Tribhuvan University

Faculty of Humanities and Social Sciences

A PROJECT REPORT

On

MOVIE RECOMMENDATION SYSTEM

Submitted to

Department of Computer Application National College

In partial fulfilment of the requirements for the Bachelors in Computer Application

Submitted by

Suyashaa Vaidya

T.U. Reg: 6-2-366-35-2018

4th Poush, 2080

Under the Supervision of

Er. Nirajan Panthee

Tribhuvan University

Faculty of Humanities and Social Sciences

National College

Supervisor's Recommendation

I hereby recommend that this project prepared under my supervision by **SUYASHAA VAIDYA** entitled "**MOVIE RECOMMENDATION SYSTEM**" in partial fulfilment of the requirements for the degree of Bachelor of Computer Application is recommended for the final evaluation.

____

SIGNATURE

Er. Nirajan Panthee

SUPERVISOR

Lecturer

Computer Science

National College, Lainchaur

Tribhuvan University

Faculty of Humanities and Social Sciences

National College

Letter of Approval

This is to certify that this project prepared by Miss **SUYASHAA VAIDYA** entitled "**MOVIE RECOMMENDATION SYSTEM**" in partial fulfilment of the requirements for the degree of Bachelor in Computer Application has been evaluated. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

SIGNATURE of Supervisor Er. Nirajan Panthee Computer Science National College ,Lainchaur	**SIGNATURE of HOD/Coordinator** Mr. Navaraj Heka Computer Science National College, Lainchaur
SIGNATURE of Internal Examiner **Internal Examiner**	**SIGNATURE of External Examiner** **External Examiner**

Acknowledgement

I am sincerely grateful to National College for their steadfast support and for providing a platform for our project. I extend my heartfelt thanks to our respected Academic Director, Sir Navaraj Heka, for his unwavering support and effective management throughout the project. My gratitude also goes to our dedicated teachers/supervisors, Mr. Prakash Kafle and Er. Nirajan Panthee, for their consistent support, which was instrumental in the success of our project.

I would like to express my appreciation to all the experts who generously shared their knowledge and contributed to the project. A special acknowledgment goes to the teachers at National College for creating a conducive learning environment, allowing us to explore various IT equipment and tools, understand their working principles, and gain insights into design and control systems related to our subject.

I want to convey my thanks to the staff in all departments, the Administration, and site organization for their continuous assistance. Special appreciation is due to the working staff involved in these subjects for providing guidance that facilitated the completion of our project.

In addition, I extend my heartfelt thanks to my parents and friends who played a crucial role in helping me finalize the project within the limited time frame. Their encouragement and assistance were invaluable throughout the process.

Abstract

The Movie Recommendation System implemented in this project makes use of the latest technologies and machine learning algorithms to provide a culturally diverse and engaging movie-watching experience. Three user modules content manager, content supervisor, and movie enthusiast are supported by the system, which was built using Python for machine learning components, Nest.js for endpoints, and React.js for the frontend. The pivotal functionality revolves around content-based filtering utilizing cosine similarity, thereby augmenting the precision of movie suggestions. The user interface allows for diverse interactions such as movie searches, detailed exploration, trailer viewing, commenting, and rating. In the machine learning domain, web scraping is used to gather data in order to create an extensive database of movie information. Preprocessing includes cleansing, normalization, and addressing missing values for the data that has been gathered. To extract pertinent data from the dataset, feature extraction is used. After that, textual data is processed via tokenization, which makes it easier to extract useful features. By computing their similarity scores, movies' degrees of resemblance are determined using cosine similarity. By recognizing films with similar qualities, this similarity metric helps to improve recommendation accuracy. The frontend easily incorporates the computed similarity ratings, enhancing the user experience by offering tailored and contextually appropriate movie recommendations. Evaluation metrics are used to evaluate the effectiveness of the recommendation system, including recall, precision, and F1 score. With its selection of Hindi, English, and Nepali films, the Movie Recommendation System not only satisfies modern user expectations but also highlights cultural diversity. By adding this, the underrepresentation of regional material in recommendation systems is lessened and the variety of recommended films is increased. To sum up, this movie recommendation system combines state-of-the-art technology, machine learning algorithms, and a multicultural perspective to provide a dynamic and tailored cinematic experience. By incorporating sophisticated assessment indicators, the system's performance is thoroughly evaluated, ensuring that consumers receive personalized and superior movie recommendations.

Keywords: Similarity, Recommendation, Analysis, Movies, IMDB, Octoparse, Kaggle

Table of Contents

List of Abbreviations

API	Application Programming Interface
AUTH	Authentication
BCA	Bachelors in Computer Application
CASE	Computer Aided Software Engineering
CBF	Content-Based Filtering
ER	Entity Relationship
ID	Identification
IDE	Integrated Development Environment
IMDb	Internet Movie Database
KNN	K-Nearest Neighbors
SciPy	Scientific Python
TF-IDF	Term Frequency-Inverse Document Frequency
TMDB	The Movie Database
TU	Tribhuvan University
UI	User Interface
UML	Unified Modeling Language
UX	User Experience

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Introduction

The Movie Recommendation System is a user-focused platform created to help people find and explore films that fit their preferences. Users can quickly search for films that interest them and receive suggestions based on their searched movie. The system uses sophisticated algorithms, such as the content-based filtering algorithm, to deliver seamless and enjoyable movie discovery experiences. The data used in the system were collected from reputable sources, including IMDb datasets through web scraping using Octoparse, ensuring a diverse and comprehensive movie database for accurate recommendations. It contains movies in three different languages: Nepali, Hindi, and English.

This system also includes a valuable review adding feature that allows users to review movies by commenting and rating them. Additionally, the system is built with three different user roles: Content Supervisor, Content Manager, and Movie Enthusiast, which helps manage the system in different ways. The Content Manager can add movies, which must be approved by the Content Supervisors, ensuring a meticulous curation process that maintains the integrity, quality, and relevance of the movie database. This approval mechanism guarantees that only content meeting predefined criteria and aligning with the platform's standards is introduced, enhancing the overall user experience and upholding the system's commitment to excellence. This empowers users to make well-informed decisions about which movies to watch, ultimately enhancing their overall movie-watching experience.

The system utilizes Python Flask for implementing the machine learning algorithm and the recommendation API, while it also uses Nest.js for creating endpoints of the system as the backend and React for the frontend, resulting in a seamless and user-friendly interface. User and movie data are efficiently stored in a MySQL database, ensuring data integrity and dependable system performance. With its user-centric approach, the Movie Recommendation System aims to be an unbiased and indispensable companion for movie enthusiasts seeking to discover the perfect films for their entertainment pleasure. With a strong focus on user satisfaction and an ever-growing repository of movie data, the Movie Recommendation System aspires to be the go-to platform for all movie enthusiasts seeking an enriched and enjoyable cinematic experience.

1.2 Problem Statement

1. Limited user contributions and engagement in content addition.
2. Lack of meticulous curation in traditional movie recommendation systems.
3. Underrepresentation of Cross-Language Recommendations especially Nepali movies.

1.3 Objectives

The objective is to create a user-focused platform for movie recommendations, integrating content-based filtering using React js for frontend, Nest js for backend and python for implementing the cosine similarity algorithm, and ensuring data security and continuous improvement.

1.4 Scope and Limitations

1.4.1 Scope

1. Comprehensive Movie Recommendations: The system will aim to provide personalized and relevant movie recommendations to users based on their preferences and behavior.
2. User-Friendly Interface: The platform will offer a user-friendly interface with intuitive navigation and search functionalities for an enhanced user experience.
3. Diverse Movie Database: The system will integrate a comprehensive movie database from reputable sources, ensuring a wide range of movie options for users. Continuous Improvement: The platform will be designed for continuous improvement, incorporating user feedback and updates to enhance functionality and features.

1.4.2 Limitations

1. Genre Dependency: This refers to a system's reliance on movie genres as a primary factor for recommendations. Such systems may struggle when users have eclectic tastes that span multiple genres.

2. Metadata Accuracy: It pertains to the precision and reliability of the data used for recommendations. If the metadata describing movies is inaccurate or incomplete, the system's ability to provide relevant suggestions is compromised.
3. Collaborative Limitation: This points to a constraint in collaborative filtering methods, where recommendations are based on user behavior and preferences. It may face challenges when dealing with new or niche items or when user interactions are sparse, limiting the accuracy of suggestions
4. Algorithm Accuracy: The movie recommendation algorithm's accuracy may vary, and some users may receive recommendations that do not align with their tastes.
5. Scalability: As the user base grows, the system's performance may face challenges in handling increased traffic and generating real-time recommendations.
6. User Privacy Concerns: Despite efforts to ensure data security, there may still be concerns about user privacy when collecting and storing user data.

1.5 Development Methodology

The movie recommendation app was created using the straightforward and sequential Waterfall Software Development Life Cycle model.

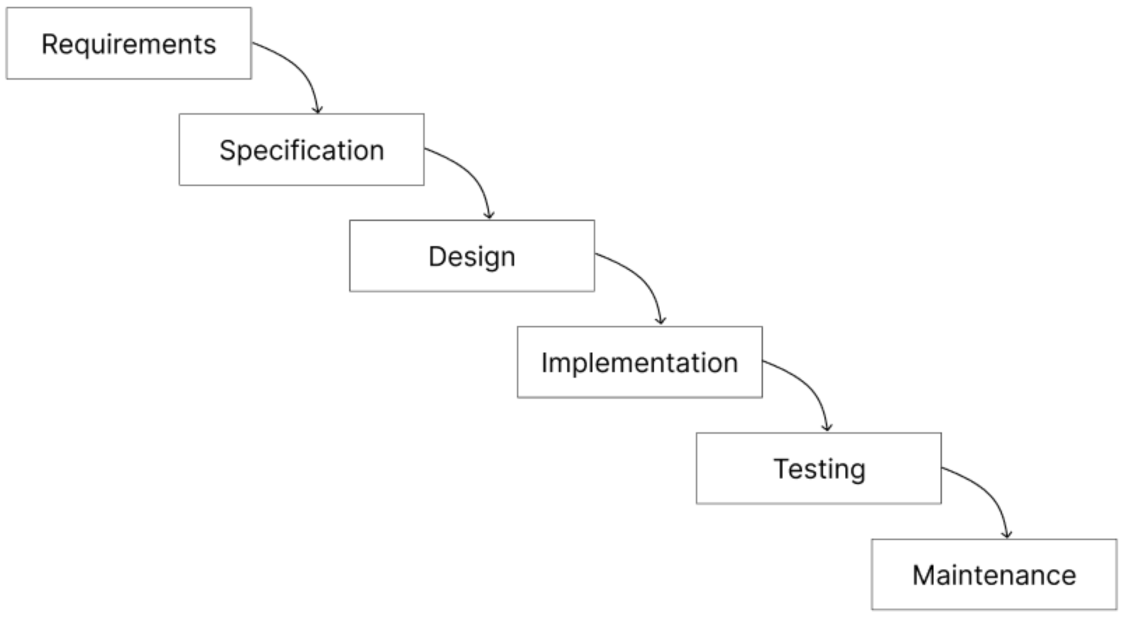


Figure 1 Waterfall Model

1. **Requirement Analysis**: All the requirements for making a Movie Recommendation System including time and resource needs, are listed. These consist of an IDE for coding, a database for storing and retrieving data, and an OS for running the program.
2. **Design**: The design of this Movie Recommendation System was done using Figma which is an online tool to design high and low-fidelity images/ diagrams.
3. **Development**: The development phase consists of coding a function or a module. In this, every class was made for the system, and their functions were added to the next iterations
4. **Testing**: After development of each module it was tested and if any test failed it was revised and the problem was found out and tested again until it is solved.
5. **Deployment**: The system was tested and reviewed and verified by the subject teacher i.e., our supervisor every week in the development process.

1.6 Report Organization

This report consists of five chapters which will cover the designing and the development of Movie Recommendation System.

Chapter One: This chapter includes the system and the problems, given an overview about the study. It includes introduction, problem of statement, objectives, and solution of the website.

Chapter Two: This chapter covers the literature review which is the previous related work that has been done before. It also includes the description of the website where we have taken the reference.

Chapter Three: This chapter explains the selected methodology that we are going in this project. This chapter shows the design of the system. It includes ER- Diagram, State Diagram, Sequence Diagram, Activity Diagram etc.

Chapter Four: This chapter discusses the implementation and testing. It includes the overall description of the modules that we have implemented into our application. We have shown the test cases.

Chapter Five: This chapter discusses the conclusion, recommendation and future works to improve this study.

Chapter 2: Background Study and Literature Review

2.1 Background Study

The background study of the movie recommendation system with review analysis focuses on evaluating and analyzing the strengths and weaknesses of current movie recommendation systems. This study aims to explore their methods for user review analysis and identify areas for enhancement. To conduct this research, we will gather and analyze relevant literature, research papers, and articles related to movie recommendation systems that incorporate user reviews and sentiment analysis. Additionally, we will assess the performance and user satisfaction of existing systems like Netflix and Amazon Prime, comparing them with platforms that utilize review analysis, such as IMDB, TMDB, and Rotten Tomatoes. The findings from this study will serve as a foundation for improving the proposed Movie Recommendation System and addressing any identified gaps and opportunities for enhancement. Some of the key aspects that we will evaluate in existing systems include:

1. User Review Analysis Techniques:

- What techniques are currently used to analyze user reviews and sentiments?
- How accurate and effective are these techniques in predicting user preferences and providing personalized recommendations?
- What are the limitations of these techniques, and how can they be improved?

2. Recommendation Algorithms:

- What algorithms are currently used to generate movie recommendations?
- How do these algorithms incorporate user review analysis, and what impact does this have on their performance?
- How can these algorithms be improved to provide more personalized and accurate recommendations?

3. User Satisfaction and Engagement:

- How satisfied are users with existing movie recommendation systems?
- How do they perceive the impact of user review analysis on their movie-watching experience?
- What factors influence user engagement and retention, and how can they be optimized?

We can find the best practices and new techniques, as well as the gaps and chances for improvement in movie recommendation systems using review analysis, by conducting a complete analysis of the existing systems.

2.2 Literature Review

The literature on movie recommendation systems is vast and encompasses a wide array of approaches and techniques to enhance user experiences and provide personalized recommendations. One prominent direction that has emerged is the utilization of deep learning-based recommender systems, which leverage their capacity to process vast amounts of data and understand complex user-item interactions [1].

Collaborative filtering algorithms have also been extensively studied and compared to determine their effectiveness in making recommendations [2]. These algorithms work by identifying patterns and similarities between users or items to generate relevant suggestions.

Online platforms like IMDb play a significant role in the realm of movie recommendation systems. IMDb provides ratings, reviews, and essential information about movies and TV shows, supporting users in their decision-making process [3].

As recommender systems continue to evolve, research on aspects like content-based filtering, context-aware filtering, and sentiment analysis remains an active area. These areas hold promise in further enhancing recommendation accuracy and personalization. [4]

Different algorithms for content based filtering were researched like cosine similarity, KNN etc. [5]

Sentiment analysis has been proposed as a valuable tool to augment recommender systems by considering user sentiment and feedback. By analyzing user reviews and ratings, sentiment analysis can aid in making more accurate recommendations that align with users' preferences and emotions [6].

Connected papers and academic resources contribute to the consolidation of research efforts and facilitate deeper insights into the challenges and potential advancements in recommender systems. Access to such resources enables researchers and practitioners to stay up-to-date with the latest developments in the field [7].

To create movie recommendation systems, various resources are available, including tutorials and guides that explore techniques like weighted hybrid approaches and creating recommendation systems using nearest neighbors [8][9]. These tutorials offer practical insights into building effective recommendation systems.

Cosine similarity and cosine distance have been widely used in measuring similarity between items and users to produce accurate and relevant recommendations [10]. These similarity metrics are fundamental in collaborative filtering and content-based filtering approaches.

The integration of APIs, such as Flask for back-end development, React for front-end, and MySQL for efficient data storage, enables the seamless functioning of movie recommendation systems [11] Utilizing these technologies streamlines the development process and allows for scalability and robustness in handling large datasets.

While significant strides have been made in movie recommendation systems, there remains a need for further exploration and development of reliable evaluation metrics to assess the performance and user satisfaction of recommender systems [12]. Robust evaluation metrics are essential to gauge the effectiveness of different recommendation algorithms accurately[13].

From deep learning-based approaches to collaborative filtering algorithms, sentiment analysis, and diverse filtering techniques, researchers and developers continue to explore ways to create robust and user-centric movie recommendation systems [15][16].

In conclusion, the literature review showcases a diverse landscape of techniques and methodologies in the movie recommendation systems domain. The integration of APIs and the availability of online platforms like IMDb play crucial roles in supporting these efforts. Further research in context-aware filtering, explainability, cold-start problems, dataset biases, and resource optimization is expected to shape the future of recommender systems.

