Full Stack Development with MERN

Project Documentation Format

1. INTRODUCTION

1.1. Project Title

i-movies: movie ticket booking system

1.2. Team Members

Riya Mandaogade (Team Lead) – Backend Developer
Chitrarth Shrivas – Frontend Developer
Aastha Pancholi– Database & Deployment
Shubhi Aginihothri – Tester

1.3. Team ID: SWTID1742751842

2. PROJECT OVERVIEW

2.1. Purpose

The primary goal of the MERN Movie App is to offer users a dynamic and seamless way to manage and explore a collection of movies. Built using the powerful MERN stack, this platform emphasizes a rich user experience by allowing individuals to add movies, create watchlists, and browse an extensive catalog. The application is divided into two main parts: an admin dashboard for content management and a user-facing listing app for interaction and exploration.

This system is ideal for film enthusiasts, reviewers, or anyone interested in maintaining a personal or public movie library. The design ensures scalability, responsiveness, and maintainability through a structured MVC architecture.

2.2. Features

Movies Dashboard App (Admin Panel):

1. Comprehensive movie listing dashboard with editing tools
2. Ability to add, update, or delete movie entries
3. Genre management features to organize the movie catalog
4. Search and filtering tools to manage large datasets efficiently
5. Pagination for improved usability and performance

Movies Listing App (User Panel):

1. Visually engaging interface to browse and explore movies
2. Personalized watchlist functionality
3. Filtering by genre and ratings
4. Search capability for quick movie discovery
5. OTP-enabled password reset and secure registration/login

6. Paginated views for optimal performance

Common System Features:

1. Secure JWT-based authentication system
2. User password encryption with Bcrypt
3. Email-based OTP verification via Nodemailer
4. Cloudinary support for poster/image uploads
5. Modern UI developed using Tailwind CSS
6. Axios used for seamless API integration

3. ARCHITECTURE

3.1. Frontend Architecture (React.js)

The frontend is built as a Single Page Application (SPA) using React. All views and UI components are managed through reusable, modular components. State is managed via Context API or Redux depending on the component complexity. React Router is used for client-side routing to ensure smooth navigation.

Key points:

1. Modular structure with components for login, register, dashboard, movie list, etc.
2. Tailwind CSS ensures a responsive, consistent interface
3. Axios for HTTP requests to the backend
4. Custom hooks and context used for state and auth handling

3.2. Backend Architecture (Node.js and Express.js)

The backend follows RESTful API design principles and leverages Express.js for routing and middleware. Controllers are modular and separated by functionality. Authentication is handled via JSON Web Tokens. All sensitive user data is encrypted and protected.

Highlights:

1. Routes: /user, /movies, /genres
2. Middleware for auth, error handling, and request validation
3. Email OTP integration for password reset
4. Environment variables for configuration security

3.3. Database Architecture (MongoDB)

MongoDB serves as the NoSQL database, structured with Mongoose schemas for users, movies, and genres. It supports indexing, filtering, and relation referencing for advanced querying.

Collections:

1. Users: name, email, hashed password, role, watchlist

2. Movies: title, genre, rating, poster, description, timestamp

3. Genres: name, description

4. SETUP INSTRUCTIONS

4.1. Prerequisites

To run the application, ensure the following are installed:

1. Node.js (v16 or above)
2. MongoDB (local or Atlas cloud version)
3. Git (latest version)
4. A modern code editor (e.g., VS Code)

4.2. Installation Steps

Backend Setup:

cd server

npm install

touch .env

.env file content:

PORT=5000

DB_URI=mongodb://localhost:27017/moviedb

SECRET=mysecretkey

CLOUD_NAME=cloudinary_name

API_KEY=cloudinary_api_key

API_SECRET=cloudinary_api_secret

HOST=smtp.gmail.com

EMAIL_PORT=587

SECURE=true

EMAIL=your_email@example.com

MAIL_PASS=your_email_password

npm run dev

Frontend Setup (Dashboard & Listing):

cd client-movies-dashboard-app

npm install

touch .env

`.env` content:

VITE_BASE_URL=http://localhost:5000

VITE_FILTERED_LIST=http://localhost:5000/api/movies/filter

VITE_All_GENRES=http://localhost:5000/api/movies/genre

VITE_LOGIN=http://localhost:5000/api/user/login

VITE_REGISTER=http://localhost:5000/api/user/register

VITE_FORGOT_PASSWORD=http://localhost:5000/api/user/forgot-password

VITE_VERIFY_OTP=http://localhost:5000/api/user/verify-otp

VITE_CHANGE_PASSWORD=http://localhost:5000/api/user/change-password

Repeat similar steps for `client-movies-listing-app`.

npm run dev

5. FOLDER STRUCTURE

Frontend Structure:

/client-movies-dashboard-app

/src

/components -> Reusable UI elements (Navbar, MovieCard)

/pages -> Route views (Login, Dashboard, AddMovie)

/services -> Axios config and API helpers

/assets -> Static files (logos, icons, posters)

App.jsx -> Root component

main.jsx -> Application entry point

Backend Structure:

/server

/controllers -> Logic for movies, genres, users

/models -> Mongoose schemas

/routes -> API route handlers

/middleware -> Auth and error handlers

index.js -> Main server entry

6. API ENDPOINTS (Sample)

User Endpoints:

1. `POST /api/user/register` - Register a new user
2. `POST /api/user/login` - Login
3. `POST /api/user/forgot-password` - Initiate OTP for password reset
4. `POST /api/user/verify-otp` - OTP verification

Movies Endpoints:

1. `GET /api/movies` - List all movies
2. `POST /api/movies` - Add a movie (Admin)
3. `PUT /api/movies/:id` - Edit movie details (Admin)
4. `DELETE /api/movies/:id` - Delete a movie (Admin)

Genres Endpoints:

1. `GET /api/genres` - List all genres
2. `POST /api/genres` - Add a genre
3. `PUT /api/genres/:id` - Edit genre
4. `DELETE /api/genres/:id` - Delete genre

7. AUTHENTICATION

The app uses JWT (JSON Web Token) authentication. Upon successful login, the server issues a token that the client stores in localStorage. This token is included in headers for protected routes. Passwords are hashed using Bcrypt and verified during login. OTP-based flow is used for password reset with token verification.

Security Features:

1. Secure token generation and validation
2. Middleware for route protection
3. Email OTP-based password reset

8. DEMO

Experience our app in action through the following videos:

Admin Panel Demo Video:
Movies Dashboard App

User Panel Demo Video:
Movies Listing App

These videos showcase login, registration, movie CRUD, watchlist handling, and dashboard management in real-time.

9. FUTURE ENHANCEMENTS

The following upgrades are planned for future releases:

1. **User Reviews & Ratings:** Allow users to leave reviews and ratings for movies.
2. **Social Login:** Integration with Google and Facebook for seamless sign-in.
3. **Recommendation Engine:** Suggest movies based on user preferences and history.
4. **Real-Time Notifications:** Alert users for watchlist updates or new releases.
5. **Mobile Application:** Cross-platform app using React Native.
6. **Analytics for Admin:** Graphs for user behavior, genre trends, and watchlist stats.
7. **Accessibility Improvements:** WCAG compliance for better reach.

Enhanced Filtering: Advanced filters using range sliders and multi-tag selection.
Github
DemoVideo
Phase Reports