Orbital 25 Milestone 2

Popcorn Together

Apollo 11

Gan Ting En

Choo Kai Xi Kasey

Table of Contents

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

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 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
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

Pain Point	App Requirements
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
- **Milestone 2 (Week 4 - 8)**: Prototype
    - Database integration added
    - Querying of database implemented
    - A working system with the core features
        - Account registration
        - Movie Search
        - Personal Profile
        - Watchedlist
        - Wishlist
        - Watch Statistics
    - User has edit access for the watchedlist
    - The app produces a list of movies given a set of filters, after search by user
- **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
    - 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

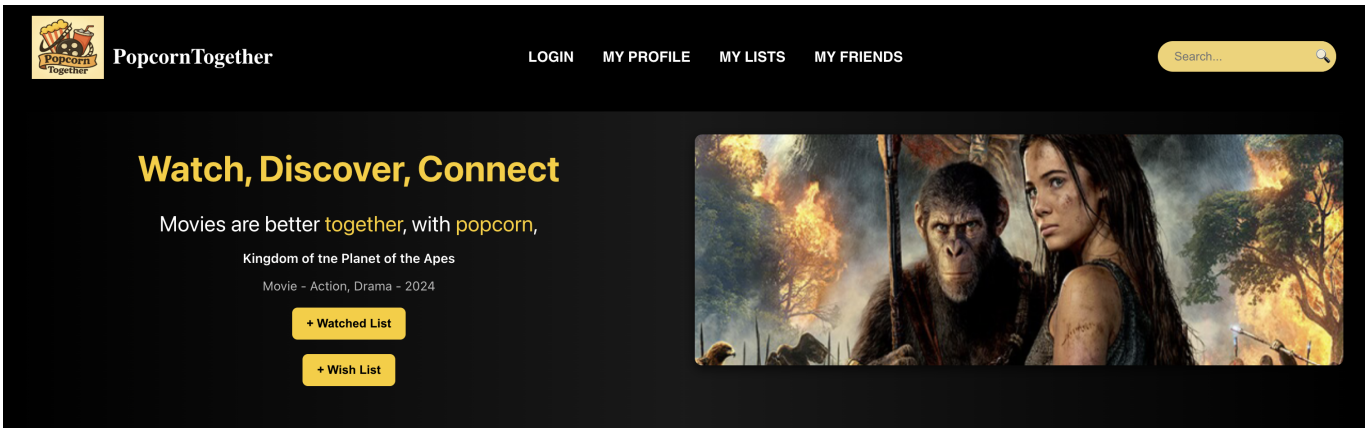
Features

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
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.
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	Gives users a better idea of what they can expect from movies they are interested in
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

Feature Descriptions

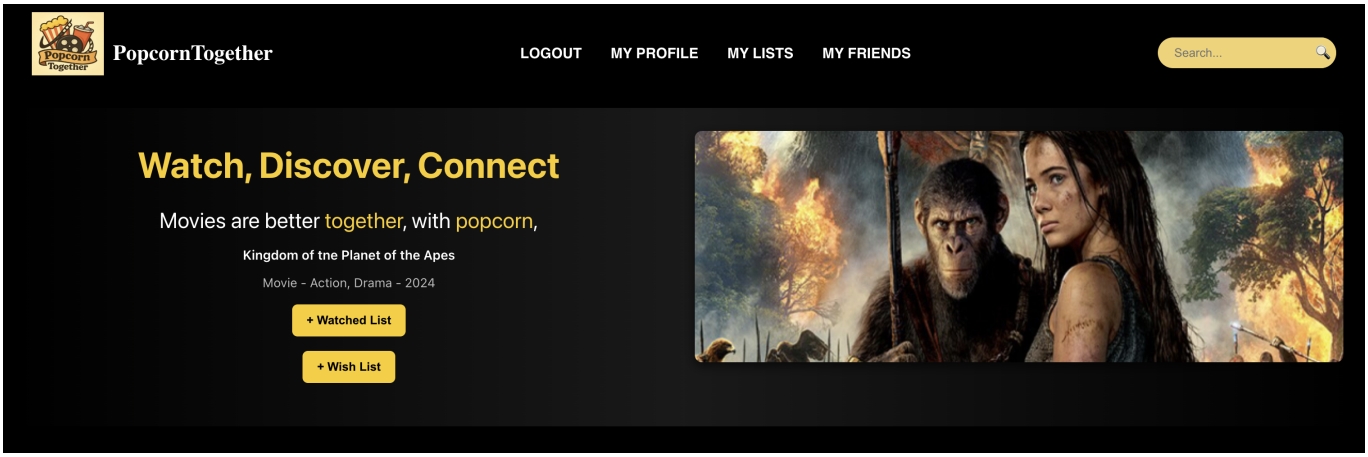
The following section details the functionality and organisation of the respective features in PopcornTogether.

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.

Other features such as Friends list, Account profile, and Watched / Wish lists will 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.



