PROJECT DOCUMENTATION

IMDBID MOVIES APPLICATION

ARAVIND PULLARCOT

07/07/2024

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Final Draft	Aravind Pullarcot	All sections being Filled	07/07/24

Review & Approval

Requirements Document Approval History

Approving Party	Version Approved	Signature	Date

Requirements Document Review History

Reviewer	Version Reviewed	Signature	Date

Contents

1. Introduction

1.1 Introduction

The purpose of this document is to define and describe the requirements of the Movie Review Application project, detailing the system's functionality and constraints. This

document serves as a comprehensive guide for the development, testing, and maintenance phases of the project.

This project is a comprehensive movie review and trailer application built using a robust tech stack that includes Spring Boot, React, MongoDB, and Axios. Spring Boot, a powerful Java-based framework, is utilized for developing the backend services, enabling seamless API creation and database interactions. React, a popular JavaScript library, drives the frontend, offering a dynamic and responsive user interface. MongoDB, a NoSQL database, ensures flexible and scalable data storage, perfectly suited for handling diverse movie data and user reviews. Axios, a promise-based HTTP client, facilitates efficient communication between the frontend and backend, ensuring smooth data fetching and posting operations. This application allows users to browse a catalog of movies, view detailed information, watch trailers, and read and write reviews, providing a rich, interactive user experience.

1.2 Scope of this Document

The users of the system include movie enthusiasts and critics, while the developers of the system are the Shock Force Software Team. This document outlines the requirements and functionality expected from the application, serving as a guide for development and testing.

1.3 Overview

The Movie Review Application allows users to view movie details, watch trailers, and read or write reviews. The backend is built using Spring Boot and MongoDB, while the frontend is developed using React. The application aims to provide a seamless user experience for movie enthusiasts to discover and interact with movie-related content.

1.4 Business Context

The application aims to provide a comprehensive platform for movie enthusiasts to discover movies, view trailers, and share their reviews. It supports the growing community of movie critics and helps users make informed decisions about movies.

2. General Description

2.1 Product Functions

The product allows users to:

- View a list of movies
- Watch movie trailers
- Read reviews from other users
- Write their own reviews The application retrieves movie data from a MongoDB database, providing a dynamic and up-to-date experience for users. It also supports the integration of third-party APIs for additional movie information.

2.2 Similar System Information

There are many movie review systems available, but this product stands out due to its integration with MongoDB and the use of modern web technologies like React for a seamless user experience. This application leverages a microservices architecture, ensuring scalability and maintainability.

2.3 User Characteristics

The users include general movie-goers, enthusiasts, and critics. Users are expected to have basic web navigation skills. The application is designed to be intuitive and user-friendly, catering to users of all ages and technical proficiencies.

2.3 General Constraints

The application should have a user-friendly interface, be accessible on multiple devices, and support both online and offline functionality to some extent. It should also comply with relevant data privacy regulations and ensure secure handling of user data.

3. Technologies Used

2.1 Tech Stack

- Java
- Spring Boot
- MongoDB
- React
- Axios
- React Router
- Material-UI
- Font Awesome
- React Bootstrap

2.2 Backend Setup

Spring Boot Application

Dependencies

We used the following dependencies for the Spring Boot application:

- Spring Boot Starter Data MongoDB
- Spring Boot Starter Web
- Spring Boot Starter Test
- These dependencies were added to the pom.xml file to enable MongoDB support, web functionalities, and testing capabilities.

2.3 Application Properties

The application properties were configured to connect to a MongoDB instance running on localhost with a database named movieDB.

2.4 Model Class

A Movie model class was created to represent the movie entity in the MongoDB database. This class included fields such as id, imdbId, title, releaseDate, trailerLink, poster, backdrops, genres, and reviewIds.

2.5 Repository Interface

A repository interface extending MongoRepository was created to handle data access. This interface included methods to retrieve movies by their imdbId.

