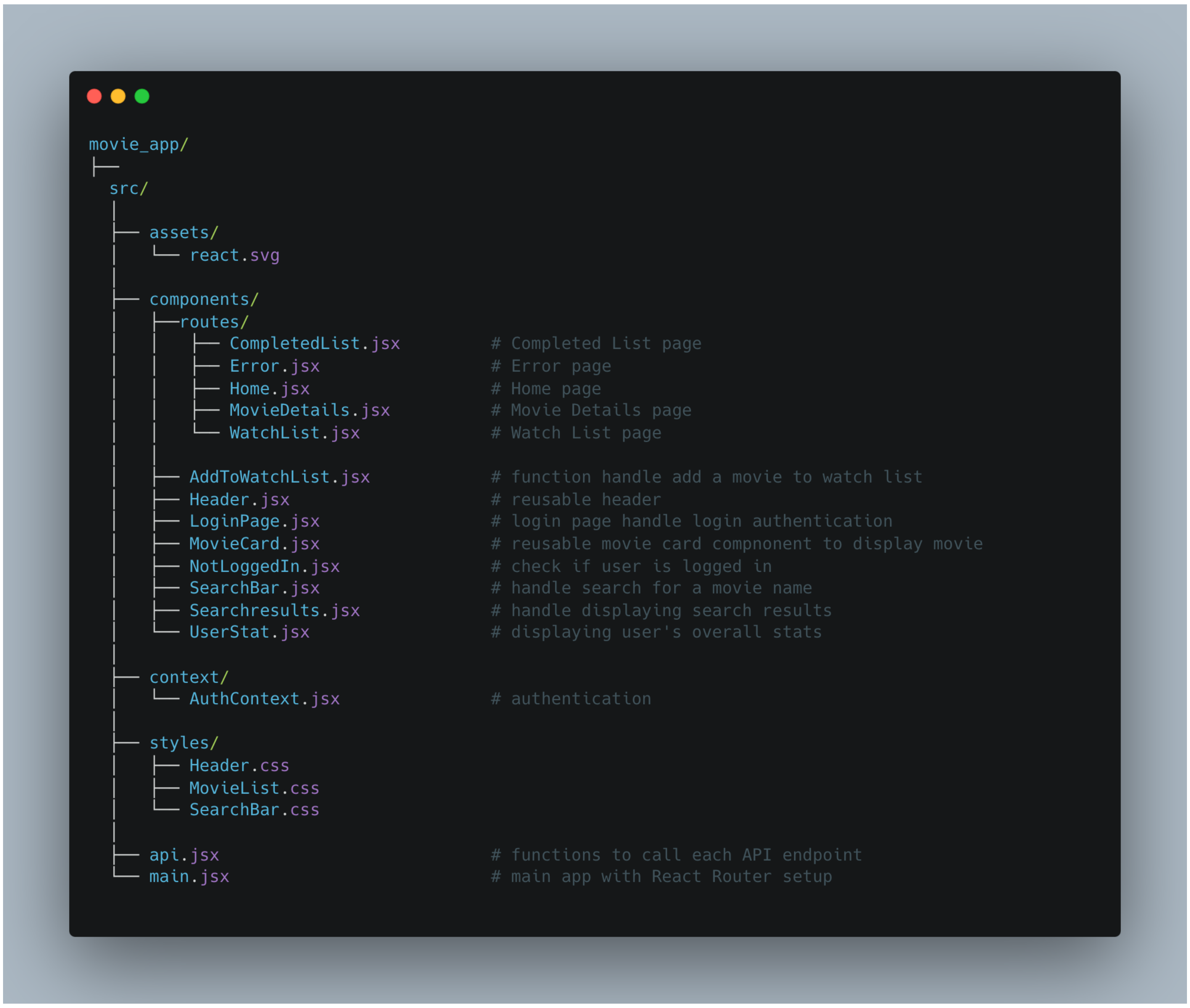
Assignment 3 Code Snippets



```
movie_app/
├─
  src/
  │
  ├── assets/
  │   └── react.svg
  │
  ├── components/
  │   ├──routes/
  │   │   ├── CompletedList.jsx          # Completed List page
  │   │   ├── Error.jsx                  # Error page
  │   │   ├── Home.jsx                   # Home page
  │   │   ├── MovieDetails.jsx           # Movie Details page
  │   │   └── WatchList.jsx              # Watch List page
  │   │
  │   ├── AddToWatchList.jsx             # function handle add a movie to watch list
  │   ├── Header.jsx                     # reusable header
  │   ├── LoginPage.jsx                  # login page handle login authentication
  │   ├── MovieCard.jsx                  # reusable movie card compnonent to display movie
  │   ├── NotLoggedIn.jsx                # check if user is logged in
  │   ├── SearchBar.jsx                  # handle search for a movie name
  │   ├── Searchresults.jsx              # handle displaying search results
  │   └── UserStat.jsx                   # displaying user's overall stats
  │
  ├── context/
  │   └── AuthContext.jsx                # authentication
  │
  ├── styles/
  │   ├── Header.css
  │   ├── MovieList.css
  │   └── SearchBar.css
  │
  ├── api.jsx                            # functions to call each API endpoint
  └── main.jsx                           # main app with React Router setup
```

