Orbital 25 Milestone 3

Popcorn Together

Apollo 11

Gan Ting En

Choo Kai Xi Kasey

Table of Contents

Testing Milestone 3

Conclusion

Project Overview

With the proliferation of movies in cinemas and streaming platforms, it may be overwhelming for users to identify what movies they would like to watch. This is especially so when they are searching with friends and family, trying to find a film they can watch together.

Ever wondered what movies you have already watched, or which of the hundreds of thousands should be next on your wishlist? PopcornTogether seeks to provide a solution to the Movie Weekend conundrum.

Additionally, the movie experience can be enhanced when undertaken with friends, making features that support this shared entertainment experience an even seamless one.

Project scope

A one-stop app for movie enthusiasts, a record of their movie-watching journey together with friends, over a bucket of popcorn!

Tech Stack

- **Backend**: MongoDB, Express, Node.js
- **Frontend**: React.js, CSS
- **Deployment**: Render

Popcorn Together's value

The proposed web application PopcornTogether seeks to create a platform where movie enjoyers can go to consolidate their movie-going journey, allowing them to record their watches, track films they want to watch, and even receive relevant recommendations. Furthermore, with a friends list integration, we empower users to find common movies to watch with their friends.

Personal note

Through this project, we aim to pick up industry relevant software engineering skills and practices. We also hope to learn how to better cater to user interests and preferences, by designing a UI/UX that is intuitive, engaging, and responsive to user needs. Additionally, the project is one that we believe can add value to one's leisure time by reducing the effort spent on collating one's watch history and finding movies. Through the user interaction feature with their friends, we also hope to deepen the bond between friends by enabling them to share and discover movies based on common interests, fostering a more connected and meaningful viewing experience.

Objectives

This project aims to create a platform for users to maintain an account with information on movies they have and watched and want to watch. It also provides discovery functions where users can find movies through active searching, recommendations, or through friends they have connected with.

Requirements

Observations

Identified area	User needs
Unsure of what movie to watch	More recommendation avenues to discover movies that interest them
Forgot what movies they have watched	A list to track movies that they have already watched
Difficulty finding films to watch with friends	A feature to match movies two or more users would like to watch
Unable to remember which movies they planned to watch	A list to record movies they plan to watch

User stories

- As a user who wants to find a movie that fits my mood and preferences today, I want to be able to type genres and other search criterias. I want to use the search bar to find movies that apply.
- As a user who wants to easily and purposefully discover new movies to watch, based on specific categories provided.
- As a user who wants to see if a movie is worth watching or not.
- As a user who wants to remember what kind of movies I have watched and also want to watch.
- As a user who wants to see what my friends are watching, I want to add them to my friends list to keep track and stay connected with them.
- As a user who wants to find a movie both my friends and I will enjoy.

Analysis

Pain Point	App Requirements
Unsure of what movie to watch	Search functions made for active finding of specific movies that fit the user's preferences as well as passive methods to push movie recommendations for the user to discover
Forgot what movies they have watched	Through a Watchedlist, allow users to find and record all films they have watched before, before translating into meaningful analytics for their use and sharing with friends
Wants to record what kind of movies I want to watch	Develop a Wishlist feature to record movies the user plans to watch

Pain Point	App Requirements
Difficulty finding films to watch with friends	A Friends list with shareable records of previously watched movies and wishlisted movies, providing ways to narrow down their search for movies to watch together by eliminating previously wathced movies and by checking for matching wishlists
Wants to see if a movie is worth watching or not	Include a communnity review function. While opinions on movies are quite subjective, allowing users to read reviews by others who have already watched the film can provide insight and help users decide if they want to watch the film
Share movie analytics with friends as a conversation topic, or to find movies with common genres they can watch together	A Watch Statistics page will be useful to analyse a user's rating levels, and track quantity of movies watched and other numbers

Developer requirements

The crux of the issue we have identified is the ability to find new movies to watch. The most important thing to consider in achieving this is narrowing the options as much as possible. PopcornTogether will form a collaborative effort with the user in this filttering process. Users can actively search for new movies and record movies they ahve already watched, PopcornTogether will provide robust features that allows users to discover new movies through a variety of methods, each with its own strength. Furthermore, through the compilation of statistics of the user's watch habits, the user can have more insight into what type of movies they may prefer.

With so many funcitons serving different purposes, the user can become lost and not maximise the use of each one. There should be dedicated pages for each function, and detailed information for each fucntion's usage.

Powerful search and filter functions enable easy user queries when searching for movies, while a robust profile system enables discovery of new movies to watch together with friends or simply by themselves.

Development plan

- **Milestone 1 (Week 1 - 3)**: Technical proof of concept
    - A minimal working system with both frontend and backend integrated for core features
    - Develop Login features
    - Watchlist and wishlist is able to store movie data
    - Design individual pages using Figma
- **Milestone 2 (Week 4 - 8)**: Prototype
    - Database integration added using MongoDb
    - Querying of database implemented
    - A working system with the core features
        - Account Authentication
        - Movie Search

        - Personal Profile & Watch Statistics
        - Watchedlist
        - Wishlist
        - Movie Discovery
    - User has edit access for the Watchedlist and Wishlist
    - The app produces a list of movies given a set of filters, after search by user
    - Deployment of core features

- **Milestone 3 (Week 9 - 12)**: Extended system

    - A working system with both the core and extension features
        - Users can add reviews to movies, which are then stored and retrieved from the database
        - Friends list
        - Friend Activity
        - Movie match feature implemented for alternate movie discovery with friends in Friends list

- **Milestone 4**: Testing and debugging

API Usage

MongoDB : database for storing user information

TMDB (The Movie Database) : database for querying movie information

Render : Render API for hosting the backend express server

Milestone 2 summary

The objective in milestone 2 for PopcornTogether is to develop a working app in order for user testing as well as extension of features. This version will incorporate the authentication functionality built in Milestone 1, improving upon it for user sessions, and adding our other core features which include: - Account Authentication - Movie Search - Movie Discovery - Personal Profile & Watch Statistics - Watchedlist - Wishlist - Friends List

After simple local testing and completion of core features, we aim to deploy PopcornTogether using Render.

At this stage, PopcornTogether has been deployed at https://popcorntogether-test.onrender.com on Render. The list of completed features are expanded on below.

Work completed

Milestone 1

- Finalised feature design
- Created a minimal working system for authentication routes
- Implemented registration function
- Implemented login function
- Setup of MongoDB database
- Created user schema and respective page designs

Milestone 2

- Created the landing page (Homepage)
- Implemented session tracking
- Implemented search function
- Implemented results rendering function
- Implemented Movie Discovery function
- Implemented Watchedlist and Wishlist function
- Implemented Profile and Watch Statistics function
- Implemented Friends List and functionality for viewing friends' Watchedlists and Wishlists
- Deployment of PopcornTogether using Render

Milestone 3 summary

The objective in milestone 3 for PopcornTogether is to develop further extension in order to complete our suite of functions. This version will incorporate the full functionality completed in Milestone 2. We aim to conduct more comprehensive testing in addition to the user testing conducted for milestone 2. After completion, we aim to finalise deployment on Render and conduct one last round of user testing on the deployed app.

Extension Features to add for milestone 3:

1. Community Reviews
2. Movie Match

Testing to be conducted:

1. Unit testing
2. Integration testing
3. User testing (local)
4. User testing (deployed)

Work completed

Milestone 3

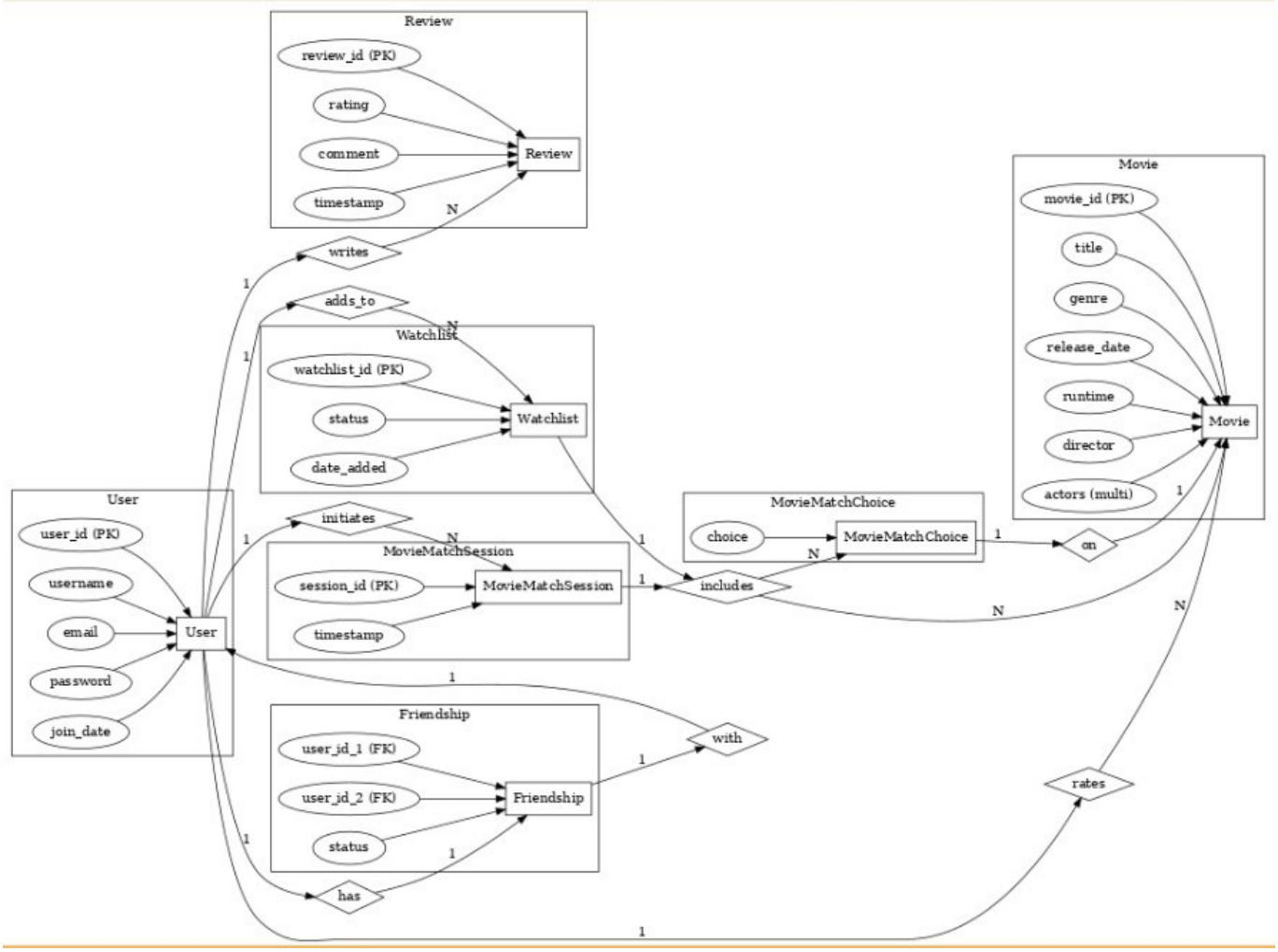
- Implemented Movie Match function
- Implemented Star Ratings function
- Implemented Reviews function
- Accounted for lack of case sensitivity in login/signup feature
- Updated Reviews Given feature in Watch Statistics
- Unit Testing
- Integration Testing
- User Testing
- Deployment of PopcornTogether using Render

Features

Feature	Description	Purpose
User Account/Profile	Allows user to maintain their data.	Serves as a way to track their movie journey.
Movie Search	Basic search function, allows users to search based on film title, genre, language or release date	Provides a way for users to search for a specific movie, or discover movies by specifying certain paramenters.
Movie Discover	With catergories displayed in the Homepage, users can click into 4 separate sections, namely Latest Movies, Timeless Favourites, Friend Activity, and Popular Franchises to discover movies under these 4 categories. Upon entering those sections, buttons are available for users to add the movies into either their Wishlist or Watchedlist.	Allows users to uncover new movies they have not watched before, and gain timely recommendations from their friends' activity.
Watchedlist	Stores all the movies the user has watched before.	Allows the user to track their films, it serves as their record so they do not double watch movies. It can also give them inspiration for films they could watch based on what they have enjoyed in the past.
Wishlist	User can add movies they want to watch in the future here.	Track movies that the user is interested in but hasnt gotten around to watching. With limited time to pursue leisure activities, users may nnot be able to watch every film immediately, the wishlist serves as a reminder for films they want to watch in the future.
Friends list	Allows users to connect with one another, friends can view each others watched lists and wishlists.	For friends who want inspiration, they can browse through their friends' lists. For friends looking for a movie to watch together, they can find common movies in their wishlists.

Feature	Description	Purpose
Watch Statistics	Tracks data based on movies they add to their watchedlist, for example, top genre.	Gives the user some basic insights into what kind of movies they have enjoyed before, as well as their most watched genre. The watch statistics seek to provide users information on their watch habits for their movie hunt.
Community reviews	Allow users to pool reviews on movies they have watched, as well as rate the movie out of 5 stars. Average star ratings will also be calculated.	Gives users a better idea of what they can expect from movies they are interested in, and curate their selection to their tastes better.
Movie Match	For users who are unsure of what movie they want to watch, they would be able to specify a set of filters and random films will be generated and suggested to the user. We aim to mimic a social media for-you page layout for this.	With a general idea of what kind of film they want to watch, users can begin to browse for movies that interest them. This feature targets users who do not know the exact movie they want to watch or are just looking for more.

User diagram

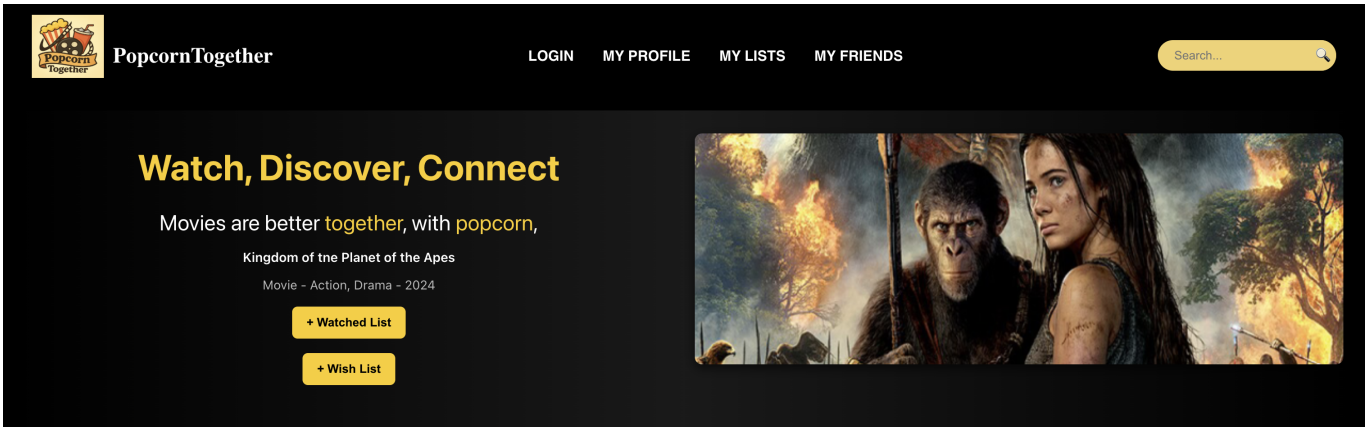


We have designed a preliminary diagram to visualise the various functionalities and the interaction between moving parts for Popcorntogether.

Feature Descriptions

The following section details the functionality and organisation of the respective features in PopcornTogether.

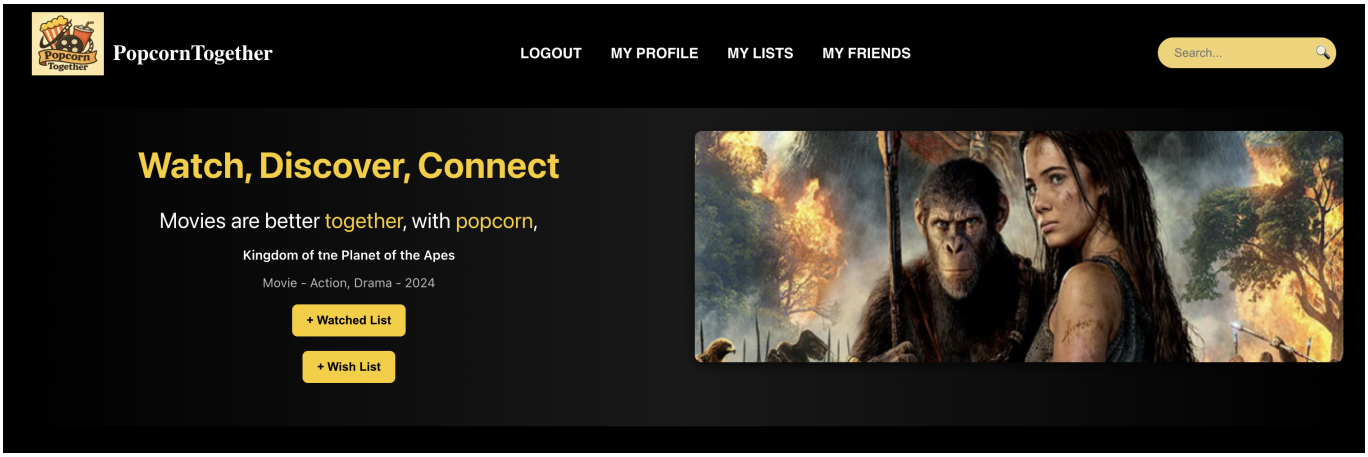
Landing page



The user will first arrive on the **landing page**. From here, they are able to access the movie search function without logging in. This will be further expanded on in the movie search feature below.

For Movie Discovery, users can also access recommended movies under the 4 categories (Latest Movies, Timeless Favourites, Friend Activity, and Popular Franchises) in the **landing page** to find more movies. However, to access the Watchedlists/Wishlists function, users will have to login, elaborated below.

Other features such as Friends List and Account Profile will also prompt the user to login first, and is protected using an isAuthenticated function which checks for an active session before allowing access, it will redirect the user to the authentication page if no valid session is detected.