Chapter 3: System Analysis and Design

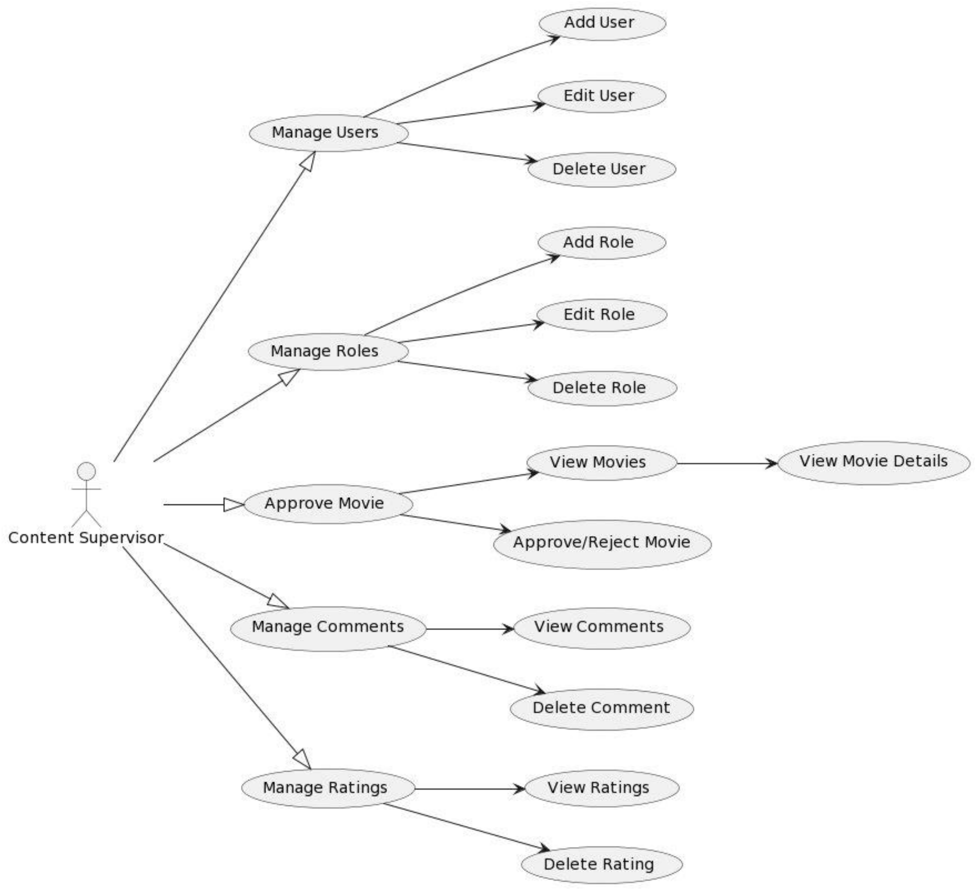
3.1 System Analysis

3.1.1 Requirement Analysis

Requirement analysis is a crucial step in the development of any software system, including Movie Recommendation System. The following are some of the key requirements for a movie recommendation system:

1. Data collection: The system should have the capability to collect data from various sources, including user reviews, ratings, and other relevant information.
2. Data preprocessing: The collected data needs to be cleaned, filtered, and transformed into a format suitable for analysis. This involves removing irrelevant or duplicated data, converting the data into a standardized format, and performing other necessary preprocessing steps.
3. Tokenization: The text data undergoes tokenization, a crucial step where the raw text is broken down into smaller units called tokens, such as words or phrases. This process facilitates the analysis of textual information by converting it into a more manageable and structured form.
4. Feature Extraction: Extracting relevant features from the processed data is essential for building effective models. Feature extraction involves selecting and transforming the most significant characteristics of the data, enabling the model to focus on key aspects during analysis and enhancing the overall performance of the system.
5. Machine learning algorithms: The system should employ machine learning algorithms to learn from the user data and make better recommendations. These algorithms can include collaborative filtering, content-based filtering, or hybrid approaches.
6. Evaluation metrics: The system should have appropriate evaluation metrics to assess the quality of the recommendations. This can include metrics such as precision, recall, and f1_score.
7. User interface: The system should have a user-friendly interface that allows users to easily provide feedback, rate movies, and view personalized recommendations.
8. Scalability: The system should be scalable to handle large amounts of data and growing user bases.

9. Security: The system should ensure the privacy and security of the user data, including encryption, access control, and authentication mechanisms.



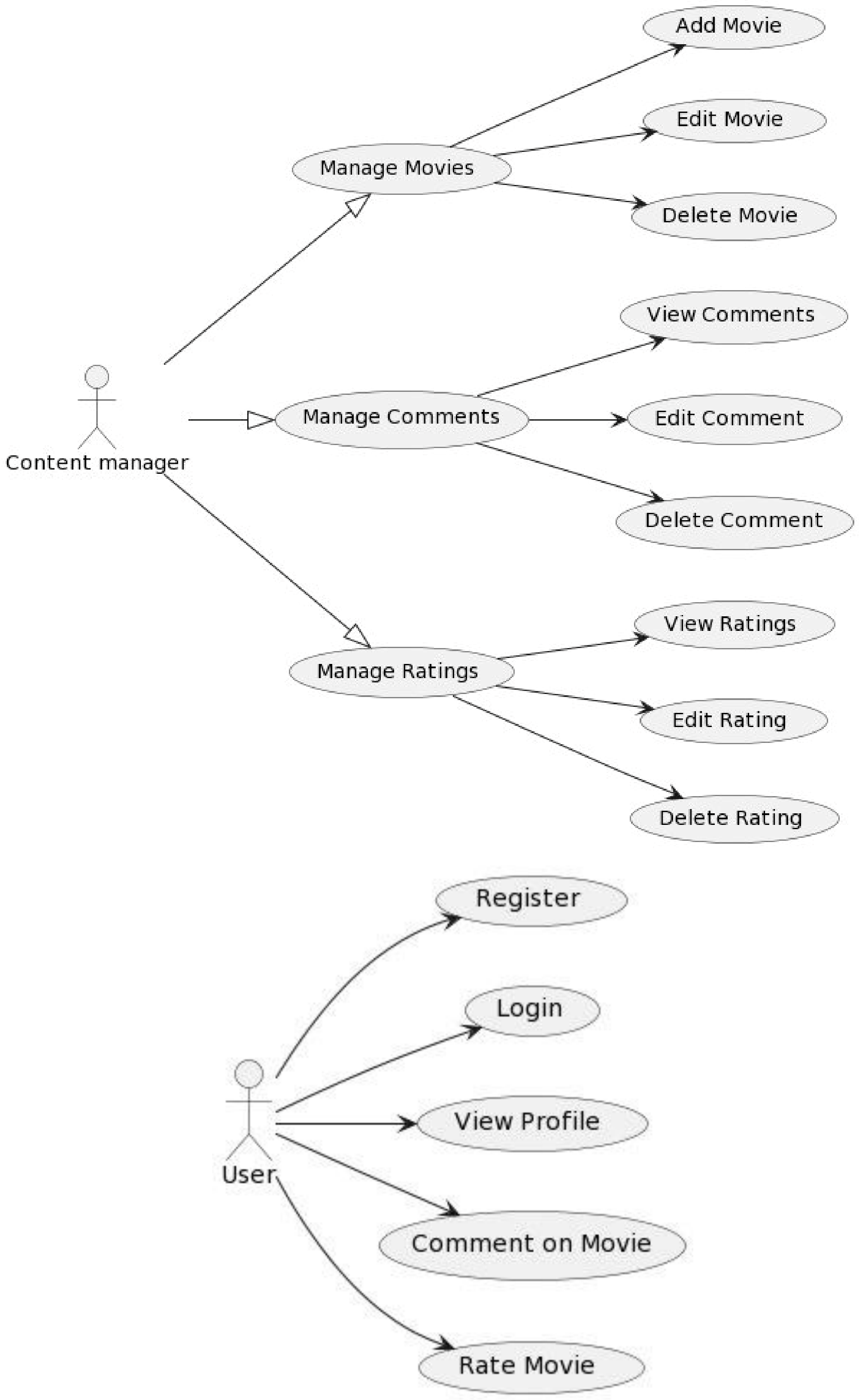


Figure 2 Use Case Diagram of Movie recommendation system

3.1.2 Feasibility Analysis (Technical, operational, economic)

In feasibility study, the Movie Recommendation System’s technical, operational, financial, & legal factors are analyzed.

i. **Technical**

The technical feasibility study aims to determine if the proposed movie recommendation system with review analysis is technically possible within the constraints of available technology and resources. The required software and hardware components will be assessed in terms of their compatibility, availability, and cost-effectiveness. The technical team will conduct tests to evaluate the performance, scalability, and reliability of the system.

ii. **Operational**

The operational feasibility study aims to determine whether the proposed movie recommendation system with review analysis can be integrated and implemented within the existing organizational processes and practices. The operational team will assess the impact of the proposed system on the current workflow, staff training needs, and user acceptance. The team will also analyze the system's maintenance and support requirements to ensure the system's sustainability.

iii. **Economic**

The economic feasibility study aims to determine the financial viability of the proposed movie recommendation system with review analysis. The economic team will analyze the system's development, implementation, and operational costs and compare them with the potential benefits and returns. The team will also evaluate the system's long-term profitability, potential risks, and impact on the organization's financial stability. Based on the analysis, the team will determine if the proposed system is financially feasible and justifiable.

3.1.3 Data Modelling (ER-Diagram)

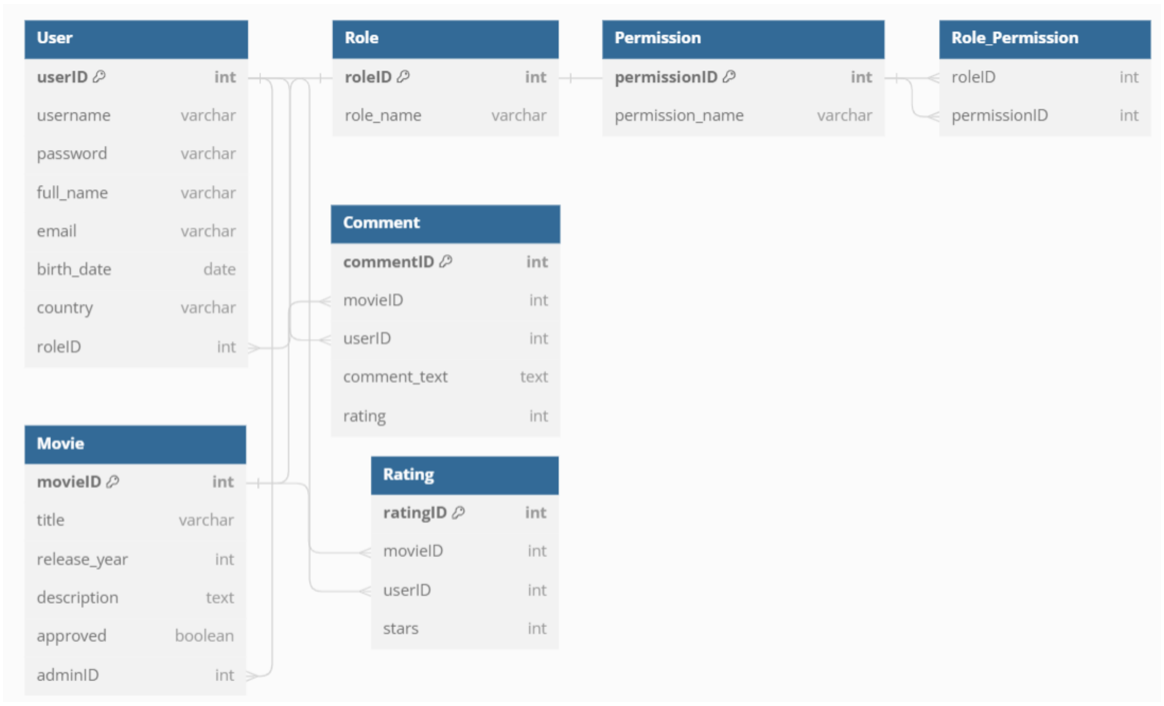


Figure 3 ER Diagram of Movie Recommendation System

In the entity-relationship diagram (ERD) of the movie recommendation system, the USER entity serves as the central hub, capturing user details such as names, email addresses, and roles like superadmin, admin, or user. Each user is associated with multiple COMMENT entities, reflecting their engagement with movies through reviews and ratings. The MOVIE entity encapsulates comprehensive information about each film, including its title, release year, runtime, description, genre, and director. Users and movies are linked through the COMMENT entity, where users express their sentiments and provide ratings for specific films. The system accommodates diverse roles, ensuring that superadmins have control over movie approval status, marked by the SUPER_ADMIN_APPROVED attribute. Additionally, the AUTH entity includes fields for password reset tokens and expiration timestamps, enhancing user account security. This ERD enables seamless data flow, facilitating user interactions, personalized movie recommendations, and content based filtering based recommendations.

3.1.4 Object Modelling using Class and Object Diagrams

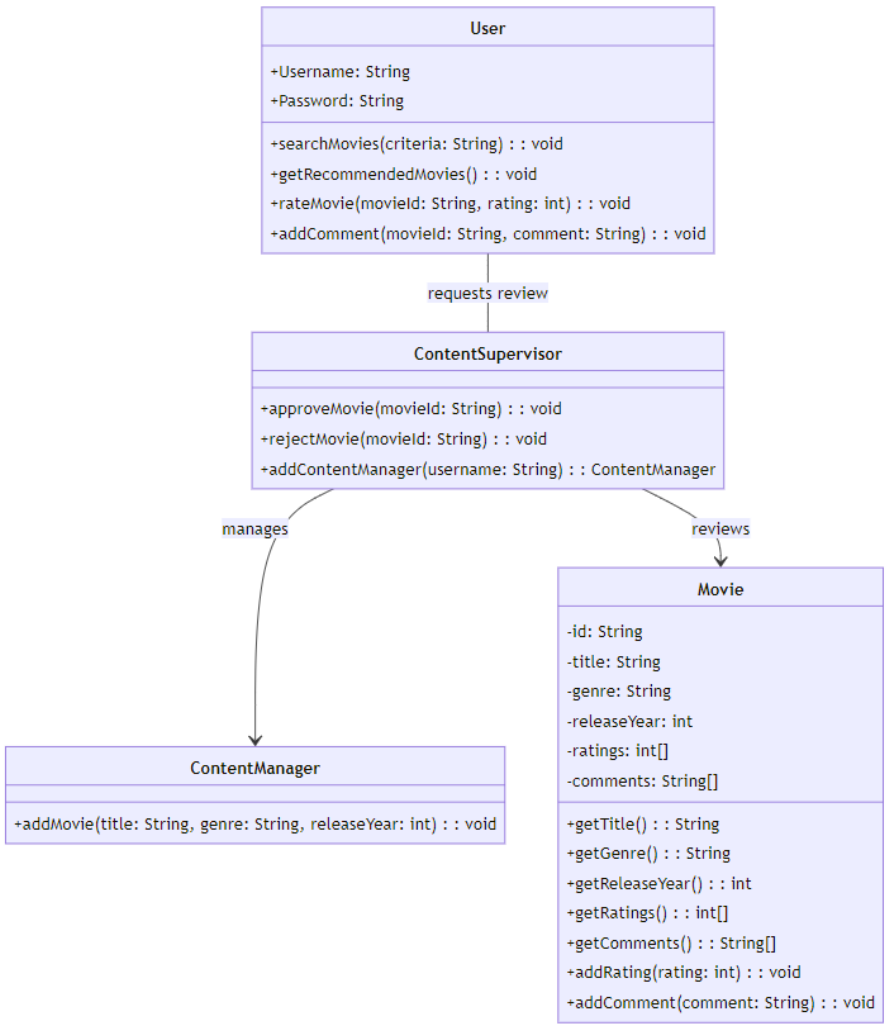

User
+Username: String
+Password: String
+searchMovies(criteria: String) : : void
+getRecommendedMovies() : : void
+rateMovie(movieId: String, rating: int) : : void
+addComment(movieId: String, comment: String) : : void
requests review
ContentSupervisor
+approveMovie(movieId: String) : : void
+rejectMovie(movieId: String) : : void
+addContentManager(username: String) : : ContentManager
manages
reviews
Movie
-id: String
-title: String
-genre: String
-releaseYear: int
-ratings: int[]
-comments: String[]
+getTitle() : : String
+getGenre() : : String
+getReleaseYear() : : int
+getRatings() : : int[]
+getComments() : : String[]
+addRating(rating: int) : : void
+addComment(comment: String) : : void
ContentManager
+addMovie(title: String, genre: String, releaseYear: int) : : void

Figure 4 Class Diagram of Movie Recommendation System

The class diagram captures the essential entities and their relationships within the movie recommendation system. The `User` class represents users with attributes such as user ID, names, email, and authentication details. Users can engage with the system by commenting (`Comment` class), expressing opinions and ratings for movies. The `Movie` class encapsulates movie details, including title, genre, director, and status. Users are associated with comments through a "makes" relationship, and movies are linked to comments through a "hasComments" relationship, establishing the core interactions in the system. The diagram highlights the connections between users, movies, and comments, providing a foundational understanding of the movie recommendation system's architecture.

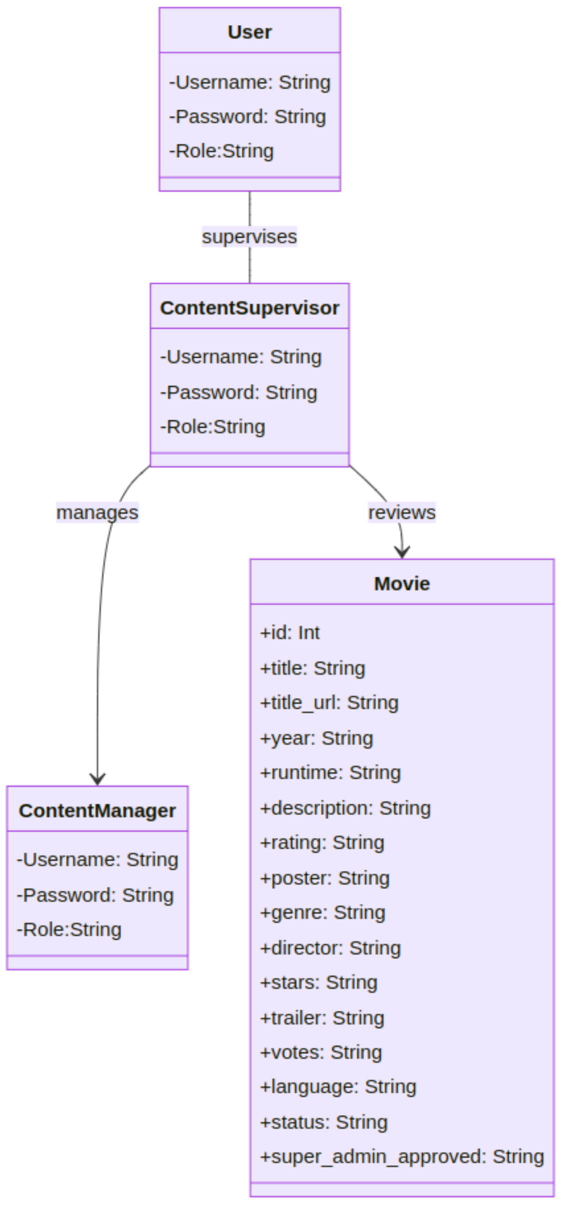


Figure 5 Object Diagram Movie Recommendation System

The object diagram provides a snapshot of instances within the movie recommendation system, focusing on the `User`, `Movie`, and `Comment` classes. Users (`user1`, `user2`) are associated with comments they make (`comment1`, `comment2`). Similarly, movies (`movie1`, `movie2`) are linked to comments they receive. To represent the connection between users, movies, and comments, an `Interaction` class is introduced, where instances (`interaction1`, `interaction2`) encapsulate specific interactions, showcasing relationships at a particular moment. This design allows for a dynamic representation of how users interact with movies through comments in the context of the movie recommendation system.

