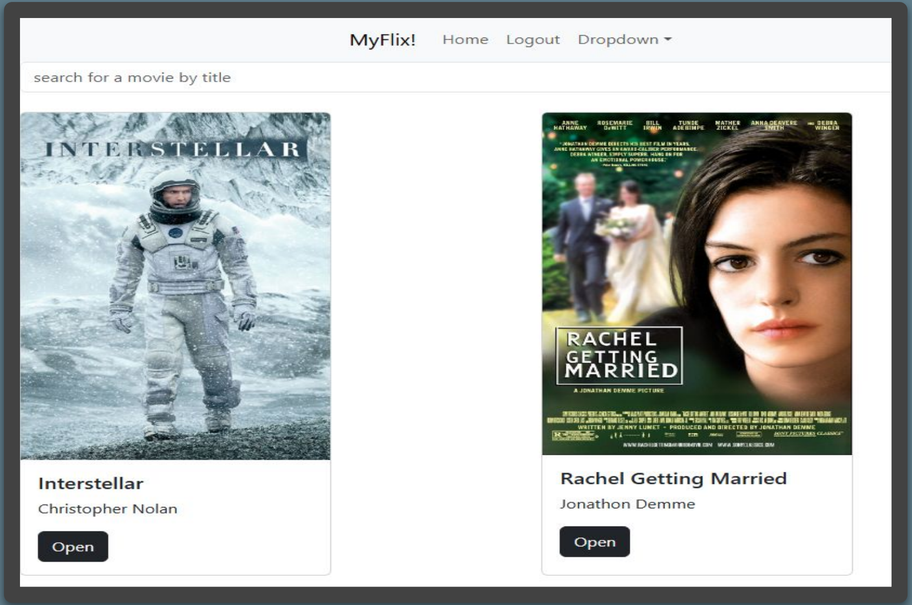
Case Study

myFlix Client

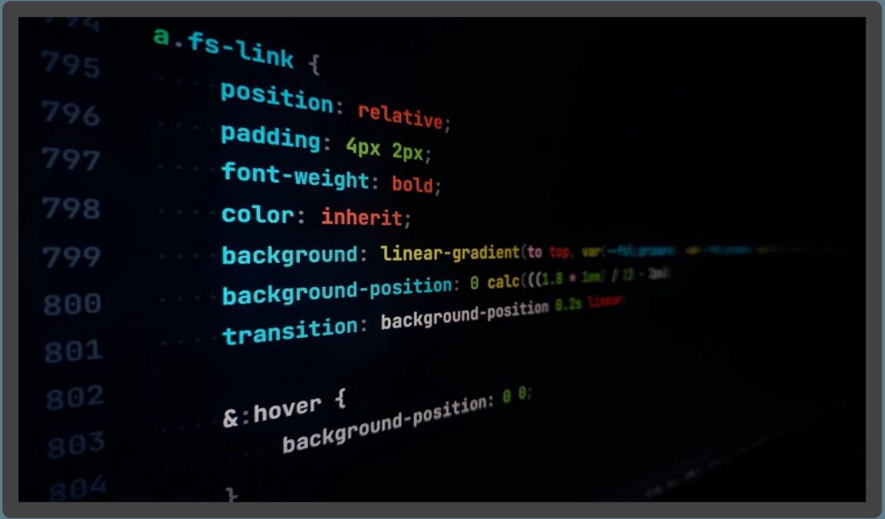
Overview

MyFlix is a web app developed with the MERN stack that allows users to signup, login, and view information on different movies. Users can also update their profile and add movies to their favorites.



Purpose and Context

MyFlix was a project I built following the CareerFoundry Web-Development course. The project focuses on deeper understanding of the framework React.

