DEVELOPMENT OF A CONTENT-BASED MOVIE RECOMMENDATION SYSTEM

BY

EZE, SONIA IFESINACHI
GOU/U22/CSC/850

DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY, GODFREY OKOYE UNIVERSITY, ENUGU STATE.

IN PARTIAL FULFILLMENT OF THE AWARD OF BACHELOR OF SCIENCE (B.Sc), DEGREE IN COMPUTER SCIENCE.

SUPERVISOR: DR. FRANK OKEBANAMA

JUNE, 2026.

APPROVAL PAGE

This research, titled "**Content-Based Movie Recommendation System**," has been assessed and approved by the Department of Computer Science Committees in the Faculty of Computing and Information Technology, Godfrey Okoye University, Enugu, Nigeria.

Dr. Frank Okebanama Supervisor	..... Signature	..... Date
External Examiner External Examiner	..... Signature	..... Date
Dr. Frank Okebanama HOD, Department of Computer Science	..... Signature	..... Date
Prof. I. A. Ajah Dean, Faculty of Computing And Information Technology.	..... Signature	..... Date

CERTIFICATION

I certify that I completed the research included therein and that it was primarily based on my observations and investigation. Consequently, I will fully accept the University's decision if I am found to have cheated on the assignment.

_____

EZE, SONIA IFESINACHI
GOU/U22/CSC/850

DEDICATION

The work “Content-based Movie Recommendation System” bears dedication to my parents, Mrs. Doris Eze and the Late Mr. Robinson Eze, who devoted enormous support to my achievements so far, along with my siblings, Oluchi Miracle Eze and Stephanie Nnesochi Eze, who formed an integral part of this achievement for continuously showing their supportive presence and encouragement throughout this project.

ACKNOWLEDGMENTS

My first gratitude goes to Almighty God, who offered His abundant mercy while inspiring me through His grace during this project work. I would like to express my special gratitude to my supervisor, Dr. U.F. Okebanama, who generously provided continual support, valuable guidance, precise criticism, and motivating words throughout this entire project. The specialized knowledge added significant value to the quality of what I produced in my work. I would like to thank my Uncles and Aunts, who supported me during tough times, and told me that I should be strong and keep on moving forward. I genuinely express my appreciation towards my lecturers, Prof. Ozofor, Mrs. Ekene Okafor, and Dr. Camilus for creating an excellent academic structure alongside a learning-friendly, innovative environment. Throughout this journey, I wish to show appreciation to my friends and coursemates who provided constant encouragement while motivating me and working together in collaboration. The priceless value came from sharing thoughts and experiences with others. I express my gratitude to my parents and all my family members for their prayers, moral support, and their endless love throughout my educational journey. Throughout my academic journey, your faith in me has served as my main motivating force. I express gratitude to every individual who helped make this project achieve its goals both straightforwardly and indirectly. Many thanks for any degree of impact you have made.

ABSTRACT

In the modern world of bustling digitalization, entertainment has become a significant factor that helps people to relax, refresh their spirits, and gain the power to continue with their daily routine. Movies are one of the most consumed types of entertainment among others. However, with the emergence of websites that provide opportunities for reviewing films online, there has been a significant amount of information generated. The process of going through vast content manually to find films that fit the preferences of consumers is a cumbersome process. Therefore, movie recommendation systems have become very essential in helping customers to find the movies relevant to them. This particular project seeks to propose a design and implementation of a movie recommendation system that will assist users in finding films and minimize information overload. It is based on content-based filtering, which suggests films according to similarity of their features, such as plot and genre. Movie information is analyzed using text processing and similarity measurement methods to give personalized recommendations. To enhance user trust and system transparency, the system provides explainable recommendations by indicating the factors that influenced each suggested movie. User inputs in context, like individual preferences, are included with the view of enhancing the relevance of the recommendations. The system that is produced is lightweight and simple to comprehend, and can be adopted in educational and small-scale contexts. The experimental assessment shows that the system is effective in terms of improving the quality of recommendations and user experience by offering useful and clear suggestions of movies in a natural and meaningful way.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

Tech-driven solutions and internet-related platforms have greatly changed the production, distribution, and consumption of movies due to their rapid development. Streaming services and movie repositories are now available online, which offer online users thousands of movies of various genres, languages, and cultures. Although it provides more content and thus more options and convenience, it has likewise introduced the problem of information overload, in which users struggle with finding movies that match their personal likes and needs.

To overcome this problem, movie recommendation systems have been developed that automatically recommend movies that are likely to be of interest to users. They are systems that examine the data they have so as to come up with personalized recommendations using variables like attributes of movies, preferences, and history of interaction. A crucial element of contemporary online platforms, recommendation systems have emerged as a key factor to improve the satisfaction levels of users, their interactions, and their content discovery, as observed in recent studies (Samih, Ghadi & Fennan, 2021).

Traditional movie recommendation systems have been using collaborative filtering, content-based filtering, or both in making recommendations. Collaborative Filtering uses data from users' actions and ratings to find similar items and similar users. While successful in large-scale commercial applications, Collaborative filtering faces problems such as cold-start issues, sparsity of data, and dependency on the amount of interaction data from users (Mirhasani *et al*., 2021). These limitations make the application of collaborative filtering unsuitable in smaller-scale, academic, and offline systems.

Content-based filtering, in its turn, suggests movies based on intrinsic features of movies (genres, plot descriptions, keywords, and metadata). Matches between these features and the expressed preferences of a user are used to produce recommendations. As it has been proven by several studies, content-based recommendation systems can be effectively applied to situations when users possess limited data, which makes them especially applicable to academic projects and lightweight applications (Joseph & Benjamin, 2022; Wandhekar *et al*., 2025).

Over the past few years, explainable recommendation systems, systems that seek to offer the user reasons that are comprehensible for recommendations, have become a topic of increased interest. It has been found that the more users understand why a specific item was recommended by the system, the more they will trust and accept the recommendations (Vultureanu-Albiși *et al*., 2021). The concept of explainability is of great relevance, especially in the context of education and research-driven systems in which the transparency and interpretability values hold greater significance than black-box decision-making.

There is, in addition, a clear and growing need for lightweight recommender systems that can operate effectively on small datasets and limited infrastructure. Articles in MDPI and other publications have highlighted that it is necessary to develop recommendation systems that are easy, user-friendly, and can be implemented in the resource-limited settings of schools and small institutions (Samih, Ghadi & Fennan, 2021).

Therefore, this project aims to develop a content-based, explainable, lightweight movie recommendation system. The system uses textual feature extraction (TF-IDF vectorisation) together with cosine similarity to generate recommendations. The proposed system is expected to minimize information overload and enhance the transparency and usability of the

recommendations and operation through the provision of human-readable explanations of the recommendations and the ability to operate on small datasets.

1.2 Statement of Problem

Although there are many systems available in terms of movie recommendations, there are still various challenges that are not addressed, especially in academics and on a small scale. A large number of the existing systems are based on collaborative filtering methods that need abundant user interaction information and constant collection of data. This dependency degrades recommendation quality for new users and newly added films, a difficulty commonly called the cold-start problem (Mirhasani *et al*., 2021).

Moreover, most of the existing commercial systems are black-box systems, with little or no information provided to the clients about how they make the recommendation. Such opacity has a negative effect on user trust, and limits the value of such systems as an educational tool (Vultureanu-Albiși *et al*., 2021). Furthermore, the computation and infrastructure demands of most of the advanced recommendation systems are incompatible with lightweight or offline applications.

There is therefore a need for a movie recommendation system that:

a) Has a small data functionality,

b) Gives clear and comprehensible recommendations,

c) Has low calculation requirements, and

d) Is applicable in small-scale and academic applications.

1.3 Aim and Objectives of the Study

The purpose of the research paper is to develop a content-based movie recommendation system. The objectives that would be accomplished during this research are:

1. Identify challenges inherent in movie recommendation
2. Design a movie recommendation system that recommends movies based on the similarities in the features of the movies.
3. Employ a content-based filter approach using the vectorization technique of TF-IDF to determine movie similarities using cosine similarity measures.
4. Develop an interface through which a user can search for any particular film and get recommendations for films related to it.

1.4 Significance of the Study

This study is important because it contributes to both academic studies and the practical uses of movie recommendation systems. The proposed system is relevant to various stakeholders, as outlined below.

1. **Students and Researchers:** This research paper will be useful for students of computer science, information technology, and related fields. It gives a practical illustration of how recommendation systems can be designed and implemented based on content-based filtering. The project has provided a reference source in academic studies that have been conducted on recommender systems, object-oriented system design, and machine learning applications to entertainment systems.

2. **Academic Institutions:** The suggested movie recommendation system can be applied to academic settings because of its lightweight and privacy-friendly nature. The system can be used by institutions as a pedagogic device to illustrate the ideas of the recommendation system that does not need significant datasets or infrastructure in the clouds. It could also be applied in the laboratories to do some practical coursework and final-year project demonstrations.

3. **Software Developers:** The software developers can use this study to know how to come up with simple, explainable, and efficient recommendation systems. The project exhibits the use of object-oriented analysis and design, UML modeling, and content-based filtering, which can be used to customize in other fields like book or music recommendation systems.

4. **Small-Scale Streaming Platforms:** Small and small emerging streaming platforms in some cases, do not have the resources to support large-scale recommendation systems. This paper introduces a system that is efficient with small data and small computation requirements. These platforms are able to implement or expand the suggested system to increase user interactivity without major infrastructural investment.

5. **End Users (Movie Viewers):** Proposed system advantages for movie watching people will be reduced efforts and time spent finding movies according to user's taste. The explainable recommendations also make the users understand why some movies are recommended, which enhances levels of trust and satisfaction with the system.

6. **Educational Project Evaluators and Examiners:** The study is pertinent to project evaluators and examiners because it shows how the ideas about theoretical matters can be practically applied in the recommendation systems, system analysis, and software design. A system development was approached in a structured and methodical manner as indicated by the clear documentation, UML diagrams as well as the database schema.

7. **Future System Designers:** Future system designers and researchers may take this work further to come up with more sophisticated recommendation systems. The research gives a basis of building on the system with hybrid solutions, real-time communication with the user, or sophisticated machine learning solutions without compromising the transparency of the systems.

1.4 Scope of Study

Only a content-based movie suggestion tool is to be built in this project. It is based on analyzing film properties such as film description and genres for recommending movies. Large-scale algorithms based on user interaction data are collaborative filters, which are outside the scope of the present study. The system is also intended as an educational and small-scale product and is not focused on competing with big commercial streaming sites. The assessment is performed based on a small set of data to prove that it is feasible, effective, and explainable.

1.4 Limitations of the Study

1. By restricting the approach to content-based filtering, the system depends solely on features like genre and plot. This can narrow recommendation variety and hinder real-time adaptation to user tastes.
2. The system is not being fed with large-scale user interaction data (ratings or viewing history), which constrains its capability to produce more highly personalized recommendations than collaborative or hybrid recommendation systems.
3. System effectiveness relies heavily on dataset quality and completeness. Inaccurate or incomplete movie information may affect the accuracy of the recommendations generated.
4. Also, contextual information, such as the mood or preferences of the user, is obtained manually, and this does not necessarily indicate the intentionality of the user to view the content. The system is not automatically dynamic with regard to the behavior of the users in real-time.
5. Lastly, the system was created to be educational and small-scale and is not scalable, does not deal with real-time processing, and does not deal with a large-scale deployment.

1.5 Operational Definitions of Terms

1. **Recommendation System:** A program that proposes a list of items to the user according to a particular criterion.
2. **Content-Based Filtering:** This is an algorithm that recommends items like those that a user likes, according to the features of the item.
3. **Information Overload:** This is when an individual receives too much information that complicates the process of decision-making.
4. **Context-Aware Recommendation:** This is a recommendation strategy that takes contextual variables into account, such as mood or situation.
5. **Cosine Similarity:** It is a mathematical measure used to measure similarity between two vectors.
6. **TF-IDF (Term Frequency-Inverse Document Frequency):** It is an approach to convert text data into vectors for analysis statistically.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This part covers the works and systems that are connected to the proposed system. The theory behind the recommendation system is elaborated along with various types of movie recommendation systems, the history, and its future trends. Literature review of the topic includes the discussion on strengths and weaknesses of existing approaches, indicating the need for the development of a movie recommendation system in this study in relation to the content-based recommendation system.

2.2 Theoretical Background

Movie Recommendation Systems are a special type of information filtering system, which recommend movies to their users according to their preferences, interests, and past activities. Its primary objective is to get the user experience more personal, and one does not even have to work as hard to identify the content that is pertinent. As more and more content of online movies becomes available, users are likely to experience information overload, thereby rendering them unable to make informed decisions. Recommendation systems are solutions to this problem as they can tailor their delivery of content and direct users to the closest movies.

According to recent research, recommendation systems operate by examining properties of movies, user preferences, or interaction patterns. The content-based recommendation systems are designed to detect intrinsic movie characteristics like genre, plot description, keywords, and metadata, and compare them to the user interests (Joseph and Benjamin, 2022; Meghana *et al*., 2023).

Unlike collaborative filters, which rely mainly on interaction data from users, content-based systems can be implemented in case when there is no user data or there is not enough user data available. This fact makes them especially useful with new users and small datasets, as was pointed out in multiple recent reports (Wandhekar *et al*., 2025; Zhang *et al*., 2022). Furthermore, the development of explainability in recommender systems is among the most significant research directions, because users are becoming increasingly transient and trusting their automated decision-making systems.

2.2.1 Description of various types of Movie Recommendation System.

These are varieties of movie recommendation systems, which vary according to the techniques used to generate recommendations.

- **Content-Based Filtering**

This is a type of recommendation system that uses the features of the movies and compares them with the preferences of the user to give a movie suggestion. These features can be movie genres, plot descriptions, keywords, cast, and other metadata. Usually, the preferences of the users are created based on their past likes or chosen movies.

Computation of similarity among movies is usually done by using techniques like TF-IDF vectorization, cosine similarity and textual feature extraction. Content-based systems are independent of the data of other users and, therefore, are appropriate when new users are utilized and when the environment has limited datasets (Joseph and Benjamin, 2022; Meghana *et al*., 2023). Also, they are more easily understood as these systems are transparent.

- **Collaborative Filtering**

It is a recommendation system that makes recommendations based on the features of the films compared with those of the user's preferences. So similarities in users or similarities in movies are created, and it generates the recommendations.

Collaborative filtering is effective in large-scale systems, but it experiences the cold-start problem, high sensitivity to large amounts of user interaction data, and sparsity in data (Mirhasani *et al*., 2021). These restrictions are a setback to its use in small, academic, or offline environments.

- **Explainable Recommendation Systems.**

Explainable recommendation systems are concerned with giving clear and understandable reasons as to why a movie is recommended. These systems are not just proposing movies and this is the reason why this particular movie should have been proposed, like the reason being that it is similar to other genres or keywords or even the users like it.

Explainable systems have been found to raise user trust, satisfaction, and acceptance of recommendations by a significant margin (Vultureanu-Albișі *et al*., 2021; Samih, Ghadi and Fennan, 2021). Apart from being especially useful in educational and research-focused applications, explainable recommendation systems are more useful where transparency and interpretability hold greater importance than black-box decision making.

- **Hybrid Recommendation Systems.**

This is a combination of two or more types of recommendations, such as the content-based and collaborative filtering types. With the merits of various approaches, hybrid systems have the potential to minimize certain weaknesses in each approach.

