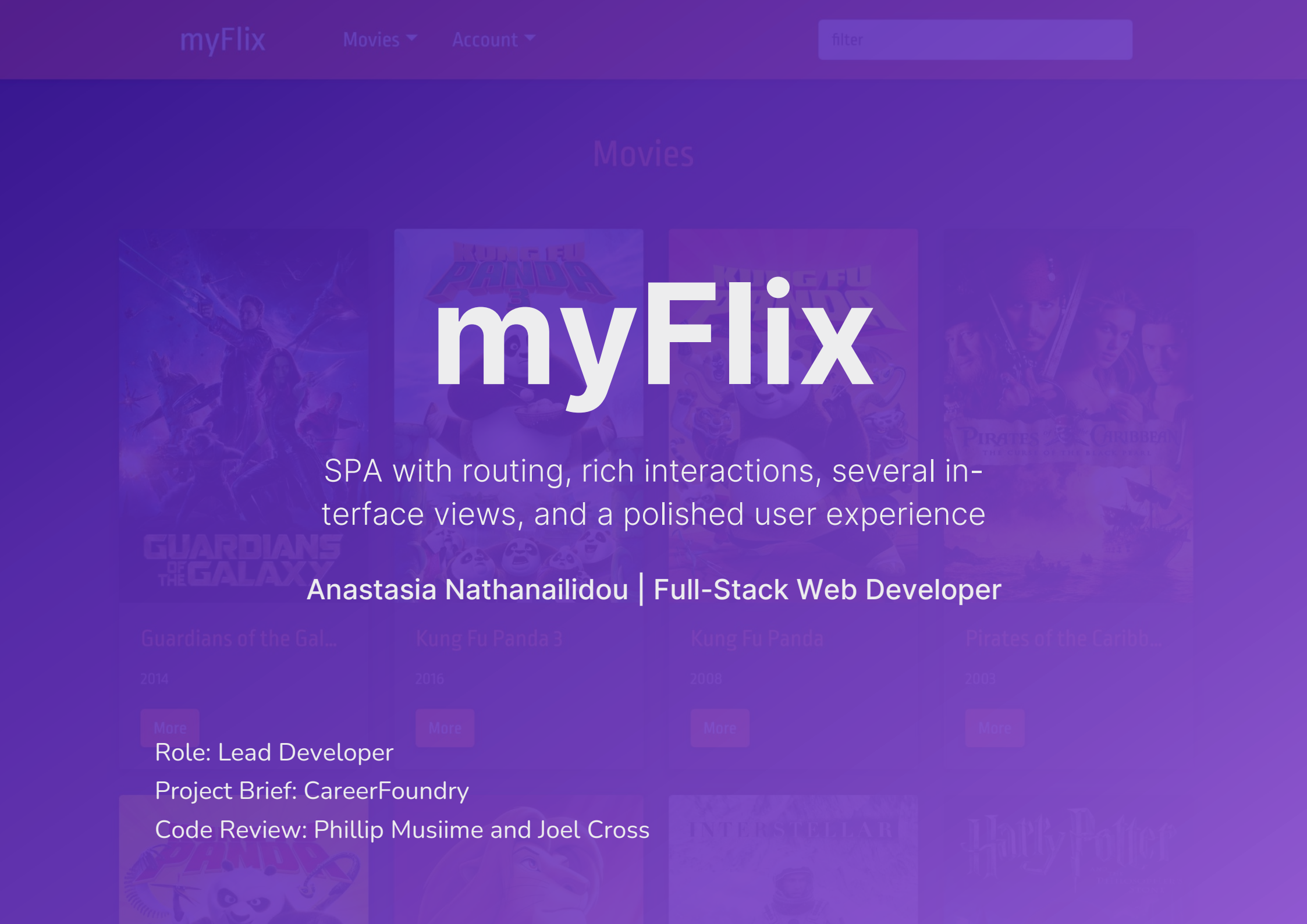


myFlix

SPA with routing, rich interactions, several interface views, and a polished user experience

Anastasia Nathanailidou | Full-Stack Web Developer

Role: Lead Developer

Project Brief: CareerFoundry

Code Review: Phillip Musiime and Joel Cross

Overview

myFlix is a full-stack web application built using the MERN stack. It provides several interface views, such as the main view that displays a list of all movies and the single movie view that shows details of a specific one.
The same kind of views exist for directors, genres, and actors. The app further includes a login view, a registration view, and a profile view which allows users to update their information and list of favorite movies.

