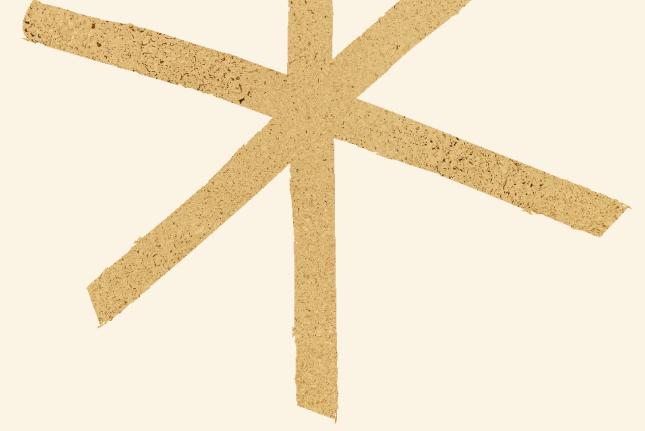
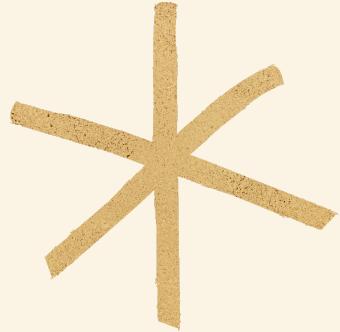


*ReactJS Web App - MyFlix*

# CASE STUDY



## *overview*

Full stack application using React, JWT, and Node.js for serverside (which queries a REST API built from scratch) allows you to create a profile, browse movies, favorite movies, explore directors, and update information..

## *Purpose and Context*

myFlix is a portfolio project built as part the Full-Stack Immersion course at CareerFoundry. This two part project covered front and backend development