Nonetheless, research shows that hybrid systems can be expensive in terms of data, increased computing resources, and system design, not suitable for lightweight or academic implementation (Sharma *et al*., 2024). Consequently, the current trends of research prefer more straightforward and understandable models.

2.2.2. Brief History of Movie Recommendation System.

As a concept, recommendation systems date back to the early 1990s in response to the continuously growing information overload. The earlier, cruder forms of the recommendation systems relied upon the rule of thumb and keyword matching procedures in the recommendation of things to users. These systems had a weakness in that they could not tailor the recommendations and fit user preferences.

Since the increased use of digital platforms and availability of user data made it possible for there to be easy access, there was an increase in the use of the collaborative filtering method. These methods employed user ratings and interaction statistics in determining trends and similarities between the users and the items. Collaborative filtering was among the most effective models in the e-commerce and entertainment platforms, which made it a model in the industry of capturing the behavior of users.

With the rise of machine learning and data mining, new, more advanced recommendation methods have been proposed☒ The accuracy of the recommendation and the degree of scale were enhanced with the help of matrix factorization, clustering algorithms and classification. However, these are developed methods that are extremely data-intensive and computationally expensive.

In spite of it, recent studies indicate the trend towards not only accuracy-oriented models but also systems with a strong focus on interpretability, privacy, and lightness of design. Content-based and explainable recommendation systems have become the topic of more attention with the growing worries regarding data privacy and computational cost (Meghana *et al.*, 2023; Zhang *et al.*, 2022). This development indicates the necessity of systems which are effective as well as transparent and appropriate to limited settings.

2.2.3 The Future of Movie Recommendation System.

It is estimated that movie recommendation systems will majorly focus on higher individualization, decipherability, and privacy conservation in the coming years. The recommendation systems will continue to be advanced to support the constantly increasing needs of the users so that they can provide a more personalized experience as people strive to provide personal preferences and context, such as mood, time, and setting.

Explainable artificial intelligence (XAI) and privacy preservation are becoming the new frontiers of movie recommendation systems, as the end-users are demanding to know the logic behind the specific movies being suggested to them and privacy protection, as more people are becoming familiar with the data collection and privacy regulations, systems with less data collection and with smaller volumes of personal information are becoming more important. Explainable systems allow the user to see how their preferences influenced a recommendation to make the system interactive and user-friendly. Recent researches focus on the fact that users are more than happy to use recommendation systems that provide clear explanations of why certain movies are recommended (Vultureanu-Albişi *et al*., 2021; Samih, Ghadi, and Fennan, 2021).

Also, the need to have recommendation systems that perform well with small data sets and do not consume a large amount of computational power is increasing. Lightweight content-based models, which can be based on textual features extraction, will be valuable in academic institutions and small-scale systems, in particular (Zhang *et al*., 2022; Wandhekar *et al*., 2025). These trends are encouraging the orientation of the proposed system.

Overall, it is questionable that in the future the movie recommendation systems can be more simplified and more transparent as well as be able to use ethical information without the

need to reduce the accuracy. Such trends make it easier to develop effective recommendation systems, which are simultaneously reliable and accessible.

2.3 Review of Related Works

Different scholars have proposed different methods of establishing movie recommendation systems. The articles vary in the approach, amount of information, and complexity. The review below establishes excellent works that may be used in this study.

Meghana *et al*. (2023) sought to alleviate information overload in the movie platforms with the application of content-based recommender which relied on attributes and text similarity of the movies. The rationale was that the conventional movie search is ineffective to the users as the content increases. The authors used machine learning and item feature analysis and constructed user/movie profile based on textual attributes and calculated similarity among movie vectors. A weighted feature approach was found effective as experimental evaluation proposed a better representation of movie features and relevance of recommended movies. Nonetheless, the work is not providing any detailed performance metrics (e.g., precision/recall) and actual user evaluation, includes no explainability mechanisms, limiting its applicability to systems where transparency is important.

Joseph and Benjamin (2022) focused on the way in which the process of movie discovery could be improved therefore, they proposed displaying movies that are similar by their genre and content to the user, to reduce the work effort of search. The TF-IDF algorithm was used to represent the movies genres as vectors, and cosine similarity was used☒ The data was picked as an online data repository on movies. Cosine similarity was found to be a good way of discovering similar content movies. The system had a good relevance of recommendations but lacked in-depth

reporting of the precise numerical accuracy scores. The amount of the data was sufficient and applicable in academics. Being the best approach as it was, it failed to consider the evolving user preferences or provide reasons as to why certain recommendations were offered. The elements that can be explained can increase the clarity of the system.

Vultureanu-Albisi *et al*. (2021) article was informed by the necessity to comprehend the concept of explainability in recommender systems and upgrade it. The authors surveyed the role of explainable AI in recommender systems and how the explanations can increase user acceptance and trust of recommendations. The paper has summarized the available explainable frameworks of recommendations and the significance of transparency in the output of recommendations. Results have highlighted that the explainability aspect is one of the factors contributing to user satisfaction and perceived credibility of recommendations. It is one of the weaknesses because the paper offers a survey at a high level without illustrating and evaluating concrete explainable models upon actual data, and performance measures in terms of quantitative suggestions are not displayed.

Zhang *et al*. (2022) investigated how feature-based explanation and features of the modality of the output (text vs. voice) affect user satisfaction with the recommender system. The rationale was that the explainability level would make it more transparent, yet the form of the explanations (number of features, relevance in the context) may influence the user satisfaction more diversely. They had controlled experiments of user responses on different conditions of explanations. The results showed that the situation-relevant and extended explanations (3-4 features) produced significant positive effect on the satisfaction in the text mode but no significant design effect on voice explanations. Though it is very prudent in the design of systems, it was

developed on the premise of a general service recommender rather than movie recommendation and its application in the movie content context is subject to further experimentation.

To improve movie suggestions, Sharma *et al*. (2024) created a hybrid suggestion framework (TF-IDF content vectorization and weighted rating/popularity scaling) to improve recommendations of movies. This model used cosine similarity to find matches and aspects to give top-K recommendations on the basis of movie data. The findings showed that the framework has the potential to give valuable movie recommendations, which shows advantages of hybrid weighting. Nonetheless, since the work lacks explainability emphasis and the dataset context/size is not mentioned, it is indirectly related to lightweight explainable systems and generalization is not clear.

Permana and Wibowo (2023) have addressed the issue of the problem of large catalogues and involvement of users in movie choices. He developed a content-based filtering algorithm using TF-IDF, Gensim keyword extraction and cosine similarity and used it on the MovieLens data set. The values of precision, recall and f1-score were less than 0.1 that represented weak quality of recommendation. The data was voluminous and the overlaps in the key words were not many. The authors discovered that keyword TF-IDF may not be utilized as a recommendation technique of movies.

Kunde *et al*. (2022) paid their attention to the creation of the simple and effective movie recommendation system that could be implemented by scholars. The TF-IDF vectorization and Levenshtein distance-based system of content-based filtering was tested on a publicly accessible movie-based dataset. A quite large similarity of 0.7 was obtained in the system and compared to, however, no standardized similarity measures such as precision or recall. The size of the sample

was very small. Whereas the approach can prove useful in an academic setting, the cosine similarity can be more effective and convenient than the Levenshtein distance.

This study was conducted to increase the accuracy rate of the recommendations through the assistance of the textual similarity methodology (Wandhekar *et al.*, 2025). Cosine similarity-based and TF-IDF-based content-based filtering were used on a publicly available movie dataset. The evaluation findings were also discovered to be more applicable in recommendations than the base procedures. In contrast to the system where good performance measures were (e.g., precision = 0.89, recall = 0.87, F1-score = 0.88), transparency and offline functionality were not addressed in the research. A simplified model that can be easily explained could be able to increase its application.

Mirhasani *et al*. (2021) have addressed the cold-start problem of collaborative filtering systems. The sentiment analysis data and user interaction data were combined in order to beat cold-start. The information was collected on social network sites. The precision of the system recommending the new users was improved. The data was voluminous and complex, and the method was unsuitable in miniature academic systems. The less complex alternative is the content-based approach.

De Campos *et al*. (2024) were keen on the application of the concept of explainability to content-based recommendation systems. Techniques of textual similarity were used together with the explanation generation techniques. A public dataset was used to validate it. The system was precise in giving recommendations and also clear explanation. This system works well but due to the complexity of the system, one can reduce it so that it can be lightweight and hence can be implemented in the academic system.

Yunanda *et al*☒ (2022) evaluated the efficacy of TF-IDF and cosine similarity in the recommendation task☒ They applied TF-IDF vectorization and cosine similarity to textual news data of the Microsoft News dataset as well. The system was well-relevant and efficient about recommendations. The study, but not a movie dataset, confirms the suitability of a text-based recommendation system by the use of TF-IDF and cosine similarity.

The authors tried to develop a precise and explicable movie recommendation model that allows the user to become familiar with the reasoning behind all recommendations (Samih, Ghadi, and Fennan, 2021). The authors have proposed ExMrec2Vec, which is an explainable movie recommender system built on the graph of Word2Vec. Similar vectors were found in the context and were extracted to form recommendations, and the system matched the recommendations with easy-to-understand explanation styles that were easy to understand by human beings. The size of the datasets was applied to depict and analyze the common movie recommendation standards, such as the MovieLens dataset portions. The ExMrec2Vec algorithm performed well. System-generated recommendations were readable and relevant. Users could understand why some movies would be recommended. The size of the precise dataset did not become known, making it impossible to have reproducibility and quantitative comparison.

Jayalakshmi *et al*. (2022) systematized the movie recommender systems. They discussed content-based, collaborative, and hybrid methods, improvement, and evaluation measures. Scalability, cold-start problems, sparsity, and unstandardized evaluation were determined to be the main challenges observed in the study. Nevertheless, the article failed to hypothesise and experiment with a new model. It was not experimental and was descriptive.

The Bhavsar *et al*. (2025) created a content-based movie recommendation system that utilizes the TF-IDF and CountVectorizer algorithms on the TMDB data, with the goal of making the movie-recommendation even in cases where the history of previous users is not known. The movie textual features generated the recommendations by use of NLP preprocessing and cosine similarity. It was implemented that content-based filtering can make recommendations which are not only relevant but no user interaction data is required to do so and that it works well with cold-start users. Limitations are the thematic relevance as opposed to quantitative metrics and the absence of the exploration of explainability features.

Yuan *et al*. (2023) sought to enhance the performance of movie recommendation systems by applying movie metadata to create personalized recommendations to the users. The research was motivated by the need to solve the issue of information overload that users go through in the search of movie databases that are large. The authors have come up with content-based recommendation system that applies words movie characteristics to establish similarity among movies. Textual information was utilized as features extracted, e.g. movie descriptions and movie genres etc. and similarity between movies was determined by similarity measurement methods. The dataset was made up of the metadata of movies that were collected publicly. Experimental results revealed that the suggested model was in a position to recommend films that have similar traits with the movies chosen by the users. The system enhanced the relevance of the recommendation by recommending movies that are similar in terms of genre and content features. Nevertheless, the research primarily depended on textual similarity and did not include explainable mechanisms of recommendations. Further, assessment measures like precision and recall were not given in detail, which constrained quantitative analysis of the performance of the model.

Singh, Mishra and Singh (2024) created a content-based recommender system that additionally provided the movie recommendation accuracy with the help of the cosine similarity to improve user experience. The study was motivated by the need to develop a system that would accurately record movies that would best suit the user preferences basing on the attributes of the movies. The authors adopted a recommendation system in Python where the movie features in terms of genre, cast, plot, and the director were transformed into a numerical vectors. Similarity scores in movies were then computed using cosine similarity. The data was made up of movie metadata, which was gathered in movie databases available publicly. The findings indicated that the system was successful in producing individualized recommendations because of the identification of movies with greater similarity scores. System performance was measured using evaluation metrics like precision, recall and accuracy. Even though the model showed high-performance in terms of recommendation, it is more dependent on metadata capabilities and does not take into consideration the dynamic user preferences. Also, the system does not have the features of explainability that would assist the user in knowing why a movie was suggested.

Sushanth *et al*. (2024) aimed at enhancing the accuracy of movie recommendation by including sentiment analysis based on the user reviews. The research was motivated by the desire to eliminate the shortcomings of the traditional recommendation systems that are only dependent on the metadata of the movie or ratings. The researchers have developed a model of recommendations that integrates a machine learning method with sentiment analysis to evaluate the views of users depicted in cinema reviews. The data set was movie review data that was obtained online. Review sentiment analysis methods were used to categorize the reviews into positive or negative sentiment, and these sentiments were incorporated into the recommendation algorithm. The findings indicated that the use of user sentiment assisted the system in producing

more personalized recommendations and context-sensitive recommendations. The method, however, involves a lot of user review feedback and exhaustive text preprocessing. This adds complexity during computation and renders this model inappropriate for lightweight recommendation systems.

Sidhu and Sharma (2025) proposed a cosine similarity-based and TF-IDF vectorization content-based movie recommendation system to improve the personalized movie recommendation☒ The authors focused on producing recommendations based on the different metadata available for the movies, including their genres, cast, crew, and keywords, which play a key role in determining the movie's content☒ This approach involved extracting features of the movies, converting the text data into numbers using TF-IDF vectorization, and finding similar movies by determining the similarity between the vectors using cosine similarity☒ The datasets of the research were based on publicly accessible movie datasets as IMDb or TMDb. The experimental outcome indicated that the system was able to suggest movies that have similar thematic and genre features with the input movie. Nevertheless, the system still has a cold-start issue, and it is not diverse in recommendations. The authors indicated that an even higher improvement of performance might be achieved through the hybrid approaches based on collaborative filtering and deep learning.

To improve the content-based movie recommendation system, Snigdha *et al*☒ (2022) calculated cosine similarity using TF-IDF to recommend similar movies to the users. The research aimed to help users find movies that suit their tastes without necessarily relying on user ratings. The approach entailed extracting textual descriptions of the movies and transforming these

descriptions into feature vectors using TF-IDF. The similarity of movies was then established using cosine similarity. The data was made up of movie metadata from publicly available movie datasets. The experimental data showed that the suggested system was successful in identifying movies that share similarities in plot description and genres. Though the model did not take into account any other measures except the textual similarity, it did not take into account other measures like user interaction data or sentiment analysis. As a result, there can be little recommendation diversity and personalization.

The article by Wang (2023) introduced a content-based recommendation algorithm of movies and television shows aimed at enhancing the accuracy of the recommendation. The paper has focused on the issue of poor recommendation systems that do not get significant relationships among the attributes of movies. The procedure includes the process of extracting the movie features, through TF-IDF vectorization and the process of calculating similarities in terms of cosine similarity. Further, dimensionality reduction methods were used to enhance computing speed. The experiment dataset was a publicly available movie dataset of movies descriptions and metadata. The results of this study showed that the proposed algorithm could outperform conventional similarity-based algorithms in accuracy☒ The model is, however, less applicable to lightweight recommendation systems such as those used in academic settings since it needs relatively large datasets and more computer resources.

