



myFlix



A Full Stack Case Study
Stephanie Leon

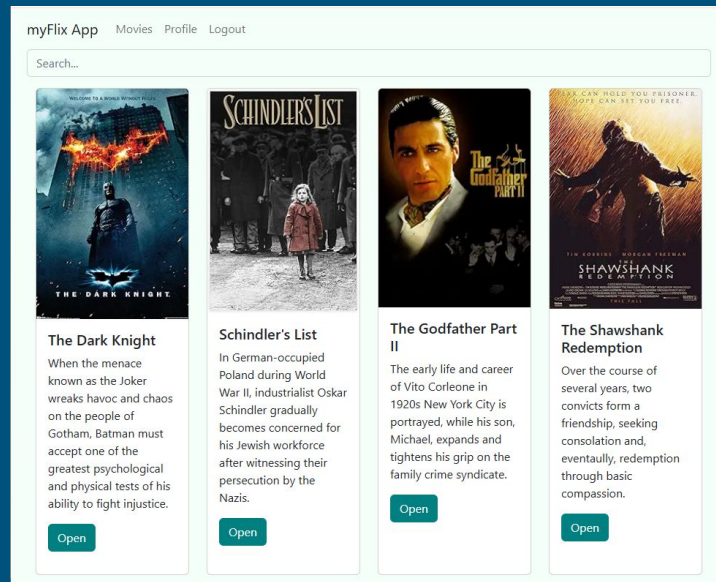


Overview

myFlix is a web application designed using the MERN stack.

Through the app, users are able to access information about the film, the director and the genre.

Users can create an account, update their personal information and save a list of favorite movies.



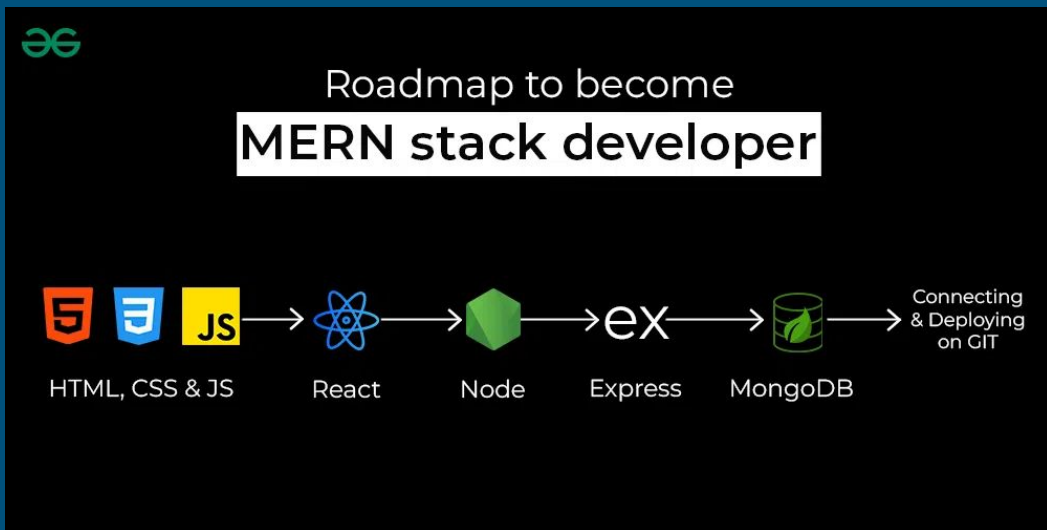
Purpose & Context



myFlix was created as part of a full stack web development course I completed through CareerFoundry. This was my first opportunity to create a full stack application.

Objective

The aim of this project was to build a complete app with a server-side and client-side using a MERN stack that can be added to a professional portfolio.