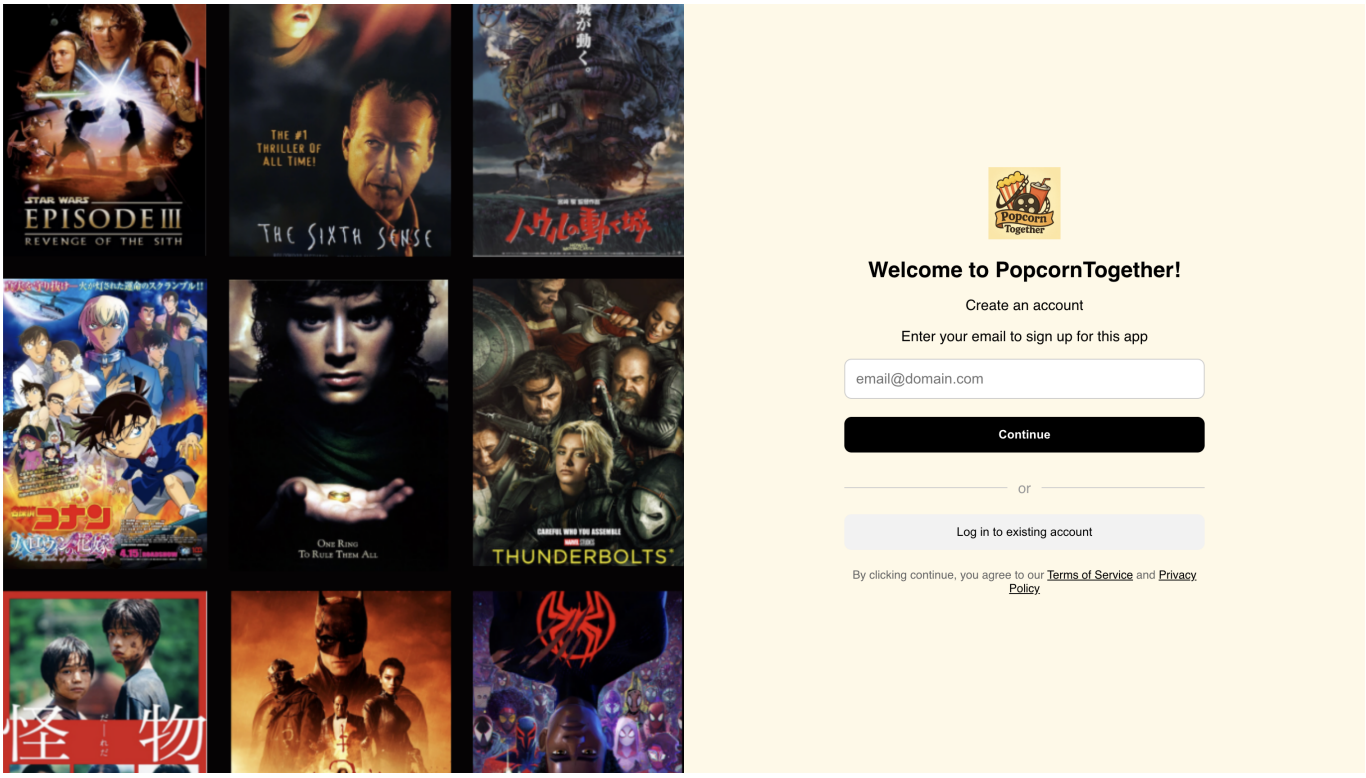
After logging in, the user will be redirected to the landing page once again, they will now be able to access the various features. They will also be able to logout of their account.

User Account/Profile

Core feature

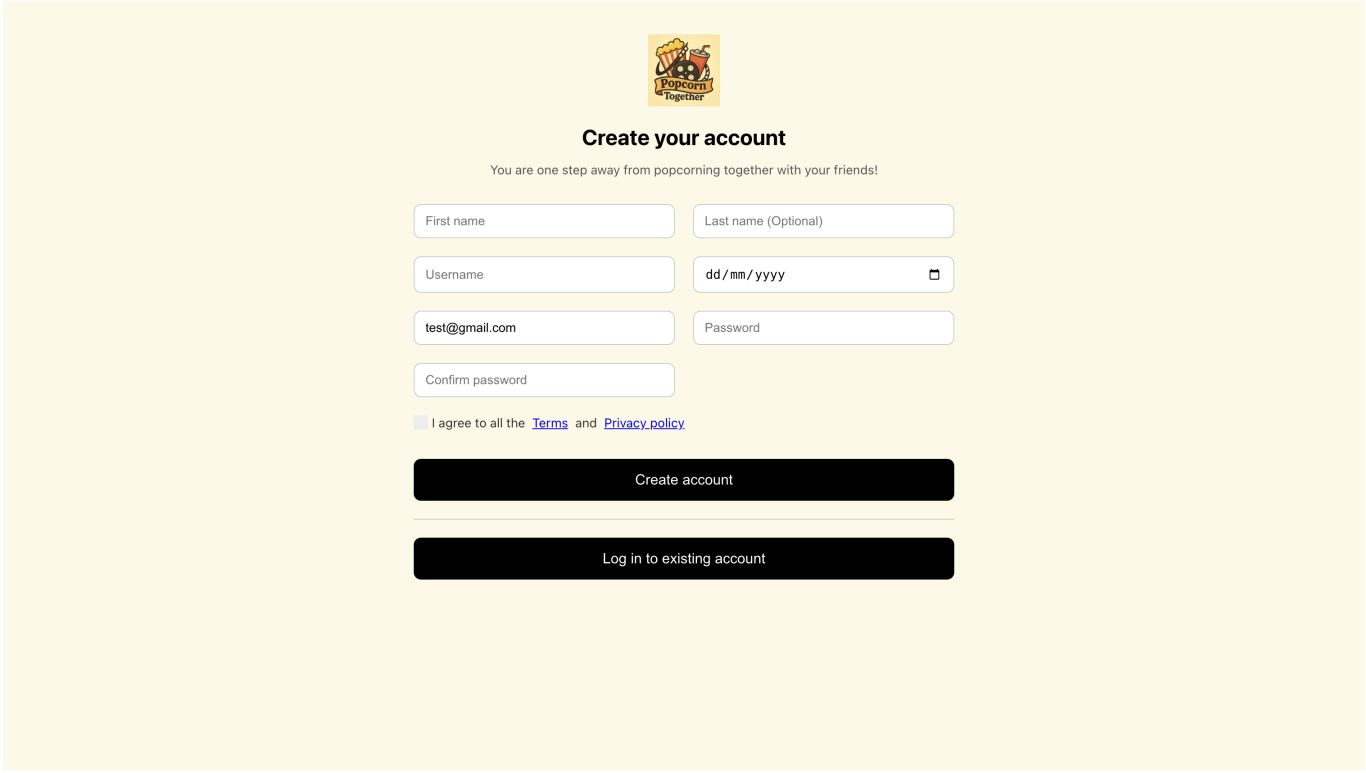
Users will first find themselves on the landing page where they can access the registration and login features. By creating an account, we are able to assign the user to their very own watched list and wishlists, as well as provide information on their watch statistics. Logging in also allows the user 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. The 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 and relationships.

