**TABLE OF CONTENTS**

[SUPERVISOR’S RECOMMENDATION ii](#_Toc199830584)

[ABSTRACT iii](#_Toc199830585)

[LETTER OF APPROVAL iv](#_Toc199830586)

[ACKNOWLEDGEMENT v](#_Toc199830587)

[CHAPTER 1: INTRODUCTION 1](#_Toc199830588)

[1.1 Introduction 1](#_Toc199830589)

[1.2. Problem Statement 1](#_Toc199830590)

[1.3. Objectives 2](#_Toc199830591)

[1.4 Scope and Limitation 2](#_Toc199830592)

[1.5 Development Methodology 3](#_Toc199830593)

[1.6 Report Organization 4](#_Toc199830594)

[CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW 5](#_Toc199830595)

[2.1 Background Study 5](#_Toc199830596)

[2.1.1 Fundamental Theories and General Concepts 5](#_Toc199830597)

[2.2 Literature Review 6](#_Toc199830598)

[CHAPTER 3: SYSTEM ANALYSIS AND DESIGN 8](#_Toc199830599)

[3.1 System Analysis 8](#_Toc199830600)

[3.1.1 Requirement Analysis: 8](#_Toc199830601)

[3.1.2 Feasibility Analysis 11](#_Toc199830602)

[3.1.3 Data modelling: ER Diagram 13](#_Toc199830603)

[3.1.4 Process Modelling: DFD 14](#_Toc199830604)

[3.2 System Design 16](#_Toc199830605)

[3.2.1 Architectural design 16](#_Toc199830606)

[3.2.2 Database Schema design 18](#_Toc199830607)

[3.2.3 Interface design (UI/UX) 19](#_Toc199830608)

[3.3 Algorithm details 20](#_Toc199830609)

[CHAPTER 4: IMPLEMENTATION AND TESTING 23](#_Toc199830610)

[4.1 Implementation 23](#_Toc199830611)

[4.1.1 Tools Used: 23](#_Toc199830612)

[4.1.2 Implementation details of modules 24](#_Toc199830613)

[4.2. Testing 26](#_Toc199830614)

[4.2.1. Test Cases for Unit Testing: 26](#_Toc199830615)

[4.2.2 Test Cases for System Testing 29](#_Toc199830616)

[CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATIONS 31](#_Toc199830617)

[5.1. Lesson Learnt/Outcome: 31](#_Toc199830618)

[5.2. Conclusion: 31](#_Toc199830619)

[5.3. Future Recommendations 32](#_Toc199830620)

[References 33](#_Toc199830621)

**LIST OF ABBREVIATIONS**

CSS Cascading Style sheets

DFD Data Flow Diagram

ERD Entity Relationship diagram

HTML Hypertext Markup Language

JS JavaScript

UI User Interface

UX User Experience

**LIST OF FIGURES**

Figure 1.1 Waterfall Model 3

Figure 3.1 Use case diagram 7

Figure 3.2 ER-Diagram 9

Figure 3.3 Level 0 DFD 10

Figure 3.4. Level 1 DFD 11

Figure 3.5 Architectural design 12

Figure 3.6 Database Schema design 13

Figure 3.8 Interface design (UI/UX) 13

Figure 3.9 Physical DFD 16

Figure 5.1 Appendices 20

# CHAPTER 1: INTRODUCTION

## Introduction

In today’s digital world, people have access to thousands of movies online, which can make it hard to decide what to watch. Movie recommendation systems solve this problem by suggesting movies based on the user's interests and preferences. These systems have become a common feature in platforms like Netflix and Amazon Prime, helping users save time and discover content they enjoy.

This project, called **Movie Recommendation System**, is designed to give users personalized movie suggestions. It uses two main techniques — content-based filtering and collaborative filtering. Content-based filtering recommends movies that are similar to the ones the user has liked, while collaborative filtering recommends movies liked by other users with similar tastes. When users are not logged in, the system simply shows random movies. After logging in, the recommendations become personalized based on their actions.

The system also allows users to like movies, write reviews, and explore other movies by the same director. We used data from IMDb and OMDb APIs to build a database of around 4000 movies. The backend was developed using Django, and the frontend uses Django Templates. For the recommendation logic, machine learning libraries like Scikit-learn, pandas, and NumPy were used. This combination of technologies provides a smooth and personalized movie browsing experience for users.

## 1.2. Problem Statement

With the huge number of movies available on the internet, it has become difficult for users to choose what to watch. People often spend a lot of time scrolling through lists of movies without finding something they really like. This leads to a poor user experience, especially for those who are looking for something specific or based on their mood and interests.

Most existing movie platforms show trending or popular movies, but they do not always match the personal preferences of every user. Without proper suggestions, users may miss out on good movies that suit their taste. A system that understands the user’s behavior, preferences, and past interactions can greatly improve the movie-watching experience.

This project aims to solve that issue by building a smart Movie Recommendation System. It uses both content-based and collaborative filtering techniques to give users personalized suggestions. It also allows users to like and review movies, which helps improve future recommendations. By using data from IMDb and OMDb, and building the system in Django with machine learning support, the project creates a better way to discover movies online.

## 1.3. Objectives

The main objectives of Movie Recommendation System are as below:

* To implement a movie recommendation system using content-based and collaborative filtering algorithms for providing personalized suggestions.
* To allow users to interact with movies by liking and reviewing them, helping improve the accuracy of recommendations.