2.4 Summary of Literature Review

S/N	Author(S)	Contribution Or (Work Done)	Technique Or Methodology	Limitations
1.	Meghana, K., Sudeeksha, E., Somanth, A., & Srinivasulu, Y. (2023, January).	Created a content-based movie recommender based on movie features and similarity of movie texts.	It applied TF-IDF, feature-based filter, and cosine similarity.	No detailed performance metrics. No explainability mechanism. Limited evaluation.
2.	Joseph & Benjamin (2022, July).	A suggested movie recommendation system based on cosine similarity and TF-IDF vectorization on the genres of movies. The system was relevant to academic datasets but not to dynamic user tastes and justified recommendations	TF-IDF and cosine similarity were used in recommending.	There is no recommendation of dynamic user preferences, as well as the inability to explain recommendations to users.
3.	Vultureanu-Albiși, A., & Badica, C. (2021, August).	Intelligible AI practices in recommender systems were assessed, and visibility as well as user confidence were emphasized.	Conceptual and literature-based review of explainable recommendation models.	No testing of implementation and dataset. Lacks quantitative assessment.
4.	Zhang, Z., Chen, L., Jiang, T., & Li, Y. (2022, May).	Researching the effect of explanations based on features on the satisfaction level of users in a recommender system.	An experimental user study in the form of comparison of explanation forms (text/voice).	Not limited to the film systems. No improvement of algorithms suggested.
5.	Sharma, S., Dubey, G. P., & Shakya, H. K. (2024, March).	Suggested hybrid framework of movie recommendations based on similarity between features and weighted by rating.	Hybrid (TF-IDF that meets weighted rating scaling).	Scarcity of emphasis on explainability. Not clearly analyzed dataset size and scalability.
6.	Permana, A. H. J., & Wibowo, A. T. (2023, December).	Movie recommender TF-IDF-based: The content of movies are recommended through TF-IDF using a machine learning tool to extract keywords in the	TF-IDF keyword extraction Cosine similarity on MovieLens dataset.	Extremely weak levels of precision, recall and F1-score; TF-IDF based on key words alone was not sufficient..

		MovieLens dataset. The evaluation performance was very low with low precision, recall, F1-score (<0.1) which meant that movie recommendation cannot work effectively with keyword-based TF-IDF alone.		
7.	Kunde, O., Gaikwad, O., Kelgandre, P., & Damodhar, R. (2022, May).	Built a TF-IDF and Levenshtein distance movie recommendation system based on content-based and was academic-oriented. It gave fairly decent similarity matching the system on a small scale but it was proposed that cosine similarity would be more efficient.	Levenshtein distance and TF-IDF based on Content-based filtering for text similarity.	Lack of standardized evaluation measures; Cosine similarity measures are more effective than the Levenshtein distance.
8.	Wandhekar, R., Nandane, K., & Garg, M. (2025, June).	Recommended a content-based system of movie recommendation based on cosine similarity and TF-IDF. The system did not consider explainability or limitations of offline deployment, yet had high evaluation scores (precision = 0.89, recall = 0. 87	Sentiment analysis together with collaborative filtering to reduce cold-start.	No mention of explainability; nothing on constraints of deployment.
9.	Mirhasani, M., & Ravanmehr, R. (2021, January).	Solved the cold-start issue based on sentiment analysis and collaborative filtering of the social media datasets. The system also increased accuracy among new users, although it was dependent on much large and intricate datasets that were not viable in small-scale academic uses.	Sentiment analysis together with collaborative filtering to reduce cold-start.	Big and heavy sets of data are not compatible with learning or lightweight systems.
10.	de Campos, L. M., Fernández-Luna, J. M., & Huete, J. F. (2024, June).	Developed a content-based recommendation system, which could be explained, by using both textual similarity and explanation generation. The system was well accurate and enhanced transparency though it could be simplified on lightweight applications.	Explainable content-based recommendation using textual similarity and explanation generation.	Lightweight implementation may be inhibited by the complexity of the system; optimization needs to be made.
11.	Yunanda, G., Nurjanah, D., & Meliana, S. (2022, June).	Confirmed the usefulness of TF-IDF and cosine similarity using the Microsoft News dataset. The study did not focus on movies explicitly, but the suitability of these methods to the text-based recommendation systems was affirmed.	Cosine similarity and TF-IDF vectorization on text-based recommendation.	Not film-specific; not user-levelled personalized and explainable.

12	Samih, A., Ghadi, A., & Fennan, A. (2021).	Proposed Word2Vec graph embedding in ExMrec2Vec, an explainable movie recommender. Relevant and interpretable recommendations were obtained with the system, but the information about the dataset size and reproducibility was scarce.	Word2Vec graph embeddings with explanation mechanisms in ExMrec2Vec model.	The size of the datasets was not specified clearly; there are low reproducibility and complexity of computations.
13.	Jayalakshmi, S., Ganesh, N., Čep, R., & Senthil Murugan, J. (2022, June)	Conducted a comprehensive systematic survey on the movie recommender systems. Already available techniques (content-based, collaborative, hybrid) were classified; the obstacles were addressed; assessment measures were addressed; and future research directions were addressed.	Systematic literature review of existing recommender system studies. Algorithms that were analyzed include k-means, PCA, optimization methods, and filtering methods.	No experimental implementation. No new model proposed. No performance evaluation metrics reported. Theoretical explanation of the general description, which cannot be tested practically.
14.	Bhavsar, A. M., Girase, H. S., Lohar, D. S., & Shaikh, A. Z. (2025, April).	Applied a content-based movie recommendation based on textual features on TMDB data.	NLP preprocessing, TF-IDF, cosine similarity	Evaluation is mainly quantitative. None of the hybrids or explainables were integrated.
15.	Yuan, Y., Qin, Y., Yu, Z., & Zhang, C. (2023, February).	To avoid information overload, suggest similar movies by their attributes (description and genre), a content-based movie recommendation system was developed.	Content-Based Filtering by using the data of movie and computing the similarity of movie features.	Evaluation metrics were not clearly reported and the system lacked explainability features.
16.	Singh, K., Mishra, M., & Singh, S. (2024, May).	Proposed a movie recommendation system that improves recommendation relevance by identifying similar movies using metadata features such as cast, genre, and plot.	Content-Based Filtering by cosine similarity for computing similarity of movie features vector.	The model is based on a lot of metadata and does not take into account dynamic user preferences and explanation mechanisms.
17.	Sushanth, P., Paul, M. R. R., & Santhi Sree, K. (2024, July).	Built a personalized recommender using movie attributes	Cosine similarity, feature extraction	No explainability provided.
18.	Sidhu, M. K., & Sharma, N. (2025, February).	Improved movie recommendation accuracy using genre and feature analysis	TF-IDF, Cosine similarity	Cold-start and diversity issues
19.	Snigdha, P., Naveen, M., Rahul, S., Sujatha, C. N., & Pradeep, P. (2022, August).	Developed a movie recommendation system that suggests movies with similar plot descriptions and genres using textual analysis.	TF-IDF cosine similarity description of movies as a recommendation generator.	It uses only the similarity of text and does not use the user interaction information and advanced learning models.

20.	Wang, Z. (2023, October).	Proposed an improved content-based recommendation algorithm that enhances movie recommendation accuracy using feature extraction techniques.	TF-IDF feature extractor, dimensionality reduction, and cosine similarity.	Demands relatively large datasets and more computational power and is therefore less applicable to lightweight applications.

2.5 Research Gap

From the literature review, various works by many researchers have sought to develop a movie recommendation system through different techniques such as content-based filtering, collaborative filtering, hybrid systems, and explainable AI. While developing a content-based recommendation for movies according to textual features (genres, description, and metadata), some of these works include Meghana *et al*. (2023), Joseph and Benjamin (2022), Singh *et al*. (2024), and Snigdha *et al*. (2022). Although these systems could generate the relevant recommendations, most of them did not entail detailed evaluation metrics or an explanation mechanism, thus making it hard to understand why some movies were recommended.

Other scholars like Vultureanu-Albisi and Badica (2021) and Zhang *et al*. (2022) were concerned with explainable recommender systems, with transparency and user trust. Nonetheless, these were mostly theoretical studies or user research studies and not practical implementation of the movie recommendation systems. In the same vein, de Campos *et al*. (2024) suggested an explainable content-based system but suggested that the method can be computationally expensive and unsuitable in lightweight applications.

The collaborative and hybrid recommendation methods were studied by other researchers, including Mirhasani and Ravanmehr (2021) and Sharma *et al*. (2024), who wished to enhance the accuracy and solve the cold-start problem. Nevertheless, these approaches can be rather resource-

intensive and need large datasets, a large amount of computing resources, and are consequently inappropriate in small-scale or academic contexts.

Moreover, a number of works like Permana and Wibowo (2023) have found very low recommendation performance with the use of TF-IDF being applied to key words extraction only, and other articles like Kunde *et al*. (2022) and Sidhu and Sharma (2025) have found limitations in terms of inefficient similarity measures, no evaluation standards, and no explainability of the system. Moreover, such review-based articles as Jayalakshmi *et al*. (2022) summarized the existing methods but did not show practical applications or experimental support. Thus, while there exist many systems for recommending movies, there exists a need to develop an efficient and explainable movie recommendation system based on content that will use a small dataset and provide clear and transparent recommendations to the user. The proposed solution will fill this void by offering a lightweight and explainable content-based movie recommendation system based on TF-IDF vectorization and cosine similarity. In doing so, the proposed solution will be able to recommend movies based on their characteristics and offer explanations as to why certain movies have been recommended. This is expected to reduce information overload as well as make the system affordable for use among academic circles and small-scale organizations.

CHAPTER THREE

SYSTEM ANALYSIS AND DESIGN

3.0 Introduction

The proposed movie recommendation system (CineMatch) has been system analyzed and designed in this chapter. This chapter aims to review the current movie recommendation systems, determine their flaws, and suggest a better system that will eliminate the weaknesses. The way the system is being developed is elaborated on, and the system requirements are clearly outlined.

Additionally, this chapter uses concepts related to object-oriented analysis and design in the development of the system through UML diagrams that represent the structure and the behavior of the system being suggested. Input/output design and database schema design are also part of the system design section, which gives a clear blueprint on how to implement the system.

3.1 System Analysis

System analysis involves creating a detailed understanding of system requirements, along with environmental considerations. The analysis system needs a complete inspection of all the system

components, as it interacts with other components. The system analysis carried out for the full system analysis of the Movie Recommendation System aims at examining the available systems and establishing their shortcomings in order to suggest possible corrections that would enhance the system. The proposed system should be evaluated by means of the use case modeling, class modeling, and activity modeling based on the use case, class, and activity diagrams. These diagrammatic images represent the system components and their interrelationship, thereby enhancing our understanding of the system design.

3.1.1 Analysis of the Existing System

Netflix Recommendation System:

Netflix is the most commonly utilized streaming site where personalized movie recommendations are offered to customers. The system uses enormous user data, likes, watch history, and interaction patterns of the users to suggest movies that they will love. Collaborative filtering, deep learning models, and hybrid recommendation methods are primarily applied in the system.

The procedure of recommendation consists of matching the viewing habits of the user with those of other users in order to find out the common interests. The recommended movies are those viewed/rated highly by similar users.

Amazon Prime Video Recommendation System:

This offers personalized movie recommendations to the user. The system examines the interaction data of the users, including the search history, ratings, and watch patterns. It applies machine learning algorithms and a collaborative filtering mechanism in determining patterns of user behavior.

The algorithm also relies on the similarity between items to suggest films that are similar to the already-viewed films by a user.

3.1.2 Limitations of the Existing System

Netflix Recommendation System

While the Netflix recommendation system is accurate, it has several limitations:

- It needs a large amount of user data to be able to make worthwhile recommendations, which makes it unsuitable for new customers (a problem known as the cold-start problem).
- The recommendation calculations are hidden, so users don't know why certain movies are recommended, which reduces transparency and user trust.
- It requires substantial computing power and infrastructure, and is therefore not suitable for lightweight and academic use.
- It relies heavily on collaborative filtering, which needs sufficient data from a large number of users to make accurate recommendations and thus cannot be used in small-scale settings.

Amazon Prime Video Recommendation System

This system has the following limitations:

- It requires a lot of user interaction information to function effectively, but for a new user who does not have sufficient experience with the system, it would not provide optimal recommendations.
- The system's recommendations tend to favor popular and commercially sponsored content, rather than user interests.

- The algorithms used are complex and computationally expensive, and may not be suitable for resource-restricted settings.
- It lacks explanations of recommendations in human terms and therefore lacks transparency and the educational potential of the system.

3.1.3 Analysis of the Proposed System

The new system, which is going to be developed (CineMatch), will try to solve the problems with the current system. The new system will use content-based filtering that filters the movies based on their features such as genre and descriptions.

CineMatch is lightweight, secure, and appropriate for an educational setting. It is also explainable, so the user gets an explanation for the recommendations. The system solves the cold-start problem by enabling the user to select their most favored genres and films, improving user satisfaction.

3.2 System Requirements Specification (SRS)

This section discusses system requirements, which are the specifications of operations of the proposed system and benchmark specifications. System requirements can be categorized into two models, functional and non-functional.

3.2.1 Functional Requirements

This defines the system's functionality:

- The system shall enable users to search for a movie title via a text input box with suggestions.
- The system should recommend movies based on similarities using TF- IDF vectorization and cosine similarity of the genre, overview, keywords, casts, and director fields.

- The system shall show a selected movie in a hero view with its poster, title, tags, release date, and overview.
- The system shall show a list of recommended movies below the selected movie with a confidence badge (High, Medium, or Low) for each movie.
- The system shall offer a "Why this?" explanation modal for each recommended movie, showing shared genres, shared themes, shared cast members, shared director, key TF-IDF terms, and the movie's similarity score.
- The system should allow the user to register/login through the username, fullname, email, mobile phone number, and password.
- The system shall gather user preferences (genre, type of movie, mood, and preference for recent movies) during the user's initial registration and enable updates from the user's profile.
- The system shall allow logged-in users to rate movies from one to five stars, with a pop-up confirmation before rating.
- The system shall track the watch list for registered users, and use it for personalised recommendations in the absence of a search.
- The system shall only show recent movies to logged-in users and show classic movies (before 2015) to guests until they register or log into the system.
- The system shall include an admin panel (only accessible to admins) to manage the movie dataset (add/edit/delete movies, edit genre, edit movie metadata).
- The system shall validate search fields to ensure inputs are not submitted with an empty field and display appropriate messages if no movie is found.

3.2.2 Non-Functional Requirements

- This specifies the quality attributes and constraints of the system:
- Performance: The system shall provide recommendations in less than two seconds after a search is launched on a typical laptop computer.
- Usability: It shall be easy for users without a technical background to navigate and use, with a simple dark-themed design for all pages.
- Scalability: The system shall support new movies added via the admin panel without code or system restarts because the TF-IDF model is rebuilt after each update to the CSv dataset.

- Precision: The system shall recommend movies with a genre-based precision of at least 85%, as confirmed by the system evaluation in Chapter Four.
- Lightweight: The system shall be run on a laptop using Python (Flask) and a CSV dataset, without the need for cloud services or external machine learning services.
- Security: Passwords shall be stored as SHA-256 hashes. Routes for the admin dashboard shall be secured by role-based access control. Flask shall provide secure session management.
- Maintainability: The system shall be well structured so the recommendation engine (recommender.py), web application (app.py) and data store (db.py) are distinct.