2.6 Service Class

A service class was implemented to contain the business logic for retrieving all movies and fetching a specific movie by its imdbId.

2.7 Controller Class

A controller class was created to define the REST API endpoints. This class included endpoints for fetching all movies and retrieving a movie by its imdbId.

2.8 MongoDB setup

To set up MongoDB, we followed these steps:

Installation: MongoDB was installed by following the official MongoDB installation instructions.
Start MongoDB: The MongoDB server was started using the mongod command.
Database and Collection: A database named movieDB and a collection named movies were created using MongoDB Compass or the Mongo shell.

2.9 Front End Setup

- React Application
- Initialize Project
- We created a new React project using Create React App and installed necessary dependencies including Axios, React Router, Material-UI, Font Awesome, and React Bootstrap.

3.0 Project Structure

The project was organized into components for the **header, home page, layout, trailer,** and **reviews.**

Axios Configuration

An Axios instance was configured with a base URL pointing to the backend API.

App Component

The main App.js component was implemented to handle fetching movies from the backend and managing the state of movies, a single movie, and reviews. Routes were defined for the home page, trailer page, and reviews page.

Hero Component

The Hero.js component was implemented to display a carousel of movie posters. It included logic to navigate to the reviews page and display the trailer link.

CSS Styling

The Hero.css file was created to style the movie cards displayed in the carousel. Initially, a gradient overlay was added, but it was later removed to display the full poster image without any gradient.

4. Functional Requirements

1. **Movie Data Storage.**
    1. Movies shall be stored in the MongoDB database.
    2. Movies shall have complete fields such as title, release date, trailer link, poster, genres, and reviews.
    3. High criticality.
    4. Limited network availability could present a technical challenge; ensure data integrity and availability.

2. **Access to movie details**
    1. Users should be able to view movie details and trailers.
    2. High criticality.
    3. The system should provide a responsive and intuitive interface for browsing and searching movies.

3. **Review Functionality**
    1. Users should be able to read and write reviews for movies.
    2. High criticality.
    3. The system should allow for moderation of reviews to ensure quality and appropriateness.

5. Interface Requirements

4.1 User Interfaces

- **4.1.1 GUI**
  The user interface is built using React. It includes forms for submitting reviews and components to display movie details and trailers. The design is responsive, ensuring a consistent experience across devices. Key components include a movie carousel, review submission forms, and a detailed movie page.
- **4.1.2 CLI**
  There is no command line interface
- **4.1.3 API**
  There is no API for the product
- **4.1.4 Diagnostics or ROM**
  There is a troubleshooting and help section provided by Microsoft

4.2 Hardware Interfaces

- The program runs on any device with a modern web browser, leveraging local storage for offline functionality. It does not require any special hardware beyond what is typically available in consumer devices.

4.3 Communications Interfaces

- The application communicates with the backend API through HTTP requests managed by Axios. The API endpoints follow RESTful principles, ensuring clarity and consistency in communication.

4.4 Software Interfaces

- The backend API is developed using Spring Boot and MongoDB, and the frontend interacts with this API using Axios. The application also integrates with third-party movie databases via their APIs to fetch additional movie details and trailers.

6. Performance Requirements

The application should be responsive and load movie details within 2 seconds under normal network conditions. It should be able to handle at least 1000 concurrent users. Performance testing will be conducted to ensure these benchmarks are met

Microsoft lists the requirements for Access 2007 as follows:
500 MHz processor or higher
256MB RAM or higher
1.5GB Available Hard Drive Space
Windows XP SP2 or later operating system.
Windows Office Professional 2007 (Windows Access)
There is also Access Available for Mac OS X, the clients have not stated a need thus far.

7. Other non-functional attributes

.

6.1 Security

- The system shall use HTTPS for secure communication. User data should be stored securely in MongoDB. Authentication and authorization mechanisms will be implemented to protect user data and ensure privacy.

6.2 Binary Compatibility

- The application is compatible with any modern web browser. Cross-browser testing will be conducted to ensure compatibility.

