

Obamela Moganedi

48332704

Practical Lab 4 Report

1. Introduction

This project implements a movie recommender engine that accepts natural-language queries (such as single words, phrases, genres, cast names, years, or full sentences) and returns ranked movie recommendations drawn from a local dataset of over 300 entries. The recommender logic uses a combined-text TF-IDF representation across key features — movie title, genre, cast, plot, and year — and ranks movies by cosine similarity to the user's query.

The system is presented through a single-page Flask web application that mimics a simplified Netflix-style interface.

This report documents both the technical description of the construct and detailed prompt documentation showing how AI (a Large Language Model) was used during development.

2. Working Construct

A. How the app functions

- The app accepts user input as free-text, e.g ‘iron man’, ‘man of steel’.
- It uses TF-IDF vectorization on a combined text feature built from: movie_name, genre, star, description, and year.
-
- The system then computes cosine similarity between the vectorized user input and all dataset entries, returning the top 10 closest matches.
- Finally, the app displays these movies as styled cards showing the movie name, genre, cast, year, and a short description.

3. Prompt documentation

A. Flask App development

1. Basic skeleton of the app: This prompt was made to construct a flask app skeleton on which everything will be implemented on.

Generate a simple GUI using Flask app with:

- index.html (homepage with search)
- style.css (styling)
- script.js (client-side functionality)

Create a Netflix-style interface for the movie recommender.

Excellent start! 🎉

We're setting up the foundation for your Netflix-style movie recommender GUI using Flask, with:

- a homepage (`index.html`) that has a search bar and movie cards,
- a `style.css` file for a cinematic interface, and
- a `script.js` file for interactive behavior (like fetching recommendations and showing loading animations).

Here's the complete first version of the project (the *structure and front end visuals only* — we'll connect the logic later).

Folder Structure

```
FlaskProject/
|__ app.py
|__ combined_file.csv      # (your dataset will go here)
|__ templates/
|   __ index.html
|__ static/
|   __ style.css
|   __ script.js
```

+ Ask anything

Folder Structure

```
FlaskProject/
|__ app.py
|__ combined_file.csv      # (your dataset will go here)
|__ templates/
|   __ index.html
|__ static/
|   __ style.css
|   __ script.js
```

app.py

```
from flask import Flask, render_template, jsonify
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/all_movies')
def all_movie():
    # placeholder sample movies
    movies = [
        {"movie_name": "The Mask", "genre": "Comedy | Fantasy", "year": 1994, "star": "Jim Carrey", "img": "https://www.imdb.com/title/tt0109830/mediaviewer/rm3449400?ref_=tt_mv_mi_sm_1"}, {"movie_name": "Titanic", "genre": "Romance | Drama", "year": 1997, "star": "Leonardo DiCaprio", "img": "https://www.imdb.com/title/tt0117500/mediaviewer/rm3449401?ref_=tt_mv_mi_sm_2"}, {"movie_name": "Black Panther", "genre": "Action | Adventure", "year": 2018, "star": "Chadwick Boseman", "img": "https://www.imdb.com/title/tt6129547/mediaviewer/rm3449402?ref_=tt_mv_mi_sm_3"}, {"movie_name": "Inception", "genre": "Sci-Fi | Thriller", "year": 2010, "star": "Leonardo DiCaprio", "img": "https://www.imdb.com/title/tt1375666/mediaviewer/rm3449403?ref_=tt_mv_mi_sm_4"}]
    return jsonify(movies)

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

+ Ask anything

templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Movie Recommender</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}"/>
</head>
<body>
<div class="container">
    <h1 class="title">Movie Recommender</h1>
    <p class="subtitle">Find your next favorite movie – type a title, genre, or year below!</p>

    <form id="movieForm">
        <input type="text" id="movie_name" placeholder="Search movies..." required>
        <button type="submit">Recommend</button>
    </form>

    <div id="loading" class="hidden">
        <div class="spinner"></div>
        <p>Fetching recommendations...</p>
    </div>

    <div id="results" class="movie-list"></div>
</div>

<script src="{{ url_for('static', filename='script.js') }}"/>
</body>
</html>
```

static/style.css

```
body {
```

.card {

```
background: rgba(255,255,255,0.8);
border-radius: 15px;
padding: 15px;
text-align: left;
transition: 0.3s;
```

* static/script.js

```
const form = document.getElementById('movieForm');
const resultsDiv = document.getElementById('results');
const loading = document.getElementById('loading');

// Load all movies initially
window.addEventListener('DOMContentLoaded', loadAllMovies);

async function loadAllMovies() {
    resultsDiv.innerHTML = '';
    loading.classList.remove('hidden');
    try {
        const res = await fetch('/all_movies');
```

Run the app

```
bash
python app.py
```

Then visit → <http://127.0.0.1:5000/>

Would you like the *next step* to be:

1. Adding TF-IDF + cosine similarity logic (so it actually recommends),
or
2. Making the UI more interactive (with Netflix-style sliding rows and movie covers)?

