Web Technologies Project Report

Group 4

28/12/2023

1 Introduction

Our group has developed a movie library that acts as an independent service in which users can get an overview of a broad range of existing movies. The movie library enables users to create a watchlist and keep an overview of the added movies. Further, the web service features a rating system allowing users to rate movies with 1-5 stars. Additionally, a public comment section is available. Thus, allowing users to make comments, which can be read by others.
The movie library differentiates its watchlist system from existing streaming services by providing access to a wider collection of movies. Thereby, creating an online environment in which users can create a collected overview.

2 Front-end

We have developed the front-end of the application using HTML, CSS, and JavaScript. However, Blade templates have also been used since we are working with Laravel which is a framework that incorporates them. Blade allows you to use PHP within your HTML, which can be very handy in order to display data from APIs, databases or other sources.

HTML 5:
HTML 5 is the most recent version of HTML, This version introduces some new semantic tags. Semantic tags can enhance accessibility and SEO. Also, they make it easier for developers to understand page structure as well as the purpose of each section. For example, <header>, <nav>, <main>, and <footer> among others have been used to define our pages' header, navigation bar, and footer.

In our project, we tried to utilize these semantic tags as best as possible in order to identify different sections on the page. These include movie information and comments when viewing a specific movie on its own page. This makes it easy to understand each section's purpose by giving an overview of how each section has been divided into. Using more descriptive markup such as the <section> instead of generic tags provides more meaningful semantics.

CSS:
CSS is used for setting up the style and visuals of the HTML page elements, and these define features such as the layout, colors, and fonts as well as the overall page structure. It improves the visual appearance and can also make it user-friendly with things like animations on hover, transitions or a number of other CSS3 capabilities that may indicate to a user that an element can be clicked.

In our application we did not want to use the default styles that the browser has, so we used CSS to style the page and application to what we wanted. However, CSS can become difficult to navigate through and maintain. The reason behind this is, that you never know if a style has been used somewhere else in the application. This can cause inadvertent changes in other sections of the app which are often hard to debug. For this reason especially being a group project, which will most of the time make it even more difficult, we had different CSS files for different pages unless they were common components across pages. So, as far as that problem was concerned we also made sure that descriptive class names were used only along with targeting IDs and classes that were specific to the page. This way others would remain unchanged by mistake. There is a downside to doing this, however, as it can lead to a lot of duplicate code, which can also make it hard to maintain, as it is likely that at some point, there will be unused styles in those CSS files. We

weighed the pros and cons of this approach and decided that it was the best approach for our project.

JavaScript:
JavaScript is a programming language that helps make web pages interactive and dynamic. Moreover, it allows for adding or removing elements from the page, changing the styles of elements as well as making API calls. Since Laravel is a server-side application most of our data are fetched from the server on page load. But we also have some dynamic elements on the page such as the movies' comment section, which can be fetched from the server when a user clicks on the comment button by making an API call to the server, the watchlist button for correctly displaying the current state of the movies' existence in a user's watchlist, and rating system used to rate movies from 1 to 5 stars.

The way that JavaScript is incorporated into HTML is by using the <script> tag. It is not only used for including JavaScript files but can also be used for writing JavaScript code directly in the HTML file. We have gone with the former approach, because it is less complex to maintain and debug since you can easily see which JavaScript files are included on the page. Nevertheless, we have also utilized the latter option, but only for minor scripts that are unique to that specific page.

3 Resource Management

In connection with the creation of the movie library, we have chosen to benefit from resource management. We are using two types of resource management: One for the TMDB API and one for the database. The TMDB API is used for fetching movie data, while the database is used for storing users, their watchlists, comments, and ratings.
A technical description of the resource management is provided in the following subsections.
API Routes: The routes for the API are defined in the `routes/api.php` file and define how the system interacts with the API to fetch the movie data. The TMDB API routes are used for fetching the movie data, such as the movie title, description, poster, cast, and so on, to display on various occasions within the website. Where our own internal API is used for various movie specific CRUD operations within the application. The internal and external API use HTTP GET, POST and DELETE requests to fetch a more specific response, and is capable of fetching:

TMDB API
GET Sorted list of movies for a specific page:

Endpoint: '/popular_movies/{page}/{sort}'

Parameters: *page* and *sort*

Description: This endpoint is used for fetching a list of movies, sorted by a specific parameter, for a specific page. The endpoint returns a list of movies, hereunder the movie title, description, poster, cast, etc.