6.3 Reliability

- Regular backups of the MongoDB database will ensure minimal data loss. Thorough testing will ensure reliability. The system should have a high uptime and be resilient to failures.

6.4 Maintainability

- The system shall be maintained by the development team and documented thoroughly for ease of future maintenance. Code should follow industry best practices and be modular to facilitate updates and bug fixes.

6.5 Portability

- The application shall be portable across devices with modern web browsers. The design should be adaptive, providing an optimal user experience on both desktop and mobile devices.

6.6 Extensibility

- The system shall be designed to allow easy addition of new features. The architecture should support plugin-like extensions for future enhancements.

6.7 Reusability

- Code should be modular to allow reusability across different parts of the application. Common components and utilities should be identified and abstracted for reuse.

6.8 Application Affinity/Compatibility

- The system works with any modern web browser and has no specific affinity or compatibility issues. It should also integrate seamlessly with third-party APIs for extended functionality.

6.9 Resource Utilization

- The application should efficiently use network and server resources to ensure fast load times and minimal server load. Proper caching strategies should be implemented to optimize performance.

6.10 Serviceability

- The system should be easily serviceable, with clear logging and error reporting mechanisms in place. A detailed monitoring and alerting system should be implemented to quickly identify and resolve issues.

8. Operational Scenarios

Scenario A: Viewing Movie List

- The user opens the application and views a list of movies. The list is fetched from the MongoDB database and displayed in a carousel format. Users can filter and sort the movies based on various criteria.

Scenario B: Watching a Trailer

- The user clicks on a movie and watches its trailer. The trailer is embedded from a third-party video service, and the application ensures smooth playback and user control options.

Scenario C: Writing a Review

- The user writes and submits a review for a movie. The review is validated and then stored in the MongoDB database. The system provides real-time feedback on the submission status.

9. Trouble shooting

1. Issue: Backdrop Not Displaying

Symptom: The `backdrops` array was `null` in the response.

Solution: Verify the data in the database. Ensure the `backdrops` field is correctly populated in MongoDB. Also, ensure that the data retrieval code in the Spring Boot application correctly fetches this field.

2. Issue: Movie Card Background Gradient

Symptom: The gradient overlay covered half the poster.

Solution: Modify the CSS to remove the gradient overlay:

Symptom: The API endpoint for fetching movies or specific movie details was not responding correctly.

Solution: Ensure the API base URL is correctly configured in the Axios instance. Check for any CORS issues and make sure the Spring Boot application is running.

4. Issue: Conditional Rendering of Movie Data

Symptom: Conditional rendering logic caused issues when `movie` or `backdrops` was `null`.

