Presentation on

# FLICKPICK – YOUR SMART MOVIE COMPANION
# A PERSONALIZED MOVIE RECOMMENDATION SYSTEM


**BY**

**VAISHNAVI PANDEY** **2214670010047**
**PRIYANKA PATEL** **2214670010041**

**Institute of Engineering and Technology**

Deen Dayal Upadhaya Gorakhpur University

Under Guidance Of

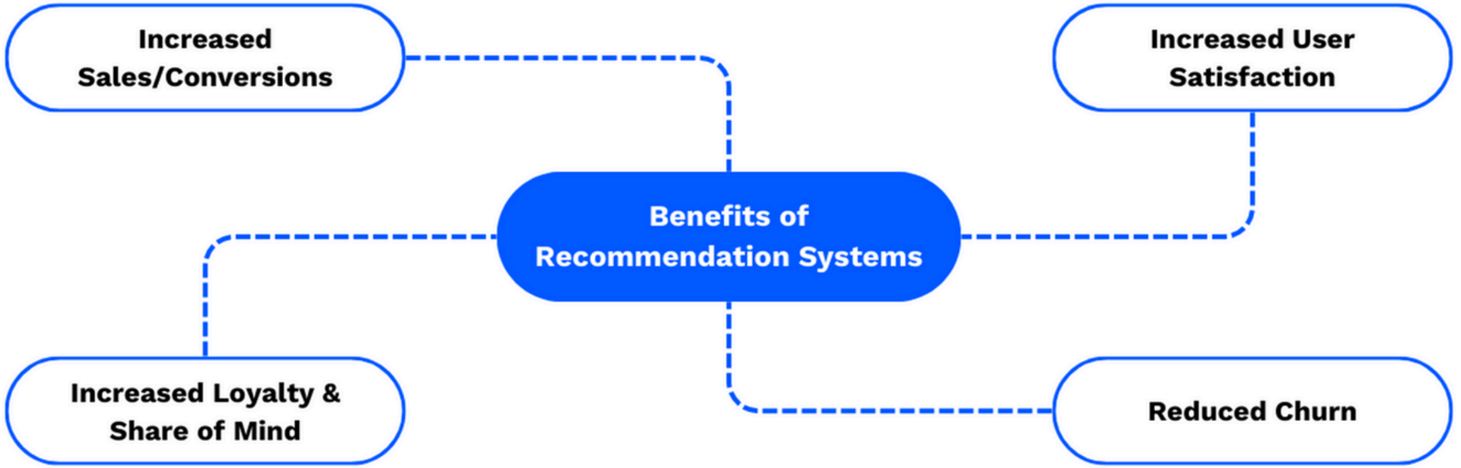
**Mr. Suryabhan sir**

**Mrs. Shivangi Srivastav**

# TABLE OF CONTENTS

# Introduction

The FlickPick movie recommendation system is designed to deliver personalized movie suggestions by combining content-based filtering and collaborative filtering techniques.It uses the Singular Value Decomposition (SVD) (Koren et al., 2009) [1], (Burke, 2002) [3] algorithm for collaborative filtering and cosine similarity for content-based filtering.The hybrid model enhances accuracy and personalization.The project is deployed using Flask on the backend and hosted via Render, with the frontend styled using React and Tailwind CSS and deployed on Netlify.



# System Architecture

**Frontend:**

- Developed using React.js and Tailwind CSS
- Contains stylish components like animated backgrounds and hoverable movie cards
- Enables user interaction and displays real-time recommendations

**Backend:**

- Built using Flask
- Hosts machine learning models for recommendation
- Connects with TMDB API (TMDB API Documentation, 2024) [5] for dynamic movie data

**Preprocessing:**

- Processes movie metadata and ratings
- Generates TF-IDF vectors for content similarity
- Creates user-item rating matrices for collaborative filtering

# System Workflow

Input Stage:

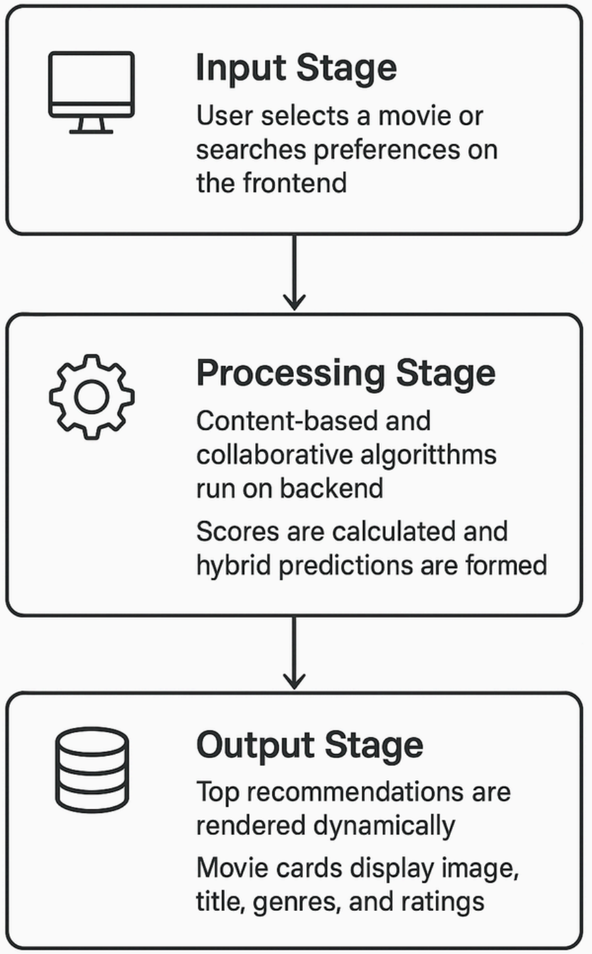
- User selects a movie or searches preferences on the frontend

Processing Stage:

- Content-based and collaborative algorithms run on backend
- Scores are calculated and hybrid predictions are formed

Output Stage:

- Top recommendations are rendered dynamically
- Movie cards display image, title, genres, and ratings



# Recommendation Techniques (Content-Based )

**Content-Based Filtering (Using NLTK):**

- Recommends movies based on similarity in features like overview, genres, keywords, director, cast.(Lops et al., 2011) [2], (Scikit-learn Developers, 2024) [7]
- Uses Natural Language Toolkit (NLTK) for text preprocessing and feature extraction.
- Computes similarity using cosine similarity on processed data.

```
get_recommendations('Inception').head(10)
```

```
6623                The Prestige
8613                Interstellar
2085                   Following
6218              Batman Begins
8031    The Dark Knight Rises
6981           The Dark Knight
4145                    Insomnia
3381                     Memento
343                      Timecop
8500                     Don Jon
Name: title, dtype: object
```

# Recommendation Techniques (Collaborative)

**Collaborative Filtering (Using SVD):**

- Uses user-item interaction data to predict preferences.
- Implements Singular Value Decomposition (SVD) to identify latent patterns in user ratings.(Koren et al., 2009) [1], (Surprise Library Documentation, 2024) [4]
- Effective in capturing hidden relationships among users and movies.

```
Evaluating SVD
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8979  0.8983  0.8966  0.8982  0.8916  0.8965  0.0025
MAE (testset)     0.6919  0.6915  0.6895  0.6911  0.6889  0.6906  0.0012
Fit time          2.47    3.00    2.82    2.51    3.10    2.78    0.25
Test time         0.44    0.50    5.50    0.30    0.76    1.50    2.00
Mean RMSE: 0.8965302728783288
Mean MAE: 0.6905544865338837
```

# Hybrid Approach Implementation

Our project combines both content-based and collaborative filtering techniques to create a hybrid recommendation system. This approach leverages the strengths of both methods to provide more accurate and diverse recommendations to users.

