CS3219 Final Report

Team 29
Gordon Foo
Ng Shuo Qi, Constance
Chen Yan Jia Jay
Chu Kim Guan

Table of Contents

1. Background and Purpose

With the restrictions due to the Covid-19 pandemic around the world, there has been an increase in people turning to entertainment content platforms such as Netflix over the years. However, our group observes there is a lack of a social platform which users can discuss and share their favourite movies/shows.

Hence, **Moave**, a web application was created with the aim of helping people in Singapore connect with friends over similar interests in movies and television shows as well as to provide a platform for users to find new shows to watch.

2. Description of application

This project serves to provide a social media platform for users to discuss their favourite movies/shows, see what movies/shows their friends are watching, keep track of what shows they are watching or have watched in the past.

3. Contributions

Our team is divided into two subteams, one working on the frontend (FE) components, and the other working on the backend (BE) components.

Name	Technical Contributions	Non-technical Contributions
Kim Guan (FE)	Built UI components on the front-end, integrating them with back-end endpoints. Set up a movie recommendation service and recommendation algorithm.	Spearheaded project idea and in charge of overall product management.
Jay (FE)	Built UI components on the front-end, integrating them with back-end endpoints. Set up changing of password on the frontend.	Documentation and adding details to the report
Gordon (BE)	Backend Restful APIs to handle all Functional Requirements.	Documentation and adding details to the report
Constance (BE)	Planned and created database schema and some sample data for local testing.	Documentation and adding details to the report

4. Deployed Endpoints

Front-End	https://fierce-hamlet-39238.herokuapp.com/
Back-End	https://boiling-citadel-87511.herokuapp.com
Movie Recommender Service	https://moave-frontend.herokuapp.com

5. Requirements

Priority	
High	Necessary for the project's success
Medium	Good to have, enhance user experience
Low	Extra features

5.1. Functional Requirements

Functional Requirements		Priority
User Account Management		
F1.1	The application should allow users to create an account with a unique username and a password.	High
F1.2	The application should allow users to sign into their account using their registered username and password.	High
F1.3	The application should allow users to logout of their account.	High
F1.4	The application should allow users to change their password.	Medium
Search functionalities		
F2.1	The application should allow users to query for a movie by keyword.	High
F2.2	The application should allow users to view movie details. This includes the title of the movie, the movie poster, description, ratings, release date and runtime.	High
F2.3	The application should allow users to search for other users.	High
Social Management		
F3.1	The application should allow users to follow another user.	High
F3.2	The application should allow users to unfollow a user they were following.	High

F3.3	The application should allow users to view the profile of other users. This includes their followers/following, their activities and movie lists.	Medium
F3.4	The application should allow users to view activities of users they are following.	Medium
Group Management		
F4.1	The application should allow users to create groups.	High
F4.2	The application should allow users to search for groups.	High
F4.3	The application should allow users to join groups to discuss common movie interests.	High
F4.4	The application should allow users to create a thread in the group with a title and a description.	High
F4.5	The application should allow users to reply to a thread in the group.	High
F4.6	The application should allow users to view all the groups they have joined.	Medium
Recommendation Functionality		
F5.1	The application should allow users to see similar movies based on the movies that they currently search for.	High
Profile Management		
F6.1	The application should allow users to view their profile page.	High
F6.2	The application should allow users to view the number of followers and their usernames in the profile page.	High
F6.3	The application should allow users to add movies to a default Plan to Watch list.	High
F6.4	The application should allow users to add movies to a default Currently Watching list.	High
F6.5	The application should allow users to add movies to a default Completed list.	High
F6.6	The application should allow users to view the number of users they are following and their usernames in the profile page.	Low

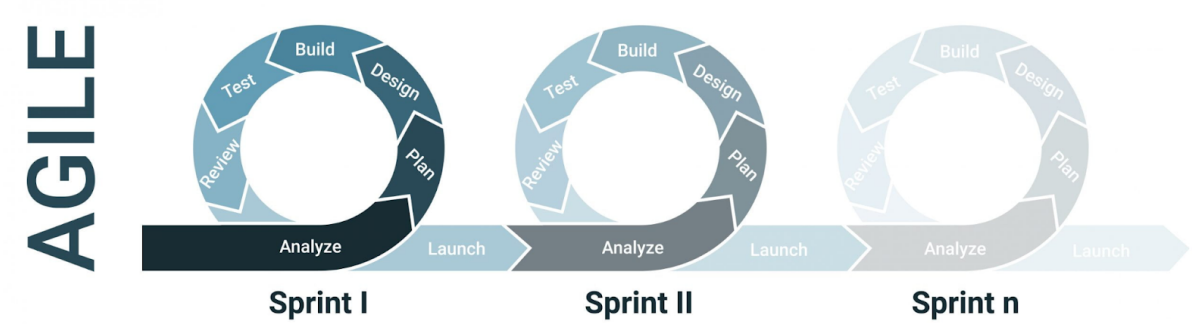
5.2. Non-Functional Requirements

Our team decided to prioritise the performance, security and usability of the application.

Non-Functional Requirements		Reasons for choice	Priority
Performance			
NF 1.1	The application should render movie details in less than 2 seconds.	We need the application to render information fast for a good user experience.	High
NF 1.2	The application should render user profile details in less than 2 seconds.		High
NF 1.3	The application should render recommended movies in less than 5 seconds.		Medium
NF 1.4	The application should render other user profile details in less than 2 seconds.		Medium
NF 1.5	The application should allow users to change their profile details in under 2 seconds.		Low
NF 1.6	The application should allow users to create their own movie list in under 2 seconds.		Low
Security			
NF 2.1	The application should only allow authenticated users to perform changes which will modify the users settings. This includes adding movies to a list, changing password, following/unfollowing a user, creating/joining groups, creating/replying to threads.	To prevent any unwanted changes to a user's settings/profile.	High
NF 2.2	The application should only allow users to login to their own account(s) which they have access to.	To prevent hacking.	High
NF 2.3	The application should only allow users to see the activities of the user they are friends with.	To protect the user's information.	Medium
Usability			
NF 3.1	A new user should be able to search for and follow another user in less than a minute.	The application should be easy to use and understand.	High

6. Development Process

Our team followed the agile methodology for our development. An iterative and depth-first approach was chosen, where each iteration focuses on completing a specific functionality. An iterative approach was chosen to ensure that a working product is available after each sprint, and to allow more flexibility in changing the design and implementation of the project if there are any problems.



Agile Development Process[1]

Our team first laid out all the functional requirements required for the project, which is listed on a collaborative Google sheet. Every Wednesday, we meet online for around an hour to update each other on our progress and provide feedback on each other's work. This meeting also facilitates the incorporation of the work done between all members, and fixing any issues that may appear. Then, a set of requirements will be decided to be implemented for the next iteration. The progress is also updated on the excel sheet so we have a clearer idea of what is done, and what is left to do. This process repeats every week.