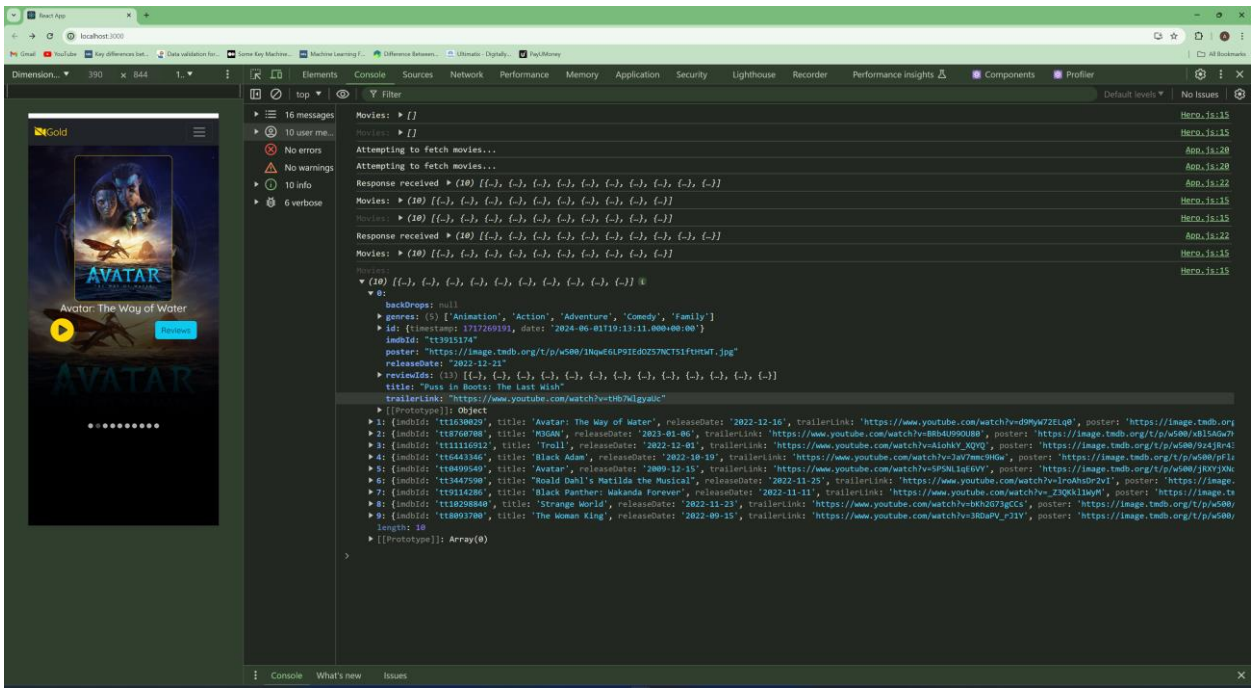
Solution: Remove conditional rendering checks and ensure the data is always structured correctly from the backend.