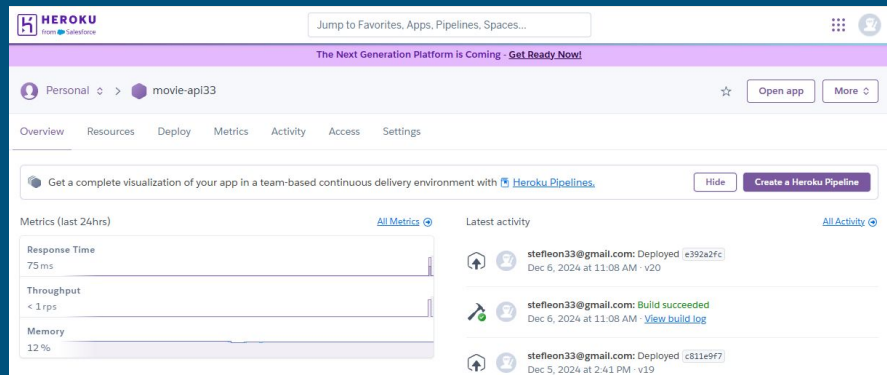
Designing the Solution: Server-Side

Objective: Develop a RESTful API to support a movie application's data operations.

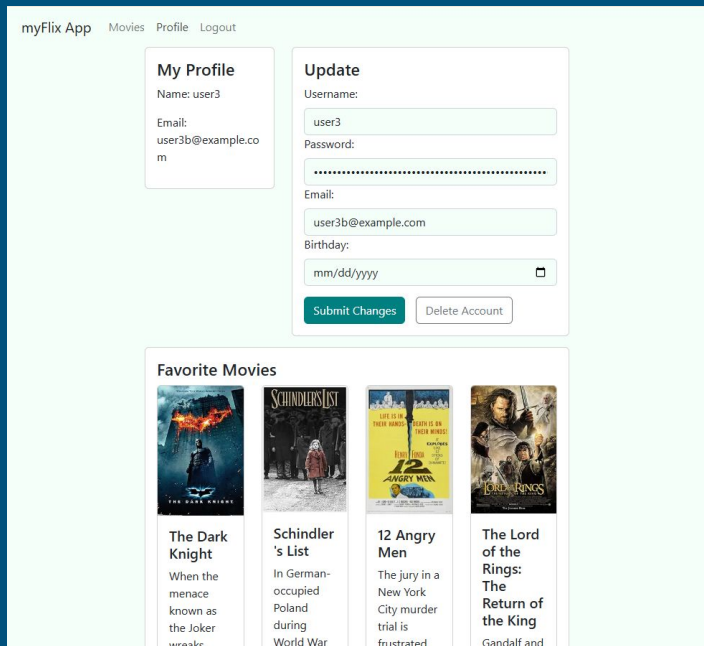
Technologies Used: Node.js, Express, MongoDB, Mongoose

Core Features:

- CRUD operations for movies, genres, and directors.
- User authentication with JWT and data security protocols.
- Data validation and error handling.
- Deployment to Heroku for public accessibility



Designing the Solution: Client-Side



Objective: Create a single-page React application as the client-side of myFlix.

