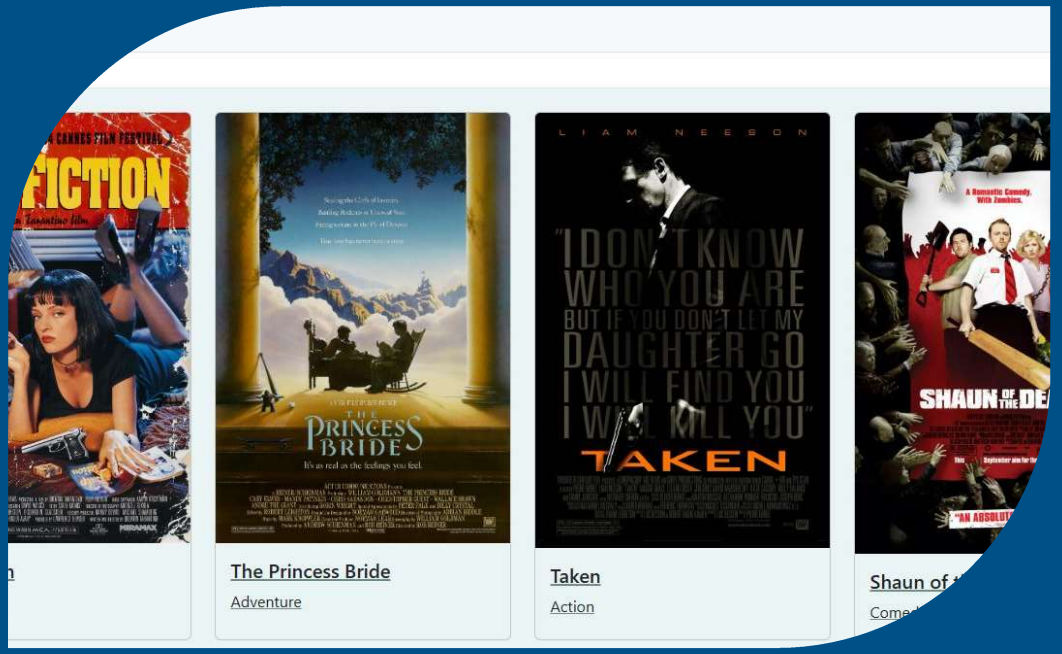
2/13/2024

MyFlix Application

A fun way to learn more about the movies you love!

MyFlix is a web application built with the MERN stack that allows users to find more information about their favorite movies, like who the director was or what genre it is. Users can create an account, log in, and add movies to their list of favorites, and they can navigate to their profile to view that list, or to make changes to their account.



Timothy Pamplin

Why Build This Project?

I created this project as part of the Full Stack Immersion course at CareerFoundry. The purpose of this app was to demonstrate my ability to build an entire web application, front and back end, from the ground up. During the development of this app I dove deep into many web development fundamentals, such as:

React
Bootstrap
Node.JS
Express
Database Management
And more!

The Process

Time:

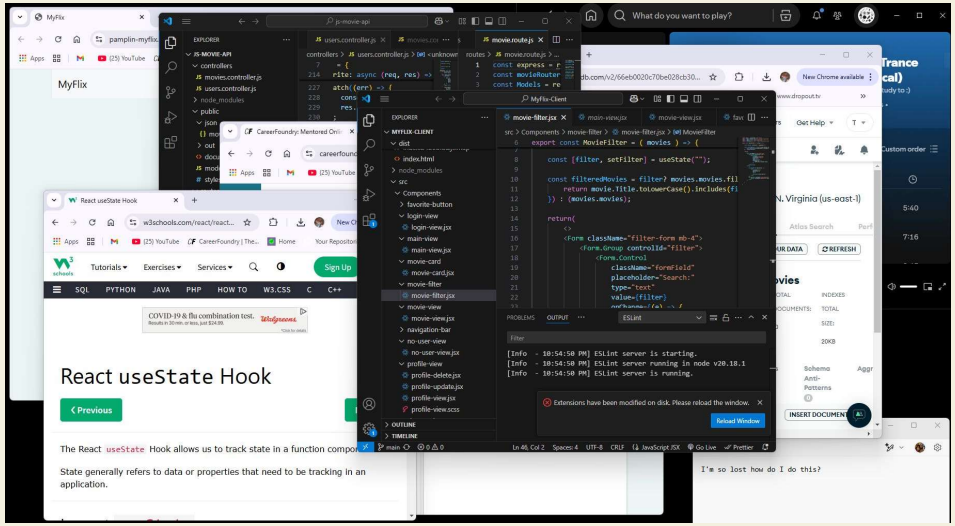
This project took me several months to complete, to my surprise, the API was the easy part. Setting up the database and Express server was fairly straightforward, and did not take that much time. Getting the front end to look and act the way I wanted was much harder and took up the bulk of the time.