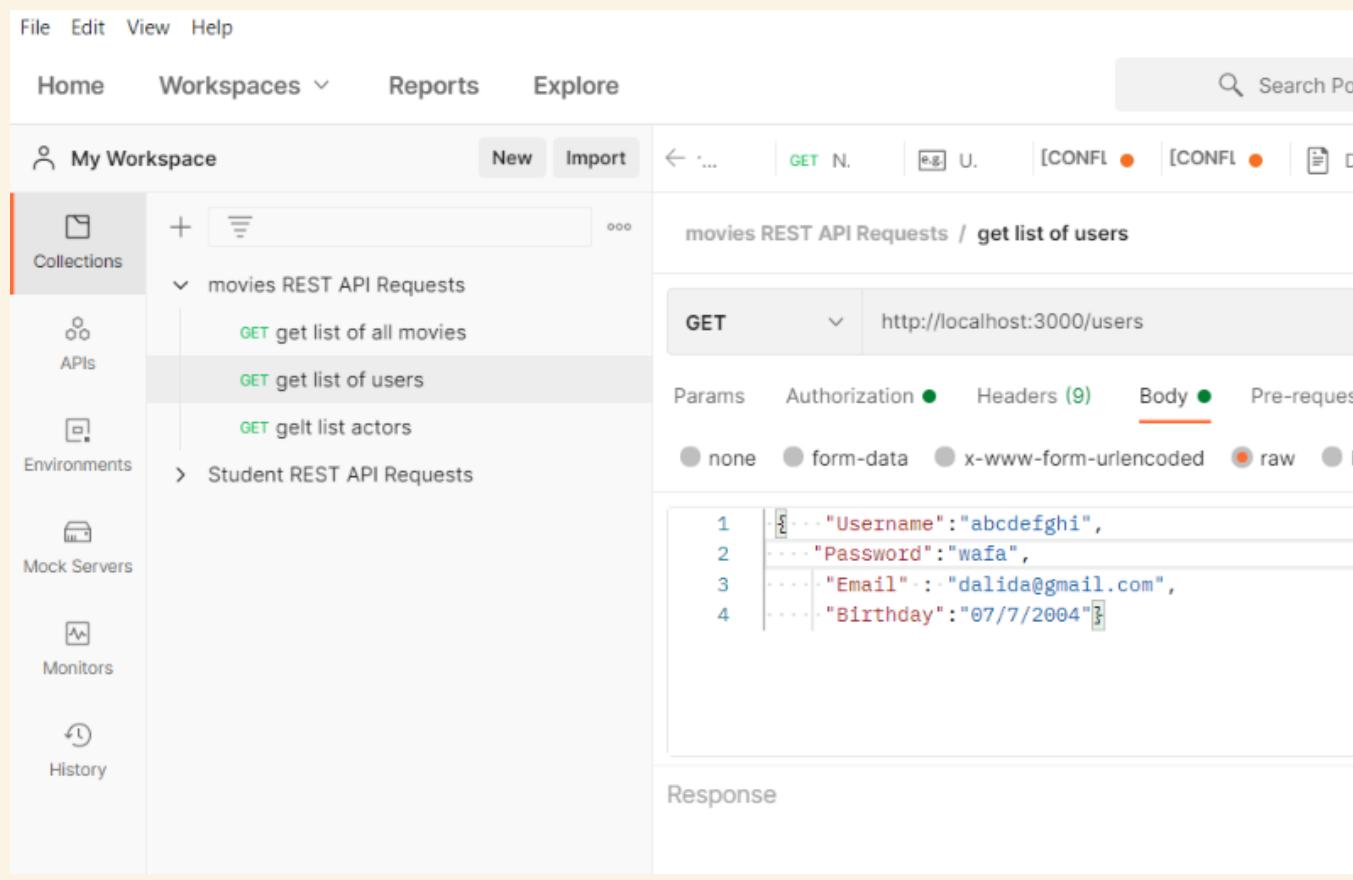
## *Objective*

The web application will provide users with access to information about different movies, directors, and genres. Users will be able to sign up, update their personal information, and create a list of their favorite movies.