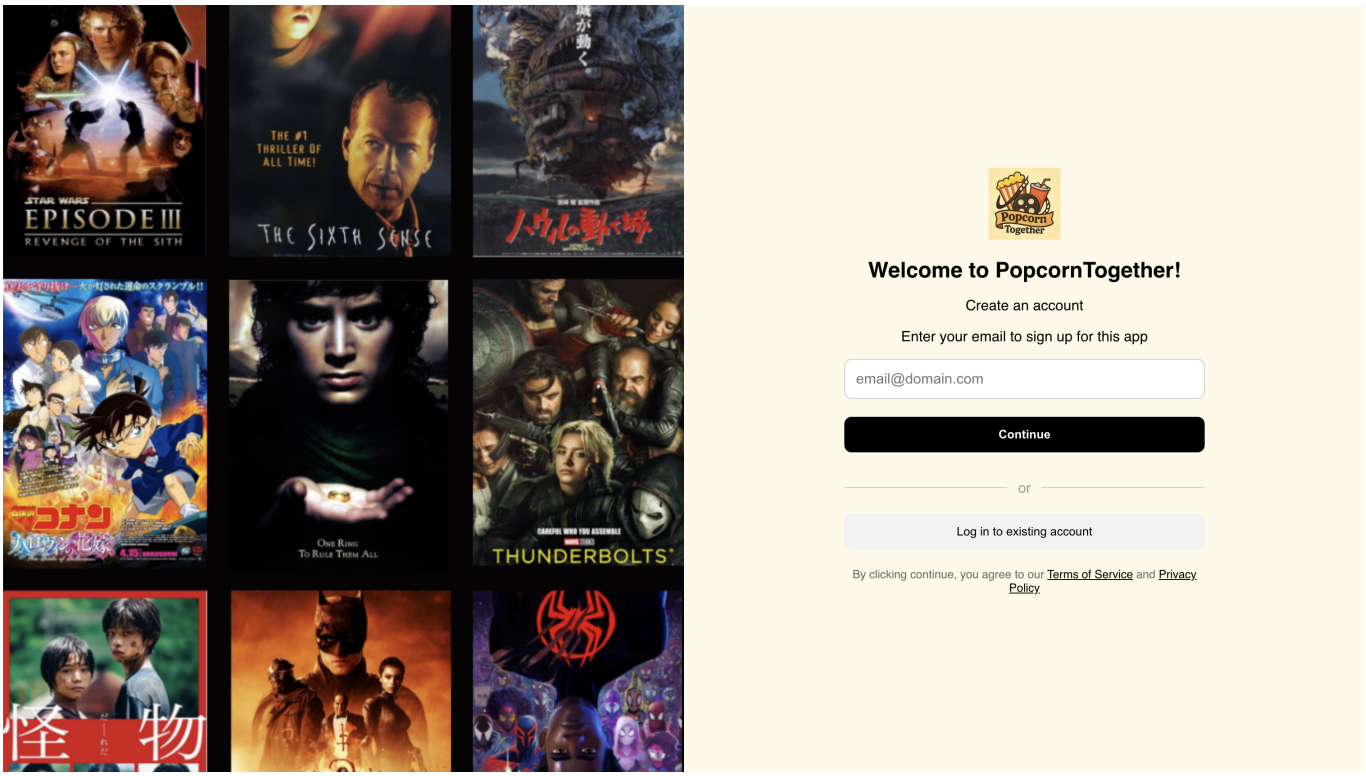
After logging in, the user will be redirected to the landing page once again, they will now be able to access the various features. They will also be able to logout of their account.

User Account/Profile

Core feature

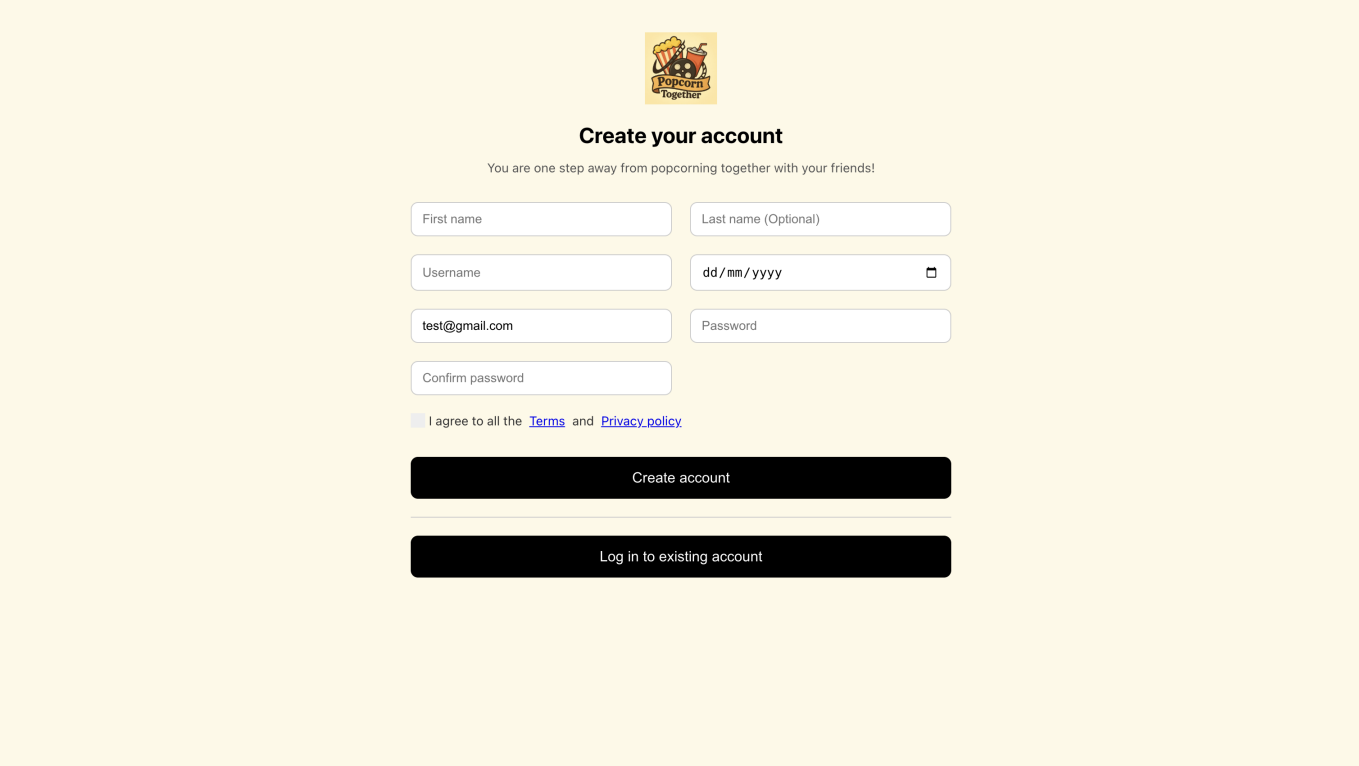
Users will first find themselves on the landing page where they can access the registration and login features. By creating an account, we are able to assign the user to their very own watched list and wishlists, as well as provide information on their watch statistics. Logging in also allows the user to have a session id, which will be used to track their session and allow them to make use of the different features of PopcornTogether.

Authentication page



Users will be redirected to this page where they can then create an account or login in with their existing account.

Register page



Users will register for an account on this registration page, or access the login page if they already have an account. Upon registration, user information will be stored on MongoDb.

MongoDB

This project uses MongoDB Atlas, a cloud-hosted NoSQL database, to securely store and manage all user-related data. MongoDB's flexible document structure makes it ideal for our needs, including handling dynamic user preferences. This is essential to our core features that provide the users functionality to curate their personal Watched List, Wishlist and Friends List, whereby data is managed through the MongoDb database.

This includes storing the below data in order to enable ths web application's features:

1. Users

```
{
    "_id": "...",
    "firstName": "Test",
    "lastName": "1",
    "username": "test1",
    "email": "test@example.com",
    "password": "...",
}
```

2. Watchedlists

```
{
    "userId": "ObjectId",
    "movies": [
        { "movieId": "123" }
    ]
}
```

3. Wishlists

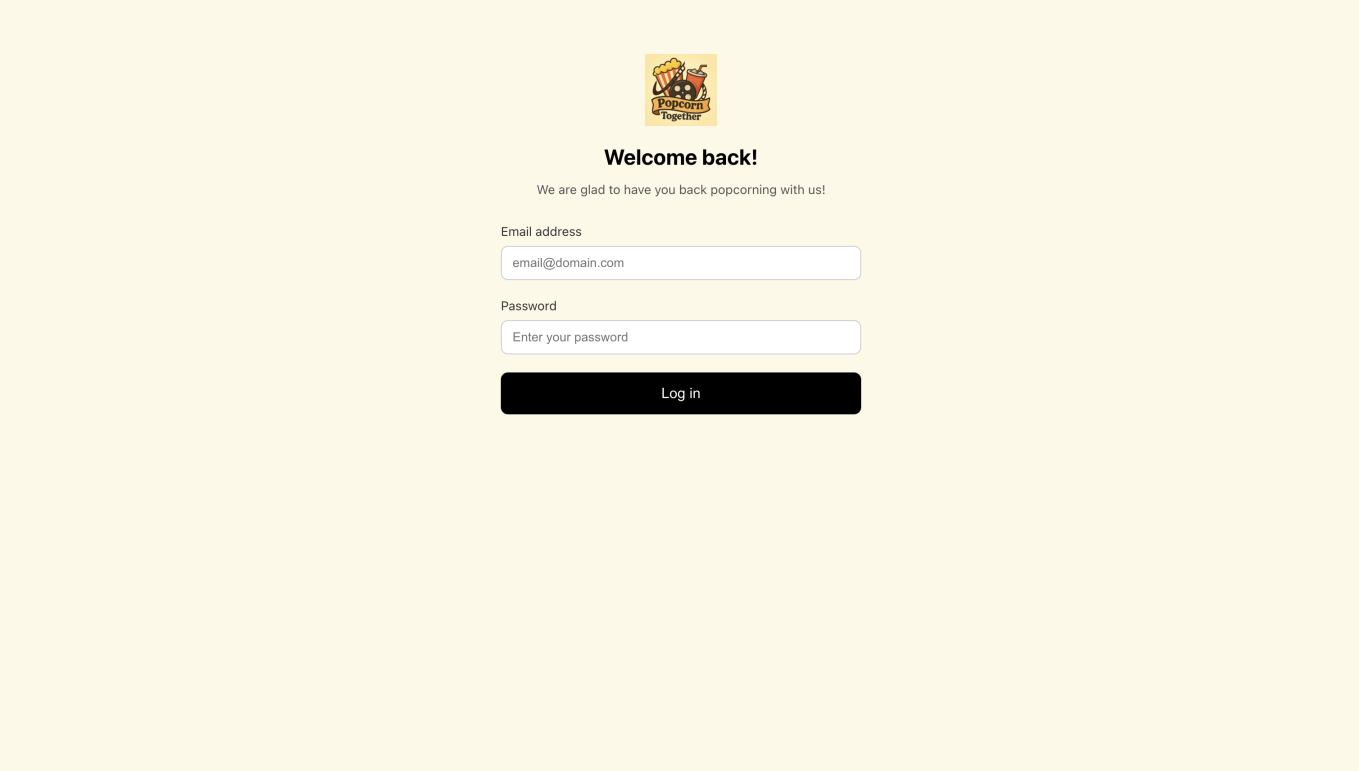
```
{
    "userId": "ObjectId",
    "movies": [
        { "movieId": "456" }
    ]
}
```

4. Friends list

```
{
    "userId": "ObjectId",
    "friends": [
        { "friendId": "ObjectId", "username": "janedoe" }
    ]
}
```

Furthermore, MongoDB provides certain security features such as hashing passwords using bcrypt before storing, and access to the MongoDB API being secured using environment variables and IP whitelisting.

Login page



Users will login through the login page by using their username and password. The details are verified by matching with an existing account that has been registered and stored on MongoDb. Upon successful login, the user is issued a session token which allows them to use PopcornTogether's features.

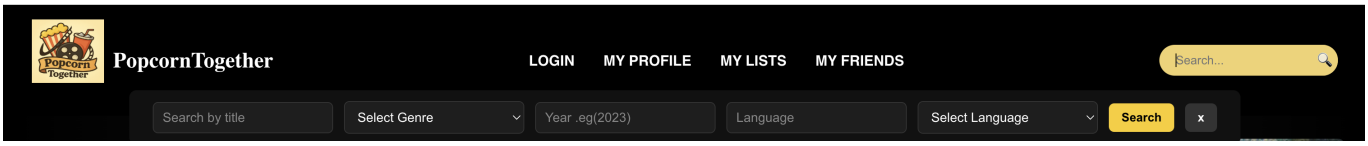
Movie Search

Core feature

The movie search feature will be the main way users can search for movies, the basic search function. It makes use of The Movie Database (TMDB) to handle queries. We plan to include filters for the following:

1. Title
2. Genre
3. Release year
4. Language

Search function



The user will be able to use the search function via the search bar. On clicking the search bar, the user will also be able to access a set of filters that they can use to search for films by passing in a set of parameters:

> eg. Genre: Action, Release year: 2020, Language: English

This component is mounted via the header and filter components respectively. This allows the user to access the movie search function on all PopcornTogether pages with the exception of authentication pages.

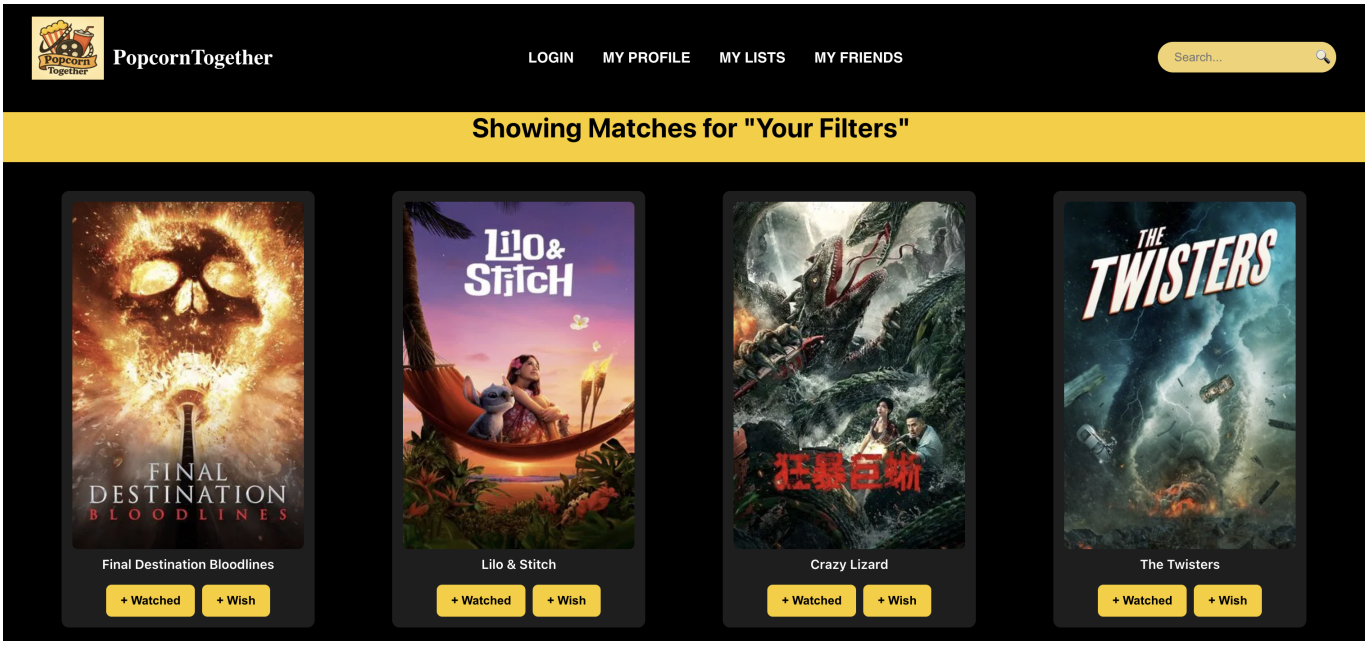
The Movie Database (TMDB)

We use TMDB as our primary source of movie data. TMDB provides a robust and comprehensive API that allows us to access up-to-date information on thousands of movies, TV shows, cast members, genres, and more.

Upon user input (searching by title, genre, language, or year), our backend queries the TMDB API to retrieve relevant movie data. This data is then displayed on the frontend for user interaction.

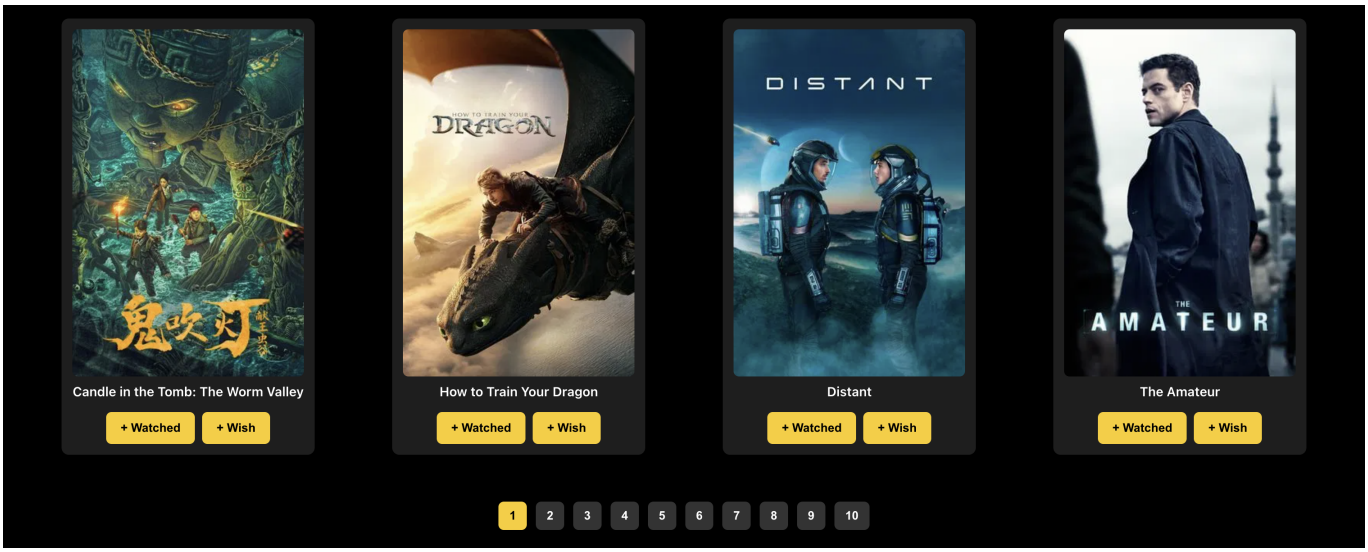
After successful query, the users will be able to view the following result page.

Results page



The results page will render results from the TMDB queery.The layout will be two rows of 4 movies, for a total of 8 movies a page. They can scroll down to view the next 4 movies, as well as view the next few pages, which will render the next 8 movies. The movie results are sorted based on TMDB's internal sorting mechanism.

Results pages



The user will be able to see up to 10 pages of suggested movies, based on their search parameters.

If the user is logged in, they will be able to access the below two functions:

1. Users can mark a movie as watched, which is then logged in the watchedlists collection for future reference or social sharing.
2. Users can save movies they are interested in watching later. This is stored in the MongoDB wishlists collection.