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" }
    ]
}
```

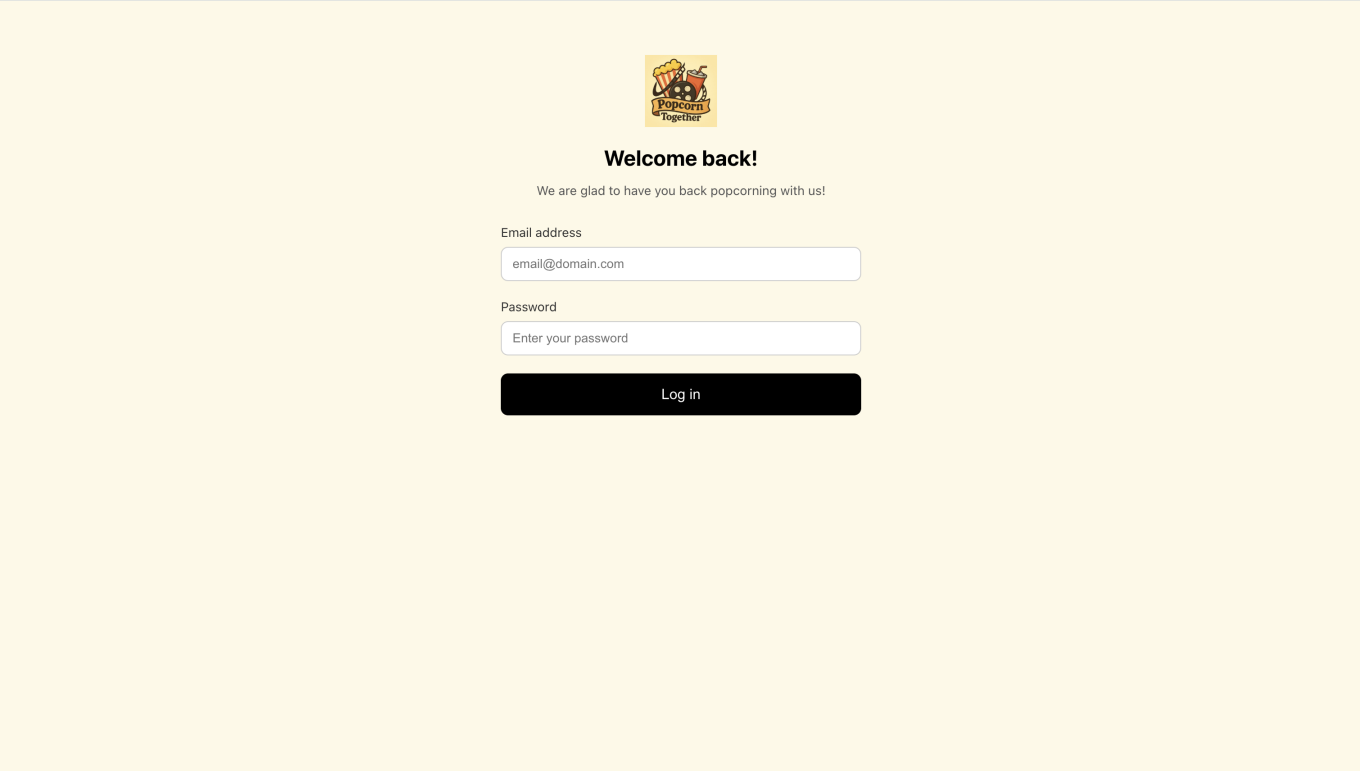
4. Friends list