## *Approach*

## *Server-Side*

Throughout this Achievement, I've added an entirely new repertoire to my bank of coding knowledge, learning how to go beyond the frontend, or client-side, of your application to create databases and server-side scripts in the backend. I now have a complete REST API I can call my own, along with a non-relational database in which to store the data for my app.



```

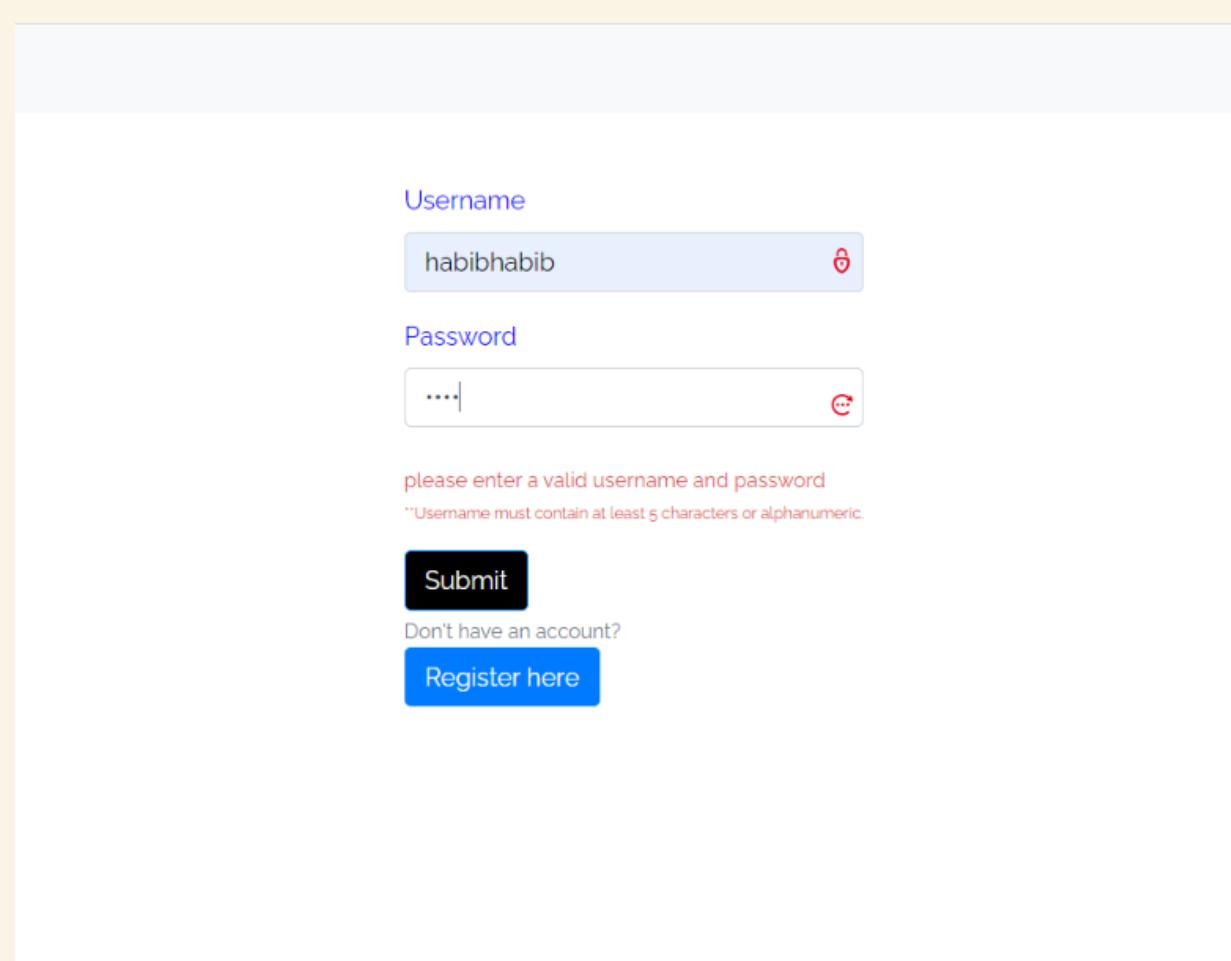
60      useUnifiedTopology: true,
61    });
62    /*
63     mongoose.Promise = global.Promise;
64     mongoose.connect(
65       'mongodb://wafa-chaari:movieapidb30@cluster0.zhrb5.mongodb.net/movieAPIDB?retryWrites=true',
66       {useNewUrlParser: true, useUnifiedTopology: true},
67     );
68
69     mongoose.connection
70       .once('open', function() {
71         console.log('Connection has been made!');
72       })
73       .on('error', function(error) {
74         console.log('Error is: ', error);
75       });
76
77     // GET requests
78   > app.get('/', (req, res) => {
79     res.json({
80       title: 'Movie API',
81       subtitle: 'A simple movie API built with Node.js, Express, MongoDB, and React'
82   >   app.get('/documentation', (req, res) => {
83     res.json({
84       title: 'Movie API Documentation',
85       subtitle: 'A simple movie API built with Node.js, Express, MongoDB, and React'
86     });
87
88     /**
89      *function get All Movies
90      *@description get ALL movies from the database
91      @returns {JSON} JSON object of all movies, each of which contain the movie's title, descri
92
93     app.get(
94       '/movies',
95       passport.authenticate('jwt', {session: false}),
96       (req, res) => {
97         Movies.find()
98           .then(movies => {
99             res.json(movies);
100            res.end();
101          })
102        .catch(err => {
103          res.json({error: err.message});
104        });
105      }
106    );
107
108    /**
109     *function post Create Movie
110     *@description Create a new movie in the database
111     @param {Object} movie - An object containing the movie details to be created
112     @returns {Object} The created movie object
113
114    app.post(
115      '/movies',
116      passport.authenticate('jwt', {session: false}),
117      (req, res) => {
118        const movie = req.body;
119        Movies.create(movie)
120          .then(createdMovie => {
121            res.json(createdMovie);
122          })
123        .catch(err => {
124          res.json({error: err.message});
125        });
126      }
127    );
128
129    /**
130     *function put Update Movie
131     *@description Update an existing movie in the database
132     @param {Object} movie - An object containing the movie details to be updated
133     @param {String} movieId - The ID of the movie to be updated
134     @returns {Object} The updated movie object
135
136    app.put(
137      '/movies/:movieId',
138      passport.authenticate('jwt', {session: false}),
139      (req, res) => {
140        const movie = req.body;
141        const movieId = req.params.movieId;
142        Movies.findByIdAndUpdate(movieId, movie)
143          .then(updatedMovie => {
144            res.json(updatedMovie);
145          })
146        .catch(err => {
147          res.json({error: err.message});
148        });
149      }
150    );
151
152    /**
153     *function delete Delete Movie
154     *@description Delete a movie from the database
155     @param {String} movieId - The ID of the movie to be deleted
156     @returns {Object} A confirmation message indicating the movie was deleted
157
158    app.delete(
159      '/movies/:movieId',
160      passport.authenticate('jwt', {session: false}),
161      (req, res) => {
162        const movieId = req.params.movieId;
163        Movies.findByIdAndDelete(movieId)
164          .then(deletedMovie => {
165            res.json({message: 'Movie deleted successfully'});
166          })
167        .catch(err => {
168          res.json({error: err.message});
169        });
170      }
171    );
172  });
173
174  // Error handling
175  app.use((err, req, res, next) => {
176    res.status(500).json({error: err.message});
177  });
178
179  // Start the server
180  const PORT = process.env.PORT || 3000;
181  app.listen(PORT, () => {
182    console.log(`Server is running on port ${PORT}`);
183  });

```

- At first I created a RESTful API by using **NodeJS** and **Express**.
- Then, I created a relational (SQL) database as a non-relational (NoSQL) database using **MongoDB**
- The REST API will be accessed via commonly used HTTP methods like GET and POST. Similar methods (**CRUD**) was used to retrieve data from the database and store that data in a non-relational way
- In order to test the API, I used Postman