GET a specific movie:

Endpoint: '/movie/{id}'

Parameters: *id*

Description: This endpoint is used for fetching a specific movie and its data. The endpoint returns a specific movie, hereunder the movie title, description, poster, cast information, etc.

GET All genres:

Endpoint: '/genres'

Parameters: None

Description: This endpoint is used for fetching all genres. The endpoint returns a list of genres, hereunder the genre id and name.

Internal API
GET comments for a Movie:

Endpoint: '/movie/{id}/comments'

Parameters: *id*

Description: This endpoint is used for fetching all comments for a specific movie. The endpoint returns a list of comments, hereunder the comment id, user id, movie id, time, etc.

POST or DELETE a Movie from watchlist:

Endpoint: '/movie/{id}/watchlist'

Parameters: *id*

Description: This endpoint is used for adding or removing a movie from a user's watchlist.

Database Tables: The database tables are defined in the `database/migrations` folder and define the structure of the database. The database is used for storing the user's watchlist, comments, and ratings within the application.
CRUD Operations: The CRUD operations are defined in the `app/Http/Controllers` folder and define how the system interacts with the database.
The CRUD operations are used for:

- *C*reating a new user, adding to the user's watchlist and adding the user's comments and ratings on movies.
- *R*eading the user's watchlist, comments, and ratings.
- *U*pdating is not used by any functions.
- *D*eleting movies from a user's watchlist.

4 Authentication and Authorization

To help fortify the systems overall security, it is very important that the system knows which users has access to certain things and can identify users that does not have access to the given resource. This is done by implementing authorization and authentication on our system.
Authentication:
Authetication the process of verifying the identity of a user. In our system we have two distinct user categories "Guest" and "User". The "Guest" user is the default user and can only access the home page and the login page. The "User" user is the authenticated user that can access all the pages in the system. The implementation of the authentication is done by verifying the user's identity. The system asks for a username and password when you login. The system then checks if the username and password match the ones stored in the database. If they match, the user is authenticated and can access the system. If the username and password do not match anything in the systems database, the user is not authenticated and can only access a limited set of resources on the page.

Authorization:
When the user is authenticated, the system needs to check if the user is authorized to access the resource, he is trying to access. In our system we have two distinct user categories "Guest" and "User". The "Guest" user is the default user and can only access the home page and the login page. The "User" user is the authenticated user and can access all the pages in the system. The implementation of the authorization is done by checking the user's role. The system checks if the user is a "Guest" or a "User". This is done within the system to ensure that the user is authorized to access the resource he is trying to access. If the guest is trying to access something that he should not be able to access, the system will redirect him to the login page. This will help fortify the security of the system and ensure that only the right users can access the right resources. As a result, the system will be more secure.
To give a better indication of the different user roles we have in our system, we have:

- **Guest:** The default user, can only access the home page and the login page.
- **User:** The authenticated user, can access all the pages in the system.

By giving our website different user roles, we can ensure that the system, is safe against incoming attacks. It also makes it so, that people can

not access information in the system that they should not be able to see without having created a user. It also works the other way around. Only people that are created on the system, can comment on the different movies on the website, while the unauthorized users can only see the different comments. If we were to develop the system further we could add some more user roles with different authorizations. We could for example make an admin user, which has access to everything including the profiles of the different users. With the way we have implemented the authorization and authentication in our system, we can easily scale the different user's roles, without any complications.

5 Conclusion

5.1 Summary

In this project, we have developed a web application that is able to display a wide variety of movies and give a quick overview of the movies' information. The application also provides a way for authorized users to add comments and ratings to the movies, as well as putting them on a watchlist.

The application is built using Laravel, and utilizes a database to store our users, watchlist data, and comments. The movies are fetched from TMDB, using their API.

5.2 Future Work

Future work on the application could include adding more user types as explained in the authorization section, as well as adding better filtration of movies when looking through the list.