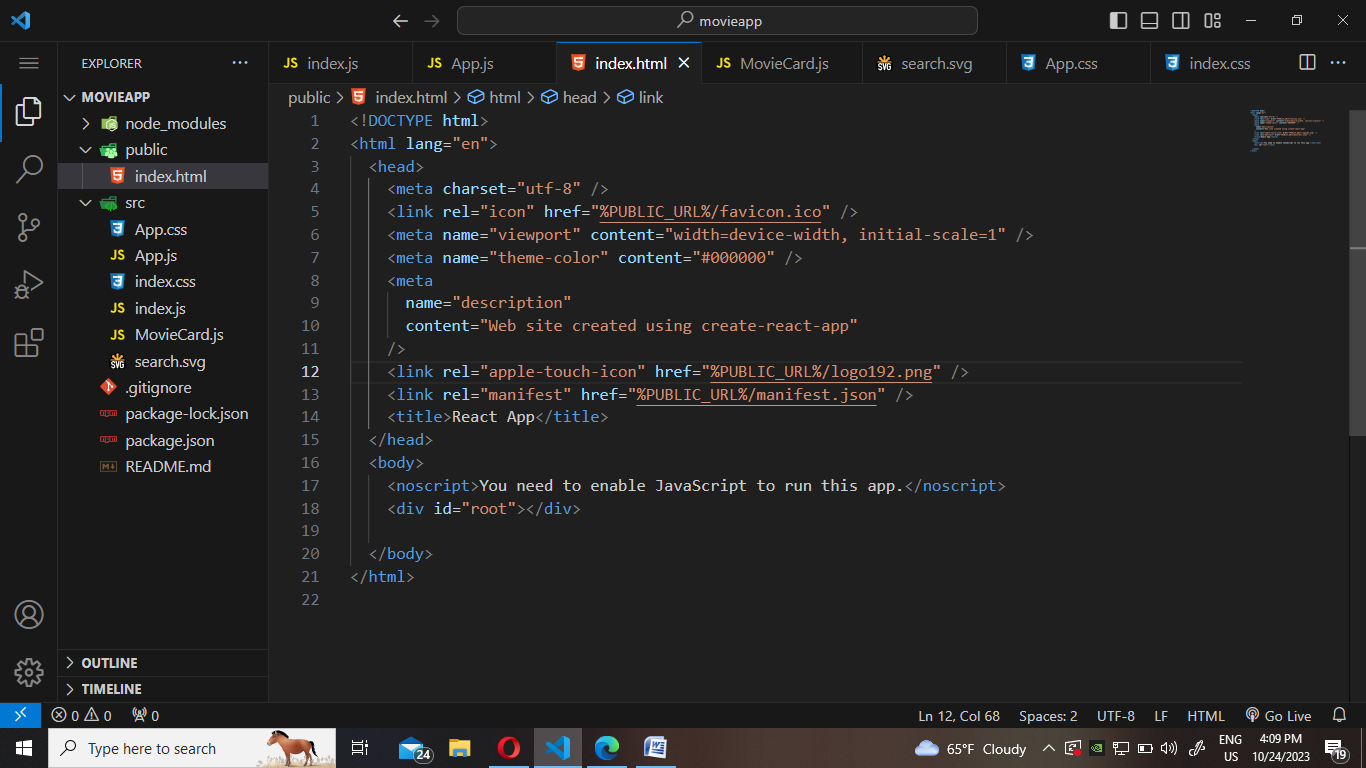
**MOVIESP**

**SCREENSHOTS AND EXPLAINATION**

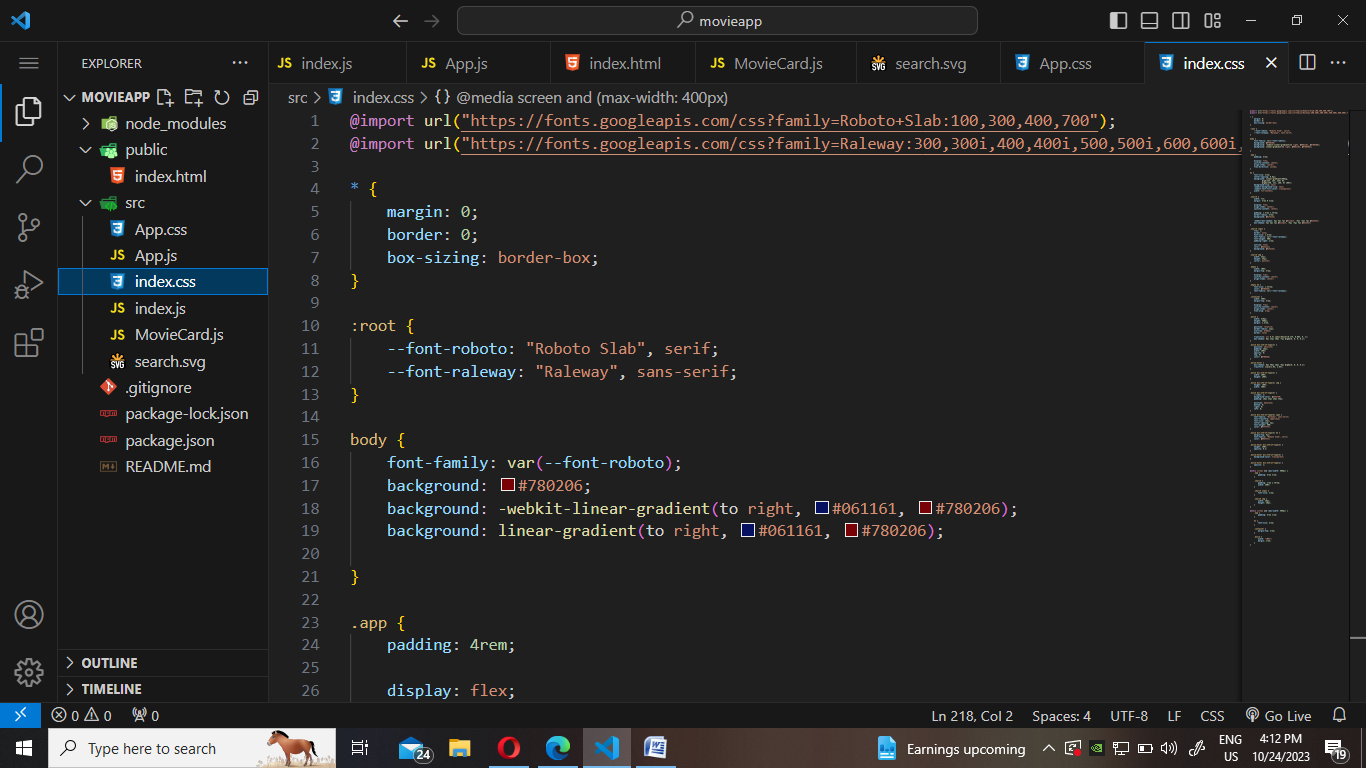
1.INDEX.HTML

****

**🡪<body>:** This is the body section of the HTML document, where the visible content of the web page is placed.

* **<div id="root"></div>**: Within the body section, there's a **div** element with an **id** attribute set to "root." This is often used as a placeholder for rendering React components. React applications typically use a single root element as the entry point for the application, and this is where the React app will be injected.

INDEX.CSS