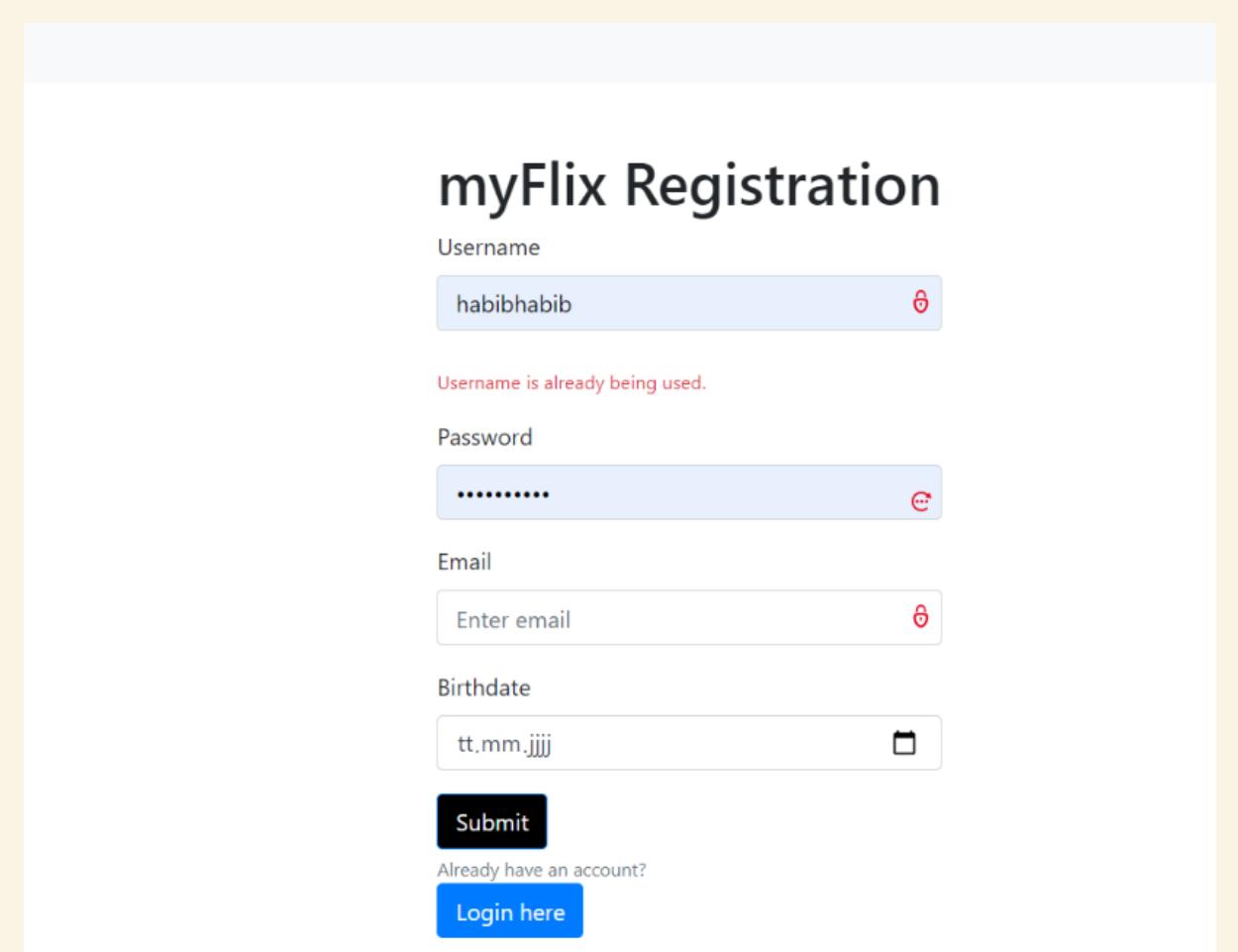
## Client-Side

After building the server-side ,I moved from the back to the front, and build the client-side of my MyFlix app. First, I developed the interface views built using the React library that users will interact with to make their requests, such as searching for movies and updating their details. Responses from the server will be rendered into the interface: movie synopses, director biographies, genres, and more.