- components/routes/CompletedList.jsx

```
import React, { useState, useEffect } from "react";
import { useAuth } from "../../context/AuthContext";
import Header from "../Header";
import { getCompletedMovies, updateTimesWatched, updateMovieRating } from
"../../api";

const CompletedList = () => {
  const { apiKey } = useAuth();
  const [completedMovies, setCompletedMovies] = useState([]);
  const [error, setError] = useState(null);
  const [loading, setLoading] = useState(true);
  const [message, setStatusMessage] = useState("");
  const [movieRatings, setMovieRatings] = useState({});
  const [showForm, setShowForm] = useState(false);
  const toggleFormVisibility = () => {
    setShowForm(!showForm);
  };

  useEffect(() => {
    if (!apiKey) {
      setError("API key is missing.");
      setLoading(false);
      return;
    }
    const fetchCompletedMovies = async () => {
      try {
        const data = await getCompletedMovies(apiKey);
        console.log("Fetched watchlist data:", data);
        setCompletedMovies(data);
      } catch (error) {
        setError("Error fetching completed movies:", error);
      } finally {
        setLoading(false);
      }
    };
    if(apiKey){
      fetchCompletedMovies();
    }
  }, [apiKey]);

  const handleUpdateTimesWatched = async (movieId) => {
    if (!apiKey) {
      setError("API key is missing.");
      return;
    }
    try {
      await updateTimesWatched(movieId, apiKey);
      const updatedMovies = await getCompletedMovies(apiKey); // refetch
completed list to display updated data
      setCompletedMovies(updatedMovies); // update the list
      setStatusMessage("Successfully updated the times watched!");
    } catch (error) {
      setError("Error updating times watched", error);
      setStatusMessage("Failed to update times watched.");
    }
  };

  const handleUpdateRating = async (movieId) => {
    if (!apiKey) {
      setError("API key is missing.");
      return;
    }

    const rating = movieRatings[movieId]; // get the rating for the
specific movie

    console.log("Updating rating for movie entry ID:", movieId, "to
rating:", rating); // debug
```

```
    // validate rating
    if (typeof rating !== 'number' || isNaN(rating)) {
      setStatusMessage("Invalid rating. Please enter a valid numeric
value.");
      return;
    }

    try {
      const updatedMovie = { new_rating: parseFloat(rating) };

      console.log("Sending updated rating:", updatedMovie.new_rating); //
debug

      // call API request to update rating
      await updateMovieRating(movieId, updatedMovie, apiKey);

      setStatusMessage("Rating updated successfully!");
    } catch (error) {
      setStatusMessage("Failed to update rating.");
      console.error("Error updating rating:", error);
    }
  };


  // handle the change in rating for a specific movie
  const handleRatingChange = (movieId, newRating) => {
    const parsedRating = parseFloat(newRating);

    setMovieRatings((prevRatings) => ({
      ...prevRatings,
      [movieId]: parsedRating,
    }));
  };

  // show loading, error, or empty message if no movies
  if (loading) return <p>Loading completed watchlist...</p>;
  if (error) return <p>{error}</p>;
  if (!completedMovies.length) return <p>Your completed list is empty!
</p>;

  return (
    <main>
      <Header />
      <div className="completed-list">

        <h1>My Completed List</h1>
        <div className="completedlist-container">
          {completedMovies.map((movie) => (
            <div key={movie.id} className="movie-card">
              <h2>{movie.title}</h2>
              <img src={movie.poster} alt={movie.title} />
              <p><strong>Movie Description: </strong>{movie.overview}</p>
              <p><strong>Movie Average Rating: </strong>{movie.rating}</p>
              <p><strong>Your Rating: </strong>{movieRatings[movie.id] ||
movie.userRating}</p>
              <p><strong>Times you have watched this movie: </strong>
{movie.times_watched}</p>
              <p><strong>The last time you watched it: </strong>
{movie.date_last_watched}</p>
              <p><strong>A note you left for this movie: </strong>
{movie.notes}</p>

              <button className="update-times-watched" onClick={() =>
handleUpdateTimesWatched(movie.id)}>
                Watched Again
              </button>

              {message && (
                <div className={message.includes("Failed") ? "error-
message" : "success-message"}>
                  {message}
                </div>
              )}

              <button className="update-rating" onClick=
{toggleFormVisibility}>
                {showForm ? "Hide Rating" : "Updating movie rating"}
              </button>

              {showForm && (
                <div>
                  <label>
                    (Re)-Rate this movie:
                    <input
                      type="number"
                      value={movieRatings[movie.id] || ""}
                      onChange={(e) => handleRatingChange(movie.id,
e.target.value)}
                      min="1"
                      max="10"
                    />
                  </label>

                  <button className="update-rating-submit" onClick={() =>
handleUpdateRating(movie.id)}>
                    Submit
                  </button>
                </div>
              )}

            </div>
          ))}
        </div>
      </div>
    </main>
  );
};
```

```
export default CompletedList;
```

- components/routes/Error.jsx

```
import React from "react";
import { Link } from "react-router-dom";

const Error = () => {
  return (
    <div>
      <h1>404 - Page Not Found</h1>
      <p>The page you are looking for does not exist.</p>
      <Link to="/">Go back to Home</Link>
    </div>
  );
};

export default Error;
```

- components/routes/Home.jsx

```
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { getMoviesPaginated } from "../../api";
import "../../styles/MovieList.css";
import Header from "../Header";
import AddToWatchlist from "../AddToWatchList";

const MoviesList = () => {
  const [movies, setMovies] = useState([]);
  const [filteredMovies, setFilteredMovies] = useState([]);
  const [loading, setLoading] = useState(true);
  const [currentPage, setCurrentPage] = useState(1);
  const [totalPages, setTotalPages] = useState(1);
  const [yearFilter, setYearFilter] = useState("all");
  const resultsPerPage = 30;
  const navigate = useNavigate();

  const handleClick = (id) => {
    navigate(`/movie/${id}`);
  };

  useEffect(() => {
    const fetchMovies = async () => {
      setLoading(true);
      try {
        const data = await getMoviesPaginated(resultsPerPage,
currentPage);
```

```
        if (!data || !data.length) { // no movies found
          console.log("No movies found.");
          setMovies([]); // reset array of movies
          setFilteredMovies([]); // reset array of filtered movie
          return;
        }

        console.log("Movies fetched:", data); // debug
        setMovies(data || []); // set movies to the fetched data
        setFilteredMovies(data);

        const totalMovies = data.length; // find total movies fetched
        const calculatedTotalPages = Math.ceil(totalMovies /
resultsPerPage); // total pages based on total movies
        setTotalPages(calculatedTotalPages); // set total pages
      } catch (error) {
        console.error("Error fetching movies:", error);
        setMovies([]); // reset array of movies
        setFilteredMovies([]);
      } finally {
        setLoading(false);
      }
    };

    fetchMovies();
  }, [currentPage]); // re-redender if current page changes

  // Handle Release Year Filter
  const handleYearFilterChange = (selectedYearRange) => {
    setYearFilter(selectedYearRange);

    if (selectedYearRange === "all") {
      setFilteredMovies(movies);
      setTotalPages(Math.ceil(movies.length / resultsPerPage)); // update
total pages for all movies
      setCurrentPage(1); // reset to the first page
      return;
    }

    const yearRanges = {
      "2000-2005": [2000, 2005],
      "2006-2010": [2006, 2010],
      "2011-2015": [2011, 2015],
      "2016-2020": [2016, 2020],
    };

    const [startYear, endYear] = yearRanges[selectedYearRange] || [];
    const filtered = movies.filter((movie) => {
      const movieYear = new Date(movie.release_date).getFullYear();
      return movieYear >= startYear && movieYear <= endYear;
    });

    setFilteredMovies(filtered);
    setTotalPages(Math.ceil(filtered.length / resultsPerPage)); // update
total pages for filtered movies
    setCurrentPage(1); // reset to the first page
  };

  // if (loading) return <p>Loading...</p>;

  const startIndex = (currentPage - 1) * resultsPerPage;
  const endIndex = startIndex + resultsPerPage;
  const paginatedMovies = filteredMovies.slice(startIndex, endIndex);

  return (
    <main>
      <Header />
      <div className="movie-list">
        <h1>Movies List</h1>

        {/* filter movie by released year */}
        <div className="filter-container">
          <label htmlFor="year-filter">Filter by Release Year:</label>
          <select
            id="year-filter"
            value={yearFilter}
            onChange={(e) => handleYearFilterChange(e.target.value)}
          >
            <option value="all">All Years</option>
            <option value="2000-2005">2000 - 2005</option>
            <option value="2006-2010">2006 - 2010</option>
            <option value="2011-2015">2011 - 2015</option>
            <option value="2016-2020">2016 - 2020</option>
          </select>
        </div>

        <div className="movie-section">
          {paginatedMovies.length ? (
            paginatedMovies.map((movie) => (
              <div key={movie.id} className="movie-item">
                <h2
                  className="movie-title"
                  title={movie.title} // tooltip showing full title
                >
                  <a onClick={() => handleClick(movie.id)}>
{movie.title.split(" ").slice(0, 4).join(" ")}</a>
                  {movie.title.split(" ").length > 4 ? "..." : ""}
                </h2>
                <img src={movie.poster} alt={movie.title}
className="movie-poster" />
                <button onClick={() => handleClick(movie.id)}>View more
details</button>
                <AddToWatchlist movieId={movie.id} />
              </div>
            ))
          ) : (
            <p>No movies available</p>
          )}
        </div>
      </div>

      <div className="pagination">
        <button
          onClick={() => setCurrentPage((prevPage) => Math.max(prevPage -
1, 1))}
          disabled={currentPage === 1}
        >
          Previous
        </button>

        {[...Array(totalPages)].map((_, index) => (
          <button
            key={index}
            onClick={() => setCurrentPage(index + 1)}
            className={currentPage === index + 1 ? "active" : ""}
          >
            {index + 1}
          </button>
        ))}

        <button
          onClick={() => setCurrentPage((prevPage) => prevPage + 1)}
          disabled={currentPage === totalPages}
        >
          Next
        </button>
      </div>

      <p>
        Page {currentPage} of {Math.ceil(filteredMovies.length /
resultsPerPage)}
      </p>
    </main>
  );
};

export default Home;
```