Movies added to the respective lists are stored in MongoDB via their movieId. This will enable generation of the next few features.

This addresses the user concern we have identified:

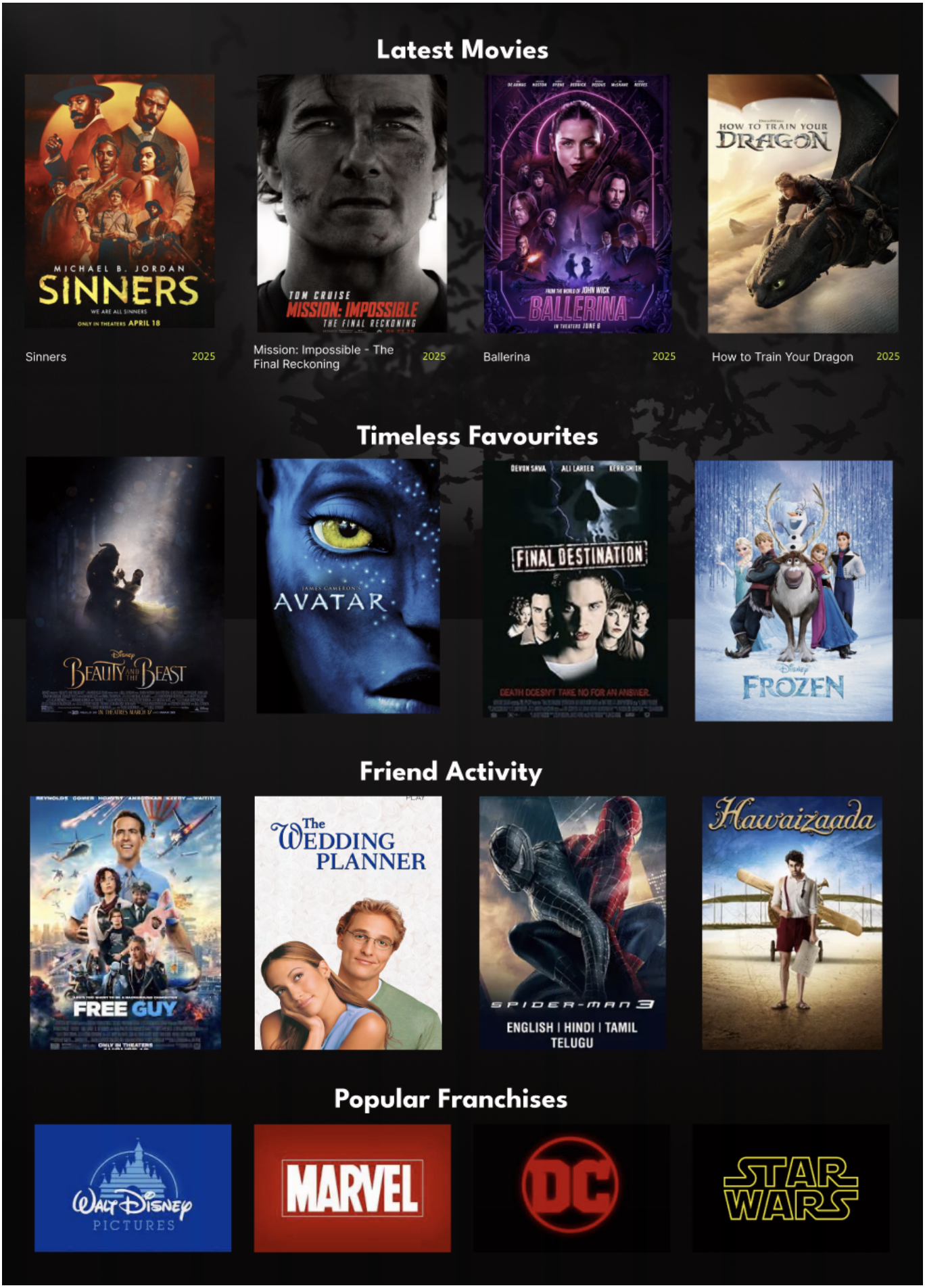
> As a user who wants to find a movie that fits my mood and preferences today, I want to be able to type genres and other search criterias. I want to use the search bar to find movies that apply.

Movie Discovery

Core feature

By clicking in either one of the 4 discover sections (Latest Movies, Timeless Favourites, Friend Activity, and Popular Franchises), users will be redirected to new pages where they can explore movies under the 4 categories respectively.

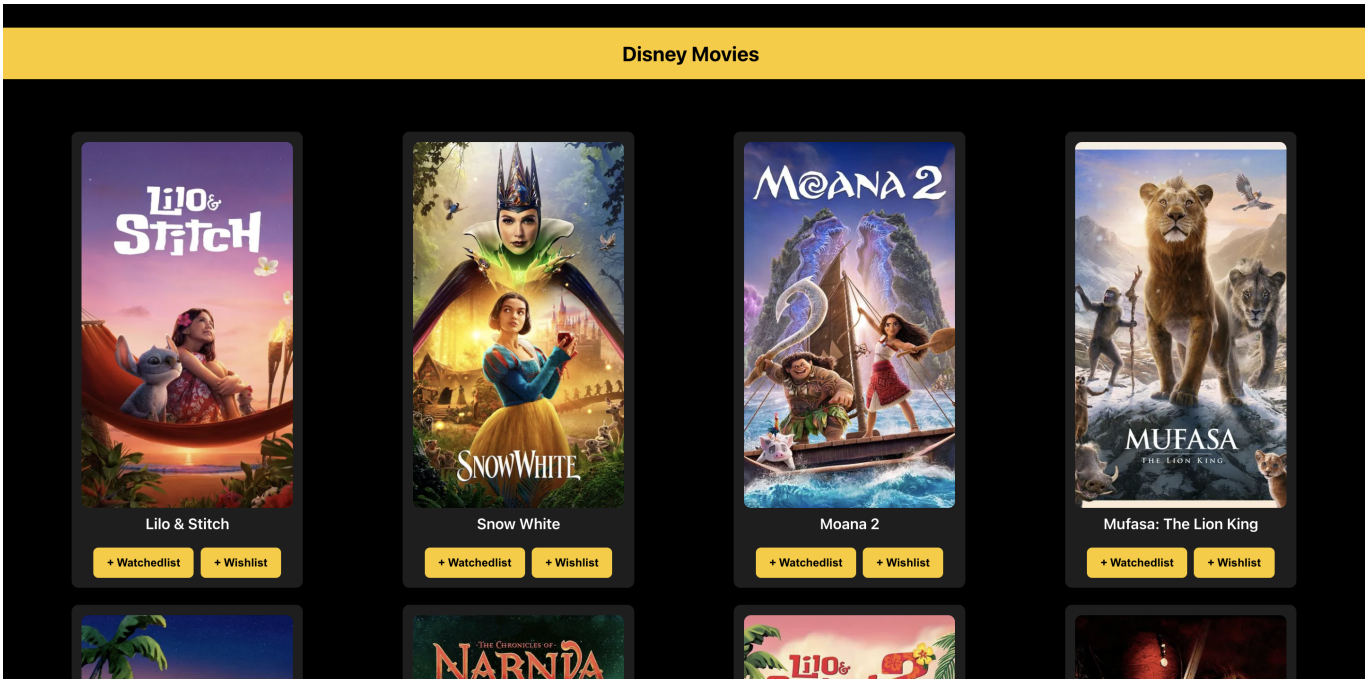
Movie Discovery in Homepage



Latest Movies provides recommendations for movies released in the last 3 months as of the current date. Timeless Favourites suggests movies released before 2015-12-31, yet still remain popular today, quantified by filtering movies that have received more than 1000 votes in the TMDB database. Friend Activity pushes all movies that have been watched by the user's friends in the past 3 months, into one consolidated list. It will be sorted in reverse chronological order, as per the date the movie was added to the friend's Watchedlist. Popular Franchises consits of 4 mainstream American franchises, namely Disney, Marvel, DC and Star Wars. By clicking into the image of each franchise, popular movies from that franchise will be featured.

The image below is a sample of what the discover pages look like, in this case for the Disney Franchise:

Disney Franchise Movie Discovery



This addresses the concern we have identified :

> As a user who wants to easily and purposefully discover new movies to watch, based on specific categories provided.

Watchedlist

Core feature

When the user creates their profile, they will be able to store movies they have already watched before in the Watched list. This can be accessed using a dropdown menu. As shown below:

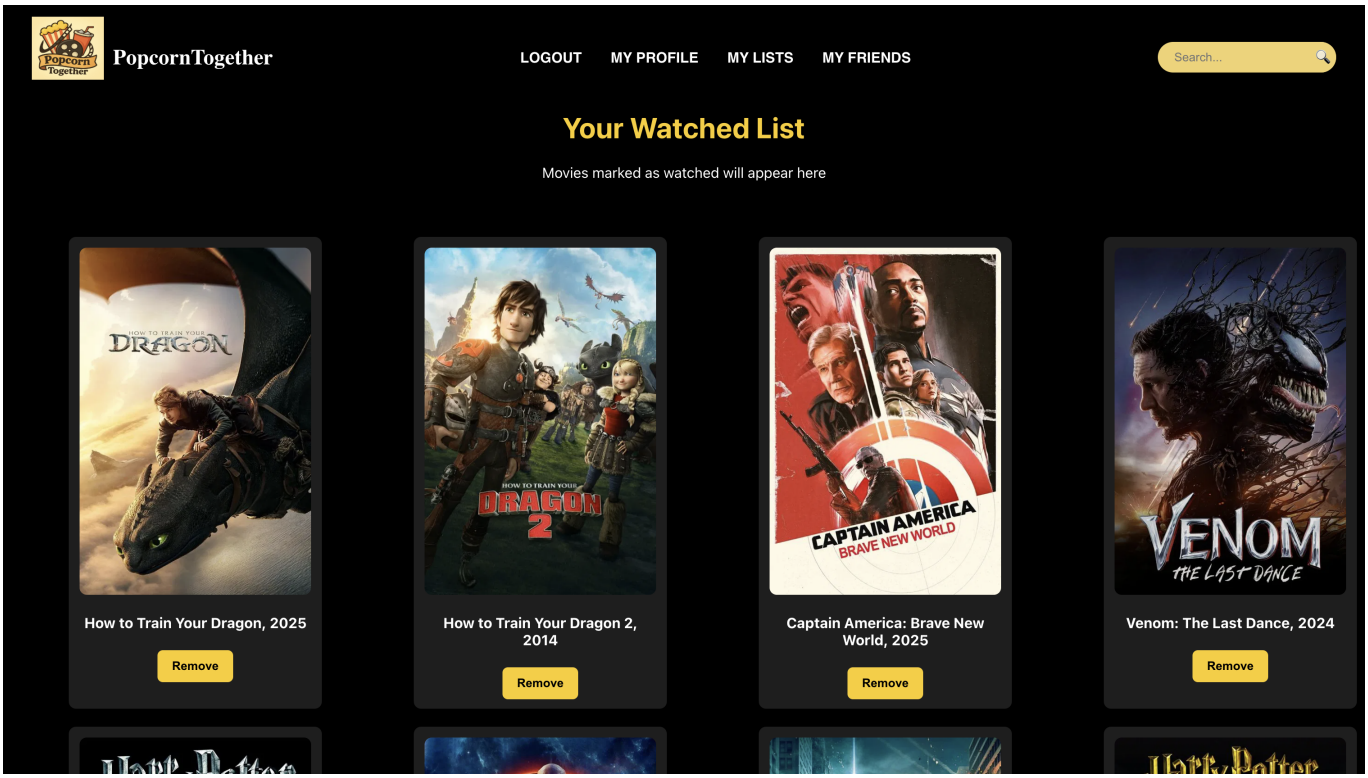
Dropdown menu

This allows the user to navigate to either the Watched list or Wishlist by clicking on the MY LISTS button in the header.

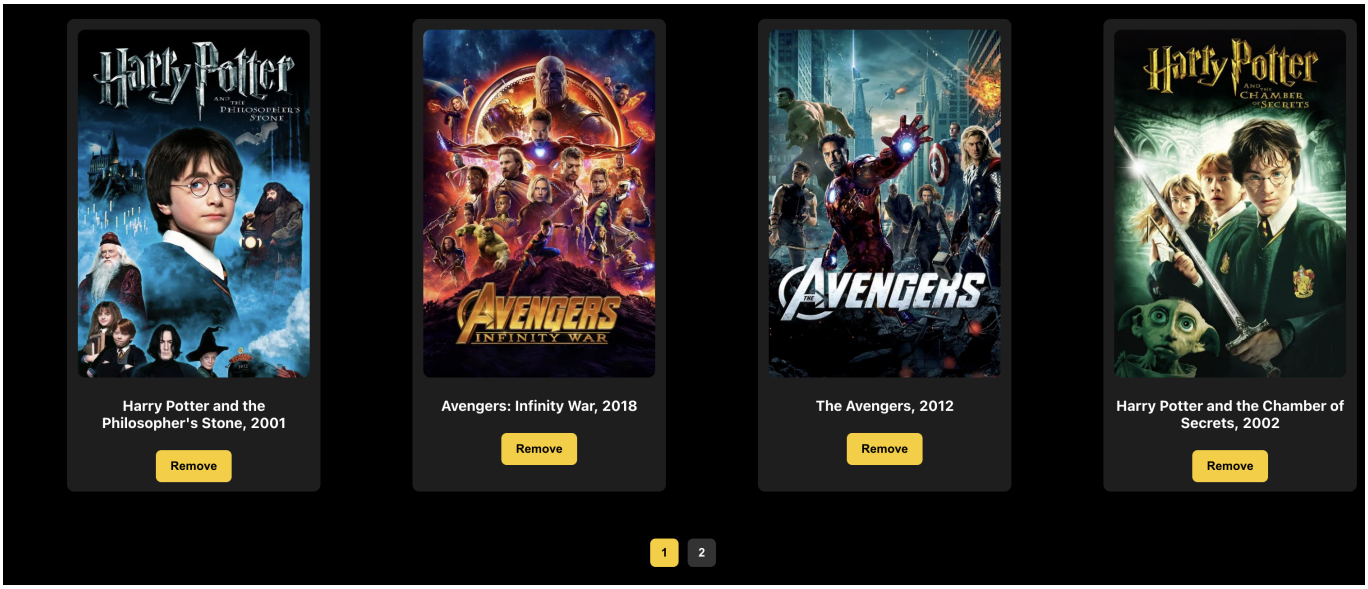
Afterwards they will be redirected to the Watched list.

This page will display all the movies that the user has marked as watched. The backend stores the movie IDs in MongoDb, the WatchedListPage then retrieves the movie IDs and queries TMDB for all the stored movie. It then displays 8 movies per page, similar to how the ResultsPage is shown but with one small change, instead of being able to press '+ watched' or '+ wish" buttons, they can now only remove the the movie from the Watched list.

