# MIT WORLD PEACE UNIVERSITY

Full Stack Development
Third Year B.Tech, Semester 1

Landing Page using React

## ASSIGNMENT 5

Prepared By PA24
Saubhagya Singh
Batch A1

**Aim:**

Design and develop an interactive user interface using React.

**Objectives:**

1. Articulate what React is and why it is useful.
2. Explore the basic architecture of a React application.
3. Use React components to build interactive interfaces

**Theory:**

**1] What is React? Steps to run React app using create-react-app.**
=>
React is an open-source JavaScript library developed by Facebook for building user interfaces or UI components, particularly for single-page applications (SPAs). It allows developers to create interactive and reusable UI components, making the development of complex UIs more manageable. React uses a component-based architecture where UIs are composed of small, reusable components, making it easier to maintain and update.

Steps to Run a React App Using create-react-app:
To start a React project using create-react-app, you need to have Node.js and npm (Node Package Manager) installed. Follow these steps:

- Install Node.js and npm: If you haven't already, download and install Node.js from the official website. npm is included with Node.js.
- Create a React App: Open your terminal or command prompt and run the following command to create a new React app named my-react-app using create-react-app.Replace my-react-app with the name you want for your React application.

  *npx create-react-app my-react-app*
- Navigate to the App Directory: Once the app is created, navigate to the app's directory using the command:

  *cd my-react-app*

- Start the Development Server: Run the following command to start the development server:

*npm start*

This command starts the development server and opens the React app in your default web browser. If it doesn't open automatically, you can access it by going to http://localhost:3000.

- Explore and Edit: You can now explore and edit your React app. The files and folders are structured, and the main file to start editing is src/App.js.
- Make Changes: Open the project in your preferred code editor and make changes to the files in the src directory. As you make changes, the development server automatically reloads the app in the browser, allowing you to see the changes in real-time.
- Stop the Development Server: To stop the development server, go to the terminal where the server is running and press Ctrl + C. Confirm by typing Y or yes.

**2] Passing data through props (Small example)**

=>

In React, passing data between components is commonly done using props (short for properties). Here's a small example demonstrating how to pass data from a parent component to a child component via props: Let's create a simple React application with a parent component (ParentComponent) passing data to a child component (ChildComponent):

```
import React from 'react';

const ChildComponent = (props) => {
  return (
    <div>
      <p>This is the Child Component</p>
      <p>Received data: {props.message}</p>
    </div>
  );
};

export default ChildComponent;
```

In the ChildComponent, we receive message as a prop and display it.

```
// ParentComponent.js
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  const messageData = "Hello from Parent!";

  return (
    <div>
      <h1>Parent Component</h1>
      <ChildComponent message={messageData} />
    </div>
  );
};

export default ParentComponent;
```

In the ParentComponent, we create a variable messageData containing the message we want to pass. Then, we render the ChildComponent, passing messageData as a prop named message.
Finally, to see this in action, let's render the ParentComponent in the main index.js file:

```
// index.js
import React from 'react';
import ReactDOM from 'react-dom';
import ParentComponent from './ParentComponent';

ReactDOM.render(
  <React.StrictMode>
    <ParentComponent />
  </React.StrictMode>,
  document.getElementById('root')
);
```

This code sets up the main entry point of our React application and renders the ParentComponent.

When you run this code, the ChildComponent will receive the message prop from the ParentComponent and display the message "Hello from Parent!" as part of its content. This demonstrates how data can be passed from a parent component to a child component using props in React.

**FAQ :**

**1] What are React states and hooks?**

=>

In React, states and hooks are essential concepts used for managing component-level data and implementing stateful logic within functional components.

- React States:

States in React are used to manage and handle component-specific data that can change over time. States enable React components to update and re-render based on changes in data, user interactions, or other events.

Class Components: Traditionally, states were primarily used in class-based components through the setState() method and the this.state object.

