FlixForum

A forum platform where users can discuss Netflix TV show episodes as they are watching it. Users must login/register to be able to post under forums and like posts, but they are still able to view posts made by others if they are not registered users.

API description

Heroes Netflix API

Description:

The API that is used for this program to get Netflix show data is from the website RapidAPI.com which provides queries that allow for all of a show's information to be grabbed such as the title, number of seasons, and number of episodes. With the grabbed data, we can display the information of shows for our main page and search results, as well as provide the option of selecting a show and its seasons and episodes to be used for creating forum posts.

Explore

- Search
    - Use this query to search for TV Show based on titles
    - Under titles → summary - grab the id of the show
    - Under titles → jawSummary - grab the title
    - Under titles → jawSummary - grab the backgroundImage

Title

- title-seasons
    - Use this query to search for titles of the seasons of a TV Show
    - Under seasons → grab the seasonId
    - Under seasons → grab the name of the season using shortName

Season

- seasons-episodes
    - Use this query to search for episodes of a seasons of a TV show
    - Under episodes → summary → grab the id of the episode
    - Under episodes → grab the title of the episode

Database description

Users Schema

```
id              INT NOT NULL AUTO_INCREMENT,
username        VARCHAR(75) NOT NULL,
password        VARCHAR(75) NOT NULL,
email           VARCHAR(75) NOT NULL,
PRIMARY KEY (id)
```

Posts Schema

```
posts_id        INT NOT NULL AUTO_INCREMENT,
user_id         INT NOT NULL AUTO_INCREMENT,
forum_id        INT NOT NULL AUTO_INCREMENT,
title           VARCHAR(75) NOT NULL,
content         VARCHAR(3500) NOT NULL,
likes           INT NOT NULL,
dislikes        INT NOT NULL,
PRIMARY KEY (post_id)
CONSTRAINT KEY poster
        FOREIGN KEY (id)
        REFERENCES users.id
        ON DELETE CASCADE
        ON UPDATE CASCADE
CONSTRAINT KEY postedIn
        FOREIGN KEY (forum_id)
        REFERENCES forums.forum_id
        ON DELETE CASCADE
        ON UPDATE CASCADE
```

Forums Schema

```
forum_id        INT NOT NULL AUTO_INCREMENT,
title           VARCHAR(75) NOT NULL,
season          INT NOT NULL,
episode         INT NOT NULL,
PRIMARY KEY (forum_id)
```

Post_likes Schema

```
forum_id        INT NOT NULL,
post_id         INT NOT NULL,
user_id         INT NOT NULL,
liked           TINYINT NOT NULL,
disliked        TINYINT NOT NULL,
PRIMARY KEY (forum_id, post_id)
CONSTRAINT KEY forum_id_key_pl
```

```
        FOREIGN KEY (forum_id)
        REFERENCES forum.forum_id
        ON DELETE CASCADE
        ON UPDATE CASCADE
CONSTRAINT KEY post_id_key_pl
        FOREIGN KEY (post_id)
        REFERENCES post.user_id
        ON DELETE CASCADE
        ON UPDATE CASCADE
CONSTRAINT KEY user_id_key_pl
        FOREIGN KEY (user_id)
        REFERENCES user.user_id
        ON DELETE CASCADE
        ON UPDATE CASCADE
```

Documentation for New Post Modal

Description :

The *NewPostModal* component is a button that opens up to a modal in the forum page. A user must have chosen a season and episode of the show they are on for the modal to be visible, and they must be logged in to make a post. Once the button is clicked, a modal that asks for a Title and Content pops up. Once they hit submit, the title and content of the post will be deployed to the database along with the forum_id to identify the forum the user made the post on, and the user_id to identify who made the post. Once the modal closes, the user can refresh the forum page and the new post with the title, content, and username will be displayed under the forum.

Logic :

The *NewPostModal* component will grab the title, season, and episode of the forum the user is in. It will also grab the user_Id, the post title, and the post content that the user inputted. Using Axios post, the *NewPostModal* component will deploy to the database's Posts table and forums table. The Forum table will hold the title, season, and episode of TV show forums that have a post under it. The Post table will hold the user_id of the user who made the post, the forum_id of the forum the user was on, the title of the post, and the content of the post.

