

| Title: Implementation of React Fundamentals. |
| --- |

**AIM:** To Implement the Basic operations of React js

**Problem Definition:**

-Demonstrate the

* React Fundamentals
* Function Component
* Styling/ Bootstrap
* React JSX
* Expressions in JSX
* React Props
* React state
* React Component Lifecycle
* React Events
* Event Binding

\*(Students have to perform the task assigned within group and demonstrate the same).

**Resources used:**

<https://react.dev/learn>

<https://www.freecodecamp.org/>

<https://nodejs.org/docs/latest/api/documentation.html>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO 1:**.Build full stack applications in JavaScript using the MERN technologies.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Shelly Powers Learning Node O’ Reilly 2 nd Edition, 2016.

**Pre Lab/ Prior Concepts:**

React JS:

React JS is a JavaScript library developed by Facebook for building user interfaces, especially single-page applications where data changes over time. It allows developers to create large web applications that can update and render efficiently with small data changes.

Setup of First Application using React js Steps:

 **Install Node.js and npm**: Download and install Node.js, which includes npm, the package manager required for React.

 **Create a React App**: Use the command npx create-react-app my-app in the terminal to generate a new React project named "my-app."

 **Run the App**: Navigate to the project directory with cd my-app and start the development server using npm start.

Importance of all Files considering the Folder Structure of First React Application:

 **public/index.html**: The single HTML file that serves as the entry point for the web app; React components are rendered inside this file.

 **src/index.js**: The main JavaScript file where the React app starts; it renders the root component to the DOM.

 **src/App.js**: The root component of the React app where you build the main UI.

Component in React:

Components are the building blocks of a React application. They are reusable pieces of code that define how a certain part of the UI should appear and behave, making the app modular and maintainable.

React JSX:

JSX (JavaScript XML) is a syntax extension for JavaScript used in React. It allows developers to write HTML-like code within JavaScript, which React then transforms into JavaScript objects, making it easier to write and visualize the structure of the UI. JSX allows embedding JavaScript expressions within HTML tags, making it powerful and flexible for dynamic content rendering. It enhances code readability and helps in creating more intuitive and maintainable component structures.

**Methodology:**

Component-Based Architecture: React follows a component-based architecture where the UI is divided into small, reusable components. This modular approach makes the development process more efficient and the application easier to manage.

Virtual DOM: React uses a virtual DOM to optimize updates and rendering. When the state of an object changes, React updates the virtual DOM first and then compares it with the actual DOM. This process, called reconciliation, ensures that only the changed elements are updated in the real DOM, improving performance.

Unidirectional Data Flow: React follows a unidirectional data flow, meaning data flows in one direction from parent to child components. This makes the application easier to debug and understand, as data is always passed down and not arbitrarily changed by child components.

**Implementation Details:**

React Fundamentals: creating an app in react using node

npx create-react-app first-app

Function component and styling using bootstrap

import React from 'react';

const Greeting = () => {

  return <h1 className="text-primary">Hello, React!</h1>;

};

export default Greeting;

Using React JSX and expressions

import React from 'react';

const Welcome = ({ name }) => {

  return <h2>Welcome, {name}!</h2>;

};

export default Welcome;

Demonstrating React Props

import React from 'react';

import Greeting from './components/Greeting';

import Welcome from './components/Welcome';

const App = () => {

  return (

    <div>

      <Greeting />

      <Welcome name="Rohit Deshpande" />

    </div>

  );

};

export default App;

Managing React state

import React, { useState } from 'react';

const Counter = () => {

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

  return (

    <div>

      <p>Count: {count}</p>

      <button onClick={() => setCount(count + 1)}>Increment</button>

    </div>

  );

};

export default Counter;

React Component Lifecycle

import React, { Component } from 'react';

class LifecycleDemo extends Component {

  componentDidMount() {

    console.log('Component did mount');

  }

  componentDidUpdate() {

    console.log('Component did update');

  }

  componentWillUnmount() {

    console.log('Component will unmount');

  }

  render() {

    return <div>Lifecycle Demo</div>;

  }

}

export default LifecycleDemo;

Handling React events and event binding

import React from 'react';

const ClickEvent = () => {

  const handleClick = () => {

    alert('Button clicked!');

  };

  return <button onClick={handleClick}>Click Me</button>;

};

export default ClickEvent;