**Benefits:**

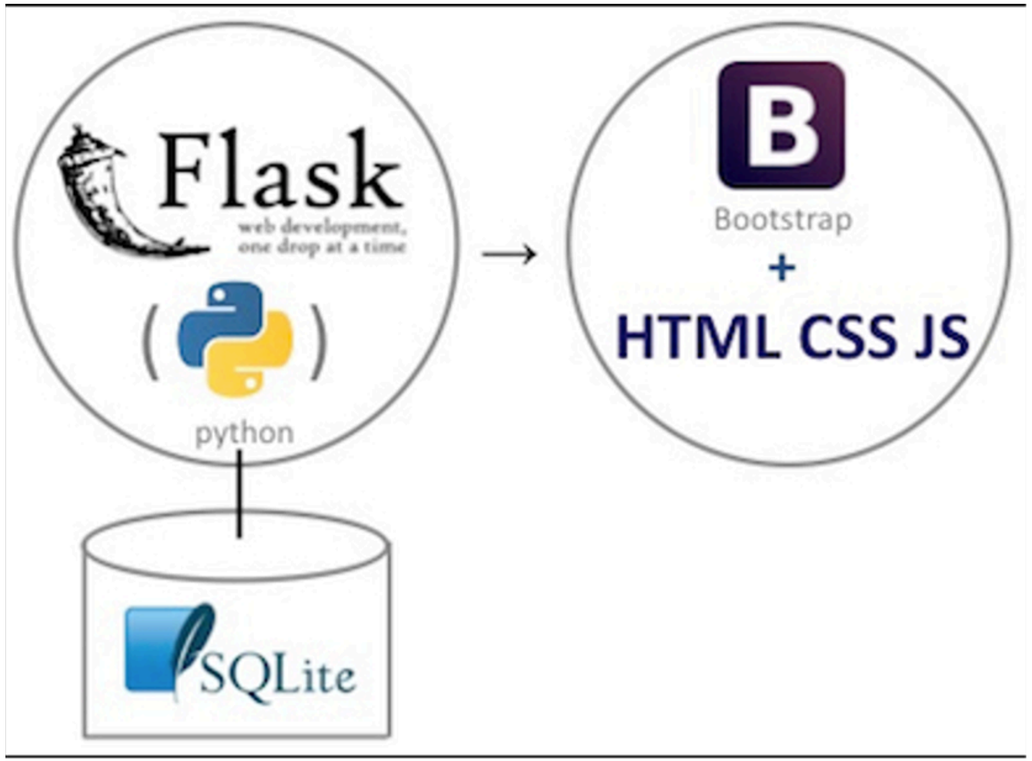
- Improved Accuracy – Better matches to user tastes
- Diverse Suggestions – Goes beyond user's viewing history
- Cold Start Solution – Handles lack of data more gracefully

```
get_recommendations('Inception').head(10)
```

```
6623                The Prestige
8613                Interstellar
2085                   Following
6218              Batman Begins
8031    The Dark Knight Rises
6981             The Dark Knight
4145                     Insomnia
3381                      Memento
343                       Timecop
8500                      Don Jon
Name: title, dtype: object
```

# Flask Integration

Flask connect backend to frontend. (Flask Documentation, 2024) [6]




**Workflow**

- GET Request: Loads homepage with movie input/search bar
- POST Request: Captures input, sends to recommend function
- Returns a list of top recommended movies

**Overview of app.py**

- Central backend logic using Flask
- Connects ML model with frontend
- Fetches movie data from TMDB API and sends results to React

