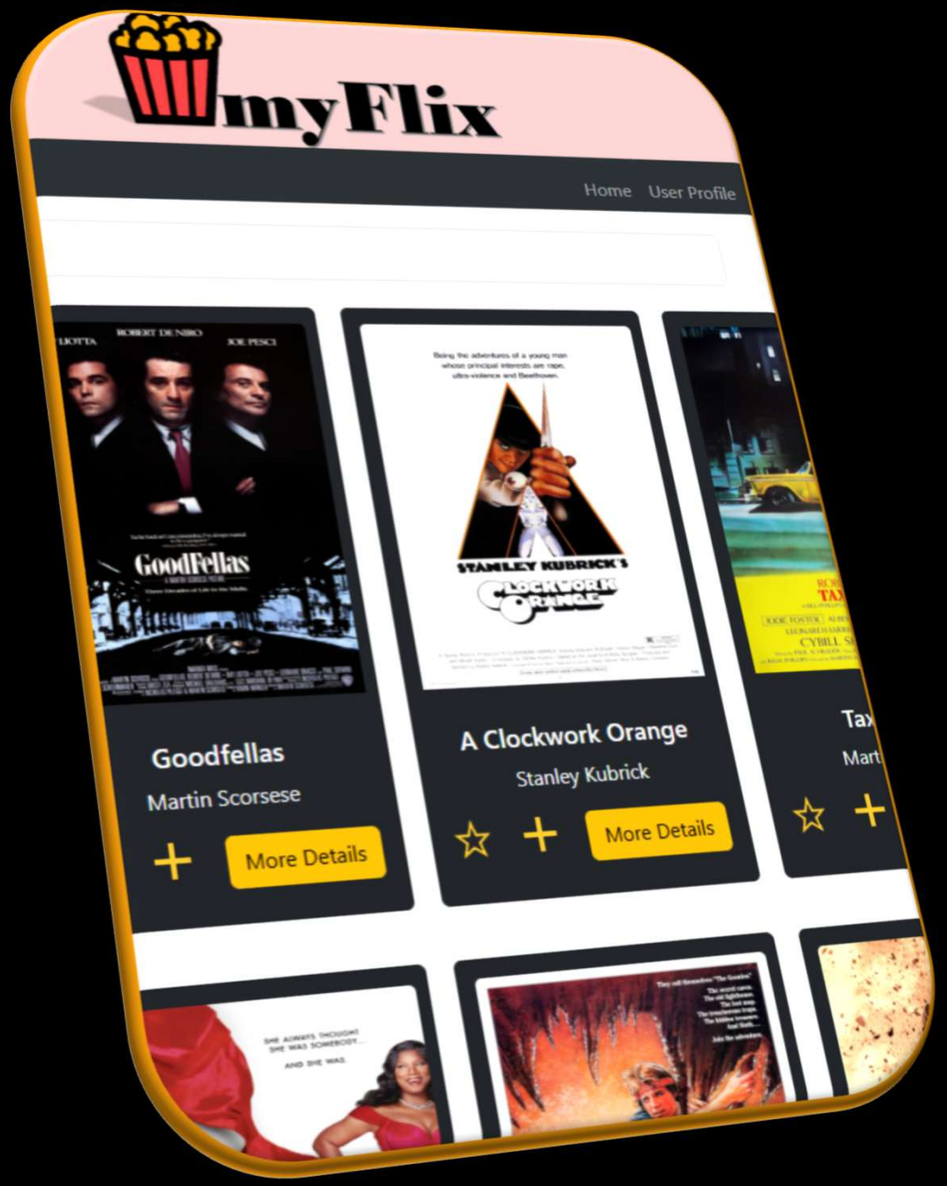
myFlix

CASE STUDY

Elizabeth Howell

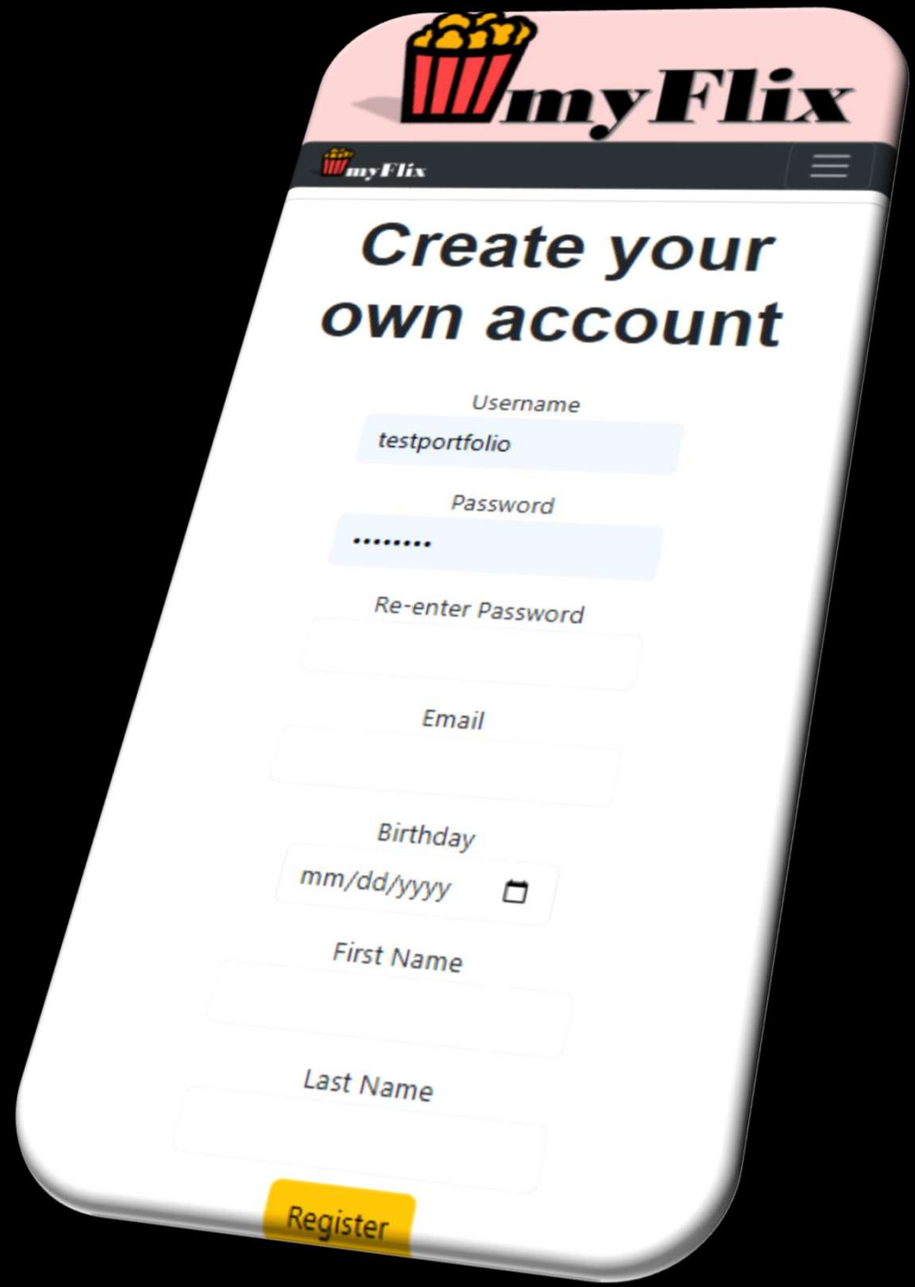


OVERVIEW

- myFlix is a full-stack (MERN) single-page web app functioning as a movie database
- Users can browse the curated list of movies with detailed info including descriptions, directors, and actors
- Registered users can create a profile, manage their account, and add movies to their favorites and a watch list

PURPOSE & CONTEXT

- A project developed during Career Foundry Full-Stack Web Development Course
- The goal was to gain experience with full-stack development by building the backend API, the database, and the frontend client from scratch
- This project emphasized RESTful architecture, database integration, secure user authentication, and dynamic frontend rendering



OBJECTIVE

myFlix

To design and develop a fully functioning full-stack web app

- Build a backend server with a structured API to interact with a NoSQL Database
- Implement secure user authentication and data validation
- Create a responsive single-page frontend UI for browsing movies, viewing details, and managing user profile