	Task	Priority	Done		Iteration
User Management				Remarks	
F1.1	The application should allow users to create an account.	High	Completed	create acc w name, email, pw	Iter 1
F1.2	The application should allow users to sign into their account.	High	Completed	login with email and password	Iter 1
F1.3	The application should allow users to logout of their account.	High	Completed		Iter 1
F1.4	The application should allow users to change their password.	High	Completed		Iter 1

Sample portion of google sheets to track plan for current iteration

Google Sheets was used instead of Github issues as this made it easier to track what was done and allowed more flexibility in writing remarks and other notes compared to Github Issues.

The deployment of the application was done when most of the features were completed.

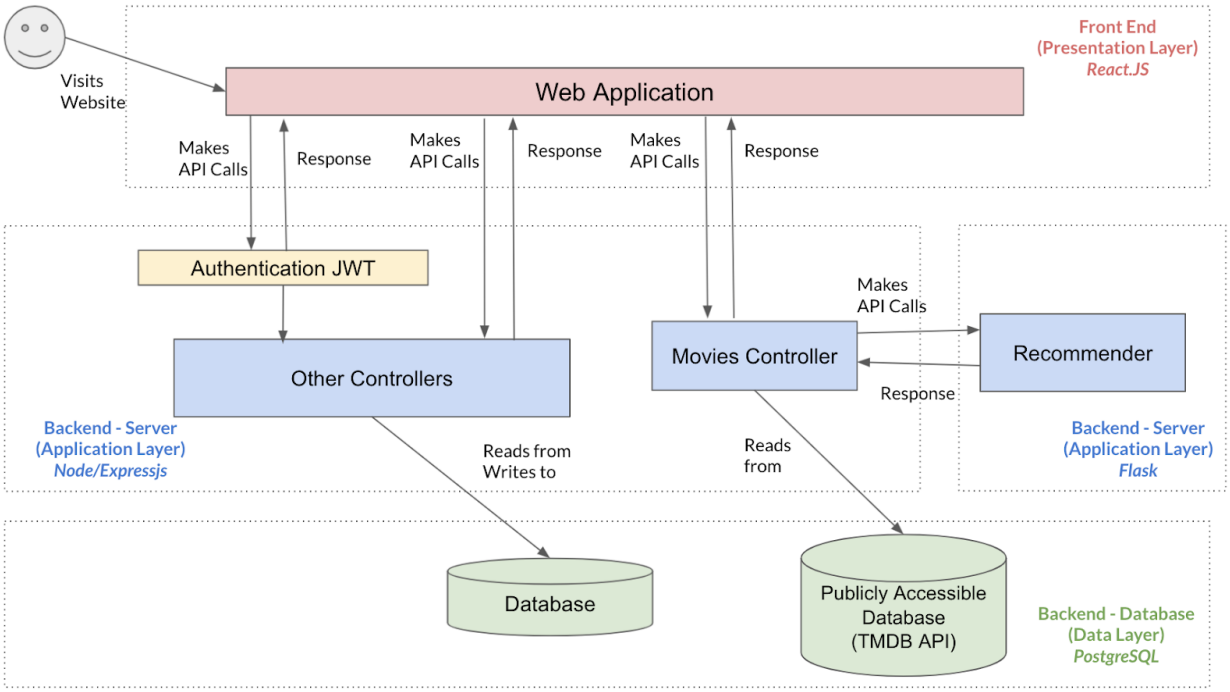
[1] Image from Google

7. Technical stack

	Technology	Rationale
Frontend	React.js	React is a front-end framework that supports reusable components which fits our use case.In addition, it also supports fast rendering of UI components.
Backend	Express.js/Node.js/Flask	Express/Node provides server-side logic for web applications, which is easily built and can be scaled. Flask is used as a lightweight backend server as a service to run Python scripts.
Database	PostgreSQL	PostgreSQL is the choice of database as it supports a relational database and uses mySQL as the primary language.
Deployment	Heroku	Heroku is free to use for deployment purposes and is flexible in being able to deploy front-end, back-end and database.
Project Management Tools	Google Sheet Tracker	Allows collaboration between team members and a clear overview of what has been completed or not completed, facilitating discussions on the set of requirements for a particular iteration.

8. Architectural Design

8.1. Overall Architecture



Overall architecture of application

Our project uses a monolith architecture with a separation between the frontend and backend (which contains the server and database). This architecture was chosen as it is easier to develop and manage, since it is a rather small project.

The project also follows a 3-tier architecture, where the application is split into three layers - presentation layer, application layer and data layer. This architectural style allows each layer to be developed in parallel, and allows changes to be made in separate layers without affecting another. For instance, changes made in the data layer, such as changing the database used will not affect the application layer will not be affected. This makes our application more flexible and adaptable to changes.

To retrieve movies information such as title, description and movie image to be displayed, our project fetches data using The Movie Database's (TMDB)[2] API, which is available for public use. To use the API, an API key is required which is stored in the environment variables of our project.

[2] https://developers.themoviedb.org/3/getting-started/introduction

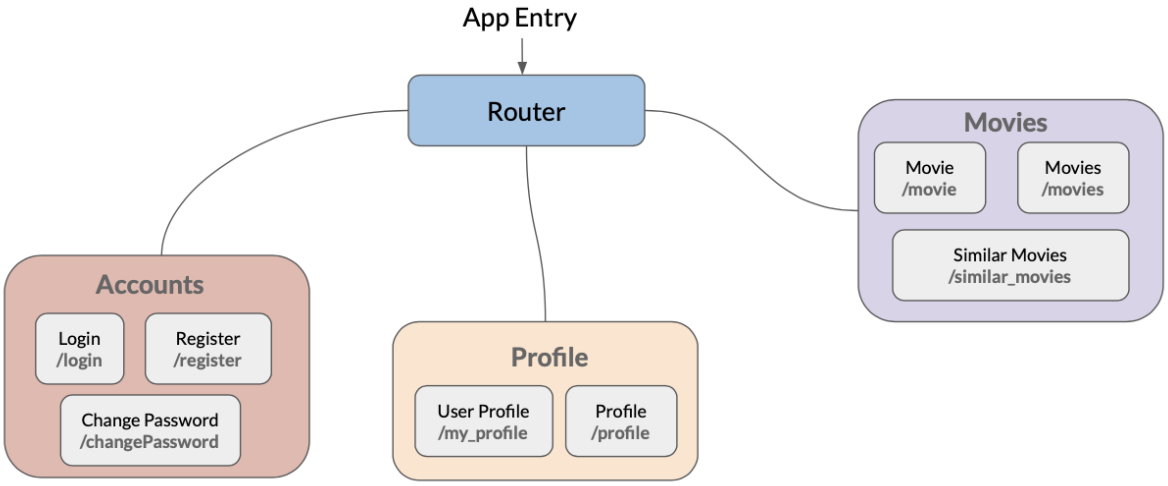
8.2. Frontend

8.2.1. Architecture

The frontend of the web application uses React.js. Different display pages are created using tsx files (TypeScript files written using JSX syntax).