3.2 Dynamic Modelling using State and Sequence Diagrams

3.2.1 State Diagram

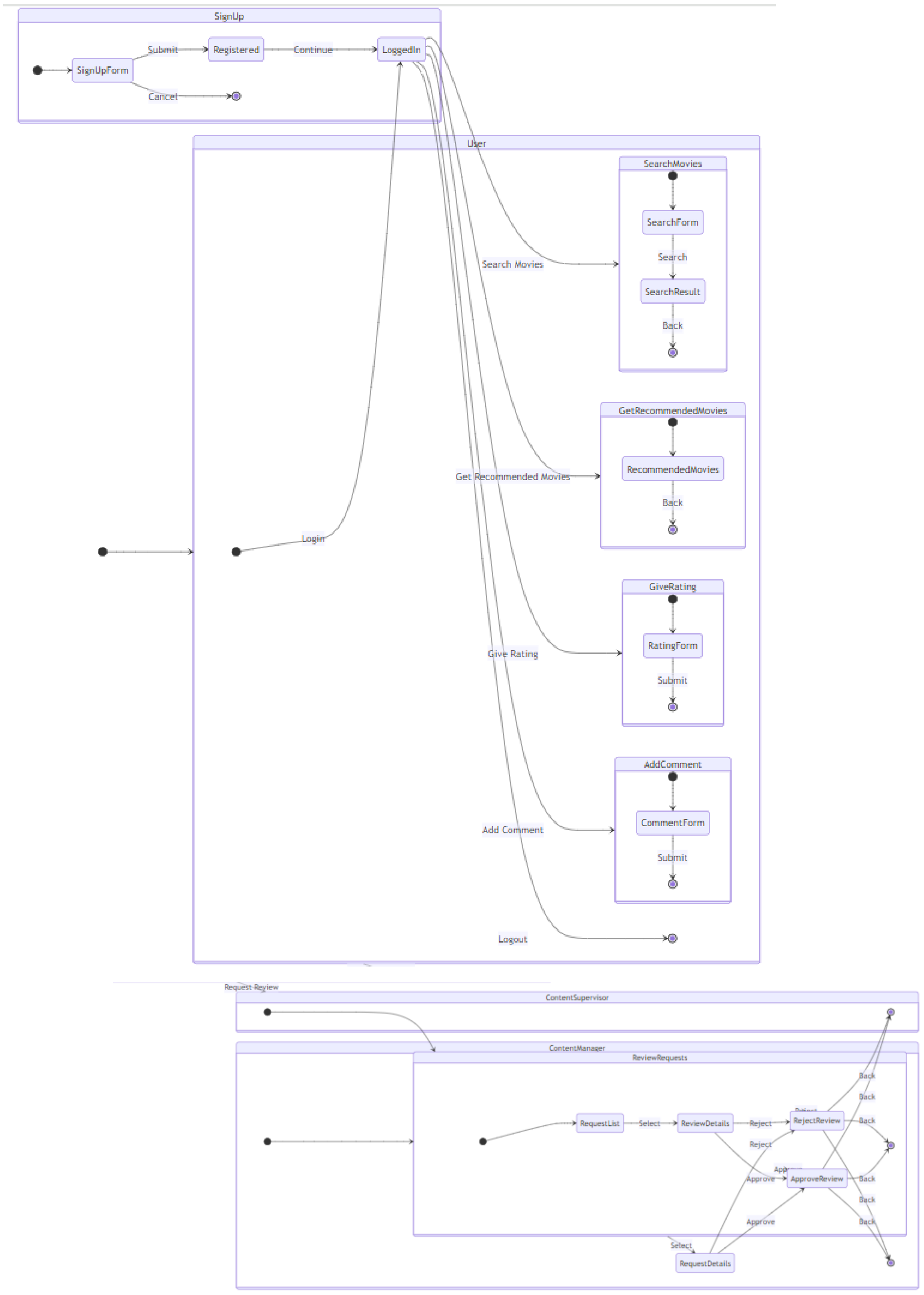


Figure 6 State Diagram of Movie Recommendation System

This state diagram captures the essential functionalities of a movie recommendation system involving three key entities: Users, Content Managers, and Content Supervisors. Users can seamlessly navigate through the system by logging in, searching for movies, receiving recommendations, giving ratings, and adding comments, concluding with a logout. Content Managers play a pivotal role in the system, initiating review requests for movies they wish to add. They can subsequently review the status of their requests, either receiving approvals or rejections from Content Supervisors. The Content Supervisor state revolves around managing these review requests, allowing for the approval or rejection of movies submitted by Content Managers. Additionally, the sign-up process is illustrated, providing a comprehensive overview of the system's dynamics and user interactions.

.

3.2.2 Sequence Diagram

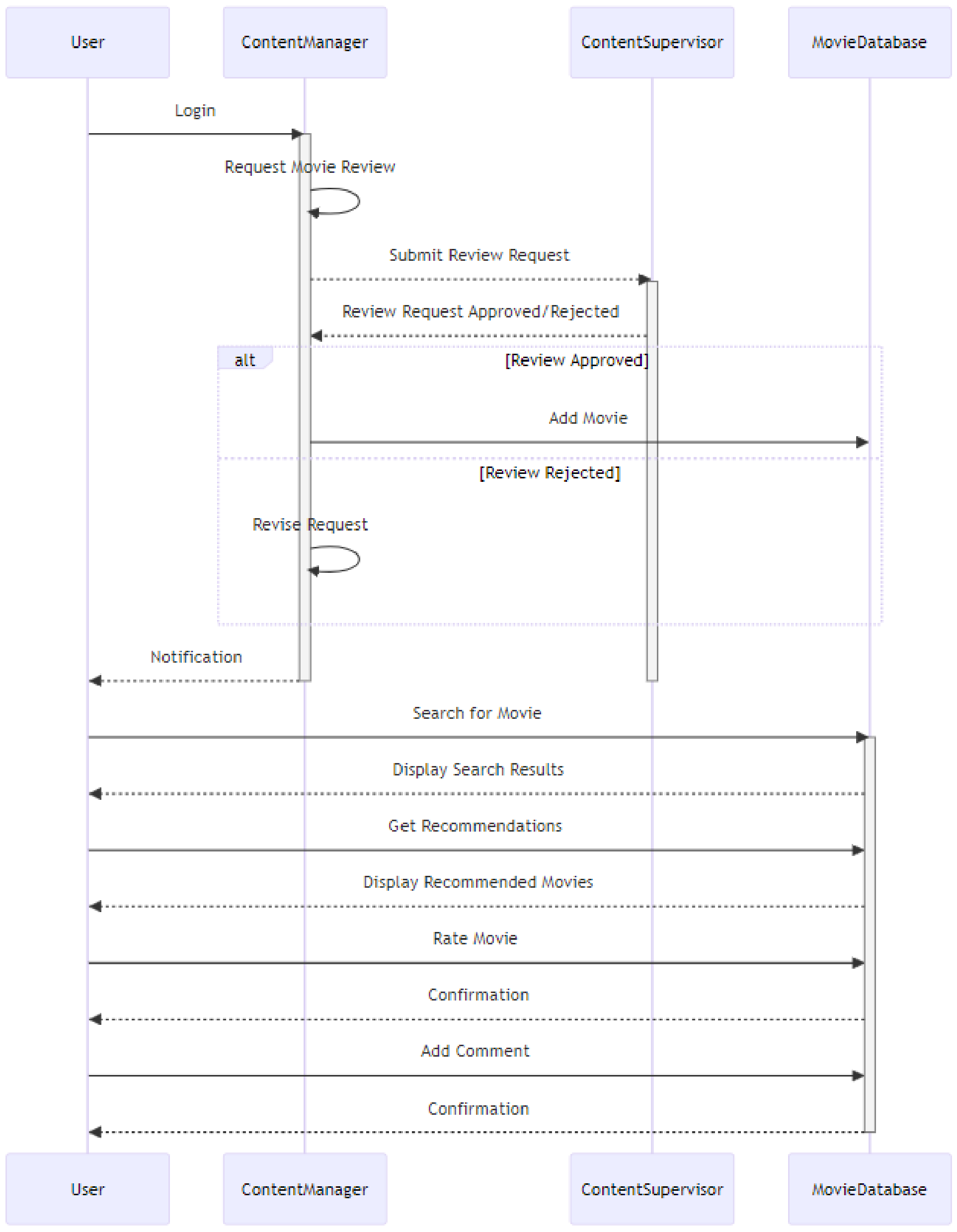


Figure 7 Sequence Diagram of Movie Recommendation System

This sequence diagram illustrates the interactions within a movie recommendation system. Users log in, allowing Content Managers to request movie reviews. Content Supervisors review these requests, and upon approval, Content Managers add the movie, notifying Users. Simultaneously, Users interact with the Movie Database by searching for movies, receiving recommendations, rating films, and adding comments. The diagram encapsulates the core dynamics, showcasing the seamless flow between administrative processes and user-driven functionalities in the system.

3.2.3 Interface design (UI/UX)

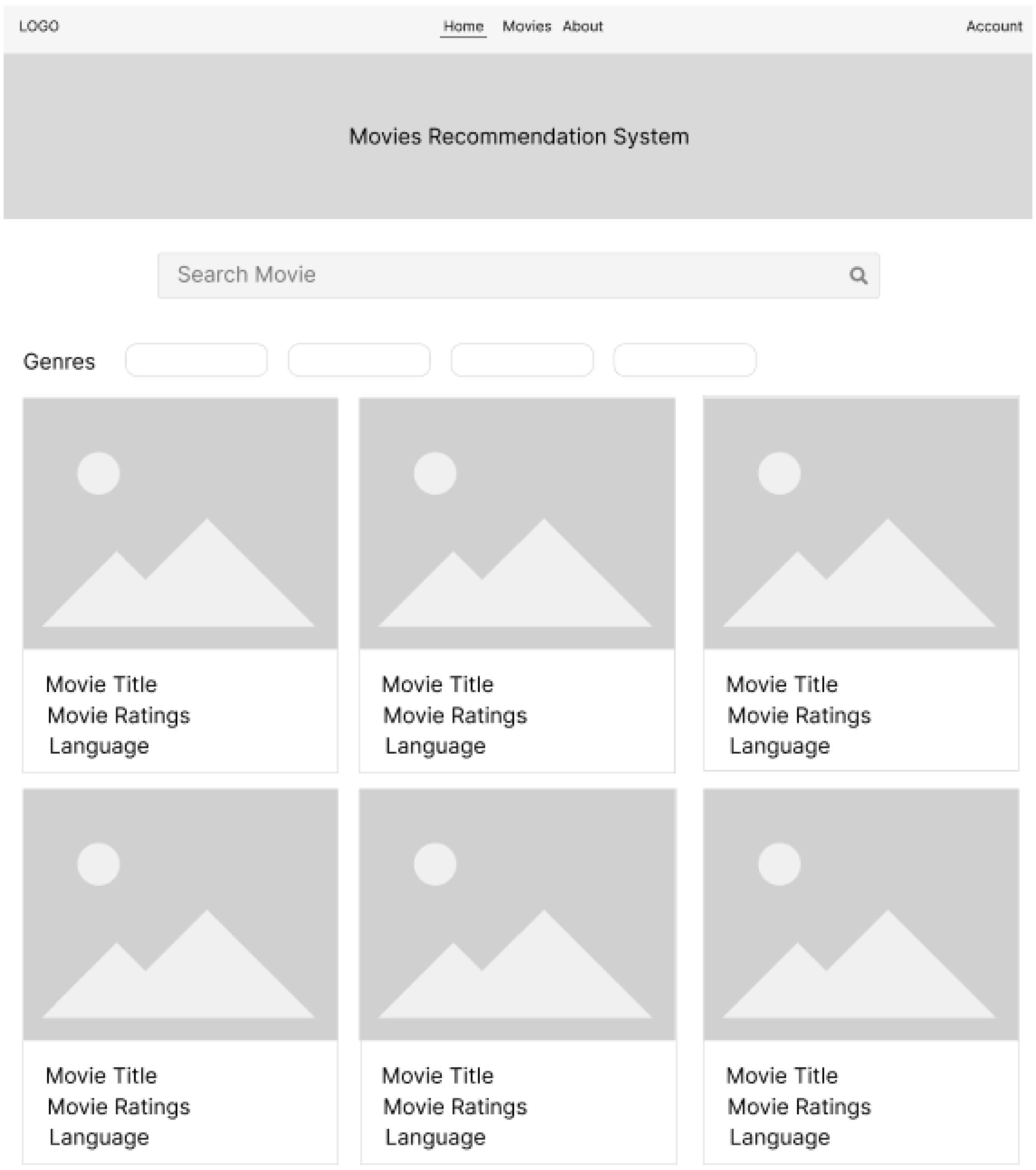


Figure 8 Home Page

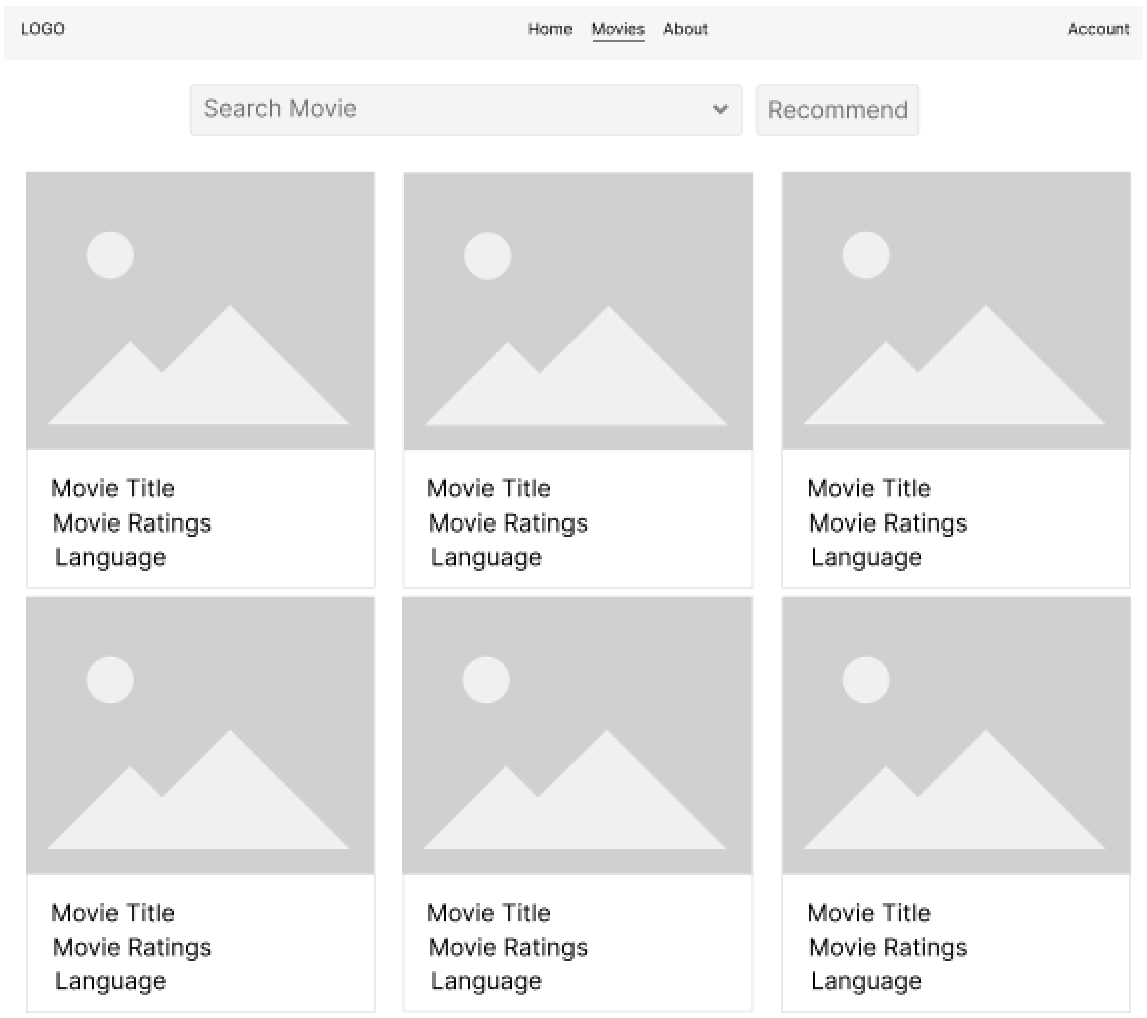
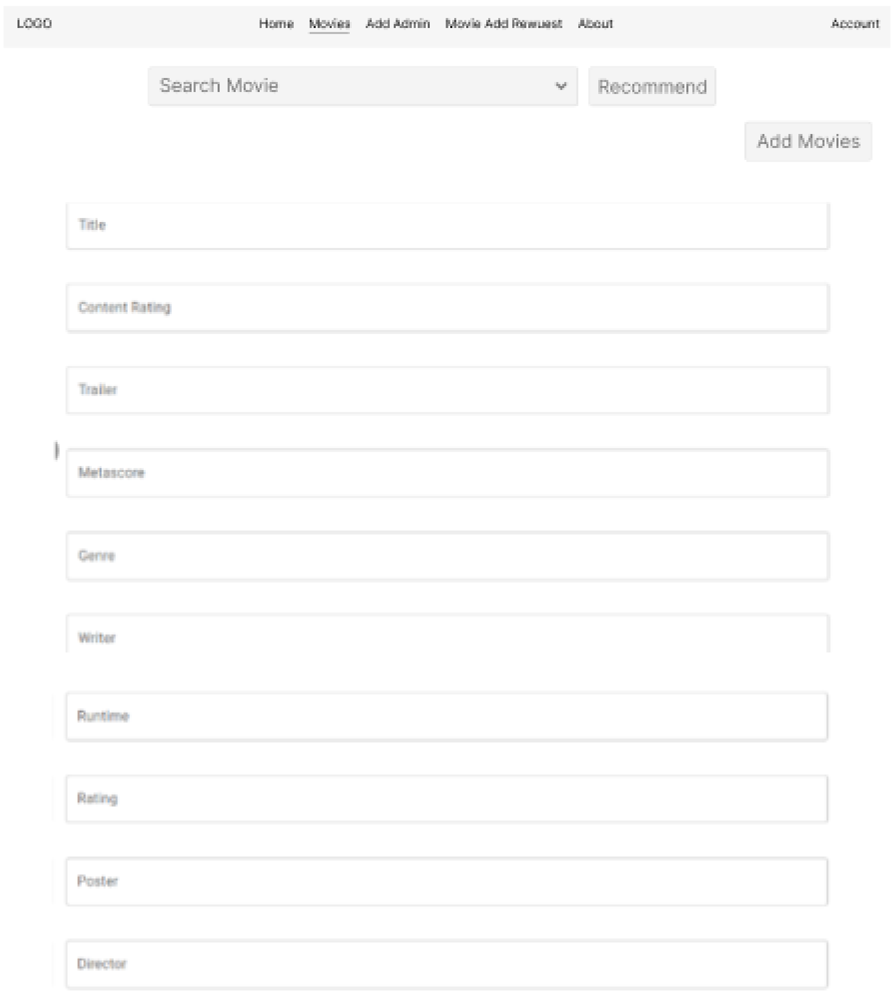


