Table of Contents

BESTNEXT

REQUIREMENT ANALYSIS DOCUMENT

1.0

28TH January 2023

MOHANAD ALTARAH
MOHAMMED ALKHODARY
BALKISU SHEHU DANMUSA

OBADA KALO
ATILA ERSUN
ABDULRAHMAN ALFAKIH

Prepared for
SOFT32502—Software ARCHITECTURE
Instructor: TUGBERK ARKOSE
FALL 2023

Introduction

The movie recommendation system is a tool designed to assist users in finding new and relevant films to watch. It utilizes advanced machine learning algorithms to analyze user preferences and viewing history, as well as external data sources such as movie ratings and reviews, to generate personalized recommendations. The system also allows users to search for movies by genre, director, actor, and other criteria, making it easy to discover new films that match their interests. The goal of the system is to enhance the movie-watching experience by helping users find films they will enjoy and to facilitate the discovery of new films.

Purpose of the System

The purpose of the system is to provide users with personalized movie recommendations based on their viewing history and preferences. The system utilizes machine learning algorithms to analyze user data and make tailored suggestions.

Scope of the System

The scope of the system is to provide users with personalized movie recommendations based on their viewing history and preferences. The system will utilize a Model-View-Controller (MVC) architecture, with the model handling the data and recommendation algorithms, the view handling the user interface, and the controller handling the communication between the two. The system will also allow users to search for movies, view movie details and trailers, and save their favorite movies. The system will be accessible through a web application and will be compatible with a variety of web browsers. Additionally, the system will be designed to handle a large number of concurrent users and be highly scalable. Overall, the goal of the system is to enhance the movie-watching experience for users by providing them with relevant and tailored recommendations.

Design Goals

The design goals of the system are to ensure security, reliability, and performance.

Security: is of paramount importance, as the system will handle sensitive user information and movie recommendations. To ensure the security of the system, we will implement robust authentication and authorization mechanisms, as well as encrypt all data transmitted between the system and its users.

Reliability: is also crucial, as the system must be able to provide accurate and up-to-date movie recommendations at all times. To achieve this, we will implement a robust system architecture and use fault-tolerant technologies.

Performance: is also an important design goal, as the system must be able to handle a large number of concurrent users and provide fast and responsive movie recommendations. To achieve this, we will use efficient data storage and retrieval mechanisms, as well as optimize the system for performance. Additionally, the system will have well-defined interfaces to make it easy for users to interact with and for developers to maintain.

Proposed Software Architecture

The proposed software architecture for the movie recommendation system is a microservices-based architecture. It will consist of several independent services that communicate with each other through well-defined APIs. The services will be organized based on their functionality, such as a user service, a recommendation service, and a movie service. These services will be developed using the Model-View-Controller (MVC) design pattern. The user service will handle user authentication and authorization, while the recommendation service will handle the generation of movie recommendations based on user preferences. The system will be divided into multiple microservices, each responsible for a specific functionality such as movie data management, user management, and recommendation generation. The microservices will communicate with each other through well-defined APIs. The movie service will handle the storage and retrieval of movie data. The system will also have a front-end interface for users to interact with the system and receive their recommendations. The system will be designed to be highly available, scalable, and secure. Additionally, well-defined interfaces will be implemented to ensure that the services can be easily integrated with other systems.

System Decomposition

The system is decomposed into three main components: the User Interface, the Recommendation Engine, and the Data Storage. The User Interface is responsible for presenting the movie recommendations to the user and allowing them to provide feedback on the recommendations. The Recommendation Engine uses the user's preferences and feedback, as well as information about the movies in the system, to generate recommendations. The Data Storage component is responsible for storing information about the movies and users, as well as any other relevant data needed for the system to function. Each of these components is built using the Model-View-Controller (MVC) design pattern, which allows for a clear separation of concerns and easier maintenance of the system. The Model is further decomposed into several sub-components, including a movie database, a user database, and a recommendation engine. The movie database stores information about the available movies, such as title, genre, and release date. The user database stores information about the users of the system, including their preferences and viewing history. The recommendation engine uses this information to generate personalized movie recommendations for each user.

