Name: Ajijolaoluwa Ajayi

Project: Movie Database (No Partner)

OpenStack Information

Instance Name: jolaajayi

Instance Username: student

Instance Password: somethingsecure

IP Address:134.117.130.50

Server Instructions

Navigate to Projects in the Desktop folder and run server.js. It should take about fifteen seconds to start up. All resources in this database are stored in objects (users, movies, reviews and people). Therefore, whenever the server is restarted, the data is reset to its default state. That is, all resources added through POST requests are not saved. An initialization script in businessLogic.js extracts relevant information from movie-data-short.json and saves it to a “movies” variable. This script also creates 5 users (information about the users can be seen in businessLogic.js before the first function definition). For a sample user, you can enter the username "Jola" with the password "tears" or "Dave" with the password "joy". If you are using the sample users, the first letter of their username has to be capital. You can also look at information about the 3 other users in businessLogic.js as stated earlier or create a user of your own through the user interface. This script also makes each user write a basic review with a random score from 1-10 for each movie. The way recommended movies were implemented, if a user gives a movie a review score greater than or equal to 8, similar movies to that movie are recommended to the user. This however could cause a discrepancy between the reviews and the recommended movies. To properly test the recommended movies functionality, it is advisable to create new users and have them write reviews through the user interface.

The database currently consists of 316 movies obtained from movie-data-short.json. The home page features a list of movie images and titles. Each title links to the corresponding movie page. The movies page features a list of movie titles with smaller movie images. Each title links to the corresponding movie page. For every page, all text in blue are links. The filter page allows users to search for movies with all required query parameters. There is also a search bar in the navigation bar of each page that allows users to search for people, users and one of the movie query parameters at a time. The sign-in link in the navigation bar appears only when a user is not logged in. Otherwise, a sign-out link is displayed. Once a user signs in (or registers), they gain access to a profile link in the navigation bar that takes them to their user profile. Once signed in, users can add reviews to movies by navigating to the movie page and typing in the "Score" field or the "Summary" and "Review Text" fields. If a user types in only the score field, it is considered a basic review. If the user does not give a summary but gives a

review text and vice versa, the server sends a 400 response and alerts the user that it was unable to add the review.

To switch between a contributing and a normal user, navigate to the logged-in user's profile page and click on the change account type button. The panel that contains the profile picture tells you what type of account you currently have. Once you switch to a contributing user, you gain access to a contributor options link. The page can be accessed by normal users as well, but only if they manually enter the URL. Otherwise, the link does not show up for them. If a normal user tries to perform any actions on the page (adding a movie, adding a director etc.). The server sends a 401 response and the user is alerted that they are not authorized to carry out the action they just performed. The contributor-options page allows contributing users to add movies and people to the database. It also allows the addition of directors, actors and writers to movies. To select which action you would like to perform, simply change the option in the dropdown menu. When you elect to add new movies, an autofill button appears. This button fills the textboxes with randomly generated information. However, only 5 movies can be generated this way without editing the contents of the textboxes. This is because the randomized movie titles are in the format "Chelsea wins the {name of trophy}" and there are only 5 random trophies to pick from. The code for this can be found in clientjs/navbar.js. When viewing these auto-generated movies, a picture of a Chelsea FC team winning the trophy indicated is displayed as the movie picture. The names used are random first names and last names from 14 current Chelsea FC players. To test the recommended movies and most frequent collaborators functionality, it is advisable to manually add people to movies. The person page shows their top five most frequent collaborators. The user profile page shows at most ten recommended movies and each movie page shows at most ten similar movies. People in the database who have collaborated with one person more than once include Selena Gomez, Joel Coen, Ethan Coen and Zac Efron to name a few. In the initialization script, I have commented out a list of a few other people currently in the database who also satisfy this criterion. When contributing users view movies, a plus button appears next to the list of directors, actors and writers. Once this button is clicked the user is prompted to type in a name that would be added to the system. This is an alternative way of adding directors, actors and writers to movies.