- components/routes/MovieDetails

```
import { useState, useEffect } from "react";
import { useParams } from "react-router-dom";
import Header from "../Header"
import { getMovieById } from "../../api";
import AddToWatchlist from "../AddToWatchList";

const MovieDetails = () => {
  const { id } = useParams();
  const [movie, setMovie] = useState(null);
```

```
    const [error, setError] = useState("");

    useEffect(() => {
      const fetchMovieDetails = async () => {
        try {
          const data = await getMovieById(id);
          console.log("Movie details data:", data);
          setMovie(data);
        } catch (error) {
          console.error("Error fetching movie details:", error);
        }
      };

      fetchMovieDetails();
    }, [id]);

    if (error) return <p>{error}</p>;
    if (!movie) return <p>Loading...</p>;

    return (
      <main>
        <Header />
        <div className="movie-detail">
          {movie ? (
            <div className="movie-card">
              <h1>{movie.title}</h1>
              <img src={movie.poster} alt={movie.title}/>
              <div className="details">
                <p><strong>Movie Description:</strong> {movie.overview}</p>
                <p><strong>Homepage:</strong> {movie.homepage}</p>
                <p><strong>Runtime:</strong> {movie.runtime} minutes</p>
                <p><strong>Tagline:</strong> {movie.tagline}</p>
                <p><strong>Rating:</strong> {movie.rating}</p>
                <p><strong>Release Date:</strong> {movie.release_date}</p>
              </div>

              <AddToWatchlist movieId={id}/>
            </div>
          ) : (
            <p>Loading movie details...</p>
          )}
        </div>
      </main>
    );
  };

  export default MovieDetails;
```

- components/routes/WatchList.jsx

```
import React, { useEffect, useState } from "react";
import { useAuth } from "../../context/AuthContext";
import { getWatchListEntries, updateWatchListPriority,
deleteWatchListEntries, markMovieAsWatched } from "../../api";
import Header from "../Header";
import MovieCard from "../MovieCard";

const WatchList = () => {
  const { apiKey } = useAuth();
  const [watchlist, setWatchlist] = useState([]);
  const [error, setError] = useState(null);
  const [loading, setLoading] = useState(true);
  const [message, setMessage] = useState("");

  const sortWatchlist = (list) => {
    return [...list].sort((a, b) => a.priority - b.priority);
  };

  useEffect(() => {
    if (!apiKey) {
      setError("API key is missing. Please log in first");
      setLoading(false);
      return;
    }

    // fetch user watchlsit
    const fetchWatchlist = async () => {
      try {
        const data = await getWatchListEntries(apiKey);
        console.log("Fetched watchlist data:", data);
        setWatchlist(data);
      } catch (err) {
        setError("Failed to fetch the watchlist.");
      } finally {
        setLoading(false);
      }
    };

    if (apiKey) {
      fetchWatchlist();
    }
  }, [apiKey]);

  // handle update priority
  const handleUpdatePriority = async (id, priority, movieID) => {
    console.log("Updating priority to", priority, "for movie ID:",
movieID);

    try {
      await updateWatchListPriority(apiKey, id, priority, movieID);
      setWatchlist((prevList) => sortWatchlist(
        prevList.map((movie) =>
          movie.id === id ? { ...movie, priority: priority } : movie
        )
      )); // Sort the updated list
      setMessage(`Priority for entry ID ${id} updated to ${priority}.`);
    } catch (error) {
      console.error("Error updating priority:", error);
      setMessage("Failed to update priority. Please try again.");
    }
  };

  // handle delete a watchlist entry
  const handleDeleteEntries = async (apiKey, entryId, movieId) => {
    console.log("Deleting movie with ID:", movieId);
    if (!movieId) {
      console.error("Missing movie ID for entry ID:", entryId);
      setMessage("Cannot delete entry. Missing movie ID.");
      return;
    }

    if (!apiKey) {
      setError("API key is missing.");
      return;
    }

    try {
      await deleteWatchListEntries(apiKey, entryId, movieId);
      setMessage(`Movie with entry ID ${entryId} deleted successfully.`);
      setWatchlist((prevList) => prevList.filter((movie) => movie.id !==
entryId)); // Filter out the deleted entry
    } catch (error) {
      console.error("Error deleting entry:", error);
      setMessage("Failed to delete entry. Please try again.");
    }
  };

  const handleMarkAsWatched = async (movieId, entryId, rating, notes) => {
    if (!apiKey) {
      setError("API key is missing.");
      return;
    }
    if (!entryId) {
      setError("Entry ID is missing.");
      return;
    }

    const parsedRating = parseFloat(rating);

    try {
      await markMovieAsWatched({
        apiKey,
        entryId,
        rating: parsedRating,
        notes,
      });

      setWatchlist((prevList) => prevList.filter((movie) => movie.id !==
entryId));
      setMessage("Movie marked as watched successfully.");
      setMessage("Movie marked as watched successfully.");
    } catch (error) {
      console.error("Error marking movie as watched:", error);
      setMessage("Failed to mark movie as watched.");
    }
  };

  if (loading) return <p>Loading watchlist...</p>;
  if (error) return <p>{error}</p>;
  if (!watchlist.length) return <p>Your watchlist is empty!</p>;

  return (
    <main>
      <Header />
      <div className="watch-list">
        <h1>My WatchList</h1>
        <div className="watchlist-container">
          {watchlist.map((movie) => (
            <MovieCard
            key={movie.id}
            movie={movie}
            onUpdatePriority={(priority) => handleUpdatePriority(movie.id,
priority, movie.movieID)}
            onDeleteEntry={(entryId, movieId) =>
handleDeleteEntries(apiKey, entryId, movieId)}
            onMarkAsWatched={handleMarkAsWatched}
          />

          ))}
        </div>
        {message && <p className="feedback-message">{message}</p>}
      </div>
    </main>
  );
};

export default WatchList;
```