## 1.4 Scope and Limitation

**I. Scope:**

The Movie Recommendation System focuses on providing users with personalized movie suggestions based on their activity and preferences. It uses both content-based filtering (by analyzing movie genres, descriptions, etc.) and collaborative filtering (based on user interactions like likes and reviews). The system displays random movies to unauthenticated users and switches to intelligent recommendations once the user logs in. Users can also explore other movies by the same director, giving them more options to find content they may enjoy.

The system is built using Django and Django templates for the backend and frontend, with SQLite used as the database. Machine learning libraries such as Scikit-learn, pandas, and NumPy are used for recommendation logic. The project also includes features such as user registration, login, liking, reviewing, and viewing movie details. With a database of around 4000 movies sourced from IMDb and OMDb, the platform offers a decent variety of content for personalized exploration.

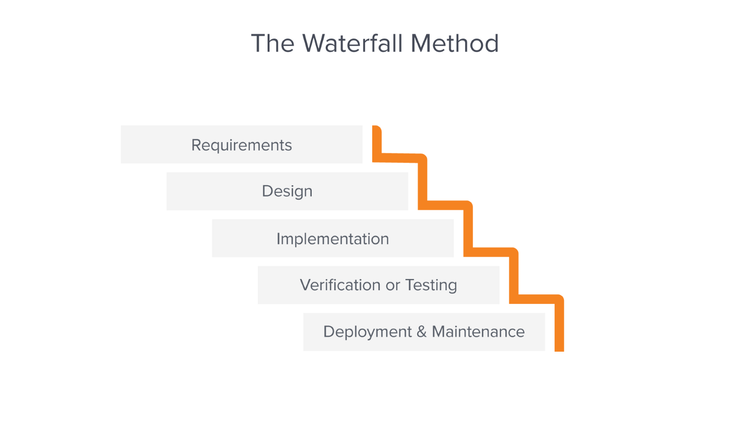
**ii. Limitations:**

Although the system works well for the current dataset and user base, it may face performance issues as the number of users or movies increases. SQLite is suitable for small projects but may not handle heavy loads in a production environment. As a result, the system might need to be upgraded to a more powerful database like PostgreSQL or MySQL for larger-scale use.

Another limitation is the cold start problem — new users who have not liked or reviewed any movies yet may not receive very accurate recommendations. Also, since the recommendations depend heavily on existing user data, the system's performance can be affected if user activity is low. Despite these limitations, the current system still offers a strong base for future improvements and scaling.

## 1.5 Development Methodology

The process flow for Movie Recommendation System includes analysis of the requirements, design, implementation, testing, and maintenance. During the requirement analysis process, every functional and non-functional requirement is examined and the system is then developed to meet the requirements. The system is integrated and tested after the design phase is followed by the coding and development phase. The system is installed if the testing is successful; if not, some maintenance is carried out prior to the system being used.



**Figure 0:1.1 Waterfall Model**

## 1.6 Report Organization

The purpose of this report is to provide a comprehensive overview of the Movie Recommendation System project. It begins with **Chapter 1:** Introduction, which presents the background of the project, the problems it aims to solve, its objectives, scope, and limitations. This chapter also outlines the development methodology used and explains the structure of the report.

**Chapter 2:** Background Study and Literature Review explores key theories, concepts, and terminology related to the project. It also examines similar systems and reviews previous research in the field, providing a foundation for understanding the project’s significance.

**Chapter 3:** System Analysis and Design focuses on the analysis and design of the system. It details the requirements, data structure, process flow, and overall system architecture, including database design and user interface development.

Finally, **Chapter 4:** Implementation and Testing describes how the system was developed and tested. It covers the tools, platforms, and technologies used, along with the deployment setup and testing methodologies to ensure system reliability and efficiency.

# CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

## 2.1 Background Study

With the rapid growth of digital media, the number of movies available online has increased massively. Users now have access to thousands of movies, but finding the right one has become a challenge. To solve this problem, recommendation systems have been introduced across various platforms. These systems analyze user preferences, past behaviors, and item features to suggest the most relevant content, saving time and improving the user experience.

There are different techniques used in building recommendation systems. Content-based filtering focuses on suggesting movies similar to those a user has already liked, by analyzing characteristics like genre, director, plot, or actors. Collaborative filtering, on the other hand, suggests movies based on the interests of other users with similar behavior. Each technique has its own advantages and challenges. Content-based systems may lack diversity, while collaborative systems face problems when new users or movies are introduced without enough data.

In this project, we designed a hybrid approach that combines both content-based and collaborative filtering methods. By using data collected from IMDb and OMDb APIs, and implementing machine learning algorithms, we aimed to create a system that improves with user activity. Features like liking, reviewing, and exploring other movies by the same director were added to make the experience richer. The use of Django, SQLite, and libraries like Scikit-learn, pandas, and NumPy allowed us to build a simple yet effective Movie Recommendation System.

### 2.1.1 Fundamental Theories and General Concepts

Recommendation Systems: A recommendation system is an intelligent filtering technique used to suggest relevant items to users based on their interests, behaviors, or interactions. It helps improve user experience by reducing search effort and providing personalized suggestions. An instance of this is content-based filtering, where the system matches search keywords and course attributes to recommend courses that align with the user’s interests. This ensures that recommendations are relevant and tailored to individual preferences.

Content-based filtering recommends movies by matching features like genre, cast, and plot, while collaborative filtering suggests movies based on the preferences of similar users. By combining both methods, the system reduces the weaknesses of each individual technique and gives users more relevant recommendations. This makes the movie discovery process faster, smarter, and more personalized for each user.