Credits:

Lead Developer/Designer: Timothy Pamplin
Tutor: Matthew Henderson
Mentor: Drew Mercer

Tools:

MongoDB Atlas
Heroku
Netlify
VSCode
React-Bootstrap
Express
Node.JS
Postman
Git/Github



The Back End

Before I could start building the user interface for my app, I needed something to interface with. I needed a place to store information about movies and users, and I needed a way to retrieve that information when I need it. So I built a RESTful API to facilitate this. There were several steps involved in this process.

Step 1: The Database

Due to the nature of the data I was dealing with, I decided a non-relational database would be a good fit for this project. Therefore, I decided to use MongoDB, as it has the flexibility to deal with different movies having different properties. MongoDB was also extremely easy to set up and to host through the integrated hosting solution MongoDB Atlas.

Step 2: The Server

Now that I had a functioning database, I needed a server to be able to send requests to, that would process and send back the information I needed from the database when I needed it. I used Express and Node.JS to quickly build a server that I could test with locally. Then I spent a much longer time building out my endpoints so that the server could receive HTTP requests for specific data.

Step 3: Integration

The final step was to actually connect the server to the database. While this seems straightforward, there were a lot of little steps involved to make sure that I was following the best security practices. From hashing incoming user passwords, to requiring a JWT token to be sent along with most requests, or simply making sure that the data sent with a request was actually valid before updating the database. This step probably took the longest of the three.

Testing and Documenting

Testing

Now that I had a complete API, I had to make sure it actually worked. I used Postman to make requests to every endpoint that I had created. I needed to make sure that each endpoint would send back the data I was expecting when I sent a request. I saved examples of both the requests and responses from every endpoint as well, which I would use when writing out my documentation.

Documenting

At this point I had finished the API and confirmed that it was working as intended. The last thing I needed to finish the back end was clear documentation that I could refer back to while building the front end client. This was an important step both because I needed to learn how to do it, and also so that I could have clear examples of how to send requests to my server, and what kind of response to expect back. The image below is the literal table of contents of that documentation, as well as a brief overview of the endpoints and what to expect from them.

Request Type	Request Body	Response	Endpoint	Description
GET	None.	JSON	/movies	Returns a JSON object containing a list of all movies.
GET	None.	JSON	/movies/:Title	Returns a JSON object containing all information about a specific movie.
GET	None.	JSON	/genres/:Name	Returns a JSON object containing data about a specific genre.
GET	None.	JSON	/directors/:Name	Returns a JSON object containing information about a specific director.
POST	JSON object with New User Data	201 "User Successfully Created"	/users	Adds a new user to the list of users.
PUT	JSON object with updated data	201 "Username Changed Successfully"	/users/:id/username	Updates the username to the new username the user has picked.
POST	JSON object with new movie data	201 "Movie added to favorites"	/users/:id/favorites	Adds a movie to a user's list of favorite movies.
DELETE	None.	201 "Movie Removed from favorites"	/users/:id/favorites	Removes a movie from a user's favorite list.
DELETE	None.	201 "User deleted"	/users/:id	Removes a user from the list of users.

The Front End

Now that I had a functional API which I could use to store and retrieve the data I needed, it was time to focus on the user interface side of the project. Which ended up being a single page application built with React, which gives the user several views through which they can interact with the data on the back end.

Step 1: The Build

The first step was to figure out what systems I would use to build and deploy the application. I used React, because it is fast, and easy to make changes in, and because learning it was an overarching goal when making this project. I used parcel to compile and build the final application.

Step 2: Components

Once I had a blank canvas to start building an application on top of, I started building react components that I could then combine into the full application. I made different components for different views, forms, buttons and more. Then all I had to do was arrange them how I wanted, and make sure each component was sending/receiving all the data that it needed to function correctly.

Step 3: Styling

Now that I had a functioning web application, I spent some time making it look pretty. This included adding background colors, borders, styling buttons, and changing the layout of my components so they fell on the screen where I wanted them to. During this process, I made sure that the application would scale to many different screen sizes, and that the text would be high enough contrast that it would be easily visible.

Views

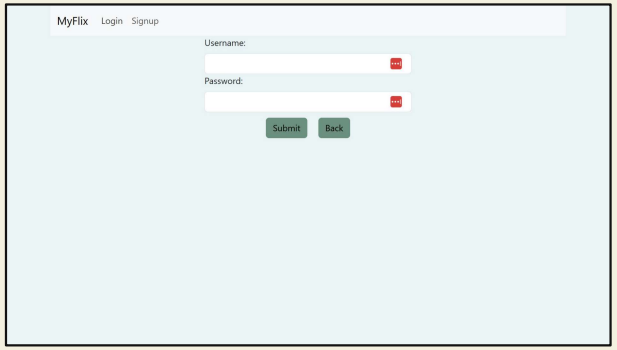
The front end application contains 6 different views that all allow the user to accomplish different tasks, or view different information.

No User View

This view contains the login and signup buttons and is what a first time or logged out user will see when they first arrive on the webpage.

