(movies.size());
        String movieID = movies.get(randomInt);
        if(!romanceMovies.contains(movieID)){
                romanceMovies.add(movieID);
        }
    }
    return romanceMovies;
}
public ArrayList<String> getDramaMovies(){
    ArrayList<String> dramaMovies = new ArrayList<String>();
    ArrayList<String> movies = MovieDatabase.filterBy(new GenreFilter("Drama"));
    for(int i=0; dramaMovies.size()<20; i++){
        Random seed = new Random();
        int randomInt = seed.nextInt(movies.size());
        String movieID = movies.get(randomInt);
        if(!dramaMovies.contains(movieID)){
                dramaMovies.add(movieID);
        }
    }
    return dramaMovies;
}
```

```
public ArrayList<String> getHorrorMovies(){
    ArrayList<String> horrorMovies = new ArrayList<String>();
    ArrayList<String> movies = MovieDatabase.filterBy(new GenreFilter("Horror"));
    for(int i=0; horrorMovies.size()<20; i++){
        Random seed = new Random();
        int randomInt = seed.nextInt(movies.size());
        String movieID = movies.get(randomInt);
        if(!horrorMovies.contains(movieID)){
                    horrorMovies.add(movieID);
        }
    }
    return horrorMovies;
}
public ArrayList<String> getDocumentaryMovies(){
    ArrayList<String> documentaryMovies = new ArrayList<String>();
    ArrayList<String> movies = MovieDatabase.filterBy(new GenreFilter("Documentary"));
    for(int i=0; documentaryMovies.size()<20; i++){
        Random seed = new Random();
        int randomInt = seed.nextInt(movies.size());
        String movieID = movies.get(randomInt);
        if(!documentaryMovies.contains(movieID)){
                    documentaryMovies.add(movieID);
        }
    }
    return documentaryMovies;
}
```

```
public ArrayList<String> get90sMovies(){
        ArrayList<String> ninetiesMovies = new ArrayList<String>();
    ArrayList<String> movies = MovieDatabase.filterBy(new YearBetweenFilter(1990,1999));
    for(int i=0; ninetiesMovies.size()<20; i++){
        Random seed = new Random();
        int randomInt = seed.nextInt(movies.size());
        String movieID = movies.get(randomInt);
        if(!ninetiesMovies.contains(movieID)){
                ninetiesMovies.add(movieID);
        }}
    return ninetiesMovies;}
public ArrayList<String> getTopMovies(){
    SecondRatings SRobj = new SecondRatings("data/ratedmoviesfull.csv", "ratings.csv");
    int minimalRaters = 20;
    ArrayList<Rating> avg = SRobj.getAverageRatings(minimalRaters);
    ArrayList<String> topMovies = new ArrayList<String>();
    for(Rating r : avg){
        if(r.getValue()>8.0){
            topMovies.add(r.getItem());
        }  }
    return topMovies;  }

public ArrayList<Rating> finalSimilarMoviesRecommended(String webRaterID){
        FourthRatings fr = new FourthRatings();
    ArrayList<Rating> recommendedMovies = fr.getSimilarRatings(webRaterID,20,5);
    return recommendedMovies;
}
```

```
    public ArrayList<String> getFilteredMovies(String[] genres, String minMinutes, String
maxMinutes, String minYear, String maxYear, String directors){
        ArrayList<String> filteredMovies = new ArrayList();
        int filterStatus = 0;
        AllFilters af = new AllFilters();
        if(genres != null) {
            for(String s : genres) {
                af.addFilter(new GenreFilter(s)); }
            filterStatus = 1;}
        if(!(minMinutes.equals("0") && maxMinutes.equals("350"))) {
            af.addFilter(new
MinutesFilter(Integer.parseInt(minMinutes),Integer.parseInt(maxMinutes)));
            filterStatus = 1;
        }
        if(!(minYear.equals("1916") && maxYear.equals("2015"))) {
            af.addFilter(new
YearBetweenFilter(Integer.parseInt(minYear),Integer.parseInt(maxYear)));
            filterStatus = 1;
        }
        if(!directors.equals("")) {
            af.addFilter(new DirectorFilters(directors));
            filterStatus = 1;
        }
        if(filterStatus == 0) { //no filter is applied then showing all the movies
            filteredMovies = MovieDatabase.filterBy(new TrueFilter());
            return filteredMovies;
        }
        filteredMovies  = MovieDatabase.filterBy(af);
        return filteredMovies; }
```

Rating Calculator:

```
public class FourthRatings {
    private double getAverageByID(String id, int minimalRaters) {
        ArrayList <Rater> myRaters = new ArrayList<Rater>();
        myRaters = RaterDatabase.getRaters();
        double ratings = 0.0;
        int count = 0;
        for(Rater r : myRaters) {
            if(r.getRating(id)!=-1){
                ratings+=r.getRating(id);
                count++;
            }
        }
        if(count<minimalRaters){
            return 0.0;
        }
        else{
            return ratings/count;
        }
    }

    public ArrayList<Rating> getAverageRatings(int minimalRaters) {
        ArrayList<String> movies = MovieDatabase.filterBy(new TrueFilter());
        ArrayList<Rating> avg = new ArrayList();
        for(String id : movies){
            double avgRating = getAverageByID(id,minimalRaters);
            if(avgRating == 0.0) {
```

```
            continue;
        }
        Rating r = new Rating(id,avgRating);
        avg.add(r);
    }
    return avg;
}

public ArrayList<Rating> getAverageRatingsByFilter(int minimalRaters,Filter filterCriteria){
    ArrayList<String> movies = MovieDatabase.filterBy(filterCriteria);
    ArrayList<Rating> avg = new ArrayList();
    for(String id : movies){
        double avgRating = getAverageByID(id,minimalRaters);
        if(avgRating == 0.0) {
            continue;
        }
        Rating r = new Rating(id,avgRating);
        avg.add(r);
    }
    return avg;
}
private double dotProduct(Rater me, Rater r) {
    //MovieDatabase.initialize("ratedmoviesfull.csv");
    ArrayList<String> movies = MovieDatabase.filterBy(new TrueFilter());
    double dotProduct = 0.0;
    for(String id : movies){
        if(me.hasRating(id) && r.hasRating(id)){
                dotProduct+=(me.getRating(id)-5)*(r.getRating(id)-5);
            }
        }
        return dotProduct;
    }
    public ArrayList<Rating> getSimilarities(String id){
        ArrayList<Rating> similarRaters = new ArrayList<Rating>();
        //RaterDatabase.initialize("ratings.csv");
        Rater me = RaterDatabase.getRater(id);
        ArrayList<Rater> raters = RaterDatabase.getRaters();
        for(Rater r : raters){
            if(r.getID().equals(id)){
                continue;
            }
            double dotProductReturn = dotProduct(me,r);
            if(dotProductReturn < 0){
                continue;
            }
            Rating ratingObj = new Rating(r.getID(),dotProductReturn);
            similarRaters.add(ratingObj);
        }
        Collections.sort(similarRaters,Collections.reverseOrder());
        return similarRaters;
    }
    public ArrayList<Rating> getSimilarRatings(String id, int numSimilarRaters, int
minimalRaters){
        ArrayList<Rating> similarRatingsOfMovie = new ArrayList<Rating>();
        ArrayList<Rating> similarRaters = getSimilarities(id);  //raterID And Similarity
        //Rater me = RaterDatabase.getRater(id);
        ArrayList<String> movies = MovieDatabase.filterBy(new TrueFilter());
        for(String movieID : movies){
            int count = 0;
            double weightedRatings = 0.0;
            for(int i=0; i<numSimilarRaters; i++){
                Rating r = similarRaters.get(i);
                String raterID = r.getItem();
                double similarity = r.getValue();
                Rater raterObj = RaterDatabase.getRater(raterID);
                if(raterObj.hasRating(movieID)){
                    double movieRating = raterObj.getRating(movieID);
                    weightedRatings+=(movieRating*similarity);
                    count++;
                }
            }
            if(count>=minimalRaters){
                 double weightedAverageRatings = weightedRatings/count;
                 Rating movieRatingObj = new Rating(movieID,weightedAverageRatings);
                 similarRatingsOfMovie.add(movieRatingObj);
            }
        }
        Collections.sort(similarRatingsOfMovie,Collections.reverseOrder());
        return similarRatingsOfMovie;
    }
    public ArrayList<Rating> getSimilarRatingsByFilter(String id, int numSimilarRaters, int
minimalRaters, Filter filterCriteria){
        ArrayList<Rating> similarRatingsOfMovie = new ArrayList<Rating>();
        ArrayList<Rating> similarRaters = getSimilarities(id);  //raterID And Similarity
        Rater me = RaterDatabase.getRater(id);
        ArrayList<String> movies = MovieDatabase.filterBy(filterCriteria);
        for(String movieID : movies){
            int count = 0;
            double weightedRatings = 0.0;
            for(int i=0; i<numSimilarRaters; i++){
                Rating r = similarRaters.get(i);