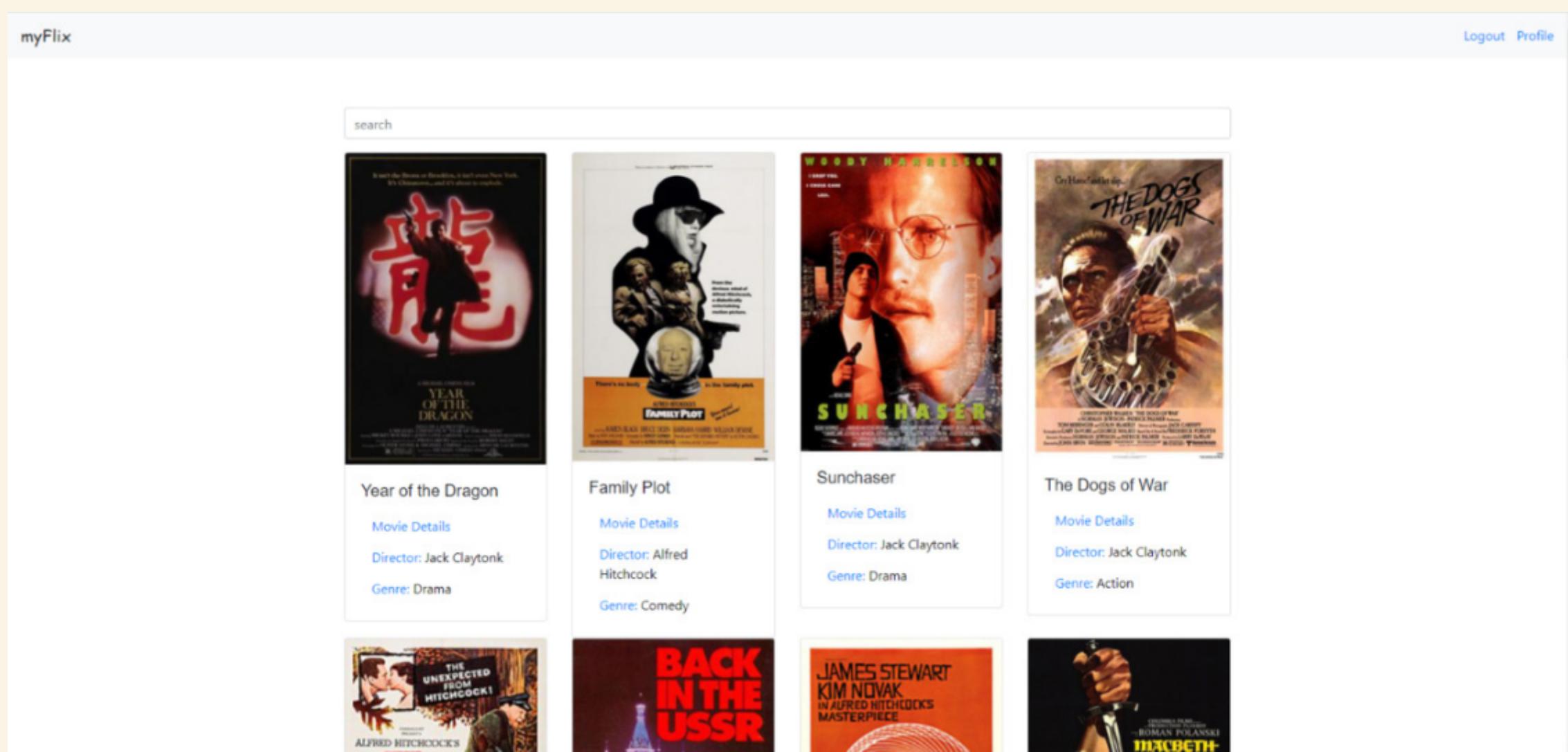
Username  
habibhabib  
Password  
....  
please enter a valid username and password  
"Username must contain at least 5 characters or alphanumeric"  
Submit  
Don't have an account?  
Register here

Login View



myFlix Registration  
Username  
habibhabib  
Username is already being used.  
Password  
.....  
Email  
Enter email  
Birthdate  
tt.mm.jjjj  
Submit  
Already have an account?  
Login here

Registration View



## Profile View

**Update-UserInformation**

username

Password

Email

Birthday

**Favorite Movies**

Family Plot Remove

## Profile View

- The application used state **routing** to navigate between views and share URLs
- The application used Parcel as its build tool
- The application must be written with **React Redux**
- The application must use **Bootstrap** as a UI library for styling and responsiveness
- The application must contain a mix of class components and function components

## *Challenges*

This was my most challenging project so far. It took me quite a while, until understanding how React and Redux work.

when I completed this project I was quite happy and proud. For the future I am planning to add some more movies and further information about the directors and actors.

## *credits*

Role: Lead Developer

Tutor: Itua Akhator

Mentor: Vinh Tuong Mai