Watched list



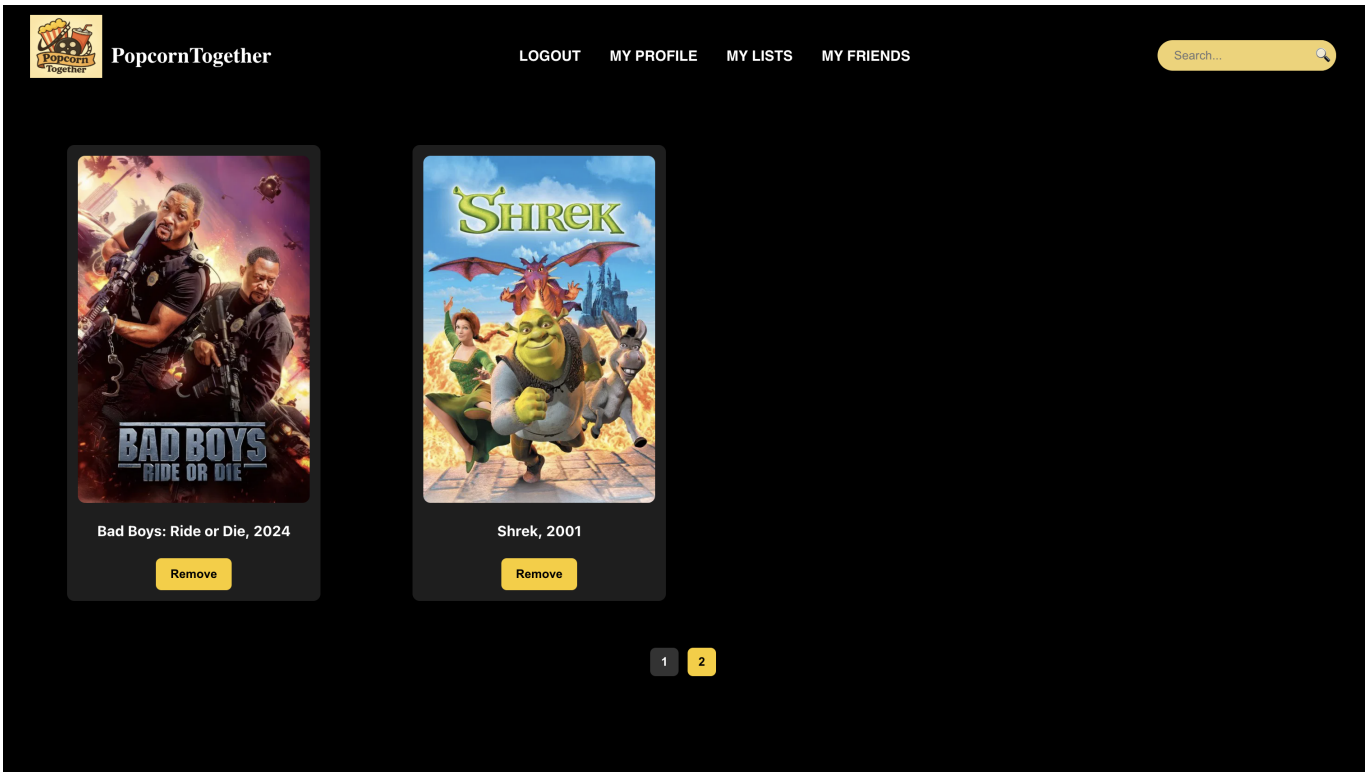
Watched list with pagination



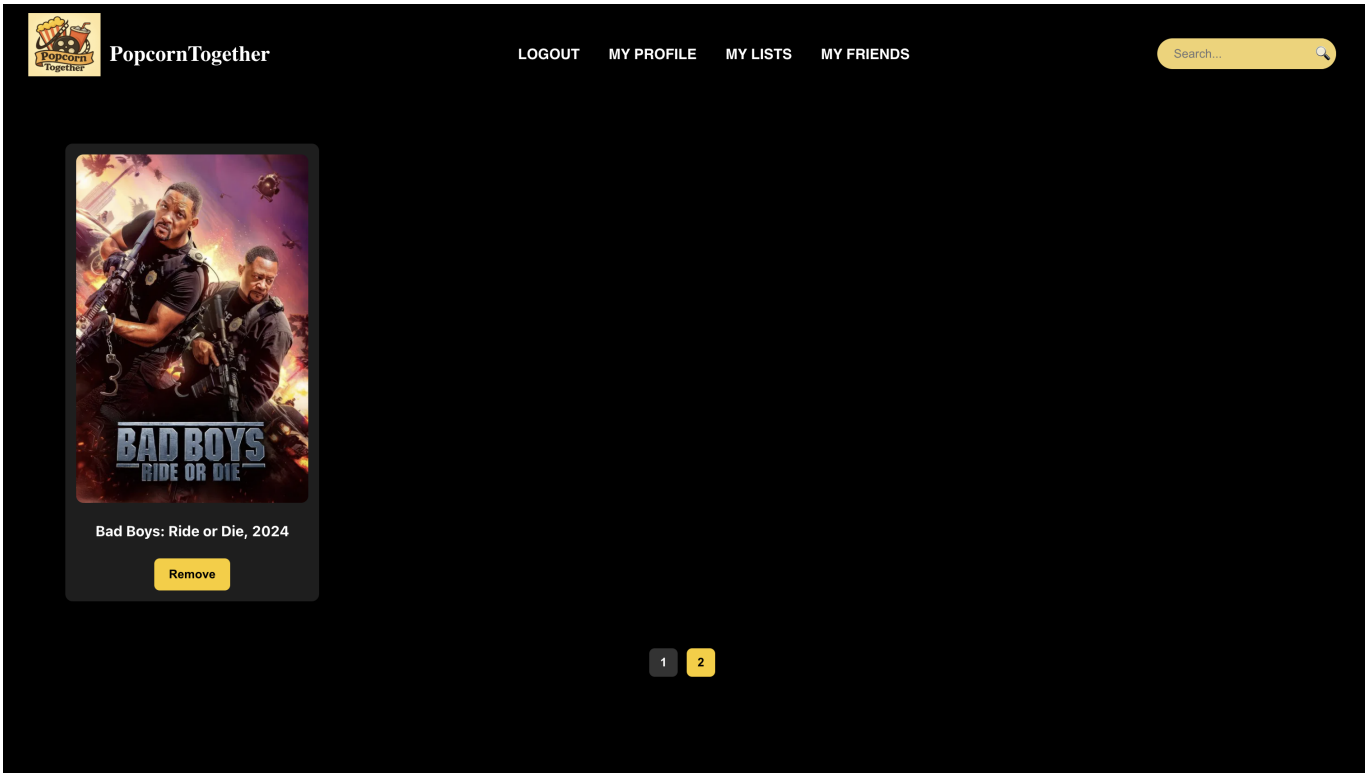
Removal from list

On the backend, thsis simply involves the removal of the respective Movie ID from the user's watchedlist array. When this occurs, the page does require a refresh to display the user's updated Watchedlist. An example of the before and after removal is shown below, note that this is without page refreshes.

Before removal



After removal



PopcornTogether also makes use of the movies added to the Watchedlist to compile statistics for our watch statistics feature. This collects information on the movies the user has watched to provide insight into their watch preferences and history. Hence, the watched list is one of the most essential parts of the logic flow for PopcornTogether.

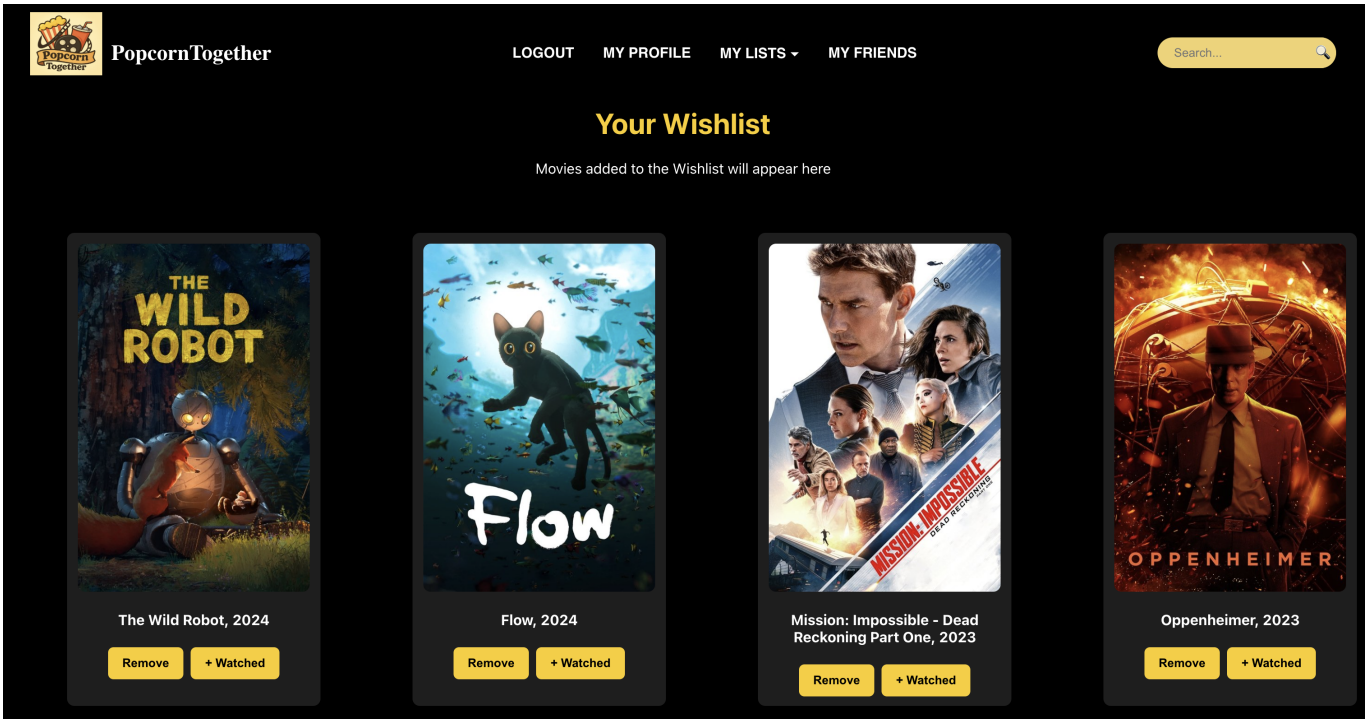
Wishlist

Core feature

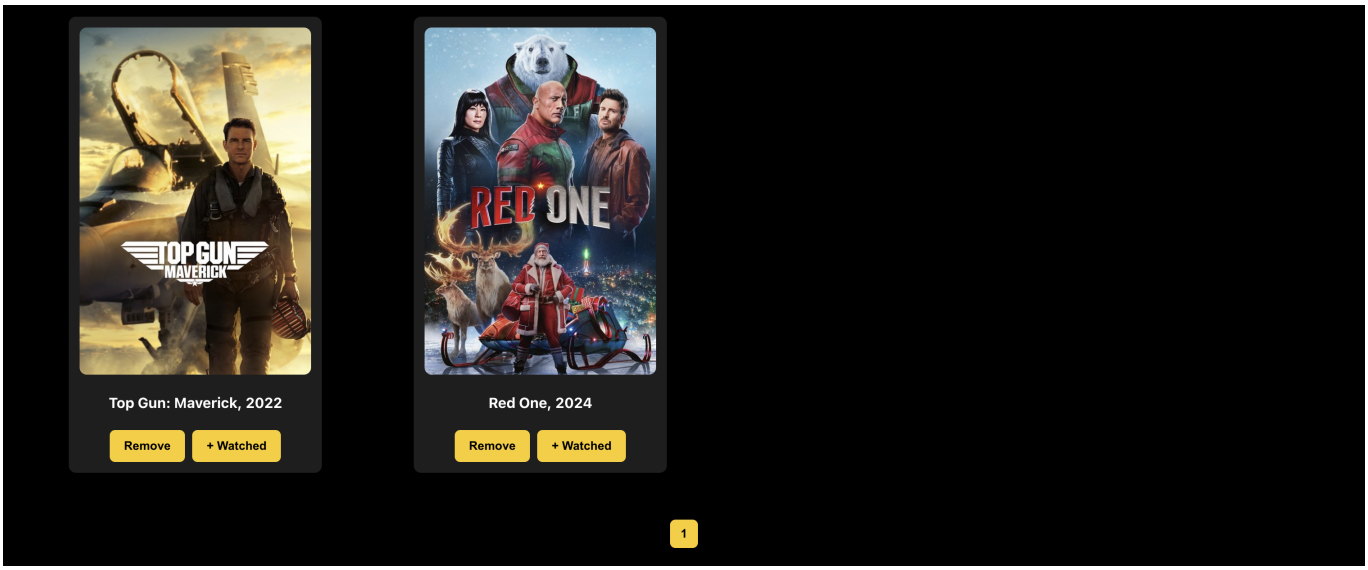
This page allows users to record down movies that they wish to watch in the future but do not have time for now. The purpose of this feature is:

1. Help the user track the movies they have not gotten around to watching.
2. Help friends discover common movies they can watch together

Wishlist



Wishlist with pagination



Similar to the watchedlist, movie IDs of films the user has marked as '+ wish' will be stored in MongoDb, then retrieved when rendering their personal Wishlist page.

Removal from List

As the Wishlist serves a different purpose to the Watched List, that is to aid the user in curating their bucket list of movies for future use, there will be an added functionality to the page. This will be the Add to Watched List function, represented by the '+ Watched button' simiilar to the button on the Results page. In addition to adding the movie to the Watched List, this button also removes the movie from the Wishlist, allowing users to tick off movies as they progress on their watching journey.

Wrapping up

Together, the Watchedlists and Wishlists address the user concern that were previously identified

> As a user who wants to remember what kind of movies I have watched and also want to watch.

Note : Future extension of this feature can include allowing the user different methods to filter their lists, and perhaps even including a search bar just for the lists.

Friends List

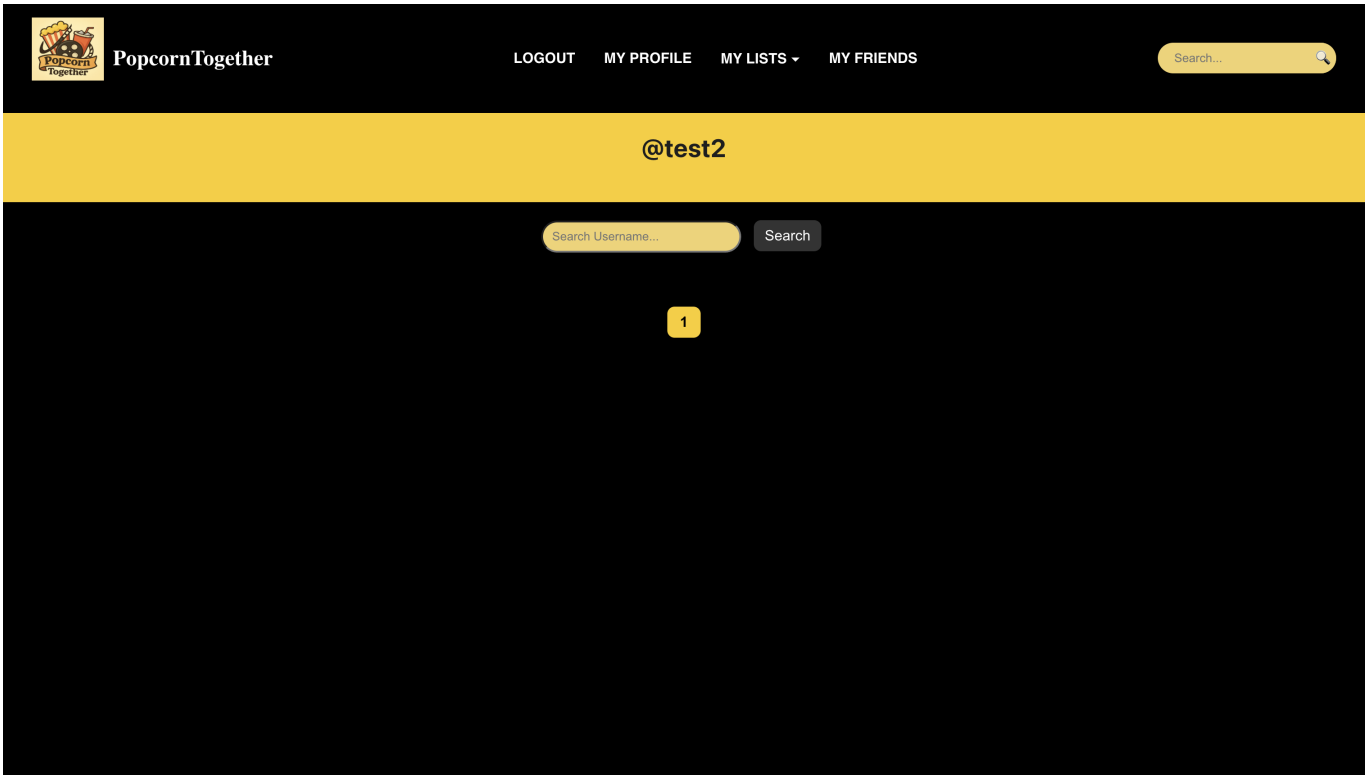
Core feature

Allow users to easily connect with others on the platform by adding them to their Friends List. This feature will allow them to:

1. View their friends' profiles.
2. See what their friends have recently watched and are planning to watch in the future.
3. Discover new titles based on their friends' interests.

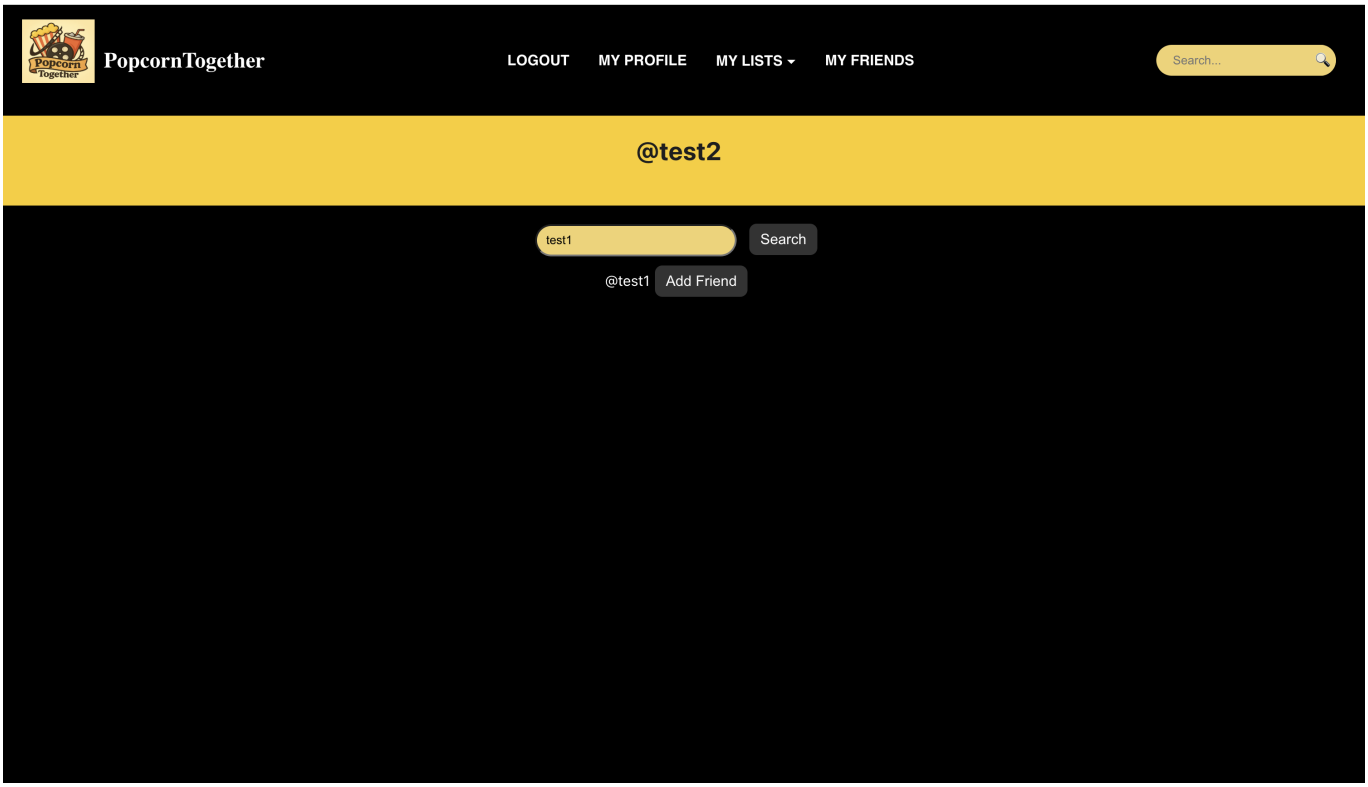
The purpose of the feature is to let friends connect their movie history as a way for each user to find a film from their friend's watched list, or to filter out movies that have already been watched. It also shows the common films that friends have in their wishlists so that they can find a film to watch together.

Friends List Page



This page will be rendered at first for new users and users who currently do not have anyone in their friends list. They will be able to access the search username function that searches for a user with a matching username. As usernames are unique, they will be able to definitively find their friends through this search function. They will not be able to search for themselves or an existing friend.