Functionality Implemented

In addition to all required functionality, I have added a search bar and an autofill button that populates textboxes with randomized information when a contributing user is adding new movies. I made these additions to simplify the testing process and enhance the overall user experience. The search bar provides the user with a way to look up people, users and movies on the website without having to manually type the query URL. The autofill button eases the process of adding new movies to the database.

Good Design Decisions

- Similar movies are determined by comparing Jaccard's index of two movies' genre lists. Movies are considered similar if they have the same rating (PG-13, PG etc.) and the Jaccard index of their genre lists is greater than 50%. Similar movies are updated every time a movie is added.
- Recommended movies are calculated based on a user's reviews. For movies in which the user gave a review greater than or equal to 8, similar movies to that movie are recommended to them.
- All user objects are stored as values in an object with usernames as the keys to allow for O(1) lookup time, as opposed to storing them in a list which would require O(n) time to check if an element is in the list. This is also the case with movies and people. Movie titles and people's names are the keys.
- Added a "followers" attribute to the person object to facilitate sending notifications to users. This reduces this runtime of updating user notifications as it makes the process independent of the number of users in the database.
- Added a personnel attribute to movie objects that stores every person involved in the movie.
  This ensured that in cases such as when a person is a director and a writer for the same movie, they are not counted twice when considering most frequent collaborators.
- Error checking and type conversion to allow for more robust code.
- Followers, people following, users following, reviews and movie variables (e.g. similar movies, recommended movies) have their IDs stored in the object they are interlinked with as opposed to storing the objects themselves. This makes the code easier to read and debug.
- Notifications of each user are stored using a stack. Once a notification is displayed to a user, it is popped off the stack. Since old notifications are popped off, less information is transferred between the server and the client. The result is a more scalable and less latent web application.
- Error messages are displayed to alert users when they supply bad input.
- The divide and conquer approach through the use of routers allows for more modular code.
- The use of template engines to create dynamic HTML pages and avoid duplication of code thus making the system more modular. Particularly, the use of "navbar.pug", a partial pug template for the navigation bar that is shared across all pages in the website. This requires the use of only one CSS and JavaScript file for the partial template, as opposed to having onclick functions in the JavaScript file for each page. This way, if the functionality of the onclick handler ever changes, it is changed in the navbar.js file and reflected across all pages. The same goes for the styling of navbar.pug.
- Escaping HTML in pug templates to prevent users from typing JavaScript as input and stealing important information.

Improvements

The most significant improvement that can be made to the project is adding a database. Storing resources (movies, users, review) is less than ideal. This project is particularly bad in

that aspect as it does not even store information in files so it is very volatile. Incorporating a database not only allow storage of data after the server has shut down, but it also allows asynchronous access of data as well as many other operations of that nature. Asynchronous operations allow for more scalable web applications as the server can continue to handle other requests while the asynchronous request has not finished. Since my application reads movie data synchronously from the movie-data-short.json file, there is a fifteen-second delay in the server start up time. This is only with 316 movies. If it were a fully functional, industry-standard movie database such as IDMB with many movies and users, the latency of the application would be far too great. Moreover, databases and ODMs ensure a more defined structure to information (schemas in Mongoose), type checking and various other things that I possibly did not account for.

Another issue with the project is the lack of pagination. Too much information is being exchanged between the server and the client at a time resulting in a less scalable, higher latency web application. If a user searched for a particular movie and there were 4000 search results, it would not be feasible to send all of that information at once as it would drastically slow down the website resulting in a poor user experience.

Modules Used

No external modules were used (other than those recommended for the course e.g., pug, express etc.).

Project Reflection

Although it is quite far from it, the way users can interact with each other and manipulate resources on the website gives the impression of a social media application. I believe the movie recommendation system is the best feature of the website despite the algorithm being very simple. I like the fact that recommended movies can be unique to each user depending on what movies they have reviewed.