Database Management: Database management involves storing, organizing, and retrieving structured data efficiently to support system functionality. A well-structured database ensures fast and accurate data retrieval for recommendation processes. An instance of this is SQL-based database querying, where the system fetches relevant courses based on user search terms and historical interactions to generate meaningful recommendations.

## 2.2 Literature Review

Recommendation systems have become an important part of modern applications, especially in areas like e-commerce, music, and movie platforms. Researchers have developed many techniques to suggest items based on user interests. Early systems mainly used content-based filtering, where suggestions are made by comparing item features with what the user liked in the past [1]. While this approach works well for individual preferences, it often struggles to introduce diversity because it keeps recommending similar types of movies.

To overcome the limits of content-based methods, collaborative filtering was introduced. Collaborative filtering focuses on user behavior rather than item features [2]. It suggests movies that similar users have liked. This technique became popular because it can recommend movies that a user might not have discovered otherwise. However, collaborative filtering also has problems like the cold start issue, where it becomes difficult to recommend anything to new users who have no previous activity [3].

Researchers found that combining content-based and collaborative filtering often gives better results. Hybrid recommendation systems take the advantages of both methods and reduce their weaknesses [4]. For example, when collaborative filtering struggles due to less user data, content-based filtering can still provide suggestions based on movie features. Many real-world platforms like Netflix and Amazon Prime now use hybrid models to recommend a wider variety of content to their users [5].

Another important area explored in literature is the use of machine learning and similarity measures. Techniques like cosine similarity, K-nearest neighbors (KNN), and clustering are often used to find similar movies or users [6]. Machine learning libraries such as Scikit-learn have made it easier to implement these techniques by providing ready-to-use functions for similarity computation and recommendation models. These tools help in building more scalable and efficient recommendation systems.

Many studies also highlight the importance of user interaction features like likes, ratings, and reviews. These interactions provide valuable data that improves recommendation quality over time [7]. Platforms that encourage users to interact more tend to build stronger and smarter recommendation engines. In this project, we applied these learnings by collecting user likes and reviews to continuously improve movie recommendations and make the system more personalized and engaging.

# CHAPTER 3: SYSTEM ANALYSIS AND DESIGN

## 3.1 System Analysis

3.1.1 Requirement Analysis:

The requirements are to be collected prior to beginning projects’ development life cycle. Both functional and non-functional requirements of the system have been researched to build and create it.

#### Functional Requirements:

**User Management**

* Users must be able to register, log in, and log out securely.
* Only logged-in users should access personalized movie recommendations.
* Users should be able to like and review movies after authentication.

**Movie Recommendation and Browsing**

* Unauthenticated users should see random movies displayed on the homepage.
* Logged-in users should receive personalized recommendations using content-based and collaborative filtering.
* Users should be able to explore other movies made by the same director.

**Movie Interaction and Search Features**

* Users should be able to search for movies by title, genre, or other attributes.
* Each movie should have a detailed page showing information, reviews, and a like button.
* Recommendations should dynamically update based on the user’s likes and reviews.

**Use Case Diagram**

The **Movie Recommendation System** is designed to enhance movie discovery by providing personalized suggestions based on user interactions. The system allows both authenticated and non-authenticated users to search for movies, with logged-in users receiving personalized recommendations based on their likes, reviews, and viewing history. Additionally, a Trending Searches section highlights frequently searched movie titles and genres, helping users explore popular and trending movies more easily.

A diagram of a movie

AI-generated content may be incorrect.

**Figure 3. 1 Use case diagram**

#### ii. Non-Functional Requirement

In addition to functional requirements, the Movie Recommendation System must meet several non-functional requirements to ensure reliability, performance, security, and usability. These define how the system should operate rather than what it should do.

**Performance Requirements:**

* The system should display random movies for unauthenticated users within 2 seconds of page load.
* Personalized movie recommendations should be generated and displayed within 3–5 seconds after user login.
* The system should support at least 50 concurrent users browsing or searching movies without significant delay.

**Scalability:**

* The system should be able to handle a growing number of users and movie records without performance loss.
* The database design should allow easy migration from SQLite to a more scalable database like PostgreSQL in the future.
* The recommendation algorithm should be optimized to recompute similarities without needing to process the entire dataset repeatedly.

**Security:**

* User passwords must be securely stored using hashing techniques like bcrypt.
* Only authenticated users should be allowed to like or review movies to prevent misuse.
* Sensitive operations (like login and registration) should be protected using HTTPS to prevent data interception.

**Usability:**

* The system should have a simple, easy-to-navigate interface suitable for users with basic computer knowledge.
* All major actions (like search, like, review) should be completed within three clicks or fewer.
* The movie recommendation results should be presented in a clean and understandable format with minimal loading screens.

**Reliability and Availability:**

* The system should maintain consistent performance without crashes or major downtime under normal usage.
* User actions like likes, reviews, and search history must be saved reliably in the database without loss.
* In case of temporary server failure, the system should automatically recover without losing user data.
* The system should be available for users at least 95% of the time (measured monthly).
* In case of system errors, users should receive clear error messages and options to retry instead of system crashes.

### 3.1.2 Feasibility Analysis

#### i. Technical Feasibility Analysis

The Movie Recommendation System is technically feasible using the selected tools and technologies. The backend is built with Django, which provides strong support for user authentication, session handling, and fast development. The frontend uses Django Templates, making integration simple and efficient. For the database, SQLite is used, which is suitable for handling around 4000 movies and moderate user traffic. Machine learning libraries like Scikit-learn, pandas, and NumPy are used to implement the recommendation algorithms, making the system efficient and easy to maintain. Overall, the technologies chosen are capable of supporting the project requirements both now and for future upgrades.

