Case Study for myFlix full-stack project

INTRODUCTION

- **Overview**
- Movie-Api [MyFlix App] uses standard coding languages and usually utilizes REST architecture that provides responses in the lightweight JSON standard, it also stores back-end data about different movies as well as provides users with a variety of information on a list of movies. Users can create an account, view a list of movies by genre or director, and can create a list of their own Favourite movies.

- **Purpose & Context**
- Movie-Api was one of my projects, built as part of the requirements for the completion of my web development course at CareerFoundry to exhibit full competence in JavaScript and HTML.
- **Objective**
- The project aimed at the ability to display and showcase my in-depth knowledge of server-side and client-side coding (REST API and database), ability to use MongoDB Atlas and deployment to Render. The problem that I was trying to solve is to build a responsive web application.

MIDDLE SECTION

MY ROLE

- In this project I've been the developer, to showcase my abilities of Full Stack Web Developer. I adopted the MERN paradigm (MongoDB, Express, React, Node).
- I started from the server-side application, with designing a REST API. I decided in which form the data would be stored and delivered to the client application. I picked MongoDB to experiment with non-relational databases and utilized Node and Express to interact with it.
- Then I moved on to the client-side application, where I had the chance to practice with React. Before starting to implement the functionalities, I created the app with several interface views including a movie view, a login view, a registration view and a profile (where users can update their user data and list of favorites). After completing the API, I built the interface users would need to interact with the server-side. I wanted the app to be intuitive and clear but still colorful and fun since the main concept is "entertainment".

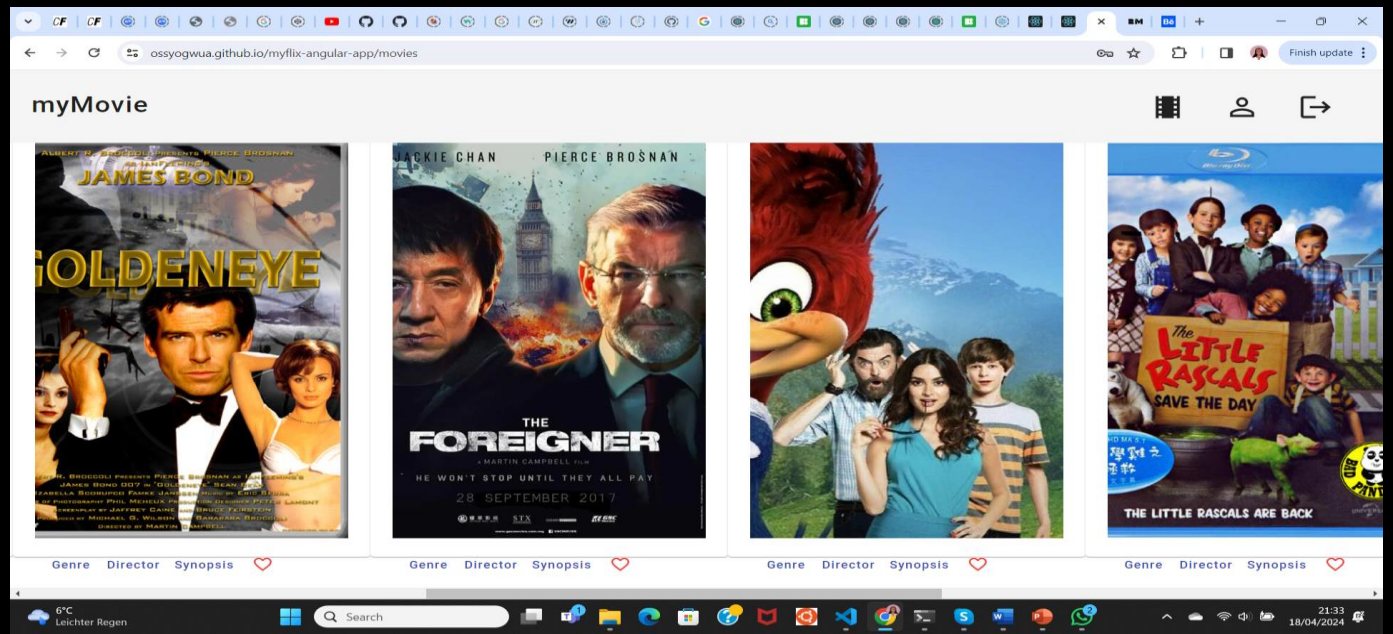
Approach

- The server application Movie- API follows a REST architecture, which has been implemented by using Node and Express to realize the web service, while the data has been stored on a NoSQL database, MongoDB.
- The information has been modelled into JavaScript objects and then turned into NoSQL documents using the JSON format. It was interesting to learn about non-relational databases and how efficient they can be with their innovative structure. Considering the small size of this application, the use of sub-documents turned out more efficient than having separate tables for each entity only.