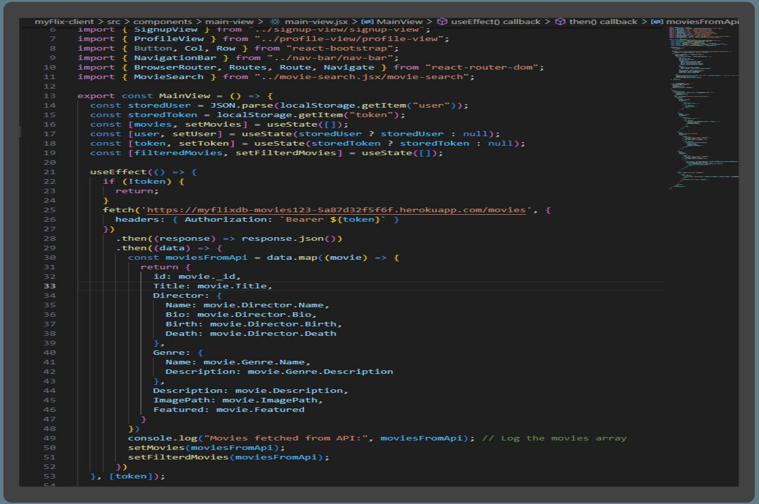


```
a.fs-link {
    position: relative;
    padding: 4px 2px;
    font-weight: bold;
    color: inherit;
    background: linear-gradient(to top, var(--[illegible]
    background-position: 0 calc(((1.8 * 1em) / 2) - 2px)
    transition: background-position 0.2s linear

    &:hover {
        background-position: 0 0;
    }
```

Objective

The objective was to create a functional front-end and back-end from scratch to display as a solid reference for my professional portfolio.



myFlix-client > src > components > main-view > main-view.jsx > MainView > useEffect() callback > then() callback > moviesFromApi

```jsx
import { SignupView } from "../signup-view/signup-view";
import { ProfileView } from "../profile-view/profile-view";
import { Button, Col, Row } from "react-bootstrap";
import { NavigationBar } from "../nav-bar/nav-bar";
import { BrowserRouter, Routes, Route, Navigate } from "react-router-dom";
import { MovieSearch } from "../movie-search.jsx/movie-search";

export const MainView = () => {
  const storedUser = JSON.parse(localStorage.getItem("user"));
  const storedToken = localStorage.getItem("token");
  const [movies, setMovies] = useState([]);
  const [user, setUser] = useState(storedUser ? storedUser : null);
  const [token, setToken] = useState(storedToken ? storedToken : null);
  const [filteredMovies, setFilterdMovies] = useState([]);

  useEffect(() => {
    if (!token) {
      return;
    }
    fetch('https://myflixdb-movies123-5a87d32f5f6f.herokuapp.com/movies', {
      headers: { Authorization: `Bearer ${token}` }
    })
      .then((response) => response.json())
      .then((data) => {
        const moviesFromApi = data.map((movie) => {
          return {
            id: movie._id,
            Title: movie.Title,
            Director: {
              Name: movie.Director.Name,
              Bio: movie.Director.Bio,
              Birth: movie.Director.Birth,
              Death: movie.Director.Death
            },
            Genre: {
              Name: movie.Genre.Name,
              Description: movie.Genre.Description
            },
            Description: movie.Description,
            ImagePath: movie.ImagePath,
            Featured: movie.Featured
          }
        })
        console.log("Movies fetched from API:", moviesFromApi); // Log the movies array
        setMovies(moviesFromApi);
        setFilterdMovies(moviesFromApi);
      })
  }, [token]);
```

Steps of Project

$$\begin{tikzcd}
\boxed{\begin{array}{l}\text{Server Side}\\ \text{Movie API}\end{array}} \arrow[r] & \boxed{\begin{array}{l}\text{Server Side}\\ \text{MongoDB}\end{array}} \arrow[d] \\
 & \boxed{\begin{array}{l}\text{Client Side}\\ \text{React Front-End}\end{array}}
\end{tikzcd}$$

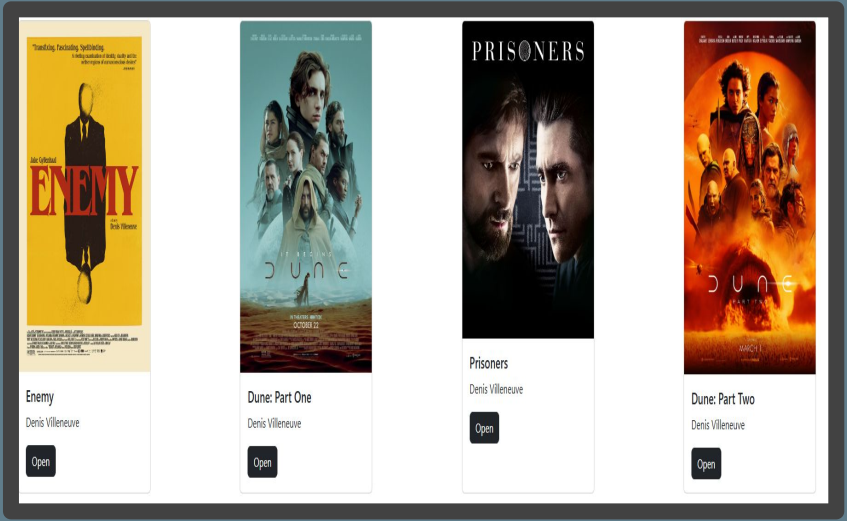
I created an API from scratch, connected with a MongoDB database, and a React Front-End.