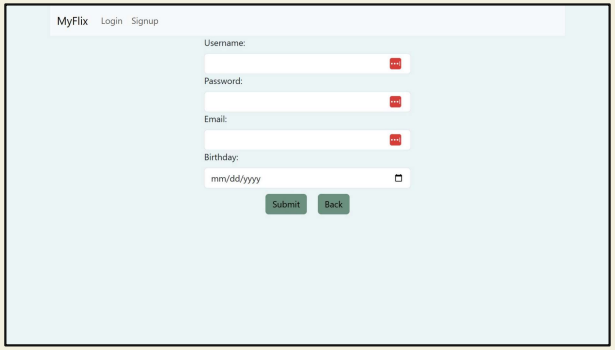
Login View

This is the view containing the login form that the user will see when they click the Login button on the no user view.



Signup View

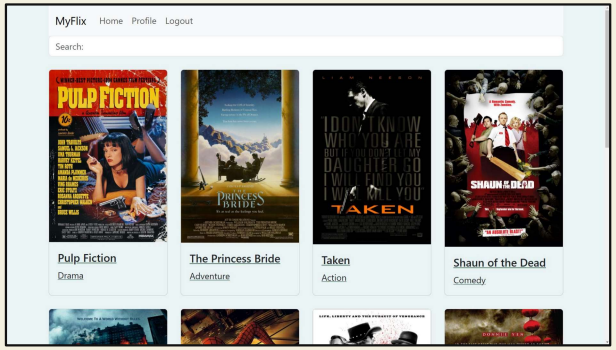
This is the view containing the sign up form that the user will see when they click the Sign Up button on the no user view.



More Views

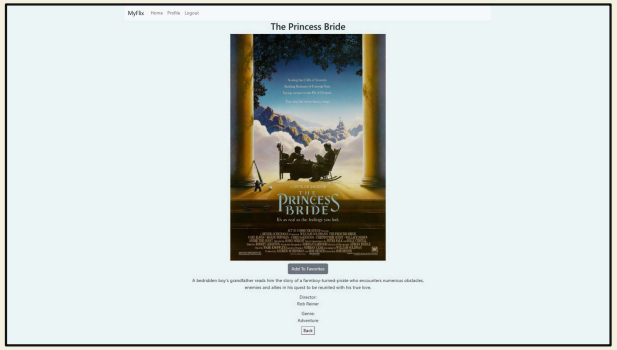
All Movies View

When the user successfully logs in, they will be shown a list of all movies available. They can use the search bar to search for a specific movie as well.



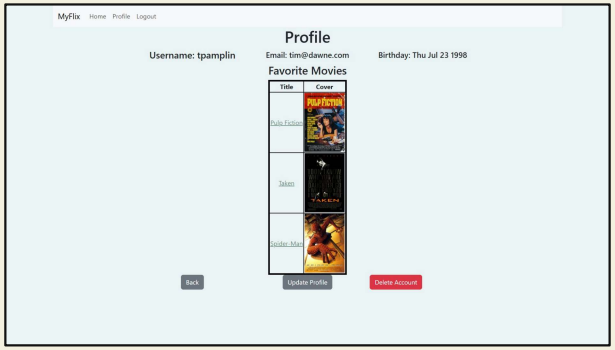
Single Movie View

When the user clicks on a movie in the all movies view, this view will open and show more details about that movie and let the user add it to favorites.



Profile View

When the user clicks the profile button in the navigation bar, they will be shown this view, which contains their account information, their favorites, and buttons to update or delete their account.



Conclusion

How did this project go?

This was easily the most difficult and time consuming project that I worked on during the Full Stack Immersion course at Career Foundry. This was probably one of the most difficult projects I've ever worked on. It was difficult for me to find adequate time to work on it, as I was working full time, and trying to make time for my family. I would sit down at my desk, and by the time I had figured out where I was and what I was working on, I had already run out of time. Ultimately, I just ended up sacrificing my sleep schedule, and staying up until two or three o'clock in the morning to work on this. I also spent ages trying to fix a bug in my front end that ended up actually being an issue with how my back end was handling the request. I had to get help from several different people before I was able to get that working. Overall, however, this was a very fulfilling project to work on, and I am proud of the final result that I produced. If you want to check it out, it is hosted on Netlify.

Click here to view the finished project.

What would I add or change?

I was on a bit of a time crunch during this project, so I didn't get to do everything that I wanted to. I always wanted to add more than ten movies to the database, or even better, I wanted to add the ability for a user to add a movie to the list; I even considered creating a second application specifically for reviewing movie submissions before they were shown on the main site. I could have spent a lifetime on styling and layout, but I had to move on to other projects in order to complete my course on time. This version of the app does not have the ability to view more information about the director or genre, even though that information is available in the database, so I would have liked to implement a pop up when you click on the director or genre that gives you that information.

Other than that, I don't think there was much I would change about this project.