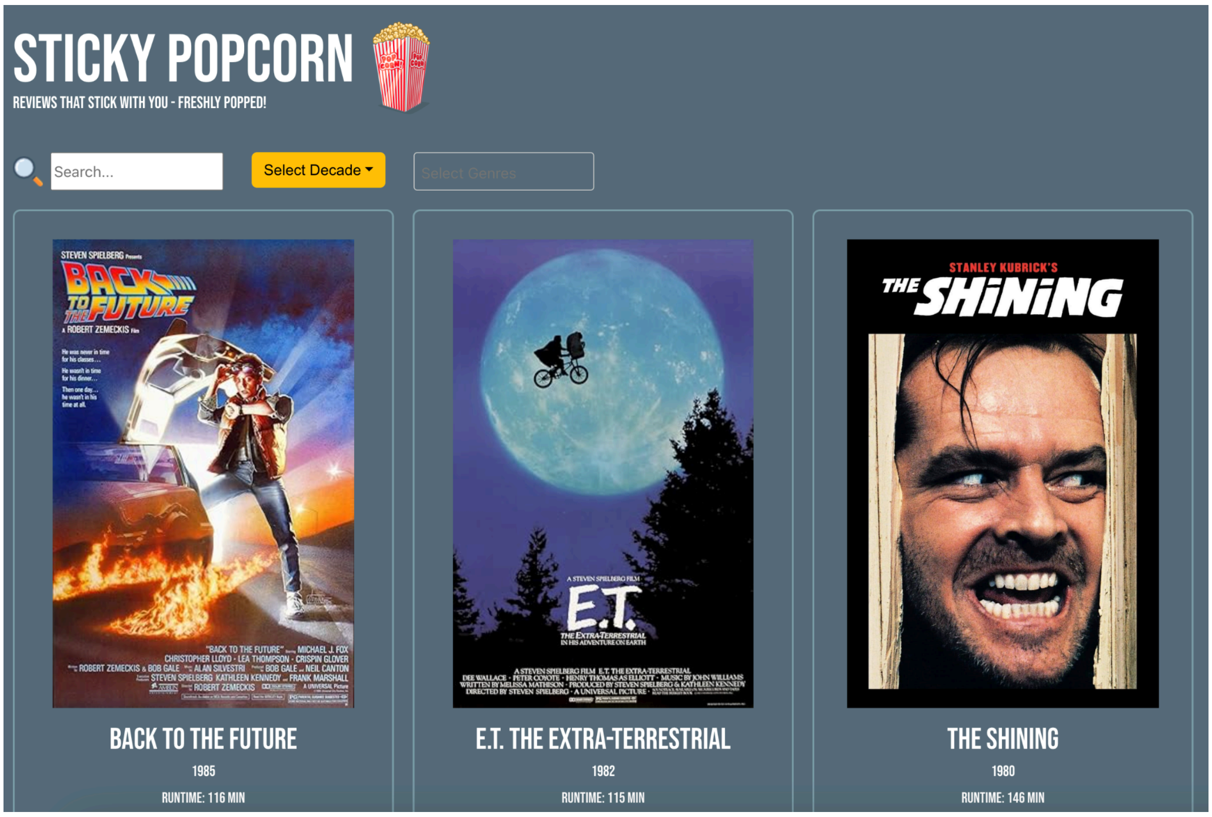
ReadMe

Description

For the third module of the course we were tasked with building a database app using Express and Node for the back-end and React for the front-end. This project was team-based with three of us working on the various components. Our team consisted of myself, Claire and Mai.

We decided to build a movies database, allowing users to filter movies and add to their favourites and watchlist, and to post reviews.



Deployment Link

https://sticky-popcorn.netlify.app/

Brief

For this project, you'll work across the back-end and the front-end to build a full-stack project with a team.

Your team will build an API on the back-end using Express and Node secured with JWTs. That API will interact with MongoDB to carry out full CRUD on resources.

On the front-end, your team will construct a React application that uses AJAX to communicate with the back-end app.

Technical requirements

- The back-end application is built with Express and Node.
- The front-end application is built with React.
- MongoDB is used as the database management system.
- The back-end and front-end applications implement JWT token-based authentication to sign up, sign in, and sign out users.
- Authorization is implemented across the front-end and back-end. Guest users (those not signed in) should not be able to create, update, or delete data in the application or access functionality allowing those actions.
- The project has at least two data entities in addition to the User model. At least one entity must have a relationship with the User model.
- The project has full CRUD functionality on both the back-end and front-end.
- The front-end application does not hold any secret keys. Public APIs that require secret keys must be accessed from the back-end application.
- The project is deployed online so that the rest of the world can use it.

