**MOVIE RECOMMENDATIONS APP:**

**Project Overview & Objectives:**

**Problem Statement**

With the rapid growth of streaming platforms and the vast amount of movies available, users often struggle to decide what to watch. Traditional browsing methods are time-consuming and overwhelming, as users need to sift through endless options without personalized suggestions. There is a clear need for an intelligent system that understands user preferences and recommends movies accordingly.

**Key Features**

1. **Personalized Recommendations** – Suggest movies based on user history, ratings, and preferences.
2. **Search & Filter Options** – Allow users to search by genre, language, release year, or actors.
3. **User Ratings & Reviews** – Enable users to rate and review movies to improve recommendation accuracy.
4. **Trending & Popular Movies** – Display currently trending, top-rated, and newly released films.
5. **Watchlist Creation** – Let users save movies they plan to watch later.
6. **AI-Powered Suggestions** – Utilize machine learning algorithms (e.g., collaborative filtering, content-based filtering) to enhance accuracy.

**Expected Outcome**

* A **user-friendly platform** that reduces decision fatigue by providing personalized movie suggestions.
* Improved **user engagement** through interactive features such as reviews, watchlists, and trending lists.
* A **smart recommendation system** capable of learning user behavior and refining suggestions over time.
* Ultimately, the app will help users **discover new movies effortlessly**, improving their overall streaming experience.

**Technology Stack & Environment Setup**

**Technology Stack**

1. **Backend**:
   * **Node.js** – Provides a fast and scalable runtime environment for handling API requests.
   * **Express.js** – Lightweight web framework for building RESTful APIs and handling routes.
2. **Frontend**:
   * **React.js** (or Angular/Vue) – For building a responsive and interactive user interface.
   * **HTML5, CSS3, JavaScript** – For designing and structuring UI components.
3. **Database**:
   * **MongoDB** – NoSQL database to store user profiles, movie data, ratings, and watchlists.
   * (Alternative: **MySQL/PostgreSQL** if a relational database is preferred).
4. **Machine Learning Recommendation System**:
   * **Python (scikit-learn / TensorFlow / PyTorch)** – To build and train recommendation models.
   * Integration with Node.js via REST API or microservices.
5. **Other Tools & Services**:
   * **Postman** – For API testing.
   * **Git & GitHub** – Version control and collaboration.
   * **Docker** (optional) – Containerization for easy deployment.
   * **VS Code** – Development IDE.

**Environment Setup**

1. **Install Node.js and npm**
   * Download and install Node.js (includes npm).
   * Verify installation:
   * node -v
   * npm -v
2. **Setup Backend (Express.js)**
3. mkdir movie-recommendation-app
4. cd movie-recommendation-app
5. npm init -y
6. npm install express mongoose cors dotenv
7. **Setup Frontend (React.js)**
8. npx create-react-app client
9. cd client
10. npm start
11. **Database Setup (MongoDB)**
    * Install MongoDB locally or use **MongoDB Atlas** (cloud).
    * Configure connection string in .env file.
12. **ML Model Integration**
    * Build recommendation model in Python.
    * Deploy as a Flask/FastAPI microservice.
    * Connect with Node.js backend via REST API.
13. **Run the Application**
    * Start backend server:
    * node index.js
    * Start frontend server:
    * npm start

**API Design & Data Model**

**Planned REST Endpoints**

1. **Authentication & User Management**
   * POST /api/auth/register → Register new user
   * POST /api/auth/login → User login & JWT token generation
   * GET /api/users/:id → Get user profile
   * PUT /api/users/:id → Update user details
2. **Movies**
   * GET /api/movies → Get all movies (with filters: genre, year, rating)
   * GET /api/movies/:id → Get details of a specific movie
   * POST /api/movies → Add new movie (Admin only)
   * PUT /api/movies/:id → Update movie details
   * DELETE /api/movies/:id → Remove movie (Admin only)
3. **Recommendations**
   * GET /api/recommendations/:userId → Get personalized movie recommendations
   * GET /api/recommendations/trending → Get trending/popular movies
4. **Ratings & Reviews**
   * POST /api/reviews → Submit a rating/review for a movie
   * GET /api/reviews/:movieId → Fetch reviews of a movie
5. **Watchlist**
   * POST /api/watchlist → Add movie to user’s watchlist
   * GET /api/watchlist/:userId → Get user’s watchlist
   * DELETE /api/watchlist/:userId/:movieId → Remove a movie from watchlist

**Request/Response Format**

**Example: Add Review**

**Request (POST /api/reviews)**

{

"userId": "u123",

"movieId": "m567",

"rating": 4.5,

"review": "Great movie with strong performances!"