                String raterID = r.getItem();
                double similarity = r.getValue();
                Rater raterObj = RaterDatabase.getRater(raterID);
                if(raterObj.hasRating(movieID)){
                    double movieRating = raterObj.getRating(movieID);
                    weightedRatings+=(movieRating*similarity);
                    count++;
                }
            }
            if(count >= minimalRaters){
                 double weightedAverageRatings = weightedRatings/count;
                 Rating movieRatingObj = new Rating(movieID,weightedAverageRatings);
                 similarRatingsOfMovie.add(movieRatingObj);
            }
        }
        Collections.sort(similarRatingsOfMovie,Collections.reverseOrder());
        return similarRatingsOfMovie;
    }
}
```

CHAPTER 9

PROJECT LIMITATIONS & FUTURE SCOPE

9.1 Limitations

• Scalability: The system's performance may degrade with a large number of users and movies due to increased computational complexity in calculating similarity scores and generating recommendations.

• Cold Start Problem: New users may not receive accurate recommendations initially due to insufficient data for similarity calculation. Addressing this issue requires implementing alternative recommendation strategies for new users.

• Data Sparsity: Sparse user-item interaction data may lead to inaccurate recommendations, especially for niche or less popular movies. Implementing techniques to handle data sparsity is essential for improving recommendation quality.

• Cold Start for New Movies: Similar to the cold start problem for new users, new movies may not receive sufficient ratings initially, impacting the accuracy of recommendations. Incorporating strategies to promote new movie ratings is necessary to mitigate this limitation.

• Limited Recommendation Diversity: The system may prioritize recommending popular or mainstream movies, leading to a lack of diversity in recommendations. Implementing techniques to enhance recommendation diversity is crucial for catering to diverse user preferences.

9.2 Future Scope

• Integration of Content-Based Filtering: Incorporating content-based filtering techniques can enhance recommendation accuracy by considering the content features of movies and users' preferences.

• Advanced Recommendation Algorithms: Exploring advanced recommendation algorithms such as matrix factorization, neural networks, or hybrid models can improve recommendation accuracy and mitigate cold start issues.

• Enhanced User Interface: Improving the user interface with interactive features, personalized recommendations, and visualizations can enhance user engagement and satisfaction.

• Real-Time Recommendation Updates: Implementing real-time recommendation updates based on user interactions and feedback can provide more timely and relevant recommendations.

• Integration with Social Media and External APIs: Integrating with social media platforms and external APIs can enrich user profiles with additional data, leading to more accurate and diverse recommendations.

• Cross-Platform Compatibility: Extending the system's compatibility to multiple platforms such as mobile devices and smart TVs can increase its accessibility and user reach.

• Integration with Streaming Platforms: Partnering with streaming platforms to integrate recommendations directly into their services can enhance user convenience and engagement.

• Personalized Filtering Options: Providing users with personalized filtering options based on their preferences, viewing history, and demographic information can enhance their movie browsing experience.

CHAPTER 10

REFERENCES

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," Computer, vol. 42, no. 8, pp. 30-37, 2009.

[2] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in Proceedings of the 10th international conference on World Wide Web, 2001, pp. 285-295.

[3] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 6, pp. 734-749, 2005.

[4] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in Recommender systems handbook, Springer, Boston, MA, 2011, pp. 1-35.

[5] Java SE Documentation, "Java Platform, Standard Edition (Java SE) 8 Documentation," [Online]. Available: https://docs.oracle.com/javase/8/docs/.

[6] Duke University, "edu.duke," [Online]. Available: https://www.duke.edu/.

[7] Apache Software Foundation, "org.apache.commons.csv," [Online]. Available: https://commons.apache.org/proper/commons-csv/.

[8] A. Barodia, N. Raj, (2024, February 12). Movie Recommender[GitHub repository]. GitHub. Available: https://github.com/AdityaBarodia/MovieRecommender.