2/19/2024

Software Design Document

Movie Recommendation System

Undupitikankanamge Don Chamika Srimantha
H2O.AI

Table of Contents

Introduction

The Movie Recommendation System is a software application designed to assist users in discovering new movies tailored to their preferences. With the proliferation of digital streaming platforms and vast movie libraries, users often face the challenge of selecting content aligned with their tastes and interests. Traditional methods of movie discovery, such as browsing through genre categories or relying on popular recommendations, may not always yield satisfactory results. To address this issue, the Movie Recommendation System employs advanced machine learning algorithms and collaborative filtering techniques to generate personalized movie suggestions.

Background

The rise of online streaming platforms has transformed the way people consume media, offering a vast array of movies and TV shows at their fingertips. However, the abundance of choices can lead to decision paralysis, where users struggle to find content that resonates with their preferences. Movie recommendation systems have emerged as a solution to this problem, leveraging user data and machine learning models to deliver personalized recommendations.

Purpose

The purpose of the Movie Recommendation System is to enhance the movie-watching experience for users by providing them with relevant and personalized movie suggestions. By analyzing user behavior, viewing history, and movie metadata, the system aims to recommend movies that align with users' tastes and preferences. Whether users are looking for new releases, classic films, or niche genres, the recommendation system aims to cater to diverse interests and preferences.

Main Goal & Objectives

Goal

The main goal of the Movie Recommendation System is to enhance the movie-watching experience for users by providing personalized and relevant movie recommendations tailored to their preferences and interests.

Objectives

- Provide personalized movie recommendations based on user preferences.
- Enhance user engagement and satisfaction by delivering relevant content.
- Improve movie discovery and exploration by suggesting diverse and tailored recommendations.
- Optimize the recommendation process through efficient algorithms and data processing techniques.
- Facilitate informed decision-making for users by presenting relevant information about recommended movies.

Functional Requirements

Movie Search and Browse:

Users should be able to search for movies by title, genre, actor, director, or keyword.

The system should provide browsing options for exploring movies by genre, release year, popularity, and other criteria.

Movie Recommendation Generation:

The system should generate personalized movie recommendations for users based on their viewing history, ratings, and preferences.

Recommendation algorithms should take into account factors such as user similarity, movie popularity, genre relevance, and collaborative filtering techniques.

Recommendation Filters and Preferences:

Users should be able to set preferences and filters for refining recommendations based on criteria such as genre, release year, rating, and language.

Preference settings should be customizable and adjustable to accommodate changing user preferences.

Non-Functional Requirements

Performance:

Response Time: The system should respond to user requests within milliseconds to ensure a smooth and responsive user experience.

Scalability: The system should be scalable to handle increasing user traffic and growing datasets without compromising performance.

Throughput: The system should support a high throughput of concurrent user interactions, ensuring optimal performance during peak usage periods.

Reliability:

Availability: The system should have high availability, with minimal downtime and uninterrupted access to movie recommendations for users.

Fault Tolerance: The system should be resilient to failures, with mechanisms in place to recover from errors and maintain service continuity.

Usability:

User Interface Design: The user interface should be intuitive, visually appealing, and easy to navigate, catering to users of all skill levels.

Accessibility: The system should be accessible to users with disabilities, complying with accessibility standards (e.g., WCAG) to ensure equal access for all users.

Consistency: The interface should maintain consistency in design, layout, and interaction patterns across different pages and features of the system.

Maintainability:

Modularity: The system should be modularly designed, with well-defined components and clear separation of concerns, facilitating easier maintenance and future enhancements.

Code Documentation: Source code should be thoroughly documented, with inline comments, README files, and developer guides to aid in understanding and maintaining the system.