#### ii. Operational Feasibility Analysis

The Movie Recommendation System is operationally feasible as it meets the needs of users by providing an easy and efficient way to discover movies. The system is designed with a simple and user-friendly interface, allowing users to search, like, and review movies without any technical difficulty. Personalized recommendations make the platform more engaging for logged-in users, while random movie suggestions keep unauthenticated users interested. Since the system is built with familiar technologies like Django and SQLite, it can be easily managed and maintained by developers. Overall, the system can operate smoothly with minimal training or technical support.

#### iii. Economic Feasibility Analysis

The Movie Recommendation System is economically feasible as it requires minimal cost for development and maintenance. Open-source technologies like Django, SQLite, Scikit-learn, pandas, and NumPy are used, which do not require any licensing fees. Hosting and deployment can also be managed using affordable cloud services or local servers during the initial phase. Since the system is designed for easy scalability, future upgrades like migrating to a larger database or adding more features can be done without major additional costs. Overall, the project provides a good balance between functionality and cost, making it financially practical to implement and operate.

**iv. Schedule Feasibility Analysis:**

Schedule feasibility evaluates whether the project can be completed within the desired or allocated time frame. For the Movie Recommendation System, the timeline was carefully analyzed based on task complexity, available resources, development tools, and team capacity. Key modules such as user authentication, movie database setup, recommendation logic, and UI development were broken into manageable milestones. Given the scope and modular design of the system, the project is schedule-feasible and can be completed within the academic deadline. Agile practices like iterative testing and modular integration ensure timely progress, reducing the risk of delays and allowing buffer time for final testing and documentation.

### 3.1.3 Data modelling: ER Diagram

The above E-R diagram represents the "Review" and "Favorite" entities in the Movie Recommendation System. The Review entity stores user ratings and comments for movies, including attributes such as rating (1–5 stars), comment, and the creation date. It connects users and movies through foreign keys, enabling multiple users to review multiple movies. The Favorite entity captures the movies that users have marked as favorites, preventing duplicates through a unique user-movie combination. Both entities have primary keys for identification and are linked to users and movies through foreign key relationships, ensuring data consistency and user-specific interactions.

A diagram of a computer system

AI-generated content may be incorrect.

**Figure 3. 2 ER-Diagram**

### 3.1.4 Process Modelling: DFD

Data Flow Diagram consists of two levels of DFD which are context diagram and level 1 DFD. This is used to make the system.

**Level 0: Context DFD**

In The Level 0 Context Diagram for the Movie Recommendation System shows the interaction between the user and the system. The user can send requests such as registration, login, movie likes, reviews, and search queries to the system. In response, the system handles these actions and returns appropriate results like registration confirmation, login responses, and personalized movie recommendations. This diagram highlights the main data flow between the user and the system, ensuring a smooth movie discovery experience without involving any admin role.

A diagram of a movie recommend system

AI-generated content may be incorrect.

**Figure 3.3 Level 0 DFD**

**Level 1 DFD:**

The Level 1 Data Flow Diagram for the Movie Recommendation System shows how the system is divided into smaller sub-processes. The user interacts with different processes such as registration, login, liking or reviewing movies, searching movies, and receiving personalized recommendations. The system manages user data through the User Database and stores movie information in the Movies Database and Favorites Database. Each process handles specific user requests and updates the relevant databases accordingly, ensuring a smooth and personalized movie discovery experience.

A diagram of a diagram

AI-generated content may be incorrect.

**Figure : 3.4 Level 1 DFD**

## 3.2 System Design

System design involves defining the architecture, components, and interfaces of a system to meet specified requirements. It includes the planning of both hardware and software elements to ensure they work together efficiently and effectively.

### 3.2.1 Architectural design

The Movie Recommendation System follows a modular and scalable architecture based on the Model-View-Controller (MVC) pattern, commonly used in Django applications. It consists of four primary layers:

* **Presentation Layer (Frontend - UI/UX):** The presentation layer is responsible for delivering an interactive and user-friendly interface for the Movie Recommendation System. It includes dynamic elements such as movie search, trending movie displays, personalized movie recommendations, and liked movies list. Built using HTML, CSS, and JavaScript, the frontend is rendered with Django’s templating engine (Jinja2), ensuring that dynamic movie data, recommendations, and reviews are shown seamlessly. The design is intuitive and responsive, allowing users to easily explore movies, like and review them, and discover new content based on their interests.
* **Application Layer (Django Views & Business Logic):** The application handles the core backend logic, managing user authentication, movie search queries, likes, reviews, and recommendation generation. Implemented using Django’s Model-View-Template (MVT) architecture, views handle requests and render the appropriate templates dynamically. Functions such as home\_view(), search\_view(), movie\_detail\_view(), and recommend\_movies() ensure that user actions like liking a movie or submitting a review dynamically update their personalized movie feed. Business logic for content-based and collaborative filtering is processed here, adapting recommendations based on user behavior.
* **Data Layer (Database Models & Storage):** The data layer manages structured storage for users, movies, likes, reviews, and search queries. Built using Django ORM with SQLite (or PostgreSQL for scalability), this layer ensures efficient storage and retrieval of relational data. Key models include User, Movie, Review, Favorite, and SearchQuery, allowing the system to track user interactions such as liked movies, movie ratings, search keywords, and viewed history. This stored data is critical for powering the recommendation engine and improving personalization over time.
* **Recommendation Engine (AI-Based Movie Recommendations**): Recommendation engine forms the core intelligence of the system, providing personalized movie suggestions based on user activity. It uses a hybrid approach, combining Content-Based Filtering (based on movie features like genre, description, and director) and Collaborative Filtering (based on similar users' likes and reviews). Techniques such as cosine similarity and user interaction analysis are employed to recommend movies that match the user's preferences. Functions like get\_similar\_movies() and get\_user\_recommendations() use stored likes, reviews, and search history to suggest relevant movies, creating an intelligent, evolving user experience.A diagram of a computer server and machine learning