**Steps for execution:**

In App.js

import React from 'react';

import Greeting from './components/Greeting';

import Welcome from './components/Welcome';

import Counter from './components/Counter';

import LifecycleDemo from './components/LifecycleDemo';

import ClickEvent from './components/ClickEvent';

const App = () => {

  return (

    <div className="container">

      <Greeting />

      <Welcome name="John Doe" />

      <Counter />

      <LifecycleDemo />

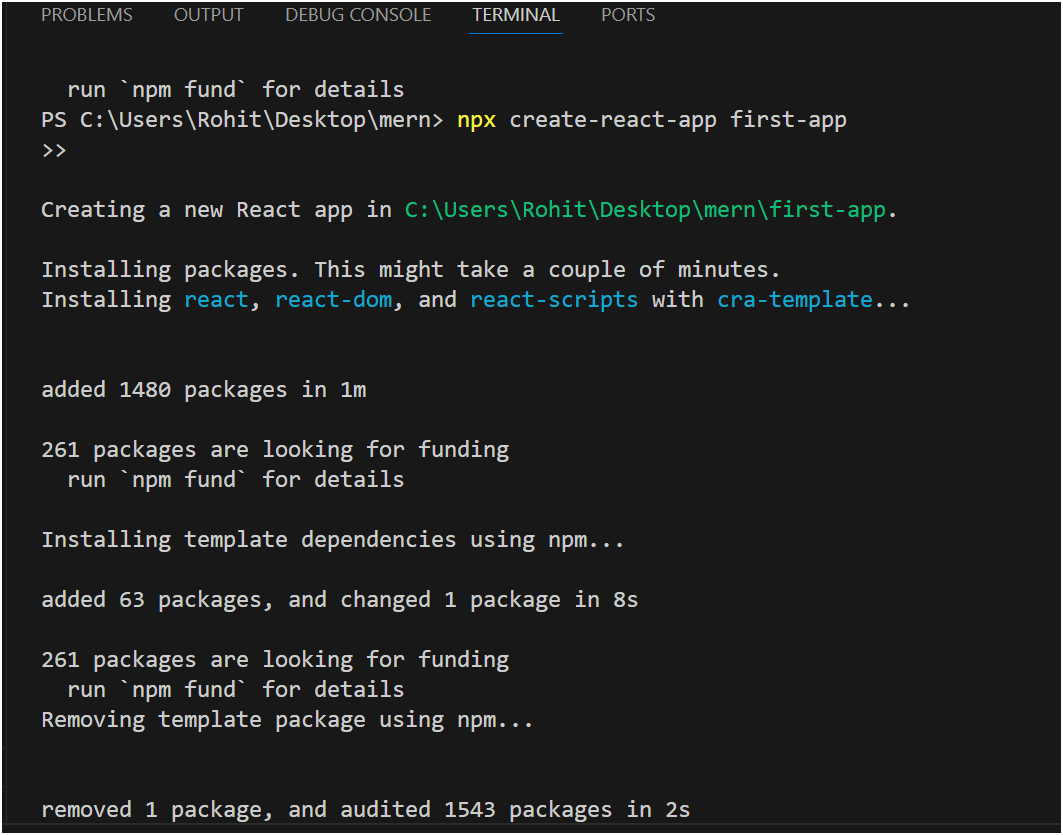
      <ClickEvent />

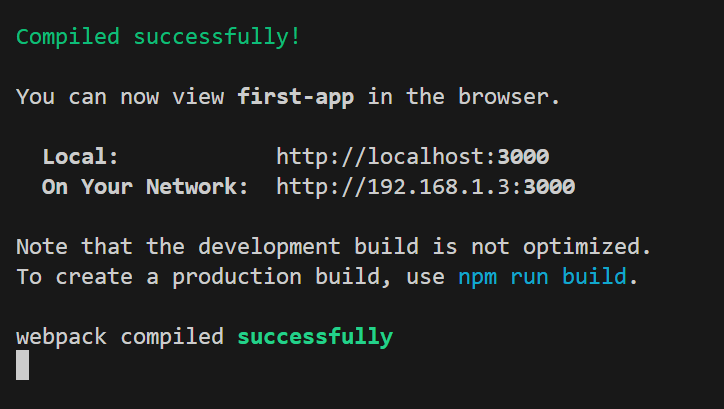
    </div>

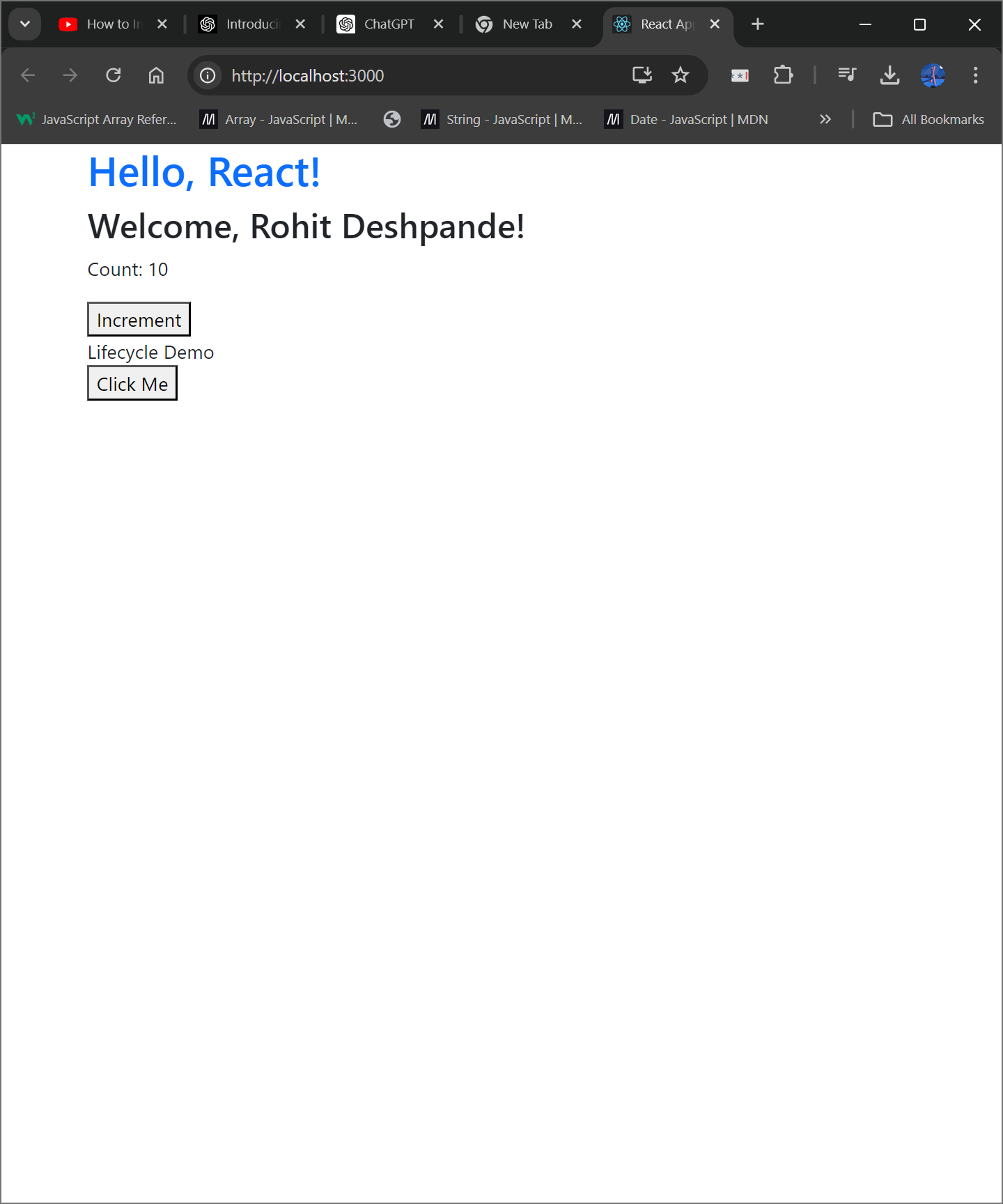
  );