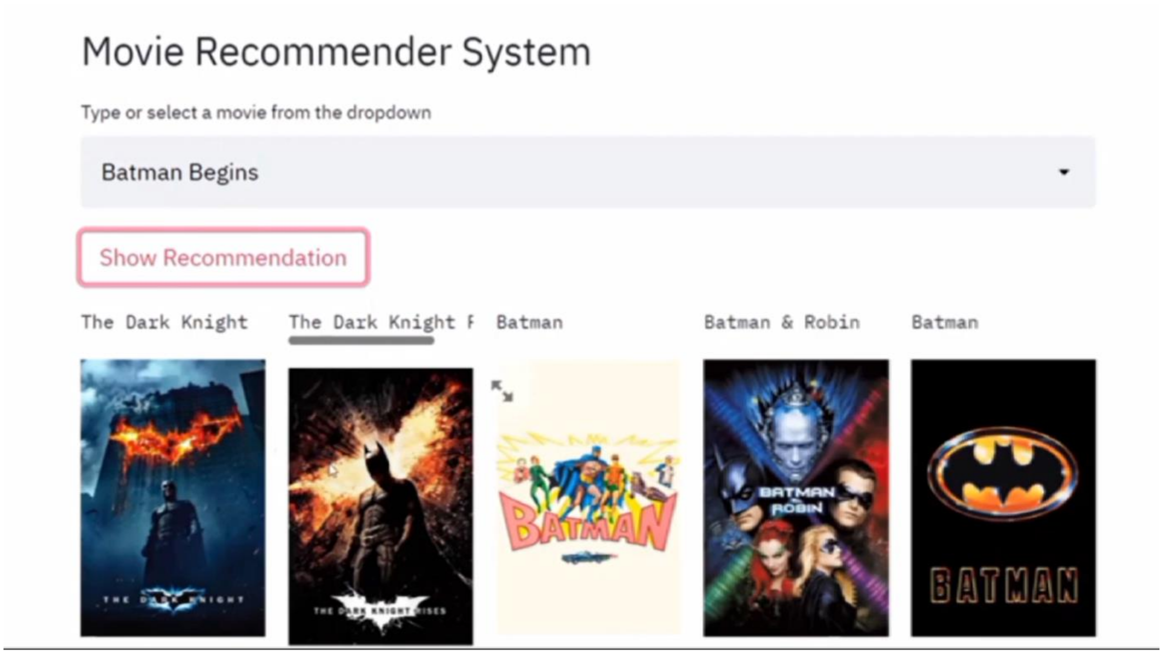
Version Control: The system codebase should be managed using version control systems (e.g., Git), allowing for collaboration, tracking changes, and rolling back to previous versions if needed.

Hardware and Software Requirements

Category	Requirement
Hardware	
Computer	Desktop 4 GHz minimum
Processor	Intel Core i5 or higher
RAM	8GB or higher
Storage	256 SSD or higher
Display	15-inch or large monitor
Software	
Operating System	• Windows 10 or 11 • Mac OS X 10.14 or higher
Web Browser	• Google Chrome • Microsoft Edge • Mozilla Firefox
Programming Languages	Python
Version Control	Git
Other tools and Libraries	• Machine learning libraries (e.g., TensorFlow, Scikit-learn) for recommendation algorithms • Data processing and analysis tools (e.g., Pandas, NumPy) • H2o wave
Development Environment	Integrated Development Environment (IDE) such as PyCharm, Visual Studio Code

Testing Tools	• Unit testing frameworks (e.g., pytest) • End-to-end testing frameworks (e.g., Cypress, Selenium)

Interfaces



Access the System:

Users navigate to the Movie Recommendation System either through a web browser or a dedicated application interface.

Search for a Movie:

On the system's homepage or search page, users will find a search box labeled "Movie Name" or similar.

Users enter the name of the movie they are looking for into the search box. The search functionality may support autocomplete suggestions to assist users in finding the correct movie title.

Once the movie name is entered, users can click on the "Search" button or press "Enter" to initiate the search query.