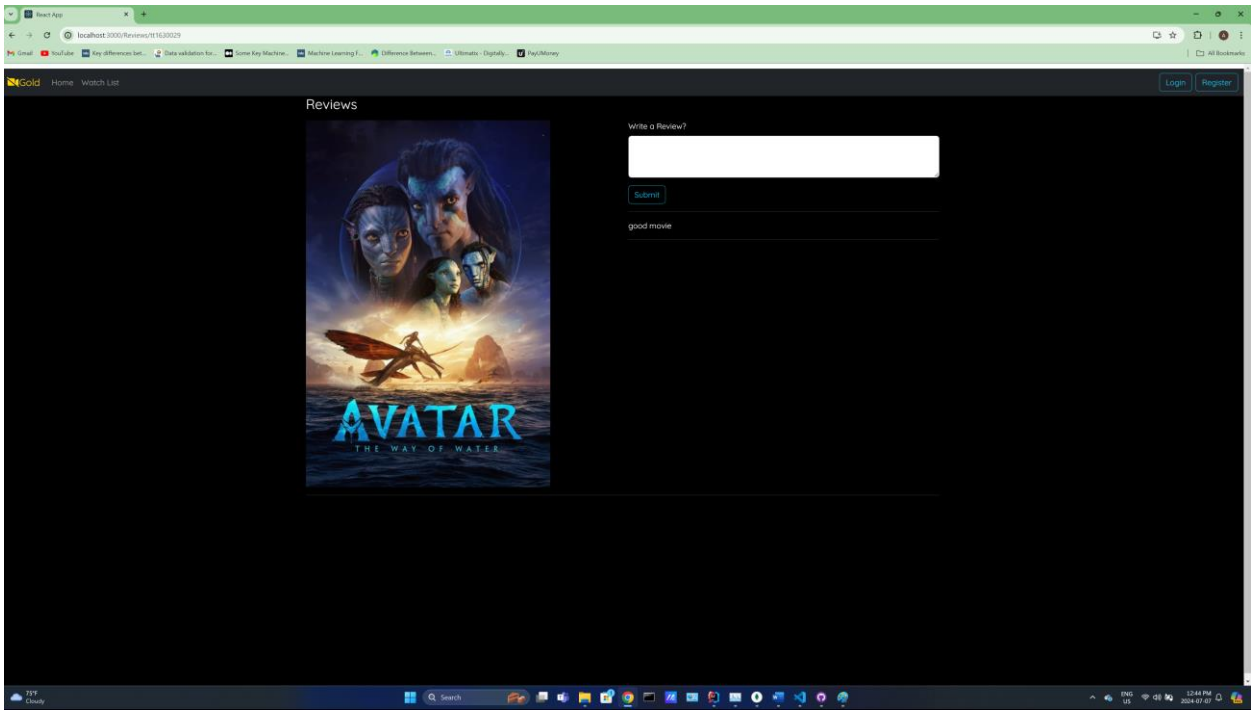
10. Appendices