Tech Stack:

1. **React**
2. **Ant-D UI**



Routing of pages

The front-end architecture is based on a Single-Page Application concept. The server will only render the web page once at the start of the application, thereafter, we dynamically generate the pages using routing. Therefore, the front-end uses client-side rendering to generate pages for the user, and there are no static pages generated when building the front-end.

Concerns when designing the Front-End as a Client-Side Rendering Application

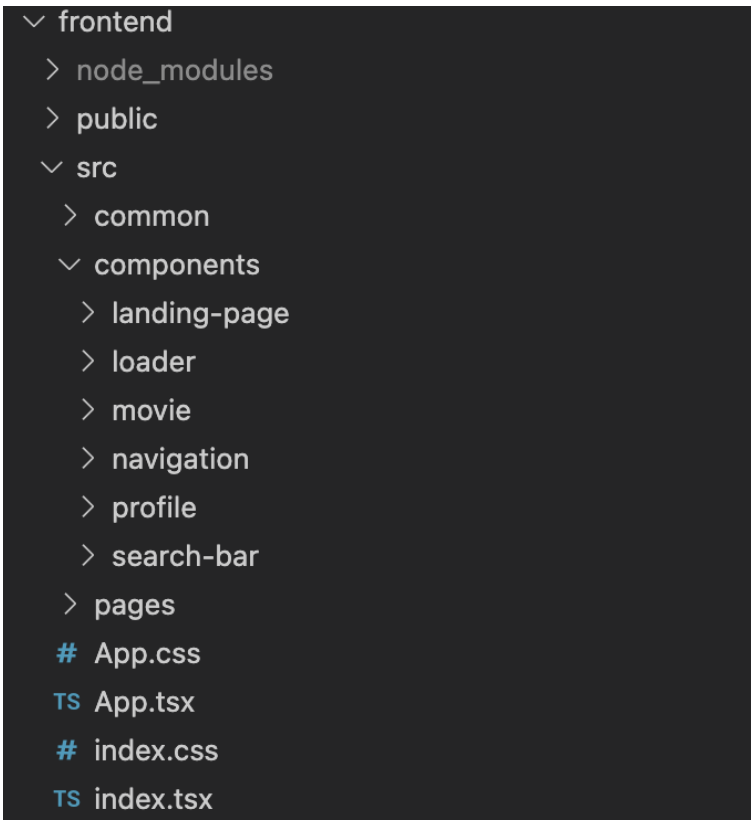
An advantage of using client-side rendering is the high performance of the web application. This is because we will not refresh or re-render the whole page. Whenever we switch between routes, it de-renders the children components and re-renders the new ones to replace them, this saves computational power and bandwidth for the user.

A disadvantage of using mainly client-side rendering is the slow rendering of the web application at the start as it has to load and compile the whole Javascript bundle when it first loads. However, this is of little concern for our use case as our main focus is giving the user a fast experience when transiting between the different pages, thus the initial load time is of little concern in exchange for fast rendering of components. Another issue with client-side rendering is the lack of optimisation for Search Engine Optimisation. The different pages are not indexed on the server, thus Google searches will not be able to return individual pages (only the main page). This is however of little concern to us as we intend to design it based on a Single-Page Application concept,therefore, there is no need to index the page for Google search.

8.2.2. Organisation of code

The front-end is based upon the principle of the composition of components. Each component can compose of other smaller components, where each component can be re-used.The code on the frontend is organised into different folders - common, components and pages. The entry point of the application to the various pages will be the ‘App’ file which determines the pages to be loaded.

This allows components to be reusable across different pages. For instance, the navigation component is required in multiple pages and this would reduce duplicated code.



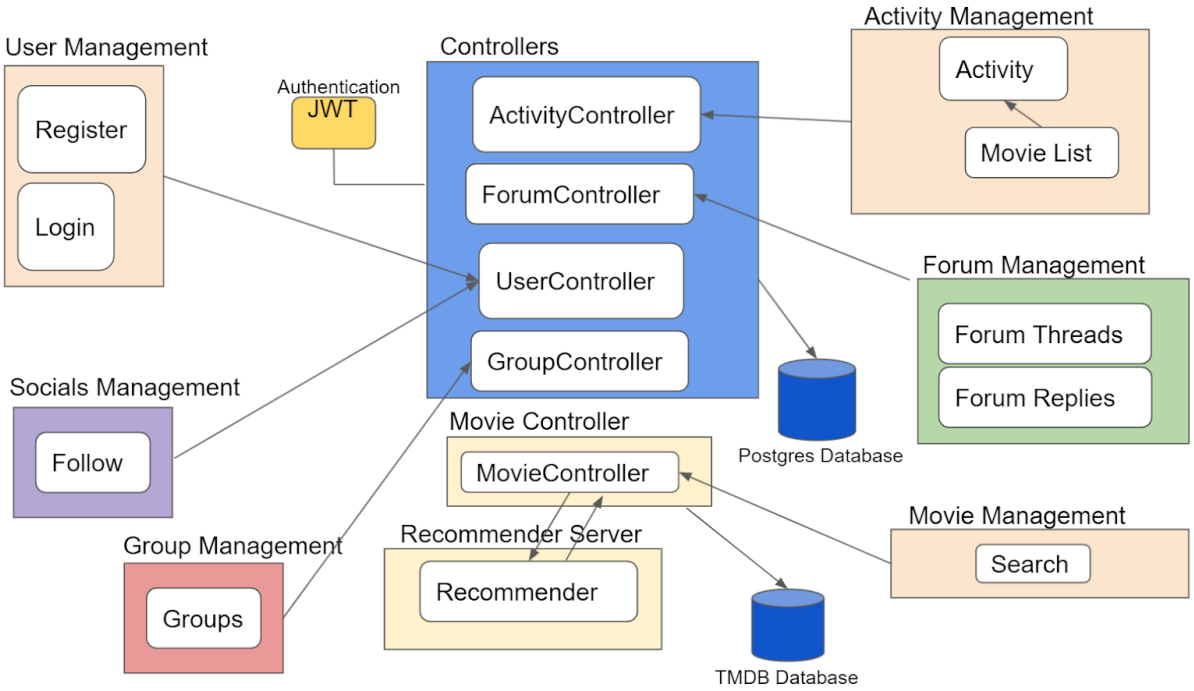
Organisation of code on the frontend

8.3. Backend

8.3.1. Architecture

The backend of the web application uses Express, a node.js framework. The backend mainly contains the controllers that handle the different APIs required that the frontend can call to retrieve or insert information to the database.

