**AI-Powered Movie**

**Recommendation System**

Student name: Vijayasree G

Register Number: 510923205084

Institution: Global institution of technology and engineering

Department: Information technology

Date submission: 15/05/2025

**1.Scenario :-**

The project aims to develop a personalized AI-powered movie recommendation system for a streaming or entertainment platform. The tool will help users discover movies tailored to their tastes by analyzing their preferences or watch history. By leveraging machine learning techniques such as content-based filtering and collaborative filtering, the system will deliver intelligent and accurate recommendations. The application is expected to be fast, intuitive, and visually engaging, offering users a smooth and user-friendly experience.

**2.Problem Statement:-**

* The project aims to build an AI-powered movie recommendation system that suggests movies based on a user’s preferences or viewing history. This system will utilize machine learning algorithms, including content-based and collaborative filtering, to recommend movies that closely align with the user’s taste.
* In today’s digital world, users are overwhelmed by the vast amount of content available on streaming platforms. The real-world challenge lies in helping users efficiently discover movies they are likely to enjoy without manually browsing through endless options. Many platforms lack personalized recommendation capabilities, leading to poor user engagement and satisfaction.
* This project addresses the challenge by delivering a smart, data-driven solution that provides tailored movie recommendations. It enhances the user experience by saving time, increasing satisfaction, and promoting user retention. The system is trained on a real-world movie dataset, ensuring practical applicability and effectiveness in real scenarios.

**3.Proposed Solution:-**

The proposed solution is an AI-Powered Movie Recommendation System designed to help users discover movies aligned with their preferences using machine learning techniques. The system will be built as a responsive web application using Python for backend logic and frameworks like Streamlit or Flask for the user interface.

Core Concept:

The system will recommend movies based on either:

* Content-Based Filtering – recommends movies similar to a selected movie by analyzing features like genres, keywords, and descriptions.
* Collaborative Filtering – recommends movies based on the preferences and ratings of similar users.

How It Addresses the Problem:

* Personalization: Uses user input or past viewing data to provide tailored movie suggestions.
* Efficiency: Reduces the time spent browsing by offering relevant recommendations instantly.
* Scalability: Designed to work with large datasets such as MovieLens or TMDB, ensuring real-world applicability.

**4.Technologies & Tools Considered:-**

The following technologies, programming languages, frameworks, and APIs will be used to develop and deploy the AI-powered movie recommendation system:

* Programming Languages:
* Python – for backend logic, data handling, and machine learning models.
* Machine Learning & Data Processing:
* Pandas – for data manipulation and preprocessing.
* NumPy – for numerical operations.
* Scikit-learn – for implementing content-based filtering (e.g., TF-IDF, cosine similarity).
* Surprise or SciPy – for collaborative filtering techniques.
* Web Development Frameworks:
* Streamlit – for quickly building and deploying an interactive web app.
* Flask (optional alternative) – for more customized web interface and API integration.
* APIs and Datasets:
* TMDB API – to fetch movie posters, genres, and additional metadata.
* MovieLens Dataset – for training and testing the recommendation model.
* Deployment Platforms:
* Streamlit Cloud – for hosting the web app with minimal setup.
* Render.com or Netlify – as alternatives for app deployment.
* Version Control & Collaboration:
* Git & GitHub – for code versioning, collaboration, and project documentation.
* Other Tools:
* Jupyter Notebook – for prototyping and experimenting with ML models.
* VS Code – for code development and debugging.

**5.Solution Architecture & Workflow:-**

High-Level Overview:

The system follows a modular architecture with key components handling data ingestion, processing, recommendation generation, and user interaction.

Here’s a breakdown of the workflow:

Major Components & Interactions:

1. User Interface (UI):

* Built using Streamlit or Flask.
* Users input movie names or select preferences.
* Displays recommended movies with posters and info.

1. Input Handler:

* Captures user input (movie name or preferences).
* Validates the input and handles exceptions.