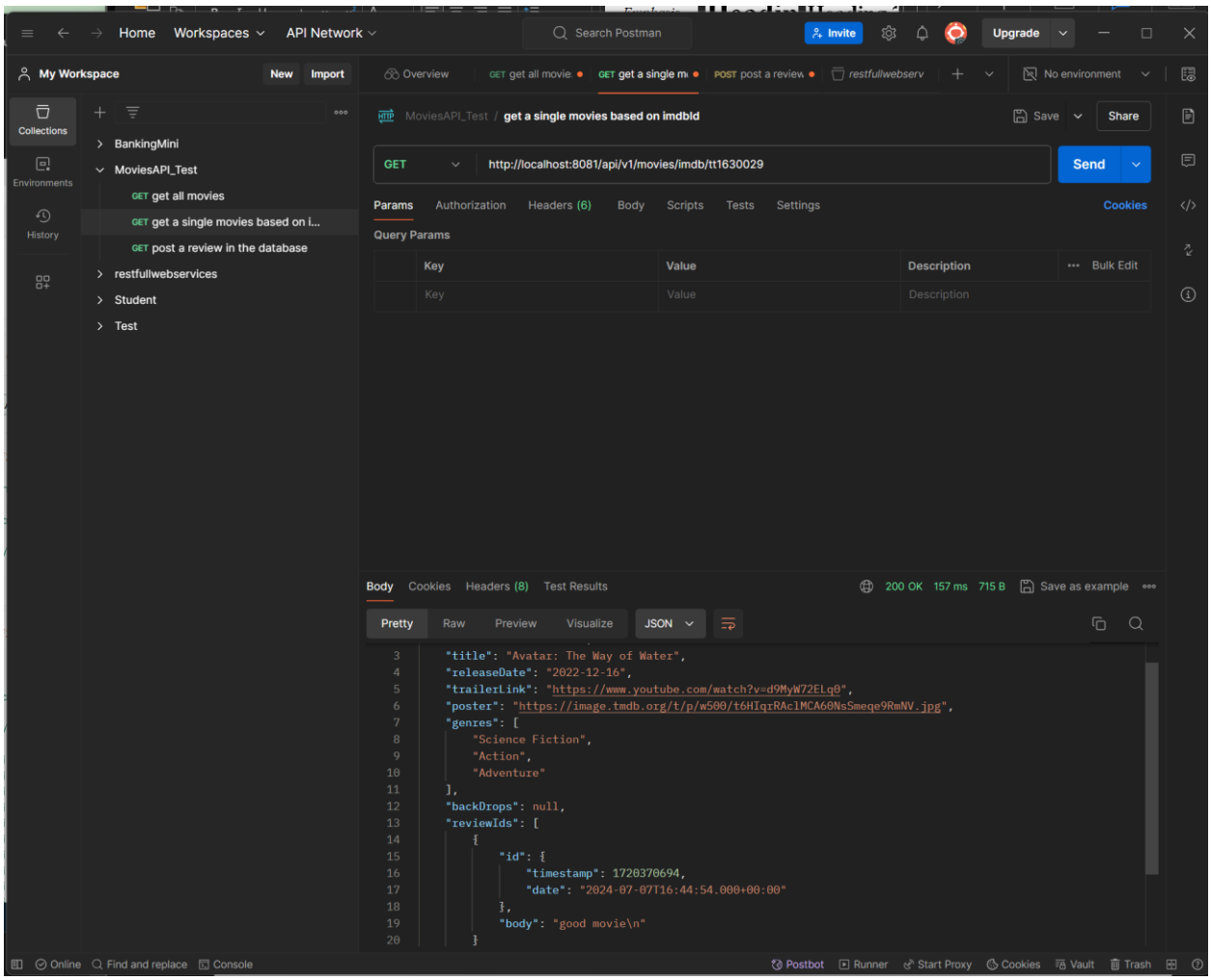
11.1 Definitions, Acronyms, Abbreviations