If the post made by the user is the first post under a forum, the forums table will have a new entry inserted with the title, season, and episode of the forum that the user made a post under. If the post made is NOT the first post, then the forum_id is simply fetched from the database and is used to insert into just the posts table.

```
if ( first post under forum ) {
        insert into forum table ( title, season, episode)
        insert into posts table ( user_id, forum_id, title, content) // grab forum_id created above
} else {
        insert into posts table ( user_id, forum_id, title, content) // grab forum_id existing in db
}
```

Documentation for Register/Login

Description :
A Register is a page where users can create their account and their username. Once they enter their username, email address, and password, and click the “Sign up” button, it will create their account and navigate to the login page. A Login page is where users can sign in to their account to post their comments on the Forums page. A user needs to enter their email address and password (using the same email address and password when they register) to log in. A user can either click the “Login” button on the homepage to log in/register or it automatically navigates to the login page if users have not logged in yet and click the “New post” button on the forums page. Once they log in to their account, they are able to make a post on the forums page.

Logic :
The registration page will grab the username, email, and password from the text box users filled out using Axios post. It will deploy to the database users table. The users table contains the username, email, and password. The login page will grab the email and password from the users table using Axios post and check if the email address they entered in the login page exists on the users table. If it is, it will check whether their password matches their email address. If the email/password does not match, it will not navigate back to the page where users are at (stay on the login page). If it matches, it navigates back to the page where they are at.

Register:
  INSERT into users table (username, email, password)

Login:
```
 If (email users entered on the login page exist on the users table) {
     If (the password users entered on the login page match) {
       Navigate back to the page where users are at
     }
     else {
       Stay Login page
     }
}
Else {
    Stay Login page
}
```

Documentation for Search Bar and Search Result:

Description:
On the homepage of FlixForum, there will be a search bar in the upper right corner. The user must click on the bar and type in a TV show name and press enter. From there, they will be taken to a new page where their results will be shown for them to click on. Once they select whichever show they want to view or make new posts about, they will get taken to the new post modal component of our application. The user will be able to search, view and select whether or not they are logged in to an account.

Logic:
When the user goes to click on the search bar, types in a TV show and presses enter, they will be navigated to the search results page. The TV show that was entered is passed to the search result page using navigation arguments. In the search results page, this argument is used to create a new API call to fetch the corresponding TV show. From there, it's very similar to the overall logic of the homepage. However, now instead of displaying multiple random shows, only the TV shows that match the inputted TV show will be shown.

```
Search_input = useLocation()
If search_input.state {
        Query = JSON.stringify(search_input.message)
}

//make the new API call fetching a specific show
fetch('https://netflix-data.p.rapidapi.com/search/?query='+ query + '&limit_suggestions=1')

//display the show in the same way as the homepage using the showcard logic
```

*PLEASE NOTE: The API will display shows that don't have the exact same title of the one the user inputted, since the returned shows are based on the API's algorithm.

Documentation for Show Card

Description:
Using the Material UI component “Card” in order to display the picture and title of the show, the card can be clicked on like a button to allow the user to navigate to the forum page for that show and have the show’s data passed to the forum page to be used later. The information is retrieved through GET fetches to the Netflix API for shows and then the show information is put into a list to be mapped out to display the show’s information in a card with just the title and picture available. Arrows for the left and right side are used to navigate through the list by moving the cards positioning depending on which arrow button was clicked, the amount of times the cards can be moved right is 5. The home page will display three rows of types of shows from the Netflix API being popular, shows with multiple seasons, and shows with the letter A in the title. If data can’t be pulled or needs time to grab data, the rows will appear empty.

Logic:
Information is grabbed from the Netflix API upon arriving at the home page, then using the map function are converted to be used with Cards and display data of shows with helper functions that control what Show Cards are able to be seen and interacted with.

```
// variables that hold the lists of shows
// functions handeClick()  handle the arrow buttons for each of the three rows of shows
// with inputting “left” navigating to the left side of the shows and vice versa with inputting “right”
useEffect(GET fetch shows from Netflix API) // used for first two rows
useEffect(GET fetch shows from Netflix API with A in the title)

return(
        Display arrows and list.map() the shows to be a row of show cards
        Display arrows and seasonList.map() the shows to be a row of show cards
        Display arrows and showAList.map() the shows to be a row of show cards
)
```

Documentation for Forum

Description :
The forum page is one the user arrives on once they have chosen a TV show displayed on the home page or by using the search bar. Once on the forum page, the users can choose a season and episode of the TV show. Once they click on the refresh button, if the forum page has posts, they will be able to view the posts made previously. They will also be able to make a new post under the forum if they are logged in.

