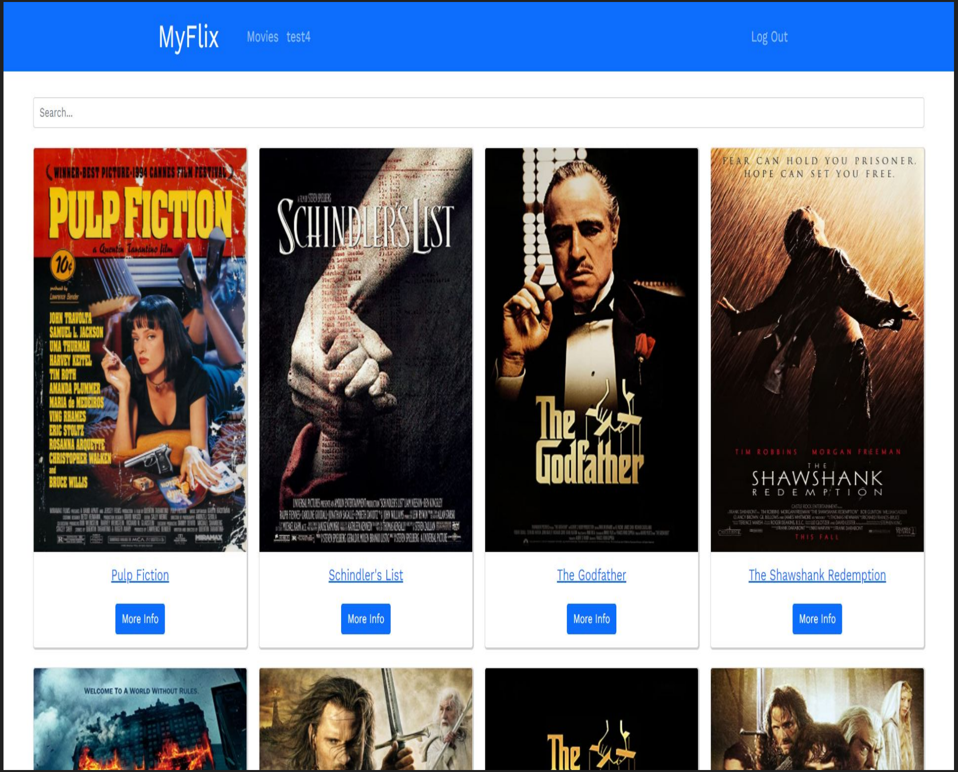
myFlix Case Study

David Caldwell

Overview

- myFlix is a full-stack web app that was developed using the MERN stack.
- It features a selection of classic movies that users can add to their personal list of favorites.
- It allows users to sign-up, sign-in, edit their account details, and delete their account.



Purpose and Objectives

- This app was learning project to deepen my understanding of the full-stack development process.

