Project Report

Authors:

- Adam Khaddaj, 101143325
- Sabrina Khaddaj, 101048402

Set-Up Instructions:

Set up the database:

1. Open the terminal and navigate to the directory containing the source code.
2. Within the project directory, create a folder named 'database' if one does not already exist.
3. In the terminal, enter the following command:

   mongod--dbpath="/NAME_OF_THE_PATH_TO_THE_DATABASE_FOLDER_YOU_CREATED"

   This runs the mongo daemon.

Initialize the database:

1. Open a second terminal, and navigate to the directory containing the source code.
2. In the terminal, enter the following command:

   node MovieDatabaseInitializer.js

   This initializes the database.

Run the server:

1. In the first terminal from the 'Set up the database' step, enter the following command:

   node ProjectServer.js

   This runs the server. Open a web browser, and enter the URL 'localhost:3000'

Design/Implementation Quality

RESTful Design Principles:

Using the feedback from our project check in, our website's URL naming scheme is now more concise composed entirely of requests that stem from "/users" "/movies" and "/people". This way, our URL routes are handled much more elegantly for our requests. Each object's

MongoDB ID is used in our URL's to organize and streamline the process of fetching a specific object (instead of using a "/search" request), adding an object to a users watchlist / followed array by using the ID queried in the URL, etc. We also made sure to use the proper HTTP methods (get, put, post, delete) where necessary, using similar URL's for as many different HTTP methods as possible.

Our website also preformed elegant error handling for when the user attempts to preform certain actions that would disrupt the database. The user is unable to make identical usernames, or add duplicates to their watched list and followed list. We also used appropriate HTTP status codes when needed, except for using error 302 for redirecting. We also should have used status code 409 for when a user attempts to make an identical username.

However, there were many places in which we could have done better. We often got confused about whether to use POST or PUT. Furthermore, our URL's for adding and removing people/movies from a users account could have been put under the same URL with different HTTP method types.

Visual Design:

Our attempts to have a cohesive visual design included the reusing of objects such as buttons, search bars and scroll-lists on most pages of our website, all of which were included in a style sheet that each page included. The color scheme was inspired by the 'dark theme' offered by the Atom text editor. While the style sheet helped keep the rest of our code from having too much duplicate style specifications, we still had in-line style specifications that may have been repetitive. The opacity of buttons and links change on hover, and we made our header's search bar have a width of 40% of the given space, so that it responds nicely to different window sizes. An improvement we could make is having the rest of our website be more responsive to different window sizes.

Extensibility:

We left the seeds of some ideas that can later be expanded upon. For example, the trailers included on movie pages are currently all a static movie trailer. Our homepage has a movie of the week, and a 'featured movies' list, which are currently filled with random movies that change each time you refresh your page. Implementing an algorithm to see which movies are currently getting highest reviews could decide which movies are featured. Pagination also allows for massive amounts of data should the database ever expand.

Algorithms

Our similar movies are generated by taking one to two genres from the movie whose page you are on, and querying the database for movies that also include, at least, those genres. We take the results and choose five of those movies at random. This could be improved by comparing genres, actors, etc.

Final Thoughts

Overall, it is clear that our server had much left unfinished. We struggled to implement MongoDB's asynchronous nature, and retrieving it's objects through ID references. We forfeited searching by people, as well as setting up reviews, as we struggled to retrieve certain objects in the MongoDB that were more complex. Despite this, the areas we did complete were done well with proper web development design principles.