- components/AddToWatchList.jsx

```


import { useState } from "react";
import { addToWatchlist } from "../api";
import { useAuth } from "../context/AuthContext";

const AddToWatchlist = ({ movieId }) => {
  const { apiKey } = useAuth(); // get the api key
  const [loading, setLoading] = useState(false); // state for loading
  const [watchlistMessage, setWatchlistMessage] = useState(""); // message
when add to watch list
  const [note, setNote] = useState("");
  const [showForm, setShowForm] = useState(false);

  const toggleFormVisibility = () => {
    setShowForm(!showForm);
  };

  const handleAddToWatchlist = async () => {
    setLoading(true);
    setWatchlistMessage("");
    try {
      await addToWatchlist(movieId, 5, note, apiKey); // default rating = 5
      setWatchlistMessage("Movie added to your watchlist!"); // success message
      setNote(""); // clear note field after add
    } catch (error) {
      console.error("Error while adding movie to watchlist:", error); // debug

      if (error.response && error.response.status === 409) {
        setWatchlistMessage("This movie is already in your list!"); // if movie already existed in the watchlist or user already completed it
      } else if (error.message) {
        setWatchlistMessage(`Error: ${error.message}`);
      } else {
        setWatchlistMessage("Failed to add movie to watchlist. Please try again.");
      }
    } finally {
      setLoading(false); // Stop loading spinner
    }
  };

  return (
    <div className="add-to-watchlist">
      <button className="add-twl-button" onClick={toggleFormVisibility}>
        {showForm ? "Hide" : "Add to Watchlist"}
      </button>
      {showForm && (
        <div>
          <textarea
            value={note}
            onChange={(e) => setNote(e.target.value)}
            placeholder="Enter a note for this movie..."
            rows="3"
            style={{ width: "100%" }}
          />
          <button className="add-twl-submit" onClick={handleAddToWatchlist} disabled={loading}>
            {loading ? "Adding..." : "Submit"}
          </button>
          {watchlistMessage && <p>{watchlistMessage}</p>}
        </div>
      )}
    </div>
  );
};

export default AddToWatchlist;
```

- components/Header.jsx

```
import React from "react";
import { NavLink } from "react-router-dom";
import { useAuth } from "../context/AuthContext";
import SearchBar from "./SearchBar";
import "../styles/Header.css";

const Header = () => {
  const { apiKey, logout } = useAuth();

  return (
    <header className="header">
      <h1>
        <NavLink to="/">MovieLand</NavLink>
      </h1>

      <nav>
        <NavLink to="/">Home</NavLink>
        <NavLink to="/watchlist">Watch List</NavLink>
        <NavLink to="/completedlist">Completed List</NavLink>
        <NavLink to="/user-stats">User Stats</NavLink>
      </nav>

      {apiKey ? (
        <button onClick={logout} className="logout-button">Log
Out</button>
      ) : (
        <NavLink to="/login" className="login-button">
          Log In
        </NavLink>
      )}

      <SearchBar />
    </header>
  );
};

export default Header;
```

- components/LoginPage.jsx

```
import React, { useState } from 'react';
import { useAuth } from '../context/AuthContext';

const LoginPage = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const { login } = useAuth();

  const handleSubmit = (e) => {
    e.preventDefault();
    login(username, password);
  };

  return (
    <div className='login-page'>
      <h1>Login</h1>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          placeholder="Username"
          required
        />
        <input
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          placeholder="Password"
          required
        />
        <button className='login-button' type="submit">Login</button>
      </form>
    </div>
  );
};

export default LoginPage;
```

- components/MovieCard.jsx

```
import React, { useState } from "react";

const MovieCard = ({ movie, onUpdatePriority, onDeleteEntry,
onMarkAsWatched }) => {
  const [showForm, setShowForm] = useState(false);
  const [showPriority, setShowPriority] = useState(false);
  const [newPriority, setNewPriority] = useState("");
  const [rating, setRating] = useState("");
  const [notes, setNotes] = useState("");
  const togglePriorityVisibility = () => {
    setShowPriority(!showPriority);
  };

  const toggleFormVisibility = () => {
    setShowForm(!showForm);
  };

  const handleChange = (e) => {
    setNewPriority(e.target.value);
  };

  const handleSubmit = () => {
    const priorityNumber = Number(newPriority); // validate priority

    if (!newPriority.trim() || isNaN(priorityNumber) || priorityNumber <=
0) {
      alert("Please enter a valid positive priority.");
      return;
    }

    console.log("Priority entered:", newPriority);  // debug
    console.log("Updating priority to:", priorityNumber); // debug

    onUpdatePriority(priorityNumber);
  };

  const handleDelete = () => {
    if (!movie.id || !movie.movieID) {
      alert("Cannot delete. Missing entry ID or movie ID.");
      return;
    }
    onDeleteEntry(movie.id, movie.movieID);
  };

  const handleWatched = async () => {
    const ratingNumber = (parseFloat)(rating);
    console.log("Rating:", rating, "Notes:", notes);

    const entryId = movie.id;
    console.log("Entry ID:", entryId); // debug

    if (rating && (isNaN(ratingNumber) || ratingNumber < 1 || ratingNumber
> 10)) {
        alert("Please enter a rating between 1 and 10.");
        return;
    }

    try {
        console.log("handleWatched called with:", { entryId, rating:
ratingNumber, notes });
        await onMarkAsWatched(movie.movieID, entryId, ratingNumber,
notes);
        setRating("");  // reset to empty after marked
        setNotes("");
    } catch (error) {
        console.error("Error marking movie as watched:", error.response ||
error.message);
        alert("Failed to mark movie as watched. Please try again later.");
    }
  };


  return (
    <div className="movie-card">
      <h2>{movie.title}</h2>
      <img src={movie.poster} alt={movie.title} />
      <p><strong>Movie Description: </strong>{movie.overview}</p>
      <p><strong>Priority in your list: </strong>{movie.priority}</p>
      <p><strong>Rating: </strong>{movie.rating}</p>
      <p><strong>Your Note: </strong>{movie.notes}</p>

      <button className="update-priority" onClick=
{togglePriorityVisibility}>
        {showPriority ? "Hide Priority" : "Update Priority"}
      </button>

      {showPriority && (
        <div>
          <div>
            <input
              type="number"
              min="1"
              placeholder="New Priority"
              value={newPriority}
              onChange={handleChange}
            />
            <button onClick={handleSubmit}>Submit</button>
          </div>
        </div>
      )}

      <button className="watched" onClick={toggleFormVisibility}>
        {showForm ? "Hide Rating & Notes" : "Mark as Watched"}
      </button>

      {showForm && ( // only show show form if user want to mark as
watched a movie
        <div>
          <label>
              Rating:
              <input
                  type="number"
                  value={rating}
                  onChange={(e) => setRating(e.target.value)}
                  min="1"
                  max="10"
              />
          </label>
          <div>
            <textarea
                placeholder="Add notes"
                value={notes}
                onChange={(e) => setNotes(e.target.value)}
            />
          </div>
          <button onClick={handleWatched}>Submit Rating & Notes</button>
        </div>
      )}

      <button className="delete-button" onClick={handleDelete}>
        Delete from List
      </button>
    </div>
  );
};

export default MovieCard;
```