TOOLS, TECH, & SKILLS

Language & Frameworks: JavaScript, HTML, CSS, React, Node.js, Express

Database: MongoDB with Mongoose ODM

Authentication: JSON Web Tokens (JWT), password hashing

State Management: Redux

Development Tools: Postman, Prettier, Parcel, Netlify, Heroku

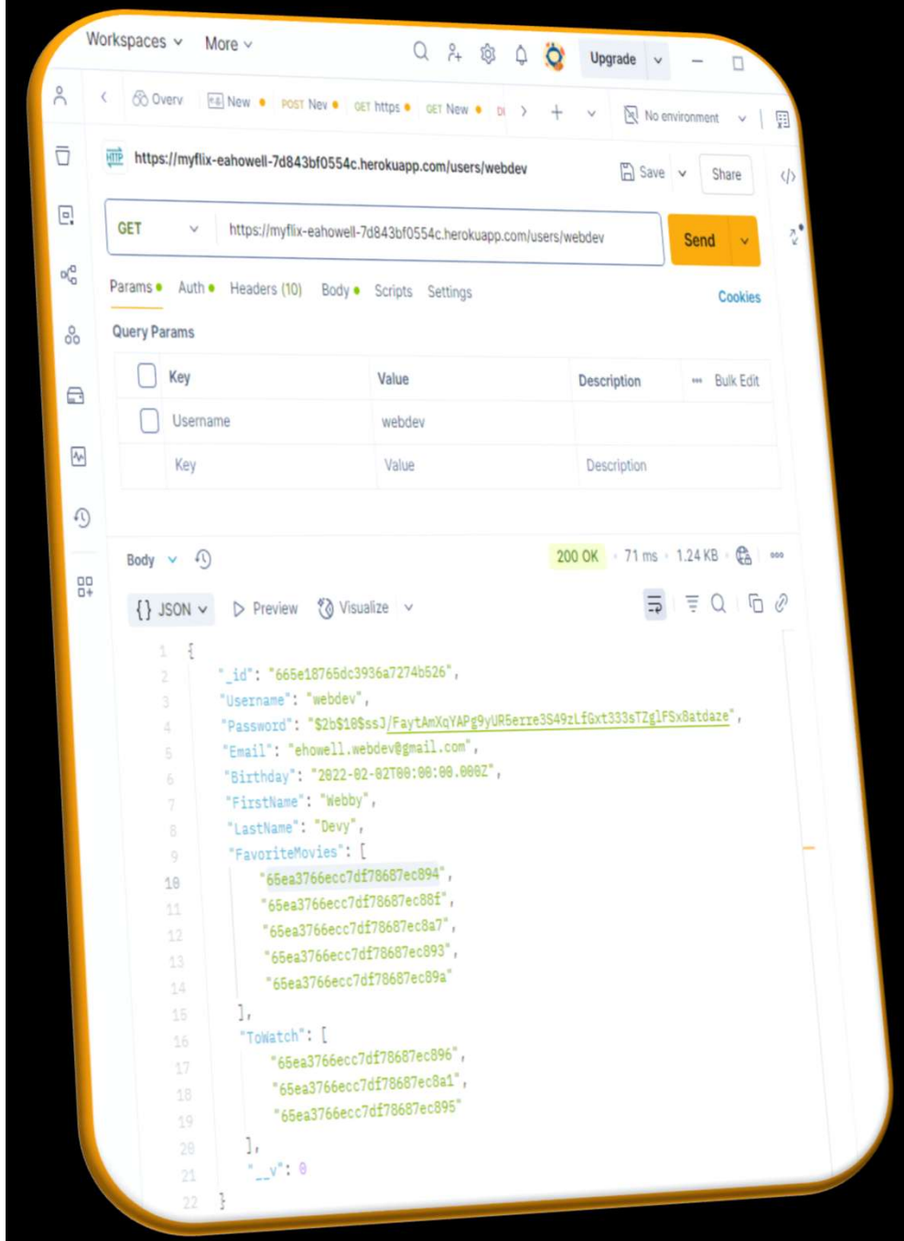
UI & Styling: React Bootstrap

Security: CORS, Express Validator

DEV APPROACH: BACKEND

(NODE.JS + EXPRESS + MONGODB)

- Set up Express server and routing architecture
- Designed RESTful API Endpoints for CRUD operations
- Created documentation in GitHub README
- Integrated MongoDB and modeled data using Mongoose