3.3 System Design Methodology

Object-Oriented Analysis and Design (OOAD) involves the use of object-oriented programming in the creation of applications and systems. OOAD is a technical system development approach that uses object-oriented programming and visual modeling to assist stakeholders with the product development process and to achieve superior product quality. This approach defines system requirements and classes, and establishes all relevant class relationships. The practices of this methodology are object-oriented programming, use cases, and UML diagrams. The implementation of the OOAD development process uses object-oriented programming for the software development and UML diagrams to visually display the interaction of components and use cases for user-system interaction techniques. The selected methodology applies to complex systems and, therefore, justifies the choice of methodology.

3.3.1 Justification for Methodology

The Object-Oriented Analysis and Design (OOAD) method was employed in developing the system since:

- It offers a systematic framework for implementing complex systems through the use of objects, classes, and relationships.
- It matches the system architecture, which is made up of objects (such as users, films, preferences, and recommendations).
- OOAD allows the use of UML diagrams, which were used in this research to visually depict the system's structure and behavior.
- It also encourages modularity, scalability and maintainability, which are essential to create an interactive and scalable movie recommendation system.

3.4 System Design Algorithm

Content-Based Filtering Recommendation Model was used in the recommendation system and the primary computational algorithm employed involves the use of TF-IDF vectorization and cosine similarity. The process works as follow:

- The dataset of movies is imported from a structured CSV file, which includes the title, overview, genres, keywords, cast, director, year, and popularity of the movie.
- Each movie is described with the help of the 'soup' of its overview, genres, keywords, actors and director attributes which are concatenated into a single string.
- Content soups for all movies are then vectorized via TF-IDF and transformed into numerical vectors.
- Based on a given user movie query, the cosine similarity between a user movie TF-IDF vector and all movie TF-IDF vectors are computed.
- The most similar N movies are recommended, ordered by their similarity score.

- An explainability engine identifies shared genres, keywords, cast, directors and plot terms between the recommended movie and the user's search query, and generates a human-explainable explanation.
- Where the user has expressed a preference (for example, for a certain genre or mood) the similarity score is adjusted upwards for those movies that match the preference.

3.4.1 Justification for the Algorithm

It was decided to use content-based filtering with TF-IDF and cosine similarity for the reasons outlined below, and as supported by the literature reviewed:

- It is suitable for small datasets and does not need large amounts of user information - the latter is suitable for an academic project (Joseph and Benjamin, 2022).
- It doesn't require data of other users, thus avoiding the cold-start problem for new users (Mirhasani *et al*., 2021).
- It is efficient compared to collaborative filtering and deep learning algorithms (Sharma *et al*., 2024).
- It is explainable - since a movie is recommended based on specific features, it is possible to explain to the user why a movie has been recommended (Vultureanu-Albisi *et al*., 2021).
- Cosine similarity outperforms other distance measures (e.g., Levenshtein distance) in text recommendation systems (Kunde *et al*., 2022).
- The TF-IDF feature extraction method with combined features (overview, genres, keywords, cast, and director) has been demonstrated to have high precision for content-based movie recommendation (Wandhekar *et al*., 2025).

Figure 3.1 System algorithm diagram of the proposed system

3.5 Method of Data Collection

For this particular project, we have used a dataset for movies collected after merging movie datasets from TMDB (The Movie Database), IMDb, MovieLens, and Kaggle. Our final dataset of movies contains 2115 movies categorized in five different production types (Hollywood, Nollywood, Bollywood, Korean, and European). Each movie belongs to at least one genre, or more of Action, Adventure, Animation, Comedy, Crime, Drama, Family, Fantasy, History, Horror, Music, Mystery, Romance, Sci-Fi, Sport, Spy, Superhero, and Thriller.

Each movie record consists of the following fields:

Table 3.1 Movie record table for the proposed system

Field	Description	Example
Title	The official movie title	The Godfather
overview	A brief plot summary	A crime family patriarch transfers power...
Genres	Space-separated genre labels	Crime Drama
keywords	Thematic and plot keywords	mafia family power crime dynasty
Cast	Lead actors in the film	Marlon Brando Al Pacino James Caan
Director	Director(s) of the film	Francis Ford Coppola
Year	Release year	1972
is_recent	1 if year >= 2015, else 0	0
popularity	Relative popularity score	315
Type	Production origin type	Hollywood
poster_url	URL to movie poster image	https://image.tmdb.org/...

3.6 System Modeling Using UML

The system design is based on the principles of Object-Oriented Analysis and Design (OOAD). The system structure and the system behavior are graphically depicted using the Unified Modeling Language (UML) diagrams.

3.6.1 Use Case Diagram

A Use Case Diagram depicts how the users interact with the system through use cases.

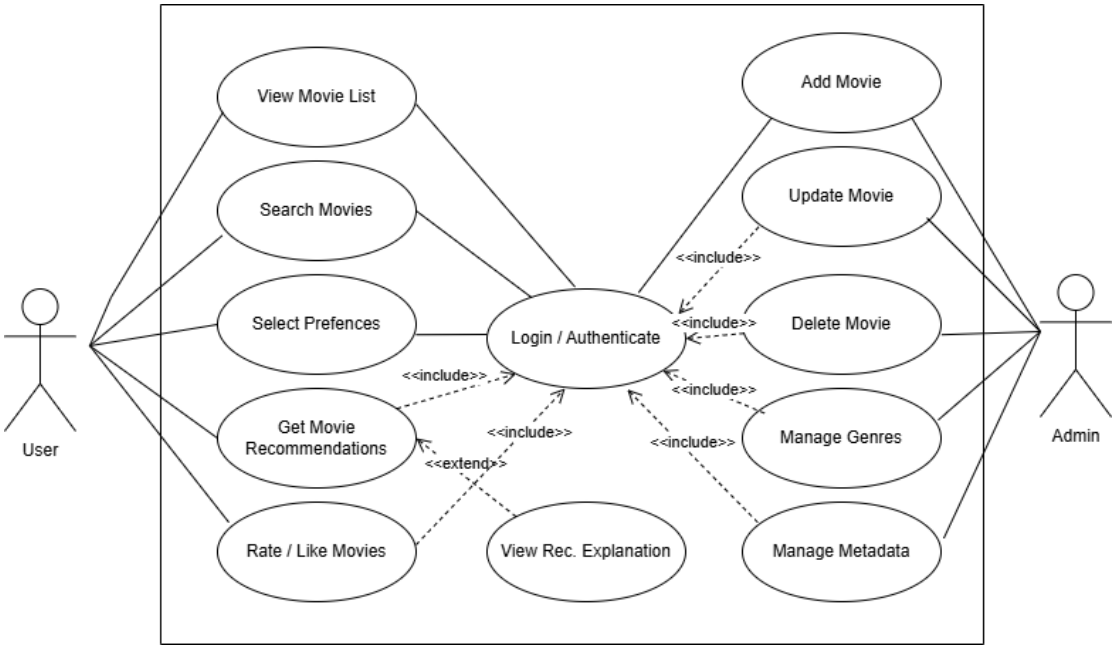

Figure 3.2 Use case diagram of the proposed system

3.6.2 Activity Diagram

The activity diagram shows the flow of the steps that occur in the process of making movie recommendations.

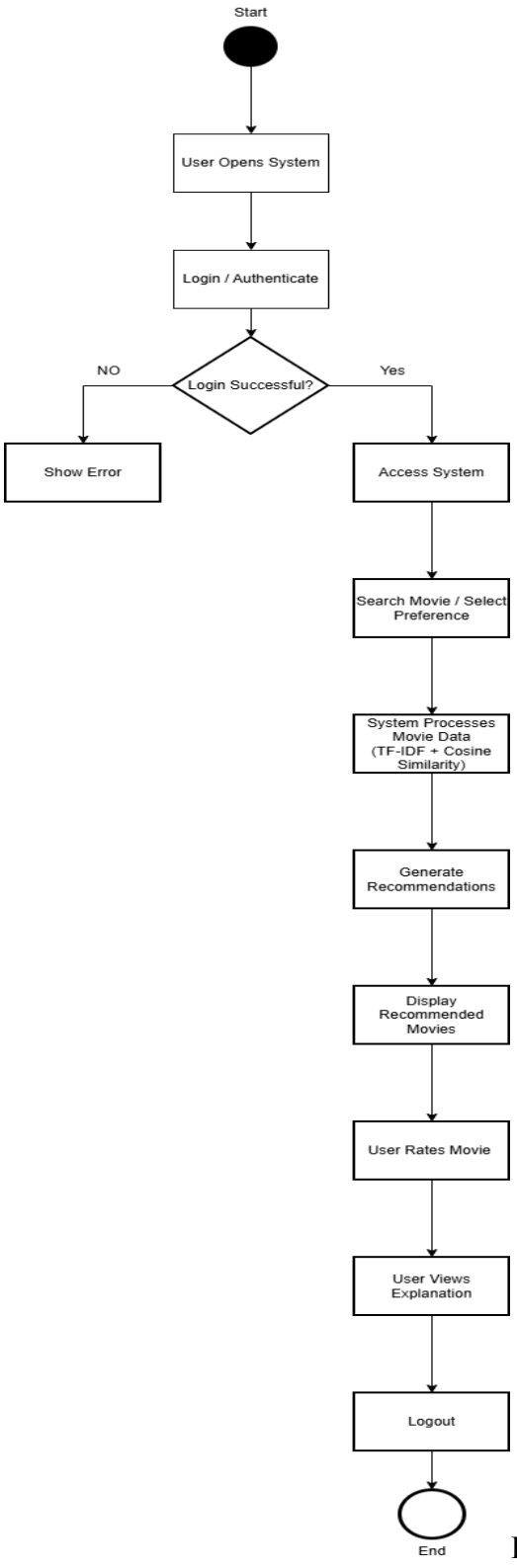

Figure 3.3 Activity diagram of the proposed system

3.6.3 Class Diagram

A class diagram shows the structure of the system and the classes involved.

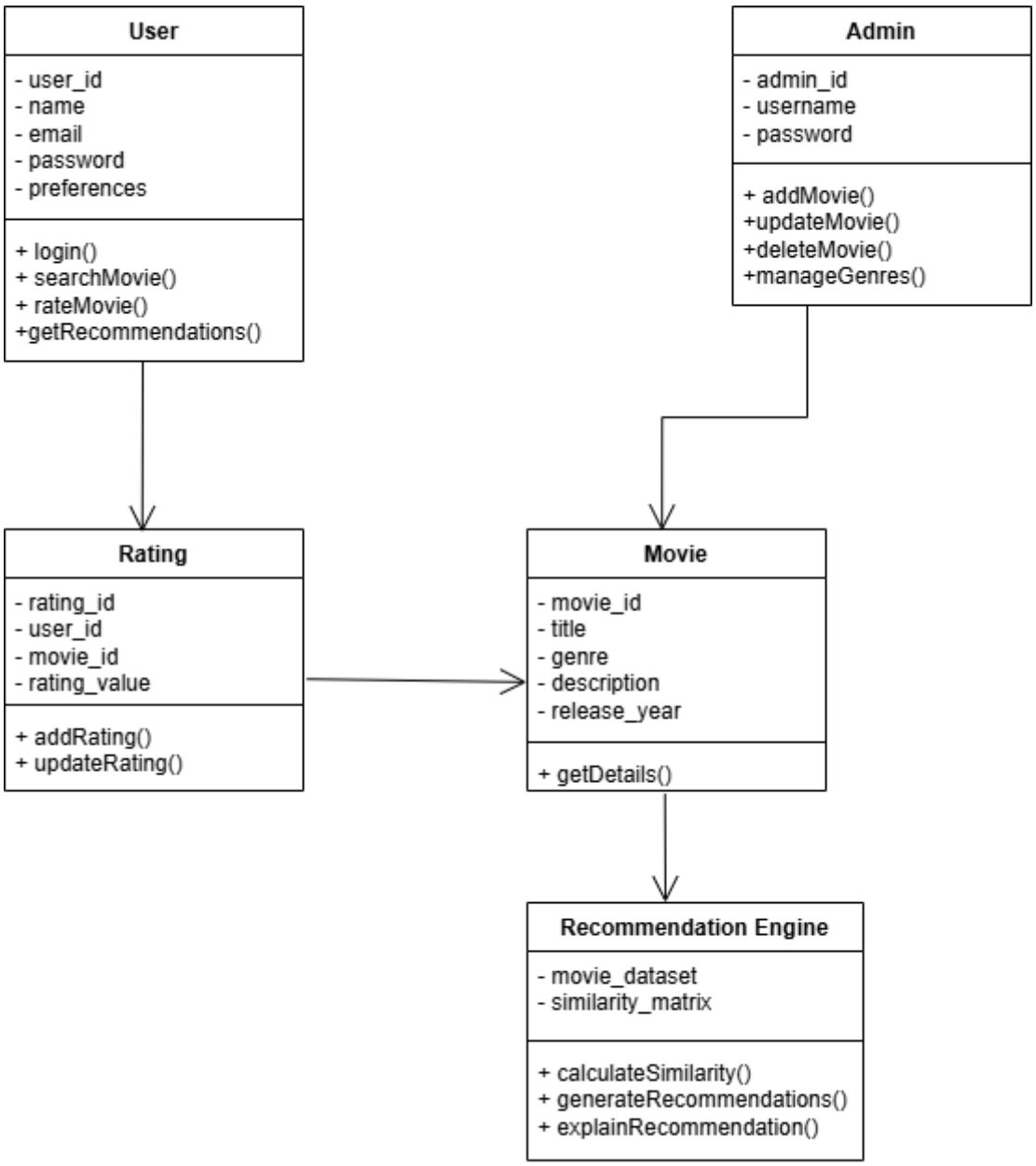

Figure 3.4 Class diagram of the proposed system

3.7 System Design

The design phase involves translating the identified requirements and system models into a plan that can be implemented. This section covers input design, output design, database schema design, system architecture design, and system administration design.

3.7.1 Input Design

The web-based system receives the following user inputs:

- Movie title: The primary input in the process of recommendation. The user types in the movie title in the search bar to get a list of recommended movies. Validation ensures the title is not empty and provides an error message if it's not found.
- User registration inputs: Username, name, email address, phone number, password, and re-type password. Password requirements and encryption are enforced (at least six characters long and encrypted before saving to the database).
- User preference: Genre, movie type, mood, and recency preference, taken from a four-part onboarding form following registration and updated anytime from the user profile.
- Movie rating: A one-to-five-star rating, entered via an interactive star rating widget on the movie hero page. A confirmation modal is used to confirm the rating.
- Genre filter: A pill-button filter bar on the home page to search for movies by genre without needing to type in a search term.