Add Friends



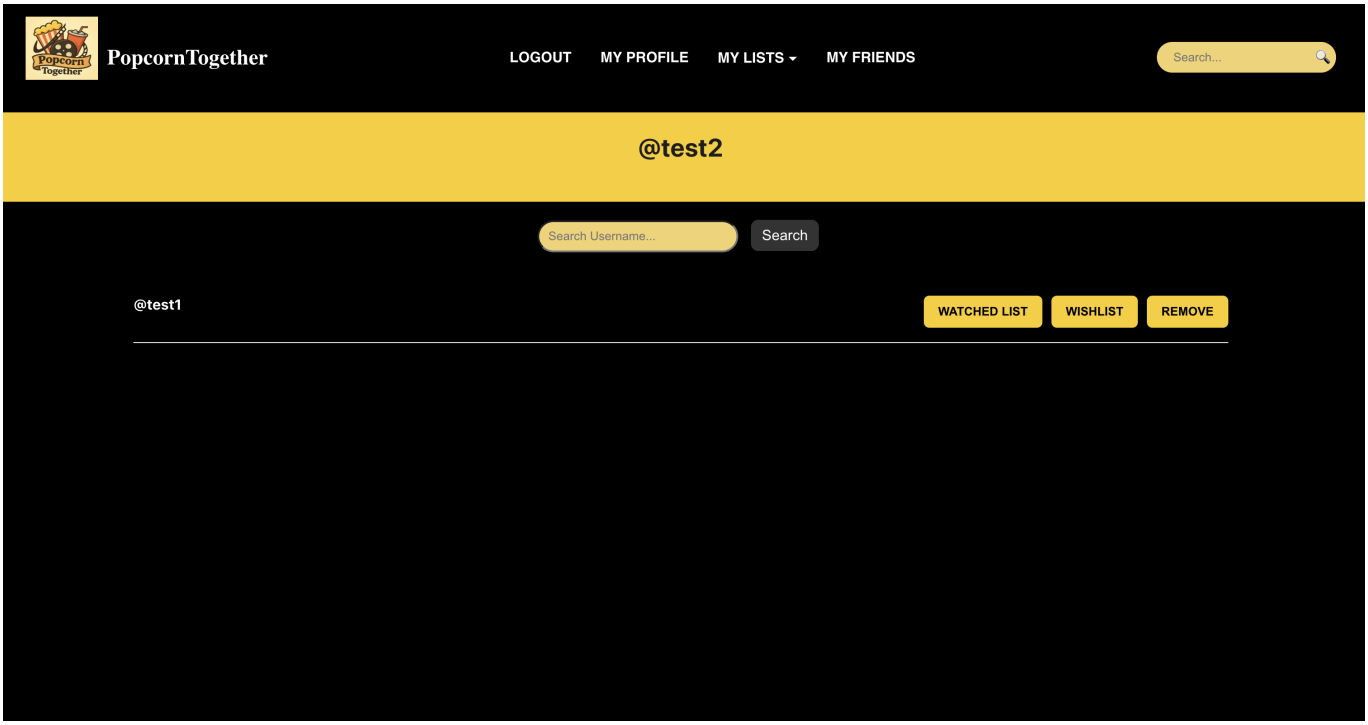
This shows the search result with the Add Friend option available. By clicking the button, the user will be able to add them to their friends list. The Friends list acts in a 'follower' style format whereby users are able to follow other users who they want to. This will not trigger a mutual follow. For example:

-> Test1 adds Test2

-> Test1 has one friend added to their friends list and that is Test2

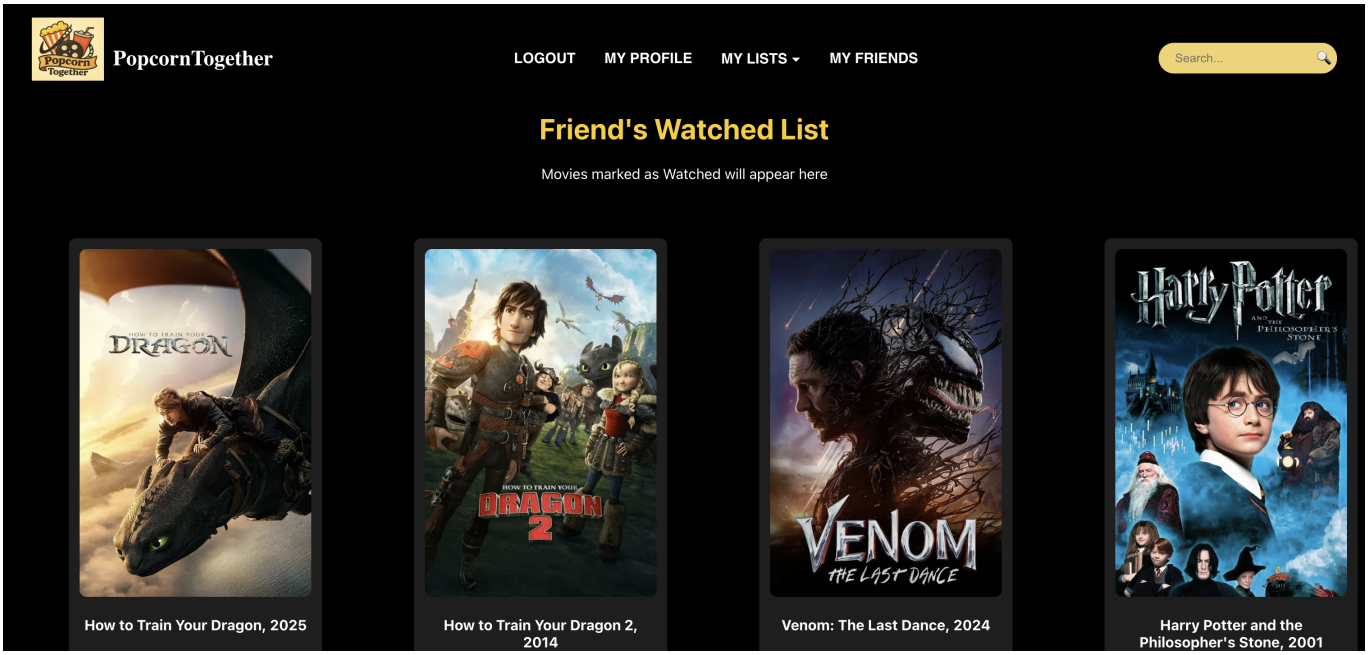
-> Test1 is not added to Test2's friends list

Added Friends



After adding a friend, the user will be able to perform several actions. The basic one would be to remove the friend as part of their friends list curation experience. The other functions are as we have described, to view the friends' watched list or wishlist to find movie inspirations or common films. An example is shown below.

Friends Watched List



This addresses the user concern we have identified

> As a user who wants to see what my friends are watching, I want to add them to my friends list to keep track and stay connected with them.

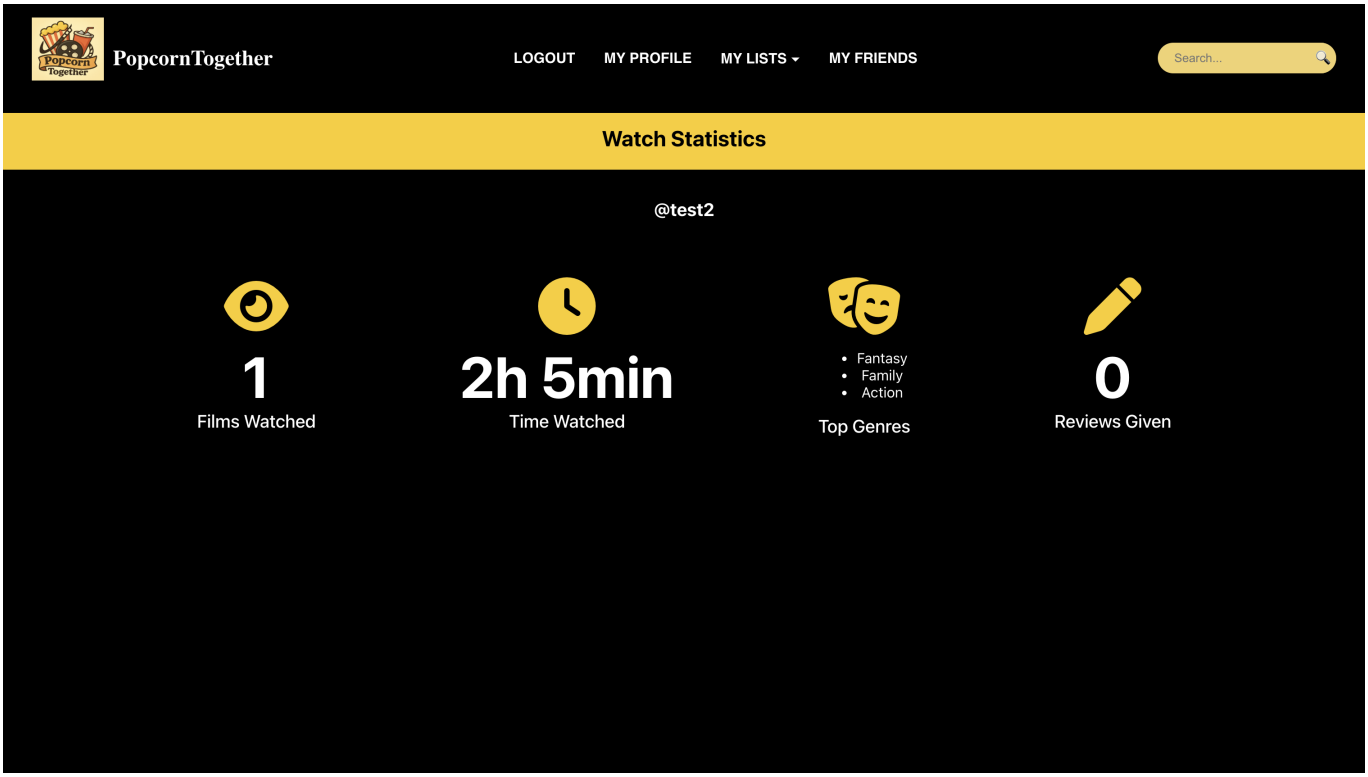
Note: future extension can involve an auto filter function or a separate page to instantly show users movies they have in common in both lists, this could be an add on to the current browsing feature users have.

Watch Statistics

Core feature

Watch statistics track your movie-watching habits with real-time statistics. Get insights on your most-watched genres, and time periods. See total runtime, number of movies watched, and average ratings. This will be the profile page that users can view for themselves.

Profile Page



This is an example of the profile page. Currently we have 4 intended statistics set up for the watch statistics. These are:

1. Films watched
    - This tracks the total number of films in the user's Watched List
2. Total time watched
    - This tracks the total runtime of all the films that the user has watched thus far
3. Top genres
    - This finds the top genres of films that users have watched thus far
4. Reviews given
    - This tracks the total number of reviews a user has given. More than one review can be given per movie.

The purpose of this feature is to present the user's viewing habits and preferences in a more objective data-oriented manner in the hopes of helping user's filter films based on what they have enjoyed in the past.

This addresses the concern we have identified :

> As a user who wants to find a movie that fits my mood and preferences today, I want to be able to type genres and other search criterias. I want to use the search bar to find movies that apply.

The Watch statistics can help the user understand a more objective take on the films they enjoy watching, aiding in their movie search journey.

Note : This feature can have more depth, currently we are planning to include the below statistics:

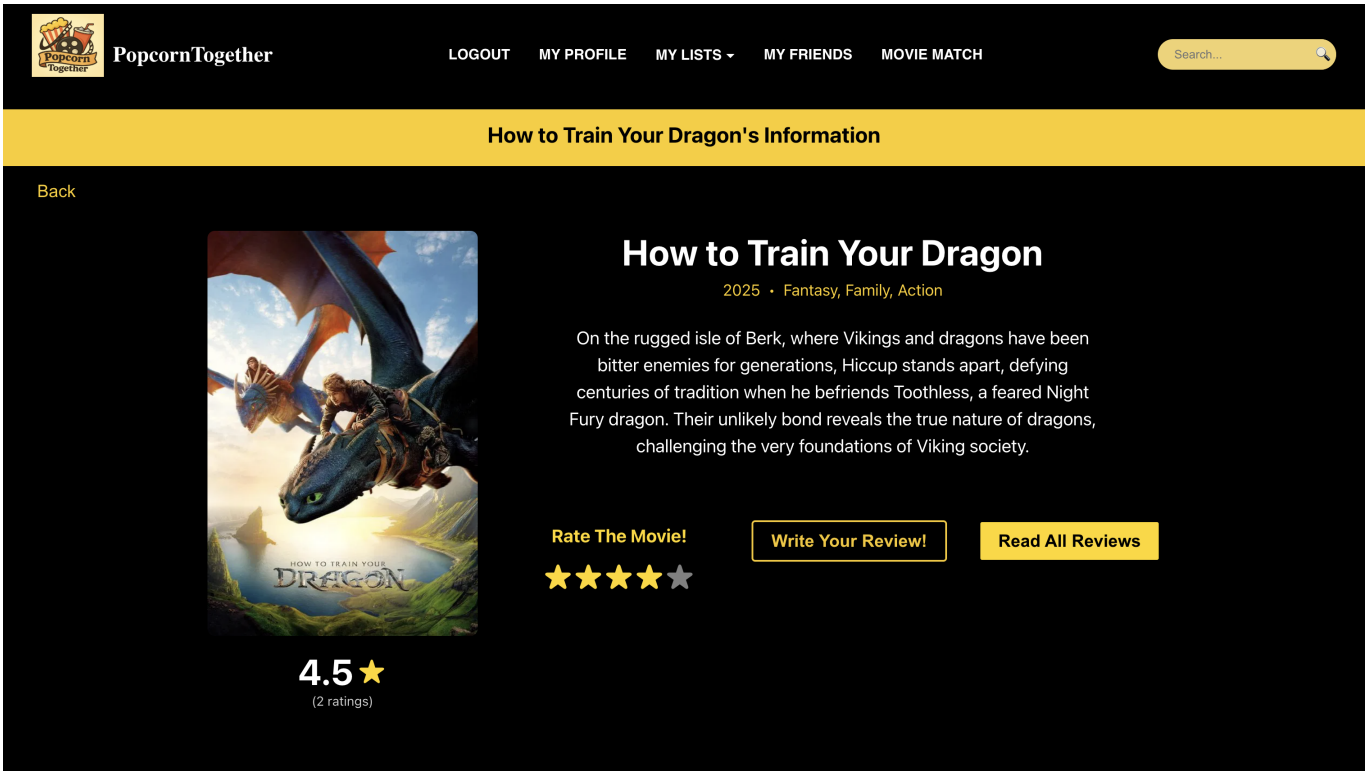
1. Top genres
2. Top time period
3. Average ratings
4. Average runtime

Community Reviews

Extension feature

Community reviews can be categorised into 2 types, namely a Star Rating based on a 5 star scale, followed by Review Writing for the movie. These features are accessed through the Header's search bar, where the search results have an 'Information' button that directs users to the Community Reviews Information page.

Community Reviews Information Page



Star Rating

For Star Rating on a 5 star scale, by collating the community opinions of a movie based with stars, users can quickly gauge the overall reception and quality of a film at a glance. This helps viewers discover highly-rated titles, avoid poorly received ones, and contribute their own opinions to shape the collective impression of each movie.

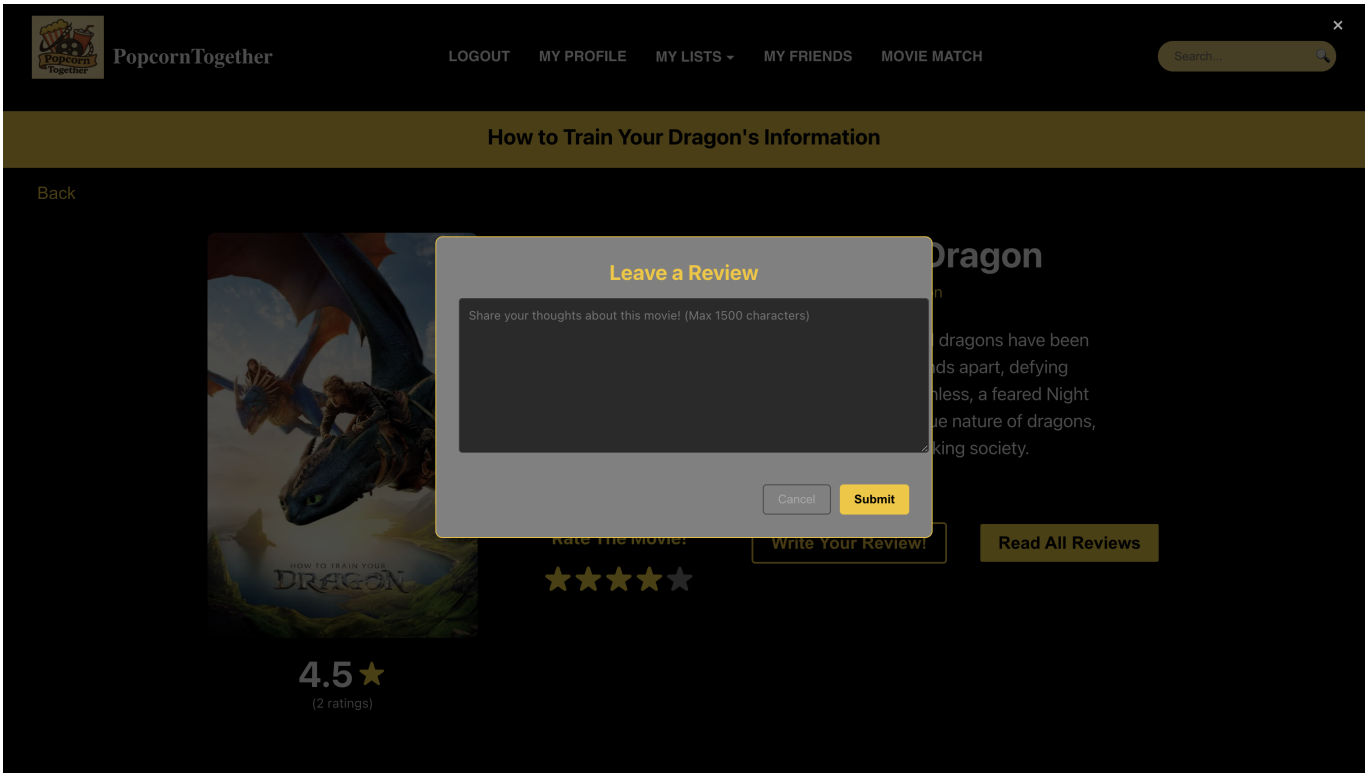
Star Rating Star Rating

Review Writing

Meanwhile, the Review Writing feature enables users to view comments from fellow users within the platform. This provides a more personalized and relatable perspective from many individuals, compared to generic critic reviews. Furthermore, critics tend to have a more skewed and distinct perspective towards movies and are not likely to give accurate, representative, or actionable information when it comes to subjective fields such as movie enjoyment.

Users can click the "Leave a Review" button to leave an unlimited amount of reviews.