Technologies

- Node.js
- Express.js
- CSS
- Postman
- React

Attribution

https://www.flaticon.com/free-icons/home-button

Planning

Having agreed on the subject being a movie database and considered the requirements, we went about planning the delivery of the project. Initially we worked together sketching out the layout and purpose of the app. At these early stages we also discussed the UI, UX and the User Story for our app and created a trello board. At the end of these discussions emerged a clearer picture of the app and its overall use. Having established the key requirements and the User Story, we then allocated each member with a task that would help complete the planning process. Mai was tasked with drawing wireframes, Claire was to develop a routing table and my task was to complete the Entity-Relationship Diagram (ERD) for the different models. Any questions that arose during this process were discussed and resolved.

https://trello.com/b/qLbGLI4H/sticky-popcorn-app

User Story

- As a visitor on the Landing Page I see all the movies on the database
- As a visitor on the Landing Page I see a nav bar
- As a visitor I can filter movies from the nav bar
- As a visitor I can search for a movie by its title
- As a visitor I can sort the movies in order of highest rating
- As a visitor I can click into and see the details including Reviews of any movie
- As a visitor I can signup and create my own account from the nav bar
- As a visitor I can login to my account
- As a user I can post a review and rate a movie
- As a user I can update/delete only my reviews
- As a user I can view movies I have liked
- As a user I can view movies in my watchlist
- Stretch goal - As a user I can update/delete only my responses
- Stretch goal - As a user I can respond to other reviews
- Stretch goal - As a user I can like other reviews
- Stretch goal - As a user I can post a new movie (only if it does not exist)
- Stretch goal - A user can reset their password from a forgotten password link
- Stretch goal - A user can search by the name of any actor/director

Build/Code Process

Mai created the Git Repo and granted us all access. Once the initial files were created and dependencies installed we each pulled the code from the Git Repo onto our local machines. We designated the three models to each member of the team. Claire was tasked with the user model, Mai with the reviews model and I was allocated the movies model. Each of us would have created models and controllers and then tested the routes in Postman. Creating the movies also involved gathering data and seeding the data. As part of the process each member of the team would create a branch for the code they were working on and each pull request was approved by somebody other than the member making the request. Any conflicts were resolved by jumping on a quick call and approving the edits. We also established a daily stand-up to discuss progress, any issues and the day-ahead.

Once the back-end was completed we moved on to the front-end and continued with the same designated sections we worked on in the back-end, which meant I worked on the movie components, Claire worked on the sign-in and register components and Mai continued with the reviews section.

The front end was built using React framework. I started by focusing on the nav bar and the landing page in general and then moved on to indexing all the movies and showing a single movie. I also implemented filtering to the movies collection, allowing users to search by movie title and filter by genre and/or decade.

Whilst we worked quite independently to begin with having designated roles, as we got deeper into the build and the three paths (sign-in, movies, reviews) started to converge we would re-group more and more frequently ensuring we committed our code and all pulled the latest version of the code before moving on to the next task.

Having completed the movie indexing, show and filtering I moved on to adding to watchlist and favourite feature. This functionality proved more complex than first anticipated and required me to establish a relationship between the user and movie collection.

In order to do so I had to re-visit my models and controllers in the back-end. I set up virtual fields in the movies schema and added many-to-many relational fields for favourites and watchlist in the user model. The idea being that every user will have an array of movie id's in their favourites and watchlist and a user can be added to a movie if they add it to their favourites and/or watchlist, so every movie will also have an array of user id's in those fields.

```
favourites: [{
   type: mongoose.Schema.Types.ObjectId,
   ref: "Movie"
 }],
 watchlist: [{
   type: mongoose.Schema.Types.ObjectId,
   ref: "Movie"
 }]
```

```
movieSchema.virtual('favouritedBy', {
   foreignField: 'favourites',
   localField: '_id',
   ref: 'User'
})

movieSchema.virtual('watchlistBy', {
   foreignField: 'watchlist',
   localField: '_id',
   ref: 'User'
})
```

Having updated the models and controllers the functionality was tested in Postman.