- components/NotLoggedIn

```
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const ProtectedRoute = ({ children }) => {
  const { apiKey, userId } = useAuth();
  if (!apiKey || !userId) {
    return <Navigate to="/login" replace />;
  }
  return children;
};

export default ProtectedRoute;
```

- components/SearchBar.jsx

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import "../styles/SearchBar.css";

const SearchBar = () => {
  const [query, setQuery] = useState(""); // Search input state
  const navigate = useNavigate();

  const handleSearch = (e) => {
    e.preventDefault();
    if (query.trim()) {
      navigate(`/search?search=${encodeURIComponent(query)}`);
    }
  };

  return (
    <form onSubmit={handleSearch}>
      <input
        type="text"
        placeholder="Search for a movie..."
        value={query}
        onChange={(e) => setQuery(e.target.value)}
      />
      <button className="search-button" type="submit"></button>
    </form>
  );
};

export default SearchBar;
```

- components/SearchResults

```
import React, { useState, useEffect } from "react";
import { useLocation } from "react-router-dom";
import { useNavigate } from "react-router-dom";
import { searchMovies } from "../api";
import AddToWatchlist from "./AddToWatchList";
import Header from "./Header";

const SearchResults = () => {
    const location = useLocation();
    const query = new URLSearchParams(location.search).get("search"); //
get the search queyr
    const [movies, setMovies] = useState([]);
    const [loading, setLoading] = useState(false);
    const [error, setError] = useState(null);

    const navigate = useNavigate();

    const handleClick = (id) => {
      navigate(`/movie/${id}`);
    };

    useEffect(() => {
    if (!query) return;

    const fetchSearchResults = async () => {
        setLoading(true);
        setError(null);

        try {
            const response = await searchMovies(query);
            console.log("Movies returned:", response);  // log the
response for debug


            if (response?.results && response.results.length > 0) {
                setMovies(response.results);
            } else {
                setMovies([]);  // if no results, clear the movies state
            }
        } catch (err) {
            setError(err.response?.data?.error || err.message || "Failed
to fetch search results.");
        } finally {
            setLoading(false);
        }
    };

    fetchSearchResults();
}, [query]);

    return (
    <main>
        <Header />
        <div className="search-results">
            <h1>Search Results for "{query}"</h1>
            {loading && <p>Loading...</p>}
            {error && <p className="error">{error}</p>}
            <div className="movie-results">
                {movies.length > 0 ? (movies.map((movie) => (
                    <div key={movie.id} className="searched-movie">
                        <h2>
                            {movie.title.split(" ").slice(0, 4).join(" ")}
                            {movie.title.split(" ").length > 4 ? "..." :
""}
                        </h2>
                        <img src={movie.poster} alt={movie.title}
className="movie-poster" />
                        <button onClick={() => handleClick(movie.id)}>View
more details</button>
                        <AddToWatchlist movieId={movie.id} />
                    </div>
                    ))
                ) : (
                    !loading && <p>No movies found for "{query}".</p>
                )}
            </div>
        </div>
    </main>
    );
};

export default SearchResults;
```

- components/UserStat.jsx

```
import React, { useState, useEffect } from "react";
import { fetchUserStats } from "../api";
import { useAuth } from "../context/AuthContext";
import Header from "./Header";

const UserStats = () => {
  const { apiKey, userId } = useAuth();
  const [stats, setStats] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchStats = async () => {
      if (!apiKey || !userId) {
        setError("Please log in to view your statistics.");
        setLoading(false);
        return;
      }

      try {
        const data = await fetchUserStats(userId, apiKey);
        setStats(data);
        setLoading(false);
      } catch (error) {
        setError("Error fetching stats: " + error.message);
        setLoading(false);
      }
    };

    fetchStats();
  }, [apiKey, userId]);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>{error}</div>;

  return (
    <main>
    <Header />
    <div className="user-stats">
      <h1>Your Statistics</h1>
      {stats && (
      <div className="stats">
        <p>Total Movies Watched: {stats.total_movies_watched}</p>
        <p>Total Watch Time: {stats.total_watched_times} hours</p>
        <p>Average Rating: {parseFloat(stats.average_rating).toFixed(1)}
</p>
        <p>Movies Planned to Watch: {stats.plan_to_watch}</p>
      </div>
      )}
    </div>
    </main>
  );
};

export default UserStats;
```

- context/ AuthContext.jsx

```
import React, { createContext, useState, useContext, useEffect } from
'react';
import { useNavigate } from 'react-router-dom';

const AuthContext = createContext();
export const useAuth = () => {
  return useContext(AuthContext);
};

export const AuthProvider = ({ children }) => {
  const [apiKey, setApiKey] = useState(null);
  const [userId, setUserId] = useState(null);
  const [loading, setLoading] = useState(true);
  const navigate = useNavigate();

  useEffect(() => {
    const storedApiKey = localStorage.getItem('apiKey');
    const storedExpiryTime = localStorage.getItem('apiKeyExpiry');
    const storedUserId = localStorage.getItem('userId');

    console.log('Stored API Key:', storedApiKey); // debug
    console.log('Stored Expiry Time:', storedExpiryTime); // debug
    console.log('Stored User ID:', storedUserId);

    if (storedApiKey && storedExpiryTime && storedUserId) {
      const currentTime = Date.now();
      console.log('Current time:', currentTime);

      if (currentTime < parseInt(storedExpiryTime)) {
        setApiKey(storedApiKey); // set api key
        setUserId(storedUserId);
      } else {
        // remove api key in local storage
        localStorage.removeItem('apiKey');
        localStorage.removeItem('apiKeyExpiry');
        localStorage.removeItem('userId');
        setApiKey(null);
        setUserId(null);
        console.log('API key has expired');
        // navigate('/login'); // redirect to log in page if time expires
      }
    // } else {
    //   console.log('No API key or userId found in localStorage');
    //   navigate('/login'); // no api key is found (not logged in)
    }
```