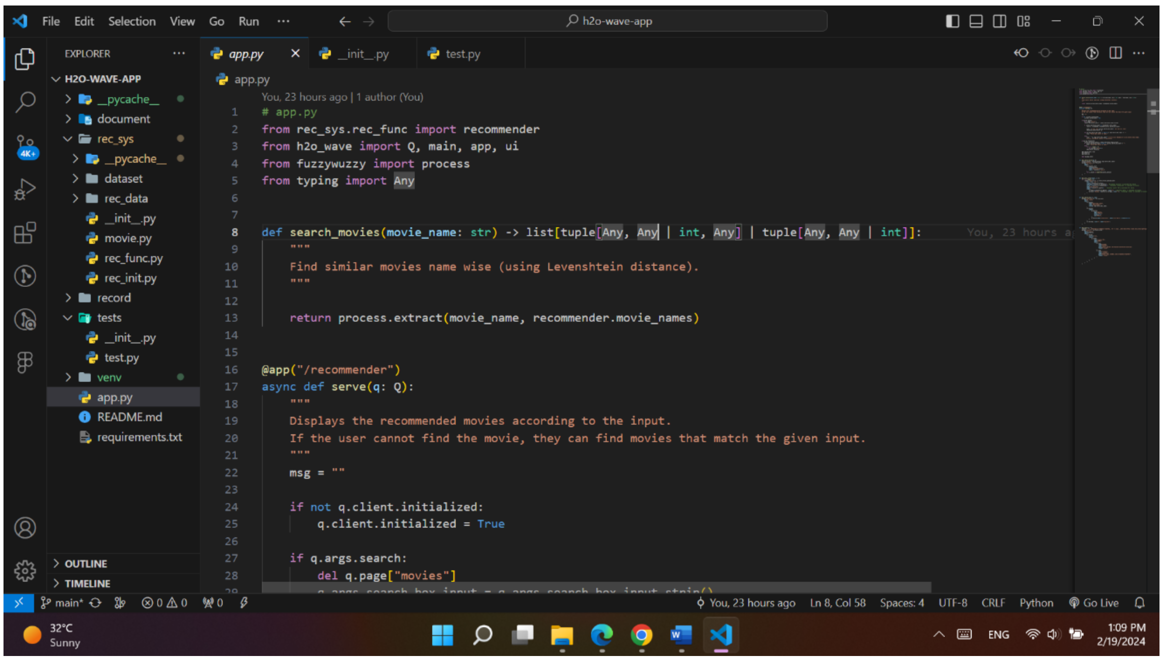
View Search Results:

The system processes the search query and retrieves relevant search results based on the entered movie name.

Search results are typically displayed on the same page, showing matching movie titles, posters, and other relevant information.

Coding

app.py



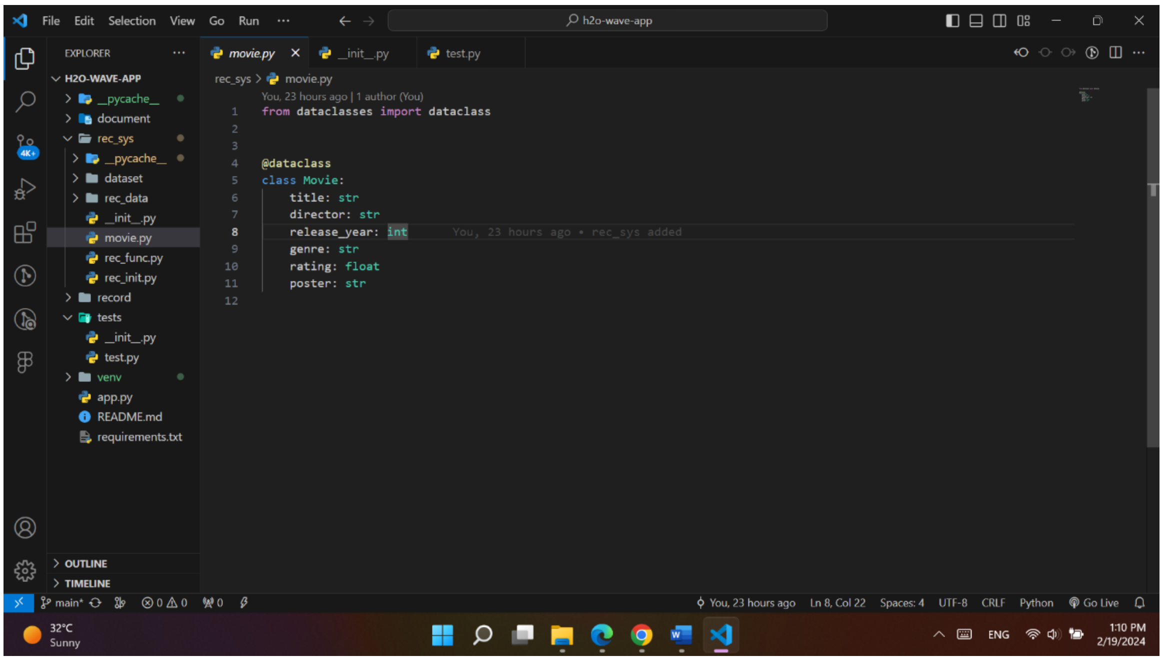
```python
# app.py
from rec_sys.rec_func import recommender
from h2o_wave import Q, main, app, ui
from fuzzywuzzy import process
from typing import Any


def search_movies(movie_name: str) -> list[tuple[Any, Any | int, Any] | tuple[Any, Any | int]]:
    """
    Find similar movies name wise (using Levenshtein distance).
    """

    return process.extract(movie_name, recommender.movie_names)


@app("/recommender")
async def serve(q: Q):
    """
    Displays the recommended movies according to the input.
    If the user cannot find the movie, they can find movies that match the given input.
    """
    msg = ""

    if not q.client.initialized:
        q.client.initialized = True

    if q.args.search:
        del q.page["movies"]
```