```
{
    "userId": "ObjectId",
```

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

Furthermore, MongoDB provides certain security features such as hashing passwords using bycrypt 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.

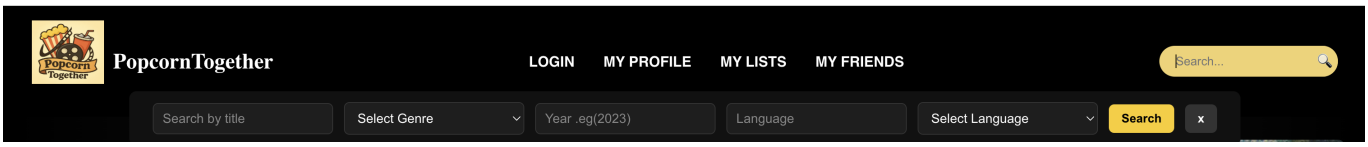
Movie Search

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.

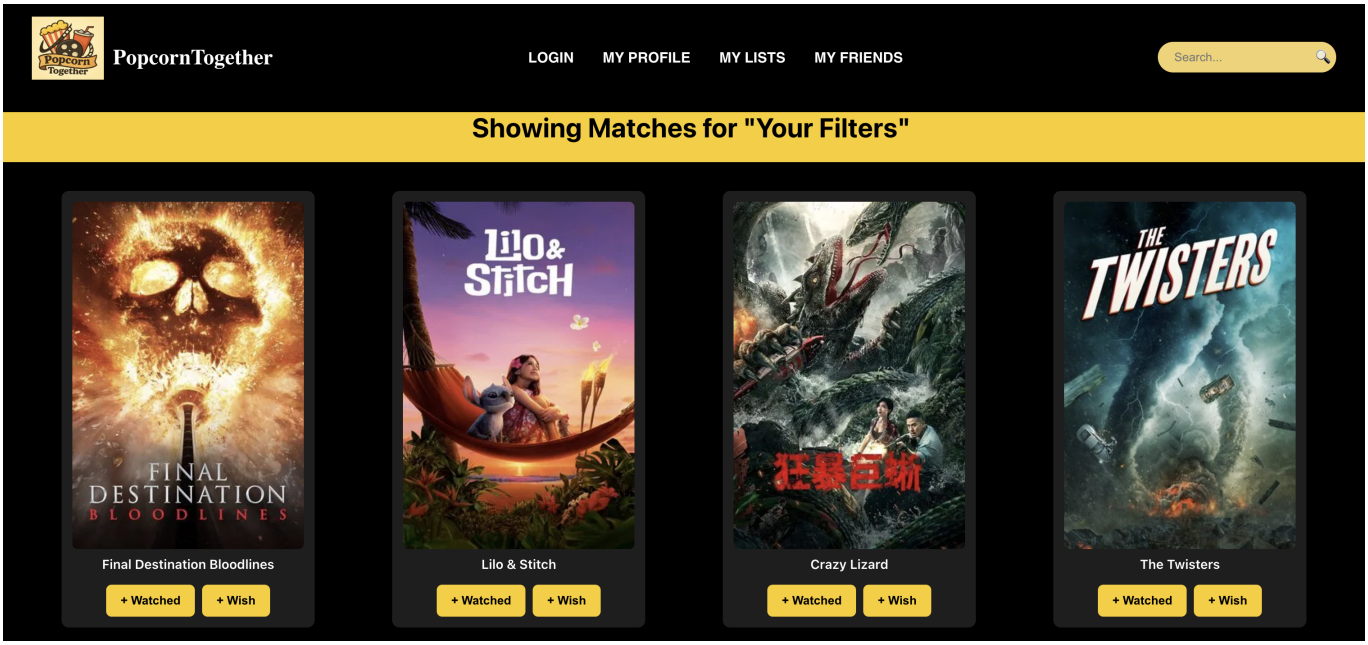
The Movie Database (TMDB)

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.

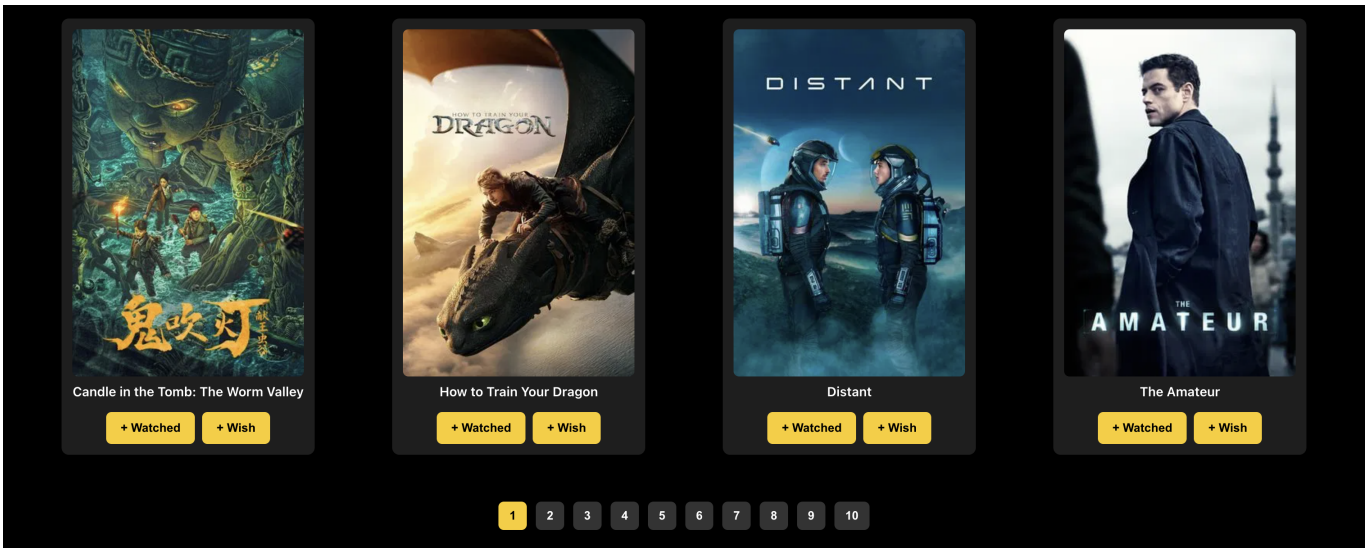
After successful query, the users will be able to view the following result page.

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.