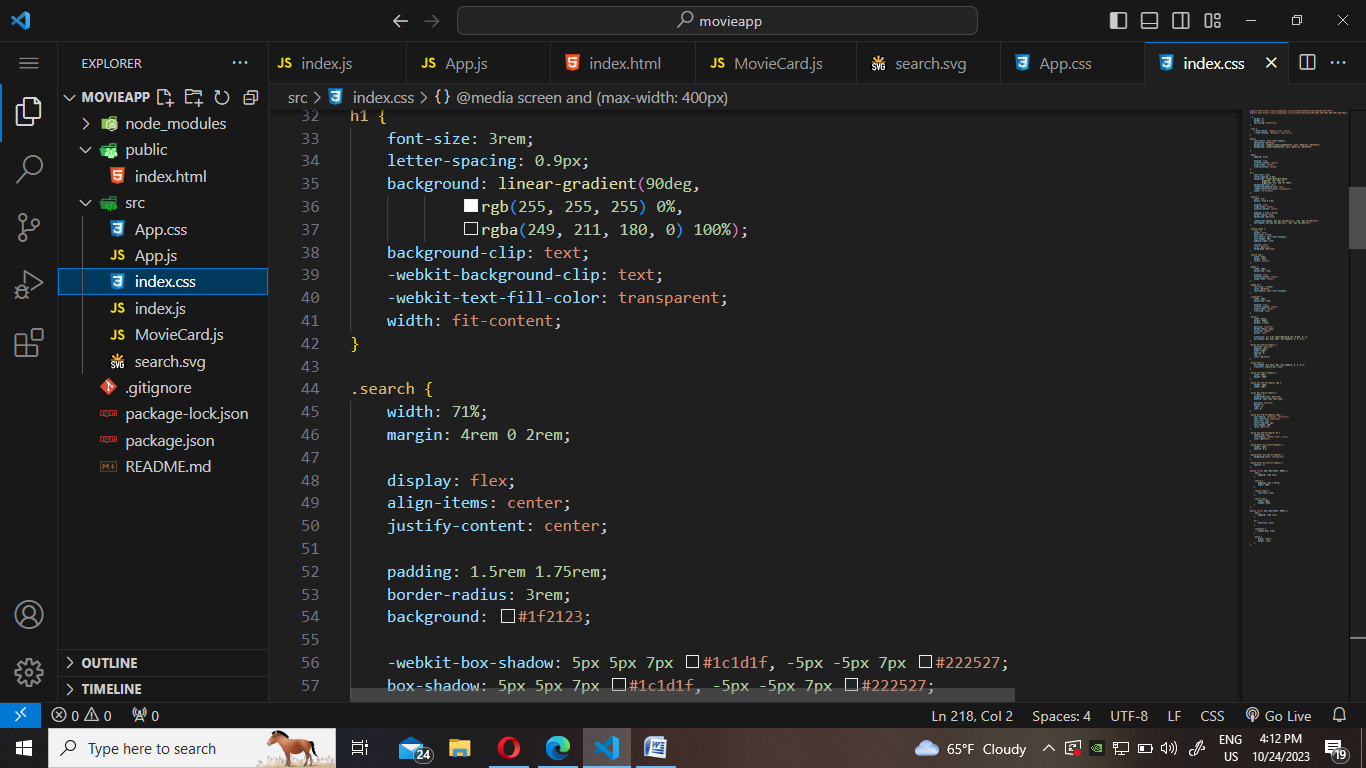


**@import** Statements:

* **@import url("https://fonts.googleapis.com/css?family=Roboto+Slab:100,300,400,700");**: This **@import** statement is used to import the "Roboto Slab" font family from Google Fonts. It specifies various font weights (100, 300, 400, and 700) that can be used within the stylesheet.
* **@import url("https://fonts.googleapis.com/css?family=Raleway:300,300i,400,400i,500,500i,600,600i,700,700i,800,800i,900,900i");**: Similarly, this statement imports the "Raleway" font family from Google Fonts and specifies a wide range of font weights and styles that can be used.