Some protected routes and resources require authentication before they can be accessed, using JSON Web Tokens (JWT) as shown in the diagram below. This will be further discussed in Section 8.2.



Backend architecture

The figure above shows the general outline of the backend architecture, giving an overview of how different components interact with the main server of controllers.

The User Management component contains the functions required for new users to create new accounts or existing users to log in. The functions will make API calls to the UserController.

The Social Management component contains functions for users to follow/unfollow each other. These requests are done by making API calls to the UserController.

The Group Management component contains functions for users to create, join and leave groups. These requests are handled by the GroupController.

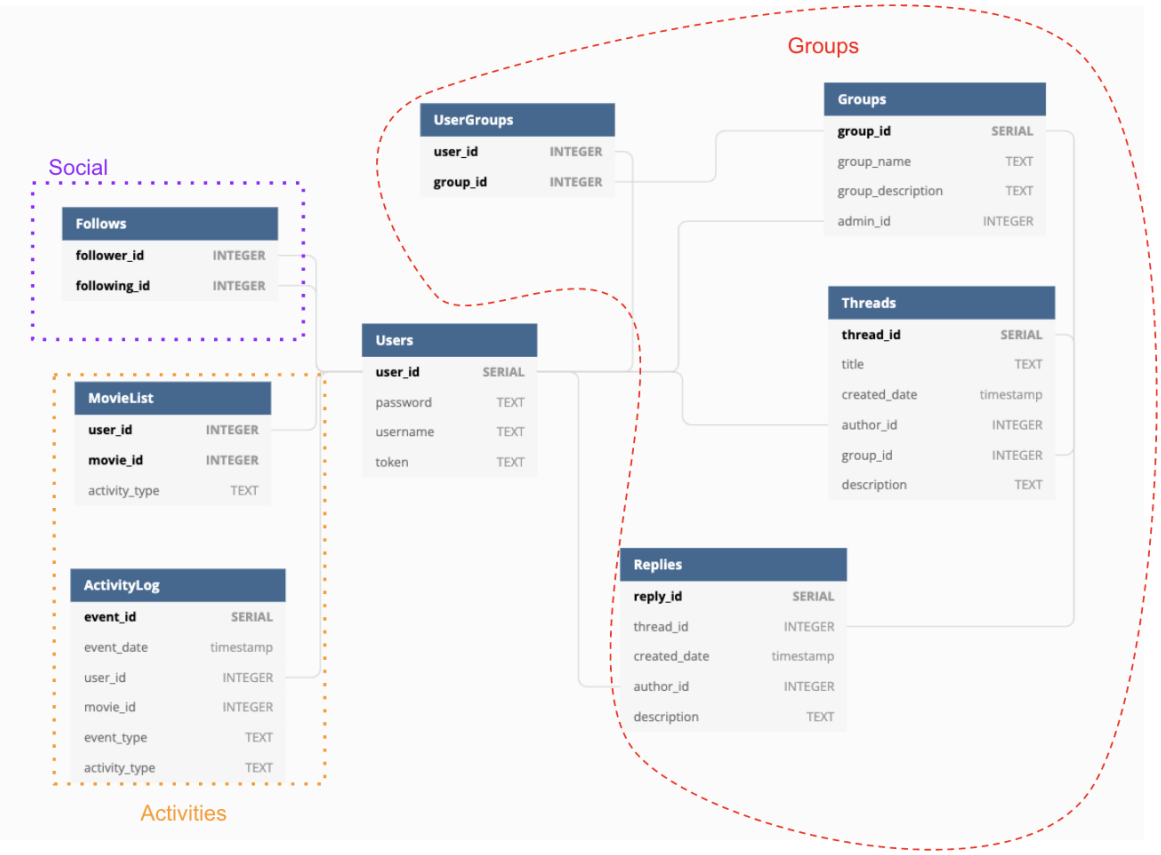
The Activity Management component contains the function for users to update their Movie Lists (Watched-list, Currently-Watching and Plan-to-Watch) with movies and such updates will be recorded as Activities and Users will be able to see the activities of people they follow. The

handling of these Activities are done by making API calls to the ActivitiesController.

The Forum Management component manages the forum functionality that exists in the groups. Group members can create threads and reply to threads and these requests are handled by the ForumController.

The Movie Management component manages the search functionality of our web application. Searching for movies is done by making API calls to the MovieController that in turn send requests to TMDB for movies. Furthermore, it communicates with the Recommender in the Recommender Service to give movie recommendations similar to the movie that is being searched for.

8.3.2. Database Design



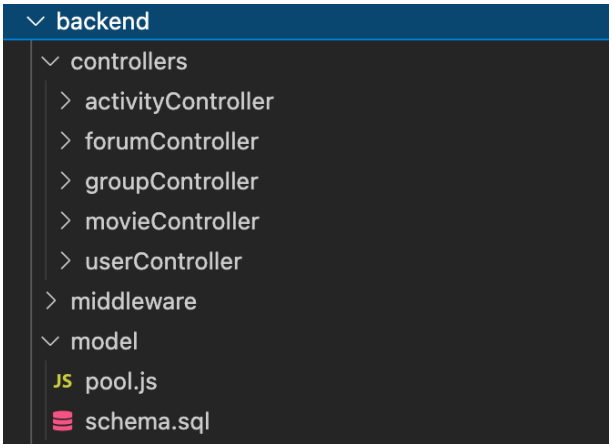
Database schema

Our project uses PostgreSQL, a relational database to store and persist information. A relational database is ideal for our use case as there are clear relationships between the different entities, and allows fast querying of the necessary information from a single or multiple tables (joins). The figure above shows the overall schemas used in our database, where the primary keys of each table are highlighted in bold, and the lines show the foreign key references.

Additionally, triggers were enforced to ensure that when an entry is inserted into **Group**, an entry is automatically inserted into **UserGroup** identifying the administrator of the group. Before an entry is deleted from **Group**, the equivalent **UserGroup** entry will also be deleted from the group.

8.3.3. Organisation of code

The backend code is organised based on the **Single Responsibility Principle**. Each controller handles a separate functionality and is hence split into different folders. Information regarding the database is also separated out into a separate folder called `'model'`, containing files for database configurations as well as the schema for the application.