Figure 9 Movies Page



Title

Content Rating

Trailer

Metascore

Genre

Writer

Runtime

Rating

Poster

Director

Figure 10 Super Admin Movie page

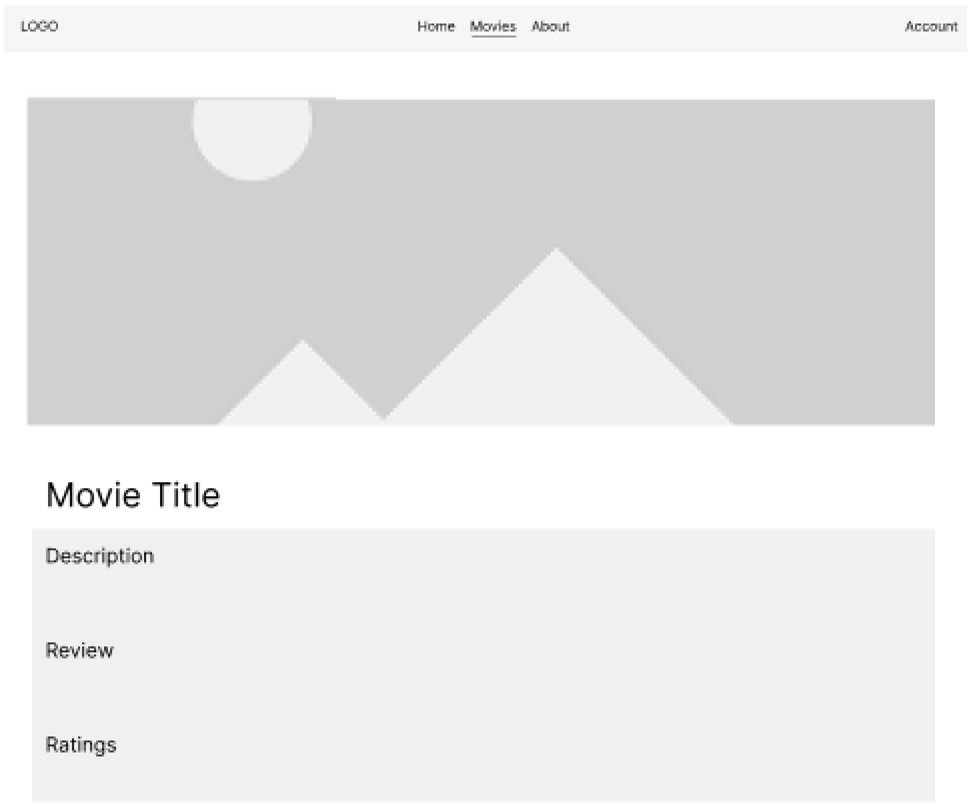


Figure 11 Movies Details Page

3.2.4 Process Modelling using Activity Diagrams

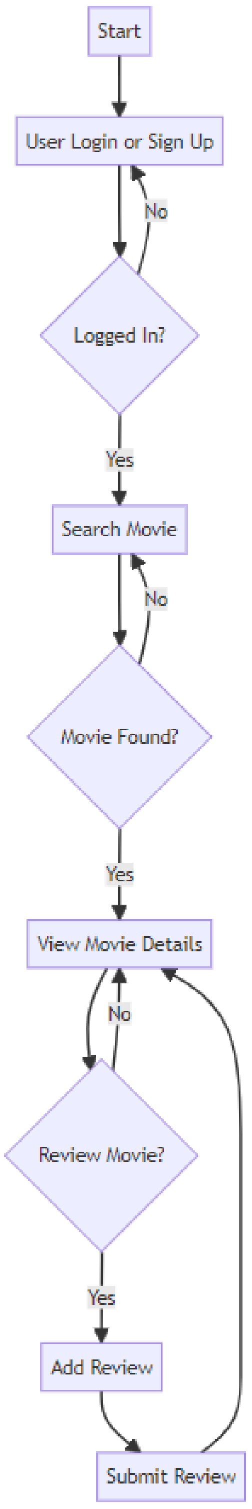


Figure 12 User Activity Diagram

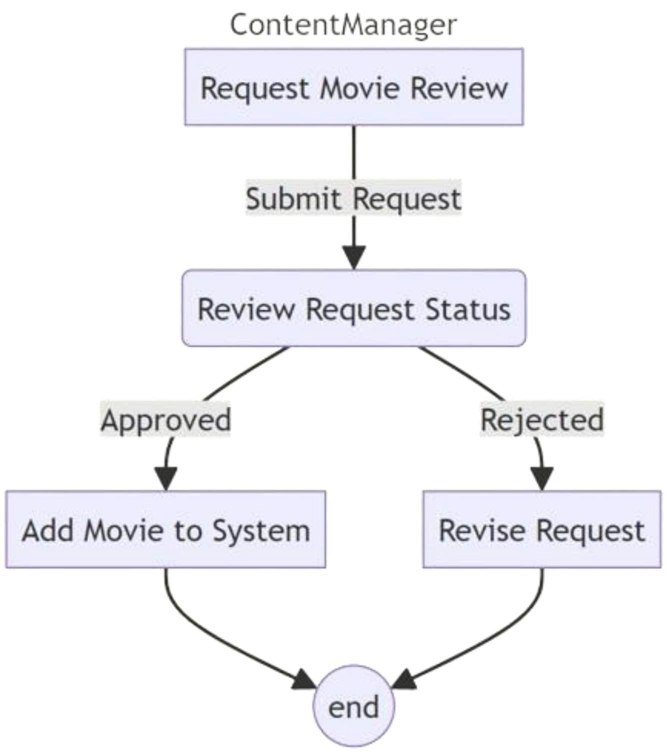


Figure 13 Content Manager Activity Diagram

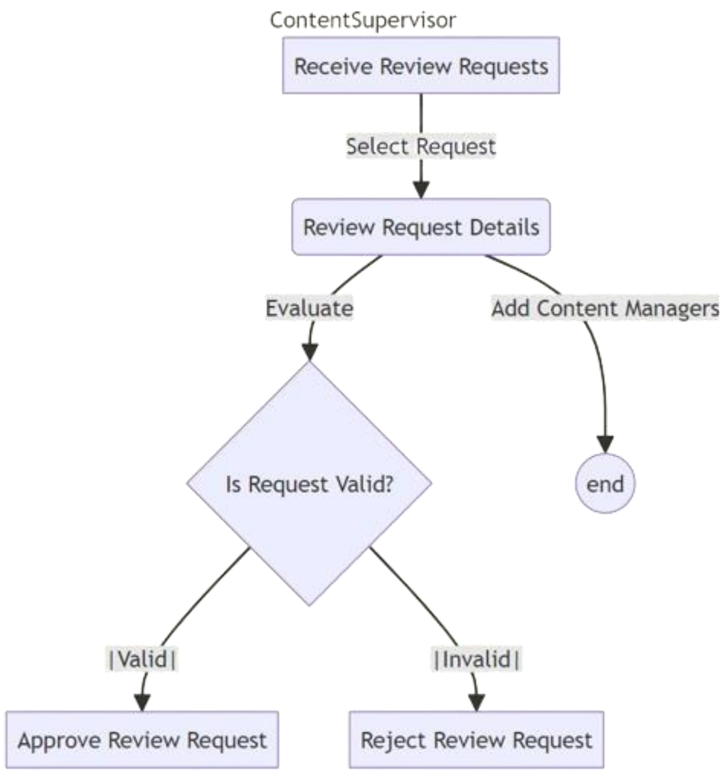


Figure 14 Content Supervisor Activity Diagram

The activity diagrams depict the sequential activities within a movie recommendation system involving Users, Content Managers, and Content Supervisors. In the User diagram, a user logs in, browses movies, receives recommendations, rates films, adds comments, and logs out. The Content Supervisor diagram illustrates the process of reviewing and managing movie review requests, with the ability to approve or reject requests and add content managers. The Content Manager diagram focuses on the manager's activities, including requesting movie reviews, reviewing the status of requests, adding approved movies, and revising requests if necessary. These diagrams offer a concise overview of the system's functionalities, highlighting the dynamic interactions between users and administrators.

3.3 Proposed Model

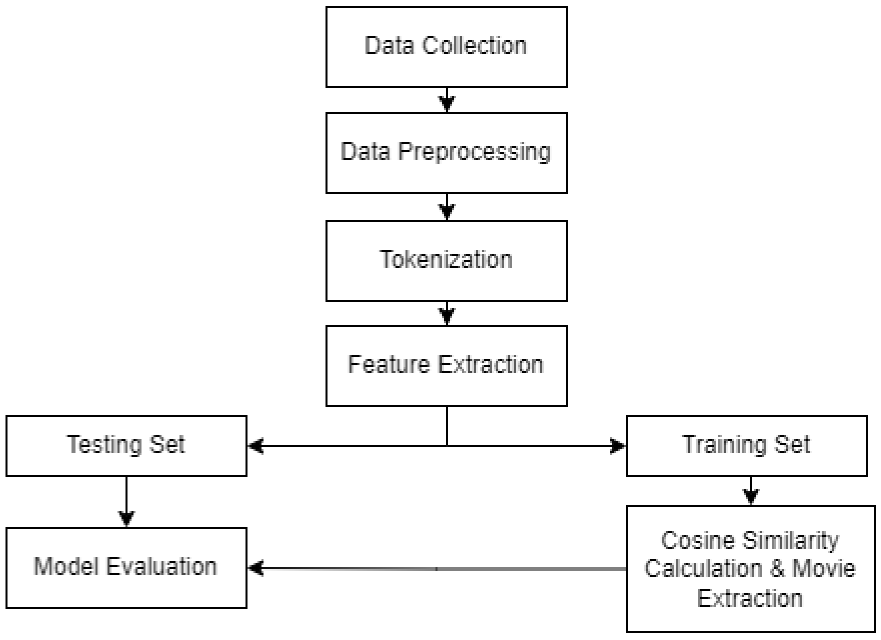


Figure 15 Proposed Model

3.4 Algorithm

1. Content-Based Filtering: Content-Based Filtering (CBF) is a machine learning approach used in recommendation systems to provide personalized recommendations based on the inherent features and characteristics of items. In the context of movie recommendations, CBF analyzes the content features of movies, such as genre and description, to suggest items that are similar in content.

   The CBF process can be outlined as follows:

a. Data Collection: Movie data, including features like title, genre, and description, etc is collected from a data source, such as imdb by web scraping using a powerful web scraping tool Octoparse and stored in a database. The data is preprocessed to create a unified representation of movie content.
b. Feature Combination: Relevant features, like genre and description, are combined into a single feature vector or text representation. For example, the 'content' feature may be a combination of genre and description.
c. Vectorization: The combined content feature is transformed into a numerical representation using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) vectorization. TF-IDF captures the importance of words within the content and creates a numerical matrix.

   Term Frequency (TF):

$$TF(t,d) = \frac{Numbers\ of\ times\ term\ t\ appears\ in\ document\ d}{Total\ number\ of\ terms\ in\ document\ d} \quad (1)$$

   Inverse Document Frequency (IDF):

$$\mathrm{IDF(t, D)} = log\ (\frac{Total\ number\ of\ documents\ in\ the\ corpus\ \ D}{Number\ of\ documents\ containing\ term\ t} \quad (2)$$

   TF-IDF:

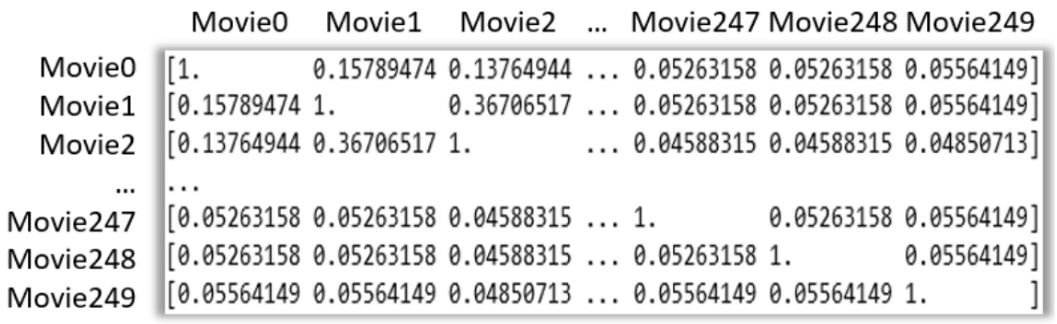
   TF-IDF(t,d,D) = TF(t,d) x IDF(t,D) (3)

d. Similarity Calculation: The cosine similarity metric is applied to compute the similarity between movies based on their TF-IDF vectors. The result is a similarity matrix where each element represents the similarity between two movies.

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{||\underset{a}{\rightarrow}||||\underset{b}{\rightarrow}||} = \frac{\sum_1 n a_i b_i}{\sqrt{\sum_1 n a_i^2}\sqrt{\sum_1 n b_i^2}} \quad (4)$$

   where, $\underset{a}{\rightarrow}.\underset{b}{\rightarrow}$ = $\sum_1 n\, a_i b_i$ = $a_i b_i$ + $a_2 b_2$ + … + $a_n b_n$ is the dot product of the vectors.

e. Recommendation Generation: Given an input movie title, the system identifies the corresponding row in the similarity matrix. Movies are sorted based on their similarity scores, and the top N movies (excluding the input movie) are recommended.



```
[1.         0.15789474 0.13764944 ... 0.05263158 0.05263158 0.05564149]
[0.15789474 1.         0.36706517 ... 0.05263158 0.05263158 0.05564149]
[0.13764944 0.36706517 1.         ... 0.04588315 0.04588315 0.04850713]
...
[0.05263158 0.05263158 0.04588315 ... 1.         0.05263158 0.05564149]
[0.05263158 0.05263158 0.04588315 ... 0.05263158 1.         0.05564149]
[0.05564149 0.05564149 0.04850713 ... 0.05564149 0.05564149 1.        ]
```

The recommendation generation step involves sorting movies based on their similarity scores and selecting the top N movies as recommendations. The similarity score between movies is typically calculated using the cosine similarity metric. The cosine similarity between two vectors A and B is computed using the formula:

$$\cos(\theta) = \frac{A.B}{||A||||B||} \qquad (5)$$

- A and B represent the TF-IDF vectors of two movies.
- A·B denotes the dot product of the two vectors.
- ‖A‖ and ‖B‖ represent the Euclidean norms of the vectors.

The cosine similarity ranges from -1 (completely dissimilar) to 1 (completely similar). The higher the cosine similarity, the more similar the movies are in terms of their content.

After calculating the cosine similarity scores for a given input movie against all other movies, the system identifies the corresponding row in the similarity matrix. Movies are then sorted based on their similarity scores in descending order. The top N movies, excluding the input movie itself, are recommended to the user.

Here's a high-level representation of the recommendation generation process:

- Calculate cosine similarity scores between the input movie and all other movies.
- Identify the row in the similarity matrix corresponding to the input movie.
- Sort movies based on their similarity scores in descending order.
- Exclude the input movie from the recommendations.
- Select the top N movies as the final recommendations.
- The value of N determines the number of recommendations to be provided to the user. The higher the N, the more recommendations will be presented to the user.

In summary, Content-Based Filtering leverages the content features of items (movies) to generate recommendations, making it suitable for scenarios where user-item interactions are not explicitly available. It is particularly effective for suggesting items with similar content characteristics, providing a form of personalized and context-aware recommendations to users.

3.5 Machine Learning

1. Data Collection:

   For the movie recommendation system, I have collected data from imdb datasets. To extract the movie data, I used Octoparse to scrape the data. The data was cleaned and the data labels are modified according to the requirements of the system. Different language movies are incorporated.

S.N.	Language	No. of movies
1.	Nepali	5000
2.	Hindi	11000
3.	English	15000

Along with this the admin has the authority to add more movies.

2. Data Preprocessing:

   a. Remove newline characters, brackets and extra space:

      Removes leading and trailing whitespaces from the columns. It ensures that there are no unnecessary spaces that might affect downstream processing. Also, removes parentheses from the column using a regular expression. It is useful if 'year' values are enclosed in brackets, ensuring that only the numeric part remains.

      Consider a DataFrame with the following sample data:

```
    genre        description   year
0   Action   A thrilling movie.   (1990)
1    Drama    Emotional drama.     (2005)
2   Romance         Love story.    (1982)
```

      After applying the special character removal process, the DataFrame is transformed as follows:

```
   genre       description  year
0  Action  A thrilling movie.  1990
1   Drama   Emotional drama.   2005
2 Romance        Love story.   1982
```

b. Split the "movie_cast" column into "Director" and "Stars":

   Splitting the column using the pipe ('|') as a delimiter and creates two new columns, containing the separated values. It assumes that 'movie_cast' has entries like "Director | Stars."

   Consider a DataFrame with the following sample data:

```
                    movie_cast
0  Director1 | Star1, Star2
1  Director2 | Star3, Star4
2  Director3 | Star5, Star6
```

   After applying the splitting process, the DataFrame is transformed as follows:

```
        movie_cast    director         stars
0  Director1 | Star1, Star2  Director1  Star1, Star2
1  Director2 | Star3, Star4  Director2  Star3, Star4
2  Director3 | Star5, Star6  Director3  Star5, Star6
```

3. Tokenization

   Tokenization is the process of breaking down text into individual words or tokens. This line combines the text from the 'title', 'genre', 'director', and 'stars' columns into a new column called 'combined_features'. It uses the fillna('') method to handle missing values, ensuring that the concatenation process doesn't break when there are NaN values in any of the columns.

   Consider a DataFrame with the following sample data:

```
  title  genre     director         stars
0  Movie1  Action  Director1  Star1, Star2
1  Movie2  Drama   Director2  Star3, Star4
2  Movie3  Comedy  Director3  Star5, Star6
```

   After applying the tokenization process, the DataFrame is transformed as follows:

```
  title  genre     director         stars               combined_features
0  Movie1  Action  Director1  Star1, Star2  Movie1 Action Director1 Star1, Star2
1  Movie2  Drama   Director2  Star3, Star4  Movie2 Drama Director2 Star3, Star4
2  Movie3  Comedy  Director3  Star5, Star6  Movie3 Comedy Director3 Star5, Star6
```

   This 'combined_features' column contains the concatenated text from the specified columns, creating a unified representation of relevant information for each movie.

4. Vectorization:

   Vectorization is the process of converting text data into numerical vectors that can be used as input for machine learning algorithms.This code uses the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer from scikit-learn. It converts the text in 'combined_features' into a sparse matrix of TF-IDF features. This matrix represents the importance of each word (term) in the context of the entire dataset.

   ```
   # Create a TF-IDF Vectorizer
   tfidf_vectorizer = TfidfVectorizer()
   # Fit and transform the vectorizer on the combined features
   tfidf_matrix = tfidf_vectorizer.fit_transform(df['combined_features'])
   ```

$$TF(t,d) = \frac{Numbers\ of\ times\ term\ t\ appears\ in\ document\ d}{Total\ number\ of\ terms\ in\ document\ d} \qquad (6)$$

5. Cosine Similarity:

   Cosine similarity is a measure of similarity between two non-zero vectors in an inner product space. In the context of your movie recommendation system, it's used to determine how similar two movies are based on their content represented as vectors. Cosine similarity is particularly useful for text-based data and is a common technique for measuring text similarity.