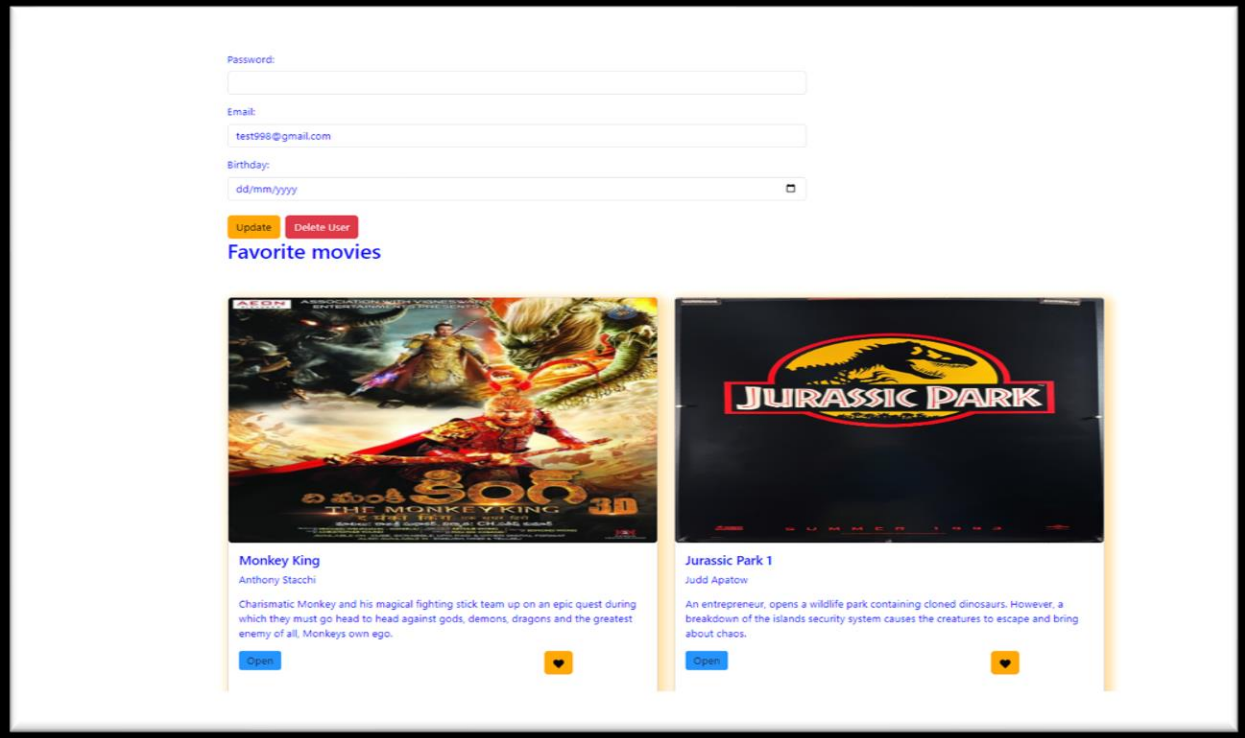
- To complete the server application, I documented it using JSDoc.
- The application has been deployed on Render.

Client –side

- I built the client-side movie app using React and React-Redux, it is a single-page, responsive application, that allows users to create and login into their account, access information about different movies and save them to their list of favorites. Users can also update their personal information and delete their account.
- The services are used to run the queries on the database, the models define how the data is structured and make sure that the types are consistent with the ones that come from the storage. I implemented JWT(JSON Web Tokens) authentication to allow access to the data, so every request that gets sent to the server contains a bearer token in the header that needs to get verified; the response will be sent only in case of successful authentication.