Logic :
When a user arrives at the forum page of the show they selected, they will be presented with the show's title and featured picture and the option to select the show's season and associated episodes. A fetch request has already been made with the show's id to find all the seasons, when a season has been picked another fetch request is made to grab all the episodes associated with that season. Once a season and episodes have been picked, pressing the GO button will call an Axios get request from the database to then display all the forum posts associated with the season and episode, if there are no posts made yet, it'll just remain empty. All the posts displayed will include the username of the poster, title, content, and like and dislike buttons. Local storage is used to help when refreshing the page or when a new post is made, so that the selected season and episode will remain until the user changes it.

// grabs from local storage that was set when the show's card was picked
// 2 useEffect() functions used for get requests to the Netflix API
fetch(seasons of show using show id)
    setSeasonList(convert_season.seasons)
fetch(episodes of season)
    setEpisodeList(convert_epi)
// fetch function used to make the request to database to get forum posts
fetch(forums from database using params of the show's id, season id, episode id, and user id)
    setForumList(response.data)
// dropdown for season of show (updates useEffect associated)
// dropdown for episodes of season (updates useEffect associated)
// button GO that calls the fetch function to grab the forum posts
forumList.map() to display all the retrieved posts associated with the current season and episode and displays the posts information

Documentation for Like/Dislike Button

Description:
Once a user selects a specific season and episode for a TV show, if posts have been made, they will be displayed under the picture. Each post will have a thumbs up and thumbs down button that displays the total amount of likes and dislikes for that specific post. A user can only press the buttons if they are logged into an account. If they press either button when they are not logged in, it will redirect them to the login page. Once they login, they will be able to press either the like/dislike button (only once).

Logic:
There are multiple components for the like/dislike buttons to function. As stated above, in the Posts table, that is where we store the total amount of likes and dislikes for each post to display next to the button. In addition, we also use a useState() to store the total likes/dislikes locally.

```
//when the page first opens up, retrieve the total likes & dislikes from the Posts table
setLikeCount(response.data[0].Likes)
setDislikeCount(response.data[0].dislikes)

//when the user likes/dislikes the post, increment the column in the database and increment the
//useState value
Axios.post("http://localhost:3001/addLike", {
        post_id: post_id,
}.then((response) => {});
setLikeCount(likeCount + 1);
```

In addition, the useState [activeBtn, setActiveBtn] = useState(0), was used to keep track of what the state the user is in. activeBtn is a field associated with each like and dislike button on the page. activeBtn has three different states: none, like, and dislike; each of these states reflect what action has been taken with that button. "none" denotes no vote, like means the "like" button was pressed, and likewise with "dislike". This means that there are 6 different scenarios to be accounted for:

- if the like button pressed and post was previously neither liked, nor disliked:
    - increment the post's like count
    - add an entry into the liked_post database and set the "liked" field to 1.
- if like button pressed and post was previously liked:
    - decrement the post's like count
    - update the liked_post entry's "liked" field to 0.
- if like button pressed and post was previously disliked:
    - decrement the post's dislike count, increment the like count
    - update the liked_post entry's "disliked" field to 0 and "liked" to 1.
- if dislike button pressed and post was previously neither liked, nor dislike:
    - increment the post's dislike count
    - add an entry into the liked_post database and set the "disliked" field to 1.

- if dislike button pressed and post was previously disliked:
    - decrement the post's dislike count
    - update the liked_post entry's "disliked" field to 0.
- if dislike button pressed and post was previously liked:
    - decrement the post's like count, increment the dislike count
    - update the liked_post entry's "liked" field to 0 and "disliked" to 1.

Example of using the activeBtn state to determine what to update the database and counter to:

```
if (activeBtn === 'like'){ // take away a like if the user presses the like button again
        Axios.post("http://localhost:3001/subLike", {
        post_id: post_id,
}).then((response) => {});
        setLikeCount(likeCount - 1);
        Axios.post("http://localhost:3001/updatePostsLikes", {
                post_i  d: post_id,
                user_id: user_id,
                forum_id: forum_id,
                like_value: 0,
                dislike_value: 0,
        }).then((response) => {})
         setActiveBtn("none");
  }

if (activeBtn === "dislike") {
        // update the DB
        Axios.post("http://localhost:3001/addLike", {
                post_id: post_id,
        }).then((response) => {});
        setLikeCount(likeCount + 1);
        Axios.post("http://localhost:3001/subDislike", {
                post_id: post_id,
         }).then((response) => {});
        setDislikeCount(dislikeCount - 1);
        Axios.post("http://localhost:3001/updatePostsLikes", {
                post_id: post_id,
                user_id: user_id,
                forum_id: forum_id,
                like_value: 1,
                dislike_value: 0,
         }).then((response) => {})
        setActiveBtn("like");
  }
```