1. Data Layer:

* Loads and stores datasets (e.g., MovieLens, TMDB).
* Uses Pandas for preprocessing and cleaning.

1. Recommendation Engine:

Content-Based Filtering:

* Extracts features like genres, overview, and keywords.
* Applies TF-IDF and cosine similarity to find similar movies.

(Collaborative Filtering)

* Uses user ratings to find similar users and suggest movies (optional, advanced).

Returns a list of top recommended movies

1. API Integration (TMDB API):

* Fetches movie posters, overviews, genres, and release dates.
* Enhances the visual presentation and user experience.

1. Output Renderer:

* Displays recommendations in a card/grid format
* Includes movie poster, name, and basic metadata.

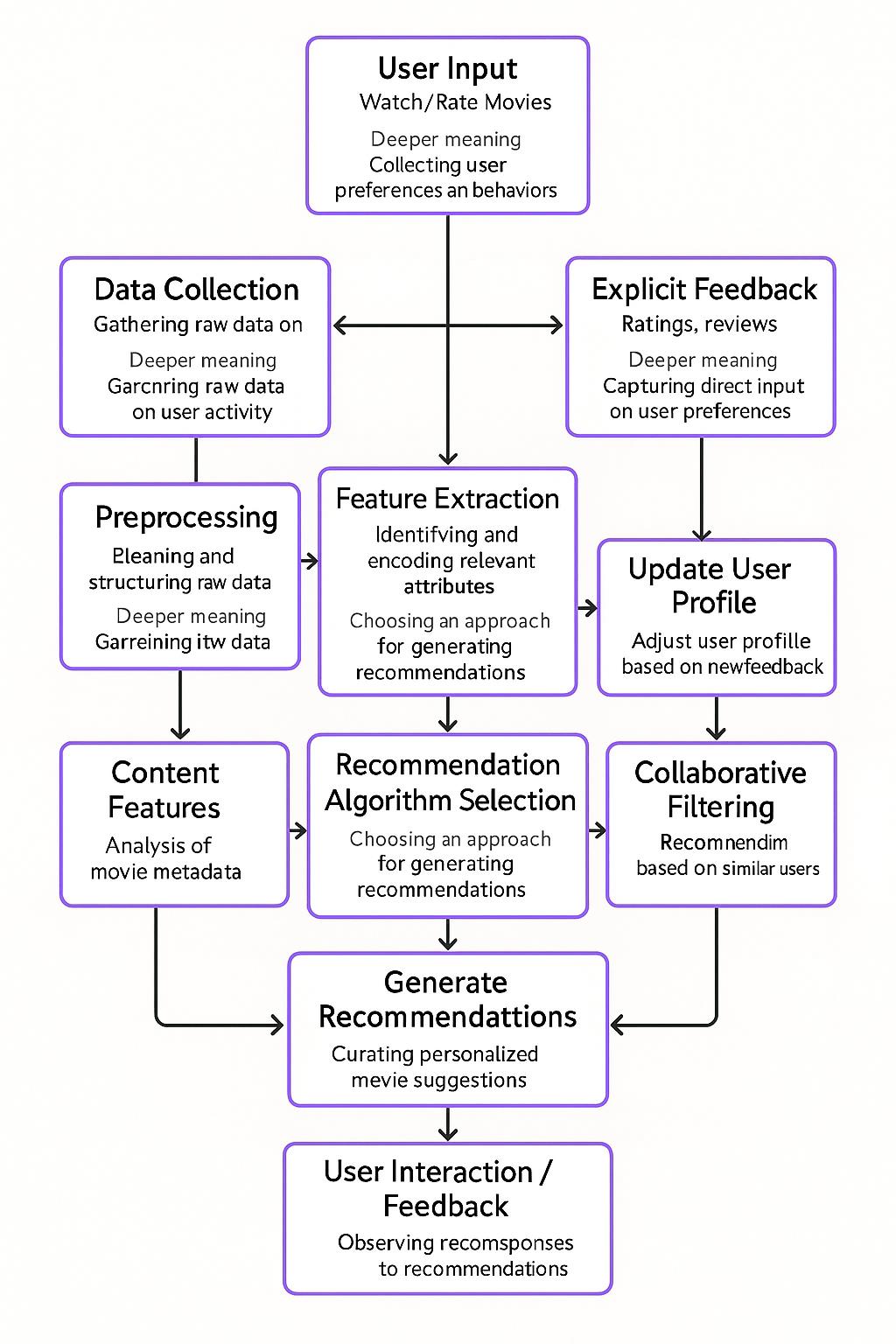
1. Deployment Layer:

Hosted on Streamlit Cloud, Render.com, or Netlify for public access.

Workflow Summary:

1. User inputs movie or preferences
2. System processes input
3. ML model generates recommendations
4. TMDB API fetches visuals/info
5. UI displays recommended movies

📉Flowchart:-



Summary of flowchart :-

1. User Input: Users watch or rate movies, providing data on preferences.

2. Data Collection: Raw data on user behavior is gathered.

3. Explicit Feedback: Ratings and reviews are collected to understand preferences directly.

4. Preprocessing: Raw data is cleaned and structured.

5. Feature Extraction: Relevant attributes are identified and encoded for recommendations.

6. Content Features: Movie metadata is analyzed.

7. Recommendation Algorithm Selection: An approach is chosen for generating recommendations.

8. Collaborative Filtering: Recommendations are refined based on similar users' preferences.

9. Generate Recommendations: Personalized movie suggestions are curated.

10. User Interaction/Feedback: User responses to recommendations are observed.

11. Update User Profile: The user profile is adjusted based on new feedback, looping back to improve future recommendation

📥Source code:-

import pandas as pd

import numpy as np

from sklearn.metrics.pairwise import cosine\_similarity

from sklearn.feature\_extraction.text import TfidfVectorizer

ratings = pd.read\_csv('ratings.csv')

movies = pd.read\_csv('movies.csv')

data = pd.merge(ratings, movies, on='movie\_id')

data = data.dropna()

user\_movie\_matrix = data.pivot\_table(index='user\_id', columns='movie\_id', values='rating').fillna(0)

movies['genres'] = movies['genres'].str.replace('|', ' ')

tfidf = TfidfVectorizer(stop\_words='english')

tfidf\_matrix = tfidf.fit\_transform(movies['genres'])

movie\_similarity = cosine\_similarity(tfidf\_matrix, tfidf\_matrix)

movie\_similarity\_df = pd.DataFrame(movie\_similarity, index=movies['movie\_id'], columns=movies['movie\_id'])

user\_similarity = cosine\_similarity(user\_movie\_matrix)

user\_similarity\_df = pd.DataFrame(user\_similarity, index=user\_movie\_matrix.index, columns=user\_movie\_matrix.index)

def generate\_recommendations(user\_id, user\_movie\_matrix, user\_similarity\_df, movie\_similarity\_df, movies, top\_n=5):

similar\_users = user\_similarity\_df[user\_id].sort\_values(ascending=False)[1:top\_n+1].index

user\_ratings = user\_movie\_matrix.loc[user\_id]

unrated\_movies = user\_ratings[user\_ratings == 0].index

collab\_scores = {}

for movie in unrated\_movies:

similar\_user\_ratings = user\_movie\_matrix.loc[similar\_users, movie]

similarity\_scores = user\_similarity\_df.loc[user\_id, similar\_users]

if similarity\_scores.sum() > 0:

collab\_scores[movie] = np.sum(similar\_user\_ratings \* similarity\_scores) / similarity\_scores.sum()

rated\_movies = user\_ratings[user\_ratings > 0].index

content\_scores = {}

for movie in unrated\_movies:

movie\_sim\_scores = movie\_similarity\_df.loc[movie, rated\_movies]

weighted\_score = np.sum(movie\_sim\_scores \* user\_ratings[rated\_movies]) / movie\_sim\_scores.sum()