}

**Response**

{

"message": "Review added successfully",

"review": {

"id": "r101",

"userId": "u123",

"movieId": "m567",

"rating": 4.5,

"review": "Great movie with strong performances!",

"createdAt": "2025-09-28T12:45:00Z"

}

}

**Database Schema (MongoDB Example)**

**Users Collection**

{

"\_id": "u123",

"name": "John Doe",

"email": "john@example.com",

"passwordHash": "...",

"watchlist": ["m567", "m890"],

"createdAt": "2025-09-20T10:30:00Z"

}

**Movies Collection**

{

"\_id": "m567",

"title": "Inception",

"genre": ["Sci-Fi", "Thriller"],

"year": 2010,

"language": "English",

"rating": 8.8,

"reviews": ["r101", "r102"]

}

**Reviews Collection**

{

"\_id": "r101",

"userId": "u123",

"movieId": "m567",

"rating": 4.5,

"review": "Great movie with strong performances!",

"createdAt": "2025-09-28T12:45:00Z"

}

**Front-End UI/UX Plan**

**Wireframes (Conceptual Layouts)**

1. **Home Page**
   * Header with navigation (Home, Search, Recommendations, Watchlist, Profile).
   * Hero section showing trending movies.
   * Sections like *“Recommended for You”*, *“Popular Now”*, *“New Releases”*.
2. **Movie Details Page**
   * Poster image, title, genre, release year, and rating.
   * Synopsis and cast list.
   * Buttons: *Add to Watchlist*, *Rate/Review*.
   * Section for user reviews.
3. **Search & Filter Page**
   * Search bar at the top.
   * Filters: Genre, Year, Language, Rating.
   * Results displayed in grid/list format.
4. **User Profile Page**
   * User info (name, email).
   * Watchlist display.
   * User reviews history.

**Navigation Flow**

* **Login/Register → Home → (Trending/Recommendations)**
* From **Home**:
  + Click movie → **Movie Details Page**
  + Use search bar → **Search & Filter Page**
* From **Movie Details Page**:
  + Add to Watchlist → **Profile → Watchlist**
  + Add Review → Updates **Movie Reviews Section**
* **Profile Page** → Manage personal details, watchlist, and reviews.

This flow ensures **intuitive navigation** with minimal clicks to access key features.

**State Management Approach**

* **React.js + Context API / Redux**
  + **Authentication State**: Logged-in user data, JWT token.
  + **Movies State**: Trending, recommended, and searched movie lists.
  + **Watchlist State**: Add/remove movies in user’s watchlist.
  + **Review State**: Manage submission and fetching of reviews.

**Example:**

* When a user logs in, auth state is stored in Redux/Context.
* Movie recommendations are fetched via API and stored in global state.
* Watchlist updates are reflected across **Home**, **Movie Details**, and **Profile pages** instantly.

**Development & Deployment Plan**

**Team Roles**

1. **Project Manager** – Oversees progress, timelines, and task allocation.
2. **Frontend Developer(s)** – Build user interface using React.js/HTML/CSS, integrate APIs.
3. **Backend Developer(s)** – Develop RESTful APIs with Node.js/Express, handle authentication and recommendation logic.
4. **Database Administrator** – Manage MongoDB schema, optimize queries, and ensure data consistency.
5. **Machine Learning Engineer** – Develop and fine-tune recommendation algorithms (collaborative filtering/content-based/hybrid).
6. **QA/Test Engineer** – Perform manual and automated testing to ensure app reliability.
7. **DevOps/Deployment Engineer** – Set up CI/CD pipelines, manage hosting, and monitor performance.

**Git Workflow**

* **Branching Strategy:**
  + main → Stable production code.
  + develop → Active development branch.
  + Feature branches → For individual features (feature/login, feature/recommendations).
  + hotfix → For urgent bug fixes in production.
* **Workflow Process:**
  + Developer creates feature branch from develop.
  + Implement feature and commit changes.
  + Push branch and create **Pull Request (PR)** for review.
  + Code review by peers.
  + Merge into develop after approval.
  + Deploy stable releases by merging develop into main.

**Testing Approach**

1. **Unit Testing** – Test individual functions (e.g., API endpoints, ML model functions).
2. **Integration Testing** – Ensure backend APIs integrate correctly with the frontend.
3. **UI/UX Testing** – Verify navigation flow, responsiveness, and accessibility.
4. **Performance Testing** – Check API response times, scalability under load.
5. **User Acceptance Testing (UAT)** – Validate the system with real users before deployment.
6. **Automated Testing Tools:** Jest, Mocha, Selenium, Cypress.

**Hosting & Deployment Strategy**

1. **Backend (Node.js/Express)**
   * Host on **Heroku, AWS Elastic Beanstalk, or Render**.
   * Use **Docker** for containerized deployments.
2. **Frontend (React.js)**
   * Deploy on **Vercel, Netlify, or AWS S3 with CloudFront**.
3. **Database (MongoDB)**
   * Use **MongoDB Atlas (Cloud-hosted NoSQL)** for scalability and global access.
4. **Machine Learning Service**
   * Host ML model on a separate **Flask/FastAPI microservice** (AWS Lambda or Google Cloud Functions).
5. **CI/CD Pipeline**
   * GitHub Actions or Jenkins for automated build, test, and deployment.
   * Continuous monitoring with tools like **New Relic / AWS CloudWatch**.