# Frontend

**Technology Stack:**

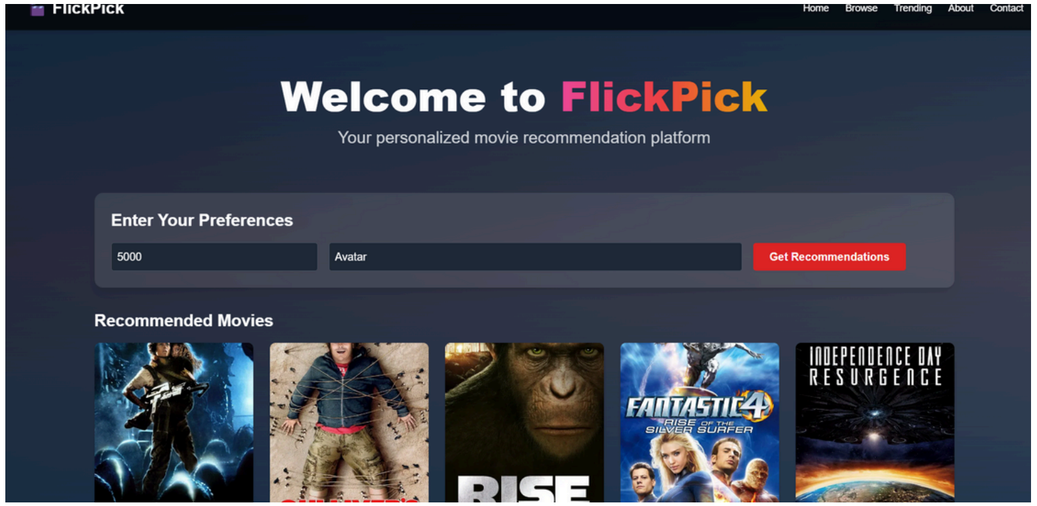
- React.js + Tailwind CSS
- Animated background using react-tsparticles
- Movie cards with hover effects (title, genre, rating, synopsis)
- Sections: Search, Trending, Top Rated, Filters

**UI Highlights:**

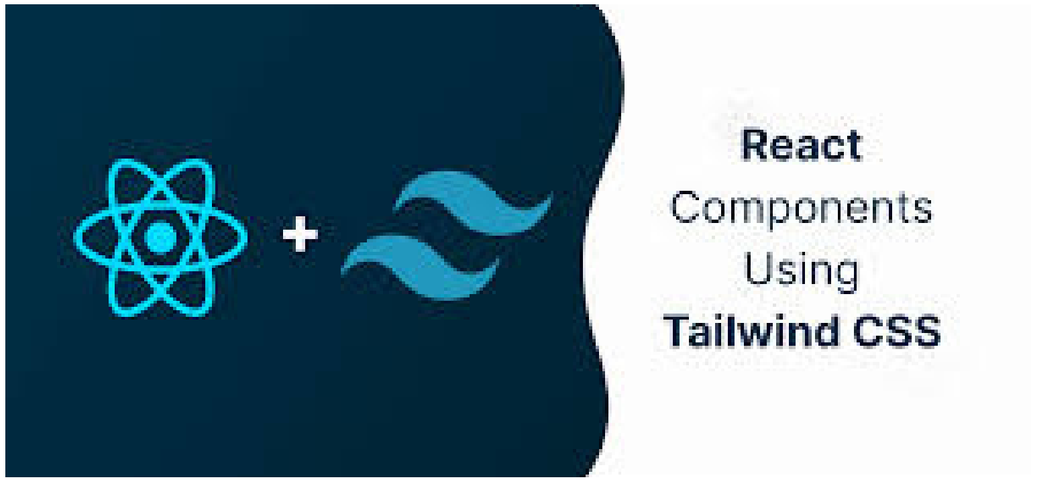
- Sticky navigation bar
- Responsive design
- Custom floating background )
- Smooth transitions for cards and filters

**Deployment:**

- Frontend hosted on Netlify
- API calls to Flask backend on Render




Images and movie data © TMDB – https://www.themoviedb.org



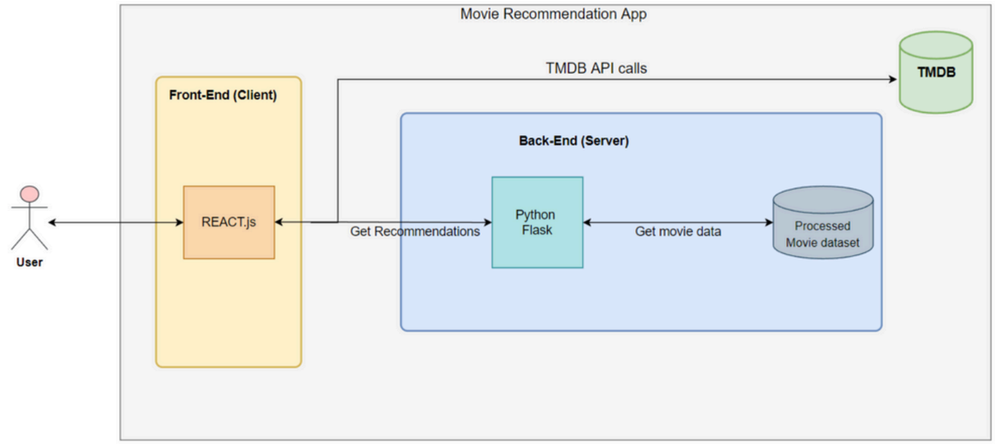
# Deployment

**Frontend Deployment:**

- Deployed using Netlify
- React.js + Tailwind CSS based UI
- Live user interface accessible via public URL
- Handles routing, animations, and fetch calls to backend