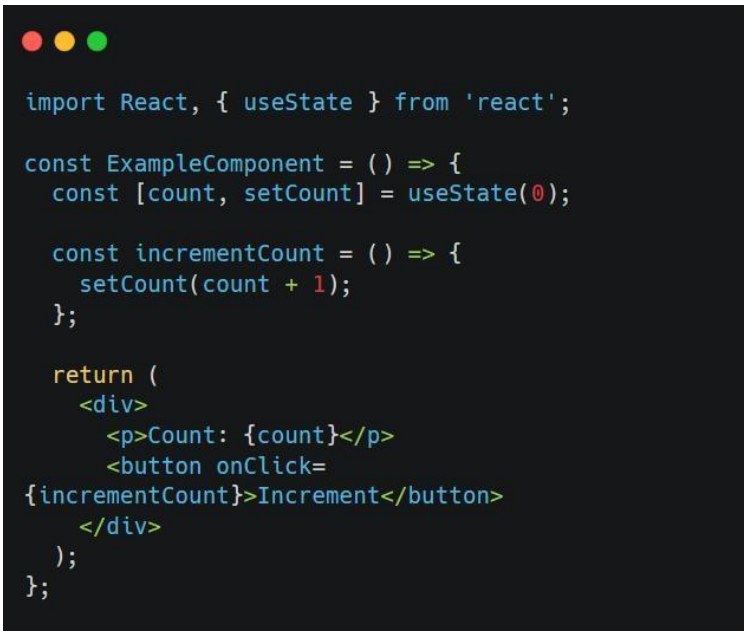
Functional Components (with Hooks): With the introduction of Hooks in React 16.8, functional components gained the capability to use state and other React features. The useState hook is commonly used in functional components to add state management.

- React Hooks:

Hooks in React are functions that allow functional components to use state, lifecycle methods, context, and other React features without needing to write a class. They enable the reuse of stateful logic between components.

useState Hook: Allows functional components to manage local state. It returns a stateful value and a function to update that value, similar to this.state and this.setState in class components.

Example of useState hook:



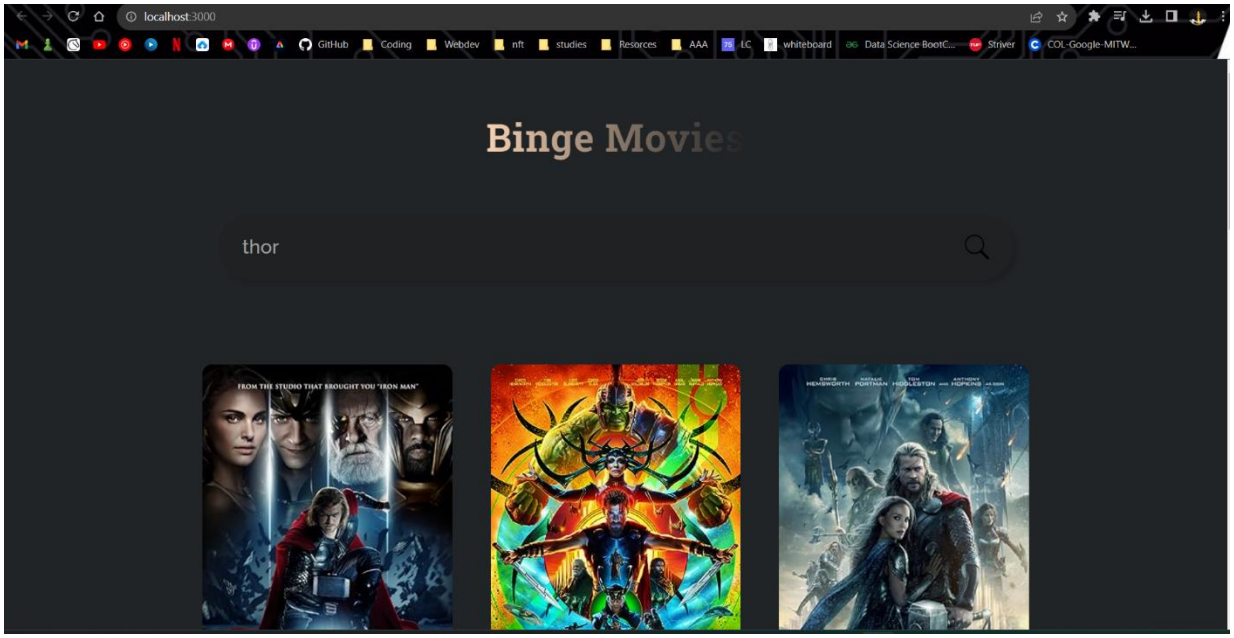
```
import React, { useState } from 'react';

const ExampleComponent = () => {
  const [count, setCount] = useState(0);

  const incrementCount = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick=
{incrementCount}>Increment</button>
    </div>
  );
};
```

- Other Hooks:

React provides various other hooks such as useEffect, useContext, useReducer, and more. These hooks empower functional components with capabilities previously exclusive to class components.
Hooks allow functional components to manage their own state and have lifecycle behaviors, enabling developers to write more concise, reusable, and readable code.

**Output & SRC :**
Landing Page Hosted at:
Movie-Index.surge.sh

GitHub Repo :
https://github.com/SaubhagyaSingh/BingeMovies




**Result :**

Created an Interactive Landing page using React and Bootstrap.