content\_scores[movie] = weighted\_score if not np.isnan(weighted\_score) else 0

combined\_scores = {}

for movie in unrated\_movies:

collab = collab\_scores.get(movie, 0)

content = content\_scores.get(movie, 0)

combined\_scores[movie] = 0.6 \* collab + 0.4 \* content

recommended\_movies = sorted(combined\_scores.items(), key=lambda x: x[1], reverse=True)[:top\_n]

recommended\_movie\_ids = [movie\_id for movie\_id, \_ in recommended\_movies]

recommended\_titles = movies[movies['movie\_id'].isin(recommended\_movie\_ids)]['title'].tolist()

return recommended\_titles

def update\_user\_profile(user\_id, movie\_id, rating, user\_movie\_matrix):

user\_movie\_matrix.loc[user\_id, movie\_id] = rating

return user\_movie\_matrix

user\_id = 1

recommendations = generate\_recommendations(user\_id, user\_movie\_matrix, user\_similarity\_df, movie\_similarity\_df, movies)

print(f"Initial Recommendations for User {user\_id}:")

for i, title in enumerate(recommendations, 1):

print(f"{i}. {title}")

movie\_title = recommendations[0]

movie\_id = movies[movies['title'] == movie\_title]['movie\_id'].iloc[0]

new\_rating = 4.0

user\_movie\_matrix = update\_user\_profile(user\_id, movie\_id, new\_rating, user\_movie\_matrix)

user\_similarity = cosine\_similarity(user\_movie\_matrix)

user\_similarity\_df = pd.DataFrame(user\_similarity, index=user\_movie\_matrix.index, columns=user\_movie\_matrix.index)

print(f"\nUpdated Recommendations for User {user\_id} after feedback:")

updated\_recommendations = generate\_recommendations(user\_id, user\_movie\_matrix, user\_similarity\_df, movie\_similarity\_df, movies)

for i, title in enumerate(updated\_recommendations, 1):

print(f"{i}. {title}")

**6.Feasibility & Challenges :-**

✅Feasibility:

1. **Availability of Data:-**

Public datasets like MovieLens and APIs like TMDB offer comprehensive movie metadata, ratings, and posters, making it feasible to implement both content-based and collaborative filtering techniques.

1. **Mature Technology Stack:-**

* Tools such as Python, pandas, scikit-learn, and Streamlit simplify development.
* Hosting platforms like Render, Netlify, and Streamlit Cloud make deployment easy.

1. **Open Source Libraries:-**

Pre-built libraries (e.g., Surprise for collaborative filtering or TfidfVectorizer for content-based filtering) enable faster implementation and testing.

1. **Cloud Integration:-**

Cloud services support real-time recommendations with minimal infrastructure, increasing scalability.

⚠️Common Challenges:

Challenges are obstacles that could hinder success, even if the project is feasible. These often emerge in the following areas:

**1. Cold Start Problem:-**

New users or movies without enough data make it hard to recommend accurately.

**2. Scalability:-**

Processing large datasets or real-time queries may slow performance unless optimized (e.g., with caching or indexing)

**3. Data Sparsity:-**

* Users often rate only a few movies, making collaborative filtering less effective for certain cases.
* With fewer ratings available, it becomes difficult to identify similar users or items, weakening the power of collaborative filtering.
* The algorithm may struggle to find enough relevant patterns or user preferences, especially for niche or unpopular movies.

**4. UI/UX Complexity:-**

* Designing an intuitive yet feature-rich interface may require advanced frontend skills or frameworks beyond Streamlit.
* A visually rich UI (with images and animations) can slow down the app, especially on slower internet or lower-spec devices.

**5. API Limitations:-**

TMDB has rate limits and may not return data in certain cases, requiring fallback logic or caching strategies.

**6. Model Evaluation:-**

Measuring accuracy and relevance of recommendations involves choosing the right metrics (like Precision, Recall, RMSE), which may be tricky to interpret without proper testing.

