Movie Database Project Report

COMP 2406

Raymond Weiming Chan
101211213
April 16, 2021

Table of Contents

Functionalities Implemented

I have implemented all functionalities specified in the requirements document.

Users are able to

- Change between 'regular' and 'contributing' user account
- View and manage people they follow.
- View watchlist
- View recommended movies
- View notifications
- Search for movies by title, actor name, and/or genre keyword. Results are paginated
- View movie data as specified
- Able to click on genres, people, add to watchlist button, and full review button
- Add basic and full reviews
- View data on any person, including frequent collaborators.
- Able to follow people and users
- View user data as specified
- Add person if user is 'contributing'
- Add movie if user is 'contributing'

Additional features:

- Users can log out of their account from the 'My Account' page
- Home page consists of clickable posters of 10 random movies.

Discussion and critique of the overall design

I think some functions in my router code can be separated into smaller functions. Right now, my code includes several nested callbacks. Calling next() and triggering another function would would improve the scalability and organization of the code. Scalability would be increased since smaller functions would be more reusable. My router code includes many functions calls on mongoose schemas. Some of those functions, especially more complex ones, could be moved to the model files as static methods that can be reused. Overall, I think my app is still quite scalable. This is due to my use of a database, mongoose, express routers, and a template engine (pug). Other than that, I feel like my code is organized and readable, and therefore can be easily modified to add features.

I put as much computation on the server side as possible so that if this app is deployed, its responsiveness would not depend on the resources of the client's computer. I tried my best to follow good RESTful design principles. My GET/POST/PUT requests and the naming of their target URLs make sense. I think my usage of different HTTP status codes is correct. Here are the most commonly used status codes in my app:

500: server error
201: created successfully
200: OK
401: unauthorized

I have mostly used asynchronous functions where possible. The use of non-blocking functions makes the app more responsive. Something I could have done was to use the asynchronous version of ‘forEach’. I have also tried to minimize the amount of data transferred between the server and the client. Using references in schemas means that each JSON sent between the server and client contain as little data as possible.

Algorithm Descriptions

Similar movie algorithm: A movie's similar movies are movies that have an exact match of genres.

Movie recommendation algorithm: A user's recommended movies are similar movies of the first movie in the user's watchlist. If the user’s watchlist is empty, then the user does not have any recommended movies.