Organisation of code on the backend

`server.js` will then be responsible for linking the correct controller for the different API calls.

```
// Controllers
const accountController = require("./controllers/userController/account");
const movieController = require("./controllers/movieController/movies");
const groupController = require("./controllers/groupController/group");
const activityController = require("./controllers/activityController/activity");
const eventController = require("./controllers/activityController/event");
const userlistController = require("./controllers/activityController/userlist");
const threadController = require("./controllers/forumController/thread");
const replyController = require("./controllers/forumController/reply");
```

Controllers in `server.js`

This approach to organising the code separates the functionalities into different folders, allowing for each component to be reusable. It also allows us to easily add new functionalities in one component with less likelihood of affecting other components.

9. Security considerations

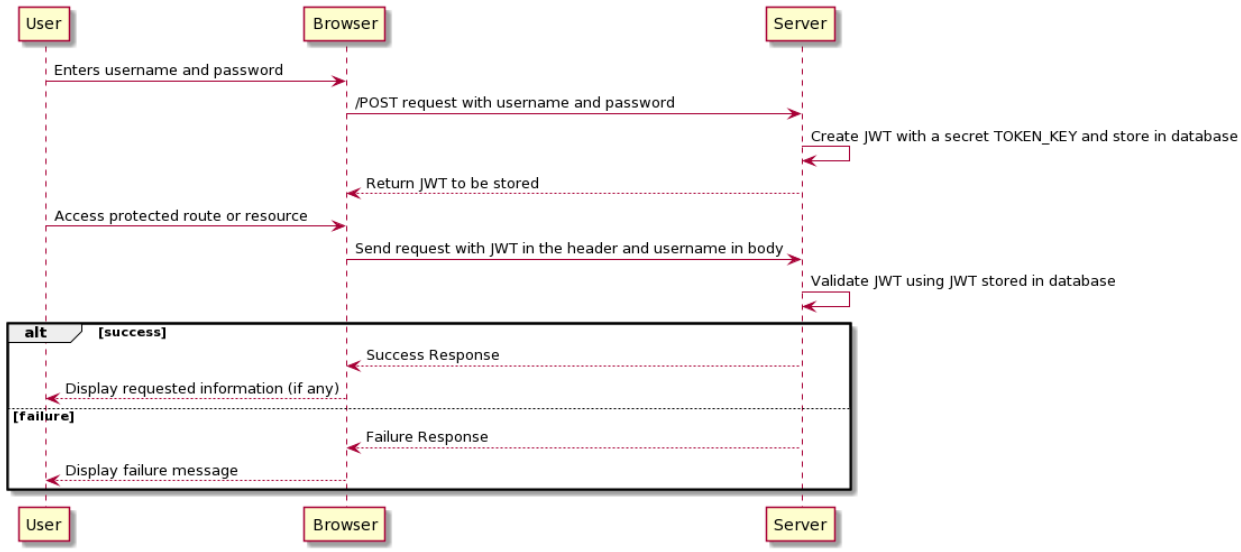
9.1. Storing passwords

Users' passwords should not be stored in plaintext to increase the security of the application. Our project uses **bcrypt**[3] to hash the password with a randomly generated salt. When a user successfully registers for an account with a username and password, the password will be hashed and stored in the database, instead of the password itself.

When users attempt to log into the account with a username and password, the password will be hashed and compared with the password in the database. With this approach, our application minimises the risk of the user's password being leaked during a database attack as attackers will only be able to retrieve the hashed passwords.

9.2. Authentication

JSON Web Tokens (JWT) are used to ensure that only authenticated users are allowed to perform certain actions *[NFR 2.1]*.



Sequence diagram showing authentication with JWT

When a user successfully logs into their account, a JWT is created and stored in the database. This JWT is returned and stored inside the browser. On subsequent API calls that access protected resources and require authentication, the browser sends the JWT token in the header, where the backend will verify the user.

JWT has a small size and is hence suitable to be passed through HTTP requests. In our application, the JWT is generated using the username of the user and a secret TOKEN_KEY that was randomly generated and stored in the environment variables of the application. Since the TOKEN_KEY is secret and hidden away from other users, it is less likely for other users to

[3] https://www.npmjs.com/package/bcrypt

be able to perform actions that require authenticated just by obtaining the user's username. This ensures that only the user that is authenticated to access the protected route or resource is able to access it.

Our application requires authentication checks for actions that will modify information for the current user. This includes:

- Adding movies to a list
- Changing password
- Following / Unfollowing a user
- Creating a group
- Joining a group
- Creating a thread in a group
- Replying to a thread in a group

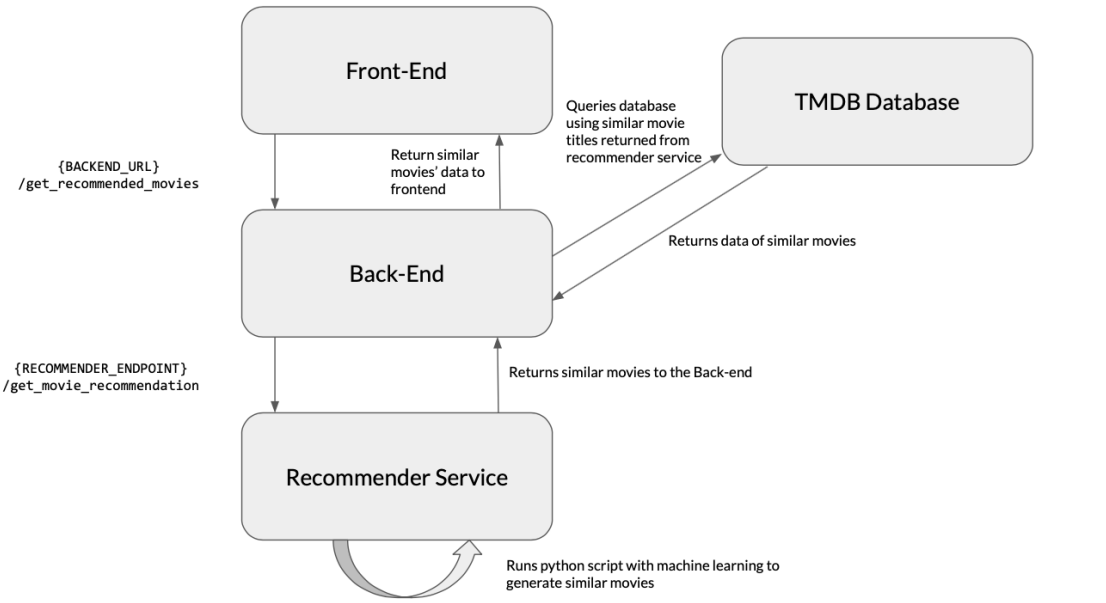
In these scenarios, the application will go through the additional JWT Authentication middleware before the requested action is processed.