```
      setLoading(false);
    }, []);

    const login = (username, password) => {
      fetch('https://loki.trentu.ca/~litran/3430/assn/assn2-
tlinhh10102003/api/users/session', {
        method: 'POST',
        body: JSON.stringify({ username, password }),
        headers: {
          'Content-Type': 'application/json',
        },
      })
        .then((response) => {
          if (!response.ok) {
            throw new Error('Failed to authenticate');
          }
          return response.json();
        })
        .then((data) => {
          if (data['Your API key'] && data.user_id) {
            const apiKey = data['Your API key'];
            const userId = data.user_id;
            setApiKey(apiKey);
            setUserId(userId);

            const expirationTime = Date.now() + 3600000; // 1 hour expiry
            localStorage.setItem('apiKey', apiKey);
            localStorage.setItem('apiKeyExpiry', expirationTime.toString());
            localStorage.setItem('userId', userId)

            console.log('API key and User ID stored in localStorage');
            navigate('/'); // naviagte to home after successful login
          } else {
            console.log('API key or User ID not found in response:', data);
            alert('Invalid credentials');
          }
        })
        .catch((error) => {
          console.error('Error logging in:', error);
          alert('Error logging in: ' + error.message);
        });
    };

    const logout = () => {
      setApiKey(null);
      setUserId(null);
      localStorage.removeItem('apiKey');
      localStorage.removeItem('apiKeyExpiry');
      localStorage.removeItem('userId');
      console.log('Logged out');
      navigate('/login');
    };
    if (loading) {
      return <div>Loading...</div>;
    }

    return (
      <AuthContext.Provider value={{ apiKey, userId, login, logout }}>
        {children}
      </AuthContext.Provider>
    );
  };
```

- styles/ Header.css

```
.header {
    display: grid;
    grid-template-columns: repeat(4, auto);
    color: #ffc567;
    font-family: "Unlock";
    width: 90%;
    margin: 0.5em auto;
    padding: 0.5em 0.8em;
    border: solid #b64931;
    border-radius: 0.6em;
    background-color: #b64931;
    justify-items: auto;
    box-shadow: 0 0 15px rgba(0, 0, 0, 0.3);
    position: relative;
    z-index: 2;
}
.header h1 {
    text-align: center;
}
.header h1 a {
    font-size: 2.9em;
    text-shadow: 2px 2px black;
    color: #ffc567;
    transition: all 0.3s ease;
}
.header h1 a:hover {
    color: #00995e;
}

nav {
    display: grid;
    grid-template-columns: repeat(4, auto);
    align-content: center;
}
nav a {
    margin: 0.5em 0.8em;
    color: #ffc567;
    text-decoration: none;
    border: solid #b64931;
    border-radius: 1em;
    text-align: center;
    padding: 0.5em;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    transition: background-color 0.3s, transform 0.2s;
}
nav a:hover {
    background-color: #6c7b9f;
    border-color: #6c7b9f;
    transform: scale(1.2);
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.1);
}

.logout-button,
.login-button {
    font-family: "Unlock";
    color: white;
    background-color: #b64931;
    border: solid #b64931;
    border-radius: 0.5em;
    padding: 0.5em;
    margin: auto;
    transition: background-color 0.3s, transform 0.2s;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
.logout-button:hover,
.login-button:hover {
    color: #ffc567;
    background-color: #6c7b9f;
    border-color: #6c7b9f;
    transform: scale(1.2);
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
}
```

- styles/ MovieList.css

```


/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
*/
@import url('https://fonts.googleapis.com/css2?
family=Unlock&display=swap');
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
    display: block;
}
body {
    line-height: 1;
}
ol, ul {
    list-style: none;
}
blockquote, q {
    quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
    content: '';
    content: none;
}
/*#########################################


##############################################*/
:root {
    --main-color: #b64931;
    --2nd-main-color: #ffc567;
}

table {
    border-collapse: collapse;
    border: solid black;
    letter-spacing: 0.1em;
    width: 90%;
    margin: 1em auto;
}
th, td {
    border: 0.1em solid black;
    padding: 1em;
    text-align: center;
}
td {
    display: table-cell;
    vertical-align:middle;
    padding: 0.5em 1em 0.5em 0.5em;
}
th {
    display: table-cell;
    vertical-align:middle;
    font-weight: bold;
    text-align: center;
    padding: 1em;
    background-color: var(--2nd-main-color);
    color: var(--main-color);
}
tr {
    display: table-row;
    vertical-align: inherit;
}

strong {
    font-weight: 700;
}

textarea {
    text-align: center;
}

input {
    cursor: pointer;
}
/*###########################################

HOME COMPONENT

##############################################*/
.movie-list h1,
.watch-list h1,
.completed-list h1,
.login-page h1,
.search-results h1,
.user-stats h1 {
    font-family: "Unlock";
    width: 50%;
    font-weight: 600;
    font-size: x-large;
    text-align: center;
    border-radius: 0.6em;
    color: var(--main-color);
    background-color: var(--2nd-main-color);
    margin: 0.5em auto 0.5em auto;
    padding: 0.5em;
}

.pagination {
    display: flex;
    justify-content: center;
    text-align: center;
}
.pagination button,
.movie-section button,
.add-to-watchlist button,
.movie-results button {
    font-family: "Unlock";
    display: flex;
    justify-content: center;
    align-items: center;
    color: #00995e;
    background-color: var(--2nd-main-color);
    border: solid var(--2nd-main-color);
    border-radius: 0.5em;
    padding: 0.5em;
    margin: 0.8em auto 1.2em auto;
    transition: background-color 0.3s, transform 0.2s;
    cursor: pointer;
}
.pagination button:hover,
.movie-section button:hover,
.movie-card button:hover
.add-to-watchlist button:hover,
.movie-results button:hover,
.update-priority:hover,
.watched:hover,
.update-times-watched:hover,
.update-rating:hover,
.add-twl-button:hover,
.add-twl-submit:hover,
.update-rating-submit:hover {
    color: var(--2nd-main-color);
    background-color: var(--main-color);
    border-color: var(--main-color);
    transform: scale(1.2);
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
}

.pagination button.active {
    transform: scale(1.2);
    background-color: var(--main-color);
    border-color: var(--main-color);
    color: var(--2nd-main-color);
}
.pagination button:disabled {
    cursor: not-allowed;
}

p {
    /* font-family: "Unlock"; */
    text-align: center;
    margin-top: 1em;
}
```

```
.movie-section {
    display: grid;
    grid-template-columns: repeat(5, 2fr);
    gap: 1.5em;
    padding: 1em;
}