The app is designed to allow users to search for movies and get recommendations based on their input. Here's a breakdown of the functionality:

- Movie Search: Users can enter a movie name in the search box and click the "Search" button to get recommendations based on that movie.

- Similar Movies: If the entered movie name is not found in the database, users can click the "Find Movie" button to find movies that match the input using fuzzy matching.
- Display Recommendations: When a movie is found in the database, the app displays recommended movies similar to the input movie.
- UI Elements: The app includes textboxes, buttons, and cards to create a user-friendly interface for interacting with the recommendation system.
- Footer: The footer provides information about the developer and includes links to GitHub and LinkedIn for further contact.

movie.py



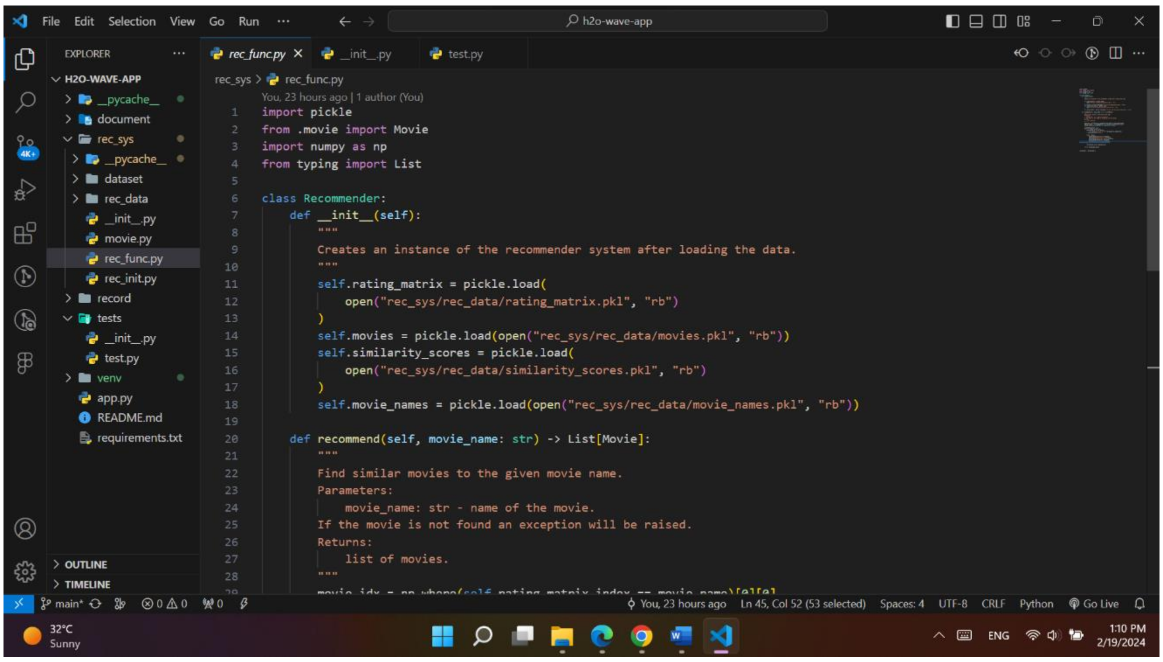
```python
from dataclasses import dataclass


@dataclass
class Movie:
    title: str
    director: str
    release_year: int
    genre: str
    rating: float
    poster: str
```

The movie.py module defines a Movie data class using the dataclass decorator from the dataclasses module. This class represents a movie entity with attributes such as title, director, release year, genre, rating, and poster URL. Here's a breakdown of the attributes:

- title: A string representing the title of the movie.
- director: A string representing the name of the director of the movie.
- release_year: An integer representing the year the movie was released.
- genre: A string representing the genre of the movie.