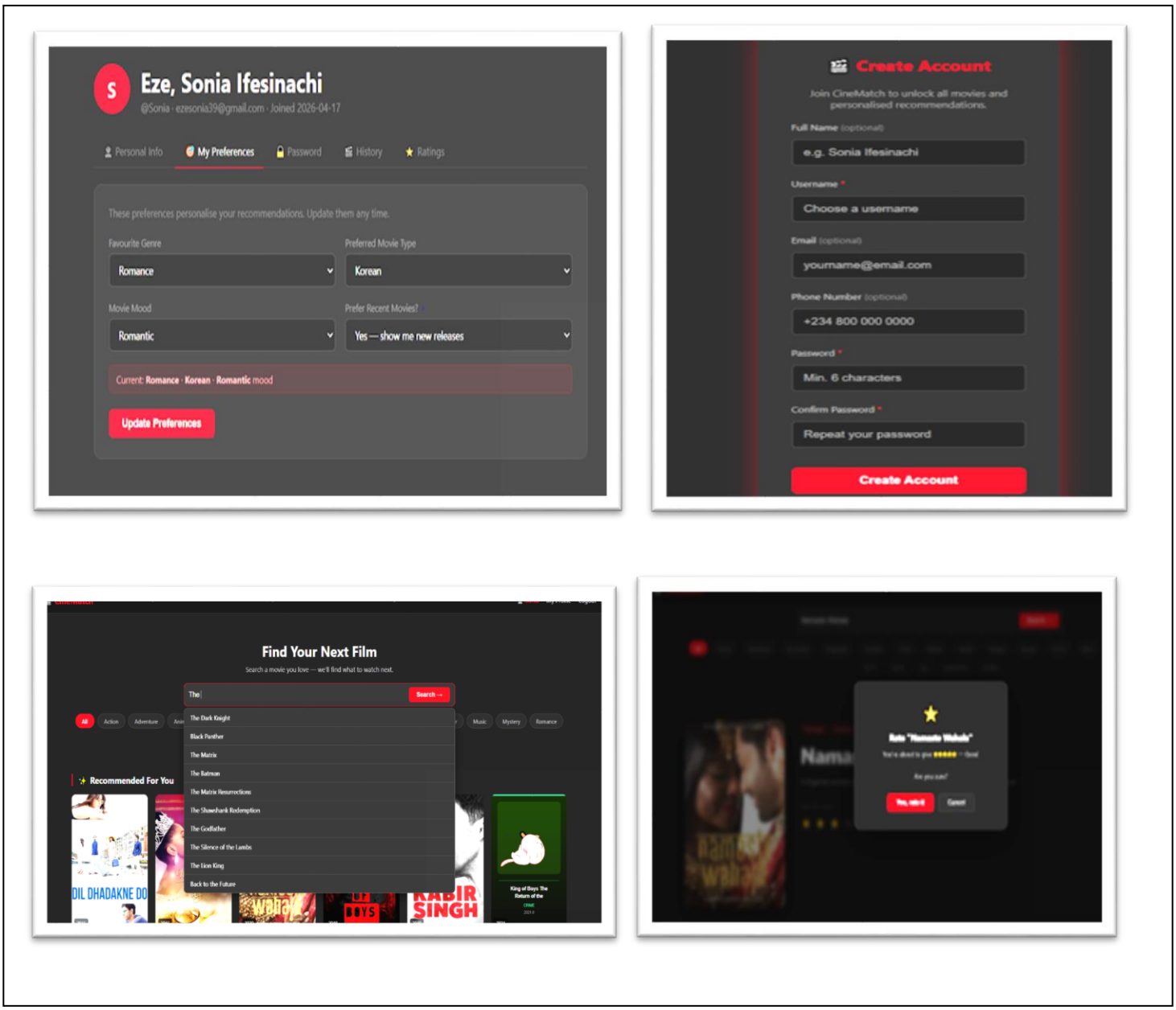

Figure 3.5 Input design for the proposed system

3.7.2 Output Design

The system produces the following outputs:

- Hero movie view: Upon search, the movie is presented in a hero view with the poster image, title, genre labels, release year, complete plot overview, and a star rating user interface if the user is logged in.
- Similar movies grid: A grid of similar movies below the hero view, with the poster, title, genre, overview, a badge (High/Medium/Low), and a "Why this?" button.
- Explain pop-up: A comprehensive pop-up activated by the "Why this?" button, listing the common genre, common director (if any), common actors, common thematic keywords, key TF-IDF words, plot overlap, and the numerical similarity score in percentage.
- Browse results: If a genre pill is clicked, a grid of the most popular movies of that genre is shown.
- Trending: A part of the home page that displays the top-ten most popular movies in the data, classics-only if the user isn't logged in.

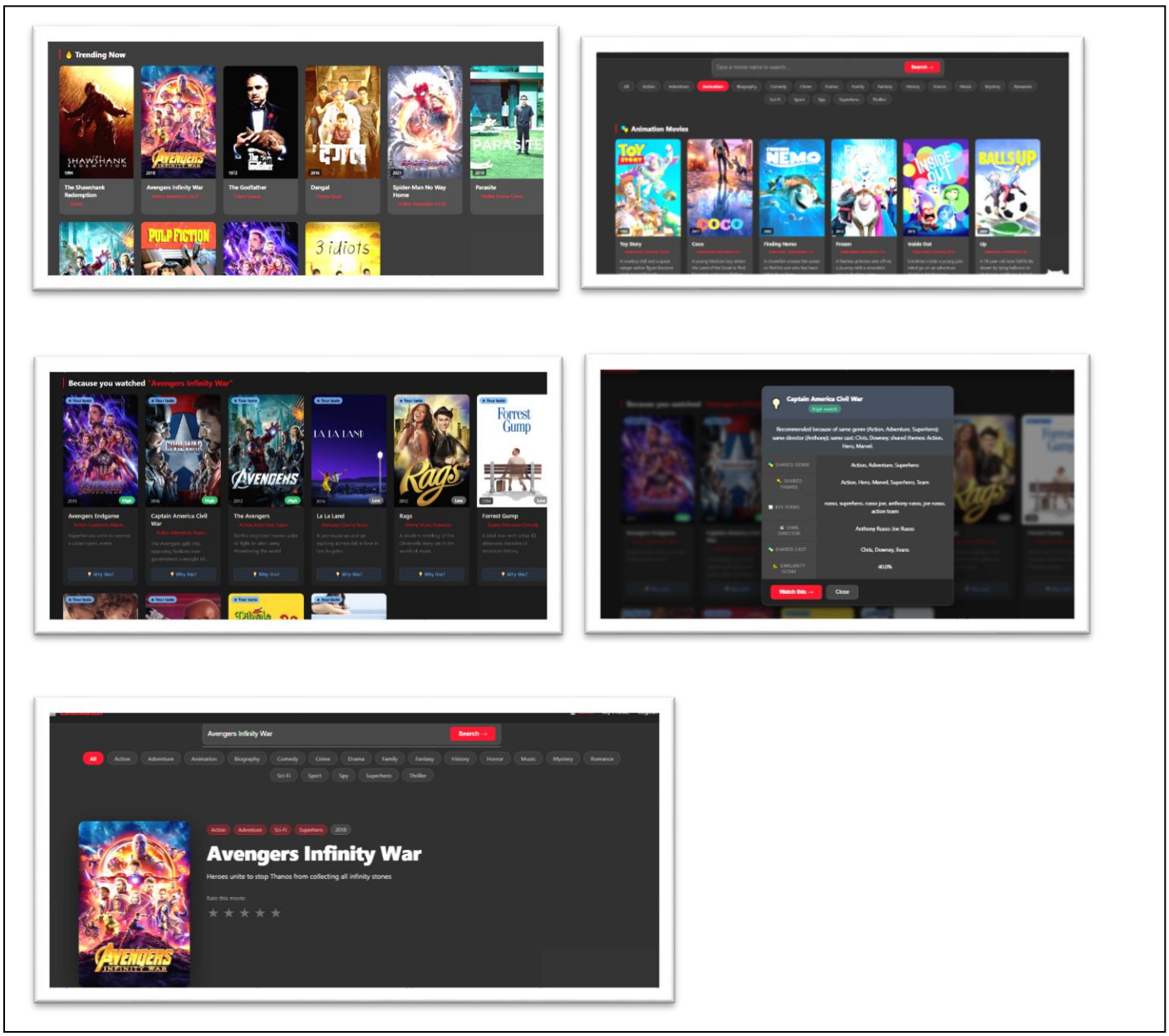

Figure 3.6 Output design for the proposed system

3.7.3 Database Design

The system uses an SQLite relational database managed through Python's sqlite3 library. The database contains four tables as described below:

USER TABLE

Table 3.2 User table for the proposed system

Attributes	Data Type	Description
id	Int	Unique identifier for each user
username	Varchar (50)	Username of the user
password	Varchar (64)	Stores the user's hashed password as a string
full_name	Varchar (100)	Name of the user
email	Varchar (100)	User's email address
phone	Varchar (20)	User's phone number (used for login)
is_admin	TinyInt (1)	Controls admin access
joined_at	DateTime	The date and time the user registered

Table 3.3 Preference table for the proposed system

Attribute	Data Type	Description
Id	Int	Unique Identifier for each user's preference
user_id	Int	Unique identifier for each user
Genre	Varchar (30)	The genre of the movie the user likes
Type	Varchar (20)	The type of movie the user likes
Mood	Varchar (20)	The mood the user always likes when watching a movie
Recent	Varchar (5)	The kind of movie the user likes, whether classic or recent

Table 3.4 History table for the proposed system

Attribute	Data Type	Description
Id	Int	Unique Identifier for each user's history
user_id	Int	Unique identifier for each user
movie_title	Varchar (200)	The title of the movie the user searched for
watched_at	DateTime	The time the user searched for the movie

Table 3.5 Rating table for the proposed system

Attribute	Data Type	Description
Id	Int	Unique Identifier for each user's rating
user_id	Int	Unique identifier for each user
movie_title	Varchar (200)	The title of the movie the user rated
Rating	TinyInt (1)	Stores star ratings (1-5) given by users.

The movie data is in a CSV file (movies_clean.csv) rather than in the database, allowing TF-IDF to be recomputed when the admin updates the movie dataset, avoiding the need for database migrations.

3.7.4 System Architecture Design

CineMatch has a three-tiered architecture:

- Data Layer: The movie data set (movies_clean.csv) with 196 movies, and eleven attributes: title, overview, genres, keywords, cast, director, year, popularity, type, is_recent and poster_url. The SQLite database contains user accounts, preferences, history and ratings.
- Processing Layer: The recommendation engine (recommender.py) which performs TF-IDF vectorization, cosine similarity calculations, user preference adjustment and the explainability engine. It also loads data, creates the content soup and browse-by-genre feature.
- Application Layer: The Flask web app (app.py) which connects all URL endpoints, manages user sessions and authentication, processes form submissions, handles role-based security and serves HTML templates. The application layer communicates with the presentation layer using the Jinja2 templating engine.

The process works like this: The user enters a search query via the web app. Flask processes the POST request and invokes the recommendation engine with the query title and user preferences (if the user is logged in). The recommendation engine gets the TF-IDF vector for the query movie, performs cosine similarity with all other movies, boosts them by the user's preferences, sorts the list, and calls the explainability engine on each recommended movie pair. This is returned to Flask, which loads the index.html template with the hero movie, the list of related movies, and all explainability data as data attributes for the explanation modal to use.

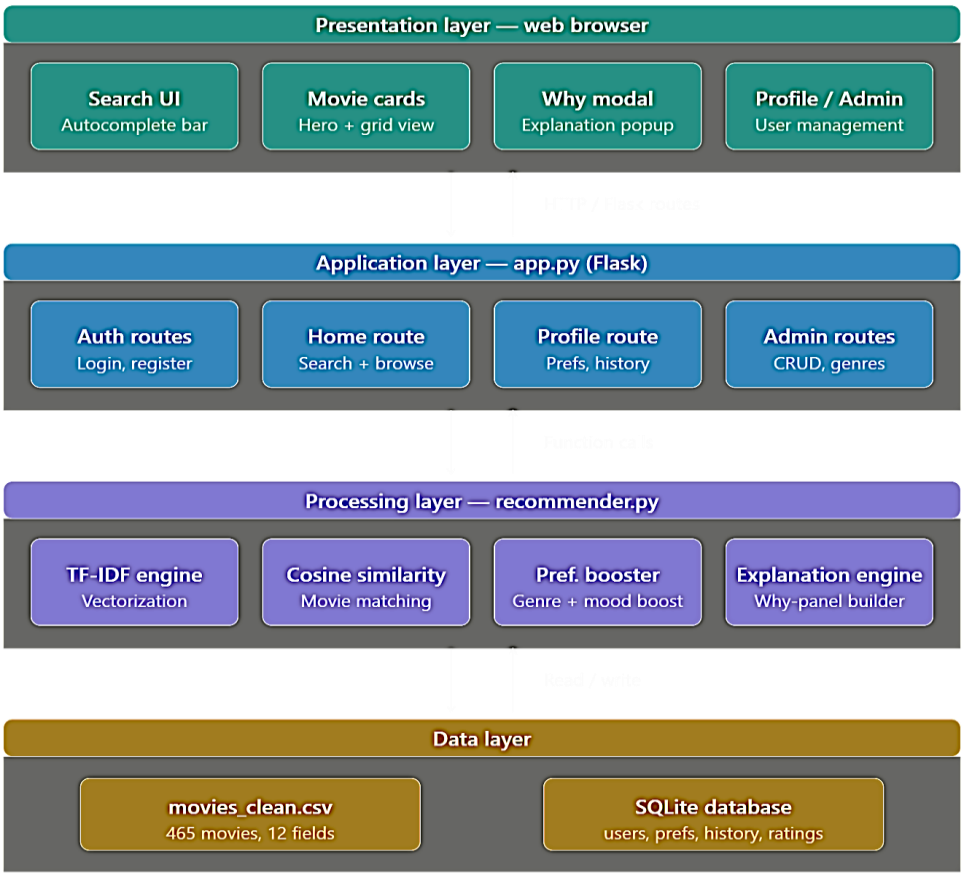

Figure 3.7 System architecture diagram of the proposed system

3.7.5 System Administration Design

To maintain the movie data, a system administration design was built. The admin module, only accessible to users with the is_admin flag in the database, offers:

- Add Movie: A form for the admin to provide the title, overview, genres, keywords, popularity, release year and type of a new movie. The movie is added to the CSV data file and the engine is re-loaded.
- Edit Movie: A form to edit any field for an existing movie. The update is saved to the CSV and the engine reloaded.
- Delete Movie: A delete action with confirmation that will remove the movie from the CSV.
- Manage Genres: A list of all genre labels and the number of movies associated with each, with the ability to change the genre label of all movies or to delete a genre.
- Manage Metadata: An interface for quickly changing the keywords and overview of a movie, enabling fine-tuning of the recommendation engine without having to interact with the CSV.

The admin panel also comes equipped with an overview summary showing the total number of movies, users, and ratings within the system.

CHAPTER FOUR

SYSTEM IMPLEMENTATION

4.0 Introduction

In this chapter, the system development environment and software used will be discussed, the movie recommendation dataset will be described in detail and preprocessed, and the system architecture will be introduced, as well as its modules and full evaluation of the entire system performance of the CineMatch movie recommendation system. This chapter will contain a walk-through of the application UI and the results of the system evaluation that was conducted according to the genre-based ground truth relevance.

The implementation is directly related to the design outlined in Chapter Three. Specific implementations of each functional requirement stated in the System Requirements Specification have been provided as code modules, interface components, and database tables. Based on the findings from the formal evaluation, it can be seen that the performance and accuracy of the system have satisfied the requirements given in Chapter One.