  Description automatically generated

**Figure 3. 5Architectural design**

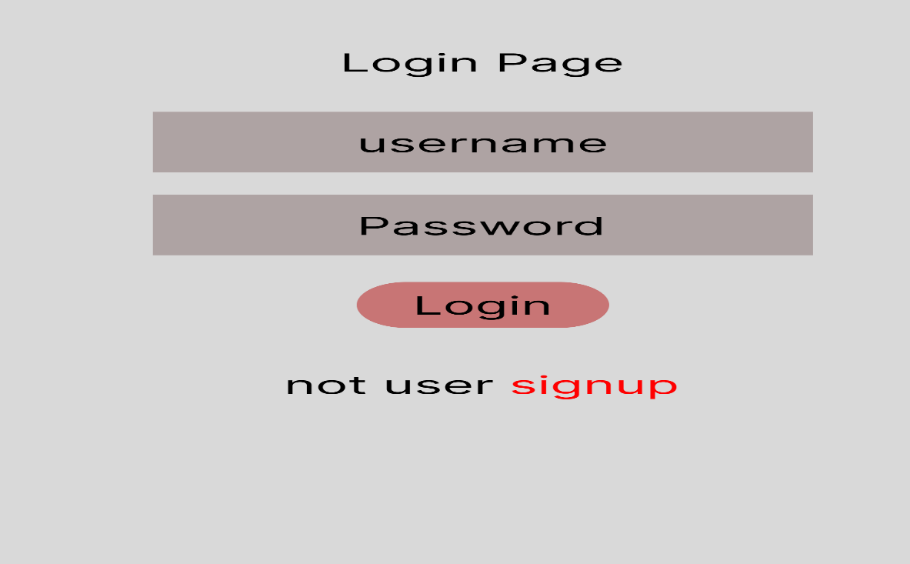
### 3.2.2 Database Schema design

A diagram of a user

AI-generated content may be incorrect.

**Figure 3. 6 Database Schema design**

### 3.2.3 Interface design (UI/UX)



A screenshot of a phone

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A close-up of a book cover

AI-generated content may be incorrect.

**Figure 3. 7 Interface design (UI/UX)**

## 3.3 Algorithm details

**Content Based Filtering**

Content-based filtering is a personalized recommendation technique that suggests movies to users based on the attributes of movies they have previously liked or interacted with. It works by analyzing metadata such as genre, director, actors, plot keywords, and language to create a unique user profile representing the user's tastes. For example, if a user frequently watches sci-fi thrillers directed by Christopher Nolan, the system might recommend similar movies that share those characteristics—even if no other user has watched them.

A key component of this approach is text analysis and feature extraction, where the system converts movie metadata into numerical vectors using techniques like TF-IDF (Term Frequency–Inverse Document Frequency). Then, it calculates the cosine similarity between vectors to find movies that are most similar to the user's past favorites. Unlike collaborative filtering, content-based filtering doesn’t rely on other users’ behavior, making it effective even for new or niche users. However, it may sometimes suffer from limited exploration, repeatedly recommending similar types of movies, thus reducing diversity in suggestions.

**Pseudo Code**

Function get\_content\_based\_recommendations(movie\_id, num\_recommendations):

// Step 1: Fetch all movies from the database

movies = Get all Movie objects

For each movie:

Combine genre, director, actors, and plot into a single string

// Step 2: Vectorize combined movie data

vectorizer = Create a TF-IDF Vectorizer (stopwords = English)

tfidf\_matrix = Apply vectorizer on combined movie strings

// Step 3: Compute similarity between movies

similarity\_matrix = Compute cosine similarity of tfidf\_matrix

// Step 4: Find similar movies

target\_index = Find index of movie with movie\_id

similarity\_scores = List of (index, similarity) for all movies

Sort similarity\_scores in descending order (exclude target movie)

// Step 5: Return top N similar movies

recommended\_movie\_ids = Get top N movie IDs from similarity\_scores

Return Movie objects for those IDs

**Collaborative Filtering:**

Collaborative filtering is a recommendation technique that suggests movies based on the interests and behaviors of other users, rather than the content of the movies themselves. It assumes that users who have shown similar preferences in the past are likely to enjoy similar movies in the future. For instance, if two users both liked “Inception” and “Interstellar,” and one of them also liked “The Martian,” the system may recommend “The Martian” to the other user—even if that user has never searched for or interacted with it directly.

This method can be further divided into user-based and item-based collaborative filtering. In user-based filtering, the algorithm finds users with similar watching patterns and suggests movies liked by them. In item-based filtering, it recommends movies that are commonly watched together by many users. These similarities are calculated using metrics like cosine similarity or Pearson correlation on a user-item interaction matrix. Collaborative filtering is powerful for discovering diverse and unexpected recommendations but can struggle with the cold start problem, where new users or new movies lack sufficient interaction history to generate reliable recommendations.