- rating: A float representing the rating of the movie.
- poster: A string representing the URL of the movie's poster image.

rec_func.py



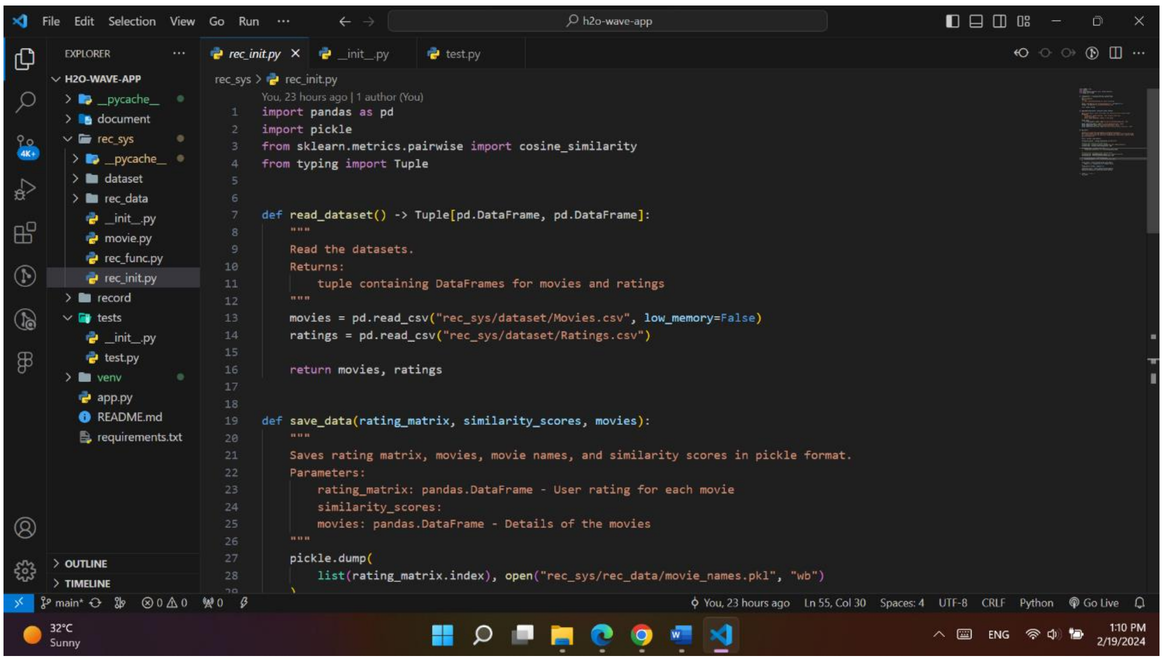
```python
import pickle
from .movie import Movie
import numpy as np
from typing import List

class Recommender:
    def __init__(self):
        """
        Creates an instance of the recommender system after loading the data.
        """
        self.rating_matrix = pickle.load(
            open("rec_sys/rec_data/rating_matrix.pkl", "rb")
        )
        self.movies = pickle.load(open("rec_sys/rec_data/movies.pkl", "rb"))
        self.similarity_scores = pickle.load(
            open("rec_sys/rec_data/similarity_scores.pkl", "rb")
        )
        self.movie_names = pickle.load(open("rec_sys/rec_data/movie_names.pkl", "rb"))

    def recommend(self, movie_name: str) -> List[Movie]:
        """
        Find similar movies to the given movie name.
        Parameters:
            movie_name: str - name of the movie.
        If the movie is not found an exception will be raised.
        Returns:
            list of movies.
        """
```

The rec_func.py module contains the Recommender class, which is responsible for providing movie recommendations based on similarity scores calculated from a rating matrix. Here's a breakdown of the module's functionality:

- Loading Data: Upon initialization, the Recommender class loads the necessary data from pickle files. These include the rating matrix, movie metadata, similarity scores, and movie names.
- Recommendation Method: The recommend method takes a movie name as input and returns a list of recommended movies. It finds the index of the input movie in the rating matrix, retrieves similar items based on similarity scores, and selects the top similar movies. Then, it creates Movie objects for each recommended movie using metadata from the movie dataset.