**Backend Deployment:**

- Hosted on Render
- Python Flask backend running ML models
- Integrated with TMDB API for real-time movie data
- REST API endpoints used by frontend for recommendations




Architecture diagram from GitHub React-Flask Movie Recommendation App

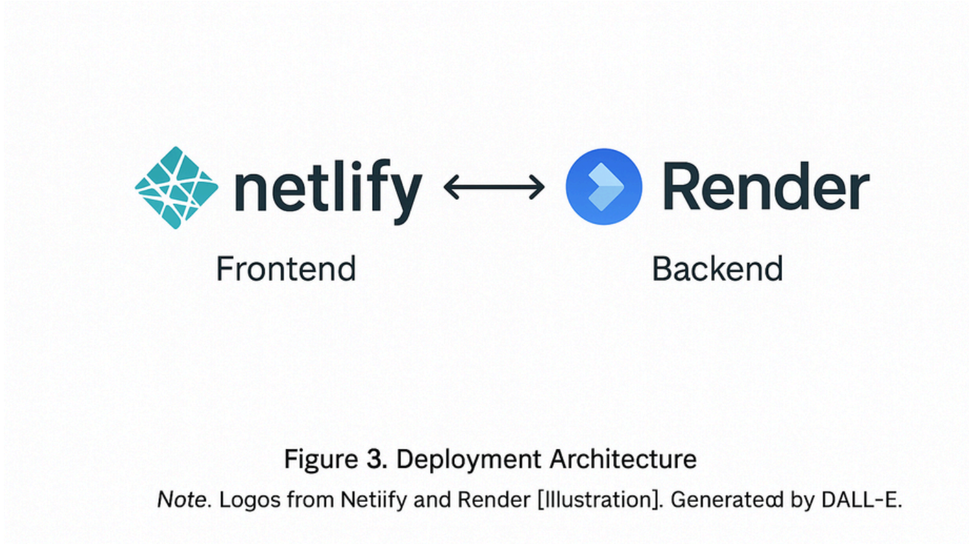
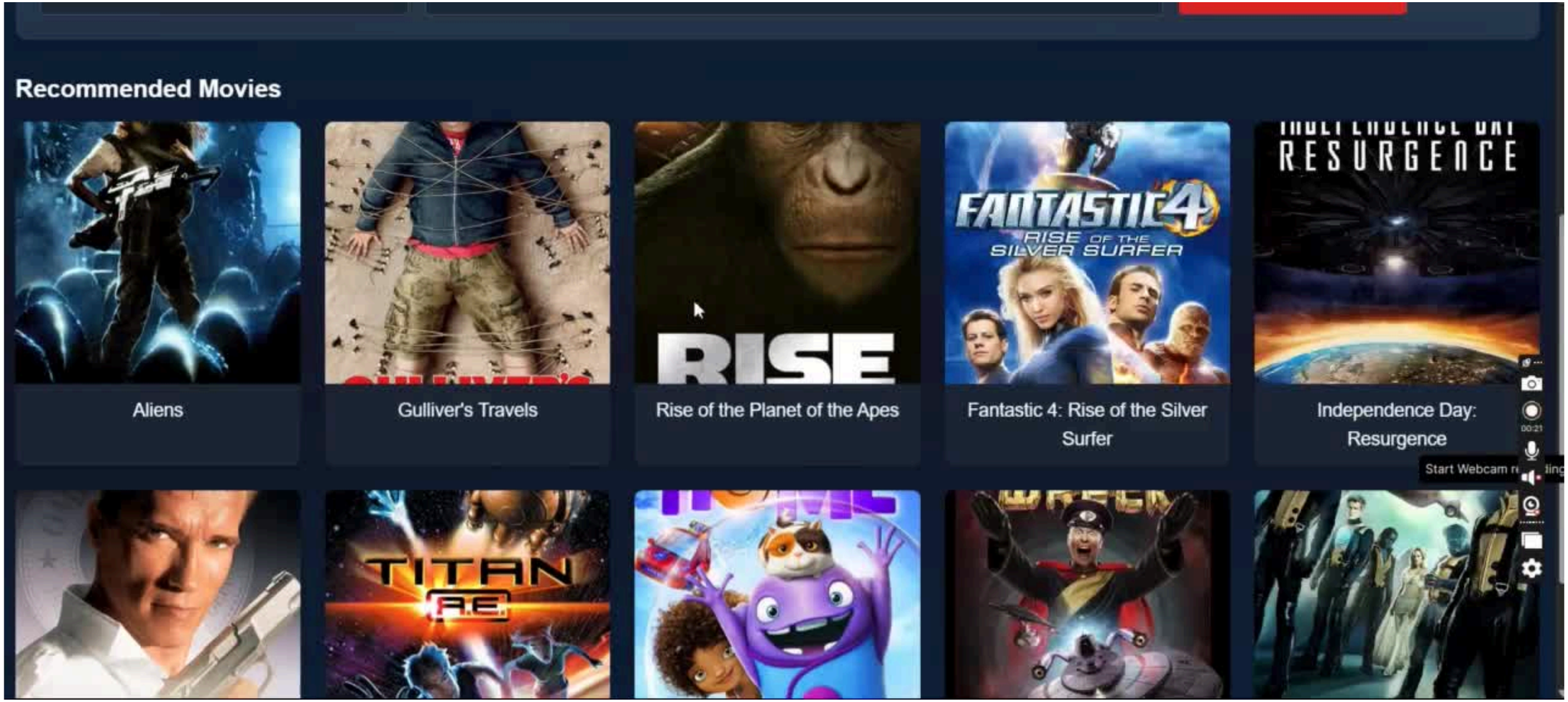