Once the back-end was working I moved on to implementing the feature in the front-end, generating new axios calls to fetch the user favourites and watchlist and creating new components to display the user's watchlist and favourite movies, e.g.

```
export const userFavouritesShow = async () => {
 try {
   const res = await axios.get(BASE_URL + `/favourites`, {
     headers: {
         Authorization: `Bearer ${getToken()}`
     }
   })
   console.log(res.data)
   return res.data
 } catch (error) {
   console.log(error)
   throw error
 }
}
```

```
useEffect(() => {
       userFavouritesShow()
           .then(data => {
               setMovies(data)
               setDisplayedMovies(data)
           })
           .catch(err => console.log(err))
           .finally(() => setIsLoading(false))
   }, [])
```

I then added some buttons to the single movie component to handle user's adding a movie to their favourite and/or watchlist.

```
{user ?
                   (<div className={styles.icons}>

                       <button className={styles.button} id="add-favourite"
onClick={() => {handleAddFavourite(movieId)}}>{ isFavourite ? <img
src="https://res.cloudinary.com/dvp3fdavw/image/upload/v1739356535/heart_1_yybbex.p
ng" /> : <img
src="https://res.cloudinary.com/dvp3fdavw/image/upload/v1739356535/heart_r6m3po.png
" />}</button>
                       <button className={styles.button} id="add-watchlist"
onClick={() => {handleAddWatchlist(movieId)}}>{ isWatchlist ? <img
src="https://res.cloudinary.com/dvp3fdavw/image/upload/v1739442031/television_djct6
l.png" /> : <img
src="https://res.cloudinary.com/dvp3fdavw/image/upload/v1739356535/tv_rkyzjg.png"
/>}</button>

                   </div>) : <></>
               }
```

```
const handleAddFavourite = async (movieId) => {
       try {
           const response = await addUserFavourite(movieId)
           setMovie(response)
           setIsFavourite(!isFavourite)
           if (!response || !response._id) {
               throw new Error("Invalid response from the server");
           }
           if (!movieId) {
               throw new Error("Incorrect movie id")
           }

       } catch (error) {
           if (error.response && error.response.data && error.response.data.errors)
{
               setError(error.response.data.errors)
           } else {
               setError({ general: "Something went wrong. Please try again." })
           }
       }
   }
```

With this implemented user's are now able to click on the favourite or watchlist icon in the nav bar and display a list of movies they have added to each respective category. They can also click into any particular movie and add to their favourite or watchlist.

As we each completed our sections we moved on to styling our respective components. I had implemented some basic placeholder styling to all my components as I went along but gave Mai and Claire the freedom to style my components as they saw fit, allowing me to focus on implementing the above feature.

Challenges

The key challenge in this project was working as a team on a single code base, maintaining clear lines of communication and remembering to git pull the latest code - one that became pertinent to myself, a valuable lesson learnt.

Implementing the add to favourites and watchlist feature proved to be much more complicated than initially thought and took up more time than anticipated.

Wins

- The team dynamic in general was very good, we all got along well and the collaborative aspect was very smooth.
- The communication between all of us was very good.
- The routing table and the ERD served as valuable reference points.
- The trello board is a very useful tool for organising.
- Movie indexing and filtering features.
- Really happy with the add to watchlist and favourites feature.
- Much better understanding of GitHub and version control.

Key Learnings/Takeaways

This was my first React App and it significantly deepened my understanding of key concepts such as components, the virtual DOM, state management and hooks like useEffect, useState, useContext. It also reinforced my knowledge of back-end technologies like Express.js and tools like Postman.

As a team project, this was as much an exercise in project management and GitHub version control as it was in coding. It provided valuable experience using GitHub and as mentioned above, a valuable lesson in how important it is to always pull the latest code from the main branch. The project management aspect consumed a not insignificant amount of time per day and highlighted the need to consider and incorporate project management in the planning of daily tasks and overall progress tracking. In a practical sense any large project with a team of developers would require significant project management time built into the estimate.

Bugs

I’m not aware of any bugs at this moment in time. While we conducted some testing, time constraints limited our ability to comprehensively test the app before submission.

Future Improvements

In the future I would like to revisit this project, improving some of the styling and perhaps implementing some of the stretch goals, such as allowing users to rate a movie and then sort movies by their rating.