**Pseudo Code:**

Function get\_collaborative\_recommendations(user\_id, num\_recommendations):

// Step 1: Build user-movie matrix from favorites

favorites = Get all Favorite(user\_id, movie\_id) records

user\_movie\_matrix = Create pivot table: rows = movies, columns = users, values = liked (1/0)

// Step 2: Compute user similarity

user\_similarities = Compute cosine similarity between users (transpose matrix)

similar\_users = Sort similarity scores of user\_id in descending order

// Step 3: Get movies liked by similar users

similar\_user\_movies = Sum columns of similar users for each movie

// Step 4: Filter out movies already liked by current user

user\_liked\_movies = Get movie\_ids from Favorite where user = user\_id

recommended\_movies = Movies from similar\_user\_movies not in user\_liked\_movies

// Step 5: Return top N recommended movies

Return top num\_recommendations from recommended\_movies

# CHAPTER 4: IMPLEMENTATION AND TESTING

## 4.1 Implementation

### 4.1.1 Tools Used:

* **CASE Tools:** *Figma*, and *Draw.io* were used for designing user interfaces and system architecture diagrams, such as the ER diagrams, schema designs, and activity flows. These tools helped in visualizing the system structure clearly and maintaining consistency in the overall design process.
* **Programming Languages:** The backend of the system is developed using **Python** with the **Django** framework, which handles routing, data processing, and API management. The frontend is implemented using **HTML**, and **Tailwind CSS** to create responsive and interactive user interfaces. JavaScript is also used to manage dynamic user actions such as searching, filtering, and playing trailers.
* **Database Platforms:** The system uses **SQLite** during development and testing due to its simplicity and ease of setup. Django’s ORM (Object-Relational Mapping) is used to manage data models and perform SQL operations. The database stores user data, movie metadata, ratings, favorites, and search history.
* **Dataset:** The movie dataset is curated using public movie databases such as IMDb and OMDb API, which provide comprehensive details like movie titles, genres, directors, cast, plot summaries, and ratings. Additional user interaction data (e.g., favorites and search logs) are generated through application usage.
* **Libraries & Algorithms:** The recommendation system uses scikit-learn for TF-IDF vectorization and cosine similarity computations in content-based filtering. Pandas and NumPy are utilized for data manipulation, and Django ORM is used for querying relational data efficiently. User-based collaborative filtering is implemented by constructing a user-movie matrix based on favorites and applying similarity calculations to recommend movies liked by similar users.

### 4.1.2 Implementation details of modules

**Movie Management Module:**

This module handles movie data, including metadata such as genre, director, and plot.

* Models:
  + Movie: Stores movie details such as title, genre, director, actors, plot, rating, and thumbnail.
  + MovieView: Records which user viewed which movie and when. Used to generate user history.
* Views(views/home\_view.py):
  + home\_view(request): Displays the home page with personalized movie recommendations.
  + movie\_detail\_view(request, movie\_id): Renders detailed info about a movie and shows similar movies .
* Templates:
  + movie\_detail.html → Displays full movie information.
  + movie\_list.html → Shows all movies with sorting and filter options.

**Movie Recommendation Module:**

This module is responsible for generating personalized movie suggestions based on user activity (views, behavior) and movie similarity. It utilizes collaborative filtering (user/item-based) and content-based filtering techniques.

* Models(models/movie\_model.py):
  + Movie: Stores movie metadata such as title, genres, release year, rating, etc.
  + MovieView: Records which user viewed which movie and when. Used to generate user history.
* Views(views/home\_view.py):
  + home\_view(request): Displays the home page with personalized movie recommendations.
  + movie\_detail\_view(request, movie\_id): Renders detailed info about a movie and shows similar movies.
* Utility Functions(utils/recommendation\_utils.py):
  + get\_user\_based\_recommendations(user): Retrieves movies liked or viewed by users with similar preferences.
  + build\_movie\_similarity\_matrix(): Builds a TF-IDF-based cosine similarity matrix using combined movie metadata for content-based recommendations.
  + build\_user\_movie\_matrix(): Constructs a user-movie matrix where each cell indicates whether a user has favorited a specific movie.
  + get\_similar\_users(user\_id, user\_movie\_matrix): Calculates and returns a list of users similar to the given user based on their favorite movie patterns.
  + recommend\_movies\_for\_user(user\_id, num\_recommendations): Recommends movies to a user by analyzing favorite movies of other users with similar tastes.
* Templates(templates/movies/):
  + home.html: Displays recommended movies personalized for the logged-in user.
  + movie\_detail.html: Shows selected movie information and suggested similar movies.
  + movie\_card.html: Reusable card component used in lists/grids.
* Flow:
  + User logs in → home\_view is triggered
  + Movie history is fetched via MovieView
  + Recommendations are computed in recommendation\_utils.py.
  + Resulting movies are displayed using home.html and movie\_card.html.

**Authentication and Profile Module:**

Manages user accounts, sessions, and profile settings

* Models (models/auth\_model.py)
  + User: Custom or extended Django user model storing username, email, password, and profile data.
* Views (views/auth\_view.py)
  + register\_view(request): Handles user registration.
  + login\_view(request): Authenticates users.
  + logout\_view(request): Logs out users securely
* Views (views/profile\_view.py)
  + profile\_view(request): Shows user details, viewing history, and saved recommendations.
* Templates (templates/movies/)
  + login.html, register.html: User authentication pages.
  + profile.html: Displays user profile and movie interaction history.
  + settings.html: Lets users manage account details.