   Importing the Necessary Library:

   ```
   from sklearn.metrics.pairwise import cosine_similarity
   ```

   In this code, you import the cosine_similarity function from scikit-learn's metrics.pairwise module. This function allows you to compute cosine similarity between vectors efficiently.

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{||\underset{a}{\rightarrow}||||\underset{b}{\rightarrow}||} = \frac{\sum_1 n a_i b_i}{\sqrt{\sum_1 n a_i^2}\sqrt{\sum_1 n b_i^2}} \qquad (7)$$

   Where, $\underset{a}{\rightarrow}.\underset{b}{\rightarrow} = \sum_1 n\, a_i b_i = a_i b_i + a_2 b_2 + \ldots + a_n b_n$ is the dot product of the vectors.

   Calculating Cosine Similarity:

   ```
   similarity = cosine_similarity(vectors)
   ```

   Here's what this code snippet does:

   vectors is a matrix where each row represents a movie, and each column represents a term (word or token) from the 'tags' column. Each element in the matrix corresponds to the frequency or presence of a term in a movie's 'tags.'

cosine_similarity(vectors) takes this matrix as input and computes the pairwise cosine similarity between all rows (movies) in the matrix. The result is a similarity matrix where similarity[i][j] represents the cosine similarity between movie 'i' and movie 'j.'

Interpreting the Cosine Similarity Matrix:

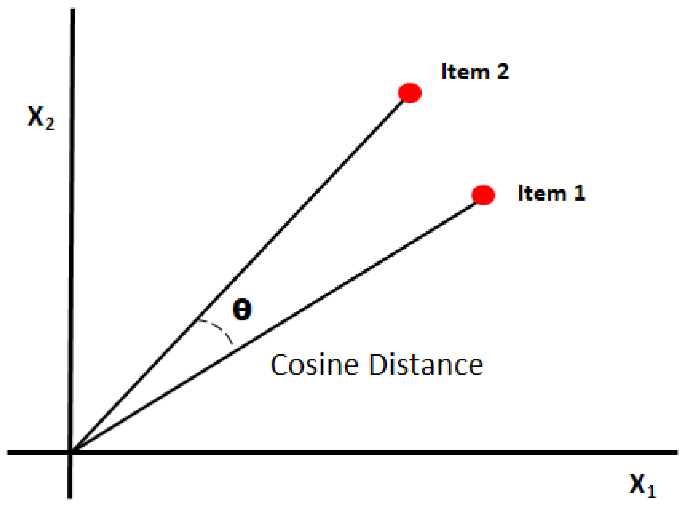


Figure 16 Cosine Similarity

The similarity matrix is a square matrix where each row and column correspond to a movie in your dataset.

Cosine similarity values range between -1 and 1.

1 indicates that two movies are perfectly similar in content.

0 indicates no similarity (orthogonal vectors).

-1 indicates perfect dissimilarity (opposite directions).

The diagonal elements (where i = j) in the matrix will have a cosine similarity of 1 since a movie is perfectly similar to itself.

How to Use Cosine Similarity for Recommendations?

To make movie recommendations, you typically follow these steps:

Given a movie as input, find its index in the similarity matrix.

Retrieve the row corresponding to the input movie, which contains similarity scores for all other movies.

Sort the similarity scores in descending order to identify the most similar movies.

Recommend the top N movies with the highest similarity scores.

Your code may include a recommendation function that performs these steps, allowing users to input a movie title and receive a list of recommended movies based on their similarity to the input movie.

In summary, cosine similarity is a mathematical measure that quantifies how similar two movies are based on their content features. It's a fundamental concept in content-based recommendation systems, helping to identify movies that share similar content characteristics with a given movie.

6. Recommendation Function:

   Finally, you define a 'recommend' function to recommend movies based on a given movie's title. This function takes a movie title as input, finds the index of that movie in the dataset, calculates the cosine similarity with other movies, and returns a list of recommended movies based on the highest similarity scores.