Purpose & Context

This app was part of my full-stack web development course at CareerFoundry. Its purpose was to develop a full-stack web app from scratch to familiarize myself with both backend and frontend technologies.

Objective

The first objective was to create a REST API that interacts with a non-relational database. The next step was to build the client-side with React.

Tools & Methodologies

Backend

Node.js
Express
MongoDB
Mongoose
Postman
Heroku

I built the API with Node.js and Express using the REST architecture and deployed it on Heroku. Next, I created the database with MongoDB, uploaded it to MongoDB Atlas, and connected it to the API.

Users are authenticated and authorized through basic HTTP authentication and JWT authentication.

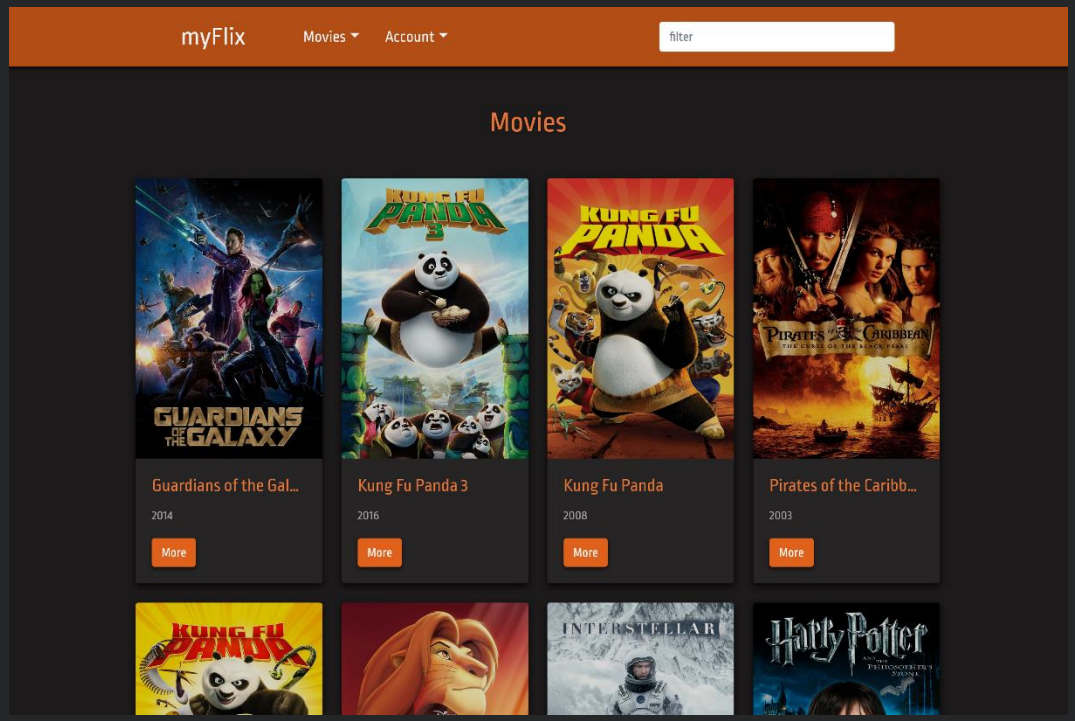
The API uses CORS, password hashing with Bcrypt, and server-side input validation with Express-Validator for data security and validation logic.

Afterward, I tested the API in Postman to make sure everything was working correctly and generated its documentation with JSDoc.

Frontend

I developed the frontend using React and Redux as a single-page application (SPA) that uses state routing to navigate between views and URLs.
Further, the app features a responsive design that I created using React-Bootstrap and SCSS.