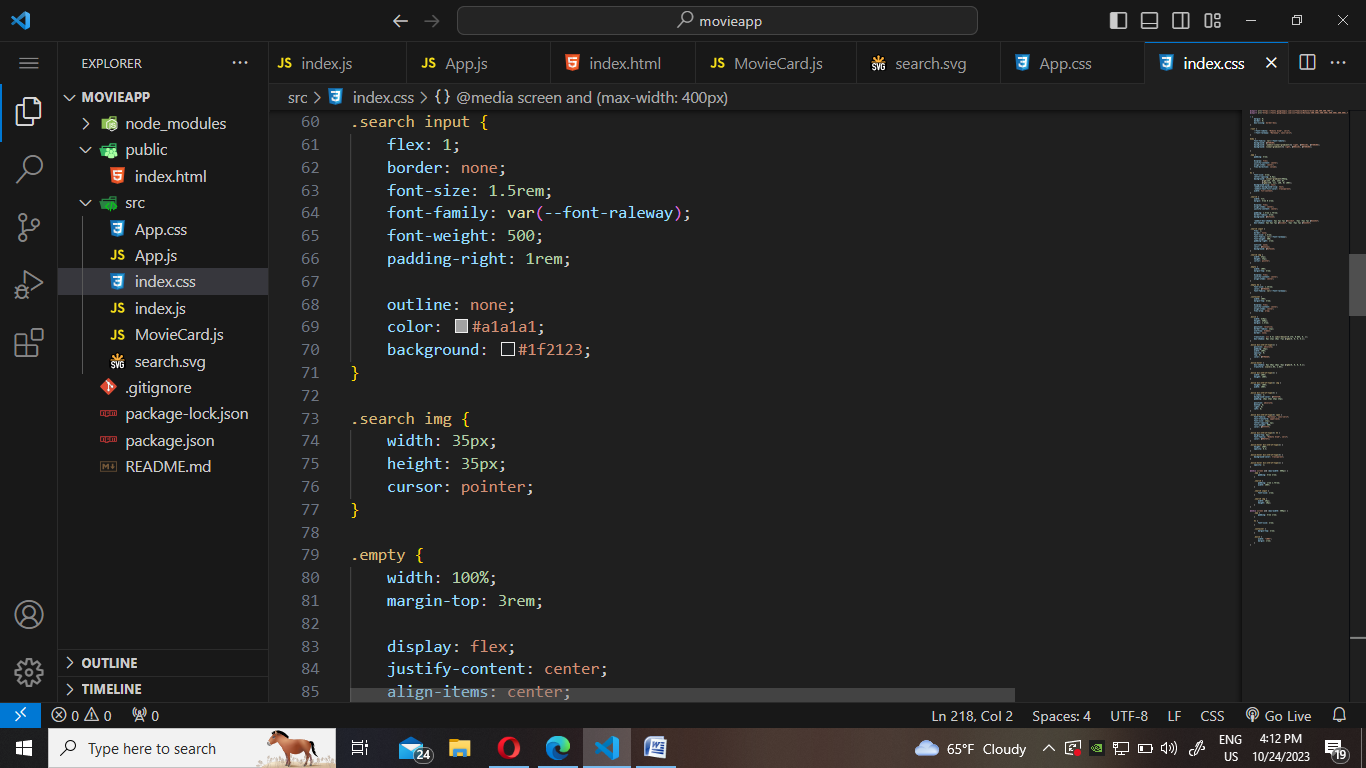
**body** Selector:

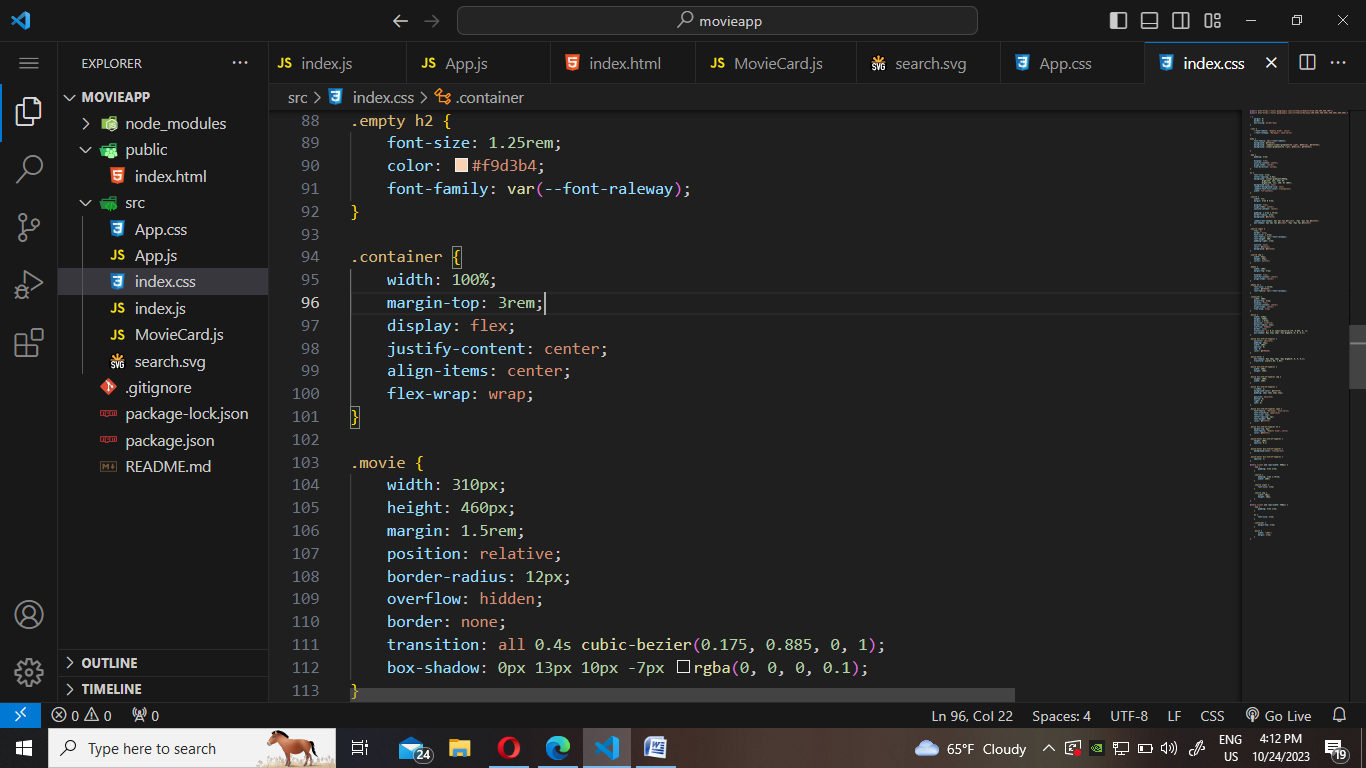
* The **body** selector is used to style the entire body of the web page.
* **font-family** is set to **var(--font-roboto)**, which means it will use the "Roboto Slab" font from Google Fonts for the text within the body.
* The **background** property is used to create a linear gradient background from #061161 to #780206. This gives the body a gradient background color.

