**Software Requirements Specification (SRS) Document for Movie Recommendation System**

1. **Introduction**

* 1.1 Purpose

The purpose of this document is to provide a detailed Software Requirements Specification (SRS) for the Movie Recommendation System. This system will recommend movies to users based on their preferences, previously watched movies, and ratings. The system will also allow users to add movies they have watched, along with their ratings and feedback, to a MongoDB database. The recommendation engine will use a similarity function based on various attributes such as cast, director, genres, and ratings.

* 1.2 Scope

The Movie Recommendation System will provide personalized movie recommendations to users. The system will have the following features:

- Movie recommendation based on similarity (using cosine similarity and count vectorization).

- User functionality to add watched movies, ratings, and feedback.

- A backend API built using Flask to handle recommendation logic.

- A frontend built using Node.js to display recommended movies and allow user interaction.

- MongoDB for storing user data and movie information.

* 1.3 Definitions, Acronyms, and Abbreviations

- SRS: Software Requirements Specification

- API: Application Programming Interface

- UI: User Interface

- DB: Database

- Flask: A Python web framework used for building APIs.

- Node.js: A JavaScript runtime used for building the frontend.

- MongoDB: A NoSQL database used for storing user and movie data.

- Cosine Similarity: A metric used to measure how similar two items are.

- Count Vectorization: A method to convert text data into numerical data for similarity comparison.

* 1.4 References

- Flask Documentation: https://flask.palletsprojects.com/

- Node.js Documentation: https://nodejs.org/en/docs/

- MongoDB Documentation: https://www.mongodb.com/docs/

- Cosine Similarity: https://en.wikipedia.org/wiki/Cosine\_similarity

- Count Vectorization: <https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction>

2**. Overall Description**

* 2.1 Product Perspective

The Movie Recommendation System is a web-based application that will provide personalized movie recommendations to users. The system will use a pre-trained dataset of movies and a similarity function to recommend movies based on various attributes such as cast, director, genres, and ratings. The system will also allow users to add movies they have watched, along with their ratings and feedback, to a MongoDB database.

* 2.2 Product Features

- Movie Recommendation: The system will recommend movies based on similarity using cosine similarity and count vectorization.

- User Profile: Users can add movies they have watched, along with their ratings and feedback.

- Backend API: The backend API will handle the recommendation logic and interact with the MongoDB database.

- Frontend: The frontend will display recommended movies and allow users to interact with the system.

- Database: MongoDB will store user data, movie information, and feedback.

* 2.3 User Classes and Characteristics

- End Users: Users who will interact with the system to get movie recommendations and add their watched movies.

- Admin: Admins will have access to the database and can manage the movie dataset.

* 2.4 Operating Environment

- Frontend: Node.js, HTML, CSS, JavaScript

- Backend: Flask (Python), MongoDB

- Database: MongoDB (NoSQL)

- Operating System: Cross-platform (Windows, Linux, macOS)

- Browser: Chrome, Firefox, Safari, Edge

* 2.5 Design and Implementation Constraints

- The system must be able to handle a large dataset of movies.

- The recommendation engine must be optimized for performance to ensure quick response times.

- The system must be scalable to handle multiple users simultaneously.

* 2.6 Assumptions and Dependencies

- The system ensures that the movie dataset is pre-trained and available.

- The system depends on MongoDB for storing user data and movie information.

- The system assumes that users have access to a modern web browser.

3**. System Features**

* 3.1 Movie Recommendation

- Description: The system will recommend movies based on similarity using cosine similarity and count vectorization.

- Input: Movie name provided by the user.

- Process: The system will compare the input movie with other movies in the dataset based on attributes such as cast, director, genres, and ratings.

- Output: A list of 8 recommended movies.

* 3.2 User Profile

- Description: Users can add movies they have watched, along with their ratings and feedback.

- Input: Movie name, rating, feedback, and genres.

- Process: The system will store the user's watched movies, ratings, and feedback in the MongoDB database.

- Output: Confirmation of successful addition of the movie to the user's profile.

* 3.3 Backend API

- Description: The backend API will handle the recommendation logic and interact with the MongoDB database.

- Input: API calls from the frontend.

- Process: The API will process the input, run the similarity function, and return the recommended movies.

- Output: A list of recommended movies.

* 3.4 Frontend

- Description: The frontend will display recommended movies and allow users to interact with the system.

- Input: User input (movie name, ratings, feedback).

- Process: The frontend will send API requests to the backend and display the results.

- Output: A list of recommended movies and user feedback.

4. **External Interface Requirements**

* 4.1 User Interfaces

- Login/Signup Page: Users can create an account or log in to their existing account.

- Movie Search Page: Users can search for a movie to get recommendations.

- Recommendation Page: Displays the list of recommended movies.

- User Profile Page: Users can add movies they have watched, along with their ratings and feedback.

* 4.2 Hardware Interfaces

- The system will run on standard web browsers and does not require any special hardware interfaces.

* 4.3 Software Interfaces

- Frontend: Node.js, HTML, CSS, JavaScript

- Backend: Flask (Python)

- Database: MongoDB

* 4.4 Communication Interfaces

- The frontend and backend will communicate via RESTful API calls.

- The backend will communicate with MongoDB using the MongoDB driver for Python.

**5. Non-Functional Requirements**

* 5.1 Performance Requirements

- The system should be able to recommend movies within 2-3 seconds.

- The system should be able to handle at least 1000 concurrent users.

* 5.2 Security Requirements

- User data should be stored securely in the MongoDB database.

- The system should use HTTPS for secure communication between the frontend and backend.

* 5.3 Usability Requirements

- The system should have a user-friendly interface that is easy to navigate.

- The system should provide clear feedback to users when they add movies or request recommendations.

* 5.4 Scalability Requirements

- The system should be scalable to handle an increasing number of users and movies in the dataset.

**6. Budget Estimation**

* 6.1 Development Costs

- Frontend Development: $5,000 - $7,000

- Node.js, HTML, CSS, JavaScript development.

- UI/UX design and implementation.

- Backend Development: $6,000 - $8,000

- Flask API development.

- Integration with MongoDB.

- Implementation of the recommendation engine (cosine similarity, count vectorization).

- Database Setup: $2,000 - $3,000

- MongoDB setup and configuration.

- Database schema design for storing user data and movie information.

* 6.2 Hosting and Infrastructure Costs

- Cloud Hosting (AWS, GCP, Azure): $1,000 - $2,000 per year

- Hosting the frontend and backend.

- MongoDB cloud hosting or self-hosted MongoDB.

7**. Appendices**

* 7.1 Glossary

- Cosine Similarity: A metric used to measure the similarity between two vectors.

- Count Vectorization: A method to convert text data into numerical data for similarity comparison.

- MongoDB: A NoSQL database used for storing data in a flexible, document-oriented format.

* 7.2 References

- Flask Documentation:<https://flask.palletsprojects.com/>

- Node.js Documentation: <https://nodejs.org/en/docs/>

- MongoDB Documentation: <https://www.mongodb.com/docs/>

- Cosine Similarity:<https://en.wikipedia.org/wiki/Cosine_similarity>

- Count Vectorization: <https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction>

This SRS document provides a comprehensive overview of the Movie Recommendation System, including its features, requirements, and budget estimation. The system is designed to provide personalised movie recommendations to users based on their preferences and previously watched movies.