3.6 System Design

3.6.1 Refinement of class diagram, object, state, sequence and activity

3.6.1.1 Refinement of class diagram

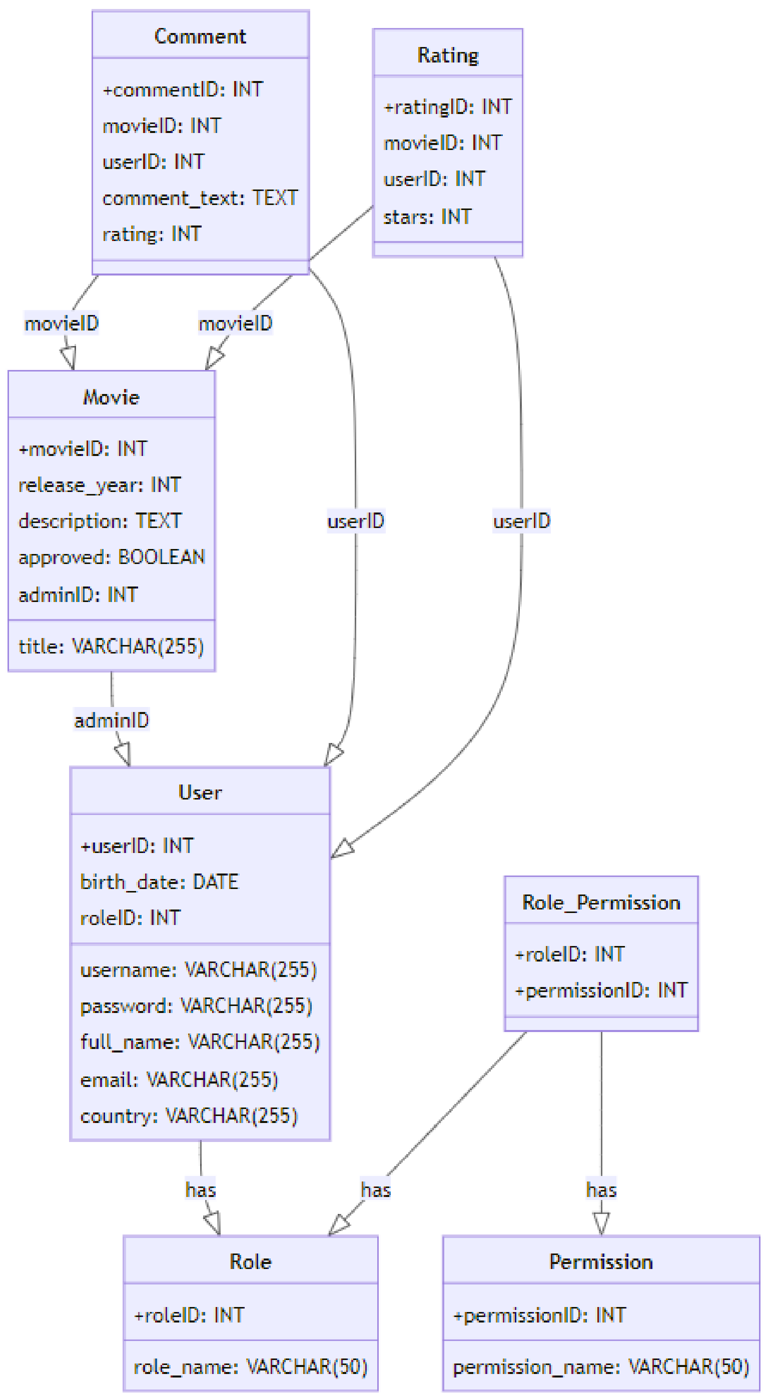


Figure 17 Refinement of class diagram

3.6.1.2 Refinement of object diagram

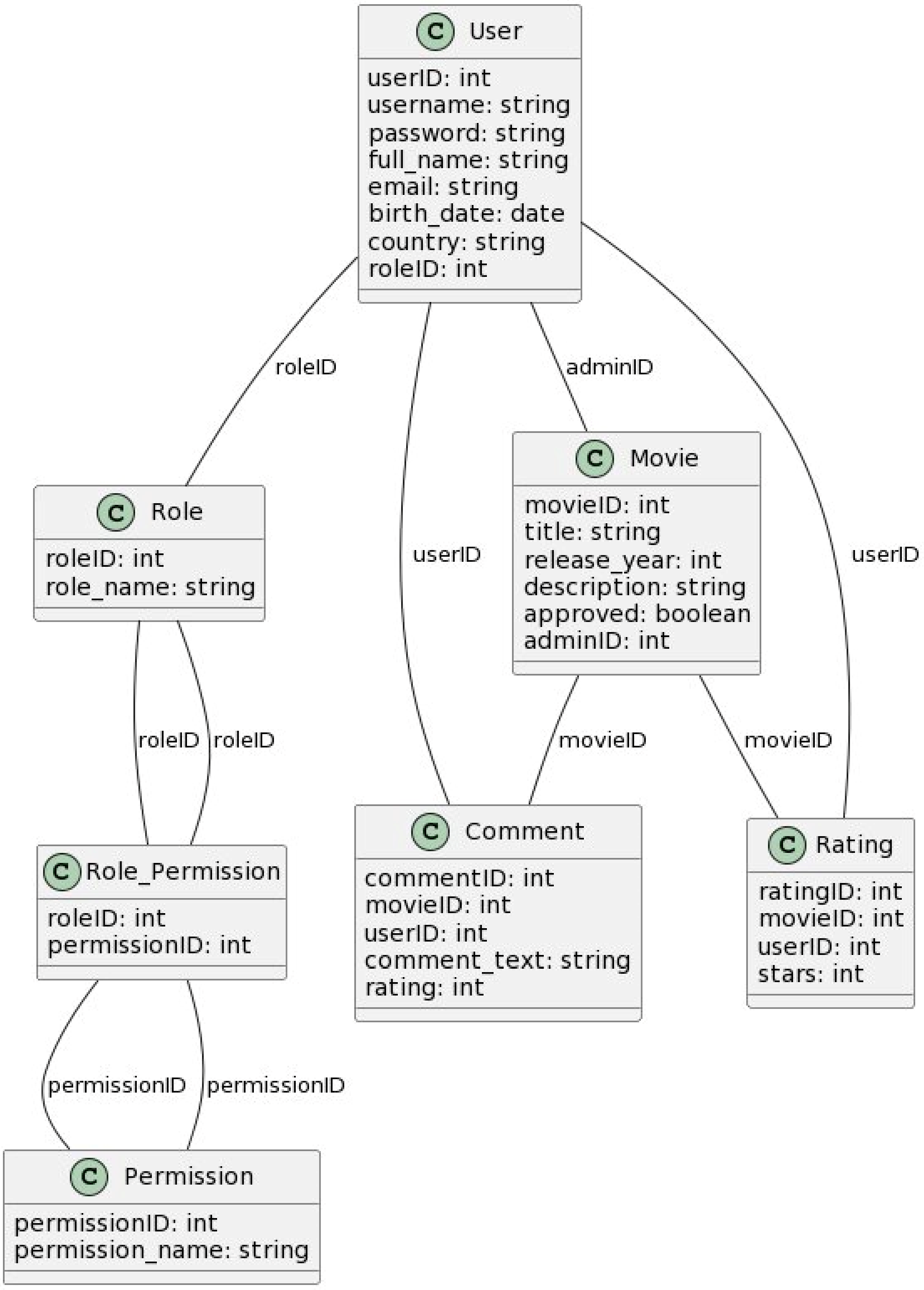


Figure 18 Refinement of object diagram

3.6.1.3 Refinement of state diagram

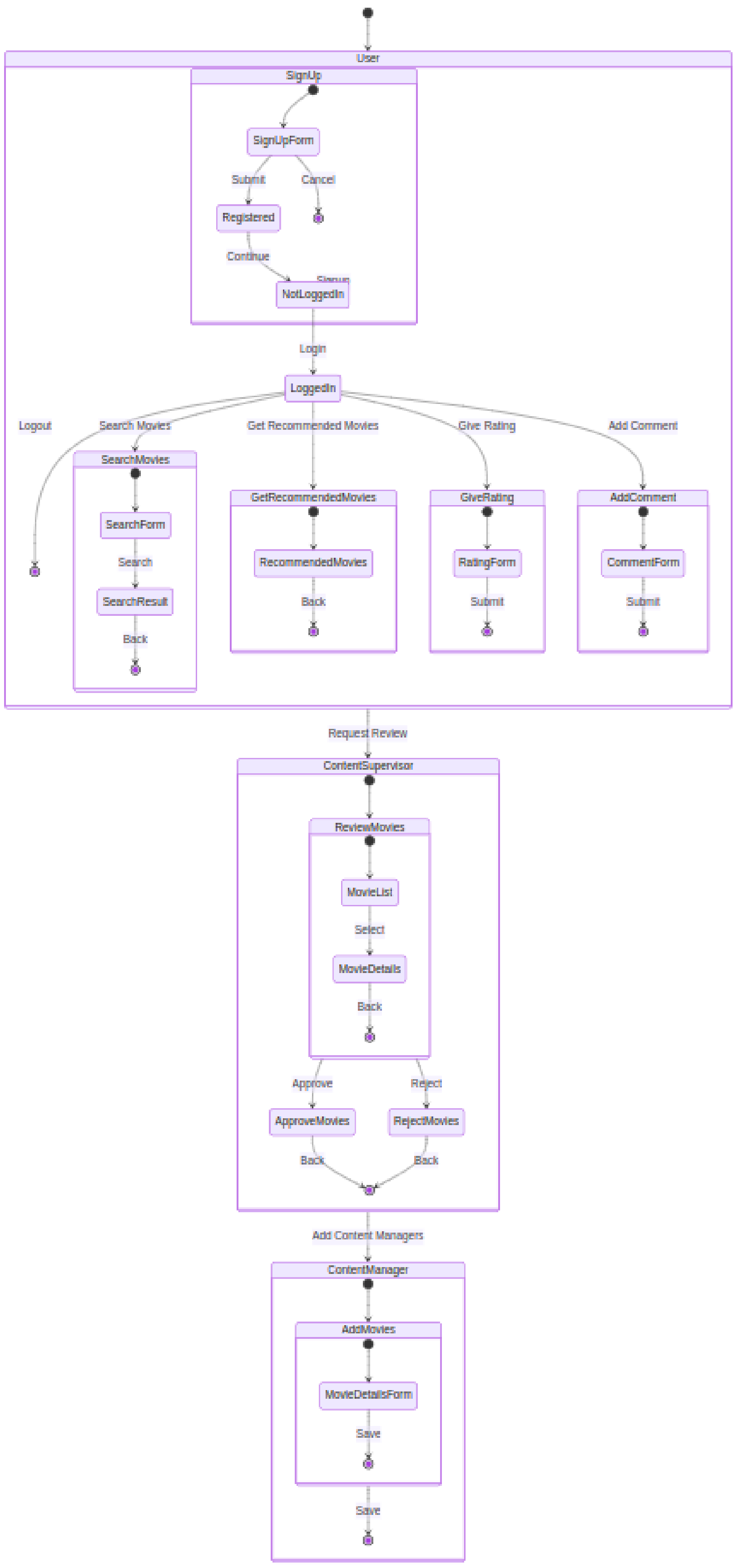


Figure 19 Refinement of state diagram

3.6.1.4 Refinement of sequence diagram

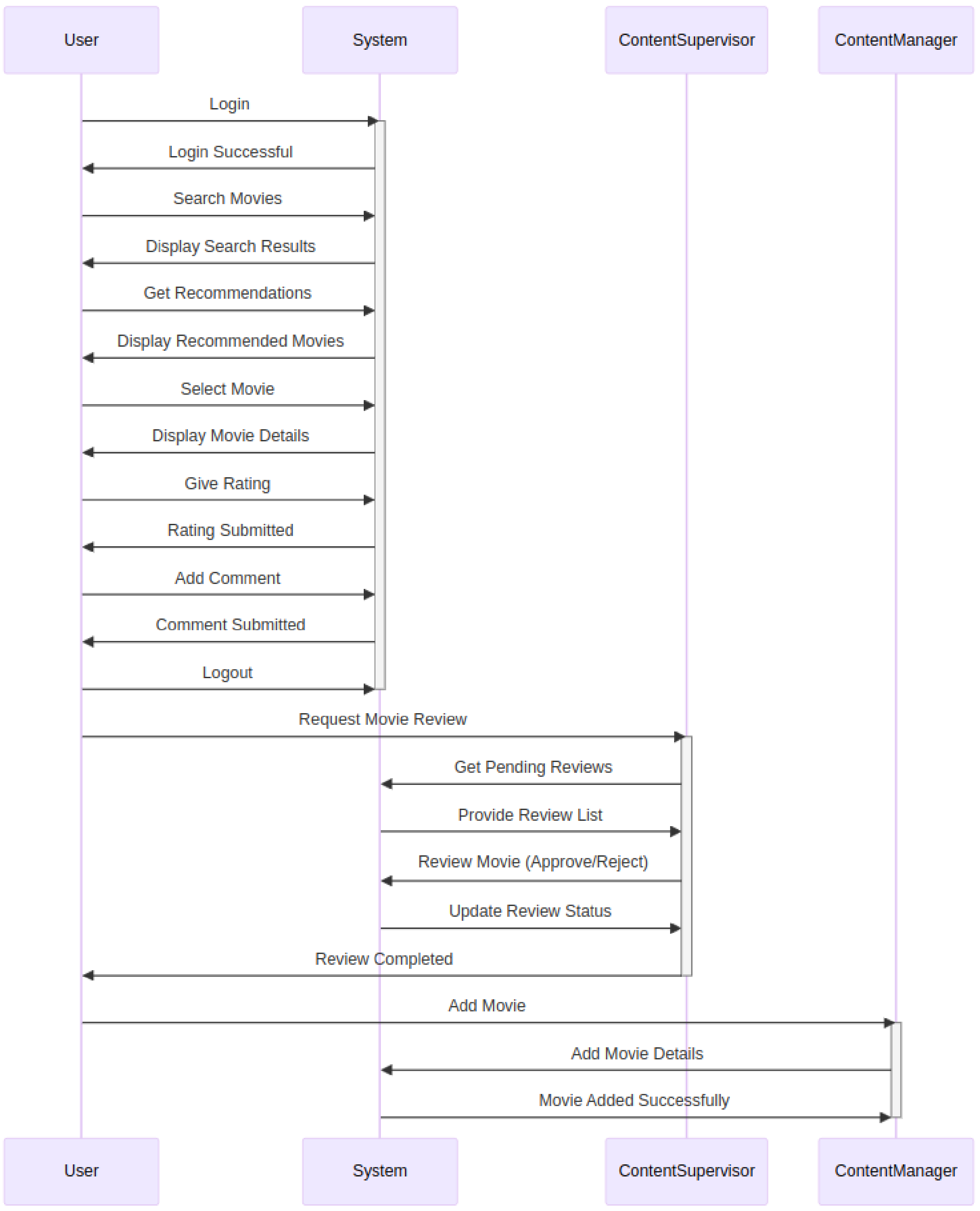


Figure 20 Refinement of sequence diagram

3.6.1.5 Refinement of activity diagram

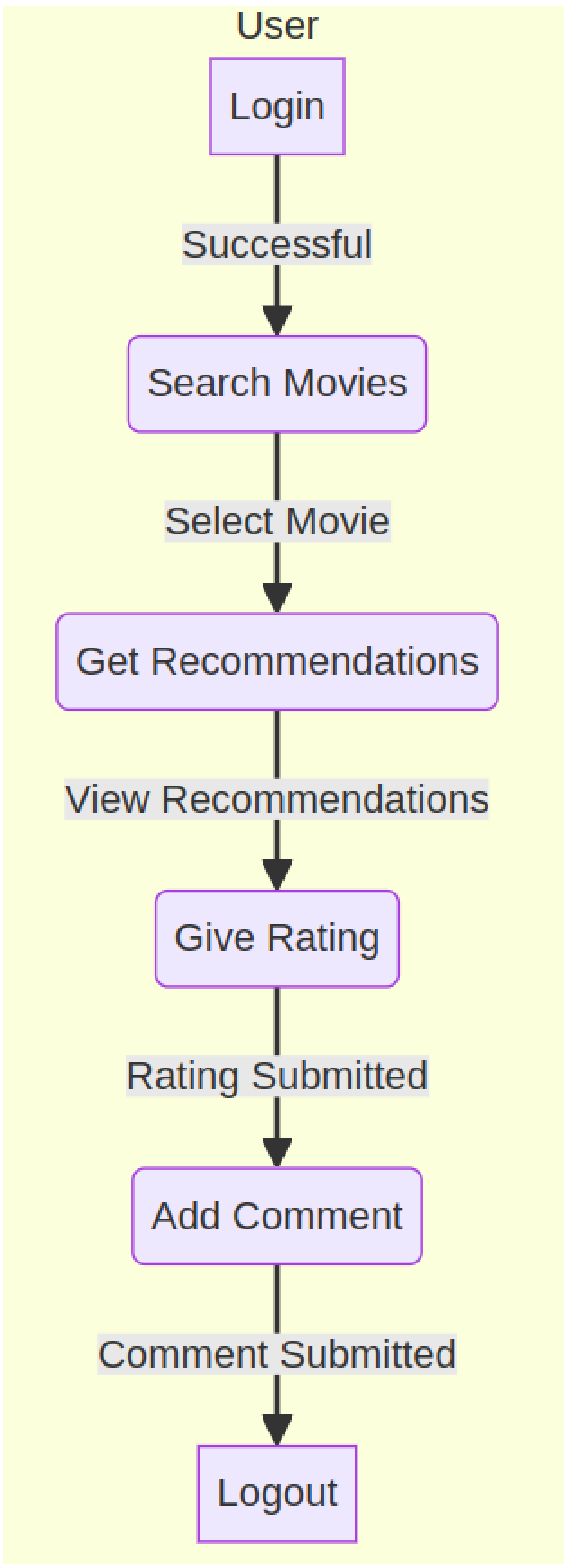


Figure 21 Refinement of User activity diagram

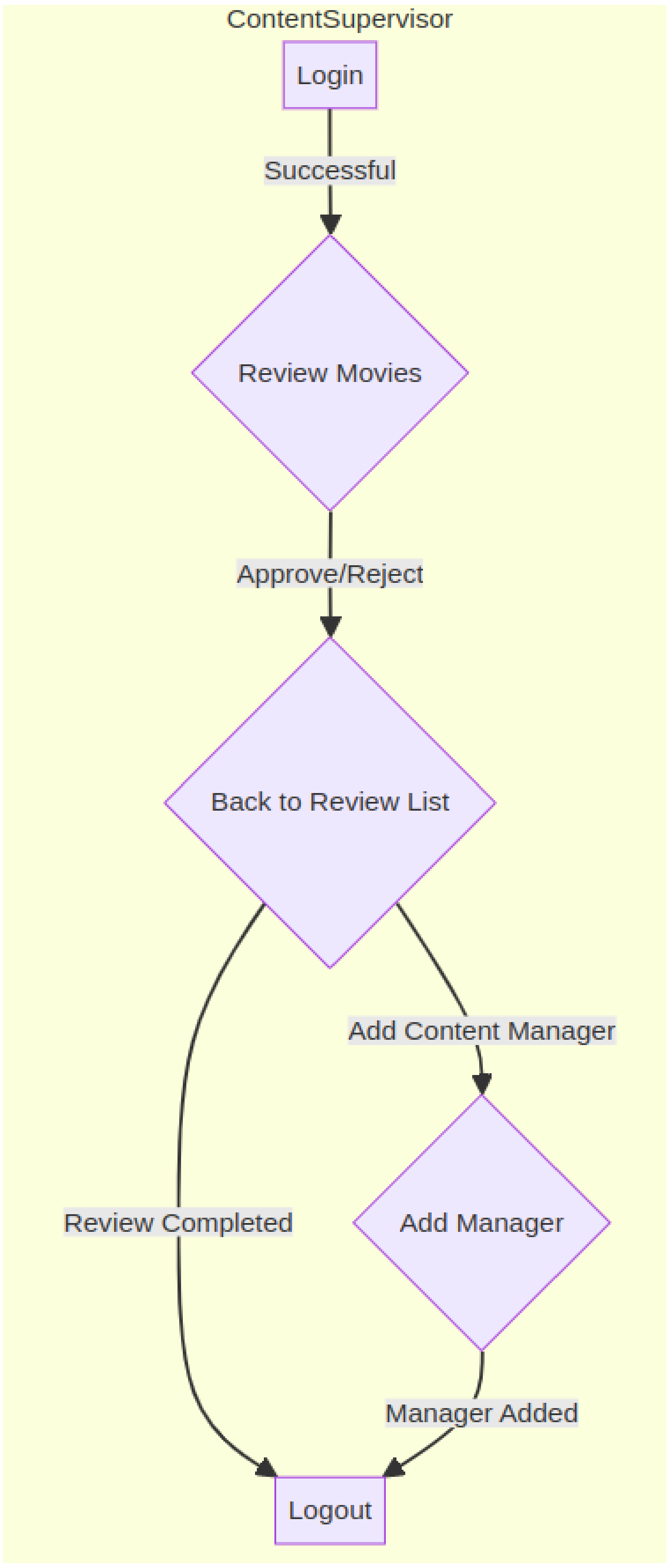


Figure 22 Refinement of Content Supervisor activity diagram

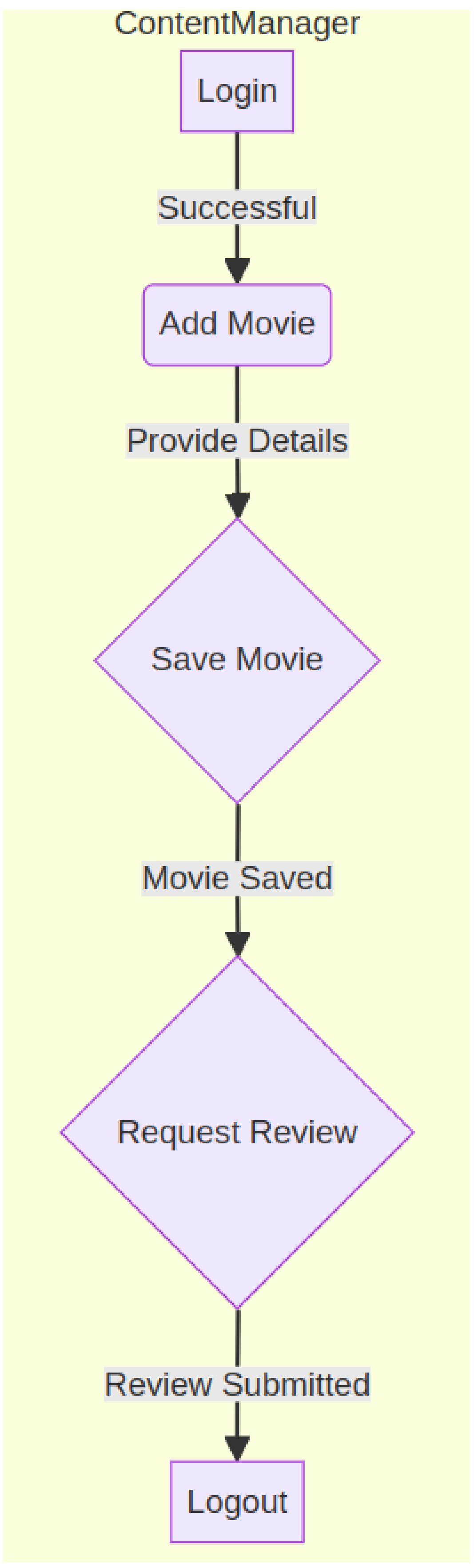
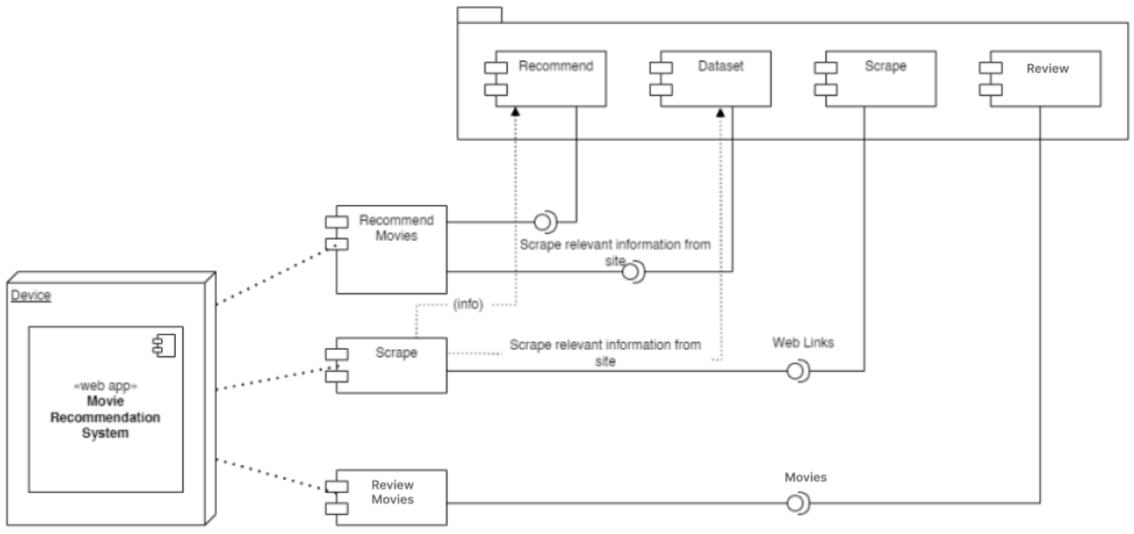


Figure 23 Refinement of Content Manager activity diagram

3.6.2 Component Diagram



Recommend
Dataset
Scrape
Review
Recommend Movies
Scrape relevant information from site
Device
«web app»
Movie Recommendation System
(info)
Scrape
Scrape relevant information from site
Web Links
Review Movies
Movies

Figure 24 Component Diagram of Movie Recommendation System

A component diagram is a type of UML diagram that shows the structural relationships and dependencies between the components of a software system. It illustrates how software components are connected and interact with each other within a system. Components can represent individual modules, libraries, executables, or other parts of a system.

3.6.3 Deployment Diagram

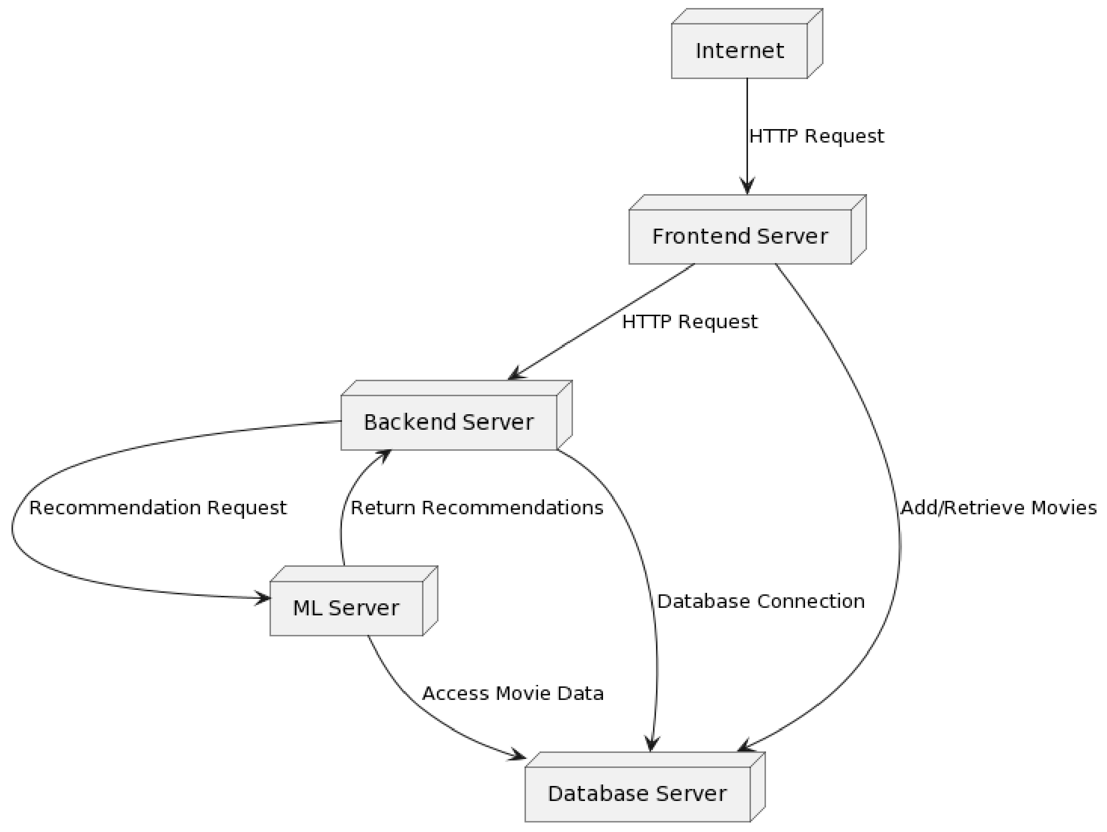


Figure 25 Deployment Diagram of Movie Recommendation System

Deployment diagram are UML structural diagrams that shows the relationships between the hardware and software components in the system and the physical distribution of the processing i.e., Deployment diagram are used to visualize the topology of the physical components of the system where software components are deployed.

Chapter 4: Implementation and Testing

4.1 Implementation

4.1.1 Tools used (CASE tools, programming languages, database platforms)

Table 1 Tools and its purposes

Tools and Programming languages	**Purposes**
React js.	For Frontend development
Python, Nest js.	For Backend development
Mysql	Database Management
Visual Studio code	For coding and development
Git and GitHub	For version controlling

Along with it other tools and libraries used for implementing machine learning algorithm are as follows:

- Python libraries: For the computation and analysis we need certain python libraries which are used to perform analytics. Packages such as SKlearn, Numpy, pandas, Matplotlib, Flask framework, etc are needed.
- SKlearn: It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
- NumPy: NumPy is a general-purpose array-processing package. It provides a highperformance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.
- Pandas: Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools. Unlike NumPy library which provides objects for multi-dimensional arrays, Pandas provides in-memory 2d table object called Data frame.

- Flask: It is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug.

4.1.2 Implementation details of modules

This section elaborates on how the modules in the system are implemented and what functions do they contribute to the system.

1. User Authentication:

   a Sign Up: Users can create a new account by providing their username, email, password, and security question answer. Upon clicking the "Create" button, their account is registered.

   b Login: Registered users can log in by entering their username and password. If the provided credentials match the database, the user is authenticated and directed to the dashboard.

2. Movie Search and Recommendations:

   a Search Movies: After logging in, users can search for movies using various criteria such as movie name, genres, or cast members' names.

   b Movie Recommendations: Based on user search, collaborative filtering and cosine similarity algorithms provide personalized movie recommendations to the users.

3. User Dashboard:

   a Dashboard: Upon logging in, users are directed to their personalized dashboard, where they can view their account information, movie search, and recommended movies, movie filtered from genre

4. Content Management:

   a. Content Supervisor Login: Content supervisors can log in with their credentials to access administrative functionalities, including the addition of content managers and reviewing movie additions.

   b. Content Manager Login: Content managers can log in to contribute movie content and manage their activities within the system.

   c. Add Content Manager: Content supervisors have the authority to add content managers to the system, granting them the ability to contribute movie content.

d. Review Movie Addition: Content managers can add movies to the system. The added movies undergo a review process by content supervisors, who have the authority to approve or reject the additions.

5. Log Out:
   a Logout Option: Users can log out of the system to end their session and protect their account privacy.
6. Data Management:
   a Database Management: User account information are stored in a MySQL database for efficient data retrieval and management.
7. User Interaction:
   a Intuitive Interface: The system provides a user-friendly interface, enabling seamless interactions between users and the movie recommendation features.
8. Continuous Improvement:
   a Algorithm Enhancement: To enhance movie recommendations and review analysis, the system is continuously updated with new movie data and user feedback. Regular updates ensure the system remains relevant and accurate in its suggestions.

4.2 Testing

4.2.1 Test cases for Unit Testing

Unit testing is the first level of testing and is often performed by the developers themselves. It is the process of ensuring individual components of a piece of software at the code level are functional and work as they were designed. It validates real-world scenarios, identifies issues, and ensures user satisfaction before deployment. It is the process of taking a module and running it in isolation from rest of the software product by using prepared test cases and comparing the actual result with the result redirected with the specifications and design of the module.

Table 2 Sign up

Serial No.	Description	Expected Result	Actual Result	Result
1.	Sign up	User information should be saved.	User information is saved in the database.	Passed
2.	Empty fields	Dialogue box should be shown saying, “Please fill the form”	Dialogue box is shown saying, “Please fill the form”	Passed

Table 3 Login

Serial No.	Description	Expected Result	Actual Result	Result
1.	Login	User should be redirected to dashboard after authentication.	User is redirected to dashboard after authentication.	Passed
2.	Empty fields	Dialogue box should be shown saying, “Please fill the form”	Dialogue box is shown saying, “Please fill the form”	Passed
3.	Incorrect username password	Snackbar should be shown saying, “Incorrect credentials”	Snackbar is shown saying, “Incorrect credentials”	Passed

Table 4 Movie Search

Serial No.	Description	Expected Result	Actual Result	Result
1.	Test the functionality of movie search based on movie name, genres, or cast members' names.	Relevant movie search results are displayed.	User enters valid movie name.	Passed

Table 5 Movie Recommendations

Serial No.	Description	Expected Result	Actual Result	Result
1.	Test the accuracy of personalized movie recommendations using cosine similarity algorithms.	Relevant and personalized movie recommendations are provided to the user.	User search are considered for recommendation.	Passed

Table 6 Genre wise filtering

Serial No.	Description	Expected Result	Actual Result	Result
1.	Genre wise filtering.	The user can filter the data by clicking on the genre and the selected genres movies will be displayed.	The user can filter the data by clicking on the genre and the selected genres movies will be displayed.	Passed

Table 7 Logout

Serial No.	Description	Expected Result	Actual Result	Result
1.	Logout	Dashboard should close and login page should open.	Dashboard will close and login page will open.	Passed

4.2.2 Test cases for System Testing

System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specification. The system is integrated, then tested, once each module has been designed, tested, and passed all of the tests.