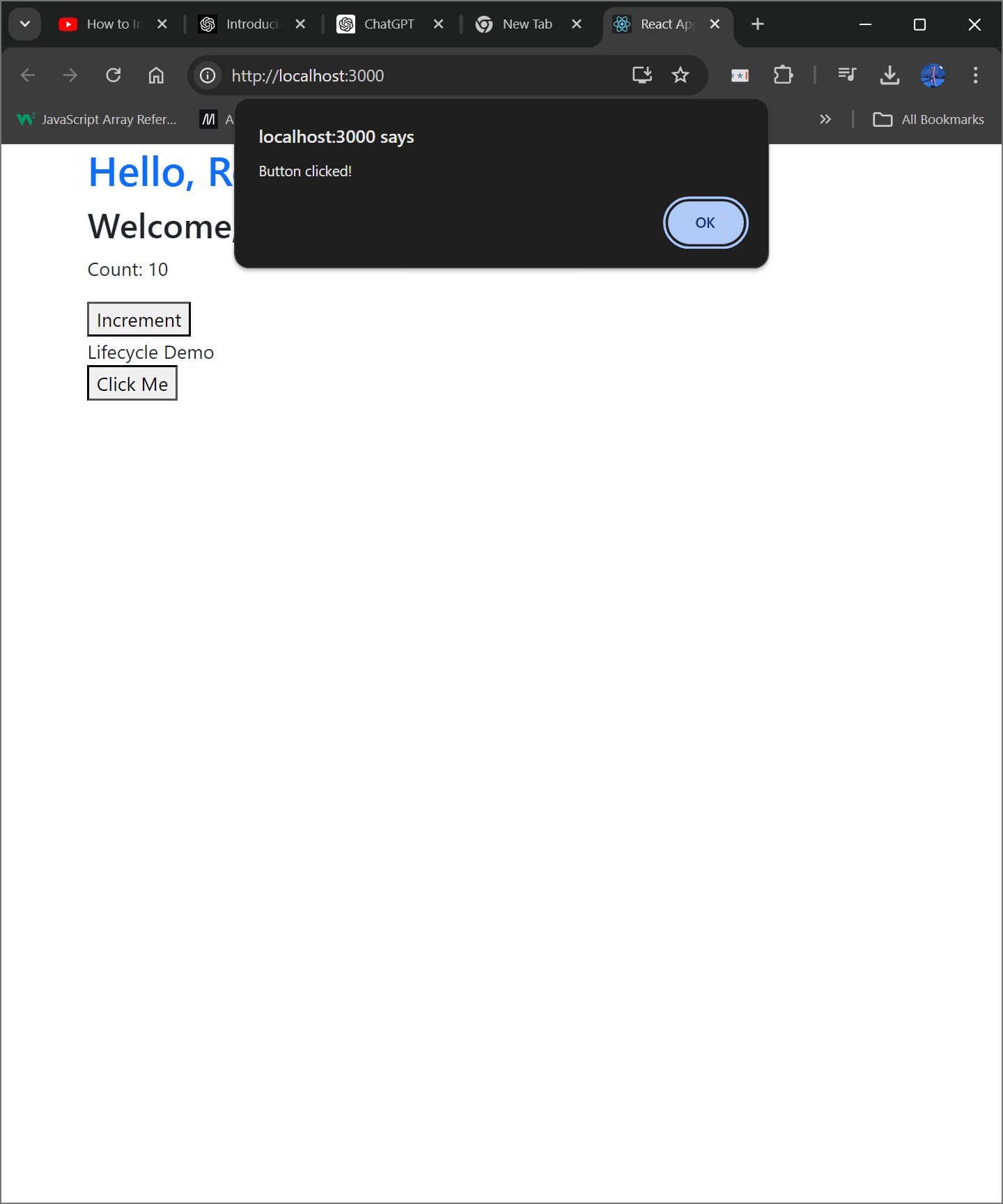
};

export default App;









**Conclusion:** We successfully demonstrated key React.js concepts including function components, JSX, props, state, component lifecycle, event handling, and styling with Bootstrap.

**Postlab questions:**

1. **Explain the Concept of SPA.**

A **Single Page Application (SPA)** is a type of web application that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from the server. This approach provides a more seamless and faster user experience. Key characteristics of SPAs include:

* **Dynamic Content Loading:** Only the necessary content is fetched and updated on the page, reducing the need for full page reloads.
* **Enhanced User Experience:** Users experience a more responsive and fluid interface, similar to a desktop application.
* **Efficient Resource Usage:** SPAs minimize server load and bandwidth usage since only partial updates are made.

1. **What is Component Lifecycle?**

he **Component Lifecycle** in React refers to the sequence of events (or phases) that a component goes through during its existence, from creation to destruction. These phases allow developers to hook into specific moments and execute code at those times. The lifecycle is typically divided into three main phases:

1. **Mounting:** When a component is being inserted into the DOM.
   * **constructor():** Called before the component is mounted.
   * **render():** Returns the JSX to be rendered.
   * **componentDidMount():** Invoked after the component is mounted. Ideal for initiating network requests, setting up subscriptions, or any other side effects.
2. **Updating:** When a component is being re-rendered as a result of changes to its props or state.
   * **componentDidUpdate(prevProps, prevState):** Called after the component's updates are flushed to the DOM. Useful for operating on the DOM after updates or making additional network requests based on changes.
3. **Unmounting:** When a component is being removed from the DOM.
   * **componentWillUnmount():** Invoked immediately before a component is unmounted and destroyed. Useful for cleanup tasks like removing event listeners, canceling network requests, or clearing timers.

By understanding and utilizing these lifecycle methods, developers can manage component behavior and side effects more effectively in a React application.