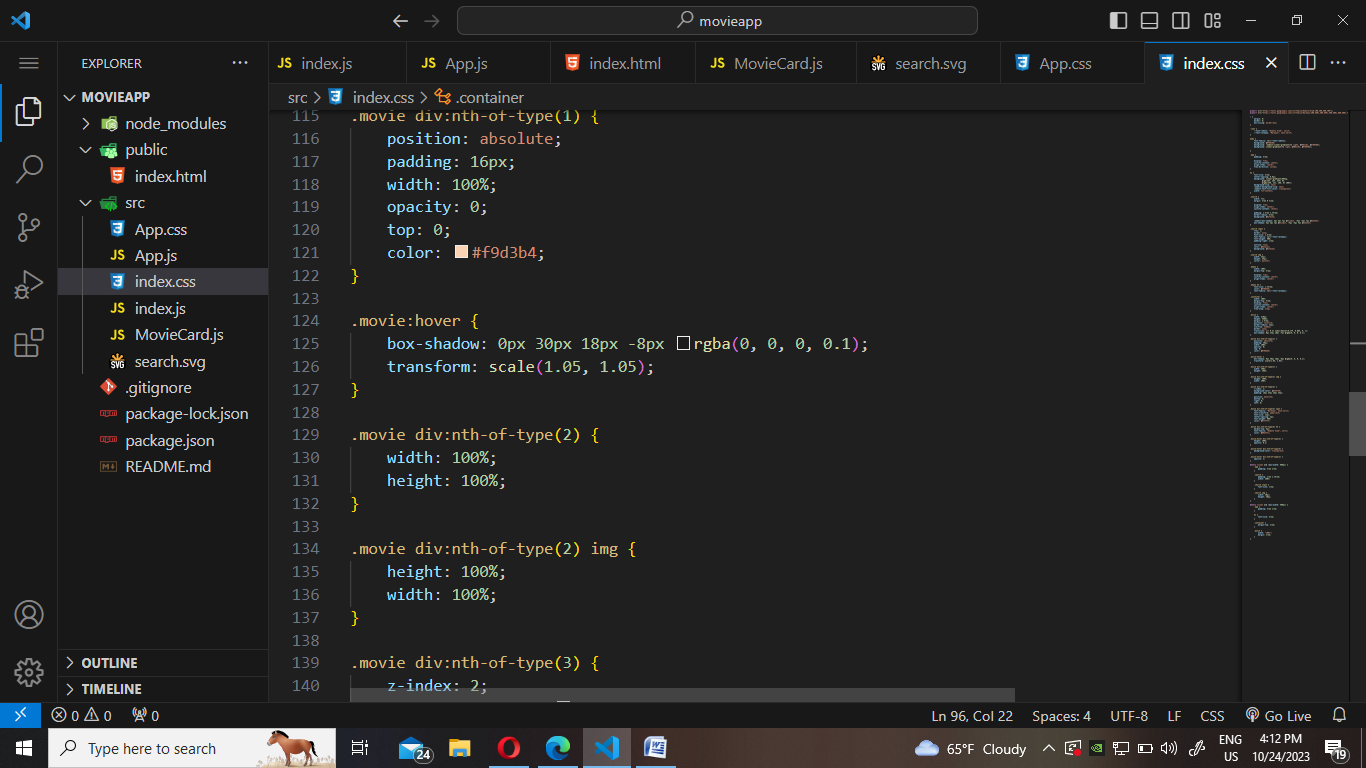


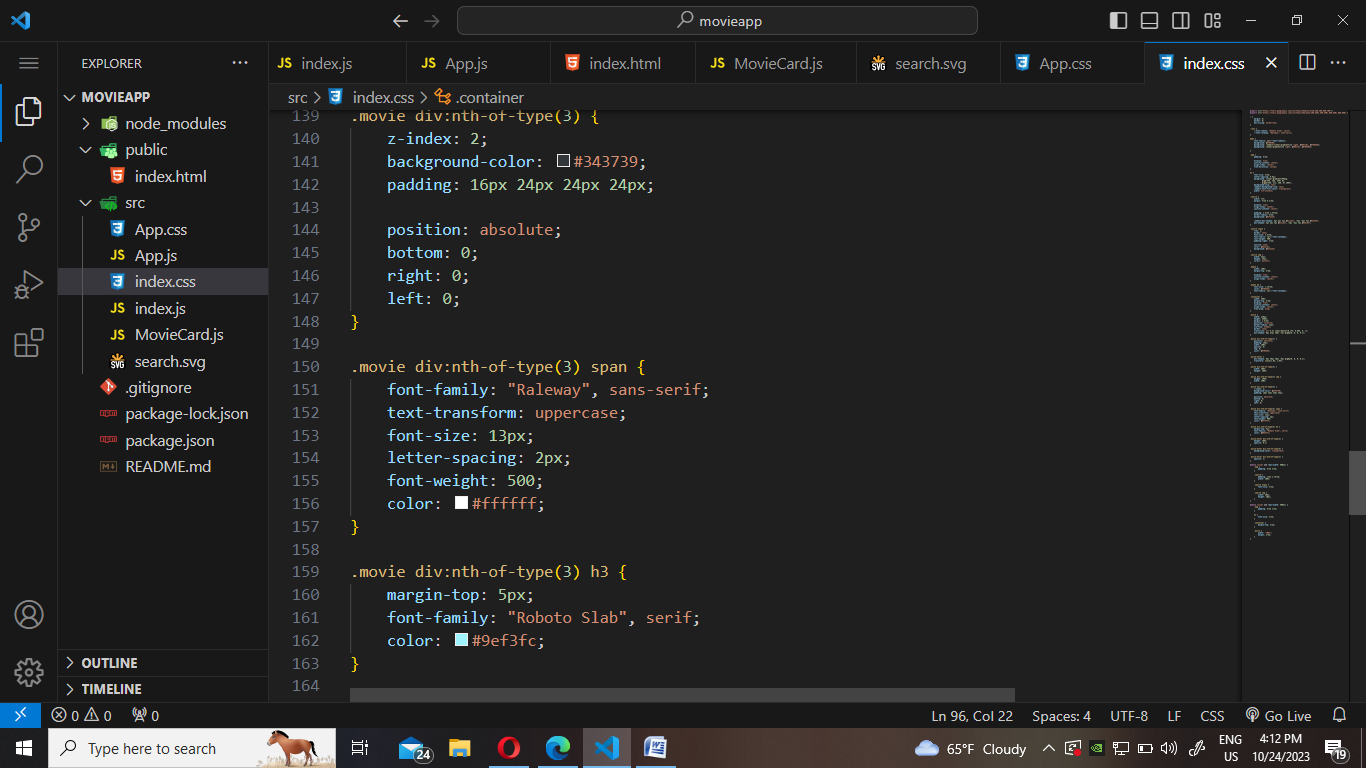
H1:

1. **-webkit-background-clip: text;**:
   * This property is used to control how a background is applied to an element. When it's set to **text**, it means that the background should be clipped to the shape of the text content. In other words, the background will only be visible within the area where the text is displayed.
2. **-webkit-text-fill-color: transparent;**:
   * This property sets the color of the text itself. When it's set to **transparent**, it effectively makes the text transparent, rendering it invisible.