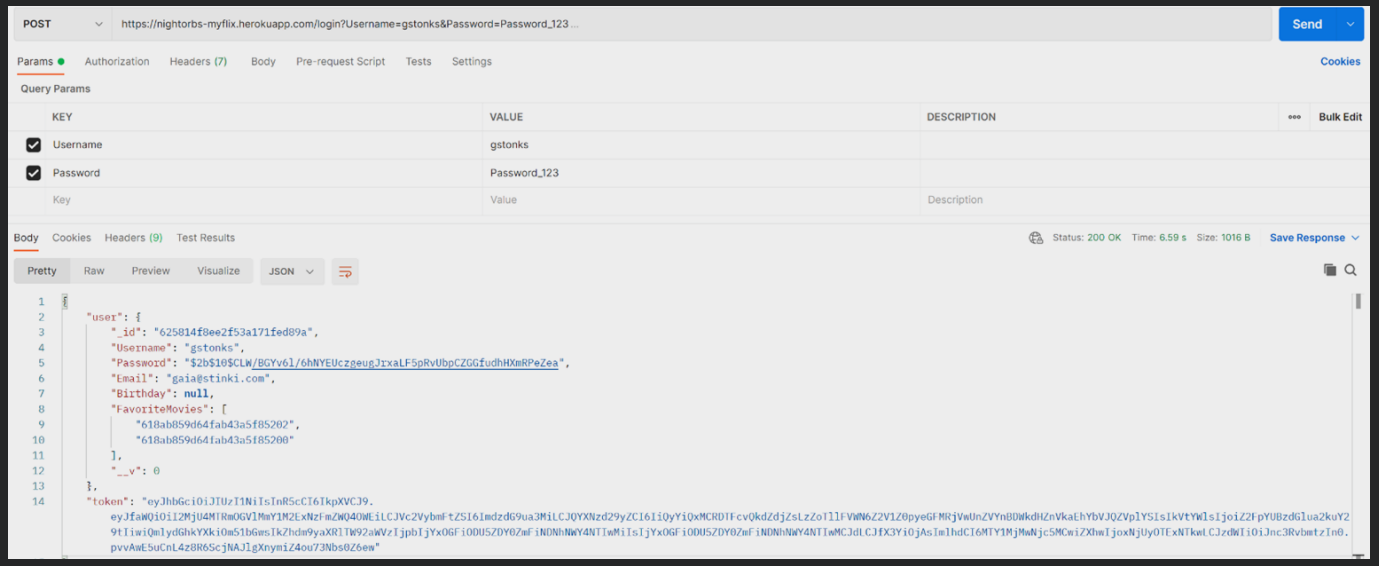
- React
- React Router
- React-Bootstrap
- Redux
- Parcel
- Axios



