Web Technologies Individual Report

University of Southern Denmark, Odense
Faculty of Engineering

MovielibLaravel: Movie Carousel Extension

Jonas Mathias Lissner (jolis21@student.sdu.dk)

Course code: T510048101-1-E23

Project period: E23

Supervisor: Miguel Enrique Campusano Araya

Github Repository: MovielibLaravel-MovieCarousel

1 Introduction

The group project provided a foundation for this individual extension.

Background: The group project aimed to create a web application that provides a foundamental movie library for the users to interact with, such as finding new movies to watch, commenting on movies, rate them, and/or add them to their watchlist.

Motivation: The motivation behind this extension aims to elevate the user experience by introducing a dynamic and engaging feature, that showcases the currently most popular movies.

Project: The individual extension aims to create a movie carousel implementation in form of an API that can be used in various projects.

Contributions: The contributions of this project highlights the development of a dynamic movie carousel, demonstrating its use in our web appliaction, its technicalities, and its potential applications in other projects.

2 Reflections

2.1 Front-end

I think the front-end is very well designed, responsive and easy to navigate, and it feels like its developed with the UX in mind due to its simplicity and minimalism. As I have a background in web development, I hadn't really any noticeable struggles with the front-end, my biggest struggle was to learn Laravel and utilize it effectively. The source code readability is generally clean and well structured, but I think additional comments and documentation would enhance the readability even further.

2.2 Resource management

Even though our project's resource management relies on utilizing an external public API for the movies, I think the implementation is as good as it can be, given the time we had and the scope of our project. A positive aspect to this is the seperation of concerns, as the resource management is handled by the external API, and our project only needs to handle the data that is returned from the API, allowing for modurlarity. Our internal resource management is also well structured, as we have a seperate controller for each resource, and a model for each resource, also allowing for modurlarity within the backend.

2.3 Authentication authorization

By utilizing Laravel's built-in authentication and authorization, we were able to implement a secure and robust authentication and authorization system.

3 Individual Extension

Motivation: The motivation behind this extension is to enhance the UX by introducing a dynamic and enganging feature on the front-page. The extension aims to capture the user's interest by stimulating their curiosity about the showcased movies.

Technical description: The technical description of this extension is a movie carousel, that showcases the currently most popular movies. The carousel is implemented as an API, that easily can be used in other projects. The API is developed with modularity and scalability in mind as the carousel scales to the boundaries of the container that it's within, and dynamically adjust to the amount of movies it showcases. Furthermore, it's relatively easy to construct a new movie carousel, as it only requires a few lines of code and 3 parameters: the container it should be within, the list of movie data, and an optional auto-scroll time.

Reflections: I think the overall implementation of the movie carousel is very well done, due to its intuitive, dynamic and scalable design. As I'm not very experienced with JavaScript and Laravel, I had to learn a lot of new techniques and syntax. Both it's design and implementation went through several iterations until I was pleased with the result. Overall, I ended being very pleased with the result as it adds a very needed and exciting visual component to the frontpage, and feel like I have achieved what I envisioned for the carousel and learned a lot during the process. However, I think the carousel could be improved further by adding additional API endpoints,

as this would allow for more flexibility and customization, such as the ability to control the scroll speed, and the ability to control the amount of movies it showcases. It could also be improved by allowing for more customization, such as the design/color scheme, and adding flexible components within the carousel itself.

4 Security Reflections

Group project:

- Sensitive data: We secured the sensitive data by utilizing Laravel's built-in authentication system, which securily hashes and salts the passwords, and stores them, along with other person sensitive data, in the database that only the server has access to.

- Database access: The database access is secured by utilizing environment variables, which is locally stored on the server to prevent unauthorized access. The data within is only accessible through our internal API, with strict restrictions on what data can be processed and therefore accessed.

- Authentication and Authorization: The authentication verification is done by comparing the hashed password with the user input, and the authorization is done by checking the user's role and verifying their access.

- Possible Attacks: I think we took security measures very serious, and countered the most common attacks, like SQL injection, brute-force attacks, and data breaches.

Individual extension: The Movie Carousel doesn't introduce any new vulnarabilities, as it's primarily a front-end components that only handles data that is already accessible through the public TMDB API.

5 Performance Scalability Reflections

The most noticeable perfomance issue we encountered was the response time of the TMDB API, as it sometimes takes several seconds to respond. This design choice was inevitable as we are relying on an external public API to get our movie data, with limited results in terms of endpoints. We tried to counter this by using the most efficient endpoint, depending on the result we wanted. But it had its limitations, as we had the choice of either fetching a whole page of movies, or a single one. Caching the data may have helped lessen the issue, but as it would require a lot of storage and was out of scope for this project, we decided not to. However, the individual extension doesn't introduce any new performance issue, as it, before the Movie Carousel extension, still had to fetch the same amount of data from the TMDB API. It's also very scaleable, as it can showcase anything the user wants, depending on the list of movie data it receives. Therefore, I think the extension is a very good trade off, as it adds a lot of UX value to the frontpage.

6 Conclusion

The individual extension aims to create a Movie Carousel implementation in form of an API that can be used in various projects to enhance the user experience upon entering the website. The extension is developed with modularity and scalability in mind, as it scales to the boundaries of the container that it's within, and dynamically adjust to the amount of movies it showcases. The construction of a new movie carousel is simple, as it only requires a few lines of code and 3 parameters: its container, list of movie data, and an optional auto-scroll time. The extension doesn't introduce any new security vulnarabilities, as it's a component that only handles already requested data from TMDB. A valued addition for improving the carousel in the future would be adding additional API endpoints for customization that would enhance the flexibility of the component, such as the ability to alter the CSS styling around and within the carousel.