|  |
| --- |
|  |
| Movie - 1  React Project - 2 |
| |  |  |  | | --- | --- | --- | | Bhavana | 6/19/24 | React | |

1. **src > index.js**

**import React from 'react';**

**import { createRoot } from 'react-dom/client';**

**import App from './App';**

1. \*\*`**import React from 'react**';`\*\*:

- This imports the React library, which is necessary for writing React components and using JSX syntax.

2. \*\*`**import { createRoot } from 'react-dom/client'**;`\*\*:

- This imports the `createRoot` function from `react-dom/client`. In React, `createRoot` is used to create a root entry point for rendering a React application in concurrent mode (introduced in React 18).

3. \*\*`**import App from './App';**`\*\*:

- This imports the `App` component from the `App.js` or `App.jsx` file in the same directory as this script. The `App` component is typically the top-level component that renders the entire application.

// Get the root element from the DOM

**const container = document.getElementById('root');**

4. \*\*`**const container = document.getElementById('root');**`\*\*:

- This line retrieves the DOM element with the id `'root'`. In many React applications, this is where the root of the React component tree will be rendered. It's assumed that there is an HTML element with `id="root"` in the `index.html` file where the React application will be mounted.

**// Create a root and render the App component**

**const root = createRoot(container);**

**root.render(<App />);**

5. \*\*`**const root = createRoot(container)**;`\*\*:

- This line uses the `createRoot` function imported earlier to create a root for rendering the React application. `createRoot` takes the container element (`container` in this case) where the React component tree will be mounted.

6. \*\*`**root.render(<App />);**`\*\*:

- Finally, `root.render(<App />);` renders the `App` component into the container (`#root` element). This kickstarts the entire React application, where React will render the `App` component and any other components it may include, recursively rendering the entire component tree starting from `App`.

**Summary**

In essence, this code sets up a basic React application by importing necessary modules (`React`, `createRoot`, and the `App` component), selecting the DOM element where the React app will be mounted (`#root`), creating a root entry point for React concurrent mode, and then rendering the `App` component into that root. This is a typical setup for a React application, ensuring that React can manage and render components efficiently.

1. **src > App.js**

This React component (`App`) sets up a simple movie search application that fetches data from the OMDB API based on user input. Let's break down the code and explain each part:

**import { useEffect, useState } from 'react';**

**import './App.css';**

**import SearchIcon from './search.svg';**

**import MovieCard from './MovieCard';**

**const API\_URL = 'https://www.omdbapi.com/?apikey=3aff002a';**

**`useEffect` and `useState**`: These are React hooks used for handling side effects (like data fetching) and managing state in functional components.

`**'./App.css'`, `'./search.svg'`, `'./MovieCard'`:** These import styles and components needed for the application.

**`API\_URL`:** This constant stores the base URL for the OMDB API with an API key.

**const App = () => {**

**const [movies, setMovies] = useState([]);**

**const [searchTerm, setSearchTerm] = useState('');**

- \*\*`useState`\*\*: Defines two state variables (`movies` and `searchTerm`).

- `movies` holds an array of movie data fetched from the API.

- `searchTerm` keeps track of the user's input in the search box.

**const searchMovie = async (title) => {**

**try {**

**const response = await fetch(`${API\_URL}&s=${title}`);**

**const data = await response.json();**

**if (data.Search) {**

**setMovies(data.Search);**

**} else {**

**setMovies([]);**

**}**

**} catch (error) {**

**console.error("Error fetching data: ", error);**

**setMovies([]);**

**}**

**}**

- \*\*`searchMovie`\*\*: This async function fetches movie data from the OMDB API based on the provided `title`.

- It updates the `movies` state with the fetched data if successful, otherwise sets `movies` to an empty array on error.

**useEffect(() => {**

**if (searchTerm) {**

**searchMovie(searchTerm);**

**} else {**

**searchMovie('spiderman');**

**}**

**}, [searchTerm]);**

- \*\*`useEffect`\*\*: Runs whenever `searchTerm` changes (i.e., when the user types in the search box).

- Calls `searchMovie` with the current `searchTerm` to fetch movies matching the input.

- Defaults to searching for 'spiderman' if `searchTerm` is empty.

**return (**

**<div className="app">**

**<h1>Movie Land</h1>**

**<div className="search">**

**<input**

**type="text"**

**placeholder="Search for movies"**

**value={searchTerm}**

**onChange={(e) => setSearchTerm(e.target.value)}**

**/>**

**<img**

**src={SearchIcon}**

**alt="search"**

**onClick={() => searchMovie(searchTerm)}**

**/>**

**</div>**

**{**

**movies?.length > 0**

**? (**

**<div className="container">**

**{movies.map((movie) => (**

**<MovieCard key={movie.imdbID} movie={movie} />**

**))}**

**</div>**

**) : (**

**<div className="noMovies">**

**<h2>No movies found......</h2>**

**</div>**

**)**

**}**

**</div>**

**);**

- \*\*JSX Structure\*\*: Defines the UI for the application.

- Displays a title (`<h1>Movie Land</h1>`).

- Includes a search input box (`<input>`) and a search icon (`<img>`) that triggers `searchMovie` when clicked.

- Conditionally renders movie data (`movies`) using `<MovieCard>` component if there are results, or shows a message (`<h2>No movies found......</h2>`) if no results are found.

**Summary**

This React component (`App`) integrates state management, data fetching using `fetch` API, conditional rendering, and event handling to create a responsive movie search application. Users can search for movies by title, and the application dynamically updates the displayed results based on user input.

1. **src > MovieCard.jsx**

**import React from 'react';**

**const MovieCard = ({ movie }) => {**

**return (**

**<div className="movie" title={`${movie.Title} (${movie.Year})`}>**

**<div> <p>{movie.Year}</p> </div>**

**<div>**

**<img src={movie.Poster !== 'N/A' ? movie.Poster : 'https://via.placeholder.com/400'} alt={movie.Title} />**

**</div>**

**<div>**

**<span>{movie.Type}</span>**

**{/\* <h3>{movie.Title}</h3> \*/}**

**</div>**

**</div>**

**);}**

**export default MovieCard;**

The `MovieCard` component in React is a functional component that displays movie information in a card format.

1. \*\***Props**\*\*: Accepts a `movie` prop which contains details like `Title`, `Year`, `Poster`, and `Type`.

2. \*\***Structure**\*\*:

- Renders the movie's year (`<p>{movie.Year}</p>`).

- Displays the movie's poster if available (`<img src={movie.Poster !== 'N/A' ? movie.Poster : 'https://via.placeholder.com/400'} alt={movie.Title} />`).

- Shows the movie's type (`<span>{movie.Type}</span>`).

- Optionally, it can display the movie's title as an `<h3>` (commented out).

3. \*\***Hover Title**\*\*: Sets the title attribute of the outer `<div>` to show the movie's title and year when hovered over.

4. \*\***Export**\*\*: Exported as the default component to be used elsewhere in the application.

**Summary**

This component is designed to render a consistent card layout for displaying movie details, enhancing reusability and maintaining a clean, organized structure within a larger React application.