101109324

Tharindu Hatharasinghage

Project Report: Movie Central

Instructions to run the server

As my project, I picked the movie database project, which I named Movie Central. Here is the information you will need to access my project on OpenStack: my OpenStack instance name is '[redacted]', my OpenStack public(floating) IP is [redacted], the password to my OpenStack instance is [redacted].

My project files are saved at /home/student/Desktop/Movie_Central, in order to run my server.js file you must be inside the 'Movie_Central' directory. Once in Movie_Central directory run the command 'node server.js' to start my server, from this point you will have access to my website given that you have successfully created the ssh tunnel from your machine to my OpenStack instance.

Successfully implemented functionality

Here is a list of functionalities I was able to implement successfully:

User Accounts-

- Users have the option to change between contributing user to a regular user.
- If a user changes user type from contributing to regular, any of the changes the user might have made as a contributing user will stay intact.
- Users can follow/unfollow any person that exists within the database and once followed, the person will be under 'following people' in the user's home page. Users will have the ability to click on the people the follow and be directed into the person's page.
- Users can follow/unfollow any user that exists within the database and once followed the users followed will be displayed under 'following users' in the user's home page.
- Users can search for a movie by title and genre, to search for a movie user must navigate to the bottom of the page where there is a navigation bar. From the nav bar, users can click on 'Search Movie' to be directed to the search movie page.

Viewing a movie-

- When viewing a movie, the view movie page displays the title of the movie, released date, average rating, plot, and the genres of the movie.
- The genre keywords are hyperlinks that will redirect the user to a page with search results for the given genre keyword.
- The view movie page also displays the actors/directors/writers involved in the given movie and these are also hyperlinks, when clicked the user will be redirected to the person they clicked on.
- If there are any existing reviews on the given movie, these will also be displayed on this page with a hyperlink to the user who wrote the review.

- Below the reviews section is a list of similar movies. These movies are generated based on the genre of the displayed movie.

Contributing users will have access to edit the actors, writers and directors of the movie as well as having the ability to add reviews to the movie.

Viewing a person-

- The display page for people will display the movies the person was involved in along with a list of frequent collaborators of the person. Both will also involve hyperlinks that will directly lead the user to the clicked page.
- User also has the ability to follow/unfollow the person.

Viewing a user-

- When viewing a user, the user will have the option to follow the user they are viewing.
- The view user page displays all the other users the given user follows as well as all the people the given user may follow.

Contributing users-

- Contributing users have the option to add a new movie to the data base and/or add a new person to the database.
- Contributing users also have the ability to edit a given movie.

RESTful API-

- GET/api/movies – this route will give all movies that exists within the movie database.
- GET/api/movies also allows query parameters, the allowed parameters are: title, genre, year, and minrating.
- GET/api/movies/:title - searches and returns a movie with the given name.
- POST/api/movies – this allows for new movies to be added; however the user must specify a Title, Released, Runtime, Genre, aveRating, Plot, Actors, Writer, Director, Poster, reviews. Poster and reviews can be empty, but user must make sure to leave '' (empty quotation marks).
- GET/api/people – returns all existing people objects, this also takes in one query parameter: name.
- GET/api/people/:person – returns the specified person's json object.
- GET/api/users – returns the json objects of all existing users, this also takes in one query parameter name.
- Get/api/users/:username – returns the json obj of the specified user.

Functionalities I failed to implement

Due to the lack of time and obligations I had for my other classes, I was unable to implement the following functionalities.

- Recommended movies for a given user.

    - The plan I had for implementing this feature was to add the movie reviews the user makes into the user's object. Then based on the genres of the movies the user reviews, I would generate recommended movies, similar to how I have implemented the similar movies feature for when viewing movies. This may not have been the best way to generate recommended movie, but chances are it will still be better than Netflix's recommended movies.
- Add a basic review by specifying a score out of 10.
    - This is also one of the simple functionalities I had no time to implement, for this all I was planning to do is take the user's input and add that to the existing rating and divide that by two and set that as the new value for the rating.
- the user should receive a notification any time a new movie is added to the database that involves a person that the user follows, or any time this person is added to an existing movie.
    - For this I was planning on using socket.io to send real-time notifications.
- See a list of all of the reviews this user has made and be able to read each full review.
    - Plan was to simply add the review to the user's json object and whenever the user's profile is loaded to load the reviews as well, using the same logic I use to load all other information about the user.
- Choose to follow this user. If a user X follows a user Y, user X should receive a notification any time user Y creates a new review.
    - Once again, the plan was to use socket.io to send real-time notifications.

Design decisions

I believe the overall my user interface (the website) looks well designed and easy to use. I have separated the search bars such that to search for a movie user must navigate to the "search movie" section from the navigation bar at the bottom of the page, to search for a person user can use the "search person" option, to search user, use "search user" option. This makes it easier to get proper search results and leads to better scalability. If I were to make all the searches to be searched on the same search bar, the search results would include any movie/user/person that matches the search query. In a large system this could be very annoying as a user wants to search for a person named tom but when they search for it the user "tom" is buried under all the movies/people that include the key word "tom".

I also wrote most of the functions used by the server in a separate file named "business_logic.js", this made it less cluttered in my server.js file and made it easier to read my code as well as allowing me to reuse code. (once a function is defined the function can we used in different parts of my server.js file instead of rewriting that in each route)

Project Improvements

I believe that adding a database to my project will be very beneficial as it allows for better robustness and scalability. It would make my system more efficient and better at handling large amounts of requests. I also believe that using a framework such as React would allow my project to be done in a more efficient and robust way. I could also add a web scraper to add YouTube trailers for the movies.

Node Modules

As for node modules, I have not used many that I was not required to use to implement the base requirements. The modules I have used are pug, body-parser, express, express-sessions. I used express to set up my server as using express makes it a lot less tedious to serve my server than just the bare bones of NodeJS. I used body-parser for parsing JSON data along with pug so I can use/compile and serve pug pages. Express-sessions was used for adding in sessions to authenticate users.

Best Feature

If I had to pick a best feature of my website, it would be the search results page for movies. I personally think the way the posters line up when a search is done looks very cool. This is also one of my favorite parts because of the struggle I had to implement this. Prior to getting this to work, I was just using HTML and CSS pages and using JavaScript and jQuery for DOM manipulation. This made me get very tired of this project very quickly, trying to find a better way to do this I came cross pug, and pug saved me from a lot of headaches, for that I am grateful.

I also am very satisfied with all that I have learned from this course, regardless of all the stress and sleepless night this course has brought me, I have learned so much. Prior to talking this class, I knew absolutely nothing about web development, I remember hearing about this project and loosing my mind because I was so overwhelmed and had no idea what I was doing. Now I am at the end of the course and I have built a functional website that allows users to do multitude of different things. Personally, I never though an intro to web applications class would cover this much material but I'm glad it did. I also realized that I'm not the biggest fan of front-end development however I also discovered my newfound love in this class, Back-end development!