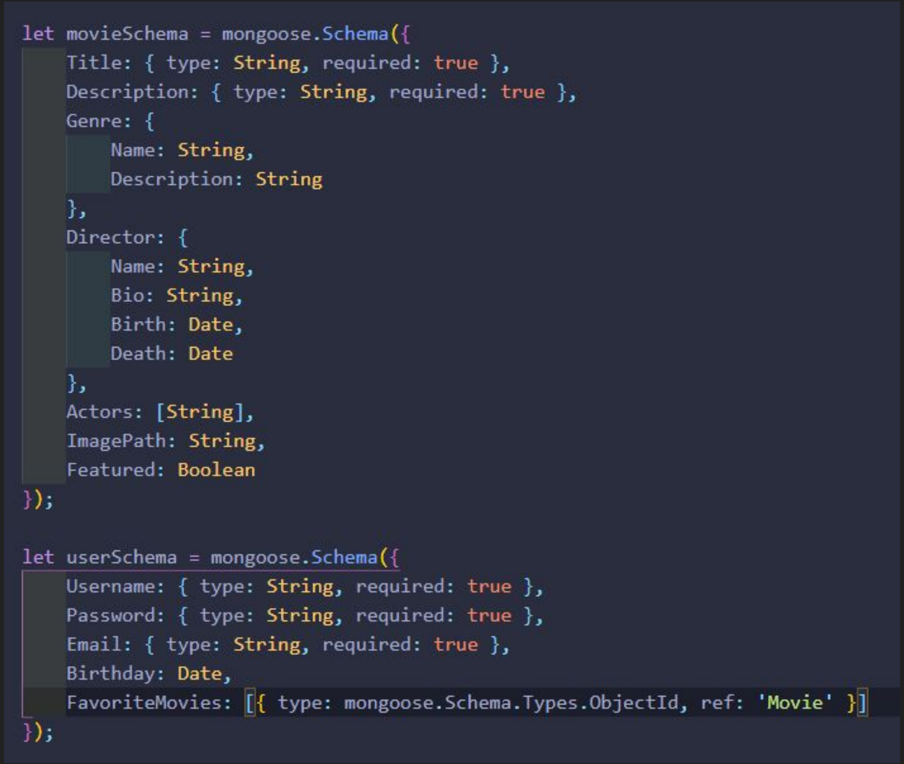
- The goal was to create a full-stack application that included a fully developed back-end – including an API (with create, read, update, and delete functionality), and a database (mongoDB).

- Also, to create a full user experience (sign-up, login, or adding favorite movies to a list) by building a custom front-end client with React.js and Redux.

- By the end, I wanted a full-stack app to show on my portfolio.

Server-Side Approach

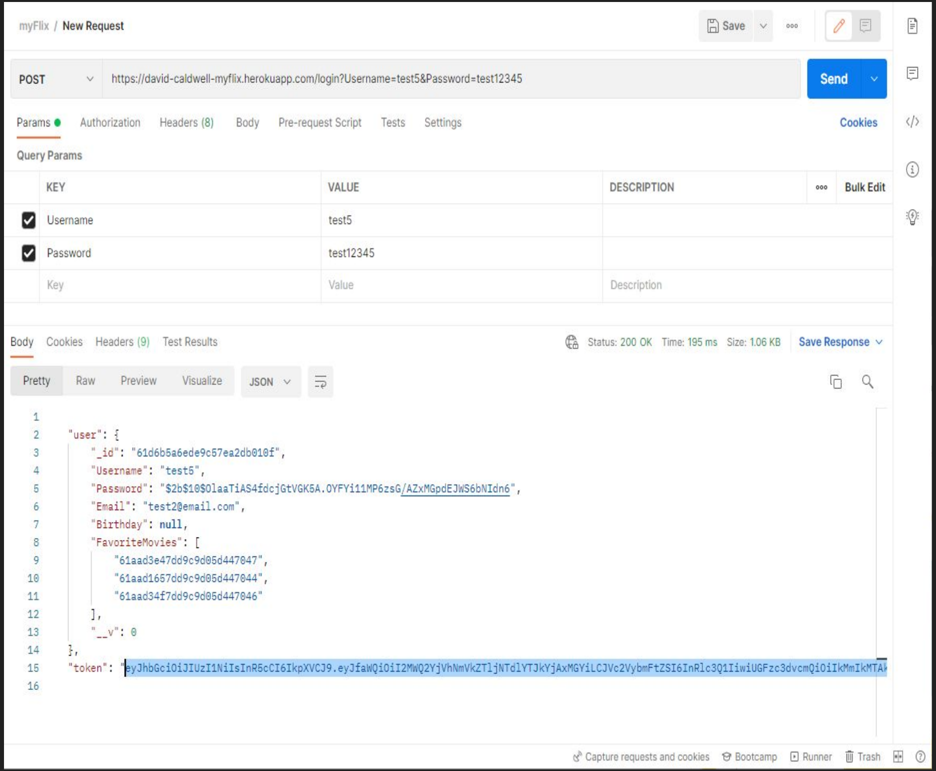
- Starting with Node.js and Express.js, I created RESTful API endpoints with the necessary CRUD functions (create, read, update, and delete). These included things like getting a list of all movies or registering a new user.
- These endpoints needed to interact with a mongoose schema that defines what the movie objects and the user objects will look like in the database.
- Then, using the schema in combination with my endpoints, I could specify what data I wanted to send or receive to and from the database.



```
let movieSchema = mongoose.Schema({
    Title: { type: String, required: true },
    Description: { type: String, required: true },
    Genre: {
        Name: String,
        Description: String
    },
    Director: {
        Name: String,
        Bio: String,
        Birth: Date,
        Death: Date
    },
    Actors: [String],
    ImagePath: String,
    Featured: Boolean
});

let userSchema = mongoose.Schema({
    Username: { type: String, required: true },
    Password: { type: String, required: true },
    Email: { type: String, required: true },
    Birthday: Date,
    FavoriteMovies: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Movie' }]
});
```