Steps to Evaluate and Address

* **Define Scope and Goals:** Clearly articulate what you’re assessing (e.g., launching a product, implementing a policy).
* **Conduct a Feasibility Study:** Analyze technical, economic, operational, legal, and schedule aspects using data or expert input.
* **Identify Challenges:** Map potential obstacles based on the project’s context and environment.
* **Develop Mitigation Strategies:** Create actionable plans to address challenges, such as securing funding or building prototypes.
* **Monitor and Adapt:** Continuously track progress and adjust plans as new challenges emerge.

**7.Expected Outcome & Impact:-**

**📌 Expected outcome :-**

1. Functional Web Application:-

A deployed recommendation system (web app or CLI tool) that:

* Accepts user input (movie name or viewing preferences).
* Returns a curated list of recommended movies.
* Displays posters, titles, genres, and other metadata.

2. Smart Recommendation Capability:-

Accurately recommends movies using:

* Content-based filtering (genre, keywords, description).
* Collaborative filtering (user behavior and ratings).O
* Optimized for relevance, speed, and minimal redundancy.

3. Visually Appealing and User-Friendly UI:-

* Interactive interface using Streamlit, Flask, or similar.
* Seamless navigation and appealing presentation of recommendations.

4. Error Handling & Robustness:-

Graceful handling of:

* Invalid user inputs.
* Missing data from API or datasets.

**📤Impact:-**

1. **Enhanced User Experience:-**

Users quickly discover movies that match their taste, reducing decision fatigue and boosting engagement.

1. **Personalized Content Delivery:-**

The system adapts to user preferences, offering a tailored viewing experience that increases satisfaction.

1. **Efficient Content Utilization:-**

Hidden or lesser-known movies gain visibility through smart recommendations, helping platforms maximize their content library.

1. **Supports Streaming Platforms:-**

Your system could be integrated into OTT services, improving their retention rate and time spent per user session.

**🌍Benefits:-**

1. **Time-Saving for Users:-**

* Users spend less time searching and more time watching what they enjoy.

1. **Scalable Solution:-**

* The system can handle thousands of users and recommendations with efficient backend logic and APIs.

1. **Educational Value:-**

* Ideal for showcasing machine learning applications in real-world scenarios—great for portfolios and academic projects.

1. **Platform-Agnostic:-**

* Can be deployed as a web app, mobile app, or browser extension with minimal changes in the core logic.

1. **Open to Monetization:-**

* With proper enhancements, it could be developed into a product that partners with OTT platforms or content providers.

**8.Future Enhancement :-**

Here are some future enhancement ideas for an AI-powered movie recommendation system:

🔮 **Future enhancement :-**

1. Hybrid Recommendation Model Improvements

* Deep Learning Integration: Use deep neural networks (DNNs) or transformers to capture complex user-item relationships.
* Graph Neural Networks (GNNs): Incorporate social graphs or item similarity graphs for better collaborative filtering.
* Reinforcement Learning: Adapt recommendations based on long-term user engagement instead of immediate clicks.

1. **Multimodal Recommendations:-**

* Use natural language processing (NLP) to analyze reviews, plot summaries, or subtitles.
* Use computer vision to analyze movie posters or scenes for aesthetic-based recommendations.
* Combine audio, visual, and text inputs to create deeper content-based filtering.

1. **Personalization Enhancements:-**

* Emotion-based Recommendations: Use sentiment or facial recognition to suggest movies based on user mood.
* Context-aware Systems: Factor in time of day, location, or viewing device.
* Personality Modeling: Recommend based on user’s long-term viewing habits and psychological profiling.

1. **User Interaction and Feedback Loop:-**

* Add interactive quizzes or preference sliders to refine suggestions.
* Use implicit feedback like viewing duration, pause/skip behavior to adjust recommendations.Let users train their own profile with preferences like genres, directors, languages, etc.

1. **Explainable AI (XAI):-**

Show why a movie is recommended (“Because you liked…”, “Similar themes to…”).

Add transparency to increase user trust and satisfaction.