**Movie Data Initialization Module**

Used during project setup to populate the database with initial movie data.

* Command File (management/commands/load\_movies.py)
  + Reads from top\_1000\_hollywood\_movies.csv.
  + Extracts each row and creates a Movie instance.
  + Also handles optional genre mapping and deduplication.

## 4.2. Testing

### 4.2.1. Test Cases for Unit Testing:

**User Registration:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. No** | **Test Name** | **Input** | **Expected Output** | **Actual Output** | **Test Result** |
| 1. | Open Registration Page | URL: http://localhost:8000/register/ | Registration Page | Registration Page | Pass |
| 2. | Submit with Missing Email | Name:Ram Email:*blank* Password:Ram@123 Username: ram123 | Show Validation Error | Show validation error | Pass |
| 3. | Valid Registration | Name: Sita Email:sita@gmail.com Password:Sita@123 Username: sita456 | Registration Success | Registration Success | Pass |

**User Login:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Test Name** | **Input** | **Expected Output** | **Actual Output** | **Test Result** |
| 1 | Open Login Page | URL: http://localhost:8000/login/ | Login Page | Login Page | Pass |
| 2 | Invalid Login | Email: fake@gmail.com Password: wrongpass | Show login error message | Show login error message | Pass |
| 3 | Valid Login | Email: sita@gmail.com Password: Sita@123 | Redirect to home | Redirect to home | Pass |

**Add to Favorite a movie**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. No** | **Test Name** | **Input** | **Expected Output** | **Actual Output** | **Test Result** |
| 1 | Add Favorite | User clicks “❤️” on Movie ID 101 | Added to favorites | Added to favorites | Pass |
| 2 | Add Duplicate | User clicks “❤️” again on Movie ID 101 | Show already added | Show already added | Pass |
| 3 | |  | | --- | | Add Without Login |  |  | | --- | |  | | Anonymous user tries to favorite Movie ID 101 | Redirect to login | Redirect to login | Pass |

**Rating and Reviewing a Movie**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. No** | **Test Name** | **Input** | **Expected Output** | **Actual Output** | **Test Result** |
| 1 | Add Rating/Review | User: ram123 Movie ID: 105 Rating: 4 Review: "Amazing direction!" | Review saved | Review saved | Pass |
| 2 | Invalid Rating | User: sita456 Movie ID: 102 Rating: 6 Review: "Too good to rate!" | |  | | --- | | Show validation error |  |  | | --- | |  | | Show validation error | Pass |
| 3 | Add Without Login | Guest user tries to submit a review | |  | | --- | | Redirect to login page |  |  | | --- | |  | | |  | | --- | | Redirect to login page |  |  | | --- | |  | | Pass |

### 

### 4.2.2 Test Cases for System Testing

.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. No** | **Test Scenario** | **Input/ Steps** | **Expected Output** | **Actual Output** | **Test Result** |
| 1 | Full User Flow – Register to Login | 1. Visit /register/ 2. Enter valid details 3. Submit form 4. Login with same credentials | Redirect to homepage with success message | Works as expected | Pass |
| 2 | Search and View Movie Detail | 1. Search "Avengers" 2. Click on result 3. Check movie details and similar movies | |  | | --- | | Movie info with related suggestions should show |  |  | | --- | |  | | Works as expected | Pass |
| 3 | Add Movie to Favorites | 1. Login  2. Visit a movie page  3. Click on ❤️ to favorite | Movie added to user’s favorite list | Confirmation toast | Pass |
| 4 | Submit Rating and Review | 1. Login 2. Visit a movie 3. Give rating (1–5) and write review | Review saved and visible under movie | Review appears correctly | Pass |
| 5 | Prevent Duplicate Favorite | 1. Add same movie twice to favorites | Should show "already favorited" or disable icon | Works as expected | Pass |
| 6 | Logout and Restrict Protected Access | 1. Login 2. Logout 3. Try to visit /profile/ or /favorites/ | Redirected to login page | Protected routes blocked | Pass |
| 7 | Handle Invalid Search Term | 1. Enter gibberish like "asdfjkl" in search | Show "No results found" message | Error handled properly | Pass |

# CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATIONS

## 5.1. Lesson Learnt/Outcome:

While building the Movie Recommendation System, we learned a lot about how platforms like Netflix or IMDb actually suggest content to users. We got hands-on experience with both content-based and collaborative filtering — two powerful techniques that use things like genres, plots, and user behavior to make recommendations. This project also helped us improve our skills in working with real-world data using tools like Pandas and Scikit-learn, and we learned how to clean data, calculate similarities, and turn all that into useful results. Structuring our code properly using Django views, models, and utility files taught us the value of writing modular and maintainable code.

In the end, we were able to create a working web app that recommends movies to users based on what they’ve watched or liked. It also lets users search for movies and see trending search terms. The UI is clean, the backend is efficient, and the whole system feels like something you'd actually use. More than just writing code, this project helped us understand how real AI-driven systems work and made us more confident in building smarter, user-friendly web applications in the future.

## 5.2. Conclusion:

The Movie Recommendation System was successfully developed to provide personalized movie suggestions based on user behavior and movie content. By combining content-based filtering (using TF-IDF and cosine similarity) with collaborative filtering (based on user preferences), the system delivers relevant and intelligent recommendations. We also implemented features like search tracking, trending keywords, and user profiles to enhance the overall experience.