- Movie Class Import: The Movie class is imported from the movie.py module. This class represents a movie entity with attributes such as title, director, release year, genre, rating, and poster URL.
- Global Instance: A global instance of the Recommender class named recommender is created. This allows other parts of the application to access the recommendation functionality without needing to create new instances.

rec_init.py



```python
import pandas as pd
import pickle
from sklearn.metrics.pairwise import cosine_similarity
from typing import Tuple


def read_dataset() -> Tuple[pd.DataFrame, pd.DataFrame]:
    """
    Read the datasets.
    Returns:
        tuple containing DataFrames for movies and ratings
    """
    movies = pd.read_csv("rec_sys/dataset/Movies.csv", low_memory=False)
    ratings = pd.read_csv("rec_sys/dataset/Ratings.csv")

    return movies, ratings


def save_data(rating_matrix, similarity_scores, movies):
    """
    Saves rating matrix, movies, movie names, and similarity scores in pickle format.
    Parameters:
        rating_matrix: pandas.DataFrame - User rating for each movie
        similarity_scores:
        movies: pandas.DataFrame - Details of the movies
    """
    pickle.dump(
        list(rating_matrix.index), open("rec_sys/rec_data/movie_names.pkl", "wb")
```

The rec_init.py script initializes the recommendation system by computing similarity scores based on collaborative filtering. Here's a breakdown of its functionality:

- Reading Datasets: The read_dataset function reads two datasets: Movies.csv and Ratings.csv. It returns a tuple containing DataFrames for movies and ratings.
- Saving Data: The save_data function saves the rating matrix, movies, movie names, and similarity scores in pickle format.
- Initialization: The rec_init function orchestrates the initialization process. It reads the datasets, filters users who have reviewed more than 200 movies, and movies with equal to or more than 50 ratings to improve the quality of recommendations.

Then, it computes the rating matrix based on the filtered data and calculates similarity scores using cosine similarity.

- Main Block: The __name__ == "__main__" block ensures that the rec_init function is executed when the script is run directly.

Run the application

Follow below steps to run the application.

1. Check the version of Python, must be Python 3.9+ but recommended to use Python 3.10+ for best experience
   ```
   python3 --version
   ```
2. Clone the repository
   ```
   git clone https://github.com/chamikasrimantha/h2o-wave-app.git
   ```
3. Create a virtual environment
   ```
   python3 -m venv venv
   ```
4. Activate the virtual environment
   - Unix/ MacOS
     ```
     source venv/bin/activate
     ```
   - Windows
     ```
     .\venv\Scripts\activate
     ```
5. Install the packages
   ```
   python3 -m pip install -U pip
   python3 -m pip install -r requirements.txt
   ```
6. Run the application
   ```
   wave run app
   ```
7. View the app
   Point your favorite web browser to http://localhost:10101/recommender

Access h2o wave documentation: https://wave.h2o.ai/docs/getting-started

Conclusion

In conclusion, the software design document for the Movie Recommendation System provides a comprehensive blueprint for the development, implementation, and maintenance of a robust and user-centric movie recommendation platform. Throughout the document, various aspects of the system, including its introduction, overview, scope, main goals, proposed system architecture, design, interfaces, codes, have been meticulously outlined.

The document begins with an introduction that sets the context for the system and highlights its significance in enhancing the movie-watching experience for users. It then proceeds to provide an overview of the system's functionalities, outlining its core features such as movie search, recommendation generation, user interaction, and system administration.

The scope of the system is clearly defined, encompassing its intended users, features, and target platforms. This includes considerations for user authentication, movie search and browsing, recommendation generation, user feedback, and system administration functionalities.

The main goals of the system are articulated to ensure that it meets the needs and expectations of its users. These goals focus on providing personalized movie recommendations, enhancing user engagement, and ensuring system scalability, reliability, and security.

The proposed system architecture is presented, detailing the high-level design of the system components, their interactions, and dependencies. This includes the backend infrastructure, frontend interface, database schema, recommendation algorithms, and integration with external services.

The system design section elaborates on the internal workings of the system, including data models, algorithms, and implementation details. It provides insights into how user data is processed, how recommendations are generated, and how user interactions are managed within the system.

Interfaces of the system are described, including user interfaces for movie search, recommendation browsing, and user profile management, as well as backend APIs for data retrieval, processing, and recommendation generation.

Code snippets and examples are provided to illustrate key components and functionalities of the system, offering developers a reference for implementation and customization.

In conclusion, the software design document serves as a comprehensive guide for the development team, stakeholders, and users, providing a clear understanding of the system's architecture, design, functionalities, and testing requirements. It lays the foundation for the successful development, deployment, and maintenance of the Movie Recommendation System, ultimately aiming to deliver an exceptional movie-watching experience for users.