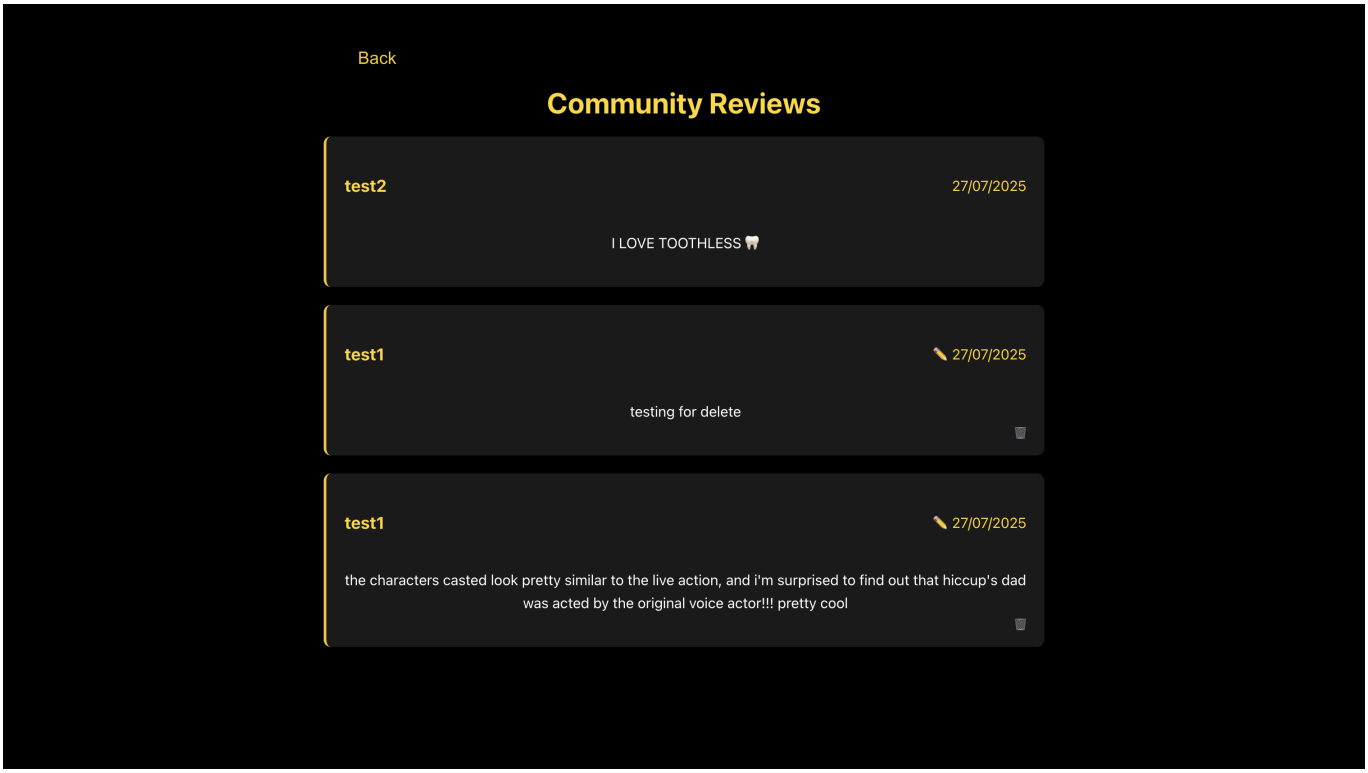
Leave a Review Popup



Average Star Rating AverageStar

To view a user's past reviews, as well as all existing reviews for a particular movie, users can click the "View All Reviews" button. Users will be directed to the ReviewsPage where they can either update or delete their own reviews in that page as well.

View All Reviews Page



By seeing what their friends or the broader community think, users can:

1. Make more informed choices about what to watch based on real, crowd-sourced experiences.

2. Gauge alignment with their own taste, especially when a review is from someone with similar preferences.
3. Avoid wasting time on low-quality or mismatched films, improving their overall viewing satisfaction.

This addresses the user concerns we have identified

> As a user who wants to see if a movie is worth watching or not.

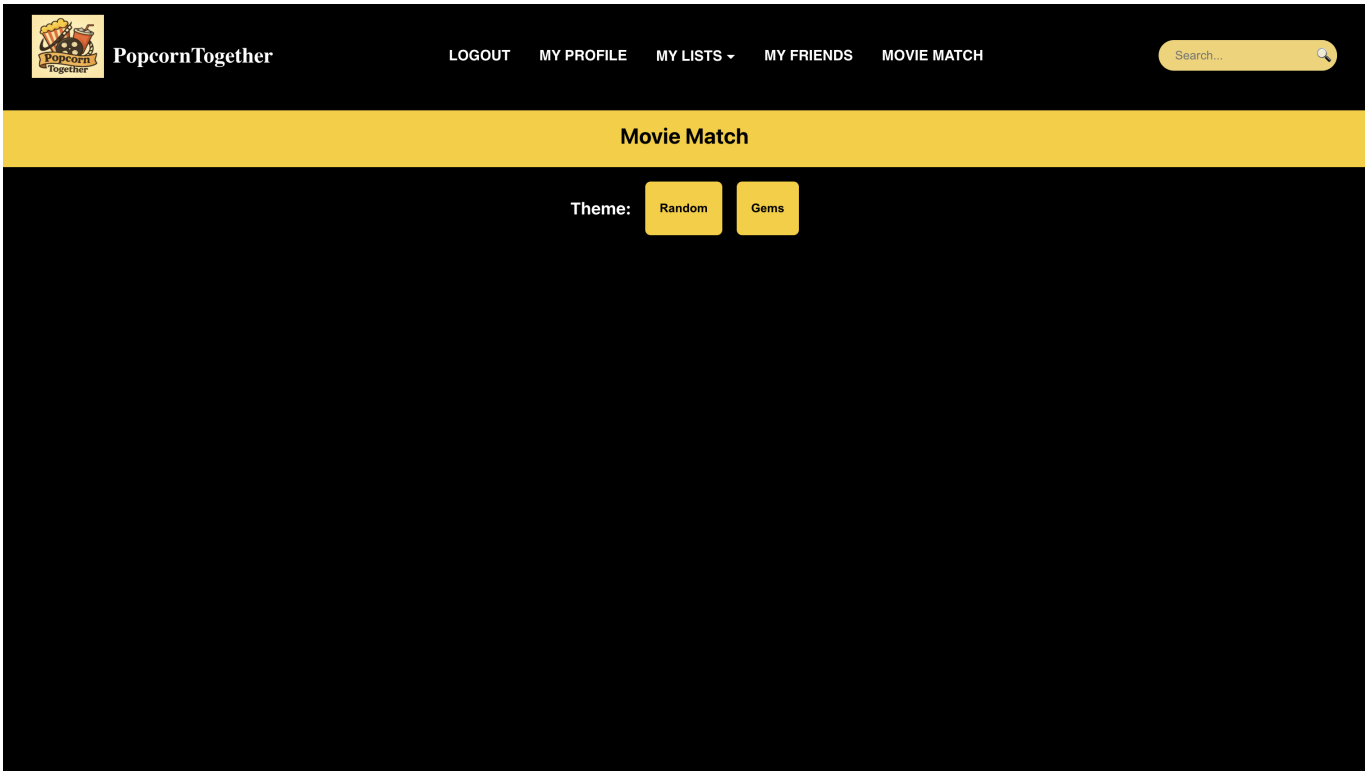
Movie Match

Extension feature

The Movie Match feature allows users to compare their wish list with their friends' wish lists to quickly find movies both parties are interested in watching. This simplifies the often time-consuming process of deciding what to watch together. This enables for streamlined coordination, eliminating the back-and-forth of suggesting and rejecting movie options. It also allows our website to be an all-in-one site for users, where they can finalise their group's movie decision in the same place as their own personal movie records.

The user can access the Movie Match page via the header component. This will render the basic Movie Match page below:

Movie Match Page Render

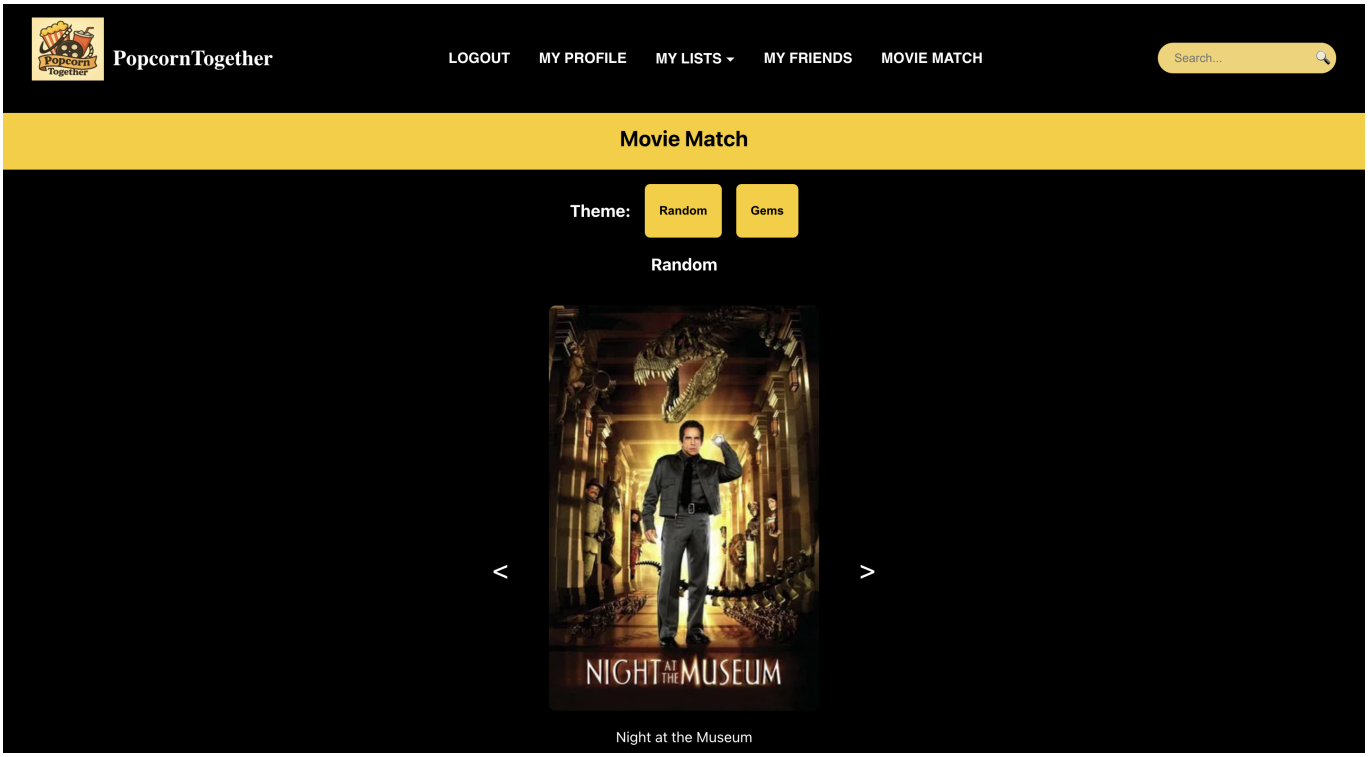


Currently, we have developed two 'genres' for users' random discovery. The first will be random, completely random films selected from TMDB without specifying any filters. Secondly will be gems, this will set filters to select only from a pool of most popular films from TMDB.

Upon selection, the below section will be rendered, we set this up to mimic dating apps where users can find 'the one' movie that appeals to them should they be unable to find inspiration for movies to watch.

This generates a collection of films below the selector. Users can add to their user lists, or view reviews directly from the movie card. They can also click left or right to view the next movies in the collection:

Movie Match page: random



Note: As an addition to this extension feature, we hope to provide Smart Recommendations in the future with further developments – if no match is found, our platform suggests similar titles based on genre, themes, or popularity among the users' networks.

This addresses the user concern we have identified

> As a user who wants to find a movie both my friends and I will enjoy.

Software engineering principles

Separation of concerns

Each layer of the application has a clearly defined role:

- Frontend (React.js) handles the user interface, routing, and rendering.
- Frontend (CSS) is used to style and layout components, ensuring a consistent, responsive, and visually appealing user interface across different devices and screen sizes.
- Backend (Express.js) is responsible for the functional logic, session management, and database operations.
- Database (MongoDB) stores persistent user data such as account information, watched/wishlist movie IDs, and friendlists.

By keeping UI, logic, and storage independent, the app is easier to maintain and modify without introducing bugs across unrelated features.

Modularity

PopcornTogether is designed such that each feature is encapsulated in its own component or route file. For example:

Frontend components like Header, Filter, and pages like LoginPage, and ResultsPage are separated into their own files, each responsible for a distinct piece of the UI.

Backend logic is organized into route modules such as authRoutes.js, friendsRoutes.js, and movieRoutes.js, allowing easier maintenance, scalability, and testing. This ensures clarity of purpose for each file, keeping the functionality of each file distinct. This approach makes the application easier to debug, extend, and collaborate on.

Don't repeat yourself (DRY)

The application avoids redundant code by abstracting repeated logic into reusable functions:

handleProtectedRoute() is reused to protect routes like /profile, /friends, and /lists, reducing duplicated authentication checks. Shared Axios configurations (e.g., { withCredentials: true }) are consistently applied across API calls using centralized options where possible. This helps improve code readability and minimizes the risk of inconsistent behavior.

The header and filter is also deployed in all pages, with the exception of the authentication pages. Hence, instead of repeating the code, it is mounted as a component using:

```
{
    <Header />
    <Filter />
}
```

Error handling

To ensure robust handling of different routes and functions, as well as easier debugging, PopcornTogether makes use of the below practices:

- Wrapping Axios API calls in try-catch blocks to handle server errors
- Using console.log to display error messages and endpoints for accurate triangulation of errors for debugging.

For example:

```
    const handleSearch = async (q) => {
        try {
            const res = await axios.get('SOME_ROUTE', {params : q});
            navigate('/SOME_OTHER_ROUTE', {state : {results: res.data}} );
        } catch (err) {
            console.error('Search failed', err);
        }
    }
```

- Specific status codes used for respective errors:

Status codes

Status Code	Meaning	When It's Used
200 OK	Success	Returned on successful GET or POST actions (e.g., login success, data fetched).
201 Created	Resource Created	After successful user registration.
400 Bad Request	Invalid Input	When required fields are missing or invalid (e.g., mismatched passwords).
401 Unauthorized	Not Authenticated	When accessing protected routes without a valid session (e.g., /me, /friends).
404 Not Found	Resource Not Found	When user/email is not found in login or DB queries.
409 Conflict	Duplicate Resource	When trying to register with an email or username that already exists.
500 Internal Server Error	Server Crash	When an unexpected error occurs (e.g., DB errors, bcrypt failures).

- Middleware function, isAuthenticated, intercepts unauthorised access by users without an active session before access to protected routes, such as friends list and profile, is granted.

Security management

As the app handles user data such as email, passwords, date of birth, and API keys, we have integrated a few layers of security integration:

- Password protection: All user passwords are securely hashed using bcrypt before being stored in MongoDB. They are not stored as their respective input strings.
- Session management: Sessions are established using express-session and stored securely in MongoDB via connect-mongo. Cookies are set as httpOnly to prevent client-side access.
- Route protection: Sensitive endpoints like adding to watched/wishlist or viewing friends are protected by middleware that checks for an active user session before allowing access.
- When authentication is required for database querying, using env files makes it easy to run the codebase from different environments. The env files are kept local with the git ignore file managing version control when running git push.

Sensitive information such as passwords and API keys are all stored locally and not included in any code lines.

Version Control

PopcornTogether uses **Git** for version control to ensure consistent and trackable development. All code changes are committed with commit messages, allowing for clearer workflow and separation of duties.

```
# Cloning repository
git clone https://github.com/username/PopcornTogether.git
cd PopcornTogether

# Make changes, then stage and commit
cd directory
git add .
git commit –m 'SOME_MESSAGE'

# Push changes to GitHub
git push

# Pulling each other's code from Github to continue edits
git pull
```

This is especially crucial given the remote nature of the collaborative work done for PopcornTogether. By implementing a version control system, changes are easier to trach and new implementations are easily traceable.

Commit Practices

All commits are accompanied by clear and concise descriptions to what work was done. This helps achieve 2 objectives:

1. Traceable points in our development history
2. Clear worklog for both parties

By utilising **Github** and maintaining proper commit practices and habits, we can avoid issues arising from untraceable bugs being pushed and requiring extra effort to address. This also avoids duplicate work streams and allows us to work on the project synchronously.

Clear Updates

After making a commit to Github, a follow up message is communicated between team members, covering a more detailed elaboration of work accomplished and changes made with the commit. This process makes changes even moer traceable, it keeps team members in the loop when it comes to the development of PopcornTogether, ensuring that team members are on the same page.

Software Development Life Cycle (SDLC)

Our team implemented an iterative approach to the development of PopcornTogehter. We chose an iterative approach to allow:

1. Continuous improvements with user feedback.
    - This is an essential process for the development of PopcornTogether. At its core, PopcornTogether serves as a Quality of Life (QoL) improvement tool for users. Hence, the user experience is a crucial part of developing both the front end and backend of this project.
2. Testing and validating features in small cycles.
    - This allowed us to better manage the functional components as well as the user interface components of PopcornTogether. By maintaing a routine of testing and validation, we are able to ensure the functionality of one component before moving on to the next or passing it on to a team member.
    - This is especially crucial for features that will be integrated or used in othe rparts of the Project.
3. Parallel progress on backend and frontend components.
    - Synchronous work done on backend and frontend components allowed for expedient testing upon completion. Rather than work on them separately, we chose to work on each feature as a singular unit.

Depth first implementation

> depth-first: an iteration focuses on fleshing out only some components.

The development of PopcornTogether followed a depth first implementation. Features were fully completed with basic user testing before beginning on the next feature. This ensured that team members were kept updated on every step of the development process, as well as ensure minimal bugs are present before beginning on the next stage of the project. As we seek to integrate certain features together, such as allowing the display of a friend's wishlist, or the use of wishlisted movies in determining watch statistics, fully completing a feature is crucial to the seamless development of the project.

SDLC Breakdown

Stage	What we did
1. Requirements	- Identified user requirements - Matched key features - Identified tech stack - created relevant API integrated
2. Design	- created individual rendering for each feature - created user schema
3. Implementation	- Fully built authentication features to allow for user level testing - Completed features by level of integration (eg. register => log in => add movies to wishlist => viewing wishlist)
4. Testing	-Executed after completion of each individual feature

Next steps	Plans
5. Deployment	- Use of Render to deploy the webapp
6. Extension	- Implementation of extension features
7. Refinement	- Add additional extension functionality for a robust Movie companion app
8. Final testing	- Ensure everything runs seamlessly - Optimisation

Testing Milestone 2: Errors encountered

This serves as a documentation of errors we have encountered during Milestone 2:

Frontend

The largest issue we faced was deciding how to integrate our most vital feature which is the movie search. We initally designed a standalone page for it. However, seeing as how many of the features will require the use of the search function in some way, we created the header component as a way of mounting the search function on all pages. This development also allowed us to tag on routes to other features such as the friends list, watched list, wishlist etc.

As we are relatively new to using html and css for such webpage designs, finding the correct keywords for the syntax of our frontend pages posed a monumental challenge. One resource we made use of was the Bootstrap library. We also made use of the npm start command to run our react app via localhost in our broswer. This enabled us to view changes to the webpage as we adjusted the css for the respective pages.

Using the Create-react-app functon gave us a quick jumpstart to creating a react app, it also provided structural syntax for our subsequent files.

