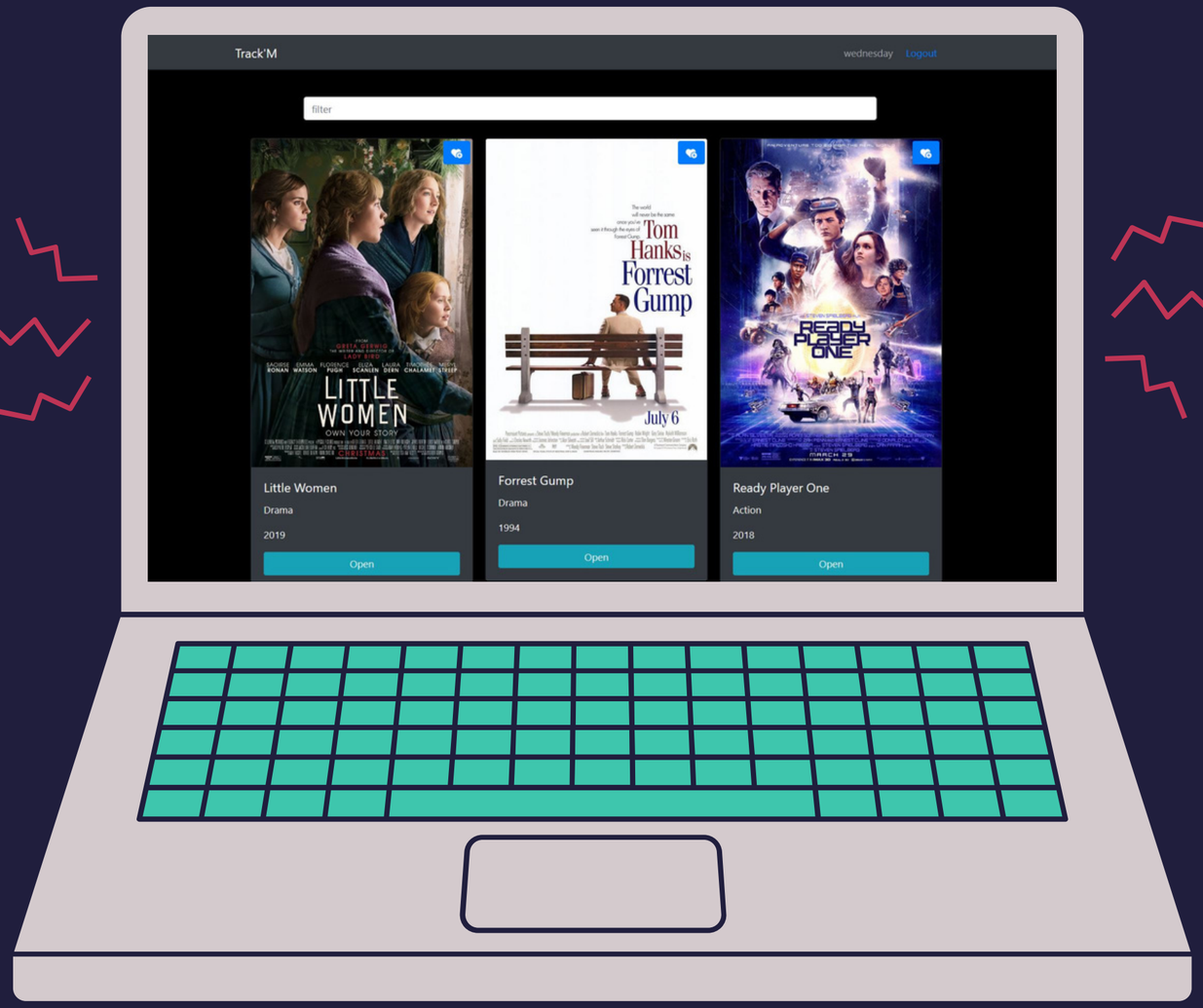
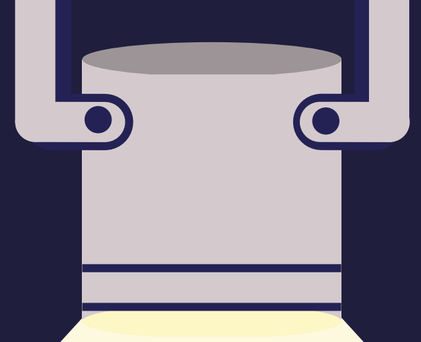


Track'M Case Study

A Full-Stack Project by Amelie Berry



Overview

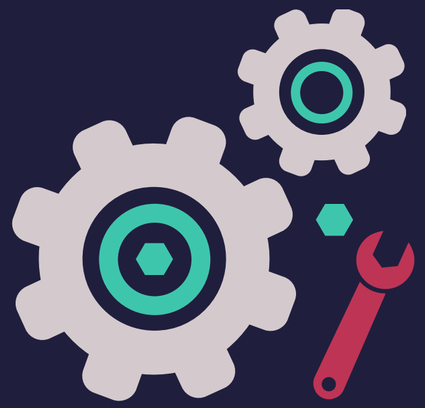
Track’M is a movie tracking app built with the MERN stack. The application's users can create an account, access information about different movies and save them to their list of favorites. Users can also update their personal information, and delete their account.