Table 8 System testing

Serial No.	Description	Expected Result	Actual Result	Result
1.	Overall system testing	All the functionalities of the system should work properly after integration.	All the functionalities of the system are working properly after integration.	Passed

4.3 Result Analysis

The system was tested through unit testing and proved to be effective in executing its intended functions. The results showed that the project was able to meet its goals, but there is still room for improvement in terms of expanding the system's capabilities and increasing community involvement.

4.3.1 Evaluating Accuracy

In machine learning, accuracy is a common metric used to evaluate the performance of a classifier model. Accuracy measures the proportion of correctly classified instances among all instances in the dataset. To calculate accuracy, the first step is to divide the dataset into two parts: a training set and a test set. The training set is used to train the model, while the test set is used to evaluate the model's performance. In classifier model the most common measure to evaluate accuracy are:

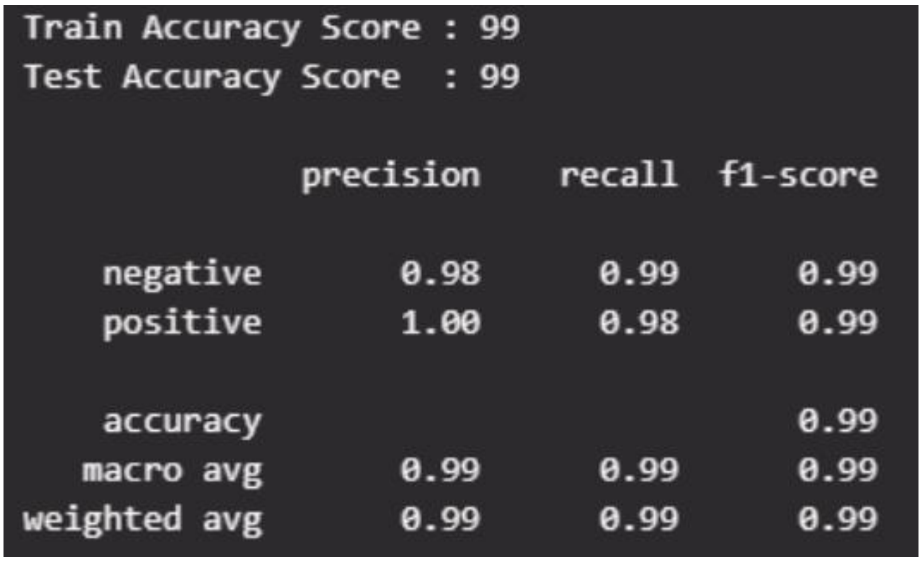
- Precision: Precision is the fraction of true positives among all the positive predictions made by the model. It measures how accurate the model is when predicting positive instances. The formula for precision is:
  Precision = True Positives / (True Positives + False Positives). (8)

- Recall: Recall is the fraction of true positives among all the actual positive instances in the dataset. It measures how well the model is able to identify positive instances. The formula for recall is:

  Recall = True Positives / (True Positives + False Negatives). (9)

- F1 score: The F1 score is the harmonic mean of precision and recall. It provides a single score that balances the tradeoff between precision and recall. The F1 score ranges from 0 to 1, where a score of 1 represents perfect precision and recall, and 0 represents the worst performance. The formula for F1 score is:

  F1 score = 2 * (Precision * Recall) / (Precision + Recall) (10)



```
Train Accuracy Score : 99
Test Accuracy Score  : 99

              precision    recall  f1-score

    negative       0.98      0.99      0.99
    positive       1.00      0.98      0.99

    accuracy                           0.99
   macro avg       0.99      0.99      0.99
weighted avg       0.99      0.99      0.99
```

Figure 26 Accuracy of the recommendation system

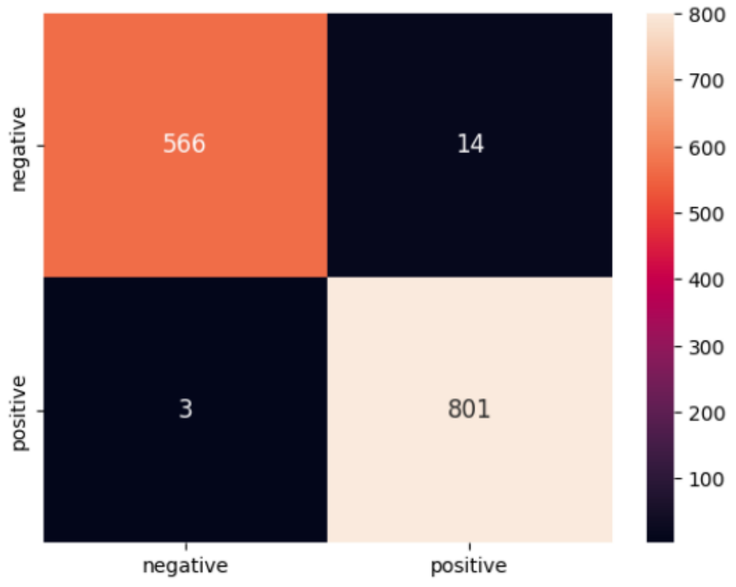


Figure 27 Confusion Matrix

Chapter 5: Conclusion and Future Recommendation

5.1 Conclusion

The movie recommendation system embodies a comprehensive and user-centric approach to enhance the cinematic experience for users. The system seamlessly integrates user authentication, advanced search and recommendation algorithms, and intuitive interfaces, ensuring a personalized and engaging journey for movie enthusiasts. The incorporation of content supervisors and managers adds a layer of content curation and review, ensuring the quality and relevance of the movie database. Robust data management through MySQL facilitates efficient storage and retrieval of user details, movie information, and reviews.

The continuous improvement aspect, marked by algorithm enhancements and regular updates, reflects the commitment to staying current and relevant in the dynamic landscape of film content. The system not only empowers users to discover and enjoy movies tailored to their preferences but also provides content managers and supervisors with the tools to contribute and curate a diverse and high-quality collection.

In essence, this movie recommendation system is designed not just as a platform for discovering movies but as a dynamic ecosystem that fosters user engagement, content curation, and ongoing improvement. With its user-friendly interface, comprehensive features, and commitment to staying at the forefront of recommendation technology, the system stands as a testament to the evolving landscape of personalized entertainment solutions.

5.2 Future Recommendations

1. Sentiment analysis for review and comments.
2. Reward based coupon systems.
3. Integration with external platforms and streaming services for enriched data.
4. Add movie watchable platforms
5. Extend language support for a global audience
6. Add other regional languages movies.
7. Create a user-friendly mobile app for on-the-go movie discovery.

References

[1] “Medium,” Medium. [Online]. Available: https://towardsdatascience.com/deep-learning-based-recommender-systems. [Accessed: 22-Jul-2023].

[2] Smu.edu. [Online]. Available: https://scholar.smu.edu/cgi/viewcontent.cgi?article=1205&context=datasciencereview. [Accessed: 22-Jul-2023].

[3] “IMDb: Ratings, reviews, and where to watch the best movies & TV shows,” IMDb. [Online]. Available: https://www.imdb.com/. [Accessed: 22-Jul-2023].

[4] R. Vidiyala, “How to build a movie recommendation system,” Towards Data Science, 02-Oct-2020. [Online]. Available: https://towardsdatascience.com/how-to-build-a-movie-recommendation-system-67e321339109. [Accessed: 22-Jul-2023].

[5] S. Das, “Create your own movie movie recommendation system,” Analytics Vidhya, 09-Nov-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/11/create-your-own-movie-movie-recommendation-system/. [Accessed: 22-Jul-2023].

[6] A. Goyal and A. Parulekar, “Sentiment Analysis for Movie Reviews,” Ucsd.edu. [Online]. Available: https://cseweb.ucsd.edu/classes/wi15/cse255-a/reports/fa15/003.pdf. [Accessed: 22-Jul-2023].

[7] “Connected papers,” Connectedpapers.com. [Online]. Available: https://www.connectedpapers.com/. [Accessed: 22-Jul-2023].

[8] K. Naik, “Tutorial 1- Weighted hybrid technique for Recommender system,” 31-Jul-2019. [Online]. Available: https://www.youtube.com/watch?v=_hf_y-_sj5Y&list=PLZoTAELRMXVN7QGpcuN-Vg35Hgjp3htvi. [Accessed: 22-Jul-2023].

[9] K. Naik, “Tutorial 2- creating recommendation systems using nearest neighbors,” 01-Aug-2019. [Online]. Available: https://www.youtube.com/watch?v=kccT0FVK6OY&list=PLZoTAELRMXVN7QGpcuN-Vg35Hgjp3htvi&index=3. [Accessed: 22-Jul-2023].

[10] K. Naik, “Cosine Similarity and cosine distance,” 19-Aug-2019. [Online]. Available: https://www.youtube.com/watch?v=ie-MjGVYw9ag&list=PLZoTAELRMXVN7QGpcuN-Vg35Hgjp3htvi&index=8. [Accessed: 22-Jul-2023].

[11] B. Rocca, “Introduction to recommender systems,” Towards Data Science, 02-Jun-2019. [Online]. Available: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada. [Accessed: 22-Jul-2023].

[12] Art of the Problem, “How recommender systems work (Netflix/Amazon),” 28-Feb-2020. [Online]. Available: https://www.youtube.com/watch?v=n3RKsY2H-NE. [Accessed: 22-Jul-2023].

[13] “Flask Tutorial,” Tutorialspoint.com. [Online]. Available: https://www.tutorialspoint.com/flask/index.htm. [Accessed: 22-Jul-2023].

[14] kindsonthegenius, “How to create an API in Python with Flask - step by step - kindson the genius,” Kindsonthegenius.com, 28-Jun-2022. [Online]. Available: https://www.kindsonthegenius.com/how-to-create-an-api-in-python-with-flask-step-by-step/. [Accessed: 22-Jul-2023].

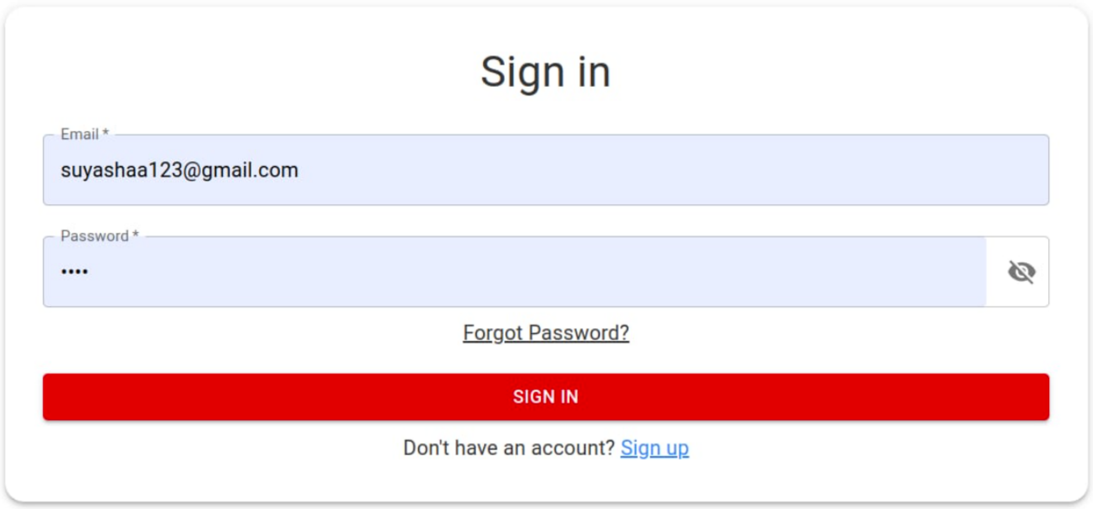
[15] K. Arellano, “How to use API with React? ReactJS API call example & beginner’s guide,” Rapid Blog, 10-Mar-2020. [Online]. Available: https://rapidapi.com/blog/how-to-use-an-api-with-react/. [Accessed: 22-Jul-2023].

[16] A. Malviya, “Python – flask mysql connection,” Codementor.io, 05-Jun-2020. [Online]. Available: https://www.codementor.io/@adityamalviya/python-flask-mysql-connection-rxblpje73. [Accessed: 22-Jul-2023].

Appendices:

Screenshots of UI:

i. Login Page



Sign in

Email *

suyashaa123@gmail.com

Password *

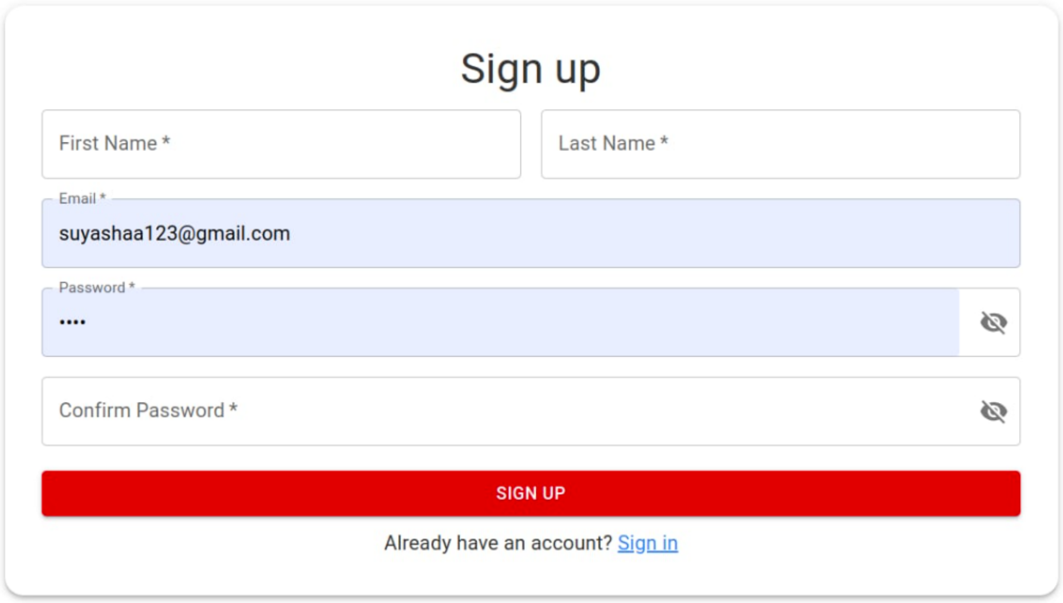
••••

Forgot Password?