Server-Side Approach cont.

- Each request to an endpoint (except the one to register a new user) is restricted unless a valid Json Web Token has been generated for that user.
- On successful login, a JWT is generated, which then authorizes the user to access the data from the other endpoints.
- This basic JWT authentication was tested with postman.
- Then, I hosted the api on heroku with a connection URI to my database which is hosted on the mongoDB Atlas.



https://david-caldwell-myflix.herokuapp.com/login?Username=test5&Password=test12345

	KEY	VALUE	DESCRIPTION
✓	Username	test5	
✓	Password	test12345	
	Key	Value	Description

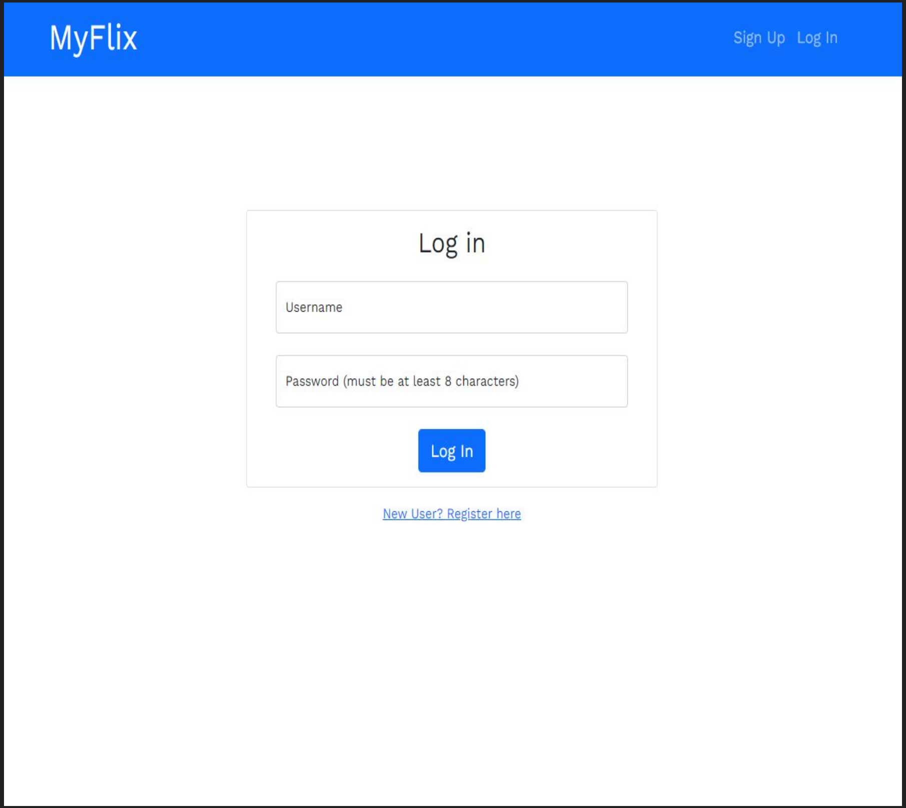
Status: 200 OK Time: 195 ms Size: 1.06 KB

```

  "user": {
    "_id": "61d6b5a6ede9c57ea2db010f",
    "Username": "test5",
    "Password": "$2b$10$OlaaTiAS4fdcjGtVGK5A.OYFYi11MP6zsG/AZxMGpdEJWS6bNIdn6",
    "Email": "test2@email.com",
    "Birthday": null,
    "FavoriteMovies": [
      "61aad3e47dd9c9d05d447047",
      "61aad1657dd9c9d05d447044",
      "61aad34f7dd9c9d05d447046"
    ],
    "__v": 0
  },
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2MWQ2YjVhNmVkZTljNTdlYTJkYjAxMGYiLCJVc2VybmFtZSI6InRlc3Q1IiwiUGFzc3dvcmQiOiIkMmIkMTAk