We have also faced CSS selector conflicts while designing the frontend layout of our pages. As we used the same generic class name .movie-title on different pages like the Homepage, Disney, and Timeless Favourites. As a result, styles from one page sometimes overrode the intended styles for another, depending on the CSS load order. We resolved this by either increasing selector specificity — for example using .results-container .movie-title — or by giving page-specific elements unique class names.

Another general issue we encountered was the integration of frontend and backend. We made many mistakes regarding the use of axios, and the backend routes. We found that incorporating logs via the use of alert, console.log, and console.error helped us to debug these issues easier.

Frontend Testing: update

- Landing page: placeholders to be replaced (**DONE**)
- Authentication page: completed
- Register page: completed (possible user schema issue from backend)
- Login page: completed
- Profile: completed
- Watched List: Completed
- Wishlist: Completed
- Friends List: Completed
- Search function: Completed (to differentiate /search and /discover for TMDB routes)
- Results page: Completed

Backend

One of the main problems we faced on the backend was route naming inconsistencies. For example, the backend defined routes like /addWatched and /addWish, but sometimes the frontend incorrectly made

requests to /add-watched or /add-wish. This mismatch caused 404 Not Found errors that took time to trace.

We also had to ensure that our isAuthenticated middleware properly checked whether a user was logged in before allowing them to add movies to their watched list or wishlist, since missing this check would have allowed unauthorized actions.

Another issue was the ordering of functions in our server.js. There were many moving parts, such as the route imports, the database connection, the express session configruation. Initially, there was no clear order to each component in our server.js, resulting in several connection issues (Error 500). This was eventually resolved when we correctly ordered our express session before the route imports.

Backend Testing: update

- authRoutes: completed
- friendsRoutes: completed
- userRoutes: completed
- movieRotues: completed
- watchStatsRotues: completed
- user.js: possible unresolved friends field error, require more testing (**DONE**)
- server.js: fallback route unresolved (**DONE**)

Unresolved error : Completed

Below is the description of a bug we have fixed from Milestone 2:

Currently, we still have an unresolved error for a feature that we are intending to rectify. That is to include a fall back route '*' such that refreshing of pages will succesfully load the page. At present, the inclusion of our fallback route leads to 'path-to-regexp' errors.

Database

As this was our first time integrating a project using a database, we faced difficulties in finding out how to integrate database functionalities with PopcornTogether. Fortunately, the MongoDb youtube channel provided many tutorials with walkthroughs on setting up a working MongoDb database cluster for personal projects. After which, we just had to adapt our user schema in user.js to suit our intended functions.

Since our authentication relies on session IDs, you needed to ensure that our database queries safely pulled the correct user based on req.session.user.id. Any session handling bugs could easily result in updating the wrong document or rejecting valid requests, so robust session and user ID management is essential when updating user-specific lists in the database.

An issue we faced when initialising our user schema was assigning unique:true to the friends array. The intention was to ensure friends added were unique and could not be added twice. However, this created an issue. Unique actually enforced the friends field to be unique amongst user -- no two users can have the exact same friends list. A unique index in MongoDb ensures that no two documents in the collection can have the same value for that field. This resulted in the backend throwing errors when:

1. adding or removing friends
2. Trying to create a new user

The issue being that there was already a user with an empty friends array. The fix was to remove the unique:true, and manually delete the friends index from MongoDb that was created using this unique tag.

The database configuration was largely seamless owing to the detailed resources provided by MongoDb for the deployment of MongoDb atlas in projects.

User.js

Unique field input for friends still being enforced despite user schema rectification and dropping the index.

Deployment

As this was our first time deploying a web app, we enconutered a wide range of issues.

Firstly, we initially wanted to deploy the frontend and backend separately with the frontend being on Vercel and the backend on Render. This gave rise to an issue with our express session as both sites assigned their own respective domains. Since the domains were different, the cookies that were used for our session tokens were not sent across domains. As such we had two options, to deploy both frontend and backend together, or purchase a custom domain for upwards of $18. We decided to instead deploy the entire repository on Render as a webservice.

During deployment, we encountered a net::ERR_CONNECTION_REFUSED error when attempting to access backend routes (e.g., /api/login). This occurred because the frontend was trying to reach the backend on localhost:5050, which only works locally. We resolved this by updating the API base URL to point to the deployed backend server and ensuring the backend was hosted and running before the frontend was built.

The deployed code required slight tweaks to our frontend backend communication as we were no longer using localhost for our testing. This included changes to all our pages and component functions, as well as our server.js. An example of an issue that caused our deployment to crash was the hardcoding of PORT for our local testing. Before discovering that Render would assign its own PORT, we had given it a hardcoded value of 5050, which resulted in our deployment timing out.

Overall, the summary of our deployment errors encountered and fixes, centers around small code adjustments when moving from local testing to Render, and figuring out how Render deployments worked.

Conclusion

Despite challenges faced during the setup and deployment for Milestone 2, we have managed to successfully develop our core features for PopcornTogether, barring some fringe functionalisties that can be completed as extension work to complement the existing web app.

Our next steps will be to debug the unresolved errors, including the fringe functionalities to complete our core app, and add in our two extension features.

Testing

In Milestone 3, we conducted multiple types of tests together. This allows our test suites to complement each other. These include:

1. Unit Testing
2. Integration Testing
3. User Testing

Unit testing

In order to conduct unit testing, we focused on complementing the user testing by introducing edge cases rather then simply relying on the user experience and 'Happy Path' testing. This involved the below steps:

1. Creating test cases according to:
    - Partition Equivalence [PE]: divides inputs into distinct equivalence classes where each class is expected to behave similarly. For each class, a representative value is tested instead of every possible value
    - Test Boundaries [TB]: focus on values at the edges of valid input ranges, which are more prone to bugs. For example, empty string on maximum length strings
    - Negative Cases [NC]: verify that the application gracefully handles invalid or unexpected inputs and doesn't break or behave unpredictably.
2. Used Jest to arrange and run test cases. The test cases were arranged using the AAA testing pattern.
    - AAA (Arrange-Act-Assert)
    - Arrange: Setting up test environment and mock objects
    - Act: Invokes the functionality to be tested
    - Assert: Compares the results of the function to the expected outcome
3. Ran frontend unit testing using the npm test command

The labels for the relevant testing heuristics are:

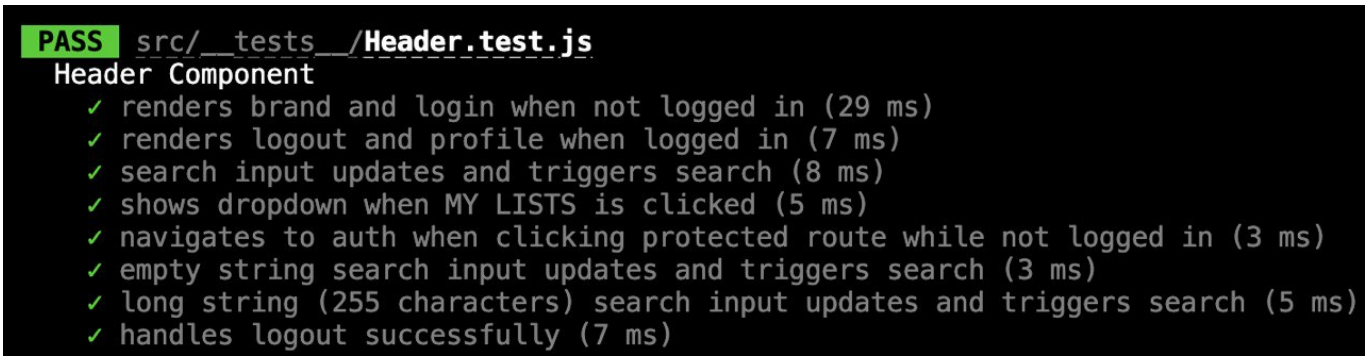
- Partition Equivalence (PE)
- Test Boundary (TB)
- Negative Case (NC)

We set up our test cases for each component below:

Header:

- 'Renders LOGOUT button when Logged in' [PE]
- 'Renders LOGIN button and redirects to authentication page when Logged out' [PE]
- 'Empty string search input updates and triggers search' [TB]
- 'Long string (255 characters) search input updates and triggers search' [TB]
- 'Handles LOGOUT successfully'

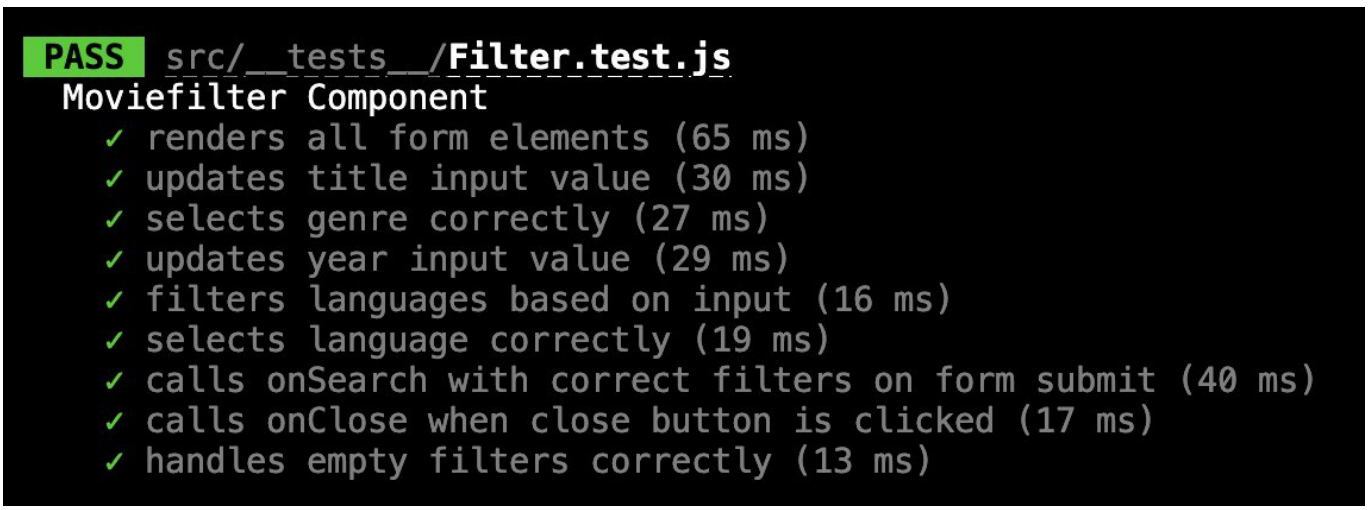
Result:



Filter:

- 'Renders filter component'
- 'Updates filter correctly' [PE]
- 'Calls onSearch with the correct filters on form submit'
- 'Handles empty filters' [TB][NC]

Result:



We set up our test cases for each Page below:

AuthPage:

- 'Renders page correctly'
- 'Passes valid email field to register page successfully' [PE]
- 'Passing empty string for email redirects to register page successfully' [TB][NC]
- 'Log In Button Redirects to login page successfully'

Result:

```
PASS src/__tests__/AuthPage.test.js
  AuthPage Component
    ✓ Renders page correctly (17 ms)
    ✓ Passes valid email field to register page successfully (3 ms)
    ✓ Passing empty string for email redirects to register page successfully (3 ms)
    ✓ Redirects to login page successfully (2 ms)
```

RegisterPage:

- 'Renders page successfully'
- 'Registers with valid entries' [PE]
- 'Does not register if required field is empty' [NC]
- 'Rejects invalid email format' [NC]
- 'Rejects submission if username/email already exists (mocked backend)' [NC]

Result:

```
PASS src/__tests__/RegisterPage.test.js
  RegisterPage Component
    ✓ Renders page successfully (67 ms)
    ✓ Registers with valid entries (23 ms)
    ✓ Rejects invalid email format (17 ms)
    ✓ Valid registration with optional field left blank (20 ms)
    ✓ Rejects submission if username/email already exists (70 ms)
```

LoginPage:

- 'Renders page successfully'
- 'Login successfully with correct parameters and redirects to homepage' [PE]
- 'Login successfully with uppercase letters and extra spaces in email' [PE]

Result:

```
PASS src/__tests__/LoginPage.test.js
  LoginPage
    ✓ Renders page successfully (79 ms)
    ✓ Logins successfully with correct parameters and redirects to homepage (71 ms)
    ✓ Login successfully with uppercase letters and extra spaces in email (13 ms)
```

HomePage (Landing page):

- 'Renders page correctly'
- 'Renders movie banner, latest movies, timeless favourites, friend activity and popular franchises correctly' [PE]

- ‘Redirects to timeless favourites page correctly’
- ‘Redirects to franchise page correctly’
- ‘Renders default image for Friend Activity when Logged out’ [PE]
- ‘Renders Friend Activity when Logged in’ [PE]

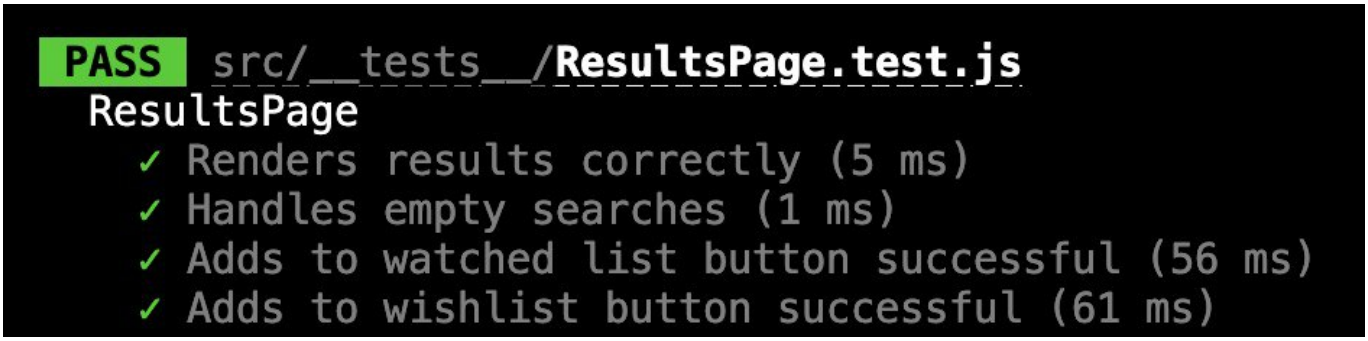
Result:

```
PASS src/__tests__/HomePage.test.js
  HomePage
    ✓ Renders page correctly (73 ms)
    ✓ Renders movie banner, latest movies, timeless favourites, friend activity and popular franchises correctly (27 ms)
    ✓ Redirects to timeless favourites page correctly (16 ms)
    ✓ Redirects to franchise page correctly (23 ms)
    ✓ Renders default image for Friend Activity when Logged out (11 ms)
    ✓ Renders friend activity when Logged in (10 ms)
```

ResultsPage (Movie Search Results):

- ‘Renders results correctly’
- ‘Handles empty searches’ [TB][NC]
- ‘Adds to watched list button successful’ [PE]
- ‘Adds to Wishlist button successful’ [PE]
- ‘View Community Reviews Information Page successful’ [PE]

Result:



WatchedlistPage and WishlistPage:

- ‘Correctly renders movies in Watchedlist’ [PE]
- ‘Renders page if list is empty’ [TB]
- ‘Case when user is Logged out but somehow lands on Watchedlist page’ [NC]
- ‘Renders friends list when id param exists’
- ‘Remove button removes movie from list’

Result:

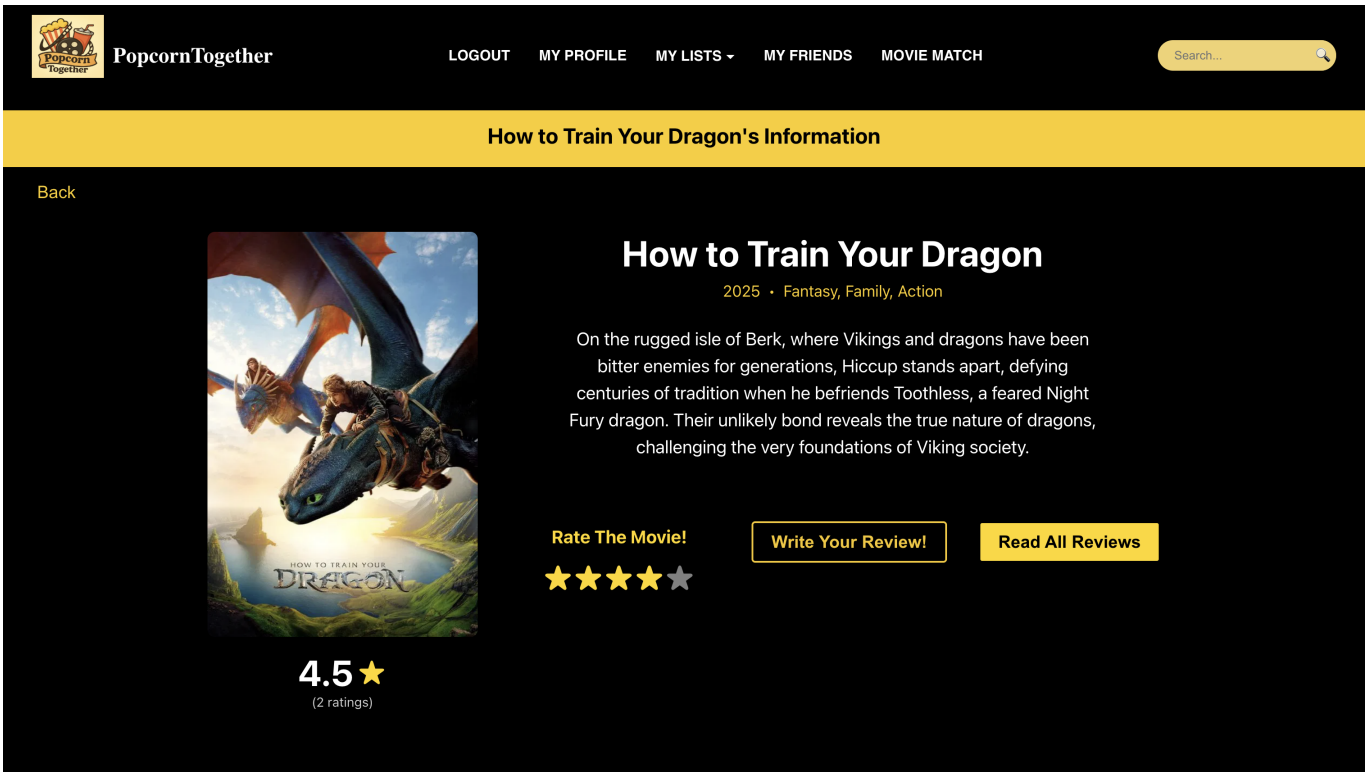
```
PASS src/__tests__/WatchedListPage.test.js
  WatchedListPage
    ✓ Correctly renders movies in watched list (172 ms)
    ✓ Renders page if list is empty (16 ms)
    ✓ Case when user is logged out but somehow lands on watched list page (13 ms)
    ✓ Renders friend's watched list when id param exists (6 ms)
    ✓ Remove button removes movie from list (10 ms)
```

Note: The functionality of Watchedlist and Wishlist is similar with a few key distinctions.