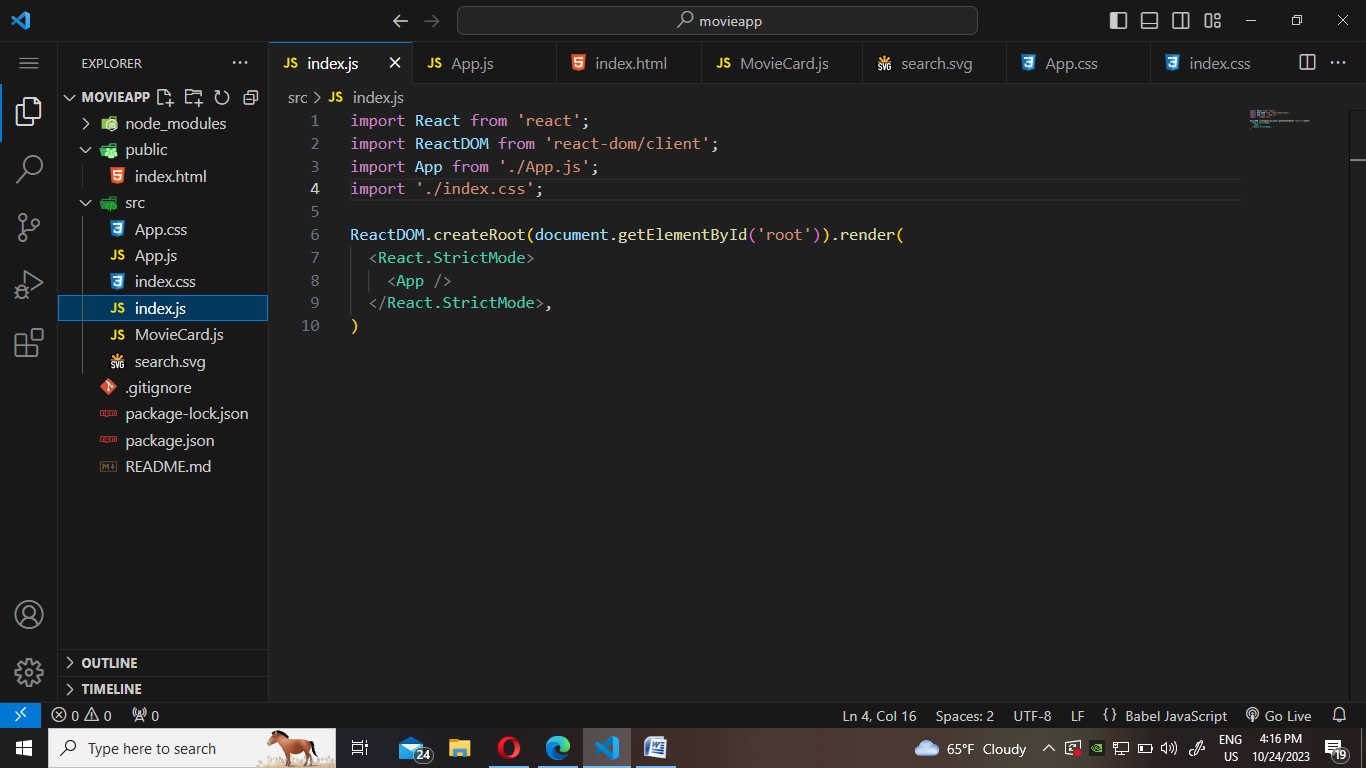






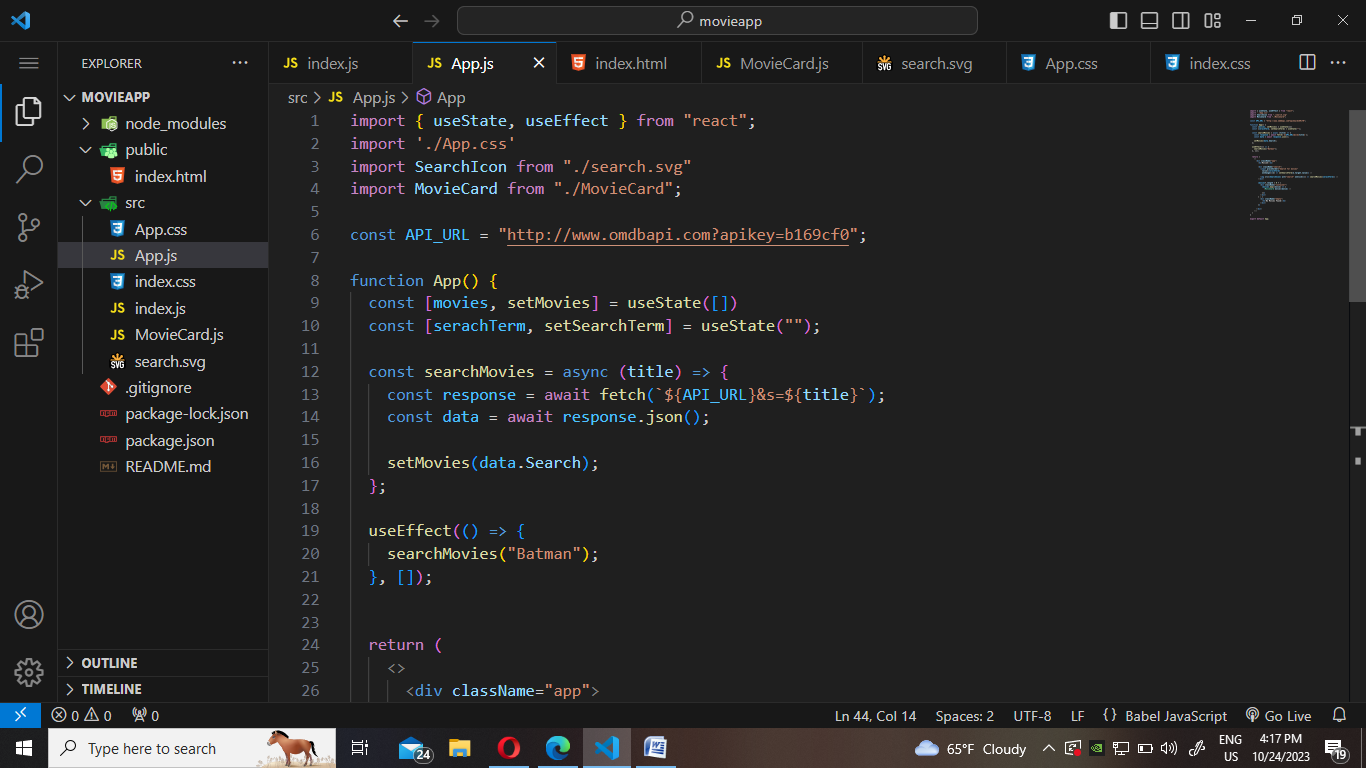


INDEX.JS

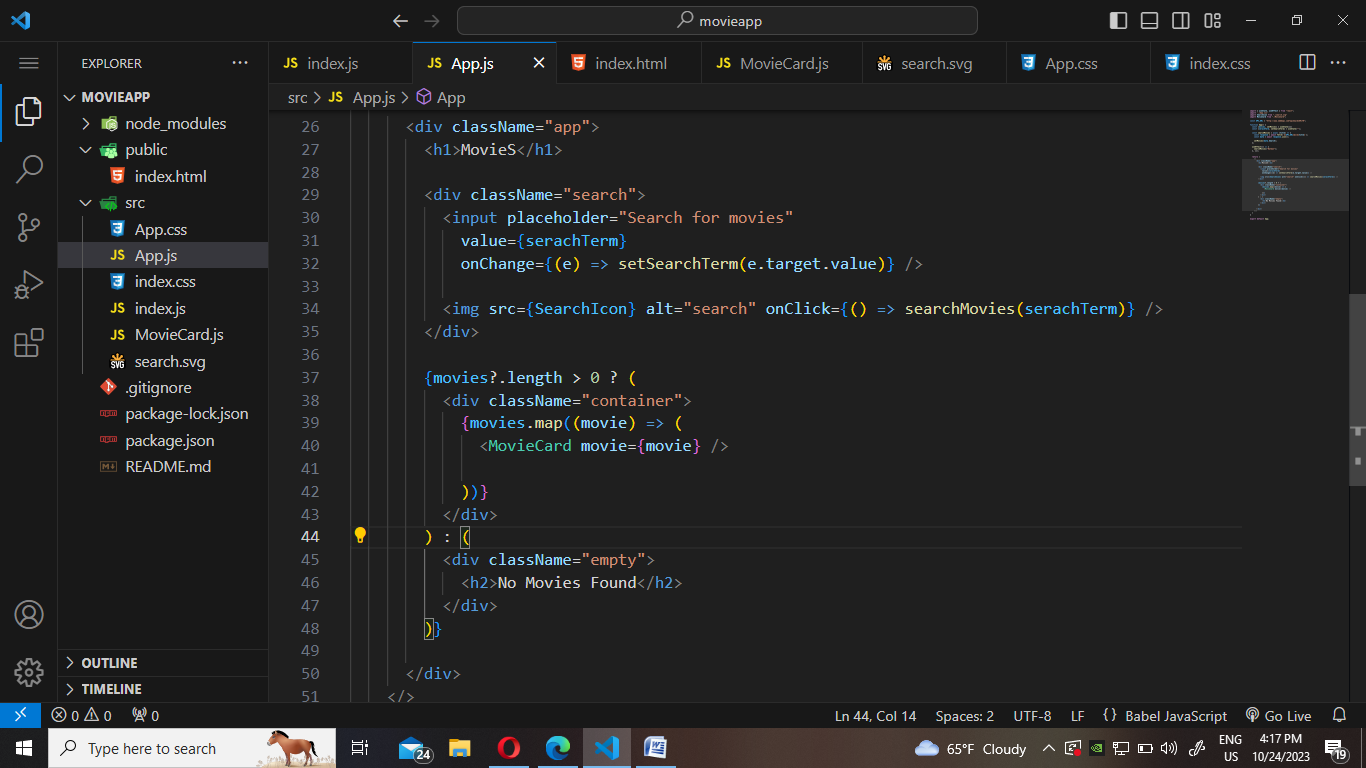


1. **import React from 'react';** and **import ReactDOM from 'react-dom/client';**:
   * These lines import the necessary modules from the React and ReactDOM libraries. React is the core library for building user interfaces, and ReactDOM is used for rendering React components in the web browser.
2. **import App from './App.js';**:
   * This line imports a custom React component named **App** from a file called "App.js." It assumes that there is an **App.js** file in the same directory as the one where this code is located.
3. **ReactDOM.createRoot(document.getElementById('root')).render(...)**:
   * This line of code performs the rendering of the React application. Let's break it down into parts:
   * **ReactDOM.createRoot(document.getElementById('root'))**:
     + **ReactDOM.createRoot()** is a method provided by ReactDOM that initializes the root of your React application. It takes an HTML element as an argument, in this case, **document.getElementById('root')**. It is used to specify where in the HTML document the React application should be rendered. The **getElementById('root')** is expected to refer to a **<div>** element in the HTML where the application will be mounted.
   * **.render(...)**:
     + The **render** method is called on the root created with **createRoot**. It's used to render the actual content of your React application.
   * **<React.StrictMode>** and **<App />**:
     + The content to be rendered is enclosed within a **<React.StrictMode>** element. **<React.StrictMode>** is a wrapper component provided by React that performs a set of checks and warnings to help identify potential issues in your application during development. It doesn't render any visible UI itself but helps catch common problems and encourages best practices.