- **API**: Application Programming Interface
- **GUI**: Graphical User Interface
- **CLI**: Command Line Interface
- **HTTPS**: HyperText Transfer Protocol Secure
- **MongoDB**: A NoSQL database used for storing data
- **React**: A JavaScript library for building user interfaces
- **Axios**: A promise-based HTTP client for the browser and Node.js
- **Spring Boot**: A Java-based framework used to create stand-alone, production-grade Spring-based Applications

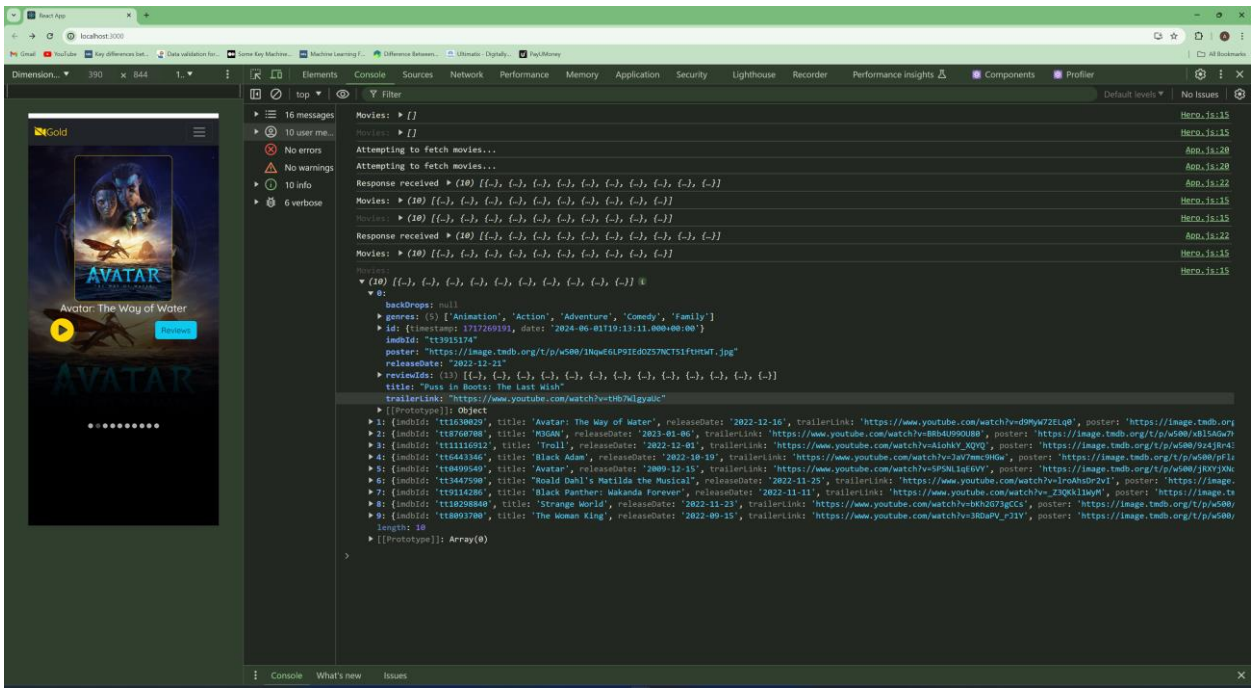
11. Screenshots

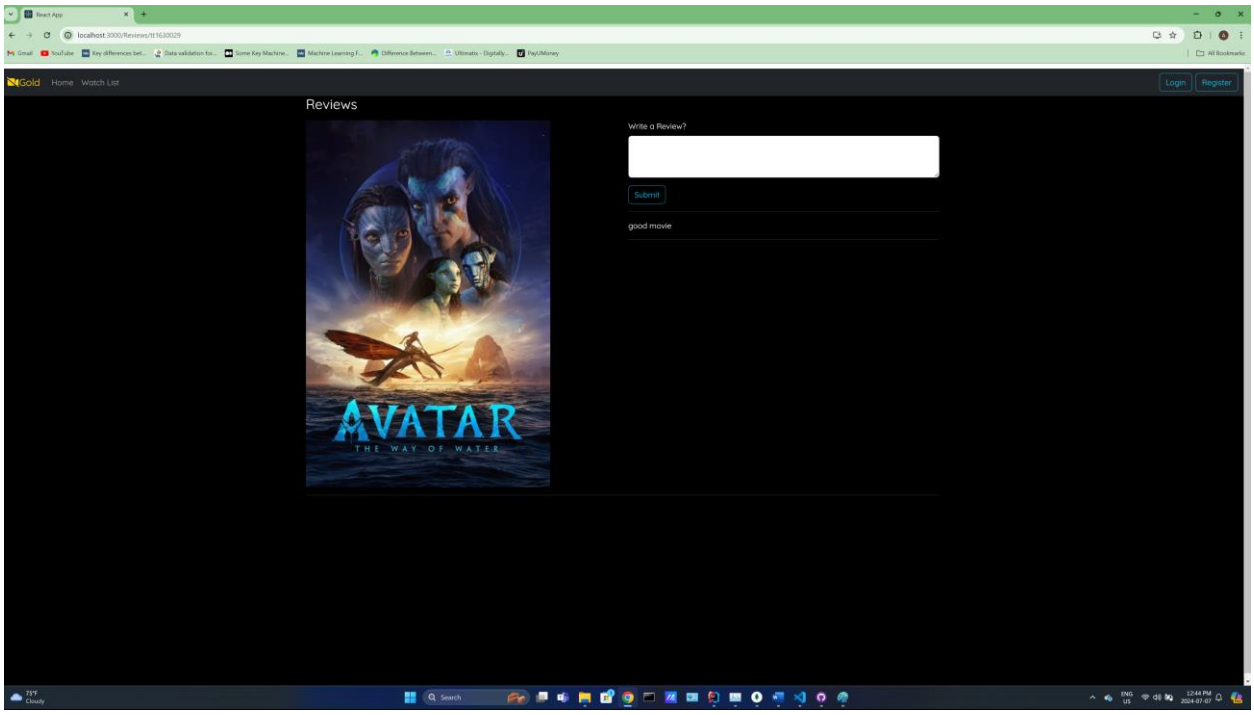

Gold
AVATAR
Avatar: The Way of Water
Reviews
Movies: ▶ []
Attempting to fetch movies...
Response received ▶ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
Movies: ▶ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
Console
What's new
Issues