CommunityReviewsPage (Movie Information):

- ‘Correctly renders movie information, Star Rating button and Review button‘ [PE]
- ‘Correctly renders average Star Rating value‘ [PE]
- ‘Add Star Rating button successful‘ [PE]
- ‘Add Valid Review successful‘ [PE]
- ‘Add Very Long Review‘ [TB]
- ‘Add Empty Review unsuccessful‘ [NC]
- ‘Visit ReviewsPage successful‘ [PE]

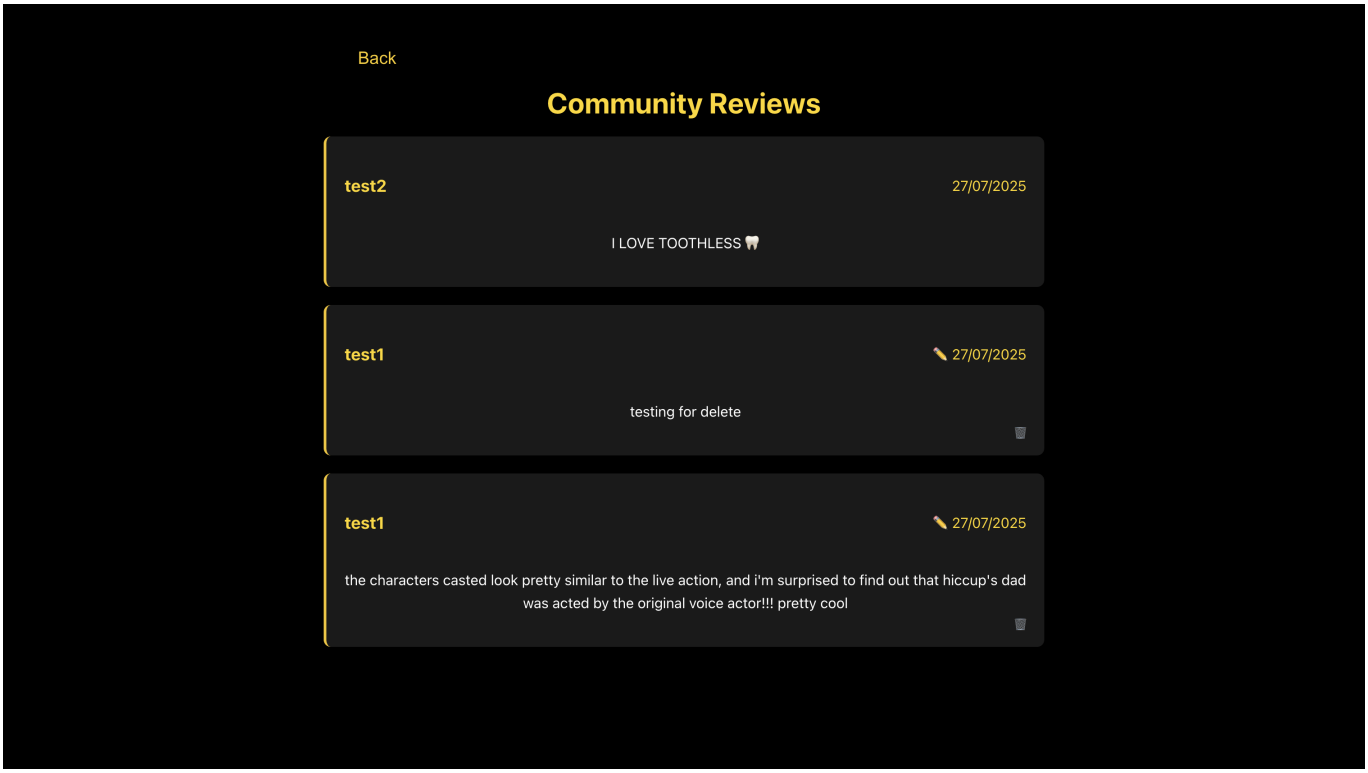
Result:



ReviewsPage:

- ‘Correctly renders all reviews' information‘ [PE]
- ‘Edit/Delete user's own review‘ [PE]
- ‘Edit/Delete another user's review‘ [NC]

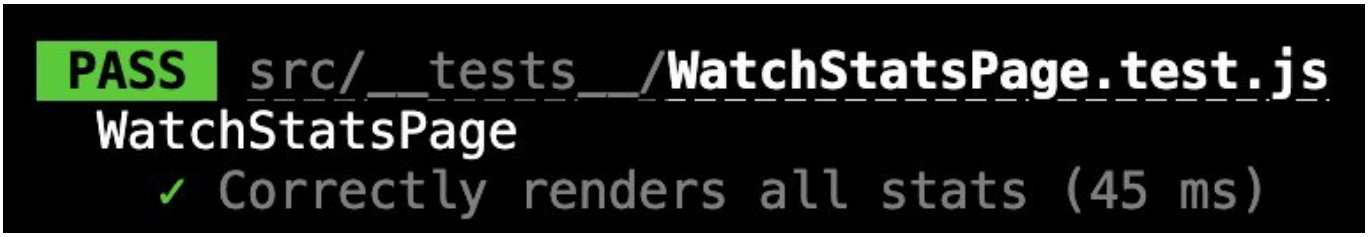
Result:



WatchStatsPage (Profile):

- 'Correctly renders all stats' [PE]

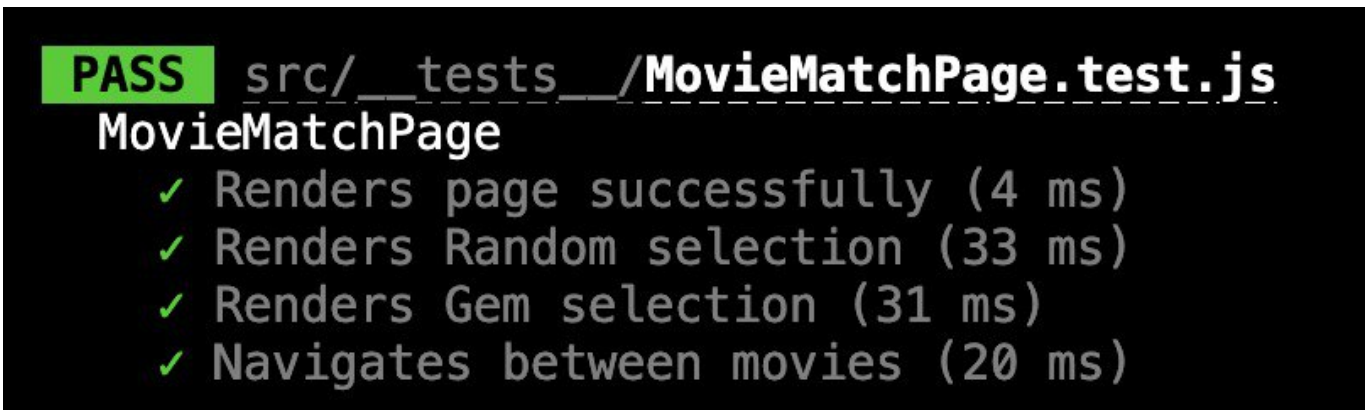
Result:



MovieMatchPage:

- 'Renders page successfully'
- 'Renders Random selection' [PE]
- 'Renders Gem selection'
- 'Navigates between movies'

Result:



Summary:

The components are able to pass our test suites, although not comprehensive, we have tried to include edge cases to account for unexpected behaviour.

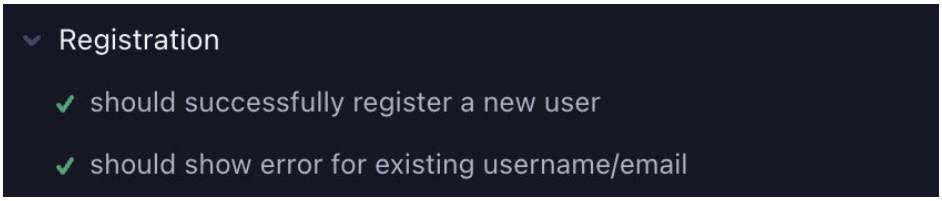
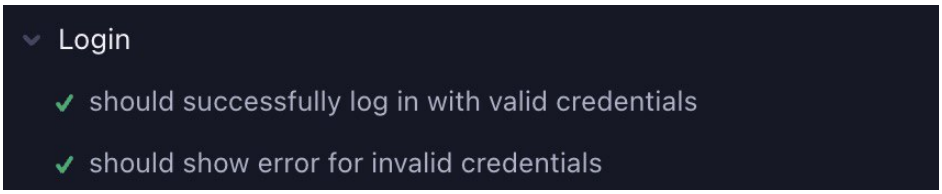
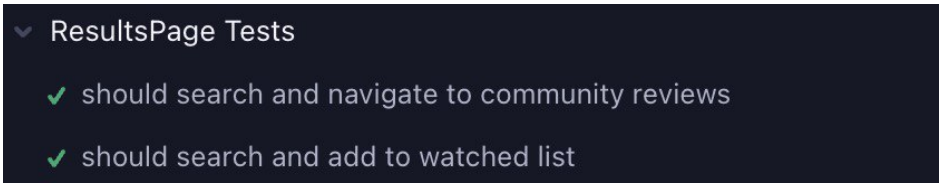
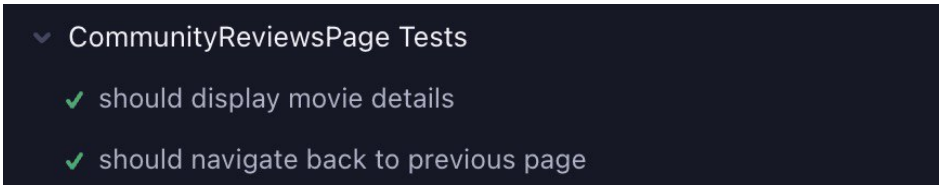
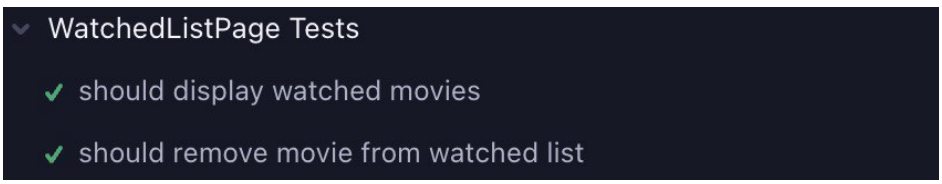
In particular, we made a few changes to our functionality as we had intially failed a few test cases:

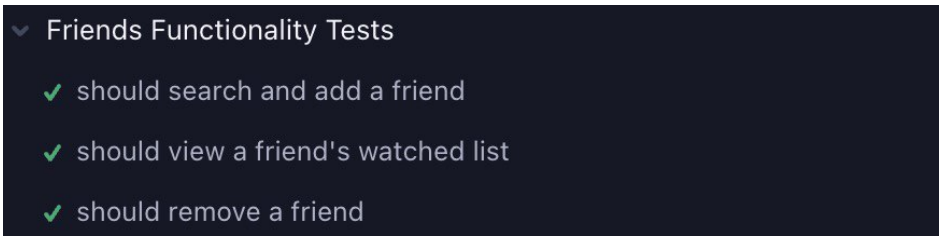
Location	Test case	Addition
RegisterPage	- ‘Rejects invalid email format’ [NC]	Add email regex for registration
LoginPage	- ‘Login successfully with uppercase letters and extra spaces in email’ [PE]	add email normalisation for case insensitivity (backend)

Integration Testing

We used Integration testing to complement the user testing and unit tesitng and it is used to simulate and mimic user interactions with the app. We made use of Cypress to conduct out integration testing as it allows us to build, test and run our test cases directly in the browser. We set up the test cases as:

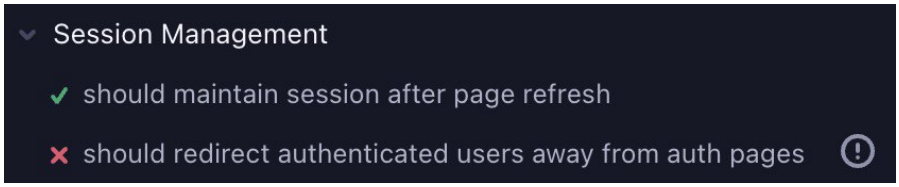
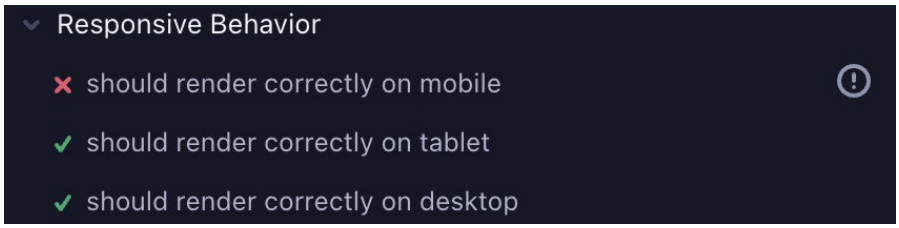
1. Registration
2. Logging in
3. Searching for movie
4. Viewing review for a movie
5. Using watchedlist
6. Using friend's list

Integration tests	Result
1. Registration - 'should successfully register a new user' - 'should show error for existing username/email'	
2. Logging in - ‘should successfully log in with valid credentials’ - ‘should show error for invalid credentials’	
3. Searching for movie - 'should search and navigate to community reviews' - 'should search and add to watched list'	
4. Viewing review for a movie - 'should display movie details' - 'should navigate back to previous page'	
5. Adding movie to watched list - 'should display watched movies' - 'should remove movie from watched list'	

Integration tests	Result
6. Adding friend - 'should search and add a friend' - 'should view a friend's watched list' - 'should remove a friend'	

We also coducted test on two other areas for overall app robustness:

1. Session Management
2. Responsive Behaviour

Integration tests	Results
1.Session Management - ‘should maintain session after page refresh’ - ‘should redirect authenticated users away from auth pages’	
2. Responsive Behaviour - ‘should render correctly on mobile’ - ‘should render correctly on tablet’ - ‘should render correctly on desktop’	

Significant findings

From our integration tests, we had two test case failures.

Failure	Next Steps
‘should redirect authenticated users away from auth pages’	Frontend functionality will have to be added to check for user authentication in the relevant Authentication pages (Auth Page, Register Page, Login Page). This functionality can be the same as the ones already set up in the landing page.
‘should render correctly on mobile’	Our app is actually set up and intended for desktop use, in future we can add better responsiveness for mobile integration.

User Testing

User testing was conducted to emulate the behaviour of our users when ineracting with the app. Below is the summary of the test cases:

Test	Steps	Feedback
1. Registration and login	1. Click on login button on header 2. Enter a email in valid format 3. Click continue 4. Register with valid details 5. Click Login with existing account 6. Login with the same credentials	Able to register and subsequently verify successful registration by logging in with the same credentials
2. Accessing Timeless Favourites page and popular franchises pages	1. Click on the Timeless Favourites banner 2. Return to homepage by clicking PopcornTogether in Header 3. Click on each of the respective franchise icons	Able to view each page and reroute to homepage via the header
3. Adding movies to watched list and wishlist from the home page	1. Attempt to add a movie each from latest movies section, and friends activity section to watched list and wish list 2. Click on My Lists and check each list to verify successful addition	Addition to watched list and wishlist succssful, navigation to each list successful

Test	Steps	Feedback
4. Search for a specific film, check the reviews and add to watched list from results page	1. Using the search bar, search for 'star wars' by title 2. Click on reviews 3. Click back on the community reviews page 4. Add the first movie to watched list 5. Verify on Watched list page	Search for star wars movies successful, able to view the reviews. Clicking back on the community reviews page preserves the search results and add the respective movie to watched list from the results page
5. Specify filters to search for movie, add to wishlist from the results page	1. Click on the search bar to render filter bar 2. Specify genre, year and language 3. Click search 4. Add a movie to the wishlist	Able to search for movies related to the filters specified. Able to add movie to wish list. Additionally, running searches for less popular filters will yield results and render results page successfully.
6. Check a movie in your wishlist as watched	1. Navigate to Wishlist page 2. Click on + Watched button for a recently watched movie	Movie marked as watched appears in watched list page. Movie is no longer in wish list.
7. View stats in my profile	1. Log out and then Log in to a fresh account 2. View My Profile to check for empty statistics. 3. Control the films watched, runtime and top genres by iteratively adding new movies to watched list and checking	Able to logout of account and be redirected to the authentication page. Able to view the My Profile page. My Profile renders the correct statistics for 0 films watched. My Profile page renders the correct statistics when new movies are added to the watched list.
8. Add a friend and view their lists, then remove the friend	1. Navgiate to My Friends Page 2. Search for test1 user and add them twice. 3. View friend's watched list and wishlist 4. Remove friend	Able to navigate to My Friends page. Able to add friend. Trying to add friend twice fails as intended. Able to view friends' lists. Able to remove friend.

Test	Steps	Feedback
9. Check the Movie Match function	1. Navigate to Movie Match page 2. Click on Random 3. Swipe left and right to see different movies 4. Add to lists and view reviews 5. Repeat for Gems	Able to access Movie Match Page. Random and Gems are able to produce distinct collections of movies. Able to view reviews and add movies to list from this page.

Final Deployment testing

By repeating our user testing for the final deployment, we were able to test the build online. In addition to the above tests, we also included a few more:

1. Adding the homepage featured movie banner to watched list
    - missed out during first round of local user testing
    - discovered error in the functionality (**FIXED**)
2. Refreshing the browser on pages other than homepage
    - testing fix to fallback route
    - works as intended

Apart from the new additions, every other test case managed to pass, the deployed app serves our intended functionality.

Known bugs

Below is a list of outstanding bugs discovered during our final tests after deployment:

Feature	Bug	Next steps
1. Results Page	Unable to run a new search when on results page	Will have to debug
2. Filter component	The title field in the filter component is redundant and does not run the search correctly	Remove title field and retest filter component

Conclusion

PopcornTogether is an app created for movie lovers. It features a wide range of movie discovery tools and serves as your trusty movie log. Discover new movies to watch with friends, or simply hijack their lists for inspiration!

Below are the completed list of features that PopcornTogether provides:

- Movie Search
- Movie Discovery
- Personal Profile & Watch Statistics
- Watchedlist
- Wishlist
- Friends List
- Community Reviews
- Movie Match

Next Steps

As we reach the end of the Orbital program, we still have further steps we can take to improve upon the app.

1. Debugging known issues
2. Cleaning up the UI

Other improvements we can make to the app are listed below:

1. Making the app more responsive to different screen sizes
2. Add more movie match selectors
3. Conduct further testing