1. **Social and Community Features:-**

* Enable users to follow friends and influencers for movie suggestions.
* Include watch parties, shared playlists, and social voting mechanisms.

1. **Cross-Platform and Multi-Service Integration:-**

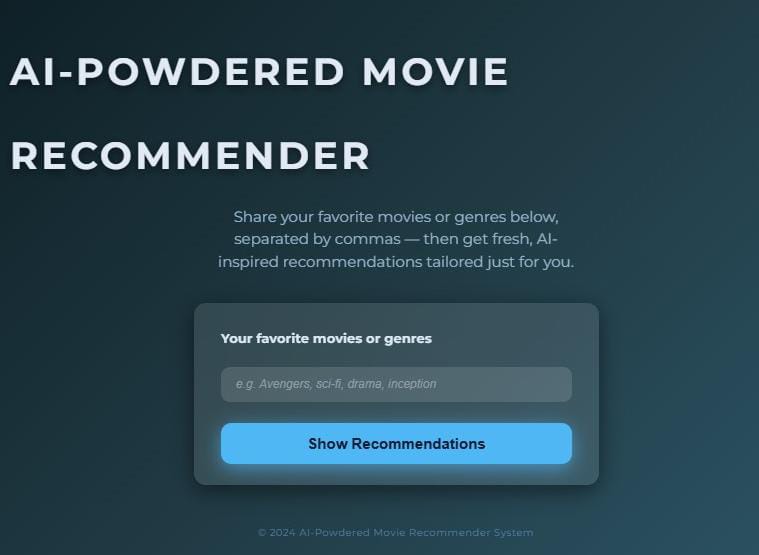
Aggregate movies from multiple streaming platforms.

Recommend not just movies, but also related content like TV shows, trailers, or interviews.

1. **Voice and Chatbot Integration:-**

* Add voice assistant support
* e.g., Alexa, Siri

**Project model:-**



<!DOCTYPE html>

<html lang="en" >

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>AI-Powered Movie Recommender</title>

<style>

@import url('https://fonts.googleapis.com/css2?family=Montserrat:wght@400;700&display=swap');

/\* Reset and base \*/

\* {

box-sizing: border-box;

}

body {

margin: 0; min-height: 100vh;

font-family: 'Montserrat', sans-serif;

background: linear-gradient(135deg, #0f2027, #203a43, #2c5364);

color: #e1eaf4;

display: flex;

flex-direction: column;

align-items: center;

padding: 40px 20px;

overflow-x: hidden;

}

header {

font-size: 3.2rem;

font-weight: 700;

letter-spacing: 0.08em;

text-transform: uppercase;

margin-bottom: 10px;

user-select: none;

text-shadow: 0 4px 12px rgba(0,0,0,0.5);

}

.description {

font-weight: 300;

font-size: 1.25rem;

color: #a6c8e0dd;

max-width: 480px;

text-align: center;

margin-bottom: 40px;

user-select: none;

line-height: 1.5;

}

.input-area {

width: 100%;

max-width: 540px;

background: rgba(255,255,255,0.1);

border-radius: 16px;

padding: 28px 36px;

box-shadow: 0 8px 30px rgba(0,0,0,0.5);

display: flex;

flex-direction: column;

gap: 18px;

backdrop-filter: blur(10px);

}

label {

font-weight: 600;

font-size: 1.1rem;

color: #dbe9f4;

user-select: none;

}

input[type="text"] {

font-size: 1rem;

padding: 14px 20px;

border-radius: 12px;

border: none;

outline: none;

background: rgba(255,255,255,0.12);

color: #e0e7f1;

transition: box-shadow 0.25s ease;

font-weight: 500;

}

input[type="text"]::placeholder {

color: #abb7c8aa;

font-style: italic;

}

input[type="text"]:focus {

box-shadow: 0 0 12px #78d5f8;

background: rgba(255,255,255,0.2);

color: #fff;