Technologies Used:

- Node.js
- React
- MongoDB
- Postman
- Heroku
- Redux
- Express

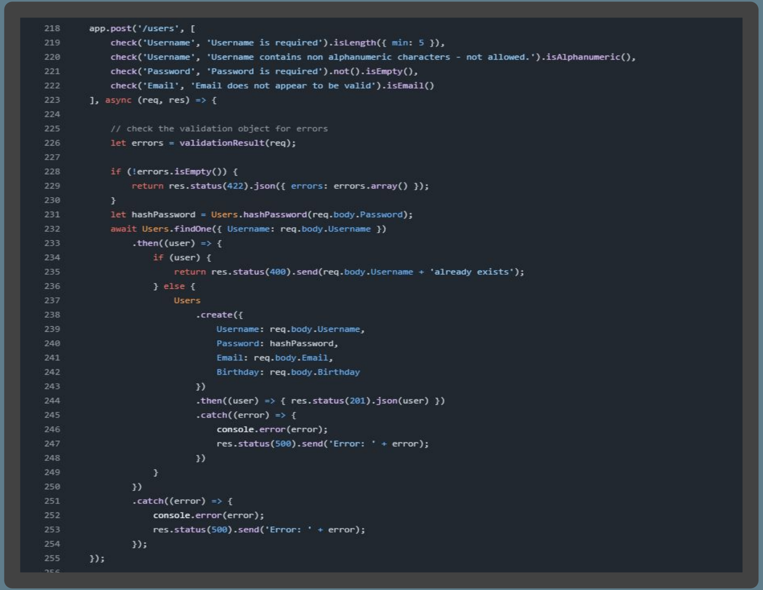
What went well?

Creating the Front-End of MyFlix went the best for me. It was much easier for me to visualize my code and test different features during development. I also had more creativity on the Front-End which allowed me to create a clean looking, easy to navigate application.



Challenges

I struggled the most with the back-end portion of the project. Creating the movie API from scratch was a difficult task because it was harder to visualize my code and solve my bugs. Coding my GET, POST, PUT, and DELETE parts of my API to communicate with my database and working on authorization were the most difficult parts. I solved my issues by rereading over the course material and using stack overflow. I also used a bit of trial and error in Postman until my API was working properly.



```
app.post('/users', [
    check('Username', 'Username is required').isLength({ min: 5 }),
    check('Username', 'Username contains non alphanumeric characters - not allowed.').isAlphanumeric(),
    check('Password', 'Password is required').not().isEmpty(),
    check('Email', 'Email does not appear to be valid').isEmail()
], async (req, res) => {

    // check the validation object for errors
    let errors = validationResult(req);

    if (!errors.isEmpty()) {
        return res.status(422).json({ errors: errors.array() });
    }
    let hashPassword = Users.hashPassword(req.body.Password);
    await Users.findOne({ Username: req.body.Username })
        .then((user) => {
            if (user) {
                return res.status(400).send(req.body.Username + 'already exists');
            } else {
                Users
                    .create({
                        Username: req.body.Username,
                        Password: hashPassword,
                        Email: req.body.Email,
                        Birthday: req.body.Birthday
                    })
                    .then((user) => { res.status(201).json(user) })
                    .catch((error) => {
                        console.error(error);
                        res.status(500).send('Error: ' + error);
                    })
            }
        })
        .catch((error) => {
            console.error(error);
            res.status(500).send('Error: ' + error);
        });
});
```

Retrospective

Final Thoughts:

MyFlix was a successful addition to my professional portfolio and accomplished the objective of the case study. The most surprising part of this project was how smoothly everything came together to create a responsive web application.

Future Steps:

I would like to add some more features and improve the UI of the application. Expanding the database and API to include more data on each movie title to increase the information displayed to the users.