Hardware Software Mapping

The hardware-software mapping for our movie recommendation system is as follows:

- The system will run on a Linux-based server with at least 16GB of RAM and an 8-core processor.
- The server will host a PostgreSQL database for storing movie data, user data, and recommendation model information.
- The system will make use of Python for the backend logic and machine learning models. The web application will be built using JavaScript and a web framework such as Flask or Django.
- The system will also make use of various Python libraries such as scikit-learn, TensorFlow, and Pandas for data processing and model building.
- The front-end of the system will be built using a JavaScript framework such as React or AngularJS.
- The system will also leverage cloud-based infrastructure such as AWS or GCP for scalability and easy deployment.

In terms of MVC architecture, the Model will be responsible for all the data processing, machine learning and other data analytics tasks and it will be connected to the database. The View will be the front-end, it will handle the display of data and the user interaction. The Controller will be the bridge between the Model and the View, it will handle the request from the View and provide the data from the Model.

How it Works

The movie recommendation website that we built together utilizes a technique called collaborative filtering to recommend movies to users. Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The basic idea behind collaborative filtering is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue x than to have the opinion on x of a person chosen randomly.

The movie recommendation website uses a specific type of collaborative filtering called user-based collaborative filtering. This method works by finding users who are similar to the current user, and then recommending movies that those similar users have liked. To find similar users, the website calculates the similarity between all pairs of users using a similarity metric, such as Pearson correlation coefficient.

Additionally, the website also uses a technique called content-based filtering. This method works by analyzing the attributes of the items being recommended, such as the genre, cast, and director of a movie, and then recommending similar items based on those attributes.

To tie this with the MVC architecture that we chose to build the website, the model component would be responsible for implementing the collaborative filtering and content-based filtering algorithms, as well as for retrieving and storing the data about users and movies in the database. The controller component would handle the logic for recommending movies to the user, by using the outputs from the model component. And the view component would display the recommended movies to the user, in the form of the carousel that we created.

Machine learning algorithm which we used here is matrix factorization which is a technique to decompose a large sparse matrix into multiple smaller matrices. This algorithm is used for collaborative filtering to find latent features which are then used to predict the ratings of movies.

Functional Requirements

- User registration and login system: Users should be able to create an account and log in to the website.
- Movie and TV show recommendations: The website should be able to recommend new movies and TV shows to users based on their preferences and viewing history.
- Display of movie and TV show ratings: The website should display the ratings of movies and TV shows from various sources, such as IMDb and Rotten Tomatoes.
- Filtering by genre: Users should be able to filter movies and TV shows by genre, such as action, comedy, or drama.
- Filtering by cast: Users should be able to filter movies and TV shows by cast members.
- Streaming services integration: The website should be able to integrate with different streaming services, such as Netflix and Hulu, to allow users to access and watch movies and TV shows.
- Review system: Users should be able to write reviews for movies and TV shows and view reviews written by other users.
- Movie and TV show synopsis: Users should be able to view the synopsis of a selected movie or TV show.
- Trailer viewing: Users should be able to watch the trailer of a selected movie or TV show.
- Watchlist: Users should be able to create and view a watchlist of movies and TV shows they want to watch in the future.
- Poster and release date viewing: Users should be able to view posters and release dates of movies and TV shows.

Non-Functional Requirements

- User interface: The website should have a visually appealing and easy-to-navigate user interface.
- Responsive design: The website should be responsive and accessible from different devices and browsers.
- Loading time: The website should have a fast-loading time.
- Security: The website should have proper encryption and protection of user data.
- Scalability: The website should be scalable to accommodate a large number of users.
- Availability: The website should have high availability and minimal downtime.
- Performance: The website should be able to handle a high volume of requests and traffic.
- Maintenance and updates: The website should be easy to maintain and update.

Technical requirements

- Technologies: The website should be built using technologies such as HTML, CSS, JavaScript, Node.js, and a database such as MySQL or MongoDB.
- Hosting: The website should be hosted on a web server with a domain name.
- Security: The website should be protected by SSL/TLS certificates to ensure secure data transmission.
- API integration: The website should be able to integrate with different streaming services and movie/tv show providers API.
- Backups: Regular backups should be taken to ensure data integrity in case of any issues.