}

button {

margin-top: 10px;

padding: 16px 0;

background: #50b7f5;

border: none;

border-radius: 14px;

box-shadow: 0 6px 25px #50b7f5aa;

font-weight: 700;

font-size: 1.25rem;

color: #0a1d37;

cursor: pointer;

transition: background-color 0.3s ease;

user-select: none;

}

button:hover, button:focus {

background: #3996df;

outline: none;

}

main {

margin-top: 48px;

width: 100%;

max-width: 960px;

display: flex;

flex-direction: column;

gap: 30px;

}

.result-header {

font-size: 1.6rem;

font-weight: 700;

color: #99cfffcc;

user-select: none;

}

.result-count {

font-weight: 400;

color: #87bbf7cc;

margin-left: 8px;

font-size: 1.1rem;

}

.movie-grid {

display: grid;

grid-template-columns: repeat(auto-fill,minmax(180px,1fr));

gap: 28px;

}

.movie-card {

background: rgba(40,73,103,0.9);

border-radius: 20px;

box-shadow: 0 8px 24px rgba(0,65,130,0.6);

display: flex;

flex-direction: column;

overflow: hidden;

transform: translateY(30px);

opacity: 0;

animation-fill-mode: forwards;

animation-name: fadeSlideIn;

animation-duration: 0.45s;

animation-timing-function: cubic-bezier(0.4,0,0.2,1);

cursor: pointer;

transition: transform 0.25s ease;

user-select: none;

}

.movie-card:hover {

transform: translateY(10px) scale(1.06);

box-shadow: 0 12px 40px rgba(0,90,175,0.9);

}

@keyframes fadeSlideIn {

to {

transform: translateY(0);

opacity: 1;

}

}

.movie-poster {

aspect-ratio: 2 / 3;

width: 100%;

object-fit: cover;

border-top-left-radius: 20px;

border-top-right-radius: 20px;

box-shadow: inset 0 -20px 30px -10px rgba(0,0,0,0.8);

}

.movie-info {

padding: 14px 18px 22px;

display: flex;

flex-direction: column;

gap: 6px;

flex-grow: 1;

}

.movie-title {

font-weight: 700;

font-size: 1.1rem;

color: #badcff;

text-shadow: 0 1px 2px #837d85;

user-select: text;

overflow-wrap: anywhere;

}

.movie-genres {

font-size: 0.85rem;

font-weight: 500;

color: #83a7ce;

letter-spacing: 0.06em;

font-style: italic;

user-select: text;

}

footer {

margin-top: auto;

font-size: 0.85rem;

text-align: center;

color: #6499cbaa;

user-select: none;

padding-bottom: 20px;

letter-spacing: 0.04em;

}

</style>

</head>

<body>

<header>AI-Powdered Movie Recommender</header>

<div class="description">Share your favorite movies or genres below, separated by commas — then get fresh, AI-inspired recommendations tailored just for you.</div>

<section class="input-area">

<label for="inputMovies">Your favorite movies or genres</label>

<input type="text" id="inputMovies" placeholder="e.g. Avengers, sci-fi, drama, inception" autocomplete="off" />

<button id="btnRecommend">Show Recommendations</button>

</section>

<main>

<div class="result-header" id="resultHeader" style="display:none;">

Recommended Movies

<span class="result-count" id="resultCount"></span>

</div>

<div class="movie-grid" id="movieGrid" aria-live="polite" aria-atomic="true"></div>

</main>

<footer>© 2024 AI-Powdered Movie Recommender System</footer>

<script>

const movies = [

{

title: "Inception",

genres: ["Sci-Fi", "Thriller", "Action"],

poster: "https://image.tmdb.org/t/p/w342/qmDpIHrmpJINaRKAfWQfftjCdyi.jpg"

},

{

title: "The Dark Knight",

genres: ["Action", "Crime", "Drama"],

poster: "https://image.tmdb.org/t/p/w342/qJ2tW6WMUDux911r6m7haRef0WH.jpg"