- Recommender logic integration to the flask app: This prompt was meant to take as input the python script that which the dataset and recommendation logic were implemented on and integrate it with the app.py file in the flask app.

[Provided existing recommendation script]
Integrate the script into the app.py folder in the flask app

Perfect 🌟 — let's integrate your existing recommendation logic (the JavaScript you showed earlier) into a Flask app so it becomes part of a full working movie recommender system.

Below is the complete `app.py` file that works with your frontend (`index.html`, `style.css`, `script.js`). It uses Flask to serve the main page, handle movie recommendation requests, and load all movies.

Here's the complete `app.py`

```
python
from flask import Flask, render_template, request, jsonify
import pandas as pd

app = Flask(__name__)

# ----- Load Dataset -----
# Make sure your CSV is in the same folder (e.g., movies.csv)
movies = pd.read_csv('movies.csv')
```

[Provided existing recommendation script]
Integrate the script into the app.py folder in the flask app

Perfect 🌟 — let's integrate your existing recommendation logic (the JavaScript you showed earlier) into a Flask app so it becomes part of a full working movie recommender system.

Below is the complete `app.py` file that works with your frontend (`index.html`, `style.css`, `script.js`). It uses Flask to serve the main page, handle movie recommendation requests, and load all movies.

Here's the complete `app.py`

```
python
from flask import Flask, render_template, request, jsonify
import pandas as pd

app = Flask(__name__)

# ----- Load Dataset -----
# Make sure your CSV is in the same folder (e.g., movies.csv)
movies = pd.read_csv('movies.csv')

# Optional: remove duplicates
movies.drop_duplicates(subset=['movie_name'], inplace=True)

# ----- Helper: Recommend Movies -----
def recommend_movies(movie_name):
    movie_name = movie_name.lower()
    results = []

    for _, row in movies.iterrows():
        # Match based on partial title or genre or cast
        if (movie_name in str(row['movie_name']).lower() or
            movie_name in str(row['genre']).lower() or
            movie_name in str(row['star']).lower()):
            results.append({
                'movie_name': row['movie_name'],
                'genre': row['genre'],
                'star': row['star'],
                'year': row['year'],
                'description': row['description']
            })
    return results[:10] # Limit to 10 results

# ----- ROUTES -----
# + Ask anything
```

The screenshot shows a code editor interface with two main sections. The top section displays Python code for a Flask application. The bottom section shows the project's folder structure and integration logic.

```

# ----- ROUTES -----
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/recommend', methods=['POST'])
def recommend():
    movie_name = request.form.get('movie_name', '').strip()
    if not movie_name:
        return jsonify({'error': 'Please enter a movie name.'})

    recs = recommend_movies(movie_name)
    return jsonify({'recs': recs})

@app.route('/all_movies')
def all_movies():
    data = movies.sample(n=min(20, len(movies))).to_dict(orient='records')
    return jsonify(data)

# ----- MAIN -----
if __name__ == '__main__':
    app.run(debug=True)

```

Folder structure:

```

cp
└── movie_recommender/
    ├── app.py
    ├── movies.csv
    ├── static/
    │   ├── style.css
    │   └── script.js
    └── templates/
        └── index.html

```

Integration logic:

The prompt output was used as a guide as to how to correctly have the logic functional and connected to the html, css, and js files without errors.

No improvements were made to the prompt.

4. Final Reflections.

Throughout this project, prompt engineering played a major role in shaping both the recommender system and its interface. The AI assistant was instrumental in generating the Flask framework structure, front-end layout (HTML/CSS/JS), and the integration of the movie recommendation logic. It quickly scaffolded the core components — such as form handling, loading animations, and card-based display of results — which greatly accelerated development. The main strengths of using prompts were the clarity of responses and the ability to refine code step by step until it matched the intended “Netflix-style” aesthetic.

However, several iterations were required to fine-tune the design. Early prompts produced functional but plain interfaces, so follow-up prompts specifying design intent (“make it vibrant,” “consistent card height,” “add background blur,” etc.) significantly improved the outcome. The most effective strategy was to use short, goal-specific prompts instead of long, complex ones — especially when styling. In future iterations, I would further refine this process by experimenting with visual prompt tools or generative design AIs for faster UI prototyping, and by documenting each iteration more systematically to capture the evolution from prompt to implementation.