Purpose & Context

The purpose of this project was to demonstrate mastery of Full-Stack (MERN) development with JavaScript as part of my CareerFoundry development course.

Objectives

1. Build the complete server-side of a web application, including the server, business logic, and business layers to ensure that user requests can be processed, and all necessary data can be stored.
2. Build the client-side of a web application with React that includes several interface views and handles data through the endpoints defined in the first objective.



Tools

- MongoDB
- Mongoose
- Express
- NodeJs
- Postman
- Heroku
- render
- React
- Parcel
- Axios
- Bootstrap
- Redux

Team

Developer: Amelie Berry

Tutor: Bless Darah

Mentor: Ramadhan Aheebwa

Time

Total: 32 days

Backend: 12 days

Frontend: 20 days

The Backend

Github Repo

Live API

First, I built the server-side component which consists of a REST API designed using Node.js and Express, with URL endpoints corresponding to the data operations that I tested using Postman.

CRUD methods were used to retrieve data from the database and store that data in a non-relational way using MongoDB, and the business logic was modeled with Mongoose.

The API is hosted online and can be accessed via HTTP methods. Basic HTTP authentication and JWT authentication were also implemented to authenticate and authorize users.

The Frontend

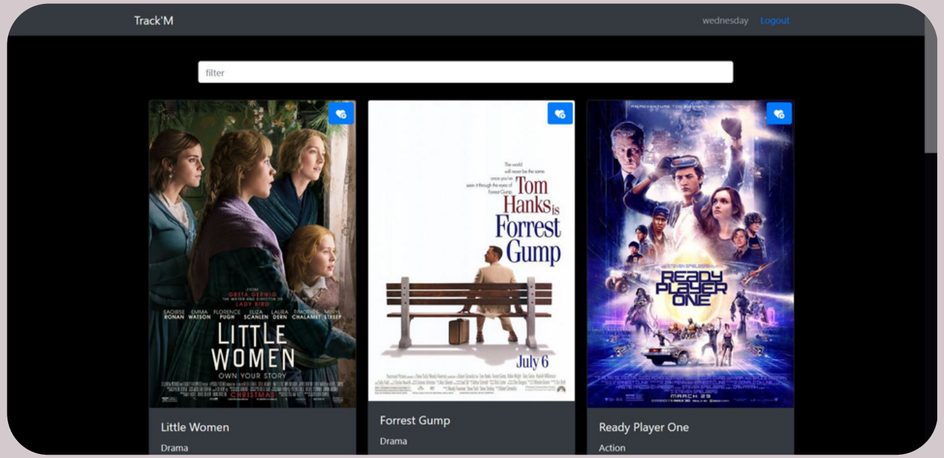
Once the API completed, I used React 18 with Parcel to build a single-page, responsive app that includes several interface views handling data through the previously defined REST API endpoints.

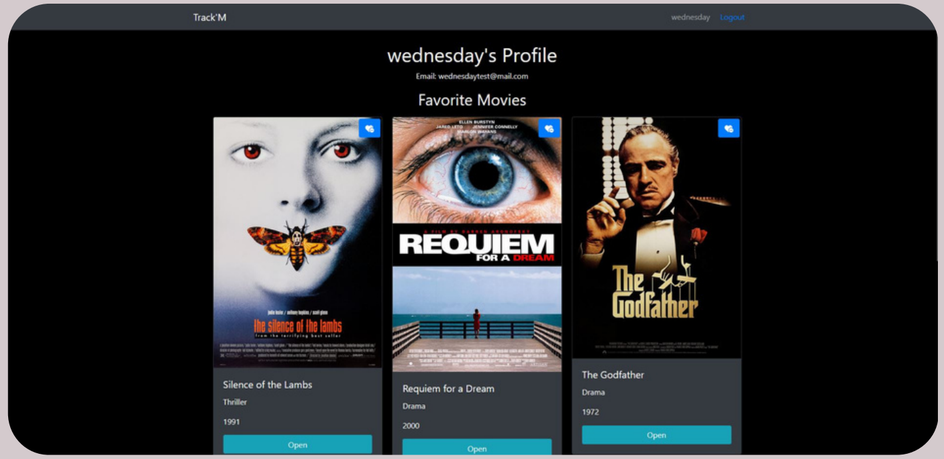
I then created the components that allow users to register or login to be able to view or interact with the list of movie cards, which hold more components. I also created the navigation component to allow users to navigate to the profile view, and finally, a favorite toggle button.

