FLICKS picks

A CASE STUDY

PROJECT OVERVIEW

This project was part of my curriculum in Full-Stack Web Development at CareerFoundry. This was my first project of such a magnitude, diving for the first time into backend development and then into the frontend.

FlickPicks is a (small) movie library that gives users information about different movies: movie title, release year, genre, director(s) and actors.

Users have to register and login to access the movie library.
Once logged in, the user can access the library content and:
- add/remove movies to/from their list of favorite movies
- edit their user information
- delete their account

This case study will take you through the steps that led to the finished project.

ROLE

designed and coded by Emeline (me)

TECH STACK

MongoDB, Express, React, Node.js

TIMEFRAME

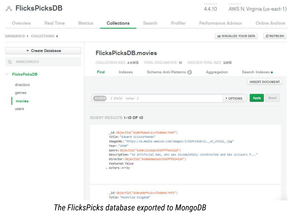
Frontend: 3 weeks
Backend: 3 weeks

HOW IT WENT

THE BACKEND

I manually entered the data using the terminal (Windows PowerShell). This was new to me and very intimidating at first. It was also a repetitive and lengthy process. But the upside is that after being done with movie #4 out of 10, I started having a better flow and was less afraid to make mistakes.

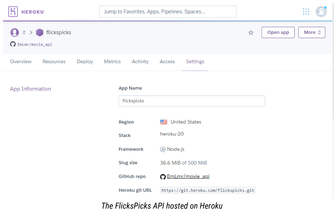
The FlicksPicks database in Windows PowerShell



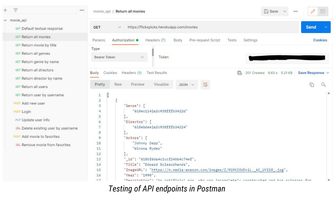
The FlicksPicks database exported to MongoDB

Once all data had been entered into the database it was only available on my machine, so the next step was to make it available online. The data transfer was made using Mongoose, an Object Data Modeling (ODM) library, and the data is put on MongoDB, which can be described as an online server.

The penultimate step of server-side development was to host my database on Heroku. Heroku is hosting all the data, and now this blob of data is a fully-fledged API, available for anyone to make HTTP requests to and get a response in return, making pieces of data available. I also created a basic documentation page detailing how HTTP requests should be formatted, and what the returned responses would look like.



The FlicksPicks API hosted on Heroku

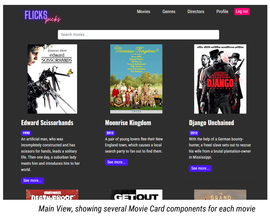


Testing of API endpoints in Postman

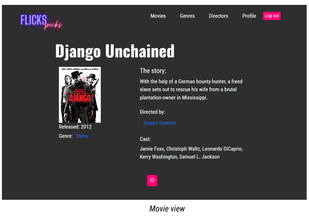
The last step of backend development was to lead some tests. The testing is made on my API endpoints, which are basically the URLs that allow access to specific bits of data. To handle testing, I used Postman. Once again, I felt a bit overwhelmed by this new tool, but after using some small test data and slowly getting to understand how it all works, the testing of my bigger movie API endpoints got clearer. A test was created for each type of HTTP request: GET to get data from the database, POST to add data to the database, PUT to modify data in the database and DELETE to delete data in the database.

THE FRONTEND

The client-side of the API was built with React and React-Redux. These libraries rely on the principle of creating a component for each view displayed in the app: a Login and Registration view, a Main View showing all the movies in the library, a Movie Card displaying a single movie in the Movie View, a Movie View displaying more details about the movie, and within this component are links to the specific Movie Genre and Director, giving more details on each.

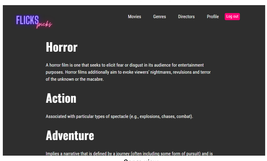


Main View, showing several Movie Card components for each movie

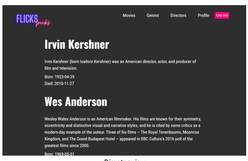


Movie view

On top of this, a Genre View and a Director View will respectively display all the genres and directors available in the database.

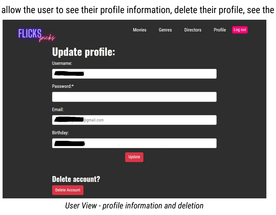


Genre view

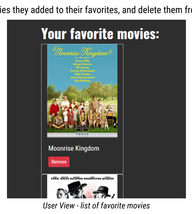


Director view

Finally, a User View will allow the user to see their profile information, delete their profile, see the movies they added to their favorites, and delete them from their list of favorites.



User View - profile information and deletion



User View - list of favorite movies

Yes, this is a lot. Remember when I wrote that this was my first project of such a magnitude?

RETROSPECTIVE

Oh, did I struggle!
JavaScript and React were still very fresh to me, and having to deal with so many components for the first time had me struggle more than once, especially towards the end of the project, sometimes spending days on a single piece of code.
I managed to get something positive out of this though, as I got more comfortable with asking for help, but I also learned that I should be better organized. Being somewhat organized is ok for small, simple projects, but when it comes to bigger, more complex projects such as this one, chaos will eventually be around the corner.

This project was a rollercoaster of emotions but I grew more confident about my coding skills, and better organized. I also discovered that server-side programming is not as complicated as I thought it would be and would like to take on more full-stack projects to get even more comfortable working with the backend and the frontend together.
In the end I was happy to create a working app that also looked good.

However, I think I could improve this app some more:

- Due to time constraints, I had to rush towards the end of the project, and focus on making things work and not so much on the looks of it: I will have to improve the looks of the list of favorites movies in the User View.
- Add movie ratings to my database and app. This was a "bonus feature" for the project.
- Add suggestions/recommendations of other movies with the same genre or director.