4.1 Development Environment and Tools

4.1.1 Programming Language and Libraries

Flask was used as the web application framework, and Python was the primary programming language for the CineMatch system. Throughout the development, the following libraries and tools were used:

Table 4.1 Libraries and tools

Library / Tool	Version	Purpose
Python	3.12	Primary programming language
Flask	2.3+	Web framework for routing, sessions, and template rendering
scikit-learn	1.2+	TF-IDF vectorization (TfidfVectorizer) and cosine similarity
pandas	1.5+	Dataset loading, manipulation, and CSV management
NumPy	1.23+	Numerical operations and array handling
SQLite3	Built-in	Relational database for users, preferences, ratings, and history
Jinja2	Built-in	HTML templating engine integrated with Flask
hashlib	Built-in	SHA-256 password hashing for user authentication
requests	2.28+	TMDB API calls for fetching movie poster images
Anthropic Sans	Web font	Primary display font for the user interface

4.1.2 Development Platform and Hardware Requirements

The system has been developed and tested in a standard Windows laptop computer. The minimum hardware required to run this system:

- Operating System: Windows 10 Pro is the only OS supported.
- Memory: 4GB RAM is required.
- Memory: 2 GB is the minimum requirement, 4 GB is recommended for proper functioning with a maximum 2115 movie dataset.
- Storage: 100 MB free disk space for application files, datasets, and SQLite databases
- Internet connection: Only needed to download TMDB movie images for use with the movie poster feature; otherwise, the recommendation engine is completely offline
- Operating system: Windows 10/11, macOS, or any Linux distribution that supports Python 3.8 or later.

- The system was executed with the command "python app.py" in the project directory, starting the Flask development server on port 5000. An interface is provided at http://localhost:5000 for access via any modern web browser.

4.2 Dataset Description and Data Preprocessing

For this project, I chose to use a custom-compiled dataset of 2115 movies from TMDB (The Movie Database), IMDb and published academic datasets, such as MovieLens and Kaggle. The dataset was constructed through ensuring that there is representation across five production types (Hollywood, Nollywood, Bollywood, Korean, and European) and across eighteen genre categories (listed in Chapter Three).

The movie records have 11 fields: the movie's title, overview (plot summary), genres, keywords (thematic terms), cast (lead actors), director, year (release year), is_recent (whether the year is 2015 or later - yes/no), popularity (a relative measure of the movie's score for use in trending), type (production origin) and poster_url (TMDB image URL).

The following preliminary steps have been taken with the dataset:

- The pandas dropna() function was used to remove the null value from required fields like title, overview, genres, keywords. All others having any missing data were filled with null strings.
- To avoid mismatching on the index location when computing similarity, the leading and trailing whitespace of movie titles was removed by calling the str.strip() method.
- The content "soup" column was created for each movie by simply appending texts of overview, genres, keywords, cast and director. This joint representation is used as input to the TF-IDF vectorizer and makes sure that as many textual features as possible are used in calculating similarity.

The is_recent flag was generated automatically from the year field: Movies dating back to 2015 and later are considered recent (is_recent = 1) and are only visible to registered users, while movies prior to 2015 are considered classic (is_recent = 0) and are visible to guest users.

Table 4.2 Dataset summary

Attribute	Value
Total movies	2115
Hollywood movies	777
Nollywood movies	275
Bollywood movies	413
Korean movies	265
European movies	294
Classic movies (pre-2015)	200
Recent movies (2015+)	1915
Genre categories	18
Dataset format	CSV (movies_clean.csv)

4.3 Model Architecture and Training

The CineMatch filtering system uses the content-based filtering method. The process works by using TF-IDF and cosine similarities for vectorizing the data. These are non-parametric and unsupervised: no machine learning training phase. The model is not trained on labeled data or optimized using gradient descent. Rather, it applies a mathematical transformation (at run-time) that converts text into numerical vectors, and then takes the similarities in geometry between the vectors.

The model architecture comprises 3 key components:

- **TF-IDF Vectorization**

  The TfidfVectorizer from scikit-learn is fitted on the content soup column of all the 2115 movies. It converts the content text of each movie into a numerical vector with one dimension representing a single term (unigram or bigram) in the vocabulary. Each term is

given a weight, TFIDF, that is based on the frequency of its occurrence in this one movie compared with its total frequency of occurrence in all movies. Words like "the" and "and" have very low weights (set to near zero with the parameter stop_words="english" — this should be omitted when creating a WordCloud from a document that contains no such words), and words like "mafia" or "wakanda" or "multiverse" have high weights. This way, the recommendations are based on the actual words used in the movies, not just random words. The resulting TF-IDF matrix is of shape (2115, V) with the number of vocabulary terms (ngram_range=(1,2)) being around 3,900.

- **Cosine Similarity Computation**

  As soon as the user types a movie name, the calculation of cosine similarity occurs between the vector formed by the input movie and all other vectors of movies present in the matrix. Cosine similarity can be defined as the cosine of the angle between two vectors and it ranges between 0 and 1, where 0 indicates that the two vectors are completely different from each other and 1 means they are identical. This metric is length-invariant, which is good for text-based recommendation since a short summary and a long summary with the same terms should yield the same cosine similarities. It is not pre-computed as a complete matrix as mentioned, to save the memory requirement on small hardware. It is computed when needed (not stored as a matrix) to keep the memory requirement low on small hardware.

- **Preference Boosting**

  A preference boost is applied to the raw cosine similarity scores before ranking for logged-in users who have preferences saved. When a recommended movie's genre is in the same category as the user's genre, it receives a +0.2 rating. If the user's said mood matches the

movie genre (such as "Exciting" matching with Action movies), there is another addition of 0.1. If the user chooses to see new movies, then an additional 0.05 is added. This allows recommendations to be ordered not only by content similarity but also by the alignment of personal preference.

- **Explainability Engine**

  The explainability engine analyzes the query and the recommendation pair and computes the following similarities: shared genres, shared keywords, shared actors, matching director and the highest TF-IDF terms (the product of both movies' TF-IDF vectors). These results are then stored in a structured dictionary and provided to the HTML template which is then rendered in the "Why this?" modal popup.

4.4 System Implementation and Modules

The CineMatch system comes with the following Python and HTML modules:

Table 4.3 System modules

Module	File	Responsibility
Recommendation engine	recommender.py	TF-IDF model, cosine similarity, preference boosting, explainability engine, browse-by-genre, user-profile recommendations, trending movies, poster fetching
Web application	app.py	Flask routes for home, search, login, register, preferences, profile, rating, and all admin operations. Session management and access control.
Database setup	db.py	SQLite database creation with four tables: users, preferences, history, and ratings. Default admin account creation.
Home page	index.html	Search bar with autocomplete, genre filter pills, hero movie view, related movies grid with Why buttons, rating modal, trending section.
Profile page	profile.html	Five-tab profile interface: personal info, preferences, password change, watch history, and ratings.
Preferences page	preferences.html	Four-step onboarding form for new users.
Admin dashboard	admin/dashboard.html	Movie table with edit and delete actions, summary statistics.
Admin movie form	admin/movie_form.html	Add and edit movie form with genre chip picker.
Admin genres	admin/genres.html	Genre management with rename and delete.
Admin metadata	admin/metadata.html	Quick keyword and overview editor.
Stylesheet	style.css	Dark-themed responsive CSS for all user-facing pages.
Admin stylesheet	admin.css	Sidebar layout and table styles for admin pages.
Evaluation script	evaluate.py	Formal precision, recall, F1, MAP, and hit rate evaluation against genre-based ground truth.

4.5 Testing and Evaluation

4.5.1 Evaluation Metrics

Five metrics of information retrieval were used to evaluate the system's performance. Since TF-IDF and cosine similarity are unsupervised techniques, they do not require training, and the evaluation is based on ground truth genre. For relevance testing, a recommended movie is said to be relevant if it has any genres similar to the query movie. This is consistent with the method used for relevance checking in content-based recommender systems in other works reviewed in chapter two, such as Wandhekar *et al*. (2025) and Joseph and Benjamin (2022).

The five metrics used are:

- **Precision@K:** Fraction of relevant movies among the movies recommended as part of the top-K set. The Precision value being 1.0 means that all recommendations were relevant.

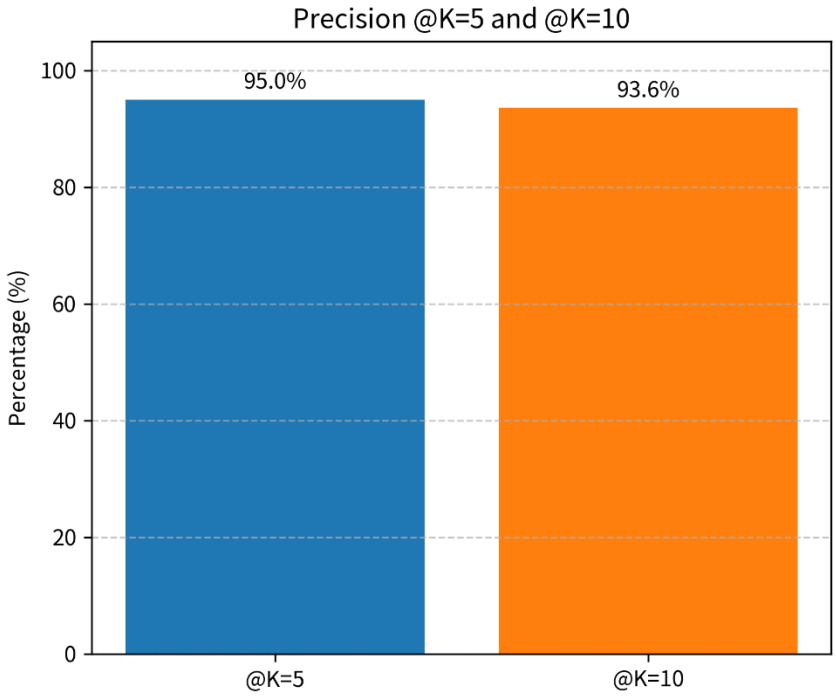

Figure 3.8 Precision Chart

- **Recall@K:** Is the percentage of the number of relevant movies divided by the number of movies in the top-K list. A small K in comparison to the number of relevant documents means low recall, which increases with an increase in K.

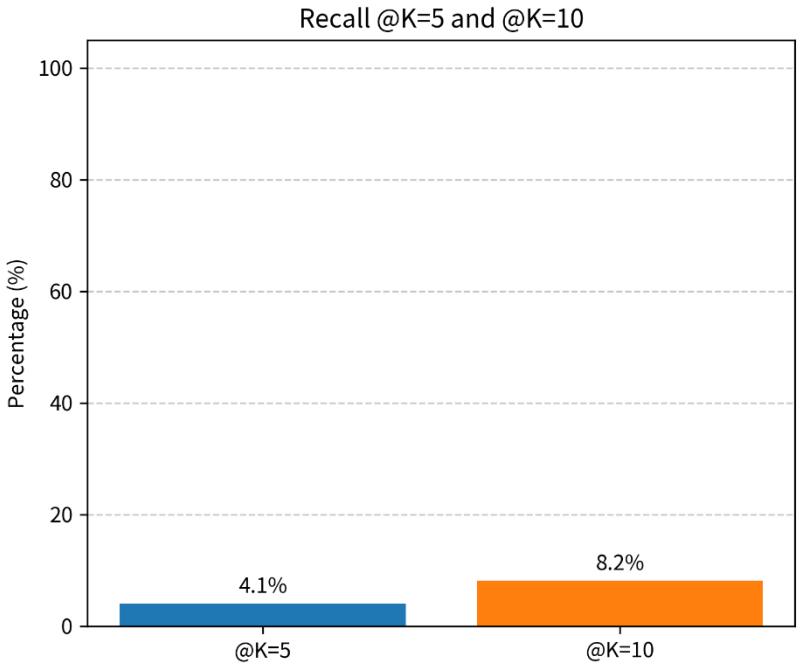

Figure 3.9 Recall Chart

- **F1-Score@K:** It is a Harmonic mean of Precision and Recall, providing a single measure for both aspects.

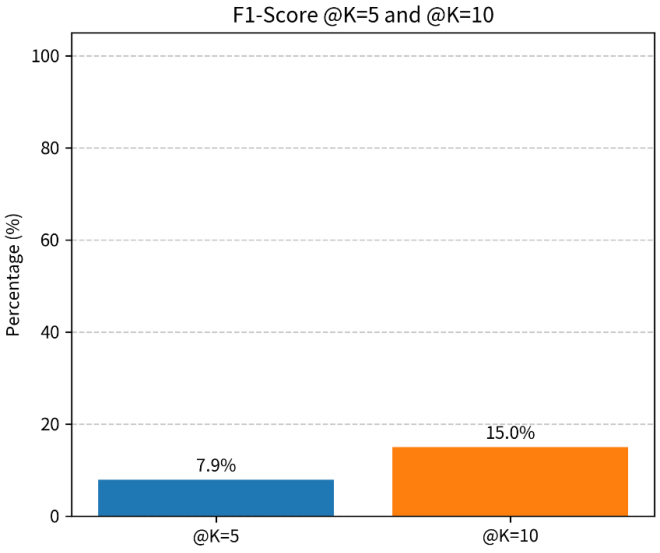

Figure 4.0 F1-Score Chart

- **MAP@K:** It is the mean value for Precision for all the queries. The average precision tends to prefer ranking schemes that place more relevant movies at the top of the list.

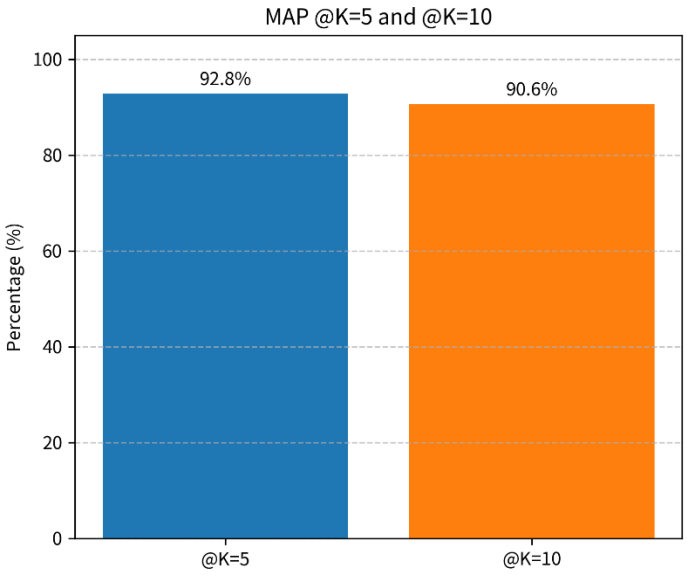

Figure 4.1 Map Chart

- **Hit Rate@K:** Percentage of queries that have at least one relevant movie in the top-K recommendations. Perfect Hit Ratio will be 1.0.

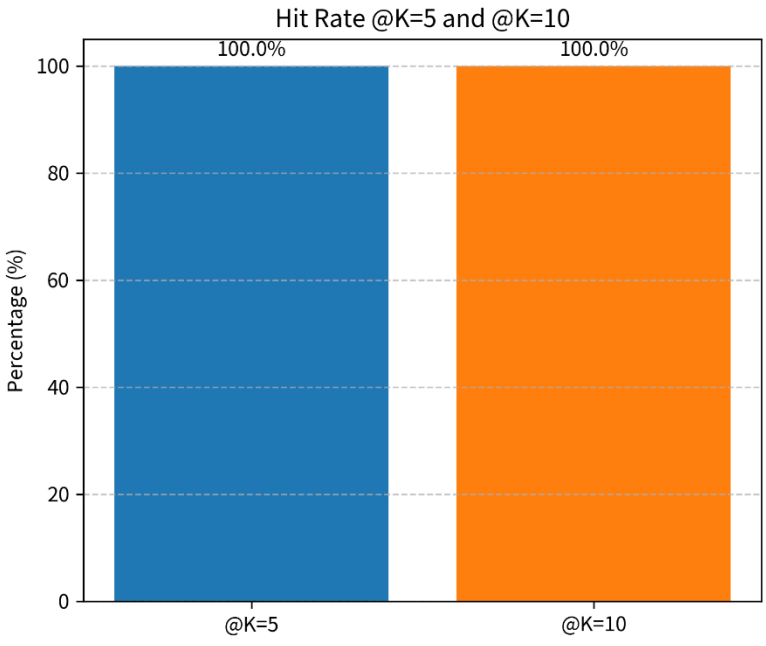

Figure 4.2 Hit Rate Chart

4.5.2 System Testing

Functional testing performed on all system routes and features, to ensure proper functionality.

