Project Report

# Movie Database Project

By: Avetik Kocharyan

Student number: 100793437

University of Carleton

Department of Computer Science

## Instructions on Operation:

**For Folder Content, VM Information, Instructions on how to Run the Server** please check the readme file for a summary of everything that is in the folder.

## Project Discussion:

**Purpose of the Project:**

The purpose of this project is to create a website that would in essence have the same (or similar) functionality as the popular IMDb website. The users of the website would be able to login or create an account (if they have not done so) to browse the various movies and artists that the website contains. The users will be able to search and leave reviews on their favourite movies. As well, follow artists and other users of the website and track their activities. Users will also have the option to upgrade from a regular to a contributing user. Contributing users have extra privileges in this website, such as being able to add new movies and artists to the website. They can also edit existing movies by adding more artists (in the form of writers, directors or actors) to that particular movie. With the addition of actual movie files this website idea promises to be an improved version of Netflix!

**Overall Design:**

The general client-server design pattern was implemented in order to achieve the main goals of this project (HTTP protocol). Specifically, requests from the client side are handled by the server which responds according to the server code (written accordingly to achieve the objectives of the project). The client-side code was separated into two general JavaScript parts, the business logic (manipulate the data of the site) and movie-server (handle request/response from the clients).

The major webpages of this website include:

1. A *localhost:3000 sign up/sign in* page (the entry point to the website).

   A new user will not be able to sign in and will have to sign up. Three users (with usernames matching their passwords usr1, usr2 and usr3) are pre-signed up to the website. Thus, the website can be accessed by entering their credentials and clicking sign in. Alternatively, a new user can be created from the same place by entering a username and a password (no restrictions on either) and clicking sign up. Once signed up, the user can sign in by entering the same credentials and clicking sign in.

2. A *userproflie (my user) page* (once users sign in this becomes their home page).

   This is the main home page of a user. On this page the user will see all the other users and people/artists that they are currently following. The current user also has a list of all it's reviews displayed. As well, see all the recommended movies and all the reviews they made. On this page the user can also perform a number of things. One action includes changing their account type (from contributing to regular and vice versa). Another available action for the signed in users is searching movies (by genre, actor or movie name). The user can also navigate to a page displaying the list of all the movies that the website has by clicking the 'Browse Movies' button. In a similar way, the user can click the 'Browse People' button to navigate to a page containing the list of all the available artists.

3. A *films* page:

   Contains a list of all the movies in the website.

4. A *film/movieid* page:

   This page contains all the relevant information the selected movie has. These include: the title, plot, release date, score (out of 10), reviews left by various users and a list of similar movies (displayed here if they have the same genre). On the movie page there also is a button called 'Leave a Review', redirecting a user to a page where they can write what they think about the movie and give it a score out of 10. Next to leaving a review there is a button 'Edit Movie', which redirects the user to a page where they can add an artist to the website by specifying their name and providing a short bio.

5. A *persons* page:

   Contains a list of all the actors/writers directors in the website.

6. A *persons/artistid* page.

   Each artist (writer, director or actor) has this page. On this page, the user can view a brief bio of the artist and see a list of this person's top collaborators and movies. The user can choose to follow or unfollow this person by clicking the appropriate buttons.

7. A *userprofile* page.

   This page is similar to the *myuserprofile* of the website but is designed to view the user profile of another user that the current user is following. By navigating to this page the current user may choose to unfollow the user by clicking the 'Unfollow' button. The current user can also view all the reviews that this user has made, all the other users and people they follow and their recommended movies.

**Website Feature Implementation and Associated Algorithms:**

<u>User actions</u>*:*

-Searching people/movies:

Each signed in user is able to search people/artists and movies. The searching algorithm is smart in that it supports searching by movie title, person name and genre. For example, searching by movie title will retrieve the movie, even if the title is not fully spelled. If a genre keyword is typed (one word at a time), it will retrieve and display all the movies that associated with that word. People can be searched by their name (again the name can be spelled partially, and the search is not case sensitive). The search results are displayed on a page (called *searchpage*) that displays the search keyword and the people and movies associated with it. Originally, it was planned to have, both, the associated people and movies displayed for each keyword, however, due to time constraints it currently does one or the other (e.g., by searching for ‘comedy’ it does not display all the movies and people associated with this term but rather only the movies and conversely typing ‘Seth Rogan’ does not display all the movies he has been in but rather just finds him). More on this will be discussed in the Future Improvements section.