**APP.JS**

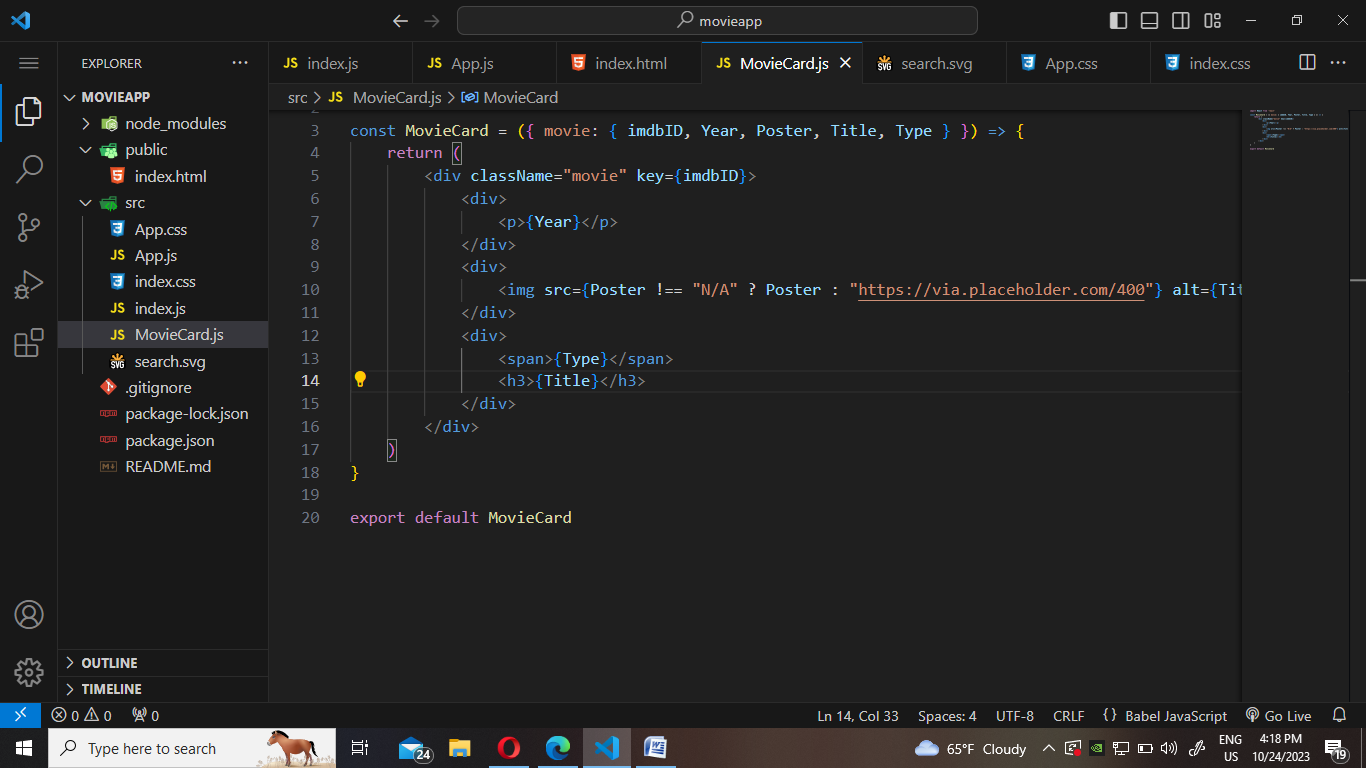
****

1. Imports:
   * **import { useState, useEffect } from "react";**: This line imports the **useState** and **useEffect** hooks from the React library. These hooks are used to manage state and side effects in functional components.
   * **import './App.css';**: This line imports a CSS file named "App.css," presumably for styling the components in the application.
   * **import SearchIcon from "./search.svg";**: This line imports an SVG image named "search.svg." This image is likely used for displaying a search icon in the user interface.
   * **import MovieCard from "./MovieCard";**: This line imports a custom component named **MovieCard** from a file named "MovieCard.js." It's common in React applications to break the UI into reusable components.
2. API
   * **const API\_URL = "http://www.omdbapi.com?apikey=b169cf0";**: This constant defines the base URL for the OMDB API, which is used for fetching movie data. It includes an API key, which is required to access the OMDB API.
3. State Management:
   * **const [movies, setMovies] = useState([]);**: This line uses the **useState** hook to declare a state variable named **movies** and a corresponding function **setMovies** to update the state. Initially, it's set to an empty array, which will eventually hold the list of movies fetched from the OMDB API.
   * **const [searchTerm, setSearchTerm] = useState("");**: Similarly, this line declares a state variable **searchTerm** to hold the user's search query and **setSearchTerm** to update it. Initially, it's an empty string.
4. **searchMovies** Function:
   * **const searchMovies = async (title) => { ... }**: This is an asynchronous function that takes a **title** argument, representing the search query. It is responsible for making a request to the OMDB API to search for movies based on the provided title.
   * **await fetch(**${API\_URL}&s=${title}**);**: This line uses the **fetch** API to send an HTTP GET request to the OMDB API with the provided search query. The **await** keyword is used to wait for the response from the API.
   * **const data = await response.json();**: Once the response is received, it's parsed as JSON using the **json()** method, and the resulting data is stored in the **data** variable.
   * **setMovies(data.Search);**: The **setMovies** function is used to update the **movies** state variable with the movie search results from the API response.
5. **useEffect** Hook:
   * **useEffect(() => { ... }, []);**: The **useEffect** hook is used to introduce side effects into the component. In this case, it's set to run the effect (i.e., the **searchMovies("Batman")** call) only once when the component mounts (as indicated by the empty dependency array **[]**). It initiates the initial search for movies with the query "Batman."

****

1. **<div className="app">**:
   * This **div** element has a **className** of "app," which is likely used for applying specific CSS styles to this component.
2. **<h1>MovieS</h1>**:
   * This is an **<h1>** heading element that displays the text "MovieS."
3. **<div className="search">**:
   * This **div** element has a **className** of "search." It's likely a container for the search input and the search button.
4. **<input placeholder="Search for movies" value={searchTerm} onChange={(e) => setSearchTerm(e.target.value)} />**:
   * This is an **input** element. It has a placeholder text, "Search for movies," to provide a hint to the user. The **value** attribute is bound to the **searchTerm** state variable, which allows the input field to be controlled by the component's state. The **onChange** event handler updates the **searchTerm** state with the user's input as they type.
5. **<img src={SearchIcon} alt="search" onClick={() => searchMovies(searchTerm)} />**:
