Project Report - MovieSwipe

DAT076 Web applications

Arian Alikashani
Ermin Fazlic
Molly James
Valeria Nafuna

Introduction

MovieSwipe is a web application that allows users to share their movie interests with their friends in an easy and fun way. The application allows users to create an account to swipe ‘Like’ or ‘Dislike’ on movies that suit their interest. These interests are then compared to the interests of all the users' friends where the intersection of the two is presented. By using MovieSwipe you can quickly find common ground with others, whether it is for “breaking the ice” or to finally settle the age old question “what movie do we watch?”, MovieSwipe is sure to aid you through the process.

Github link

https://github.com/ErminFazlic/Movie__Swipe

Please read

We had some issues with our repo around 30% through the project where the project worked on some computers and not on others. We therefore created an entirely new repo and did a big initial commit. That is the reason for there not being any commits prior to that. This might also skew the contribution amount a bit but hopefully not too much.

If the project doesn’t run because of missing icon pack, run the command “npm install react-icons --save” in the frontend directory. This is needed since the icons were added last minute and we did not have time to figure out how to make it work without running the command on every machine.

Use Cases

A short description of the application use cases.

1. **Create an account:**

The user can create an account using email, username and password. The information that is linked to this account is the liked and disliked movies, as well as a list of friends.

2. **Log in:**

Users who successfully registered an account can log in.

3. **Log out:**

Users who are logged in can log out.

4. **Add friend:**

Users who are logged in and input a valid username can add other users to their friends list.

5. **Remove friend:**

Users can press the remove button next to a friend's name on their friends list to remove them from their friends list.

6. **Like Movie:**

Users can like a movie. The movieID will be added to their liked movies. It also presents all of your friends that have liked the same movie if there are any.

7. **Dislike movie:**

Users can dislike a movie. The movieID will be added to their disliked movies.

8. **View liked movies:**

Users can view a list of all their liked movies.

9. **Remove liked movie:**

Users can click the remove button next to a movie to remove it from their liked list.

10. **Get common liked movies :**

Users can click on a friend's name on the friends list to view the liked movies they have in common.

11. **Change password:**

Users can click on their name in the top-right corner to get to their profile page and change their password.

User Manual

A description on how to navigate and use the core functionalities of the application.

Register

You are able to navigate to the registration page by clicking on the “Register an account” button on the first landing page. Enter your username along with an email and password that you want associated with the account. To proceed click the “Register” and you will be taken back to the first landing page where you can log in to your newly created account.

Log in to your account

At the first landing page, proceed by navigating to the two input fields titled under “MovieSwipe”. Input the email and password associated with the account and click the “Login” button. If you input the correct information you will be taken to the home page, otherwise you will be asked to input the correct username/password.

Log out of your account

At the home page, click on your username on the top-right to get to your profile page. From here you can press the “Logout” button to log out.

Description of home page

At the center of the homepage you can view options for liking or disliking movies, and a short description of each title. The description involves the movie’s genre, release year and a poster.

Add friend

Navigate to the “Friends” title, there you will see an input field where you enter the username of the user to add. If the username exists in our database then the user will be added to your friends list. If the username does not exist you will be told that the user could not be found and you may retry with another username.

Remove friend

Click the “Remove” button next to a friend's name after navigating to the friends list.

View friends

All your friends' usernames will be listed at the top part of the friends page which can be reached by clicking the friends link on the navbar..

Like/dislike a movie

To the sides of the movie poster there is a thumbs up icon which represents the “like” option, the thumbs down icon represents the “dislike” option. Choose the option which best suits your interest or opinion of the displayed movie.

View your liked movies

Click on the link “Liked Movies” on the navbar to reach a page displaying a list of your liked movies.

Remove a like from a movie

Click on the link “Liked Movies” on the navbar to reach a page displaying a list of your liked movies. Press the name of the movie to remove it from your likes.

View common liked movies

Click “view matches” after navigating to the friends list where you get a drop down menu of options by clicking one of your friends username.

Change password

Click on your username on the top-right to get to your profile page. From here you can input a new password then press the “Change password” button. It will then take you to the login page where you can now login with your new password.

Design

The application is split into two directories, Server and Client. The Server directory takes care of all things for the backend while the Client takes care of all things for the frontend. The whole application is built with the MERN-stack, meaning MongoDB, Express, React and NodeJS. Aswell with typescript instead of regular javascript.

Client

In the index.tsx file the different routes are mapped to different components (see below), so that when a user visits localhost:3000/home they are viewing the homepage component.

In the api.ts are all methods for communicating with the server. These do Axios calls to the api and then pass on the data from the call to make the code in components a bit cleaner and to make these calls reusable wherever needed.

The file type.d.ts contains type and interface definitions of Users and Movies as well as a special ApiDataType for each to make the passing of data from the api calls in api.ts easier.

Includes all the components for the frontend. The components are in the components folder and are: Friends.tsx, Home.tsx, Likes.tsx, Login.tsx, Movies.tsx, Navigation.tsx, Profile.tsx, Register.tsx. These components take care of displaying the appropriate things on the page, through the use of the methods from api.ts, and navigating between them on button clicks etc.

