Case Study - MoviesMoviesMovies

Project Overview

Challenge: creating from scratch each piece of a fully-functional movie database into which users can log in and view a list of titles; search by title, director, or genre; save their favorite movies to their account; and edit their account.

Process: establishing the movie database was the first step, by learning to use SQL, then converting to MongoDB. Once online, the database was used by the backend API endpoints written using Express, with endpoints performing functions such as “get list of movies,” “add movie to favorites,” and “update user password.” Next was frontend work, using React to create views such as “login,” “movie-view,” and “profile-view.” Finally, live hosting online was done using Heroku, for the front and back ends.

Goal: demonstrating the ability to use all the skills I had learned so far, summarized as databasing, backend API, and frontend client interfacing. Superseding the usage of these skills was the necessity to design an interface and ensure efficient functionality, such that the app would stand out among other instances of students creating the same project.

Middle

Database: created first with SQL to introduce long-standing industry standards, learning commands for querying and creating schemas. Once complete, the database was converted to a cloud system on MongoDB, installing their CLI and uploading to their cloud. The database holds 2 collections: users and movies. Within each are the documents representing each user, and each movie.

Backend: using Express I wrote all the necessary API endpoints for the yet-to-be written frontend to use, testing each API endpoint with Postman to ensure functionality. This required a lot of thinking ahead to deduce what kinds of endpoints would be necessary, to get/create/update/delete any of the users and movie data in the different contexts they would be needed. For example “getting” a full list of movie titles and some of their details, for the “main-view” page that would show movie cards for all titles; “creating” a new favorite by adding a movie-id to a user’s database listing as a favorite from the “movie-view”; and “updating” a user’s database information from the “update-view” page, from which a user can change their database information such as password or email address.

Frontend: using React I created views for each page I wanted the user to be able to view. The user would “land” on the “main-view” page, which would default to a “log-in” page if no user was logged in, and have a link to the “registration-view” page if the user had no account. Once logged in the user would be presented with a list of all movies in the database, displayed on

“cards” which included a movie poster, the movie’s premise, the director’s name, and the genre. The user can click the movie card to see more about the movie, the director to see movies by the director, or the genre to see more movies of the genre. On the movie’s information page (and on the movie card on the “main-view”) the user can add the movie as a favorite. In the navigation bar the user has a link to their account profile, where their account information is displayed as well as their favorite movies they have added, and can remove favorites from here as well.

End

Working on this app was when I started taking detailed notes on what I was doing, going over and comprehending what I had just coded to assist the learning process. Writing things out also helped tremendously in actually describing a problem, so I could further dissect and work on each piece, and organize details to create a sort of roadmap on where to focus my efforts. I learned to keep a revolving notes file to record issues and important details, as well as pin items like useful CLI commands and locations of files I would need later.

For example this helped me discern why {Movie.Genre} would evaluate as undefined, by pasting snippets of code from different parts of the project together in my notes, to compare and see what was missing; in this specific case the API’s endpoint address I needed was .genre, and the database path I needed was Genre.Name, so I was able to determine .genre.Genre.Name was the correct code, because I had all the details I needed together in one place.

```
render() {
    const { movies, genre } = this.props
    if (!genre) return null;
    return (
        <div>
            <div className="cardTitle">
                <h1>{genre.Genre.Name}</h1>
            </div>
            <hr/>
            <div className="cardDesc">
                {genre.Genre.Description}
            </div>
            <div className="allMoviesLink">
                <Link to={'/'}>All Movies</Link>
            </div>
            <hr/>
            <Row className="moviesOfGenre">
                {movies.map((movie) => {
                    if (movie.Genre.Name === genre.Genre.Name) {
                        return (
                            <MovieCard key={movie._id} movie={movie} />
                        );
                    }
                })}
            </Row>
        </div>
    )
}
```

I also have several points noted about this project to work on in the future, for instance using IMDB's movie database API instead of my own limited database of movies.