Approach

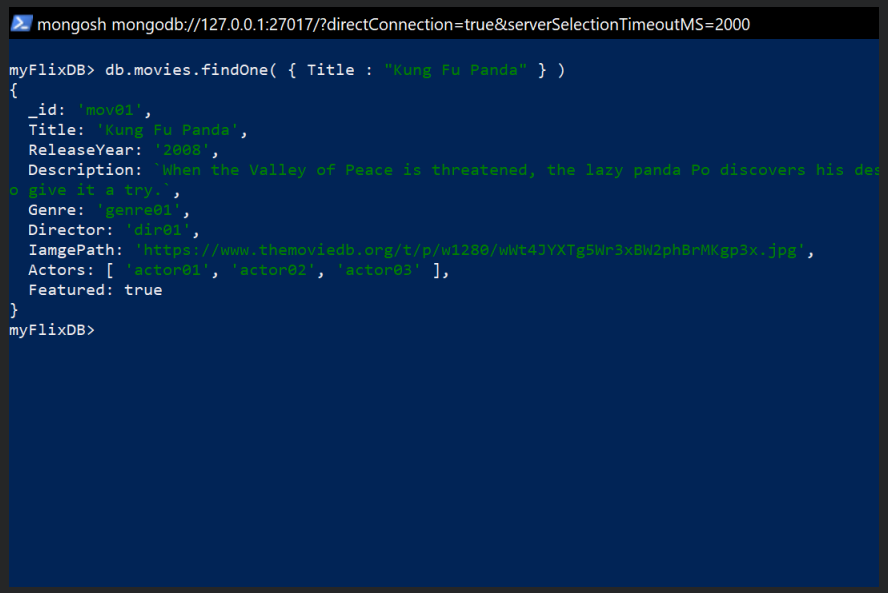
Server-Side

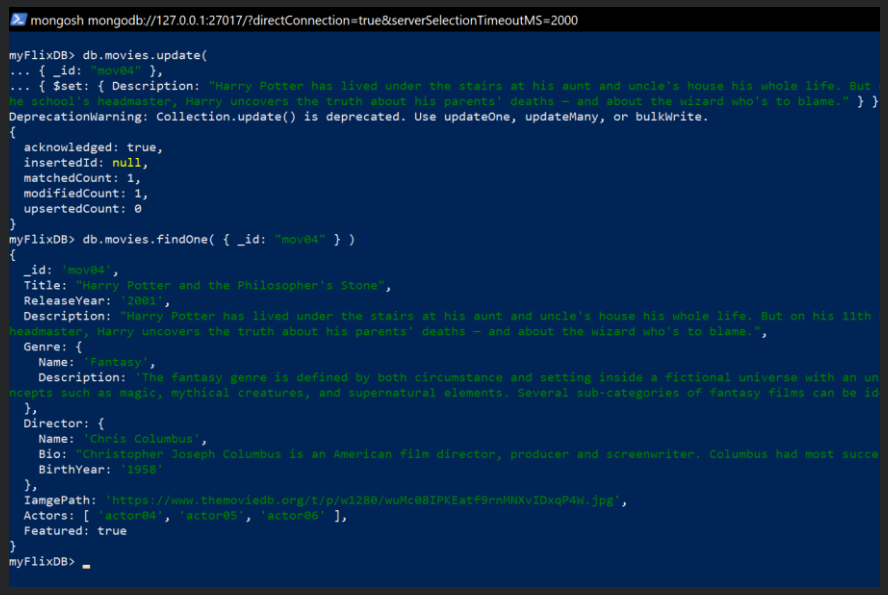
Before writing any code for the API, I took some time to familiarize myself with the terminal and Node. Then I began to define the desired features and endpoints and set up a documentation file with instructions on this API. According to this, I routed the endpoints using Express and tested everything in Postman, as shown in the image below. I also included basic HTTP authentication, JWT authentication, and password hashing with Bcrypt.



The other key element of the API is the database that contains information on all movies and users in JSON format.
I created a non-relational MongoDB database using the command-line client MongoDB Shell (mongosh).
The API uses CRUD methods – Create, Read, Update, Delete – to interact with data on the database.

The images on the right demonstrate what it looks like to use CRUD methods in the terminal when using mongosh.
The upper one shows an example of the Read method, while the bottom image shows the Update method.

