

Overview

|  |  |
| --- | --- |
| ***Project Name*** | Movie Recommendation Website |
| ***Description*** | The **Movie Recommendation Website** is a web-based platform that provides users with personalized movie recommendations. The app uses a machine learning-based recommendation system to suggest similar movies based on user input. It features a modern, responsive frontend built with React and a backend powered by Flask. The backend processes movie data and computes recommendations using text vectorization and cosine similarity. |
| ***Functional Requirements*** | **User Features**   * **Search for Movies**:   + Users can search for movies by entering a title in the search bar. * **View Recommendations**:   + The app provides a list of similar movies based on the user’s search query.   **2. Admin Features**   * **Dataset Management**:   + Admins can update the movie dataset with new titles, genres, or metadata.   **3. Core Backend Functionalities**   * **Recommendation Engine**:   + Compute recommendations using a precomputed similarity matrix. * **APIs**:   + Recommend - Accepts a movie title and returns a list of recommended movies. * **Database Management**:   + Load and fetch movie data from a CSV or relational database.   **4. Core Frontend Functionalities**   * **Interactive UI**:   + A clean, responsive design with React components for seamless interaction. * **Dynamic Results**:   + Display search results and recommendations dynamically without page reloads. * **Navigation**:   + Intuitive routing between home, search results, and watchlist. |
| ***Non-Functional Requirements*** | **1. Performance**   * The app must provide recommendations within **2 seconds** after the user submits a query. * The similarity matrix should be precomputed to ensure fast responses.   **2. Scalability**   * Support a dataset of at least **10,000 movies** with efficient recommendations. * Scalable architecture to handle multiple concurrent users.   **3. Security**   * Protect APIs from unauthorized access.   **4. Compatibility**   * Fully responsive and compatible with modern browsers (Chrome, Firefox, Edge, Safari). |
| ***Dataset Requirements*** | * **Columns**:   + movie title: The name of the movie.   + genre: Genre of the movie (e.g., Action, Comedy).   + Director: Director of the movie.   + actors: Leading actors in the movie.   + comb: A combined text column (e.g., genre + director + actors). * **File Format**: CSV with UTF-8 encoding. * **Sample Entry**:   6f5c885f2b778eb2a50fde6dc8866d81.png |
| ***Use-Case Scenarios*** | **Use Case 1: Searching for a Movie**   1. **Actor**: User 2. **Trigger**: User types a movie name in the search bar and clicks "Search." 3. **Precondition**: Movie dataset is loaded. 4. **Normal Flow**:    * Frontend sends the search query to the backend.    * Backend retrieves movie recommendations and sends them back to the frontend.    * Frontend displays the recommendations to the user. 5. **Postcondition**: User sees a list of recommended movies. |

**Technical Specifications**

**1. Frontend**

* **Framework**: React.js
* **Styling**: Bootstrap or Material-UI for responsive design.
* **State Management**: React Hooks (use State, use Effect) or Context API.
* **API Communication**: Fetch API for sending HTTP requests.
* **Routing**: React Router for navigation between pages.

**2. Backend**

* **Framework**: Flask
* **Data Processing**:
  + **Text Vectorization**: Count Vectorizer from scikit-learn.
  + **Similarity Calculation**: cosine similarity from scikit-learn.
* **APIs**: Flask-RESTful for building REST APIs.
* **Database**:
  + Primary: CSV-based dataset (main\_data.csv).
  + Optional: SQLite or PostgreSQL for persistent storage.
* **Dependencies**:
  + Flask, pandas, scikit-learn, requests

**3. Deployment**

* **Continuous Integration**: GitHub Actions for automated testing and deployment.

**Project Workflow**

**1. Development Workflow**

* **Version Control**: GitHub for source control and collaboration.
* **Scrum Board**:
  + Track tasks with GitHub Projects or Trello.
  + Columns: To Do, In Progress, Done.

**2. Testing**

* **Unit Tests**:
  + Backend: Test recommendation logic and API endpoints using pytest.
  + Frontend: Test components and API integrations using Jest or React Testing Library.
* **Integration Tests**:
  + Verify end-to-end functionality between frontend and backend.

**3. Deployment Workflow**

* Push code to GitHub.
* Use GitHub Actions to:
  + Run tests automatically.
  + Deploy frontend and backend to respective hosting platforms.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Milestone 1: Initial Setup (Week 1)*** | ***Milestone 2: Core Features (Weeks 2-3)*** | ***Milestone 3: Watchlist and Testing (Weeks 4-5)*** | **Milestone 4: Deployment and Finalization (Week 6)** |
| * Set up GitHub repository. * Create initial Flask backend and React frontend structure | * Implement /recommend API in the backend. * Build search bar and results page in the frontend. | * Add watchlist functionality. * Write unit and integration tests. | * Deploy the app to hosting platforms. * Finalize documentation and polish the UI |
|  |  |  |  |
|  |  |  |  |

Conclusion

|  |  |
| --- | --- |
| *Project Aim* | This project aims to deliver a scalable, efficient, and user-friendly movie recommendation app. By leveraging modern technologies and adhering to best practices, the app ensures a seamless experience for users while providing accurate and engaging recommendations. |