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:

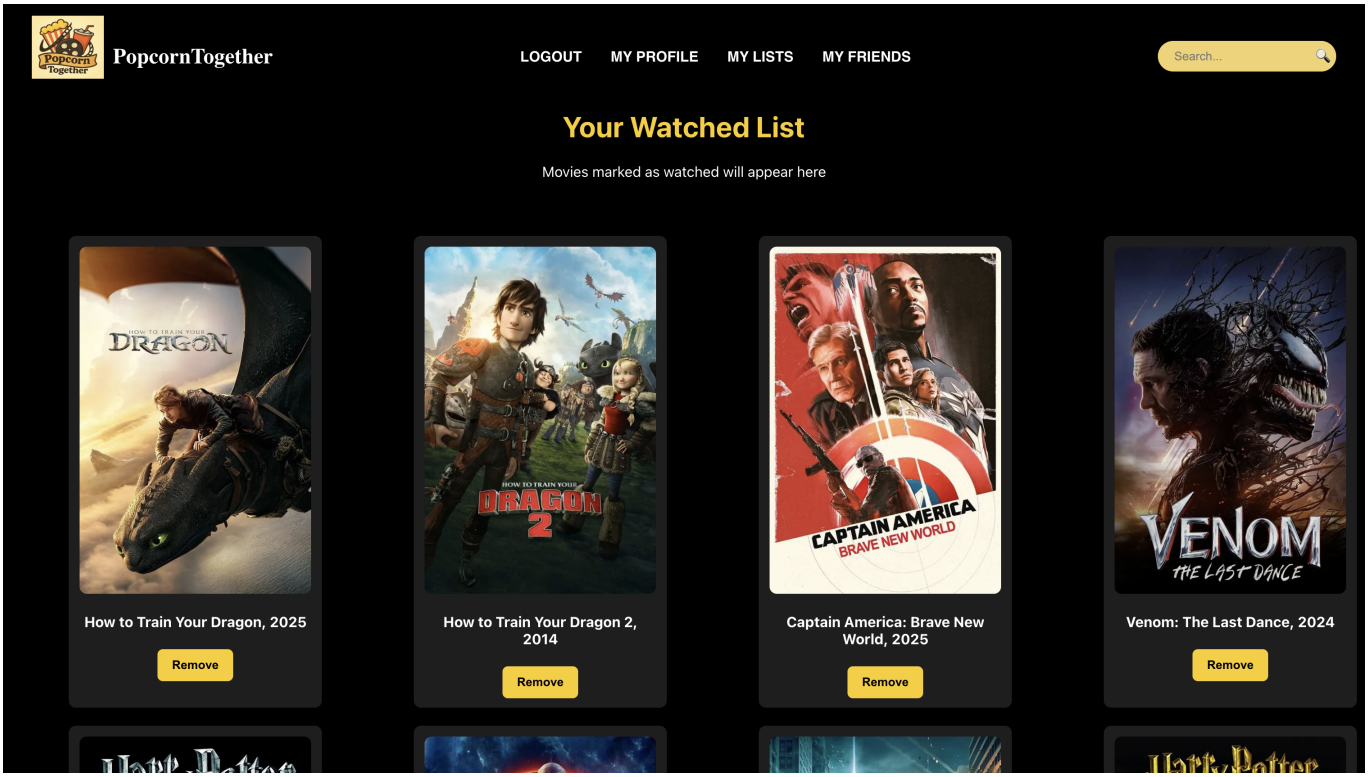
Dropdown menu

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

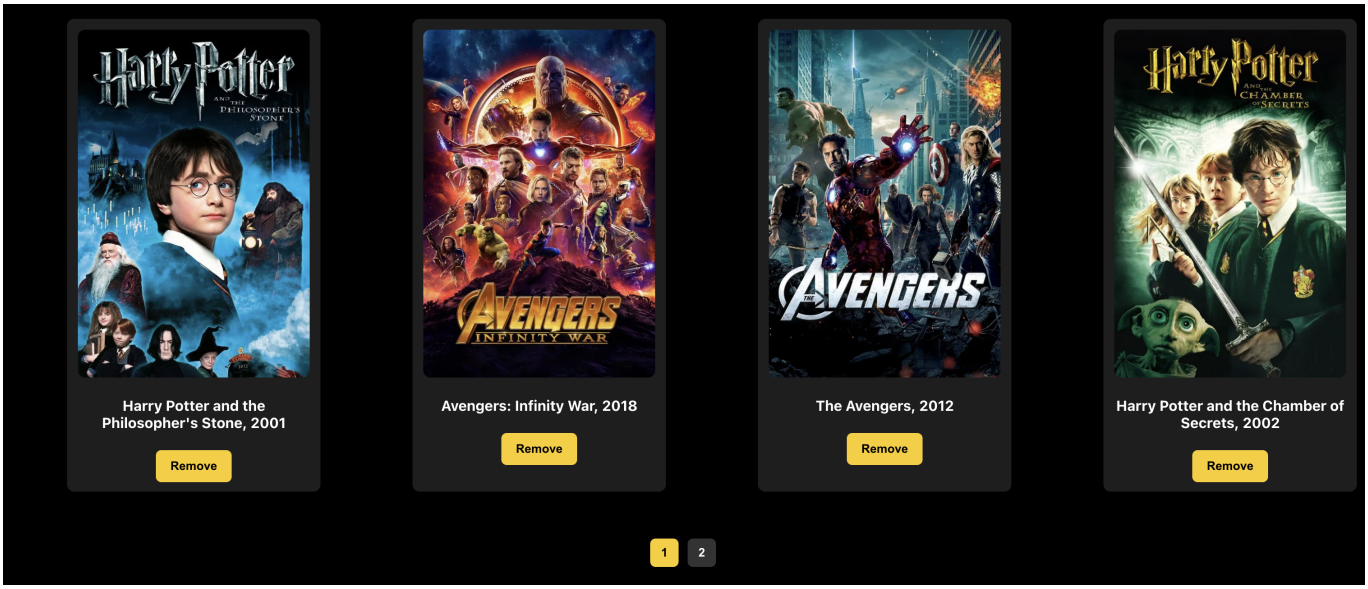
Afterwards they will be redirected to the Watched list.

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



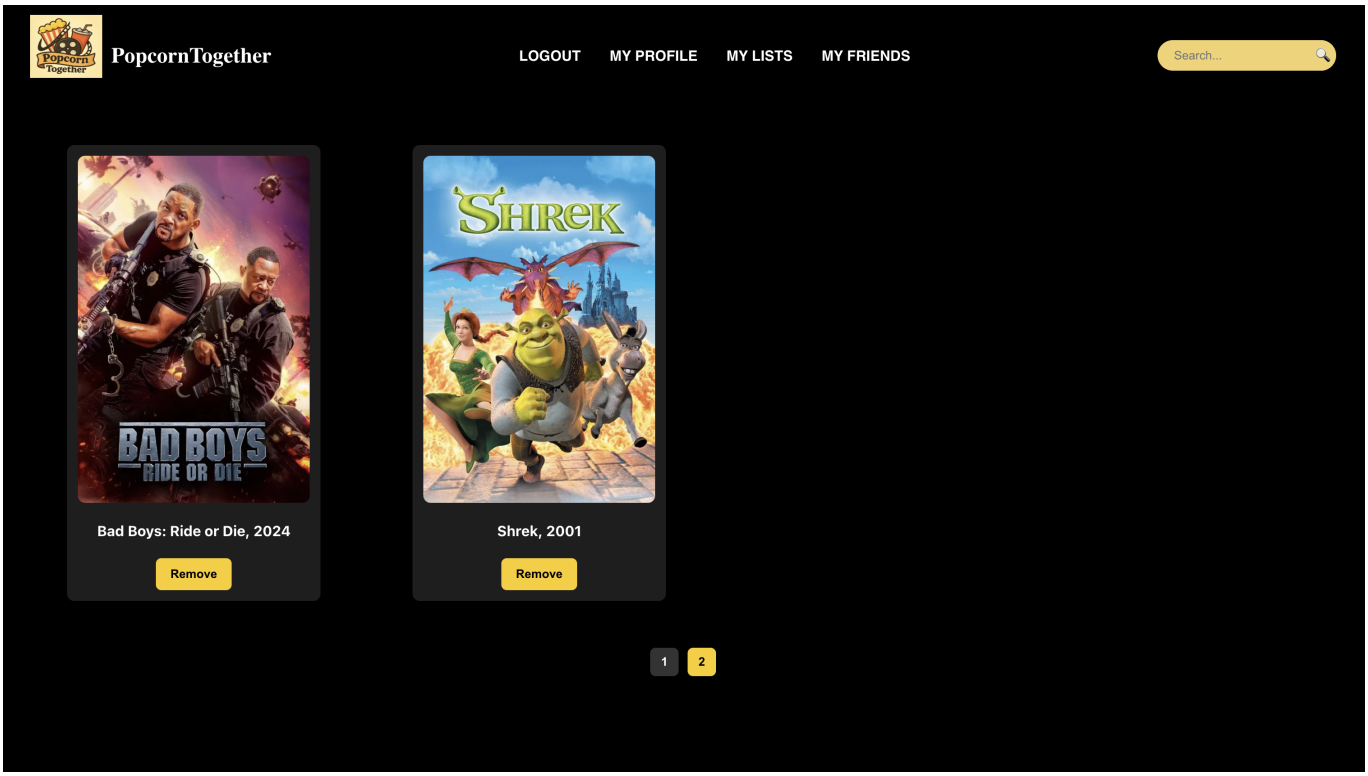
Watched list with pagination



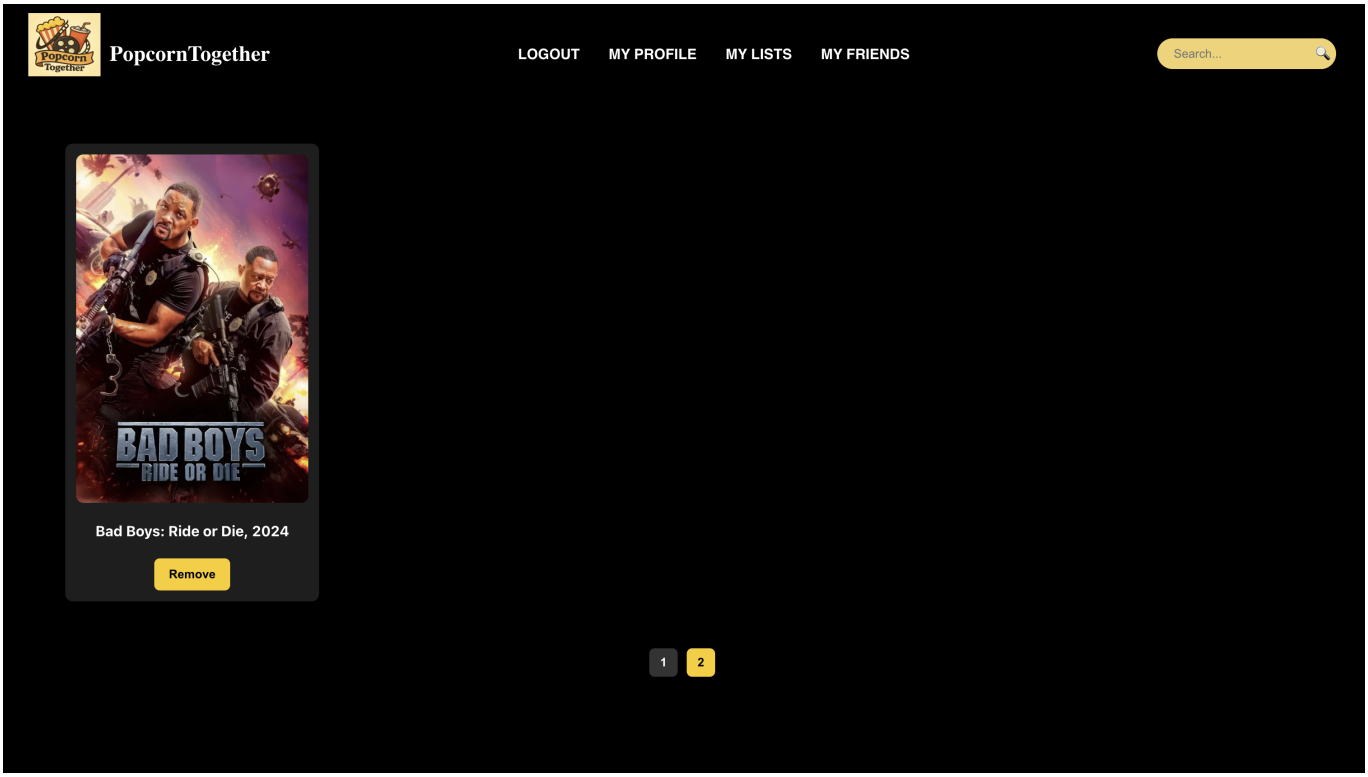
Removal from list

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.

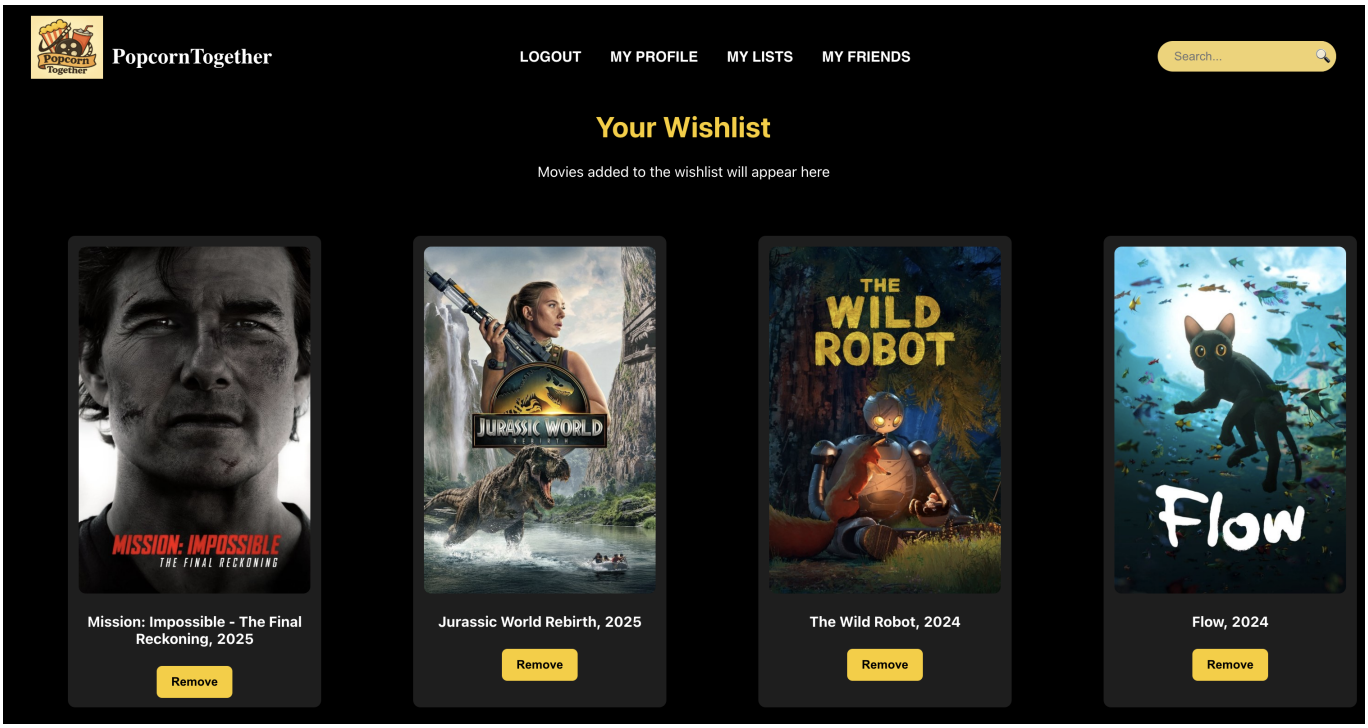
Wishlist

Core feature

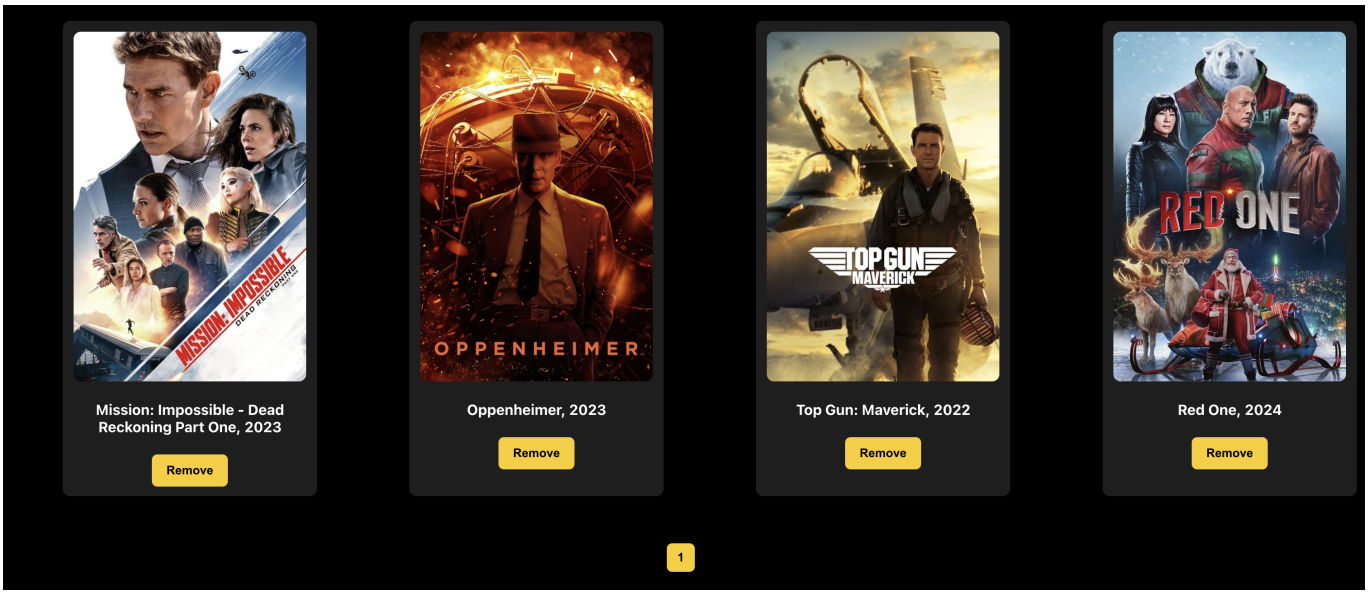
This page allows users to record down movies that they wish to watch in the future but do not have time for now. The purpose of this feature is:

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.

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 and their favorite movies.
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.

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.

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.

Filter stats by month or genre to understand your viewing trends better.

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.

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

By collating the community opinions of a movie based on a 5 star scale, our community reviews feature enables users to view movie ratings and short reviews from fellow users within the platform. This provides a more personalized and relatable perspective compared to generic critic reviews.

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.

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
```

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 Vercel to deploy the webapp
6. Extension	- Implementation of extension features
7. Refinement	- Add additional functionality for a robust Movie companion app
8. Final testing	- Ensure everything runs seamlessly - Optimisation

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

- Implemented session tracking
- Implemented search function
- Implemented results rendering function
- Implemented Watched list and Wish list function

Errors encountered

This serves as a documentation of errors we have encountered so far 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.

Another general issues 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.

Backend

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.

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

Deployment

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.