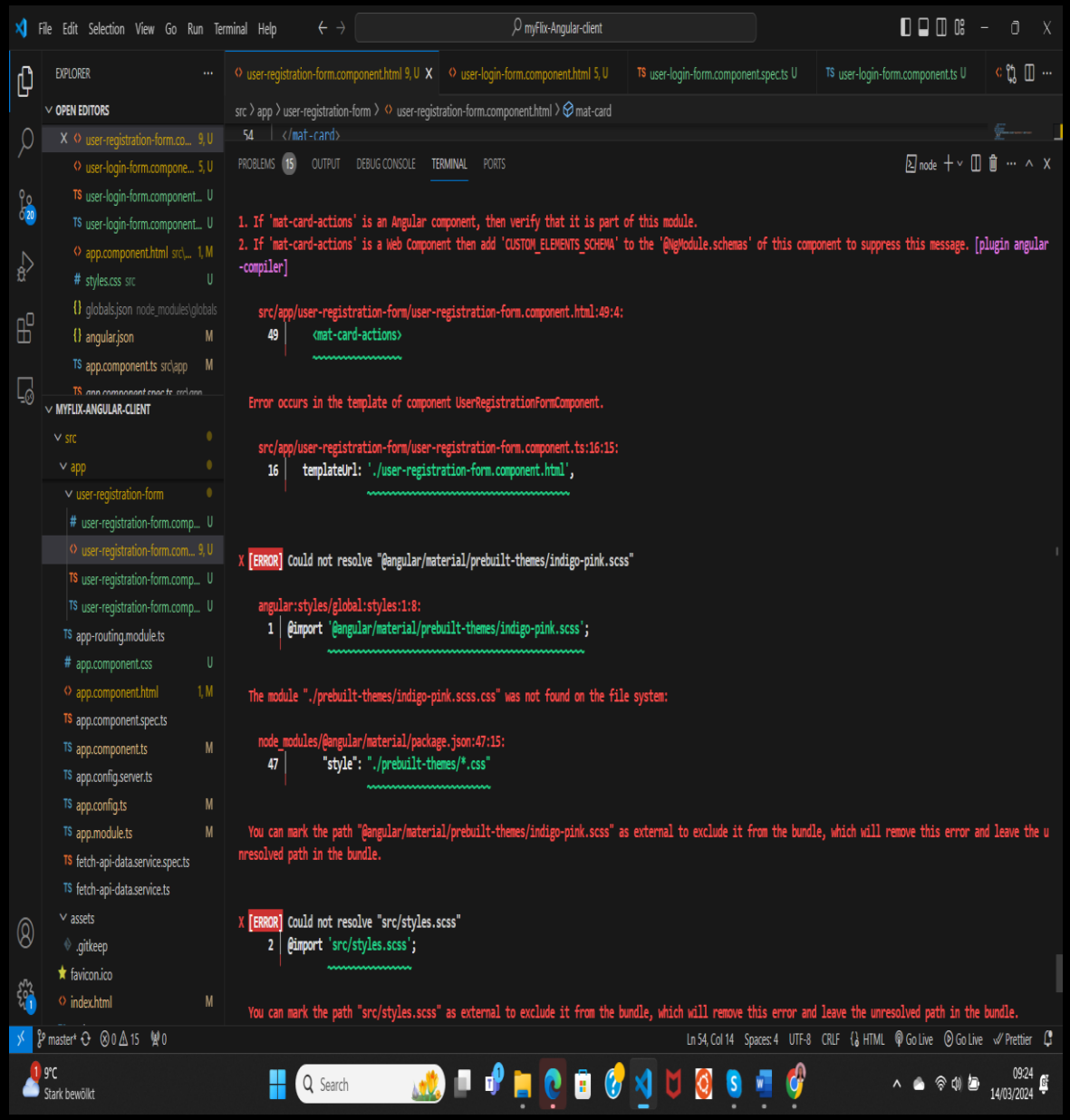


- The routes define the actual endpoints so, through the Express methods, I was able to define the URLs performing the various operations on the data.
- Each URL sent as HTTP request invokes a specified HTTP method to execute CRUD (Create, Read, Update, Delete) operations on the data. The HTTP response sent back by the web server contains either the resource itself or a message about the operations that were applied to the resource.
- To implement the design for the frontend, which I previously designed myself using React Bootstrap to give a skeleton to the interface and the cards and then I customized it with CSS flexbox and colorful gradients.
- All the requests need to be authenticated, so that the API can allow the client to receive the response it is requiring. To do that, I added a bearer token to the header of each API request, where the token is returned by the server after authenticating the user during the login phase, and then stored locally



- **Server-Side**

- I completed the server-side of the myFlix application, including the server, business logic, and business layers using Node.js, JavaScript, MongoDB, and Express .
- It consists of a well-designed REST API . The REST API is accessed via commonly used HTTP methods like GET and POST. Similar methods (CRUD) are used to retrieve data from the database and store that data in a non-relational way.
  To test the API, I used Postman.



Conclusion

- **Challenges**

The greatest challenges I had as a beginner was the issue of indentation and syntax errors which are hard to spot and fix, making me to spend a great amount of time trying to fulfilling YAML requirements for formatting.

I was able to conquer this challenges through constant practicing and with the help of my mentor and tutor.

- **Duration**

The duration for the development of the myFlix application was 3months , which was due to syntax errors , though at the end of the project it was worth it.

- **Credits**

Tutor: Terver Aosu

Mentor: Renish Bhaskaran

App Documentation https://github.com/ossyogwua/movie-api/blob/main/public/documentation.html

app https://myflix-922o.onrender.com

: https://github.com/ossyogwua/movie-api