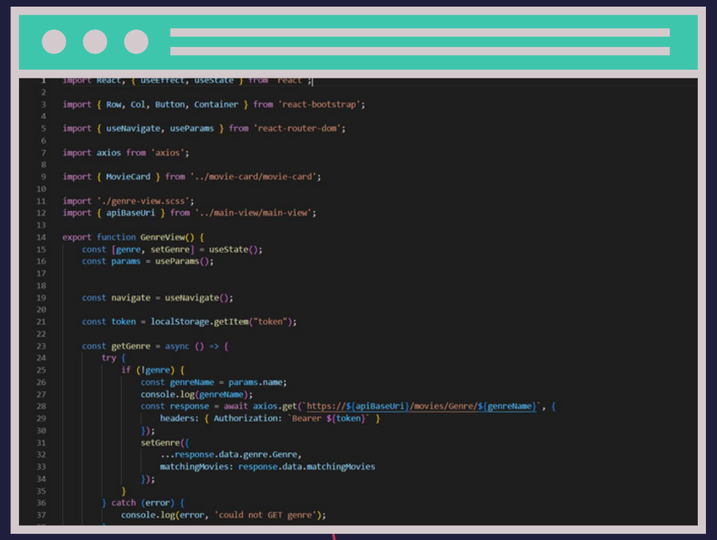
For layout and styling of the app I used React Bootstrap, then routed my views using React Router. As a final step, I added Redux for better state management and scalability before hosting the project on Netlify.

Github Repo

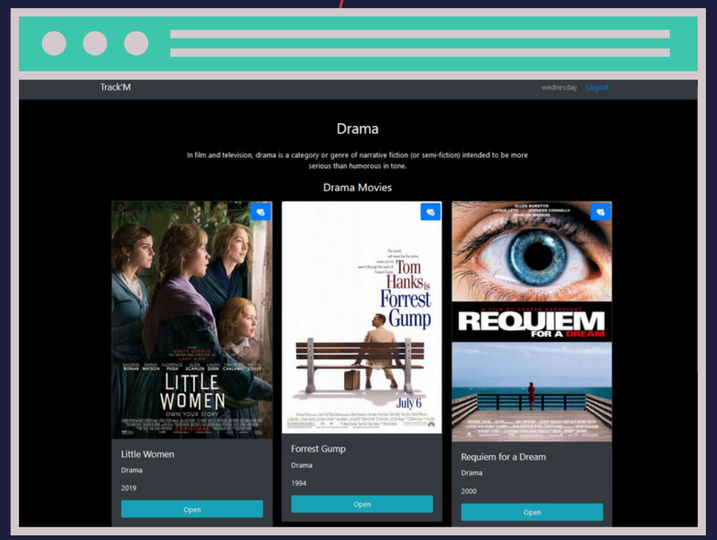
Live App







```
import React, { useEffect, useState } from 'react';

import { Row, Col, Button, Container } from 'react-bootstrap';

import { useNavigate, useParams } from 'react-router-dom';

import axios from 'axios';

import { MovieCard } from '../movie-card/movie-card';

import './genre-view.scss';
import { apiBaseUri } from '../main-view/main-view';

export function GenreView() {
    const [genre, setGenre] = useState();
    const params = useParams();


    const navigate = useNavigate();

    const token = localStorage.getItem("token");

    const getGenre = async () => {
        try {
            if (!genre) {
                const genreName = params.name;
                console.log(genreName);
                const response = await axios.get(`https://${apiBaseUri}/movies/Genre/${genreName}`, {
                    headers: { Authorization: `Bearer ${token}` }
                });
                setGenre({
                    ...response.data.genre.Genre,
                    matchingMovies: response.data.matchingMovies
                });
            }
        } catch (error) {
            console.log(error, 'could not GET genre');
```



Challenges

The first challenge I faced was with Parcel, beginning with hot-reload not working as expected, then running into caching and networking issues. After a quick Google search I switched from using WSL2 to Powershell and the above issues got resolved.

Once I was ready to implement routing to my nearly completed app, I installed the latest version of React Router and noticed differences between the older version (which was being taught in the course material) and this one. The newer version uses hooks which were not compatible with my existing class components, which meant I could either downgrade to React Router v5 to follow along with the exercise or switch to using function components. I chose the latter and although it took more time and effort, I got to learn more about hooks in the process.

Retrospective

Looking Back

This was my first full-stack build, I had therefore allocated extra time for this project and could freely experiment without going over my deadline. Doing that allowed me to spend time getting used to the command line, play around with the database, and take time understanding the (very) numerous errors. Errors that I am grateful for, as they encouraged me to dig into my code and better understand the inner-workings of React.

Looking Forward

I wrote a similar app with TypeScript and found it incredibly helpful in reducing the errors I was facing while building the React app with JavaScript. For future projects I would therefore like to get more familiar with TypeScript. As for the React app, I would like to learn context API so that I can use it instead of Redux for State Management. Last but not least, I will be dedicating time to re-styling the application.