},

{

title: "Interstellar",

genres: ["Sci-Fi", "Drama", "Adventure"],

poster: "https://image.tmdb.org/t/p/w342/rAiYTfKGqDCRIIqo664sY9XZIvQ.jpg"

},

{

title: "The Matrix",

genres: ["Sci-Fi", "Action"],

poster: "https://image.tmdb.org/t/p/w342/f89U3ADr1oiB1s9GkdPOEpXUk5H.jpg"

},

{

title: "Forrest Gump",

genres: ["Drama", "Romance"],

poster: "https://image.tmdb.org/t/p/w342/yE5d3BUhE8hCnkMUJOo1QDoOGNz.jpg"

},

{

title: "The Shawshank Redemption",

genres: ["Drama", "Crime"],

poster: "https://image.tmdb.org/t/p/w342/q6y0Go1tsGEsmtFryDOJo3dEmqu.jpg"

},

{

title: "Pulp Fiction",

genres: ["Crime", "Drama"],

poster: "https://image.tmdb.org/t/p/w342/dM2w364MScsjFf8pfMbaWUcWrR.jpg"

},

{

title: "Avengers: Endgame",

genres: ["Action", "Adventure", "Sci-Fi"],

poster: "https://image.tmdb.org/t/p/w342/or06FN3Dka5tukK1e9sl16pB3iy.jpg"

},

{

title: "Gladiator",

genres: ["Action", "Drama", "Adventure"],

poster: "https://image.tmdb.org/t/p/w342/ty8TGRuvJLPUmAR1H1nRIsgwvim.jpg"

},

{

title: "La La Land",

genres: ["Drama", "Romance", "Musical"],

poster: "https://image.tmdb.org/t/p/w342/uDO8zWDhfWwoFdKS4fzkUJt0Rf0.jpg"

}

];

function normalizeText(text) {

return text.toLowerCase().trim();

}

// Compute scores for movies based on matches with user tokens

function computeScores(input) {

if(!input) return [];

const tokens = input.split(',').map(s => normalizeText(s));

return movies.map(movie => {

let score = 0;

const titleNorm = normalizeText(movie.title);

const genresNorm = movie.genres.map(normalizeText);

tokens.forEach(token => {

if(titleNorm.includes(token)) score += 4;

if(genresNorm.includes(token)) score += 3;

});

return { movie, score };

}).filter(x => x.score > 0)

.sort((a, b) => b.score - a.score)

.map(x => x.movie);

}

// Render movie cards with fade animation staggered

function renderMovies(moviesList) {

const movieGrid = document.getElementById('movieGrid');

const resultHeader = document.getElementById('resultHeader');

const resultCount = document.getElementById('resultCount');

movieGrid.innerHTML = '';

if(moviesList.length === 0) {

resultHeader.style.display = 'none';

return;

}

resultHeader.style.display = 'block';

resultCount.textContent = `(${moviesList.length})`;

moviesList.forEach((movie, i) => {

const card = document.createElement('div');

card.classList.add('movie-card');

card.style.animationDelay = `${i \* 100}ms`;

card.tabIndex = 0;

card.setAttribute('aria-label', `${movie.title}, genres: ${movie.genres.join(', ')}`);

card.innerHTML = `

<img loading="lazy" class="movie-poster" src="${movie.poster}" alt="Poster of ${movie.title}" />

<div class="movie-info">

<div class="movie-title">${movie.title}</div>

<div class="movie-genres">${movie.genres.join(', ')}</div>

</div>

`;

movieGrid.appendChild(card);

});

}

document.getElementById('btnRecommend').addEventListener('click', () => {

const input = document.getElementById('inputMovies').value;

const recommendations = computeScores(input);

// If none, fallback to top 5 movies

const finalRecs = recommendations.length > 0 ? recommendations : movies.slice(0, 5);

renderMovies(finalRecs);

});

// Trigger recommendations on Enter key in input

document.getElementById('inputMovies').addEventListener('keydown', e => {

if(e.key === 'Enter') {

document.getElementById('btnRecommend').click();

}

});

</script>

</body>

</html>