The App.tsx file is the landing page and contains the Login component as well as a nice background.

The following libraries have been used in the Client section:

- Axios, to easily make calls to the backend API.
- React-bootstrap, to make use of pre-built parts such as buttons and forms.
- React-router-dom, to easily be able to navigate between the different pages.
- Jest, for testing.
- React-icons, for icons.

Server

In the server.ts file there is some configuration to use cors, express.json and a mapping between the routers and links such as “/api/v1/users” => users.route.

In the index.ts file we can find the configuration to connect to the MongoDB Atlas Database.

In the types folder we find interfaces for movies and users. Both of which extend Mongoose's document to make them represent documents in MongoDB. In the model folder we find Schemas for both the users and movies for MongoDB.

The service folder contains the services for both users and movies. The methods in the service files are called to perform actions and it is here that the application communicates with the database. The _tests_ folder under this folder contains the unit tests for movie service and user service.

The router folder contains the router files for both of the above mentioned services. These router files make sure that the endpoints connect to the correct method in the service files.

The following libraries have been used in the Server section:

- Express, to more easily develop the api and backend,
- Mongoose, to be able to communicate with the MongoDB Atlas database.
- Cors, for added security.
- Jest, for testing.
- Dotenv, to make use of environmental variables.

API

The base URL is localhost:3000/api/v1
For all the endpoints if an error occurs on the server, it sends status 500 with the error message.

/movie/like/:loggedInUser (Like a movie)
Parameters: Logged in user's username in the url
Body: Movie's name
HTTP PUT

If successful returns status 200 with the friends that have also liked that movie.
If there are no matches it returns status 200 with the message 'No matches'.
If a user is not logged in, it will return status 401.
If the movie can't be found returns status 400.

/movie/dislike/:loggedInUser (Dislike a movie)
Parameters: Logged in user's username in the URL
Body: Movie's name
HTTP PUT

If successful returns status 200.

If a user is not logged in, it will return status 401.
If the movie can't be found returns status 400.

/movie/:loggedInUser/matches/:friendUser (Get matches between two friends)
Parameters:Logged in user's username and friend's username in the URL
HTTP GET

If successful returns status 200 with the movie's that both users have liked.
If there are no matches it returns status 200 with the message 'No matches'.
If a user is not logged in, it will return status 401.
If the friend can't be found returns status 400.

/movie/:loggedInUser/liked (Get list of liked movies)
Parameters: Logged in user's username
HTTP GET

If successful returns status 200 with the list of liked movies.
If a user is not logged in, it will return status 401.

/movie/:loggedInUser/movie (Retrieve a movie for the user to like/dislike)
Parameters: Logged in user's username
HTTP GET

If successful returns status 200 with a movie.
If successful but there are no more movies it returns status 200 with message 'No more movies to show'.
If a user is not logged in, it will return status 401.

/movie/:loggedInUser/remove (Remove a movie from a user's liked list)
Parameters: Logged in user's username
Body: Movie's id
HTTP PUT

If successful returns status 200.
If a user is not logged in, it will return status 401.

/users/ (Create an account)
Body: Email, password and username
HTTP POST

Returns status 201 if successful with the message ‘created user’.

/users/ (Login user)
Body: Email and password
HTTP PUT

Returns status 200 if successful with the user’s username.
Returns status 401 if the user inputted the wrong password or invalid email.

/users/friends/:usernameToAdd (Add a friend)
Params: Username of the friend to add
Body: Email of the logged in account
HTTP PUT

Returns status 200 with message ‘Could not find user’ if the user they are trying to add does not exist.
Returns status 200 with message ‘You have already added that user as a friend’ if the user tries to add a friend that is already on their friends list.
Returns status 200 with message ‘Added friend’ if successful.

/users/friends (Get list of friends)
Body: Email of the logged in account
HTTP PUT

Returns status 401 if the user is not logged in.
Returns status 200 with the friends list if successful.

/users/friends/remove/:userIDToRemove (Remove a friend)
Params: UserID of the friend to remove.
Body: Email of the logged in account
HTTP PUT

Returns status 401 if the user is not logged in.
Returns status 200 with message ‘Removed friend’ if successful.

/users/changePassword (Change password)
Body: Email of logged in user and new password.
HTTP PUT

Returns status 401 if the user is not logged in.
Returns status 200 with message ‘Updated Password’ if successful.

If we had more time

If we had more time we would implement the following things:

- JWToken authentication, to make it more secure and only allow one person to be logged in on the same account at a time.
- A movie API, for example IMDB, to automatically fetch movies. As we did not have time we hard-coded around 16 movies to the database.
- Some bug fixes, for example there is a bug where you have to refresh the page after logging in to display the movie.
- More tests, right now the testing is a bit thin.
- Prettier popup windows, right now we only made use of the js alert function. But we would implement a custom popup window if we had the time.

Responsibilities

For the most part we split up into pairs when working, these pairs were as follows; Ermin & Arian, Molly & Valeria. For the backend we all contributed mostly equally. Ermin and Arian focused on frontend while Molly and Valeria focused on the testing part.