DEV APPROACH: BACKEND

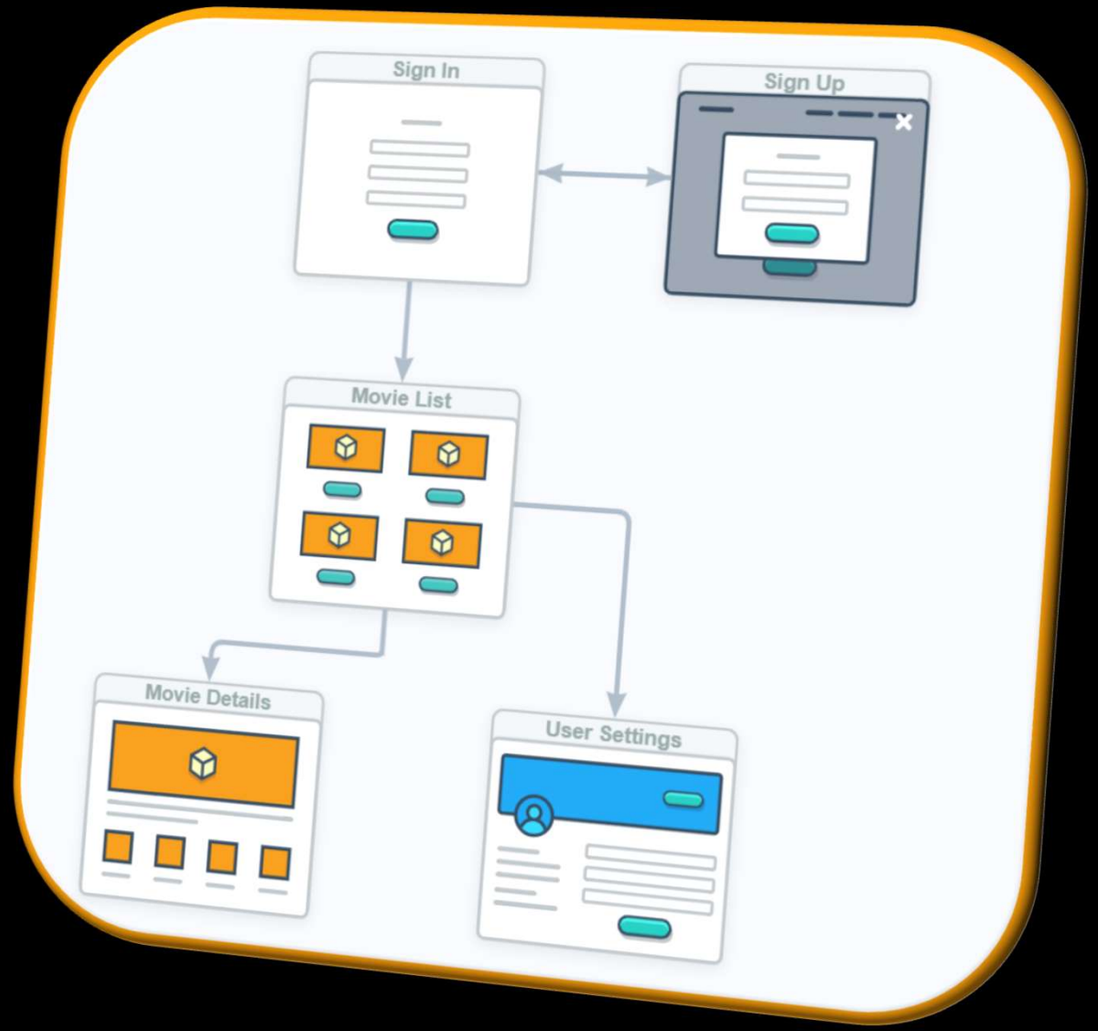
(NODE.JS + EXPRESS + MONGODB)

- Implemented user auth with JWT and hashed passwords
- Applied data validation using Express Validator
- Handled cross-origin resources sharing
- Test endpoints and API logic with Postman

DEV APPROACH: FRONTEND

(REACT + REDUX)

- Mapped user flows and designed architecture
- Managed application states using Redux
- Enable client-side routing for smooth navigation



DEV APPROACH: FRONTEND

(REACT + REDUX)

- Built modular views and components
- Developed shared components
- Maintained consistent UI with custom branding and React Bootstrap

FINAL PRODUCT

Backend

[API Repo]

Frontend

[Client Repo]

Live App

[myFlix App]