   * This is an **img** element. It displays an image sourced from the **SearchIcon** import. The **alt** attribute provides alternative text for the image. When the image is clicked (via the **onClick** event), it triggers the **searchMovies** function with the current **searchTerm** as an argument.
6. Conditional Rendering:
   * **{movies?.length > 0 ? (... : ...}**: This is a conditional rendering statement based on the length of the **movies** array.
     + If **movies** is an array with a length greater than 0, it will render the content inside the first set of parentheses. This typically means that movies were found and can be displayed.
     + If there are no movies found (length is 0 or **movies** is **null** or **undefined**), it will render the content inside the second set of parentheses. This displays a message indicating that no movies were found.
7. Movie Card Rendering:
   * **{movies.map((movie) => (<MovieCard movie={movie} />))}**: If movies are found, this part of the code maps over the **movies** array and renders a **MovieCard** component for each movie in the array. The **movie** object is passed as a prop to the **MovieCard** component.
8. Empty State:
   * If no movies are found, the code renders a "No Movies Found" message within a **div** element with a **className** of "empty."
9. The entire JSX structure is enclosed within the **<div className="app">** container.

MOVIECARD.JS



1. **<div className="app">**:
   * This **div** element has a **className** of "app," which is likely used for applying specific CSS styles to this component.
2. **<h1>MovieS</h1>**:
   * This is an **<h1>** heading element that displays the text "MovieS."
3. **<div className="search">**:
   * This **div** element has a **className** of "search." It's likely a container for the search input and the search button.
4. **<input placeholder="Search for movies" value={searchTerm} onChange={(e) => setSearchTerm(e.target.value)} />**:
   * This is an **input** element. It has a placeholder text, "Search for movies," to provide a hint to the user. The **value** attribute is bound to the **searchTerm** state variable, which allows the input field to be controlled by the component's state. The **onChange** event handler updates the **searchTerm** state with the user's input as they type.
5. **<img src={SearchIcon} alt="search" onClick={() => searchMovies(searchTerm)} />**:
   * This is an **img** element. It displays an image sourced from the **SearchIcon** import. The **alt** attribute provides alternative text for the image. When the image is clicked (via the **onClick** event), it triggers the **searchMovies** function with the current **searchTerm** as an argument.
6. Conditional Rendering:
   * **{movies?.length > 0 ? (... : ...}**: This is a conditional rendering statement based on the length of the **movies** array.
     + If **movies** is an array with a length greater than 0, it will render the content inside the first set of parentheses. This typically means that movies were found and can be displayed.
     + If there are no movies found (length is 0 or **movies** is **null** or **undefined**), it will render the content inside the second set of parentheses. This displays a message indicating that no movies were found.
7. Movie Card Rendering:
   * **{movies.map((movie) => (<MovieCard movie={movie} />))}**: If movies are found, this part of the code maps over the **movies** array and renders a **MovieCard** component for each movie in the array. The **movie** object is passed as a prop to the **MovieCard** component.
8. Empty State:
   * If no movies are found, the code renders a "No Movies Found" message within a **div** element with a **className** of "empty."
9. The entire JSX structure is enclosed within the **<div className="app">** container.