Technologies Used: React, React Router, Bootstrap, Parcel and Redux

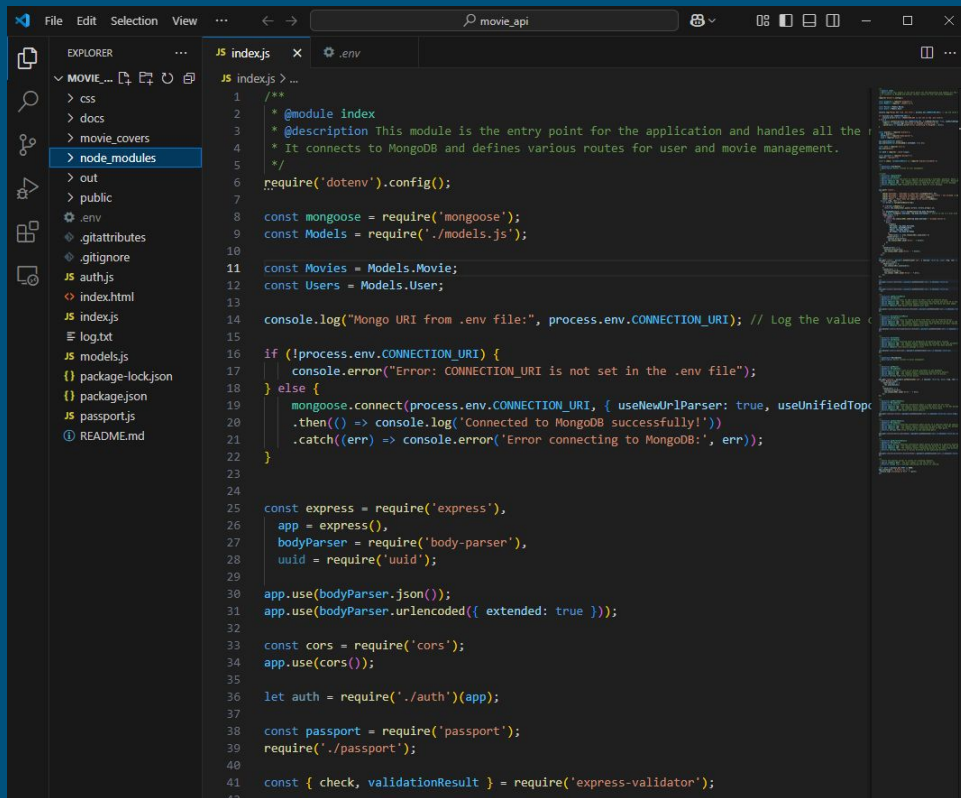
Core Features:

- Interactive views: Login, Signup, Main, Movie, Profile
- Responsive design for seamless use across devices
- Integration with the backend API for real-time data handling

Development Process: Server-Side

Backend Implementation:

- Set up the Node.js and Express server and defined RESTful endpoints
- Connected the backend to a MongoDB database using Mongoose for data modeling
- Integrated JWT-based user authentication and authorization
- Validated and secured data with middleware like body-parser and bcrypt
- Deployed the backend to Heroku



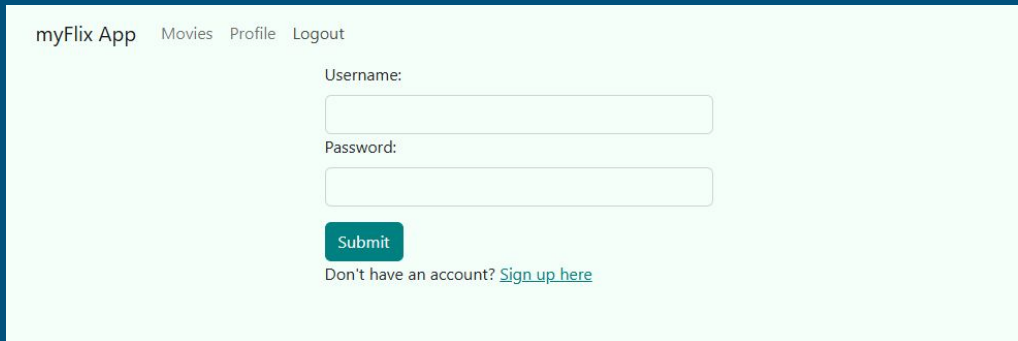
The screenshot shows a VS Code editor with a project named 'movie_api'. The Explorer sidebar on the left shows the file structure: 'MOVIE_API' (expanded), 'css', 'docs', 'movie_covers', 'node_modules', 'out', 'public', '.env', '.gitattributes', '.gitignore', 'JS' (expanded), 'auth.js', 'index.html', 'index.js', 'log.txt', 'models.js', 'package-lock.json', 'package.json', 'passport.js', and 'README.md'. The main editor area shows the 'index.js' file with the following code:

```
1  /**
2   * @module index
3   * @description This module is the entry point for the application and handles all the
4   * It connects to MongoDB and defines various routes for user and movie management.
5   */
6  require('dotenv').config();
7
8  const mongoose = require('mongoose');
9  const Models = require('./models.js');
10
11  const Movies = Models.Movie;
12  const Users = Models.User;
13
14  console.log("Mongo URI from .env file:", process.env.CONNECTION_URI); // Log the value
15
16  if (!process.env.CONNECTION_URI) {
17    console.error("Error: CONNECTION_URI is not set in the .env file");
18  } else {
19    mongoose.connect(process.env.CONNECTION_URI, { useNewUrlParser: true, useUnifiedTopology: true })
20      .then(() => console.log('Connected to MongoDB successfully!'))
21      .catch(err => console.error('Error connecting to MongoDB:', err));
22  }
23
24
25  const express = require('express');
26  app = express();
27  bodyParser = require('body-parser');
28  uuid = require('uuid');
29
30  app.use(bodyParser.json());
31  app.use(bodyParser.urlencoded({ extended: true }));
32
33  const cors = require('cors');
34  app.use(cors());
35
36  let auth = require('./auth')(app);
37
38  const passport = require('passport');
39  require('./passport');
40
41  const { check, validationResult } = require('express-validator');
```