```

Client-Side Approach

- The client-side is a single page application built with React. The state-management is done with React-Redux.
- It uses axios to make calls to the API and retrieve the relevant data.
- It uses React-Router at the top of the component tree to manage which view should be shown.
- Instead of passing down the movie and user data as props from the top level, Redux allowed me to access the necessary data from whichever view I needed using the global store.
- It has a login and registration form that, when submitted, will send the data to the corresponding API endpoint.
- If the login is successful, the user will have access to the main list of movies as well as the navigation bar which contains options to view the user’s profile or logout.
- Also on login, the JWT is stored in local storage and is retrieved and sent with every axios call.
- Selecting a movie will show its relevant info as well as an option for the user to add it to their list of favorites
- The hosting I used for this app was netlify.

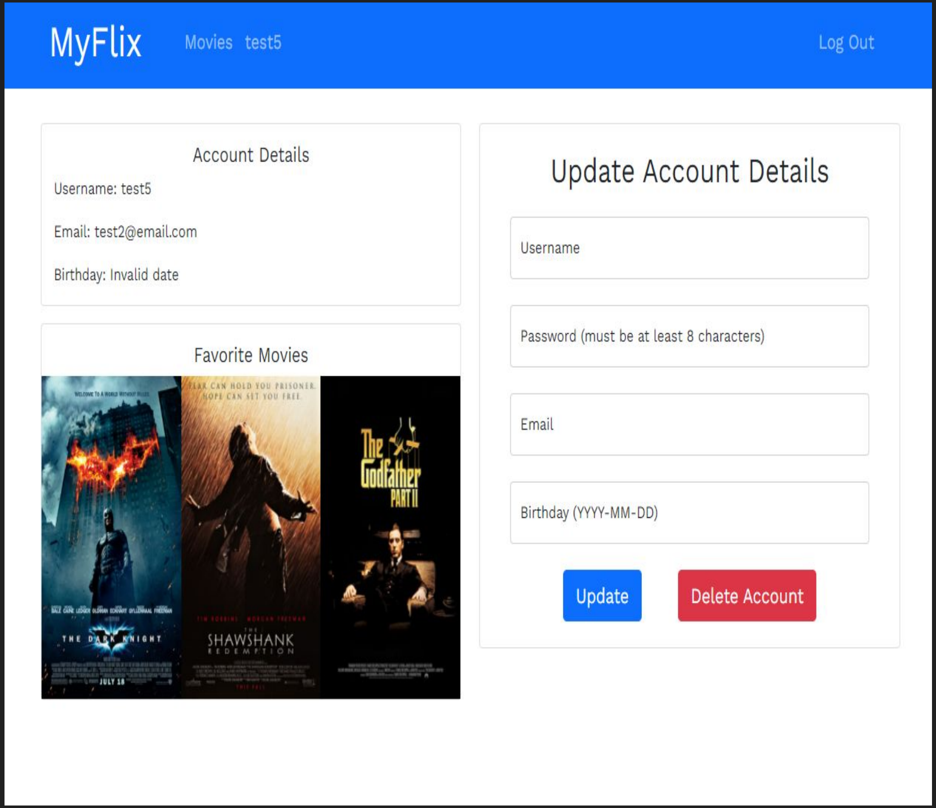
Client-Side Approach cont.