Acceptance Criteria

- The system should be able to recommend movies to users based on their viewing history and preferences.
- The system should be able to handle a large number of concurrent users.
- The system should be able to integrate with external movie databases to retrieve movie information.
- The system should have a user-friendly interface.
- The system should be able to handle updates to movie information and user preferences.
- The system should have secure user authentication and authorization mechanisms.
- The system should be able to generate reports on user viewing history and movie recommendation performance.
- The system should be scalable and able to handle future growth in users and data.
- The system should have a robust and reliable backend system.
- The system should be tested and verified to meet all functional and non-functional requirements before deployment.

Test Cases

1. **Test Case: User registration and login system**

- Test Scenario 1: Verify that a new user can successfully register for an account
  - Input: User provides a valid email address and password
  - Output: User account is created and the user is logged in to the website
- Test Scenario 2: Verify that an existing user can successfully log in to their account
  - Input: User provides a valid email address and password
  - Output: User is logged in to their account
- Test Scenario 3: Verify that an error message is displayed when an invalid email address or password is entered
  - Input: User provides an invalid email address or password
  - Output: Error message is displayed, "Invalid email address or password"

2. **Test Case: Movie and TV show recommendations**

- Test Scenario 1: Verify that the website can recommend new movies and TV shows based on user preferences and viewing history
    - Input: User has selected certain genres and actors as their preferences
    - Output: Website recommends movies and TV shows that match the user's preferences
- Test Scenario 2: Verify that the website can recommend new movies and TV shows based on the user's watchlist
    - Input: User has added movies and TV shows to their watchlist
    - Output: Website recommends movies and TV shows that are similar to the ones on the user's watchlist.

3. **Test Case: Display of movie and TV show ratings**

- Test Scenario 1: Verify that the website can display the ratings of movies and TV shows from various sources such as IMDB and Rotten Tomatoes
  - Input: User selects a movie or TV show from the website - Output: The website displays the ratings of the movie or TV show from various sources
- Test Scenario 2: Verify that the website can display the user reviews
    - Input: User selects a movie or TV show from the website
    - Output: The website displays the user reviews of the movie or TV show.

4. **Test Case: Filtering by genre and cast**

- Test Scenario 1: Verify that the website can filter movies and TV shows by genre
    - Input: User selects the genre "action"
    - Output: The website displays a list of action movies and TV shows
- Test Scenario 2: Verify that the website can filter movies and TV shows by cast
    - Input: User enters the name of a specific actor
    - Output: The website displays a list of movies and TV shows that feature the selected actor.

5. **Test Case: Streaming services integration**

- Test Scenario 1: Verify that the website can integrate with different streaming services
    - Input: User selects a movie or TV show from the website
    - Output: The website displays the different streaming services where the movie or TV show is available
- Test Scenario 2: Verify that the website can redirect the user to the streaming service page
    - Input: User selects a streaming service from the website
    - Output: The website redirects the user to the streaming service page where they can access the movie or TV show.

6. **Test Case: Review system**

- Test Scenario 1: Verify that the users can write reviews for movies and TV shows
    - Input: User writes a review for a movie or TV show
    - Output: The website saves the review and displays it on the movie or TV show page
- Test Scenario 2: Verify that the users can view reviews written by other users
    - Input: User selects a movie or TV show from the website

    - Output: The website displays the reviews written by other users.

7. **Test Case: Movie and TV show synopsis and trailer viewing**

- Test Scenario 1: Verify that the users can view the synopsis of a movie or TV show
    - Input: User selects a movie or TV show from the website
    - Output: The website displays the synopsis of the movie or TV show
- Test Scenario 2: Verify that the users can watch the trailer of a selected movie or TV show
    - Input: User selects a movie or TV show from the website
    - Output: The website displays the trailer of the movie or TV show.

8. **Test Case: Watchlist**

- Test Scenario 1: Verify that the users can create a watchlist of movies and TV shows
    - Input: User adds a movie or TV show to their watchlist
    - Output: The movie or TV show is added to the user's watchlist
- Test Scenario 2: Verify that the users can view their watchlist
    - Input: User navigates to their watchlist
    - Output: The website displays the movies and TV shows that are in the user's watchlist.

9. **Test Case: Poster and release date viewing**

- Test Scenario 1: Verify that the users can view posters of movies and TV shows
    - Input: User selects a movie or TV show from the website
    - Output: The website displays the poster of the movie or TV show
- Test Scenario 2: Verify that the users can view the release date of movies and TV shows - Input: User selects a movie or TV show from the website - Output: The website displays the release date of the movie or TV show.