Movie Watchlist App Documentation

June 20, 2025

1 Overview

The Movie Watchlist App is a full-stack web application designed to manage a list of movies, enabling users to add, view, update, and delete movie entries. The backend is built using Node.js, Express.js, and MongoDB, while the frontend interacts with the backend via RESTful API endpoints. This document details the backend setup, API endpoints, database schema, and instructions for running and testing the application.

2 Project Structure

The project is divided into two main directories:

- **frontend**: Contains the client-side code (e.g., React, Vue, or static HTML/CSS/-JavaScript).
- **backend**: Contains the server-side code, including the Express.js server, MongoDB connection, and API routes.

2.1 Backend Directory Structure

```
movie-watchlist-app/
 backend/
    node_modules/
    package.json
    server.js
    routes/
       movies.js
    models/
       Movie.js
    .env
 frontend/
```

3 Prerequisites

To run the backend, ensure the following are installed:

- Node.js (version 16 or higher recommended)
- npm (included with Node.js)
- MongoDB (local instance or cloud-based, e.g., MongoDB Atlas)
- Git (optional, for version control)

4 Backend Setup

4.1 Dependencies

The backend relies on the following npm packages:

- `express`: Web framework for Node.js.
- `mongoose`: MongoDB object modeling tool.
- `nodemon`: Auto-restarts the server during development.
- `cors`: Enables Cross-Origin Resource Sharing.
- `dotenv`: Loads environment variables from a `.env` file.

Install dependencies by running:

```
cd backend
npm install
```

4.2 Environment Variables

Create a `.env` file in the `backend` directory with:

```
MONGO_URI=<your-mongodb-connection-string>
PORT=5000
```

- `MONGO_URI`: MongoDB connection string (e.g., `mongodb://localhost:27017/movie-watchlist` or MongoDB Atlas URI).
- `PORT`: Server port (default: 5000).

4.3 MongoDB Schema

The MongoDB database uses a `movies` collection with the following schema:

File: `models/Movie.js`

```
const mongoose = require('mongoose');

const movieSchema = new mongoose.Schema({
  title: { type: String, required: true },
  genre: { type: String, required: true },
  watched: { type: Boolean, default: false },
  rating: { type: Number, default: null },
});

module.exports = mongoose.model('Movie', movieSchema);
```

Fields:

- `title`: Movie title (string, required).
- `genre`: Movie genre (string, required).

- `watched`: Whether the movie has been watched (boolean, default: `false`).
- `rating`: User-assigned rating (number, default: `null`).
- `_id`: Auto-generated MongoDB ID.
- `__v`: Version key for document versioning.

5 API Endpoints

The backend provides a RESTful API at `http://localhost:5000/api/movies`.

5.1 GET /api/movies

- **Description**: Retrieves all movies.
- **Method**: GET
- **URL**: `http://localhost:5000/api/movies`
- **Response**:
  - **Status**: 200 OK
  - **Body**: JSON array of movie objects.
  - **Example**:

```
[
  {
    "_id": "6853a930e7c1b0c3f1cc333b",
    "title": "Test Movie",
    "genre": "Drama",
    "watched": false,
    "rating": null,
    "__v": 0

  }
]
```

5.2 POST /api/movies

- **Description**: Adds a new movie.
- **Method**: POST
- **URL**: `http://localhost:5000/api/movies`
- **Request Body**:

```
{
  "title": "New Movie",
  "genre": "Action",
  "watched": false,
  "rating": 8
}
```

- **Response**:
  - **Status**: 201 Created
  - **Body**: Created movie object with `_id`.

5.3 PUT /api/movies/:id

- **Description**: Updates a movie doubtsby ID.
- **Method**: PUT
- **URL**: http://localhost:5000/api/movies/:id
- **Request Body**:

```
{
  "watched": true,
  "rating": 7
}
```

- **Response**:
  - **Status**: 200 OK
  - **Body**: Updated movie object.

5.4 DELETE /api/movies/:id

- **Description**: Deletes a movie by ID.
- **Method**: DELETE
- **URL**: http://localhost:5000/api/movies/:id
- **Response**:
  - **Status**: 200 OK
  - **Body**: Confirmation message or deleted movie object.

6 Server Setup

File: `server.js`

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const dotenv = require('dotenv');
const movieRoutes = require('./routes/movies');

dotenv.config();

const app = express();

app.use(cors());
app.use(express.json());

mongoose
  .connect(process.env.MONGO_URI)
  .then(() => console.log('MongoDB connected'))
  .catch((err) => console.error('MongoDB connection error:', err)
    );

app.use('/api/movies', movieRoutes);

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT
   }`));
```

Note: Remove deprecated `useNewUrlParser` and `useUnifiedTopology` options from `mongoose.connect` to avoid warnings.

7 Running the Application

7.1 Backend

1. Navigate to the `backend` directory:

```
cd C:\Users\lokes\OneDrive\Desktop\movie-watchlist-app\
   backend
```

2. Install dependencies:

```
npm install
```

3. Ensure MongoDB is running.

4. Start the development server:

```
npm run dev
```

   The server runs on http://localhost:5000.

7.2 Frontend

- Navigate to the `frontend` directory:

```
cd ../frontend
```

- Follow frontend-specific setup instructions (e.g., `npm install` and `npm start`).

- Ensure API requests target http://localhost:5000/api/movies.

8 Testing the API

Use tools like Postman or cURL to test endpoints.

Example: GET Movies

```
curl http://localhost:5000/api/movies
```

Response:

```
[
  {
    "_id": "6853a930e7c1b0c3f1cc333b",
    "title": "Test Movie",
    "genre": "Drama",
    "watched": false,
    "rating": null,
    "__v": 0
  }
]
```

Example: POST Movie

```
curl -X POST http://localhost:5000/api/movies -H "Content-Type:
   application/json" -d '{"title":"New Movie","genre":"Action","
   watched":false,"rating":8}'
```

9 Troubleshooting

- **MongoDB Connection Issues**:
  - Verify `MONGO_URI` in `.env`.
  - Ensure MongoDB is running.
  - Check network connectivity for MongoDB Atlas.
- **CORS Errors**:
  - Ensure `cors` middleware is enabled.
  - Configure specific origins if needed:

    ```
    app.use(cors({ origin: 'http://localhost:3000' }));
    ```
- **Deprecated MongoDB Warnings**:
  - Remove `useNewUrlParser` and `useUnifiedTopology` from `mongoose.connect`.
- **API Errors**:
  - Check server logs.
  - Add error-handling middleware:

    ```
    app.use((err, req, res, next) => {
      console.error(err.stack);
      res.status(500).json({ message: 'Server error' });
    });
    ```

10 Future Enhancements

- Add filtering and sorting (e.g., `?watched=true`).
- Implement pagination (e.g., `?page=1&limit=10`).
- Add user authentication (e.g., JWT).
- Use `express-validator` for request validation.
- Deploy to platforms like Heroku, Render, or AWS.

11 Contact

For issues or contributions, contact the project maintainer or open an issue in the project repository.