-Adding a review:

Each user is able to leave a review where they can choose a score from a dropdown menu (0-10), leave a short comment and lengthy one. Though originally it was planned to have the user pick and choose what kind of review to leave (or combination of reviews), currently, a user cannot do just one of these but has to fill out all three fields for a review to register. The scoring algorithm updates the average score of the field

-Following/Unfollowing another user:

Following a user displays their username on the current users home page and allows to navigate to their page. Unfollowing a user, removes this user from the ‘Other users you follow’ filed. In either case the user does not get notification. This feature was not implemented due to time constrain.

-Adding a movie:

Only contributing users can add new movies to the website (regular users can attempt but will not succeed) by specifying the title, plot, release date, an actor, a director and a writer. One limitation of this action is that the user must add only one name per artist field (i.e. only one actor, one writer and one director) with the option of editing the movie later by adding more people. If a person from a newly entered movie is not in the website, a new person object is created and added to the person.json file. This process creates a movie object and adds it to the array that also holds objects from the movie.json data file. The new objects are not saved into the file, so after the server is disconnected the website data is reset.

-Adding a person:

Only contributing users can add new people to the website (regular users can attempt but will not succeed) by specifying the name and a short bio of the person. This process creates a person object and adds it to the array that also holds objects from the person.json data file. As with adding movies, the new objects are not saved into the file, so after the server is disconnected the website data is reset.

-Editing a movie:

Editing a movie can only be performed by a contributing user. A regular user can still access the movieEdit page, however any of the changes this user makes will not result in an edited movie. The contributing user may only add an existing person to the movie (as stated in the requirements) as part of an editing action. For this reason, a drop-down menu design was implemented in order to ensure that only existing people can be added to the movie. This design allows to avoid writing code to check if the person exists before adding them to the movie.

-Logging out:

Ends the session of the user and redirects to the sign in/sig up page.

*Other features:*

-Recommended movies for a user:

To display recommended movies for a user, the algorithm checks the people the current signed in user follows and populates this section with the movies that those people have been in/contributed to.

-Similar movies for a movie:

On a particular movie page, the user chooses to view, a list of similar movies can be seen. This list is generated using the genre keyword and populating the list with movies that have the same genre.

-Artist collaborators:

On each person page a list of other people this person has collaborated with is generated. This is handled by a simple algorithm that traverses the list of movies and populates this section of the page with people that the set person has worked with.

-Artist movies:

On each person page a list of movies this person has been involved in is generated. This is handled by a simple algorithm that traverses the list of movies and adds it to this section of the page if viewed person has contributed to the movie (as an actor, writer or director).

**Website data**

The data of this website is stored in Json files which are loaded into RAM by the client JavaScript and stored as arrays of Json objects. Hence, data performance is not achieved (i.e. once the server is restarted, data is reset to it's original). The initial website data contains 3 movies, 3 users, 8 reviews (this array of objects was not used in this version of the website, though was updated) and 9 people (actors, directors, writers). Please check the readme file and appropriate folders to see the format of the aforementioned objects.

**Technologies Used:**

Express was used to handle the server operations (HTTP protocol). Sessions was used to handle authentication. The pug template engine was used to build all the website pages and populate them with data. Ajax was used to send data from the client side to the server.

**Assumptions:**

Some of the assumptions are:

-Each movie has only one genre keyword associated with it.

-While adding a movie only one name per actor/director/writer can be entered (more people can be added to a movie by performing the edit movie action).

-Reviewers can only give a full review for a movie (includes a short text, a long text and a score out of 10).

-Security is of no concern at this time.

-If server is restarted, the entered data will be lost (since data stored in RAM)

**Testing:**

Web-based client testing:

In order to test the website, the tester can either sign in as one of the existing users (usr1, usr2, usr3 [the passwords are the same as the usernames]) or create a new user by signing up. The tester can then start by adding movies, people and reviews in order to populate the website. The various objects (movies, people, users) will be interlinked and interactive. Example of testing:

The tester logs in as usr1 with an incorrect password -> an appropriate message will be displayed.

Tester logs in as usr5 (non-existent) -> an appropriate message will be displayed.

Tester signs up as usr5 -> an appropriate message will be displayed. Now, the new user will be able to sign in.