SIGN IN

Don't have an account? Sign up

ii. Sign Up Page



Sign up

First Name *

Last Name *

Email *

suyashaa123@gmail.com

Password *

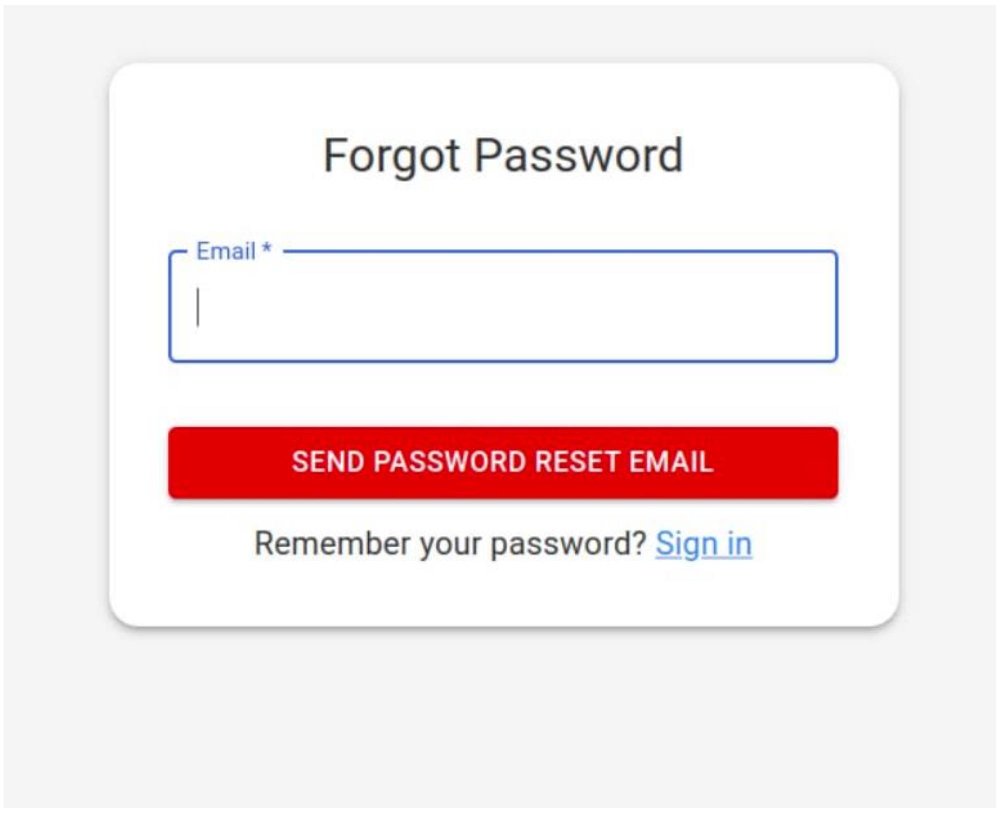
••••

Confirm Password *

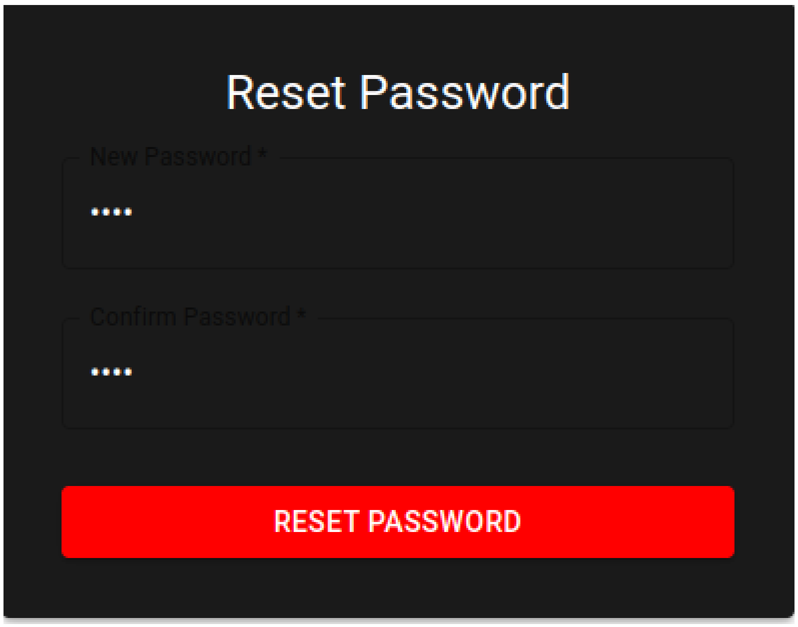
SIGN UP

Already have an account? Sign in

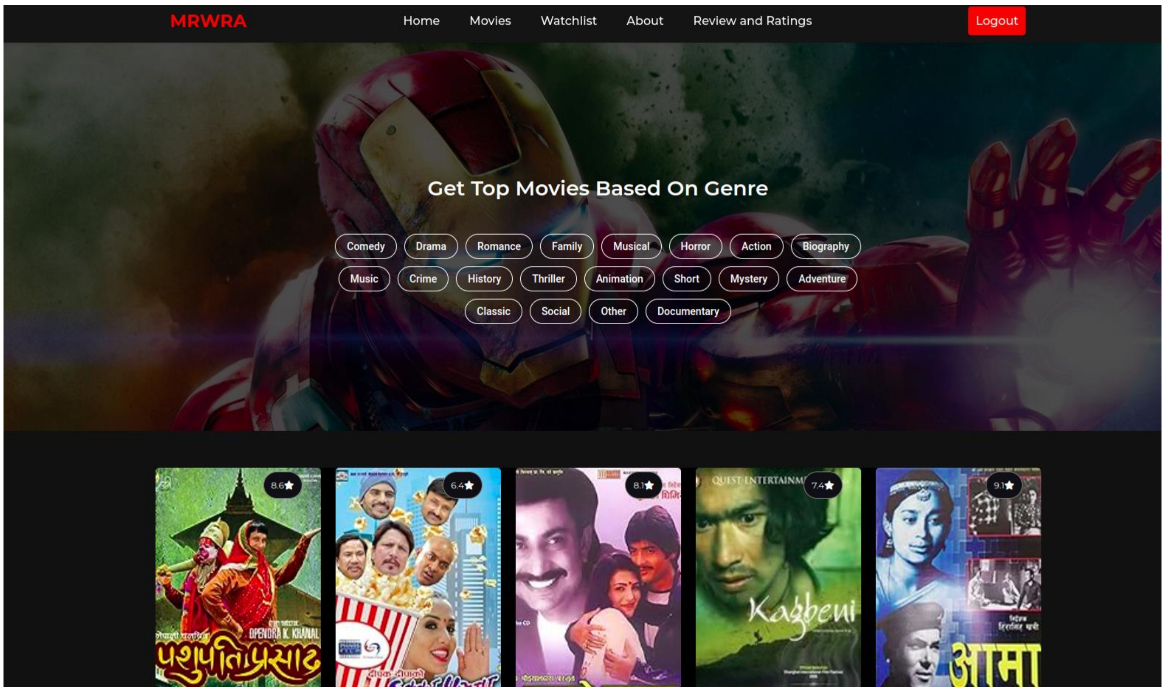
iii. Forgot Password



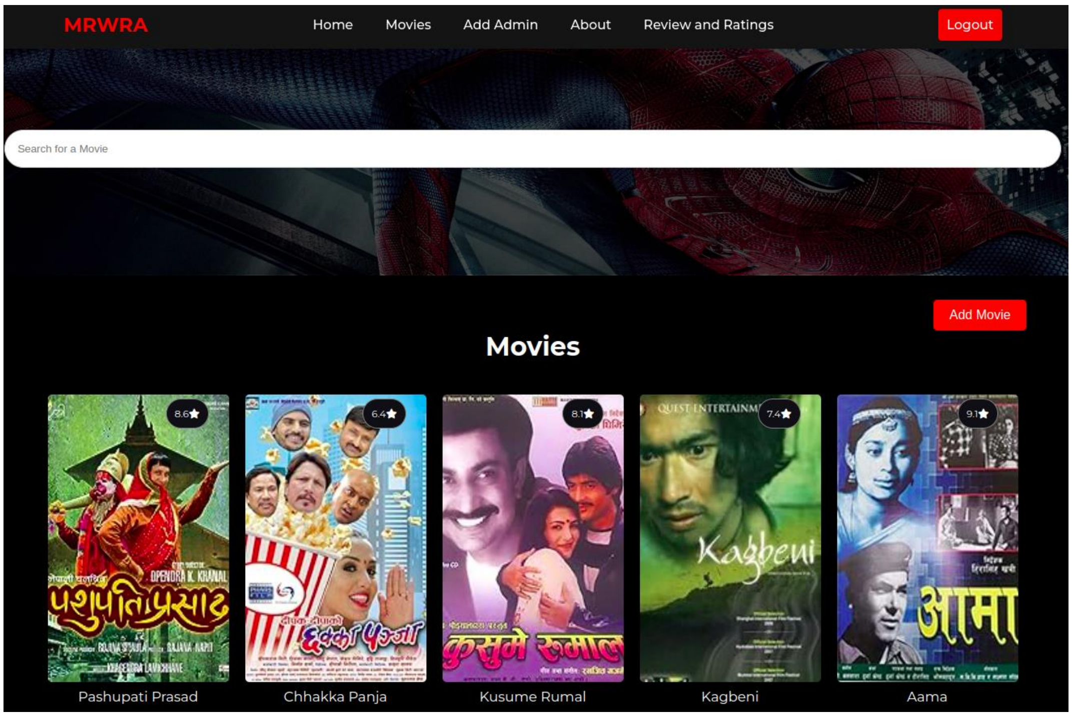
iv. Reset Password



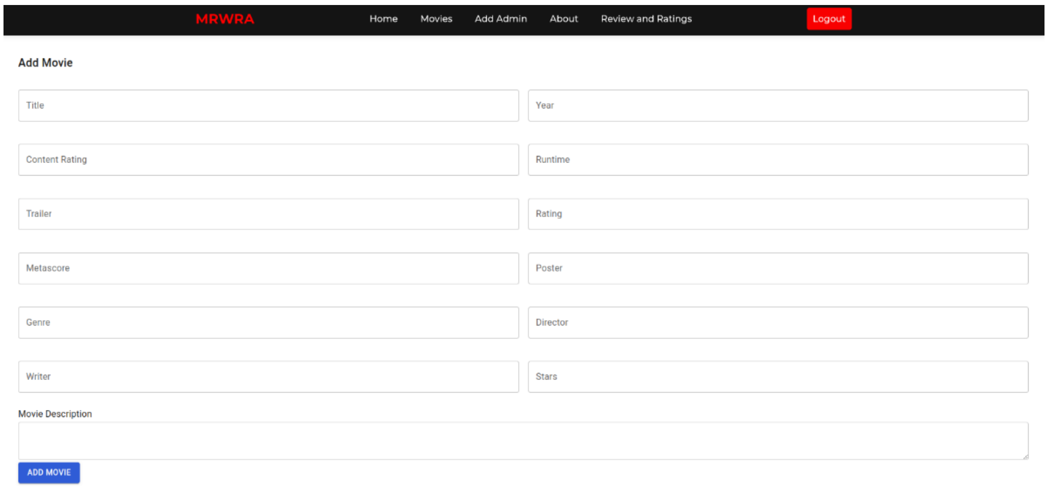
v. Home Page



vi. Movies Page for Admin



vii. Add Movie Page


MRWRA Home Movies Add Admin About Review and Ratings Logout

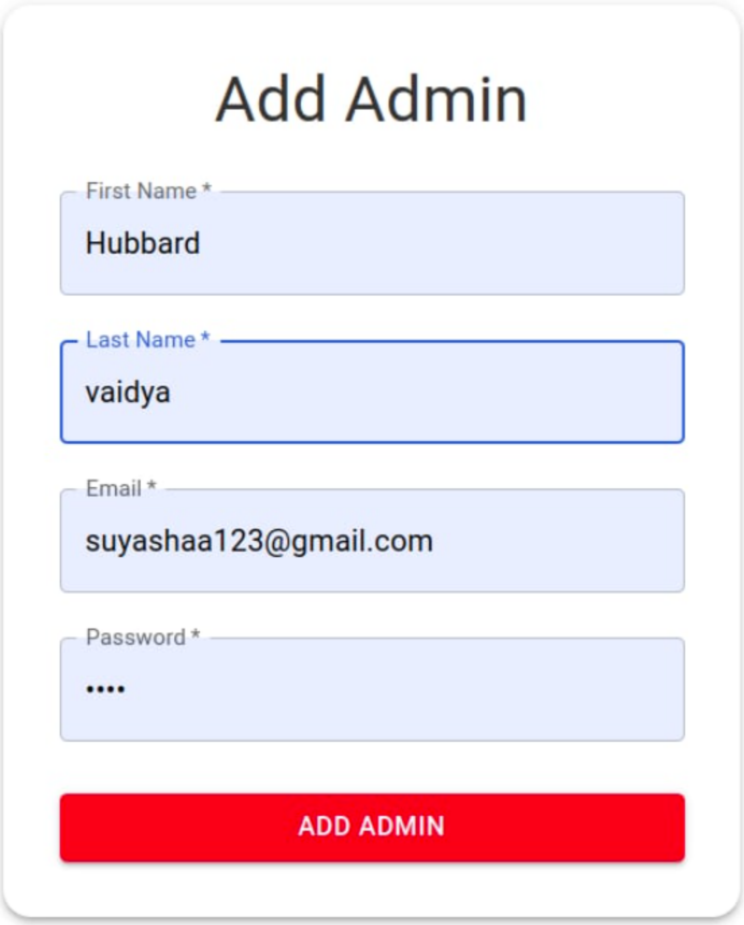
Add Movie

Title	Year
Content Rating	Runtime
Trailer	Rating
Metascore	Poster
Genre	Director
Writer	Stars

Movie Description

ADD MOVIE

viii. Add Admin Page


Add Admin

First Name * Hubbard

Last Name * vaidya

Email * suyashaa123@gmail.com

Password * ••••

ADD ADMIN

Source code:

Login

```
const LoginForm = () => {
  const navigate = useNavigate();
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const [showPassword, setShowPassword] = useState(false);
  const [openSnackbar, setOpenSnackbar] = useState(false);
  const handleSnackbarClose = (event, reason) => {
    if (reason === "clickaway") {
      return;
    }
    setOpenSnackbar(false);
  };
  const handleSnackbarOpen = () => {
    setOpenSnackbar(true);
  };
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post("http://localhost:8005/auth/login", {
        email,
        password,
      });
      const { token } = response.data;
      localStorage.setItem("token", token);
      handleSnackbarOpen();
      setTimeout(() => {
        navigate("/home");
      }, 500);
    } catch (error) {
      setError("Invalid Credentials. Please try again.");
    }
  };
```

Signup

```
function ResetPassword() {
  const { token } = useParams();
  const navigate = useNavigate();

  const [formData, setFormData] = useState({
    password: "",
    confirmPassword: "",
  });
  const [error, setError] = useState("");
  const [passwordsMatch, setPasswordsMatch] = useState(true);
const [showPassword, setShowPassword] = useState(false);
const [showConfirmPassword, setShowConfirmPassword] = useState(false);

const handleChange = (e) => {
  const { name, value } = e.target;
  setFormData((prevData) => ({
    ...prevData,
    [name]: value,
  }));
  if (name === "confirmPassword" && value !== "") {
       setPasswordsMatch(value === formData.password);
  }
};
const handleShowPassword = () => {
  setShowPassword(!showPassword);
};
const handleShowConfirmPassword = () => {
  setShowConfirmPassword(!showConfirmPassword);
};
const handleSubmit = async (e) => {
  e.preventDefault();
  setError(""); // Clear previous error message

  try {
    if (!passwordsMatch) {
      throw new Error("Passwords do not match");
    }
    // Make an API request to reset the password
    await axios.post(`http://localhost:8005/auth/reset-password/${token}`, {
      password: formData.password,
    });

    // Display a success message or redirect to a login page
    alert("Password reset successfully");
    navigate("/login");
  } catch (error) {
    console.error("Reset Password failed:", error);
    setError(error.message || "Failed to reset password. Please try again.");
  }
};
```

Movie

```
const MoviePage = () => {
 const navigate = useNavigate();
```

```
  const [searchText, setSearchText] = useState("");
  const [userData, setUserData] = useState("");
  const [movies, setMovies] = useState([]);
  const [searchResults, setSearchResults] = useState([]);
  const [recommendedMovies, setRecommendedMovies] = useState([]);
  useEffect(() => {
    getUserData();
    getAllMovies();
  }, []);
  const getUserData = () => {
    const storedToken = localStorage.getItem("token");
    if (storedToken) {
      const decodedToken = jwtDecode(storedToken);
      setUserData(decodedToken);
    }
  }
  const getAllMovies = () => {
    axios
      .get("http://localhost:8005/movies")
      .then((response) => {
        setMovies(response.data);
      })
      .catch((error) => {
        console.error("Error fetching data: ", error);
      });
  };
  const handleAddAdmin = () => {
    navigate("/admin");
  };
  const handleInlineSearch = (searchText) => {
    setSearchText(searchText);
    // Filter the movie list based on the search text
    const results = movies.filter(
      (movie) =>
        movie.title.toLowerCase().includes(searchText.toLowerCase()) ||
        movie.genre.toLowerCase().includes(searchText.toLowerCase())
    );
    setSearchResults(results);
    if (results.length > 0) {
      // Extract the genre of the first movie in the search results
      const genre = results[0].genre;
      // Filter recommended movies with the same genre, excluding the first movie
      const recommendations = movies.filter(
        (movie) => movie.genre === genre && movie.id !== results[0].id
      );
      setRecommendedMovies(recommendations.slice(0, 5)); // Ensure at least 5 recommended movies
    } else {
      setRecommendedMovies([]);
    }
  };
```

Home

```
 const Home = () => {
  const [homeGenreList, setHomeGenreList] = useState([]);
```

```
const [selectedGenres, setSelectedGenres] = useState([]);
const [currMovies, setCurrMovies] = useState([]);
const [recommendedMovies, setRecommendedMovies] = useState([]);
const [sortOrder, setSortOrder] = useState("aesc");
const [movies, setMovies] = useState([]);
const moviesSectionRef = useRef(null);
const navigate = useNavigate();
useEffect(() => {
  // Fetch movies and genres from your NestJS API
  axios
    .get("/movies")
    .then((response) => {
      setMovies(response.data);
      // Extract unique genres from the movies
      const uniqueGenres = [
        ...new Set(response.data.map((movie) => movie.genre)),
      ];
      setHomeGenreList(uniqueGenres);
    })
    .catch((error) => {
      console.error("Error fetching data: ", error);
    });
}, []);
useEffect(() => {
  if (selectedGenres.length > 0) {
    // Fetch movies based on selected genres from your NestJS API
   axios.get(`/movies/filterByGenres?genres=${selectedGenres.join(",")}`)
      .then((response) => {
        let sortedMovies = response.data;
        if (sortOrder === "desc") {
          sortedMovies = sortedMovies.sort(
            (a, b) => b.vote_average - a.vote_average
          );
        } else {
          sortedMovies = sortedMovies.sort(
            (a, b) => a.vote_average - b.vote_average
          );
        }
        setCurrMovies(sortedMovies);
      })
      .catch((error) => {
        console.error("Error fetching filtered movies: ", error);
      });
  } else {
    // If no genres are selected, show all movies
    setCurrMovies([...movies]);
  }
}, [selectedGenres, sortOrder, movies]);
const onTagClick = (genreId) => {
  let isPresent = selectedGenres.includes(genreId);
  if (isPresent) {
    setSelectedGenres(selectedGenres.filter((item) => item !== genreId));
  } else {
    setSelectedGenres([...selectedGenres, genreId]);
  }
  window.scrollTo({
      top: moviesSectionRef.current.offsetTop,
      behavior: "smooth",
    });
  };
  const handleClick = (id) => {
    navigate(`/movie/${id}`);
  };
  const renderMovies = () =>
    currMovies.map((movie) => (
      <div key={movie.id} onClick={() => handleClick(movie.id)}>
        <MovieCard movie={movie} />
      </div>
    ));
```

Add Movies

```
 const AddMovie = () => {
  const [movieData, setMovieData] = useState({
    title: "",
    year: null,
    contentRating: "",
    runtime: "",
    description: "",
    rating: null,
    poster: "",
    genre: "",
    director: "",
    metascore: null,
    writer: "",
    stars: "",
    trailer: "",
  });
  const navigate = useNavigate();

  const handleChange = (e) => {
    const { name, value } = e.target;
    let parsedValue = value;
    if (name === "year" || name === "rating" || name === "metascore") {
      // Use parseFloat for rating and metascore, and parseInt for year
      parsedValue = name === "year" ? parseInt(value) : parseFloat(value);
    }
    // Uncomment the following line to update the state
    setMovieData({ ...movieData, [name]: parsedValue });
  };
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post(
        `http://localhost:8005/movies/create`,
        movieData
      );
      navigate("/movie");
      if (response.status === 200) {
        console.log("Movie data sent successfully!");
      } else {
        console.error("Request failed with status:", response.status);
    }
  } catch (error) {
    console.error("An error occurred:", error);
  }
};
```

Backend

```
@Controller('movies')
export class MoviesController {
 constructor(
   private readonly moviesService: MoviesService,
   private readonly recommendationService: RecommendationService,
 ) {}
 @Post('create')
 create(@Body() createMovieDto: CreateMovieDto) {
   return this.moviesService.create(createMovieDto);
 }
 @Get()
 findAll() {
   return this.moviesService.findAll();
 }
 @Get('superAdmin')
 findAllBySuperAdmin() {
   return this.moviesService.findAllBySuperAdmin();
 }
 @Get('filterByGenres')
 filterByGenres(@Query('genres') genres: string) {
   const selectedGenres = genres.split(',');
   return this.moviesService.filterByGenres(selectedGenres);
 }
 @Post(':id/comments')
 createComment(@Param('id') movieId: string, @Body() data: { comment: string; userId: Auth;
userRating?: number }) {
   return this.moviesService.createComment(+movieId, data);
 }
  @Get(':id/comments')
 getComments(@Param('id') movieId: string){
   return this.moviesService.getComments(+movieId);
 }
 @Get(':id')
 findOne(@Param('id') id: string) {
   return this.moviesService.findOne(+id);
 }
 @Patch(':id')
 update(@Param('id') id: string, @Body() updateMovieDto: UpdateMovieDto) {
   return this.moviesService.update(+id, updateMovieDto);
 }
 @Delete(':id')
 remove(@Param('id') id: string) {
   return this.moviesService.remove(+id);
 }
}
```