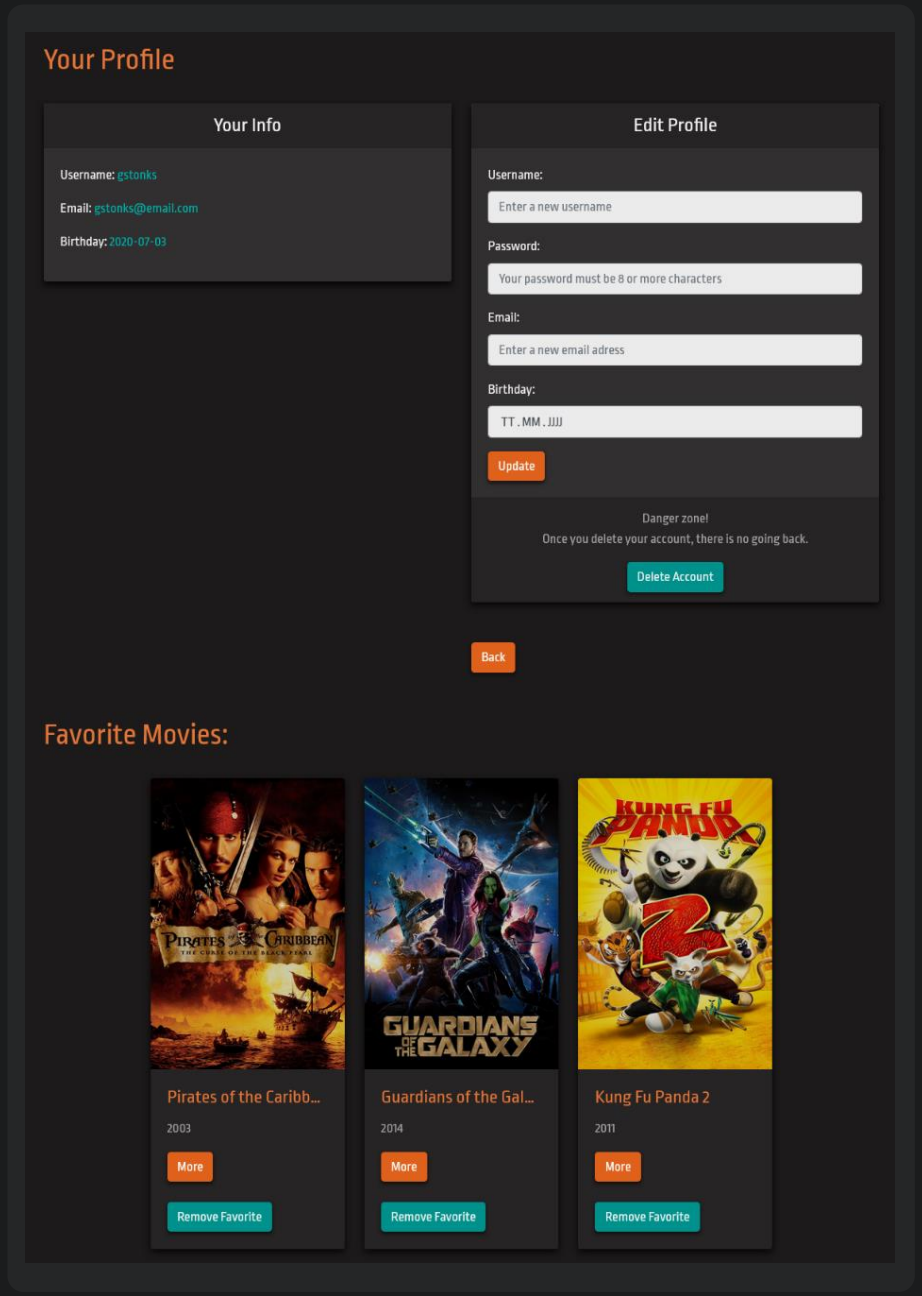




Client-Side

After completing the server side, I was very excited to work on the client side to finally do what I love most – design the look of my app.

To start with, I set up Parcel and React. As soon as I had prototypes of most views, I added React Router to allow navigation between those different views.
Then I populated each view with data and created functioning login and registration forms.
The image on the right shows an example of a single movie view.
It contains information on a movie's genre, director, actors, a short description, and a button to add it to the list of favorites.



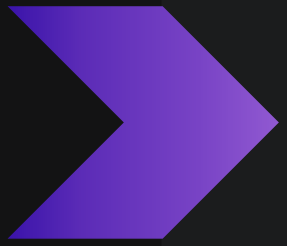
After completing the easy tasks, I added functionality to the user profile page that contains a form to edit user information and a list of favorite movies, as seen on the left image.
While I was stuck on said tasks, I finalized the app design using React-Bootstrap and SCSS.

Finally, after every view and functionality worked as intended, I implemented the Redux store to manage the state.

Challenges

myFlix was my first web development project of this size – and the most challenging one of my CareerFoundry program! I have both enjoyed and hated it. I went from being excited and proud to being frustrated and lost. But why?

Let us have a closer look at the server-side first. The first challenge I encountered was getting familiar with the terminal, as I needed more time to wrap my head around this. One message I have sent to my tutor back then sums it up pretty well:

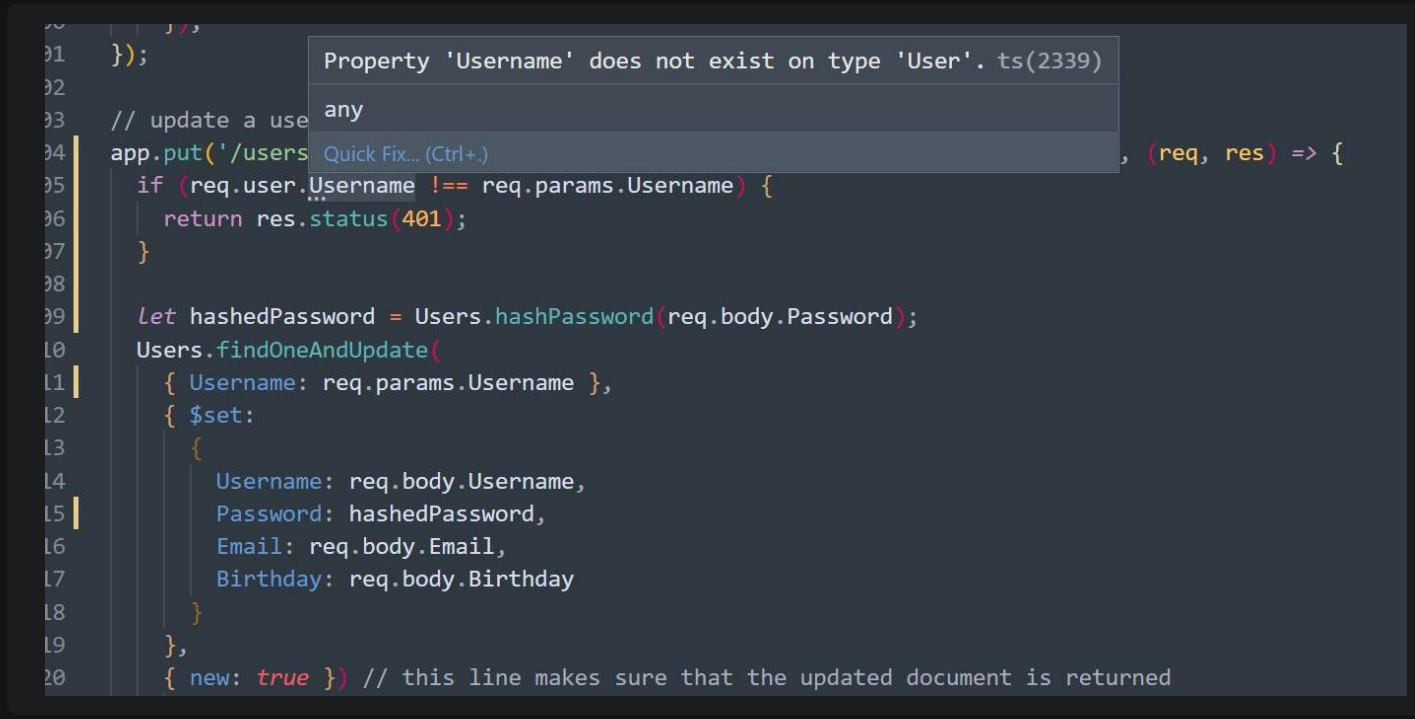


> *"I have produced countless error messages as I tried to understand the terminal and Node, even though it was not that hard to understand."*

Sometimes trial and error is the best way to learn something new.

When creating the database, I had a similar trial and error experience that helped me grasp the concept of MongoDB. Working on the database was very time-consuming as it included researching all the information about the different movies, directors, actors, and genres.

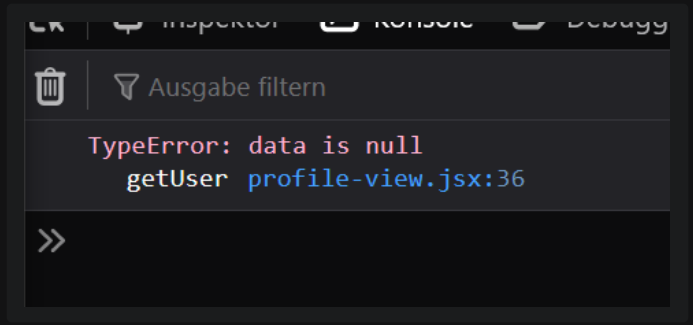
Another challenge on the server-side was to get the API endpoints to work as expected. However, that was not because of any big mistakes, but rather because I confused myself while trying to structure and write everything perfectly.