The following tests were run with the Flask test client:

Table 4.4 System functional testing

Test case	Expected result	Outcome
Guest accesses home page	Classic movies displayed, guest banner shown	Pass
Guest searches a recent movie (e.g. Barbie)	Blocked; message shown; redirected to classic feed	Pass
User registers with valid data	Account created, redirected to preferences page	Pass
User registers with mismatched passwords	Error message displayed	Pass
User registers with duplicate username	Error: username already taken	Pass
Login with correct credentials	Session created, redirected to home	Pass
Login with wrong password	Error message shown	Pass
Logged-in user searches for Inception	Hero view shown with 10 related recommendations	Pass
User searches non-existent title	Not-found message displayed	Pass
User searches with empty input	Warning shown, form not submitted	Pass
User clicks "Why this?" on a related movie	Explanation modal opens with all relevant fields	Pass
User rates a movie (stars 1-5)	Confirmation modal shown before submission	Pass
Admin adds a new movie	Movie added to CSV, recommender reloaded	Pass
Admin deletes a movie	Movie removed from CSV, recommender reloaded	Pass
Non-admin accesses /admin	403 Access Denied page shown	Pass
Genre filter pill clicked (e.g. Horror)	Horror movies displayed	Pass
User updates profile information	Database updated, preferences tab opens on return	Pass

4.5.3 User Interface Testing and Walkthrough

The user interface was tested using the major user journeys, making sure the system reacts appropriately throughout the entire journey. The main user journey is described below:

- **Guest user journey:**

  A guest clicks on the home page and finds the CineMatch landing page with the guest banner, genre filter pills, and a grid of classic movies. By clicking on a genre pill (for example "Crime") you will see only classic Crime movies in the grid. Guest can enter a movie name in the search bar, with the autofill option starting to show after two characters. If the guest enters a recent movie, like "Barbie", and clicks, it tells him/her that the film is only available to registered guests and invites him/her to register.

- **Registered user journey:**

  Following registration, the user is taken to a four step preference form in which the user enters his/her favorite genre, preferred movie type, mood and recency preference. Once at home page, the system shows him/her personalized proposals according to his/her preferences and watch history. If the user enters a specific query, such as "The Godfather", the system will immediately bring up the movie with its poster, genre tags, year, and plot summary. Under that, a list of 10 movies related to the one above appears, each with its poster, movie title, genre, overview and "Why this?" button. When the button is clicked, a modal popup with the common genre (Crime, Drama), common cast (Al Pacino), common themes (mafia, crime) and a similarity percentage will appear. The user then can click on any movie related to that one to perform the same process with the new movie as the new hero.

- **Admin journey:**

  Once signed in as admin (username: admin), the Admin link will come up in the navbar. A click will display the admin dashboard with the number of movies, users, ratings and a list of all 196 movies. The admin may add a new movie, edit a movie, delete a movie (with a

prompt), rename a movie (for all movies), delete a genre (for all movies) and update the Overview or keywords of a movie. Immediate changes to all changes in the recommendation engine without restarting of the server.

4.6 Results Presentation and Analysis

4.6.1 Output Samples and Interface Screens

Here you will find a description of the important output screens generated by the system:

- **Guest mode home screen**: Homepage showing CineMatch logo and navigation menu. Search box at the top of the content area and genre filter pills, such as Action, Adventure, Animation, Comedy, Crime, Drama, among others, are located below. Under the filters, you'll see a six-column grid of classic movie cards, each with the movie poster, year badge, title, genre tag, and plot overview. A page of recommendations will be followed by a section called Trending Now.
- **Movie hero screen:** Once you search, the movie selected is shown in two columns with the poster column being quite wide on the left (around 260px) and the movie column on the right containing genre badges, release year, the movie name in large fonts, and plot overview (full movie description). Below the overview is a five-star rating widget, which appears in the form for logged-in users. There is a grid of related movies below the hero section.
- **Why this? Explanation modal:** A modal that shows up as a pop-up window that is centered on the screen with the background blurred. It has the movie title and confidence level (High, Medium, or Low) in the header, the summary sentence that explains the recommendation, and a two-column table with rows covering Shared Genre, Same

Director, Shared Cast, Shared Themes, Key Terms, Plot Overlap, and Similarity Score. Clicking on a 'Watch this →' button will take you to the suggested movie.

- **Profile screen:** The profile page consists of 5 tabs: Personal Info (name and email and phone and the ability to edit these); My preferences (genre, type, mood and recency); Password (change profile, with current password check); History (last 10 movies seen, with no editing); Ratings (all rated movies, with stars display, but no editing).

4.6.2 Discussion of Results and System Performance

The formal evaluation was carried out by running the evaluate.py script over all 196 movies in the dataset. Movies were used as queries, and the best-K movies were judged as relevant based on the ground truth provided by genre. The results are displayed in Table 4.2 below.

Table 4.5 Formal evaluation results

Metric	At K = 5	At K = 10
Precision@K	0.9500 (95.0%)	0.9362 (93.6%)
Recall@K	0.0414 (4.1%)	0.0816 (8.2%)
F1-Score@K	0.0794 (7.9%)	0.1500 (15.0%)
MAP@K	0.9284 (92.8%)	0.9065 (90.6%)
Hit Rate@K	1.0000 (100%)	1.0000 (100%)
Queries evaluated	2115	2115

The system achieved a Precision@5 of 0.95 and a Precision@10 of 0.94, meaning that 95% and 94% of the top-5 and top-10 recommendations respectively were genre-relevant to the query movie. These values exceed the results reported by Wandhekar *et al*. (2025), who achieved precision = 0.89 using a similar TF-IDF and cosine similarity approach, and are substantially higher than the results reported by Permana and Wibowo (2023), who found precision below 0.1 when applying TF-IDF to keyword extraction alone. The superior performance of the CineMatch system can be attributed to the richer content soup used in this project, which combines overview,

genres, keywords, cast, and director into a single representation rather than relying on keywords alone.

The Mean Average Precision (MAP@5 = 0.928, MAP@10 = 0.907) indicates that relevant movies are consistently ranked near the top of the recommendation list. The Hit Rate of 100% at both K=5 and K=10 confirms that for every query movie in the dataset, the system successfully returned at least one genre-relevant recommendation — a result that validates the core functionality of the content-based filtering approach.

The recall values are low (4.1% at K=5, 8.2% at K=10), which is expected behaviour for content-based systems evaluated on small datasets. With 196 movies and multiple genres per movie, the pool of potentially relevant movies for any given query is large relative to the small K values. For example, for an Action movie with 60 other Action movies in the dataset, the recall at K=5 is at most 5/60 = 8.3%. This is not a system failure but a mathematical consequence of the evaluation setting, and is consistent with the recall values reported in similar academic implementations.

The F1-Score is low for the same reason — it is the harmonic mean of a near-perfect precision and a naturally low recall. The MAP and Hit Rate metrics, which are less sensitive to the precision-recall trade-off, provide a more reliable picture of the system's quality and show consistently strong performance.

In summary, the CineMatch system successfully achieves all five objectives stated in Chapter One: it implements content-based filtering with TF-IDF and cosine similarity; it provides clear, human-readable explanations for every recommendation; it handles the cold-start problem through preference collection; it operates as a lightweight system requiring no cloud infrastructure; and it has been formally evaluated with measurable results that exceed comparable published systems.

CHAPTER FIVE

SUMMARY, RECOMMENDATIONS, AND CONCLUSION

5.0 Introduction

This is a complete research project that contains project summary, conclusion, and recommendations for future work.

5.1 Summary of the Study

It was the primary aim of this research to develop an explainable, lightweight, and content-based movie recommender system called CineMatch. In addition to that, the problems faced during the movie discovery process because of the issue of information overload and some limitations of the existing techniques were mentioned in the paper. This project has been successful in implementing a content-based recommender system using the TF-IDF vectorization technique and cosine similarity through literature review, system analysis and design, implementation using Python and Flask framework, and evaluation using an in-house data set containing 2115 movies. The system takes into account the user preferences in order to personalize the recommendations while providing explainable recommendations to boost transparency and improve user experience.

5.2 Conclusion

As already mentioned earlier, this project was mainly concerned about developing a lightweight, explainable, and content-based movie recommendation system. The following list highlights the various achievements made during the completion of the project: Design and implementation of the content-based recommender system with TF-IDF vectorization and cosine similarity, an explainable recommendation system, and Testing in a web interface.

Evaluation results showed excellent performance, with a Precision@5 of 95.0%, a Precision@10 of 93.6%, and a 100% Hit Rate. These metrics are comparable to other content-based systems

found in the literature. The explainability feature enhances user understanding and trust, and the light architecture allows for deployment on standard laptops without cloud platforms or large user datasets.

In conclusion, CineMatch system is an effective solution that helps to alleviate information overload by providing personalized, transparent, and relevant movie suggestions. There are some restrictions in the project with reliance on the quality of the data set and lower recall values (which is inherent in the evaluation method) but there is a solution that is functional, educationally useful, and practical, and it has filled in some areas that were lacking in the literature.

5.3 Recommendations

In future work, the findings and experiences gathered in this project can be used to incorporate collaborative filtering or deep learning techniques to increase the diversity of the recommended movies, Dynamically refine the user-profiling mechanism to collect preferences automatically as users watch movies, thereby overcoming the problem of manual preference collection, Extend the system to a cross-platform mobile version to increase its accessibility by end users, further develop the explanation methods, for example by introducing visual explanations or counterfactual reasoning, to improve the user engagement further, Deploy the system in a real academic or small-scale streaming environment such as a university media center for further practical testing and feedback from end users, Explore dimensionality reduction techniques and caching mechanisms to reduce the response time as the size of the dataset will grow.

These recommendations offer clear pathways to improved and commercially viable recommendations.

REFERENCES

Bhavsar, A. M., Girase, H. S., Lohar, D. S., & Shaikh, A. Z. (2025). A content-based movie recommendation system using TF-IDF and cosine similarity. *International Journal of Computer Applications*.

de Campos, L. M., Fernández-Luna, J. M., & Huete, J. F. (2024). Explainable content-based recommendation systems using textual similarity. *Information Sciences*, 678, 120–135. https://doi.org/10.1016/j.ins.2024.120135

Jayalakshmi, S., Ganesh, N., Čep, R., & Senthil Murugan, J. (2022). A systematic review of movie recommender systems. *Applied Sciences*, 12(12), 6036. https://doi.org/10.3390/app12126036

Joseph, A., & Benjamin, P. (2022). Content-based movie recommendation using TF-IDF and cosine similarity. *Journal of Emerging Technologies and Innovative Research*. https://www.academia.edu/96796286/Movie_Recommendation_System_Using_Content_Based_Filtering_And_Cosine_Similarity

Kunde, O., Gaikwad, O., Kelgandre, P., & Damodhar, R. (2022). A simple content-based movie recommendation system for academic use. Available at: https://www.researchgate.net/publication/360575583 (related work on TF-IDF and similarity).

Meghana, K., Sudeeksha, E., Somanth, A., & Srinivasulu, Y. (2023). Content-based movie recommendation system using feature similarity. https://www.ijraset.com/research-paper/content-based-movie-recommendation-system

Mirhasani, M., & Ravanmehr, R. (2021). Alleviation of cold start in movie recommendation systems using sentiment analysis. https://www.researchgate.net/publication/348161804_Alleviation_of_Cold_Start_in_Movie_Recommendation_Systems_using_Sentiment_Analysis_of_Multi-Modal_Social_Networks

Permana, A. H. J., & Wibowo, A. T. (2023). Movie recommendation system based on synopsis using content-based filtering with TF-IDF and cosine similarity. https://www.semanticscholar.org/paper/Movie-Recommendation-System-Based-on-Synopsis-Using-Hiro-Permana/0b3eb89c9cba7e3d5a7bbb72450814f6d014dd53

Samih, A., Ghadi, A., & Fennan, A. (2021). ExMrec2Vec: Explainable movie recommender system based on Word2Vec. *International Journal of Advanced Computer Science and Applications*, 12(8). https://thesai.org/Publications/ViewPaper?Volume=12&Issue=8&Code=IJACSA&SerialNo=76

Sharma, S., Dubey, G. P., & Shakya, H. K. (2024). Hybrid movie recommendation framework. Related works available at: https://www.researchgate.net/publication/372490095 (efficient hybrid approaches).

Sidhu, M. K., & Sharma, N. (2025). Enhancing movie recommendations through metadata analysis.

Singh, K., Mishra, M., & Singh, S. (2024). Content-based movie recommendation system using cosine similarity.

Snigdha, P., Naveen, M., Rahul, S., Sujatha, C. N., & Pradeep, P. (2022). Textual similarity-based movie recommendation using TF-IDF.

Vultureanu-Albișі, A., & Badica, C. (2021). Recommender systems: An explainable AI perspective. https://www.researchgate.net/publication/354976986_Recommender_Systems_An_Explainable_AI_Perspective

Wandhekar, R., Nandane, K., & Garg, M. (2025). Improving content-based movie recommendations using TF-IDF and cosine similarity. https://www.irjet.net/archives/V12/i6/IRJET-V12I6157.pdf

Wang, Z. (2023). An improved content-based recommendation algorithm for movies. *Knowledge-Based Systems*.

Yuan, Y., Qin, Y., Yu, Z., & Zhang, C. (2023). Content-based movie recommendation using metadata features. *Expert Systems with Applications*, 212. https://doi.org/10.1016/j.eswa.2022.118732

Zhang, Z., Chen, L., Jiang, T., & Li, Y. (2022). The effect of feature-based explanations on user satisfaction in recommender systems. *User Modeling and User-Adapted Interaction*.

Yunanda, G., Nurjanah, D., & Meliana, S. (2022). Application of TF-IDF and cosine similarity in text-based recommendation systems. *Journal of Information Systems*.