Log in

Username

Password (must be at least 8 characters)



Account Details

Username: test5

Email: test2@email.com

Birthday: Invalid date

Favorite Movies

Update Account Details

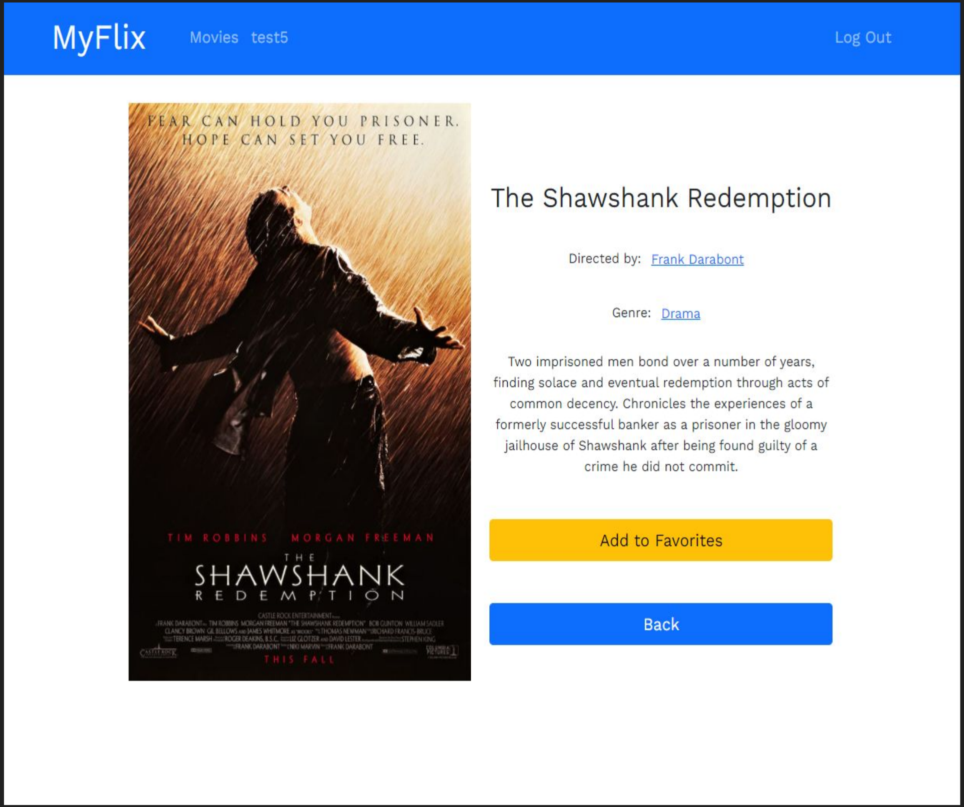
Username

Password (must be at least 8 characters)

Email

Birthday (YYYY-MM-DD)

Client-Side Approach cont.



MyFlix Movies test5 Log Out

The Shawshank Redemption

Directed by: Frank Darabont

Genre: Drama

Two imprisoned men bond over a number of years, finding solace and eventual redemption through acts of common decency. Chronicles the experiences of a formerly successful banker as a prisoner in the gloomy jailhouse of Shawshank after being found guilty of a crime he did not commit.

Add to Favorites

Back

FEAR CAN HOLD YOU PRISONER. HOPE CAN SET YOU FREE.
TIM ROBBINS MORGAN FREEMAN
THE SHAWSHANK REDEMPTION
THIS FALL

Challenges

- This was a very challenging project because there a multiple pieces that each require careful planning and execution on their own. User authentication and authorization were particularly complicated at first, but became clear with time and careful reading.
- At first the client side gave me trouble because in React, it can be difficult to manage the state of some data when that data needs to be used and modified by many different views.
- Redux was a bit of a light-bulb moment when it came to this problem. Having a global state to refer to made accessing data between the different views much easier.