As usr5, the tester will be able to search for people and movies. The tester will be able to follow people and review movies. The user can also upgrade to a contributing user to be able to add people, movies as well as edit movies. Initially (before following any people and looking up movies), usr5 will have no information about other users (no way to search for other users). Upon searching and viewing movies, usr5 will see what other users have left commented on set movies. Usr5 will be able to follow these users if they choose to.

The same kind of testing can be done by logging in as an existing user (usr1, usr2, usr3). In this case some initial data will be seen (i.e. followers, recommendations, etc) according to the originally defined data file (please look at the readme and see the data files).

REST API Instructions testing (public JSON REST API):

The features of the public json API were tested using the Postman software (downloaded). To test please perform the following operations.

1) GET/movies: On Postman select GET and enter localhost:3000/movies -> this will display all the movies in Json that are currently on the website.

GET/movies + any of the query parameters (**title, genre, year, minrating**) -> will display an array of movies that match the entered queries.

2) GET/movies/:movieID (for example http://localhost:3000/movies/3000) -> will display the movie object associated with this ID.

3) POST/movies: On Postman select POST and enter localhost:3000/movies, select body and change from none to x-www-form-urlencoded, then enter key (need to be exact): value pairs (for example **title** :

The Godfather, **date**: 1972, **genre**: crime, **stars**: Al Pacino, **writer**: Mario Puzo, **director**: Francis Coppola, **runtime**: 300 min, **plot**: epic mafia movie) -> will display all the movies (the added one included) and all the people (added ones included).

4) GET /people: On Postman select GET and enter localhost:3000/people -> this will display all the people in Json that are currently on the website.

5) GET/ people /: personID (for example http://localhost:3000/movies/1000) -> will display the person object associated with this ID.

6) GET /users: On Postman select GET and enter localhost:3000/users -> this will display all the people in Json that are currently on the website.

GET /users: + **name** query parameter (i.e., name: usr1)-> this will display the selected user.

7) GET/users /: userID (for example http://localhost:3000/movies/2000) -> will display the correct object associated with this ID.

**Shortcomings:**

The biggest limitation of this project is the lack of a database to permanently store the website data. The original plan was to use MongoDB for this purpose, however due to time constraints and bugs in the code it was dropped. Alternatively, data could have been just written to the Json files, however, the professor said that data in RAM is sufficient for the final submission

Some minor limitations include not meeting several of the project requirements such as no notifications upon following/unfollowing and not being able to leave a short review (currently, only full review is supported where the user must enter a long review a short comment and a score).

Also, it would be more proper to disallow a regular user from accessing the pages reserved for a contributing user. In this version, as stated earlier, the regular user is able to access these pages and attempt the actions reserved for a contributing user (adding a movie, adding a person, editing a movie), however, the actions do not go through.

As mentioned in the assumptions each movie can be of only one genre. For example, in this website a movie cannot be a comedy and an action. This is not the case as movies can have multiple genres. An improvement to this code will be to handle multiple genres. Though, this will not be complicated to implement, hefty amount of code needs to be written. Due to time constrains this was not implemented.

Another important shortcoming is that this website is not secure (passwords/usernames are not encrypted, HTTP is used instead of HTTPS). Data is also not secure which means it can be corrupted.

An issue worth mentioning is that the client-side code is not modular (i.e., most of the client-side code is in 2 files, please see readme file). Furthermore, some of the business logic functions were not implemented, but functions were written inside the movie-server code. Proper routing was not done due to time constrain. This, however, can be easily fixed (an attempt was done to compartmentalize the code but unfortunately it was buggy and due to time constrains was not fixed).

**Future Improvements:**

Biggest improvement will be to implement a database to be able to store changes to the websites data.

The search algorithm can be improved to display everything associated with a search keyword. This will essentially correct the problem raised in the Website Feature Implementation and Associated Algorithms section/ search.

The various small requirements that were adjusted, I am sure could have been met in full if it were not for time constrain.

Better compartmentalization of data and code files would also look much more professional.

**Concluding remarks**

Overall, it was a great experience working on this project. Though, originally I wasn't particularly interested in web programming, I found this process to be very stimulating and informative. The project essentially summarised everything that was covered during the term and provided a great overview about how websites operate. I wish I was not involved in 4 COMP courses to devote more time