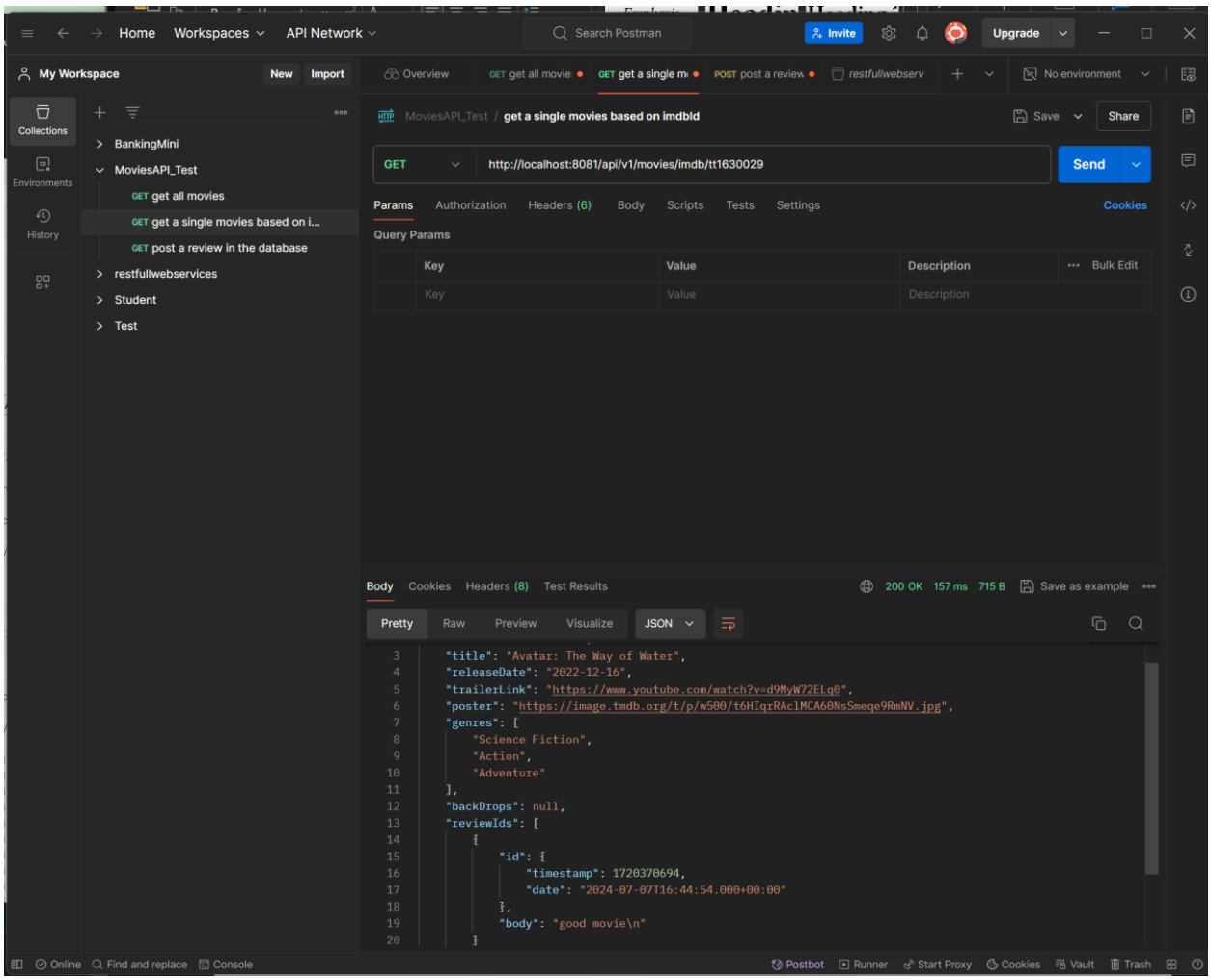

Gold
Home
Watch List
Login
Register
Reviews
Write a Review?
Submit
good movie
AVATAR
THE WAY OF WATER



Chatgpt: https://chatgpt.com/share/6bbec39c-9868-47f6-ab54-e4d7b5b03567


Gold
AVATAR
Avatar: The Way of Water
Reviews
Movies: ▶ []
Attempting to fetch movies...
Response received ▶ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
Movies: ▶ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
Console
What's new
Issues


Gold
Home
Watch List
Login
Register
Reviews
Write a Review?
Submit
good movie
AVATAR
THE WAY OF WATER



Chatgpt: https://chatgpt.com/share/6bbec39c-9868-47f6-ab54-e4d7b5b03567