.movie-item,
.searched-movie {
    padding: 1em;
    text-align: center;
    border-radius: 0.5em;
    transition: transform 0.3s;
    box-shadow: 0 0 15px rgba(0,0,0,0.1);
}
.movie-item:hover,
.watchlist-container .movie-card:hover,
.completedlist-container .movie-card:hover,
.searched-movie:hover {
    transform: scale(1.05);
    color: white;
    border: 0.1em solid #ccc;
    padding: 1em;
    text-align: center;
    background: #0B3F30;
    border-radius: 0.5em;
    box-shadow: 0 0 15px rgba(0,0,0,0.1);
}
.movie-item h2,
.searched-movie h2,
.movie-card h2 {
    font-family: "Unlock";
    margin: 0 auto 0.5em auto;
    text-align: center;
}

.movie-poster {
    width: 100%;
    height: auto;
    margin-bottom: 1em;
    border-radius: 0.5em;
}

.movie-title {
    overflow: hidden;
    white-space: nowrap;
    text-overflow: ellipsis;
    display: inline-block;
    cursor: pointer;
}

/*############################################

MOVIE CARD COMPONENT
```

```
##############################################*/
.movie-card {
    width: 70%;
    margin: 1em auto;
    padding: 1em;
    text-align: center;
    border-radius: 0.5em;
    transition: transform 0.3s;
    box-shadow: 0 0 15px rgba(0,0,0,0.1);
}

.movie-card img,
.searched-movie {
    width: 20em;
    height: auto;
    margin: 0.5em auto;
    border-radius: 0.5em;
}

.movie-card input {
    border-radius: 0.4em;
    padding: 0.5em;
    margin: 1em auto 0.5em auto;
}

.movie-card button {
    font-family: "Unlock";
    display: flex;
    justify-content: center;
    align-items: center;
    color: #00995e;
    background-color: var(--2nd-main-color);
    border: solid var(--2nd-main-color);
    border-radius: 0.5em;
    padding: 0.5em;
    margin: 1em auto 0 auto;
    transition: background-color 0.3s, transform 0.2s;
    cursor: pointer;
}

.movie-card .delete-button {
    margin: 1.2em auto 0.5em auto;
    transition: background-color 0.3s, transform 0.2s;
}
.movie-card .delete-button:hover {
    color: white;
    background-color: red;
    border: red;
    transform: scale(1.2);
}

.movie-detail textarea {
    width: 60%;
    padding: 0.5em;
    margin: 1em auto;
}

.movie-detail h1 {
    font-family: "Unlock";
    font-size: xx-large;
    margin: 0 auto 0.5em auto;
    text-align: center;
}

.details {
    display: flex;
    flex-direction: column;
    justify-self: center;
    width: 50%;
}

.update-priority {
    display: flex;
    flex-direction: row;
}
.update-priority button {
    margin-left: 0;
}
.update-priority input {
    width: 30%;
    margin: 0.85em 0.5em auto auto;
}

/*###########################################

WATCH LIST, COMPLETED LIST, ADD TO WATCH LIST, SEARCH RESULTS, USER STATS

############################################*/
.watchlist-container,
.completedlist-container,
.movie-results {
    display: grid;
    grid-template-columns: repeat(3, 2fr);
    gap: 1.5em;
    padding: 1em;
}
.watchlist-container .movie-card,
.completedlist-container .movie-card {
    border-radius: 0.5em;
    transition: transform 0.3s;
    box-shadow: 0 0 15px rgba(0,0,0,0.1);
}

.watch-list h3,
.completed-list h3 {
    font-family: "Unlock";
    font-size: x-large;
    margin: 0 auto 0.5em auto;
    text-align: center;
}

.add-to-watchlist {
    margin: 0.2em auto 0 auto;
    width: 60%;
}

.rating input {
    margin-left: 1em;
}

.stats p {
    font-family: "Unlock";
}

.feedback-message,
.success-message {
    color: green; /* Success */
    font-size: 0.9rem;
    margin-top: 10px;
}

.feedback-message.error,
.error-message {
    color: red; /* Error */
}

/*###########################################

LOGIN PAGE

############################################*/
.login-page form {
    display: flex;
    flex-direction: column;
    justify-items: center;
    margin: 0.5em auto 0 auto;
    padding: 0.5em;
    width: 30%;
    transition: border-color 0.3s;
}
.login-page form input {
    justify-items: center;
    margin: 1em auto 1em auto;
    padding: 1em;
    border-color: black;
    border-radius: 1em;
    width: 50%;
    transition: border-color 0.3s;
}
.login-page form input:hover,
.login-page form input:focus {
    border-color: var(--2nd-main-color);
}
.login-button {
    margin: 0.5em auto;
}

/*##########################################

FILTER MOVIES

#############################################*/
.filter-container {
    width: 20%;
    font-family: "Unlock";
    display: flex;
    justify-content: center;
    align-items: left;
    border-radius: 0.5em;
    padding: 0.5em;
    margin-left: 1em;
    transition: background-color 0.3s, transform 0.2s;
}
#year-filter {
    font-family: "Unlock";
    margin-left: 1em;
}
```

- styles/ SearchBar.css

```


form {
    display: flex;
    justify-content: center;
    align-items: center;
}
form input {
    display: flex;
    justify-content: center;
    align-items: center;
    font-family: "Unlock";
    border-radius: 0.5em;
    padding: 0.8em;
    width: 70%;
}
.search-button {
    padding: 0;
    margin-left: 1em;
    border: none;
    color: #b64931;
    background-color: #b64931;
}
```

- api.jsx

```
import axios from "axios";

// API URL
const API_URL = "https://loki.trentu.ca/~litran/3430/assn/assn2-
tlinhh10102003/api";

export const getMoviesPaginated = async (resultsPerPage, currentPage) => {
    try {
      const response = await axios.get(`${API_URL}/movies`, {
        params: {
          page: currentPage,
          results_per_page: resultsPerPage,
        },
      });

      return response.data;
    } catch (error) {
      console.error("Error fetching movies:", error);
      throw error;
    }
};


export const getMovieById = async (id) => {
    try {
        const response = await axios.get(`${API_URL}/movies/${id}`,
            // headers: { "x-api-key": `${apiKey}` },
        );
        return response.data;
    } catch (error) {
        console.error("Error fetching movie details:", error);
        throw error;
    }
};

export const getMovieRating = async (id, apiKey) => {
    if (!apiKey) {
        throw new Error("API key is missing. Please log in.");
    }

    try {
        const response = await axios.get(`${API_URL}/movies/${id}/rating`,
{
            headers: { "x-api-key": `${apiKey}` },
        });
        return response.data;
    } catch (error) {
        console.error("Error fetching movie rating:", error);
        throw error;
    }
```