This project not only improved our understanding of recommendation algorithms but also helped us apply machine learning concepts in a real web application using Django. It reflects the potential of combining AI with user-centered design to create smarter platforms. Overall, the project met its objectives, remained within scope and schedule, and provided a solid foundation for further enhancements like ratings, reviews, or deep learning-based recommendations in the future.

## 5.3. Future Recommendations

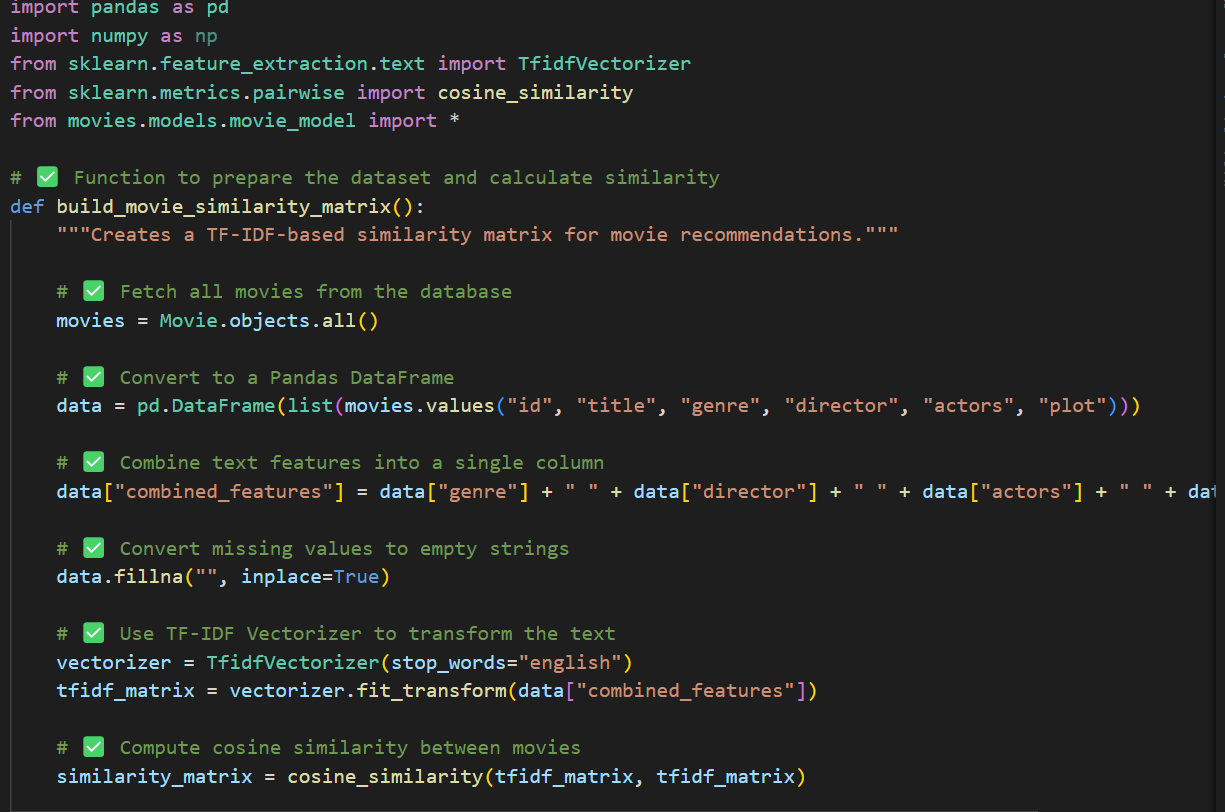
While the current system meets its core objectives, there are several ways it can be enhanced in the future:

* **Hybrid Recommendation Model**: Combining content-based and collaborative filtering into a hybrid model can further improve accuracy and personalization.
* **Deep Learning Integration**: Using deep learning models like autoencoders or neural collaborative filtering could provide more advanced and scalable recommendations, especially with larger datasets.
* **Improved Search and Filters**: Enhancing the search feature with filters for genre, language, release year, and popularity can make it more user-friendly and dynamic.
* **Mobile App Version**: Creating a mobile version of the system (using React Native or Flutter) could make the platform more accessible to a wider audience.

# References

|  |  |
| --- | --- |
| [1] | G. T. A. Adomavicius, "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE Transactions on Knowledge and Data Engineering,* 2005. |
| [2] | J. O. F. H. A. B. J. Bobadilla, "A Collaborative Filtering Approach to Mitigate the New User Cold Start Problem," *Knowledge-Based Systems,* no. Elsevier, 2012. |
| [3] | R. Burke, "Hybrid Recommender Systems: Survey and Experiments," *User Modeling and User-Adapted Interaction,* no. Springer, 2002. |
| [4] | C. A. H. N. Gomez-Uribe, "The Netflix Recommender System: Algorithms, Business Value, and Innovation," *ACM Transactions on Management Information Systems (TMIS),* no. ACM, 2015. |
| [5] | M. B. D. Pazzani, Content-Based Recommendation Systems, Berlin: Springer, 2007. |
| [6] | F. R. L. S. B. Ricci, Recommender Systems Handbook, Boston: Springer, 2011. |
| [7] | B. K. G. K. J. R. J. Sarwar, " Item-based Collaborative Filtering Recommendation Algorithms," in *Proceedings of the 10th International Conference on World Wide Web (WWW '01)*, New York, 2001. |

# APPENDIX



A screen shot of a computer program

AI-generated content may be incorrect.