Figure 3. Deployment Architecture

*Note.* Logos from Netiify and Render [Illustration]. Generated by DALL-E.

# Project Demo




Images and movie data © TMDB – https://www.themoviedb.org

# Challenges Faced

- **Data Quality:** Ensuring the quality and completeness of the movie datasets for accurate recommendations.

- **Algorithm Complexity:** Understanding and implementing the SVD algorithm for collaborative filtering.

- **Integration Complexity:** Integrating content-based and collaborative filtering into a hybrid model with seamless functionality.

- **Deployment Challenges:** Overcoming deployment challenges, such as hosting the Flask application on Render and ensuring scalability and performance.

## Future Enhancements

- Add user login system with watch history
- Support user ratings and feedback loop
- Integrate with transformer models like BERT for deeper NLP-based similarity
- Add multi-language support and subtitle preview
- Create mobile app version

## Conclusion

- FlickPick blends smart ML algorithms with a dynamic frontend to deliver accurate and visually appealing movie recommendations.
- Its hybrid engine ensures personalization, while future upgrades aim to make it smarter and more inclusive.

# References


1. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. IEEE Computer, 42(8), 30–37.
2. Lops, P., de Gemmis, M., & Semeraro, G. (2011). Content-based Recommender Systems: State of the Art and Trends. In Recommender Systems Handbook (pp. 73–105). Springer.
3. Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction, 12, 331–370.
4. Surprise Library Documentation. (2024). An Easy-to-Use Python Library for Collaborative Filtering. Retrieved from https://surpriselib.com
5. TMDB API Documentation. (2024). The Movie Database (TMDb) API. Retrieved from https://developers.themoviedb.org
6. Flask Documentation. (2024). Flask: Web Development with Python. Retrieved from https://flask.palletsprojects.com
7. Scikit-learn Developers. (2024). Machine Learning in Python - Scikit-learn Documentation. Retrieved from https://scikit-learn.org

# Thank You