Chloe Baroody 101146436
Ellie Ng 101142783

Final Project Report

Functionality

Implemented:

- Change between a ‘regular’ user account and a ‘contributing’ user account.
- Search for movies by title, actor name, and/or genre keyword, at minimum.
- The history of the person’s work should be separated into categories for directed, written, and acted in. Category should not be shown on the page if person has not directed, wrote, or acted in a movie
- Contributor can navigate to an “Add Person and Movie” page
- See the basic movie information, including: the title, release year, average review rating, runtime, and plot.
- See each of the genre keywords for the movie
- See the director, writer, and actors the movie has
- See movie reviews that have been added for the movie.
- View the people and users they follow.
- View the list of movies on the user’s ‘watch list’
- View recommended movies
-

Not implemented:

- Manage the people and users they follow.
- Remove any movies from the watched list.
- Allow the user to navigate to search results that contain movies with the same genre keyword.
- Navigate directly to each person’s page.
- See a list of similar movies to this one and allow the user to navigate to the page for any of those movies.
- Add the movie to their ‘watched list’.
- Receive and view any notifications that they have received about people or users they are following. Currently only displays a placeholder of “None”, which has been implemented in order to ensure the notifications array of the user objects exists.
- Add a basic review by specifying a score out of 10.
- Add a full review by specifying a score out of 10, a brief summary, and a full review text.
- See a history of a person’s work.
- See a list of frequent collaborators of a person.
- Choose to follow a person.

- See a list of all of the reviews a user has made and be able to read each full review.
- See a list of all of the people a user has followed and be able to navigate to each person's page.
- See a user's 'watch list' and navigate to any of the movies listed.
- Choose to follow a user + give the followed user a notification
- Contributor adding a Person or Movie object
- Rating for a movie is not shown as a number out of 10, but rather an array of rating values

Data Type: Movie
_id: object ID/unique identifier for the Movie object (a string made of integers and letters)
rating: scores out of 10 from all reviews of the movie (an array of integers, which is used by the movie PUG to display the average of integers)
genre: genres of the movie (an array of strings)
director: names of people who have directed the movie, used to access Person objects (an array of strings)
writer: names of people who have written the movie, used to access Person objects (an array of strings)
actors: names of people who have acted in the movie, used to access Person objects (an array of strings)
reviews: reviews of the movie (an array of Review objects)
title: title of the movie (string)
year: year that the movie was released (number/integer)
runtime: length of the movie in minutes (number/integer)
plot: summary of the movie (string)
poster: picture for the current movie… we have not implemented this yet
awards: string listing awards **NOT IMPLEMENTED

Data Type: User
_id: object ID/unique identifier for the User object (a string made of integers and letters)
roles: what the current user can do, i.e. only read, read and write, atlas admin, owner (string) ** NOT USED
recommendedMovies: list of names of recommended movies, used to access recommend movies (an array of strings)
watchedMovies: list of names of recommended movies, used to access recommend movies (an array of strings)
usersFollowing: usernames of users the current user follows, used to access User objects (an array of strings)
peopleFollowing: people the current user follows (an array of strings)
reviews: reviews of the movie made by the current user (an array of Review objects)
name: username of the user, used to access User object (string)
password: password for the user to sign in (string)
loggedIn: status of user, shows which User object is the currently logged-in one (boolean: logged in = true, else = false)
contributing: user account type, shows if User object is a contributor or not (boolean: contributor = true, else = false)

notifications: list of notifications of activity, used to access the Users related to the notification (array of Notification objects) NOT IN THE MODEL.JS

Data Type: People
_id: object ID/unique identifier for the People object (a string made of integers and letters)
directed: names of movies that the current person has directed, used to access Movie objects (an array of strings)
wrote: names of movies that the current person has written, used to access Movie objects (an array of strings)
acted: names of movies that the current person has acted in, used to access Movie objects (an array of strings)
name: first and last name of the current person, used in other objects (such as Movie and Notification) to access the corresponding People object (string)

Data Type: Review
_id: object ID/unique identifier for the Review object (a string made of integers and letters)
reviewer: object ID/unique identifier of the User object of the user that made the review (a string made of integers and letters)
movie: object ID/unique identifier of the Movie object of the movie that the reviewer is talking about (a string made of integers and letters)
rating: score out of 10 for the movie (number/integer)
summary: short summary text of the review, optional (string)
review: review text of a movie, optional (string)

Data Type: Notification
user: user that gets the notification (User object)
text: display notification to the user (string)
person: the person that is affiliated with the notification (People object)

Data Type: Follows
user: the user
people: a list representing the people the user is following (array of People objects)
users: a list representing the users the user is following (array of User objects)
followers: a list representing the users that follow the user that is being viewed (array of User objects)

Additional features

- Usernames are required to be unique when creating a new account
- In adding a movie (/contribute), the contributing user is REQUIRED to enter a title, release year, runtime, and genre
- Allows user to log out
- User must be logged in to access /profile
- Login + create user options only show up when user is not logged in

- Logout option only shows up when user is logged in
- Profile option only shows up when user is logged in
- User is not able to follow themselves (button removed)
- Movies can be searched for by year

Critique of Application's Design

- Overall, we think the design of this project could have been more efficient and well-planned through to the endpoints from the beginning. One example is that we initially rendered all our pug files by having them take in individual properties as input rather than an object. We have now changed the input to generally take in a single object, and it's properties are accessed from the pug file itself - which is much more efficient and makes the code easier to navigate.
- Scalability works well for our rendered pages
- CSS is not very user-friendly, and certainly could look better. Some improvements could be to include pictures (i.e. movie posters, user profile icon) and more colours/fonts to make the website more visually-appealing and have a coherent design theme. We could also better divide the webpage content into div classes, in order to have a more coherent design between webpages, and so each page can be better divided into columns and rows for a better and more organized user interface that separates the elements of the page more naturally.
- We attempted to have a clear distinction between the files that dealt with the front-end/client, versus the back-end/api. This resulted in two folders (client and api) that contained multiple files handling router.get, post, and push requests. It is difficult to navigate the back-end code and this causes redundancy. Because of this, it can be confusing to navigate between all the requests to check what they do. It may have been better to have either merged the client and api folders and files into one efficient and organized folder, or been more intentional with distinguishing which requests should go in the client files as opposed to those in the api.
- To be more efficient, we could have also done methods within the object schemas themselves, instead of creating route functions. This also would have eliminated the need to have certain object properties which could instead be accessed through schema methods - the biggest benefit being that through methods, the functions can far more easily fetch new data from other objects. That makes it far easier to organize data that involves multiple objects, as you only need to update the properties for one of the objects, and the other objects can access that data through their methods. Furthermore, methods reduce the need for long Object.find() functions in the route files themselves, which can be difficult to read, and due to their length and complexity, it makes it harder to catch errors.
- A lot of the routes and buttons do not connect properly, or are simply left out.
- Looking back, we should have made proper error pages or alerts… however, we left the errors to be displayed using res.status(xxx), which did not match the css of the website or display the correct error code at times.

Recommendation and Similar Movie algorithms

Note: these were only implemented in the generated database, and not for movies created through the api. Having these implemented as a schema method as opposed to a schema property would have made it easier to implement for all database elements, rather than only the manually generated ones.

- Recommendation algorithm:
    - For each movie in a user's watchedMovies array, the first element of the movie's similar movie array is added to the user's recommended list.
- Similar movie algorithm:
    - For each movie, the similar movies were added based on those who contain the first element of the movie's genre array.