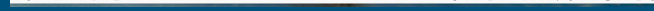
Development Process: Client-Side

Frontend Implementation:

- Structured the React application with distinct components for each view
- Implemented routing for seamless navigation
- Integrated the React app with the REST API to handle CRUD operations
- Deployed the frontend online for public access



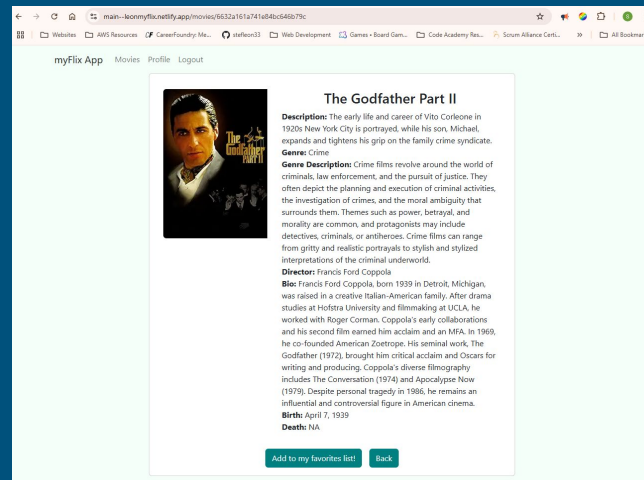
The screenshot displays the login interface of the 'myFlix App'. At the top left, the app's name 'myFlix App' is shown, followed by navigation links for 'Movies', 'Profile', and 'Logout'. The main form area contains two input fields: 'Username:' and 'Password:'. Below these fields is a green 'Submit' button. At the bottom, there is a link for users who do not have an account: 'Don't have an account? [Sign up here](#)'.



Final Solution

The completed myFlix application provides:

- A robust backend supporting movie data and user operations
- A polished, responsive, and user-friendly front end allowing users to:
 - Browse and search for movies
 - View detailed movie, genre, and director information
 - Manage user profiles and favorite movie lists



Lessons Learned

Backend:

- Stick to the exercise brief:
 - I spent a lot of time setting up the movie data in second exercise when it wasn't necessary, then had to set the data up in the proper format later in the achievement.
- Pay attention to the details when coding
 - the placement of quotes, the letters that are capitalized, the exact name of variables

Frontend:

- If the frontend code should be working, and isn't, take a look at the backend code.

Future Iterations

- Additional features like actor information, movie ratings, “To Watch” lists
- Improved scalability and performance through serverless architecture

Duration:

I completed the server-side in 2 months and the client-side in a month and a half.

Credits:

Lead Developer: Stephanie Leon
Tutor: Jesus Diaz
Mentor: Shreyansh Kumar