10. Movie Recommendation Service

Tech-stack:

1. **Flask**
2. **Pandas**
3. **Sklearn**

To build our recommendation service[4], we had to build a new server which runs using Python as the main language. This is because our recommendation service uses Python libraries (namely *Pandas,Numpy and Sklearn*). The framework of choice for this server is Flask as it is easy to set-up and lightweight. In addition, the server only has to support a single endpoint which is the recommendation of the movie endpoint.



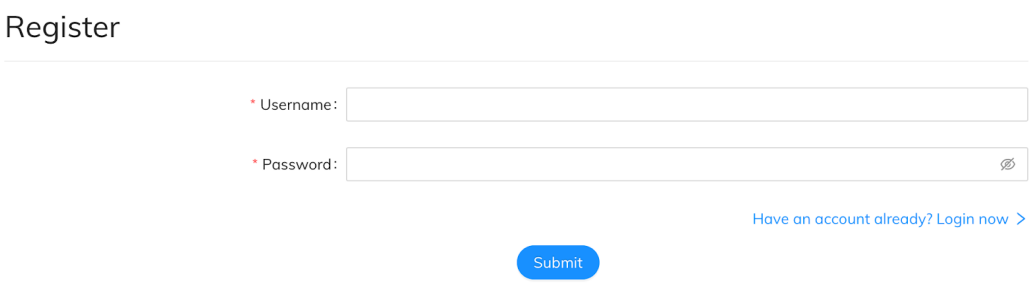
Application flow to retrieve recommended movies

Firstly, the front-end sends an API request to the backend on `/get_recommended_movies` endpoint, upon receiving the API request, the backend sends another API request to Recommender Service Server on `/get_movie_recommendation` endpoint. Upon receiving the request on the Recommender Service Server, the server calls its script to return a list of similar movies depending on the requested movie title. The script primarily uses the Sklearn library's pairwise cosine similarity algorithm to calculate the similarity between the requested movies and the other movies in the database. Upon finishing execution, the server returns a list of similar movie titles to the backend. The backend then queries for the data of these movie titles from the TMDB database. The response will then finally be returned to the front-end to be displayed to the user the list of similar movies and their respective movie data.

[4] With reference from: https://www.youtube.com/watch?v=XoTwndOgXBM&t=4525s

11. Application Screenshots

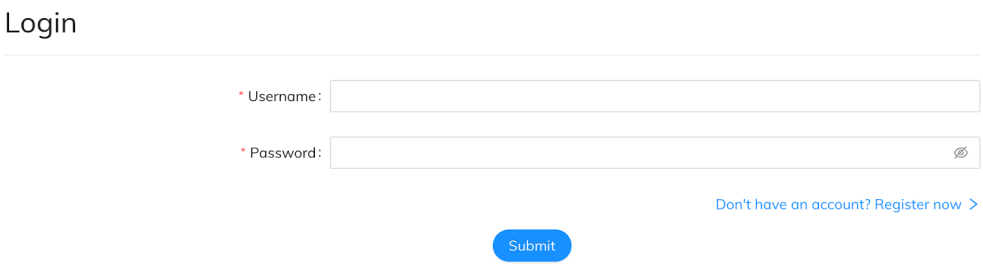
11.1. User Registration



Registration Page

Registration page asks for **Username** and **Password** input. Users will not be allowed to register if the username entered already exists. Users can navigate back to the Login page if they realise they already have an account registered.

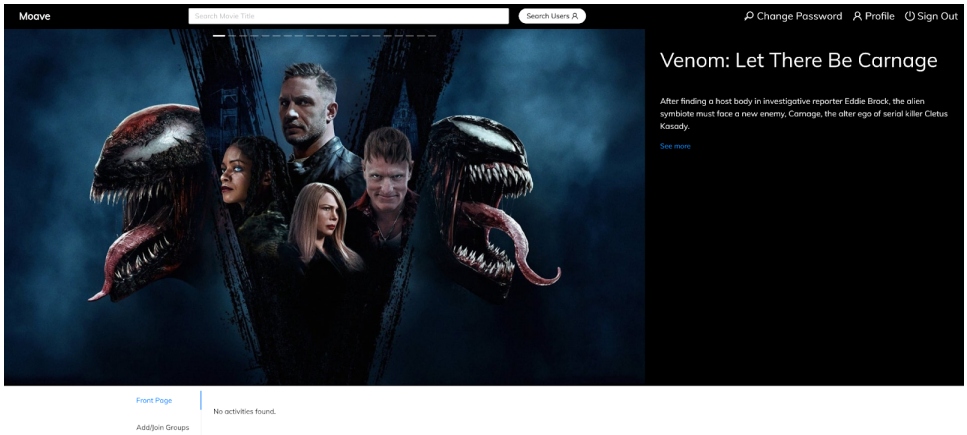
11.2. User Login



Login Page

Login page asks for **Username** and **Password** input. Users will not be able to login if they enter a wrong username or password. Users can navigate to the registration page if they do not have an account.