```
};

export const searchMovies = async (query) => {
    console.log("Passed query:", query);  // debug

    try {
        const encodedQuery = encodeURIComponent(query);
        const response = await axios.get(
            `${API_URL}/movies/search?q=${encodedQuery}`,
            {
                headers: {
                    'Content-Type': 'application/json',
                    // 'x-api-key': apiKey,
                },
            }
        );
        console.log("API Response:", response.data);
        return response.data;
    } catch (error) {
        console.error('Error searching for movies:', error.response?.data
|| error.message);
    }
};

export const getWatchListEntries = async (apiKey) => {
    try {
      const response = await axios.get(`${API_URL}/towatchlist/entries`, {
        headers: {
          "x-api-key": `${apiKey}`,
        },
      });
      return response.data;
    } catch (error) {
      console.error("Error fetching watchlist:", error);
      throw error; // Re-throw to be caught in the component
    }
};

export const addToWatchlist = async (movieId, priority = 5, notes = "",
apiKey) => {
    if (!apiKey) {
        throw new Error("API key is missing. Please log in.");
    }

    try {
        const response = await
axios.post(`${API_URL}/towatchlist/entries`,
        {
            movie_id: movieId,
            priority,
            notes
        },
        {
            headers: { "x-api-key": `${apiKey}` }
        });
        return response.data;
    } catch (error) {
        console.error("Error adding to watchlist:", error.response?.data
|| error.message);
        throw error;
    }
};


export const updateWatchListPriority = async (apiKey, entryId, priority,
movieID) => {
    try {
      console.log("Updating priority...");
      console.log(`Entry ID: ${entryId}, Priority: ${priority}, Movie ID:
${movieID}`);

      const response = await axios.put(
        `${API_URL}/towatchlist/entries/${entryId}/priority`,
        {
          priority: priority,
          movie_id: movieID,
        },
        {
          headers: {
            "x-api-key": apiKey,
            "Content-Type": "application/json"
          }
        }
      );

      console.log("Response:", response);
      return response.data;
    } catch (error) {
      console.error("Error updating priority:", error);
      if (error.response) {
        console.error("Response Error: ", error.response.data);
      }
      throw error;
    }
};


export const deleteWatchListEntries = async (apiKey, entryId, movieId) =>
{
    try {
        const response = await axios.delete(
          `${API_URL}/towatchlist/entries/${entryId}`,
          {
            headers: { "x-api-key": apiKey },
            data: { movie_id: movieId }
          }
        );
        return response.data;
    } catch (error) {
        console.error("Error in API request:", error);
        throw error;
    }
};

// Fetch completed watch movies
export const getCompletedMovies = async (apiKey) => {
    try {
        const response = await
axios.get(`${API_URL}/completedwatchlist/entries`, {
            headers: { "x-api-key": `${apiKey}` },
        });
        console.log("API Response:", response.data);
        return response.data;
    } catch (error) {
        console.error("Error fetching completed movies:", error);
        throw error;
    }
};

// Update times watched for a completed movie
export const updateTimesWatched = async (entryId, apiKey) => {
    if (!apiKey) {
        throw new Error("API key is missing. Please log in.");
    }

    try {
        const response  = await axios.patch(
            `${API_URL}/completedwatchlist/entries/${entryId}/times-
watched`,
            {},
            { headers: { "x-api-key": `${apiKey}` } }
        );

        if (response.status === 200) {
            console.log('Successfully updated times watched:',
response.data);
        }
    } catch (error) {
        console.error("Error updating times watched:", error);
        throw error;
    }
};

// Update a movie's rating
export const updateMovieRating = async (entryId, rating, apiKey) => {
    if (!apiKey) {
        throw new Error("API key is missing. Please log in.");
    }

    try {
        console.log("Received rating:", rating);
        if (rating && typeof rating === 'object' && rating.new_rating) {
            rating = rating.new_rating;
        }

        if (isNaN(rating) || rating == null) {
            console.error("Invalid rating:", rating);
            return;
        }

        console.log("Sending updated rating:", rating);

        const updatedMovie = { new_rating: rating };

        const response = await axios.patch(
            `${API_URL}/completedwatchlist/entries/${entryId}/rating`,
            updatedMovie,
            { headers: { 'Content-Type': 'application/json', 'x-api-key':
`${apiKey}` } }
        );

        console.log("Response from server:", response.data);
        return response.data;
    } catch (error) {
        console.error("Error updating movie rating:", error.response ||
error);
        if (error.response?.status === 404) {
            console.error(`Entry ID ${entryId} not found.`);
        }
        throw error;
    }
};

export const markMovieAsWatched = async ({ apiKey, entryId, rating, notes
}) => {
    try {
        // if not provided set to null
        const data = {
            note: notes && notes.trim() !== "" ? notes : null,
            rating: rating ? rating : null,
        };

        console.log("Request body:", data); // debug

        const response = await axios.post(
            `${API_URL}/towatchlist/entries/${entryId}/watched`,
            data,
            { headers: { 'Content-Type': 'application/json', 'x-api-key':
apiKey } }
        );

        console.log("Response:", response);

    } catch (error) {
        if (error.response) {
            console.error("Error marking movie as watched:",
error.response.data);
            console.error("Status Code:", error.response.status);
        } else if (error.request) {
            console.error("Error: No response received from server.");
        } else {
            console.error("Error in request setup:", error.message);
        }
    }
};

export const fetchUserStats = async (userId, apiKey) => {
    if (!userId) throw new Error("User ID is missing.");
    if (!apiKey) throw new Error("API key is missing.");

    try {
        const response = await
axios.get(`${API_URL}/users/${userId}/stats`, {
        headers: { 'x-api-key': apiKey },
        });
        return response.data;
    } catch (error) {
        throw new Error(error.response?.data?.error || "Failed to fetch
user stats.");
    }
};
```

- main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import { AuthProvider } from './context/AuthContext';
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from "./commponents/routes/Home";
import MovieDetails from "./commponents/routes/MovieDetails";
import WatchList from "./commponents/routes/WatchList";
import CompletedList from "./commponents/routes/CompletedList";
import Error from "./commponents/routes/Error";
import SearchResults from "./commponents/SearchResults";
import LoginPage from "./commponents/LoginPage";
import NotLoggedIn from "./commponents/NotLoggedIn";
import UserStats from "./commponents/UserStat";


const base = import.meta.env.BASE_URL;
console.log("url" + base);

ReactDOM.createRoot(document.getElementById("root")).render(

  <BrowserRouter basename={base}>
    <AuthProvider>
      <Routes>
```

```
          <Route path="/login" element={<LoginPage />} />
          <Route path="/" element={ <Home /> } />

          <Route path="/movie/:id" element={ <MovieDetails /> } />

          <Route path="/watchlist" element={
          <NotLoggedIn>
            <WatchList />
          </NotLoggedIn>
          } />

          <Route path="/completedlist" element={
          <NotLoggedIn>
            <CompletedList />
          </NotLoggedIn>
          } />

          <Route path="/user-stats" element={
          <NotLoggedIn>
            <UserStats />
          </NotLoggedIn>
          } />

          <Route path="/search" element={<SearchResults />} />

          <Route path="*" element={<Error />} />
        </Routes>
      </AuthProvider>
    </BrowserRouter>
);
```