Distributed Systems: Webservices

Axel De Leeuw

April 2025

1 Introduction

This is an assignment for the Distributed Systems course of the Bachelor Computer Science at the University of Antwerp. The goal was to develop a restful API that consumes data from 2 other API's: The Movie Database (TMDB) and Quickchart, here are the links to both of these sites:

- TMDB
- Quickchart

This project is also available on Github: Axelmans/WebServices.
Plagiarism on this project is strictly forbidden and will be prosecuted!

2 Mandatory Parts

2.1 RESTful Principles

2.1.1 Uniformity

This is realised by:

- Clear routes (`/movies`, `/movies/favourites`, ...).
- Usage of standard `HTTP`-methods: `GET`, `POST`.
- General return format is `JSON`.

2.1.2 Separation of Concerns

The frontend consumes the API without knowing any implementation details.

2.1.3 Statelessness

All requests contain the necessary parameters and headers for processing.

2.1.4 Caching

Was an optional and possible extension to the project, but not implemented.

2.1.5 Layered System

Not applicable here, but could be realised with `Nginx`.

2.1.6 Code on Demand

Optional principle, not applicable on this service but could be added if needed.

2.2 Efficiency

Python requests are not always reliable when it comes to speedy performance, this was especially detrimental in the `handle_genres_runtime` helper function that iterates over a list of movies and executes a GET request for each one.
The speed of this was improved by using threading instead of iteration.

Even now, the loading times for movie lists can vary. This could be improved by introducing a caching mechanism, for which Flask has useful libraries.

2.3 Fault Tolerance

The fault tolerance in this project is limited. Each request does take into account the possibility of a bad response, using `raise_for_status()`.
The only specific case is that the `Popular` resource throws a bad request if the input value for the parameter n is invalid (i.e. $n \notin [1, 20]$).
An idea could be to set n to 1 or 20 if the input value is out of range.

3 Extension: Frontend

3.1 Motivation

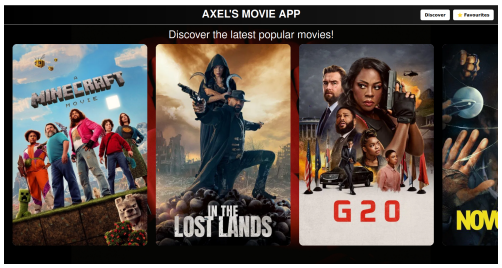
I chose to extend the project with a frontend because of the following reasons:

- Make the service more tangible.
- Wanting to improve my frontend development skills.

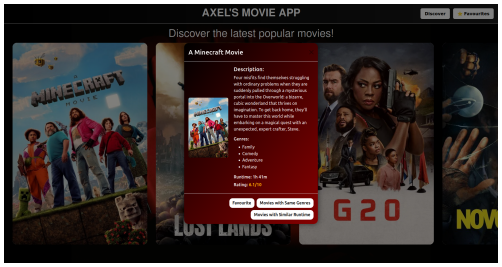
This was realised using React Typescript, just like our Bachelor Assignment.

3.2 Features

Upon startup, the user is presented with a list of popular movies:



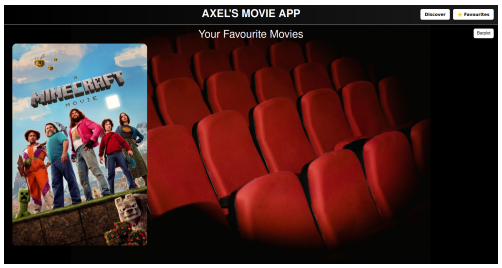
The list can be scrolled horizontally using the scroll-bar at the bottom.
Each movie can be clicked on, and if so, a card appears:



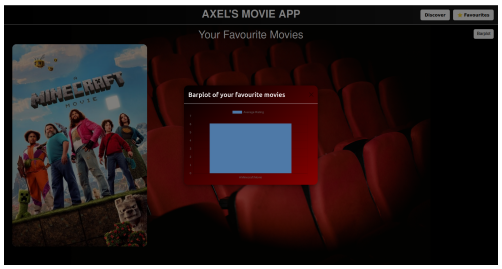
This card displays information about the movie and allows the user to:

1. Add/Remove the movie to/from their favourites.
2. Discover movies with the same genres as this movie.
3. Discover movies with a similar runtime as this movie.

The (un)favourite button also automatically closes the card, the other 2 buttons redirect the user to an appropriate page akin to the regular movie list.
Additionally, each page has a border at the top with a "favourites" button, which redirects the user to a page listing their favourite movies:



This page also contains a button that allows the user to generate a barplot of their favourite movies, it is also displayed in a card in the middle of the screen:



If the user did not choose favourites yet, the page will notify them, and display another "discover" button that redirects back to the list of popular movies.
In this case, barplots will also be disabled due to a lack of data.
Favourites are stored locally so these can be maintained between website-visits.

3.3 Possible Improvements

There are 4 obvious things that could be improved upon in my code:

1. The navigations only work because of a forced refresh in `Base.tsx`, however they should be able to work without it. I've only noticed this issue close to the project deadline, and lacked the insights to properly fix it.

2. My context could be extended and be used more, an idea could be to store a movies array inside it that all the components could access. In addition, the session sometimes throws an exception, stating it is called outside its provider even though it isn't, clicking it away does not break the site.

3. Favourites are also stored locally, but if the user ever clears their browser cache the local storage will not match the actual favourites anymore. This handling should be improved so they always match, this probably requires a change in how the session initializes the favourites list.

4. The colours of the barplot don't match well with those of the site.

I'm definitely planning to improve my React skills even further in the future.

3.4 Reflection

The goal of the extension was to learn, so I'll gladly share what I learned:

- React has a plethora of useful components, always check whether the component you need has already been created before you create it yourself.
- Contexts in React are useful in case multiple components manipulate the same data, which in this case was useful with handling favourites.
  Both the Favourites and MovieCard components use this context.
- Local Storage facilitates keeping certain data between visits.
- Making your website look beautiful is an art on its own, that's why...

I would like to give a huge thank you to Asumi, my online friend, for helping me with the visual design of the frontend, couldn't have done it without her.

4 Time Spent

I didn't keep exact numbers, but the frontend took way longer than the API. This however, is because I overdid it, mainly because I had a lot of fun brainstorming and creating an as user-friendly as possible website, that is also visually pleasing to the eyes, and of course to gather frontend-development experience.