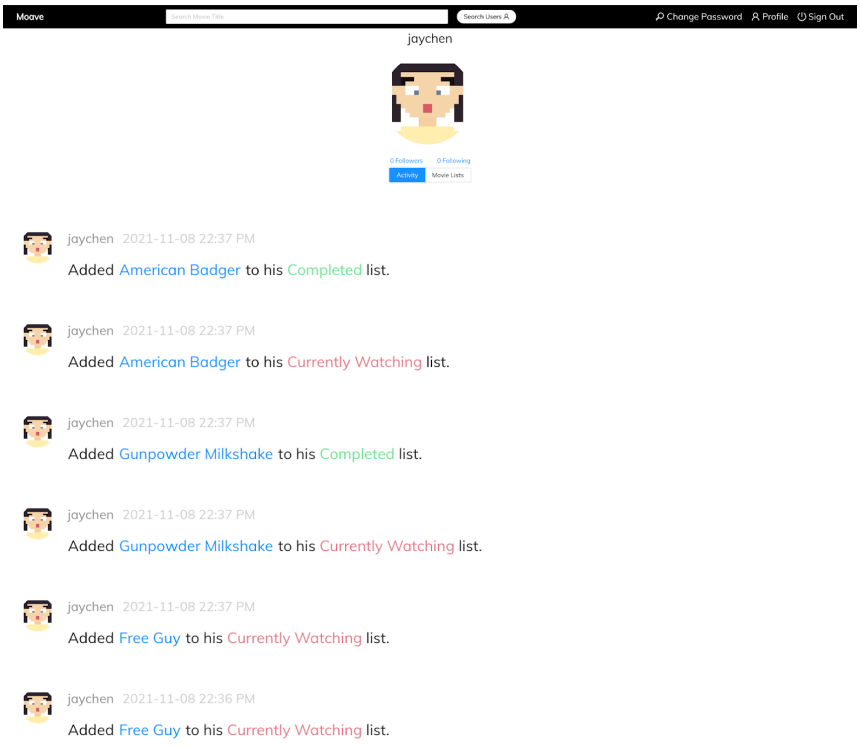
11.3. Landing Page



Landing Page

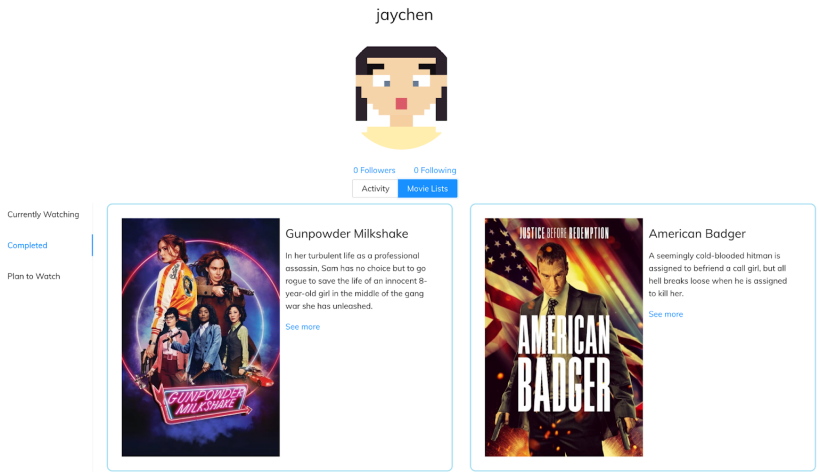
When users log into the website, they will be shown with this landing page, showing the latest trending movies and their friends' activities.

11.4. User Profile



Activities on Profile page

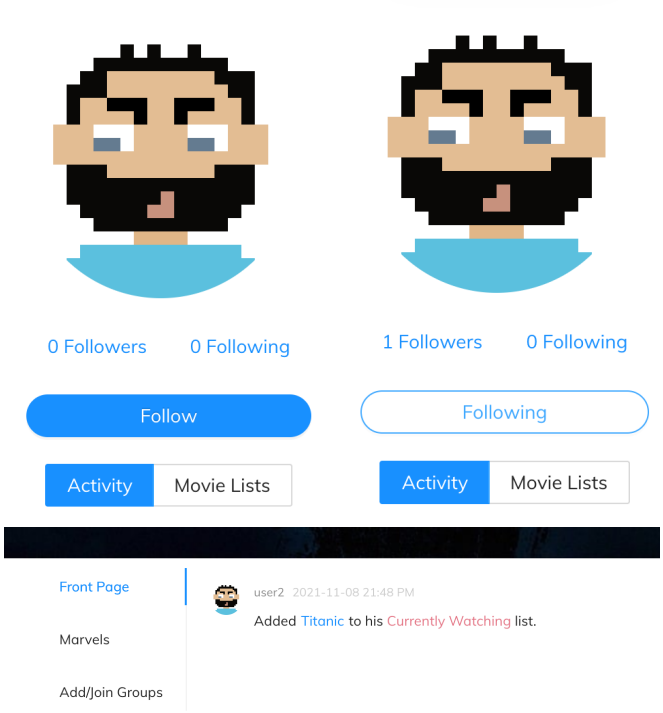
Users can view their own activities on their profiles.



Movie Lists on Profile page

Users can look through the movies they added into “Currently Watching”, “Completed” and “Plan to Watch” on their profiles.

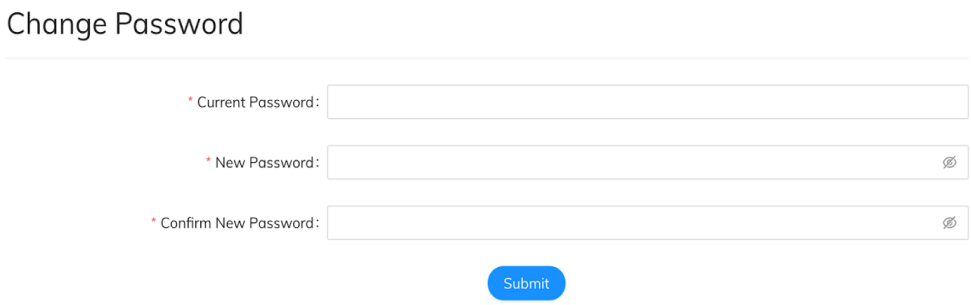
11.5. Follow / Unfollow Users Function



Following other users

Users will be able to follow and unfollow other users and see their activities on the landing page.

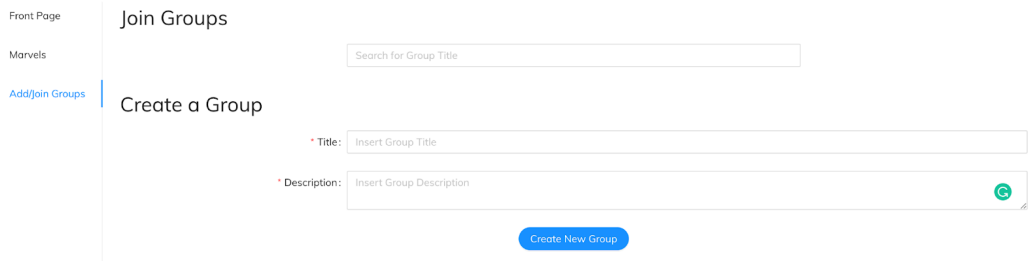
11.6. Change Password Page



Change password page

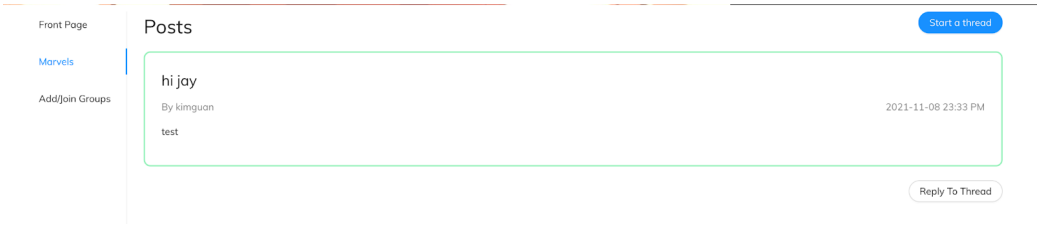
Users are allowed to change their password when they are signed in. The frontend will check if “New Password” and “Confirm New Password” inputs match before sending an API call to the backend to do the remaining checks and update the user’s password.