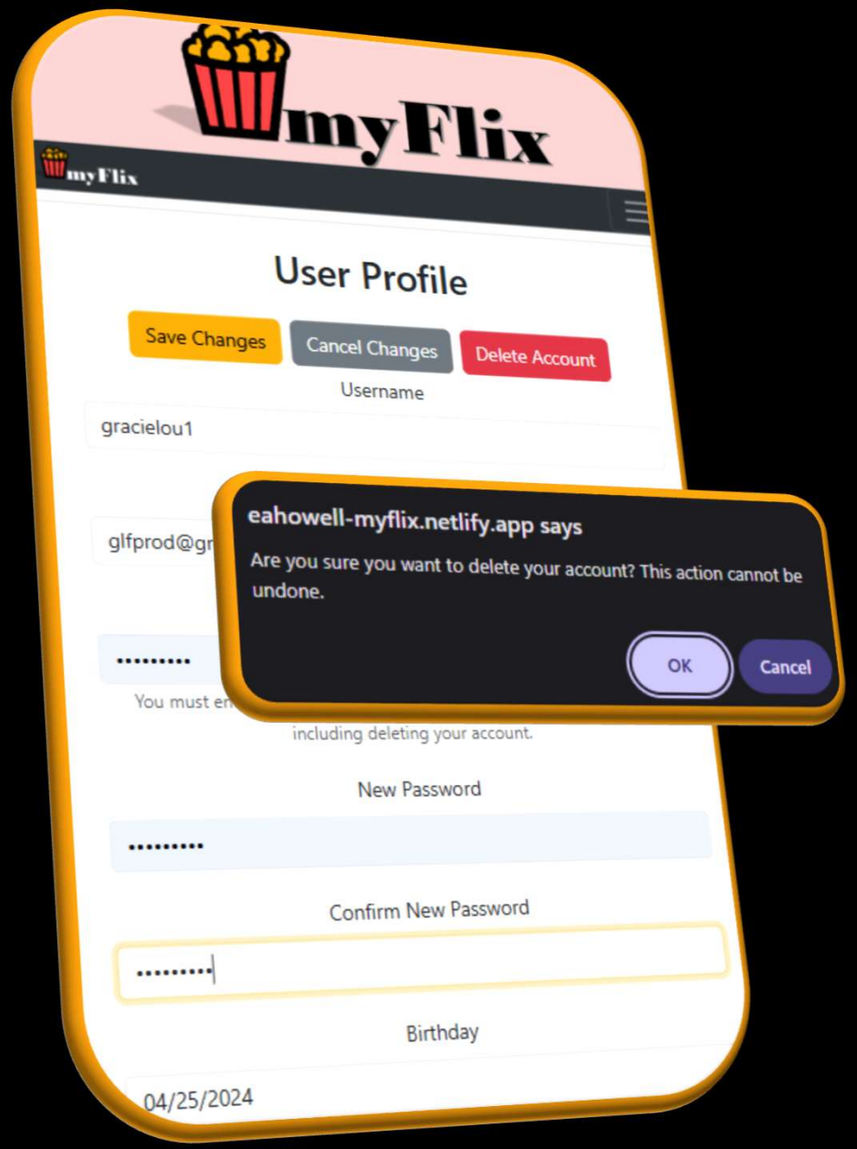
Username : testportfolio

Password : 12345678

```
app.post(
  '/users',
  [
    // Username must be between 5 and 15 letters and be alphanumeric
    check('Username', 'Username is required').isLength({ min: 5, max: 15 }),
    check(
      'Username',
      'Username contains non alphanumeric characters - not allowed.'
    ).isAlphanumeric(),
    // Password must be present and between 8 and 25 characters
    check('Password', 'Password is required').not().isEmpty(),
    check(
      'Password',
      'Password must be between 8 and 25 characters long.'
    ).isLength({ min: 8, max: 25 }),
    // First & Last Name, Email, and Birthday must be present
    check('FirstName', 'First Name is required').not().isEmpty(),
    check('LastName', 'Last Name is required').not().isEmpty(),
    check('Birthday', 'Birthday is required').not().isEmpty(),
    check('Email', 'Email is required').not().isEmpty(),
    check('Email', 'Email does not appear to be valid').isEmail(),
  ],
  async (req, res, info) => {
    // Check the validation object for errors
    let errors = validationResult(req)
    if (!errors.isEmpty()) {
      return res.status(422).json({ errors: errors.array() })
    }

    await Users.findOne({ Username: req.body.Username })
      .then((user) => {
        if (user) {
          console.log('Username ' + req.body.Username + ' already exists')

          return {
            res: res
              .status(409)
              .send('Username ' + req.body.Username + ' already exists'),
          }
        } else {
          let hashPassword = Users.hashPassword(req.body.Password)
          Users.create({
            Username: req.body.Username.toLowerCase(),
            Password: hashPassword,
            Email: req.body.Email,
            Birthday: req.body.Birthday,
            FirstName: req.body.FirstName,
            LastName: req.body.LastName,
          })
            .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)
      })
  }
)

```

CHALLENGES

- Adding Redux after the fact, in the future would start with using Redux for state management
- Identifying where the error was originating: the server or the client
  - Utilized console logging (both on Heroku and in browser)
  - Added more error code and more descriptive error messages



FINAL THOUGHTS

- Able to achieve completing the full-stack and meet requirements
- Improved the UI beyond the requirements to have a more real world set up

Examples:

- Requiring the password to make changes to the account
- "Are you sure?" Confirmation dialogue when deleting account
- When changing password, having a re-enter new password field