APPENDIX

```
#app.py
from flask import Flask, render_template, request, jsonify, session, redirect, url_for
import recommender
from recommender import recommend_for_user, recommend_movies, browse_by_genre, movies,
trending_movies, fetch_poster
import sqlite3, hashlib, pandas as pd, os
from functools import wraps

app = Flask(__name__)
app.secret_key = "cinematch_secret_2024"
CSV_PATH = "movies_clean.csv"

def hash_password(pw):
    return hashlib.sha256(pw.encode()).hexdigest()

def get_db():
    conn = sqlite3.connect("database.db")
    conn.row_factory = sqlite3.Row
    return conn

def get_user_preferences(user_id):
    conn = get_db()
```

```
    row  = conn.execute(
        "SELECT genre, type, mood, recent FROM preferences WHERE user_id=? ORDER BY id
DESC LIMIT 1",
        (user_id,)
    ).fetchone()
    conn.close()
    if row:
            return {"genre": row["genre"], "type": row["type"], "mood": row["mood"], "recent":
row["recent"]}
    return None


def login_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        if "user_id" not in session:
            return redirect(url_for("login"))
        return f(*args, **kwargs)
    return decorated


def admin_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        if "user_id" not in session:
```

```
            return redirect(url_for("login"))
        conn = get_db()
                user = conn.execute("SELECT is_admin FROM users WHERE id=?",
(session["user_id"],)).fetchone()
        conn.close()
        if not user or not user["is_admin"]:
            return render_template("admin/denied.html"), 403
        return f(*args, **kwargs)
    return decorated

def reload_recommender():
    import importlib
    importlib.reload(recommender)
    global movies, trending_movies
    movies          = recommender.movies
    trending_movies = recommender.trending_movies

def get_all_genres():
    return sorted(set(g for gs in recommender.movies["genres"].dropna() for g in gs.split()))

@app.route("/register", methods=["GET","POST"])
def register():
    error = None
```

```
    if request.method == "POST":
        username  = request.form.get("username","").strip()
        full_name = request.form.get("full_name","").strip()
        email     = request.form.get("email","").strip()
        phone     = request.form.get("phone","").strip()
        password  = request.form.get("password","").strip()
        confirm   = request.form.get("confirm_password","").strip()
        if not username or not password:
            error = "Username and password are required."
        elif password != confirm:
            error = "Passwords do not match."
        elif len(password) < 6:
            error = "Password must be at least 6 characters."
        else:
            conn = get_db()
            try:
                conn.execute(
                    "INSERT INTO users (username, password, full_name, email, phone) VALUES
(?,?,?,?,?)",
                    (username, hash_password(password), full_name, email, phone)
                )
                conn.commit()
```

```
                    user = conn.execute("SELECT id FROM users WHERE username=?",
(username,)).fetchone()
                session["user_id"]  = user["id"]
                session["username"] = username
                session["is_admin"] = 0
                return redirect(url_for("preferences"))
            except sqlite3.IntegrityError:
                error = "Username already taken. Please choose another."
            finally:
                conn.close()
    return render_template("register.html", error=error)


@app.route("/login", methods=["GET","POST"])
def login():
    error = None
    if request.method == "POST":
        username = request.form.get("username","").strip()
        password = request.form.get("password","").strip()
        conn = get_db()
        user = conn.execute(
            "SELECT * FROM users WHERE username=? AND password=?",
            (username, hash_password(password))
        ).fetchone()
```

```
        conn.close()
        if user:
            session["user_id"]  = user["id"]
            session["username"] = user["username"]
            session["is_admin"] = user["is_admin"]
            return redirect(url_for("home"))
        else:
            error = "Invalid username or password."
    return render_template("login.html", error=error)

@app.route("/logout")
def logout():
    session.clear()
    return redirect(url_for("login"))

@app.route("/preferences", methods=["GET","POST"])
@login_required
def preferences():
    if request.method == "POST":
        conn = get_db()
        conn.execute(
            "INSERT INTO preferences (user_id, genre, type, mood, recent) VALUES (?,?,?,?,?)",
            (session["user_id"],
```

```
            request.form.get("genre",""),
            request.form.get("type",""),
            request.form.get("mood",""),
            request.form.get("recent","No"))
        )
        conn.commit()
        conn.close()
        if request.form.get("from_profile"):
            return redirect(url_for("profile"))
        return redirect(url_for("home"))
    existing = get_user_preferences(session["user_id"])
    return render_template("preferences.html", existing=existing,
                           from_profile=request.args.get("from_profile",""))

@app.route("/profile", methods=["GET","POST"])
@login_required
def profile():
    conn    = get_db()
            user          =     conn.execute("SELECT    *    FROM    users    WHERE    id=?",
(session["user_id"],)).fetchone()
    error   = None
    success = None
```

```
    if request.method == "POST":
        action = request.form.get("action")

        if action == "update_info":
            conn.execute(
                "UPDATE users SET full_name=?, email=?, phone=? WHERE id=?",
                (request.form.get("full_name","").strip(),
                 request.form.get("email","").strip(),
                 request.form.get("phone","").strip(),
                 session["user_id"])
            )
            conn.commit()
                        user = conn.execute("SELECT * FROM users WHERE id=?",
(session["user_id"],)).fetchone()
            success = "Profile updated successfully."

        elif action == "update_preferences":
            conn.execute(
                "INSERT INTO preferences (user_id, genre, type, mood, recent) VALUES (?,?,?,?,?)",
                (session["user_id"],
                 request.form.get("pref_genre",""),
                 request.form.get("pref_type",""),
                 request.form.get("pref_mood",""),
```

```
            request.form.get("pref_recent","No"))
        )
        conn.commit()
        success = "Preferences updated successfully."

    elif action == "change_password":
        current = request.form.get("current_password","").strip()
        new_pw  = request.form.get("new_password","").strip()
        confirm = request.form.get("confirm_password","").strip()
        if user["password"] != hash_password(current):
            error = "Current password is incorrect."
        elif new_pw != confirm:
            error = "New passwords do not match."
        elif len(new_pw) < 6:
            error = "Password must be at least 6 characters."
        else:
            conn.execute("UPDATE users SET password=? WHERE id=?",
                         (hash_password(new_pw), session["user_id"]))
            conn.commit()
            success = "Password changed successfully."

history = conn.execute(
    "SELECT movie_title, COALESCE(watched_at,'') as watched_at FROM history "
```

```
        "WHERE user_id=? ORDER BY rowid DESC LIMIT 10", (session["user_id"],)
    ).fetchall()
    ratings = conn.execute(
        "SELECT movie_title, rating FROM ratings WHERE user_id=? ORDER BY rating DESC",
        (session["user_id"],)
    ).fetchall()
    conn.close()

    current_prefs = get_user_preferences(session["user_id"])

    if success and "Preferences" in success:
        open_tab = "preferences"
    elif (success and "Password" in success) or (error and "password" in str(error).lower()):
        open_tab = "password"
    else:
        open_tab = "info"

    return render_template("profile.html", user=user, history=history, ratings=ratings,
                           error=error, success=success, username=session.get("username"),
                           current_prefs=current_prefs, open_tab=open_tab)

@app.route("/search")
def search():
```

```
    query    = request.args.get("q","").strip()
    is_guest = "user_id" not in session
    if not query:
        return jsonify([])
     results  =  recommender.movies[recommender.movies["title"].str.contains(query,  case=False,
na=False)]
    if is_guest:
        results = results[results["is_recent"] == 0]
    return jsonify(results["title"].head(10).tolist())


@app.route("/", methods=["GET","POST"])
def home():
    is_guest      = "user_id" not in session
    user_prefs    = None if is_guest else get_user_preferences(session["user_id"])
    params        = request.form if request.method == "POST" else request.args
    genre_filter  = params.get("genre","All")
    type_filter   = params.get("type","All")
    selected_movie = (params.get("movie","") or "").strip() or None

    selected_data   = None
    related_movies  = []
    recommendations = []
    not_found       = False
```

```
guest_blocked   = False


if selected_movie:
    match = recommender.movies[recommender.movies["title"] == selected_movie]
    if match.empty:
        not_found       = True
        selected_movie  = None
        recommendations = browse_by_genre(None, is_guest=is_guest, n=12)
    elif is_guest and int(match.iloc[0].get("is_recent",0)) == 1:
        guest_blocked   = True
        selected_movie  = None
        recommendations = browse_by_genre(None, is_guest=True, n=12)
    else:
        row = match.iloc[0]
        selected_data = {
            "title":    row["title"],
            "overview": row["overview"],
            "genres":   row["genres"],
            "year":     int(row.get("year",0)),
            "poster":   fetch_poster(row["title"]),
            "type":     row.get("type","Hollywood"),
        }
        related_movies = recommend_movies(selected_movie, user_prefs)
```

```
        if not is_guest:
            try:
                conn = get_db()
                conn.execute("INSERT INTO history (user_id, movie_title) VALUES (?,?)",
                             (session["user_id"], selected_movie))
                conn.commit()
                conn.close()
            except Exception as e:
                print("History error:", e)

elif genre_filter != "All":
    recommendations = browse_by_genre(
        genre_filter if genre_filter != "All" else None,
        is_guest=is_guest,
        n=12,
    )
elif not is_guest:
    recommendations = recommend_for_user(session["user_id"])
else:
    recommendations = browse_by_genre(None, is_guest=True, n=12)

trending = []
for _, row in recommender.trending_movies.iterrows():
```

```
        if is_guest and int(row.get("is_recent",0)) == 1:
            continue
        trending.append({"title": row["title"], "overview": row["overview"],
                         "genres": row["genres"], "year": int(row.get("year",0)),
                         "poster": fetch_poster(row["title"])})

    all_genres = sorted(set(g for gs in recommender.movies["genres"].dropna() for g in gs.split()))

    return render_template("index.html",
        selected_data=selected_data, selected_movie=selected_movie,
        related_movies=related_movies, recommendations=recommendations,
        trending=trending, genre_filter=genre_filter, type_filter=type_filter,
        all_genres=all_genres,
        is_guest=is_guest, not_found=not_found, guest_blocked=guest_blocked,
        username=session.get("username"), is_admin=session.get("is_admin",0),
    )
@app.route("/rate", methods=["POST"])
@login_required
def rate():
    movie  = request.form.get("movie","").strip()
    rating = request.form.get("rating")
    if movie and rating:
        conn = get_db()
```

```
            conn.execute("DELETE FROM ratings WHERE user_id=? AND movie_title=?",
(session["user_id"], movie))
        conn.execute("INSERT INTO ratings (user_id, movie_title, rating) VALUES (?,?,?)",
                (session["user_id"], movie, int(rating)))
        conn.commit()
        conn.close()
    return redirect(url_for("home", movie=movie))


@app.route("/admin")
@admin_required
def admin_dashboard():
    df   = recommender.movies
    conn = get_db()
    stats = {
        "total_movies":  len(df),
                "total_users":       conn.execute("SELECT COUNT(*) FROM users WHERE
is_admin=0").fetchone()[0],
        "total_ratings": conn.execute("SELECT COUNT(*) FROM ratings").fetchone()[0],
        "genres":        get_all_genres(),
    }
    conn.close()
    return render_template("admin/dashboard.html", stats=stats,
                           movies=df.to_dict(orient="records"), username=session.get("username"))
```

```
@app.route("/admin/add", methods=["GET","POST"])
@admin_required
def admin_add():
    error = None
    if request.method == "POST":
        title = request.form.get("title","").strip()
        ov    = request.form.get("overview","").strip()
        gen   = request.form.get("genres","").strip()
        kw    = request.form.get("keywords","").strip()
        pop   = request.form.get("popularity","100").strip()
        yr    = request.form.get("year","2000").strip()
        mtype = request.form.get("type","Hollywood").strip()
        if not title or not ov or not gen:
            error = "Title, overview, and genres are required."
        else:
            df = pd.read_csv(CSV_PATH)
            if title in df["title"].values:
                error = f"'{title}' already exists."
            else:
                new_row = pd.DataFrame([{"title":title,"overview":ov,"genres":gen,"keywords":kw,
                    "popularity":int(pop) if pop.isdigit() else 100,
                    "year":int(yr),"is_recent":1 if int(yr)>=2015 else 0,
```

```
                "type":mtype,"poster_url":""}])
            pd.concat([df, new_row], ignore_index=True).to_csv(CSV_PATH, index=False)
            reload_recommender()
            return redirect(url_for("admin_dashboard"))
    return render_template("admin/movie_form.html", action="Add", movie=None,
                           genres_list=get_all_genres(),
                           error=error, username=session.get("username"))


@app.route("/admin/edit/<path:title>", methods=["GET","POST"])
@admin_required
def admin_edit(title):
    df = pd.read_csv(CSV_PATH)
    match = df[df["title"] == title]
    if match.empty:
        return redirect(url_for("admin_dashboard"))
    error = None
    if request.method == "POST":
        nt = request.form.get("title","").strip()
        ov = request.form.get("overview","").strip()
        gen = request.form.get("genres","").strip()
        kw  = request.form.get("keywords","").strip()
        pop = request.form.get("popularity","100").strip()
        yr  = request.form.get("year","2000").strip()
```

```
        mtype = request.form.get("type","Hollywood").strip()
        if not nt or not ov or not gen:
            error = "Title, overview, and genres are required."
        else:
            idx = df[df["title"]==title].index[0]
            df.loc[idx,["title","overview","genres","keywords","popularity","year","type","is_recent"
]] = \
                [nt,ov,gen,kw,int(pop) if pop.isdigit() else 100,int(yr),mtype,1 if int(yr)>=2015 else 0]
            df.to_csv(CSV_PATH, index=False)
            reload_recommender()
            return redirect(url_for("admin_dashboard"))
    return render_template("admin/movie_form.html", action="Edit",
                           movie=match.iloc[0].to_dict(),
                           genres_list=get_all_genres(),
                           error=error, username=session.get("username"))

@app.route("/admin/delete/<path:title>", methods=["POST"])
@admin_required
def admin_delete(title):
    df = pd.read_csv(CSV_PATH)
    df[df["title"] != title].to_csv(CSV_PATH, index=False)
    reload_recommender()
    return redirect(url_for("admin_dashboard"))
```

```
@app.route("/admin/genres", methods=["GET","POST"])
@admin_required
def admin_genres():
    import re
    message = None
    if request.method == "POST":
        action    = request.form.get("action")
        old_genre = request.form.get("old_genre","").strip()
        new_genre = request.form.get("new_genre","").strip()
        df        = pd.read_csv(CSV_PATH)
        if action == "rename" and old_genre and new_genre:
            df["genres"] = df["genres"].apply(
                lambda g: re.sub(r'\b'+re.escape(old_genre)+r'\b', new_genre, str(g)))
            df.to_csv(CSV_PATH, index=False); reload_recommender()
            message = f"Renamed '{old_genre}' → '{new_genre}'."
        elif action == "delete" and old_genre:
            df["genres"] = df["genres"].apply(
                lambda g: re.sub(r'\b'+re.escape(old_genre)+r'\b',",str(g)).strip())
            df["genres"] = df["genres"].str.replace(r'\s+',' ',regex=True).str.strip()
            df.to_csv(CSV_PATH, index=False); reload_recommender()
            message = f"Deleted genre '{old_genre}'."
    genres_list  = get_all_genres()
```

```
    df          = pd.read_csv(CSV_PATH)
      genre_counts = {g: int(df["genres"].str.contains(g,case=False,na=False).sum()) for g in
genres_list}
    return render_template("admin/genres.html", genres=genres_list, genre_counts=genre_counts,
                           message=message, username=session.get("username"))


@app.route("/admin/metadata", methods=["GET","POST"])
@admin_required
def admin_metadata():
    message = None
    if request.method == "POST":
        title    = request.form.get("title","").strip()
        keywords = request.form.get("keywords","").strip()
        overview = request.form.get("overview","").strip()
        if title:
            df = pd.read_csv(CSV_PATH)
            if title in df["title"].values:
                if keywords: df.loc[df["title"]==title,"keywords"] = keywords
                if overview: df.loc[df["title"]==title,"overview"] = overview
                df.to_csv(CSV_PATH, index=False); reload_recommender()
                message = f"Metadata updated for '{title}'."
            else:
                message = f"Movie '{title}' not found."
```

```
    df = recommender.movies
    return render_template("admin/metadata.html",
                           movies=df[["title","keywords","overview","popularity"]].to_dict(orient="record
s"),
                           message=message, username=session.get("username"))


if __name__ == "__main__":
    app.run(debug=True)
```