11.7. Groups



Joining and creating groups

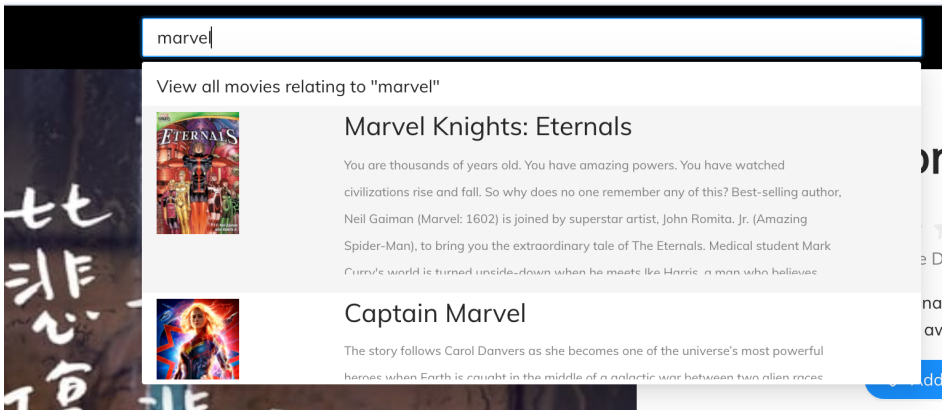
Groups can be created by users to discuss various movie topics that they are interested in. Users can join various existing groups as well by doing a simple search.



Threads in groups

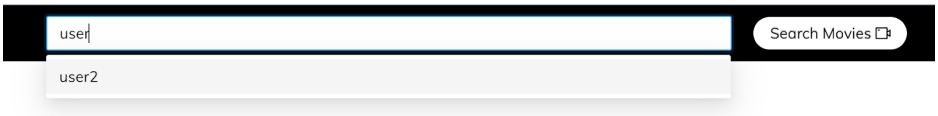
Threads can be created and replied to by users in different groups.

11.8. Search Function



Searching for a movie

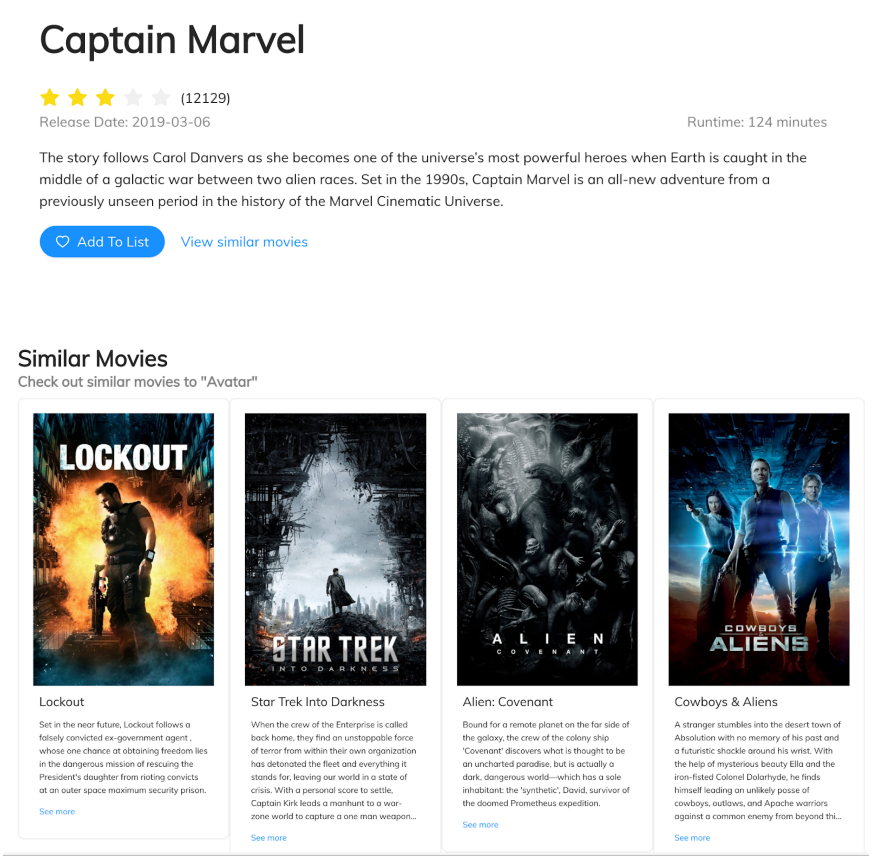
Users can search for movies they are interested in. The search bar will appear showing movies that match the keywords typed in the search bar by substring. Clicking on a movie will bring them to a specific movie page with more information on the movie.



Searching for a user

Users can search for other registered users as well. Clicking on the user will bring them to the specific user page.

11.9. Similar Movies Function



Finding similar movies

Users can search for similar movies from a movie that they are interested in.

12. Future improvements and enhancements

12.1. Email authentication

As an improvement, we can add email authentication during registration. Emails can be verified before users are able to sign in to the application. This increases the security of the system as we are able to verify the user by their email address. Furthermore, we can use the email address as a sign in together with password instead of username with password.

12.2. Reset Password

As an improvement, users should be allowed to reset their password. This will not be difficult once the email authentication is set up. Users who forget their password will be allowed to send a reset password email to their email to reset their password. This increases the Usability of the system, as users who forget their password will not lose all the information to their account and will be able to retrieve it with a reset of the password.

12.3. Change profile picture

As a small enhancement, users should be allowed to update their profile picture. This gives a more customised look for each user. Being a social platform, this would allow for users to identify others through their profile picture as well.

12.4. Group features

Within a group, more features could be added to encourage discussions between members. This can include allowing users to upload pictures, create polls, or having a ‘like’ feature for posts and comments. This would increase interactions between users on the platform, which is in line with the aim of being a social platform. With more interactions in the platform, it is likely that users will return and be more active on the application.

13. Reflections

- Initially, we were too ambitious in crafting our functional requirements and did not take into account the limited amount of time we had. As such, we had to remove some of the features along the way. However, working with an iterative development process and having weekly meetings provided us with more flexibility in changing the features that we want to implement.
- We had issues coordinating our database due to it being stored locally initially. It would have been better to use a deployed database from the start of the project, which would allow for all members to be working on the same schema. This would also reduce duplicate effort in re-creating tables in the database when changes are made from one member.
- It is important to lay out the functional requirements more specifically, so that everyone on the team is on the same page. At the start, we had different ideas of what a feature would be like. During the process of the project, we came up with more refined requirements and clarified with each other on the different ideas, coming to decide on how the feature works together.
- This project also pushed us to consider more aspects of security, including the use of technology such as Json Web Tokens to enhance the security of the system. Security is an important and fundamental aspect of any application, and hence it is important to consider it in any project and to improve upon it.