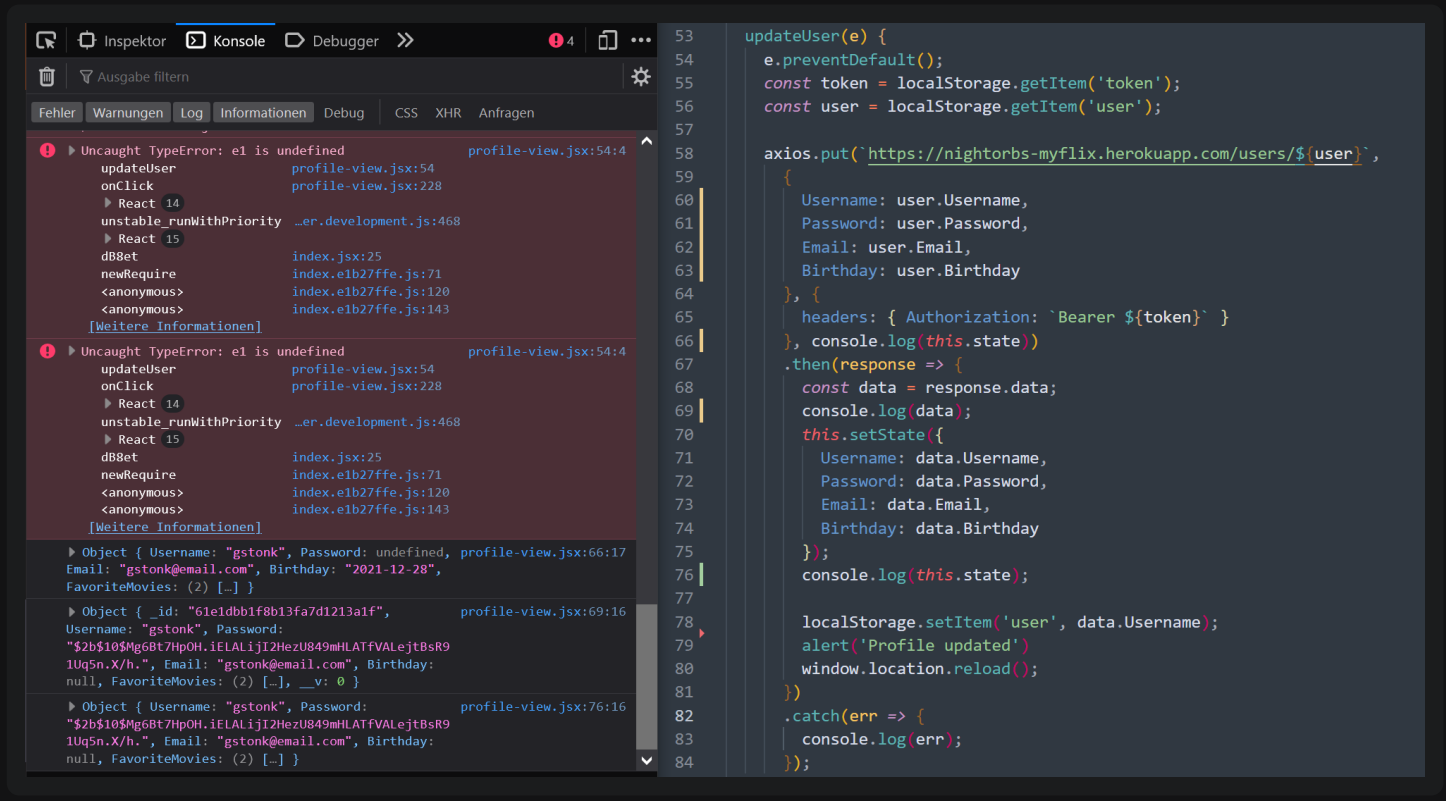


I went back and forth between modifying the database and optimizing the logic for each endpoint – until it stopped working. In many cases, the solution was simple, as it sometimes was just a matter of capitalization (see previous image).
There were some other issues I had not considered, but luckily, I had my mentor who pointed them out.
One of these issues was that my *users/:Username* endpoint was vulnerable to attacks from any authorized user, meaning that any user could edit or delete another account.

That was all for the server-side, so let us look at the client side next. Learning and understanding React were the biggest challenges for me which led to the most frustrating phase of this project.

Many errors occurred due to incorrectly defined functions and props, paired with countless TypeErrors.
One common issue caused by those errors was that a specific view or information would not display correctly or not at all.





The code above is the function for updating user information which I struggled with a lot.

I ended up being stuck, frustrated, and lost, to the point where I could not proceed with this project. It was like an art block – or code block – so the best idea was to take a break to clear my mind. I reached out for help way too late in the process. Eventually, the coding sessions with my mentor and discussions with my tutor helped me out of this misery.

Retrospective

The most challenging part was developing the client-side. I was not expecting to have such a hard time understanding React. However, the whole process taught me to reach out for help early on, and I also leveled up my debugging skills.
As much as I wanted to quit now and then, I am prouder that I successfully pushed through. In the end, I learned to love React, especially for its component-based approach.

What comes next?

Even though I am overall happy with the outcome, there still are a few things to improve. Most importantly, I need to add client-side form validation because currently, users would not know why their registration fails as no error message shows up.
Another flaw is the *Add to Favorites* button that still shows said text after clicking it. Instead, it should change to *Remove from Favorites*.
The last issue is the user's list of favorite movies not being handled by Redux. The relevant code seems incorrect, so I should look at it again.