**Batch: C1 Roll No.: 16010122221**

**Experiment / assignment / tutorial No.1**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| 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:**

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

**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 is a JavaScript library for building user interfaces. It allows developers to create reusable UI components and manage the state of web applications efficiently.

Setup of First Application using React js Steps:

* Install Node.js and npm
* Open terminal and run: npx create-react-app my-react-app
* Navigate to the project directory: cd my-react-app
* Start the development server: npm start

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

* public/index.html: The main HTML file where React will render the app
* src/index.js: The entry point of the React application
* src/App.js: The main component of the application
* package.json: Contains project dependencies and scripts

Component in React:

* A component is a reusable piece of UI. Components can be functional (using hooks) or class-based.

React JSX:

* JSX is a syntax extension for JavaScript that allows you to write HTML-like code in your JavaScript files.

**Methodology:**

* Create a new React application using Create React App
* Implement functional components to demonstrate various React concepts
* Use JSX to create the component structure
* Implement state management using useState hook
* Demonstrate props passing between components
* Implement event handling and binding
* Use useEffect hook to demonstrate component lifecycle
* Apply basic styling using CSS and Bootstrap

**Implementation Details:**

**Function Component:**

**Concept**: A function component is a plain JavaScript function that returns a React element. It's a simple way to create a component in React.

**Methodology**:

* Use JavaScript functions to define components.
* The function's name should start with a capital letter.
* It can accept props as an argument and return JSX.

**Implementation**:

function MyComponent(props) {

return <h1>Hello, {props.name}!</h1>;

}

**Styling/Bootstrap:**

**Concept**: Styling in React can be done using various methods such as inline styles, CSS files, or by using frameworks like Bootstrap for pre-defined styles.

**Methodology**:

* **Inline Styles**: Use the style attribute with a JavaScript object.
* **CSS**: Import a CSS file and apply classes.
* **Bootstrap**: Use Bootstrap classes by importing the Bootstrap CSS.

**Implementation**:

import 'bootstrap/dist/css/bootstrap.min.css';

function MyComponent() {

return <button className="btn btn-primary">Click Me</button>;

}

**React JSX:**

**Concept**: JSX is a syntax extension for JavaScript that looks like HTML and allows you to write HTML elements in JavaScript. It's transpiled to React.createElement() calls.

**Methodology**:

* JSX allows embedding JavaScript expressions using {}.
* JSX must return a single parent element.

**Implementation**:

function MyComponent() {

return (

<div>

<h1>Hello, world!</h1>

<p>This is a React component.</p>

</div>

);

}

**Expressions in JSX:**

**Concept**: JSX allows you to embed any JavaScript expression within curly braces {}.

**Methodology**:

* Use {} to embed variables, function calls, or any valid JavaScript expression.

**Implementation**:

function MyComponent() {

const name = "John";

return <h1>Hello, {name}!</h1>;

}

**React Props:**

**Concept**: Props (short for properties) are used to pass data from one component to another, typically from a parent to a child component.

**Methodology**:

* Props are passed to components as attributes.
* Components receive props as an object argument.

**Implementation**:

function MyComponent(props) {

return <h1>Hello, {props.name}!</h1>;

}

function ParentComponent() {

return <MyComponent name="John" />;

}

**React State:**

**Concept**: State is a built-in object that holds data that may change over the lifecycle of the component.

**Methodology**:

* Use the useState hook to manage state in functional components.
* State updates cause the component to re-render.

**Implementation**:

import { useState } from 'react';

function MyComponent() {

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

return (

<div>

<p>Count: {count}</p>

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

</div>

);

}

**React Component Lifecycle:**

**Concept**: React components have a lifecycle, which includes stages like mounting, updating, and unmounting. Lifecycle methods/hooks allow you to execute code at different stages.

**Methodology**:

* Use lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount in class components.
* Use the useEffect hook in functional components for similar behavior.

**Implementation** (Functional Component with useEffect):

import { useEffect } from 'react';

function MyComponent() {

useEffect(() => {

// Code to run on mount

return () => {

// Code to run on unmount

};

}, []); // Empty dependency array ensures it runs only on mount/unmount

return <div>Component Content</div>;

}

**React Events:**

**Concept**: React events are similar to DOM events but are named differently, e.g., onClick, onChange. They handle user interactions like clicks, typing, etc.

**Methodology**:

* Events in React are camelCase.
* Event handlers can be passed as functions.

**Implementation**:

function MyComponent() {

const handleClick = () => {

alert('Button clicked!');

};

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

}

**React Binding:**

**Concept**: Event binding ensures that event handlers are bound to the correct context (this keyword) within a component.

**Methodology**:

* In class components, you may need to bind event handlers in the constructor using bind().
* Arrow functions automatically bind this context in class components.

**Implementation** (Class Component):

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.handleClick = this.handleClick.bind(this);

}

handleClick() {

alert('Button clicked!');

}

render() {

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

}

}

**Setup of First Application using React js Steps:**

**Install Node.js and npm**:

* Ensure Node.js and npm (Node Package Manager) are installed on your system.
* Verify installation by running:

node -v

npm -v

**Create a New React Application**:

* Use create-react-app to set up a new React project.
* Run the following command in your terminal:

npx create-react-app my-first-app

Replace my-first-app with your desired project name.

**Navigate to the Project Directory**:

* Move into the newly created project folder:

cd my-first-app

**Start the Development Server**:

* Launch the application in your default web browser:

npm start

* The app will run at http://localhost:3000 by default.

**Modify the Application**:

* Open the project in your preferred code editor (e.g., VS Code).
* Modify the src/App.js file to change the content and see the live updates in your browser.

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

* 1. **node\_modules/**:
* Contains all the dependencies and packages installed via npm.
* This folder is automatically generated and managed by npm based on the package.json file.
  1. **public/**:
* **index.html**:
  + The single HTML file used by the React application. React components are injected into the div with the id of root.
  + Important for meta tags, SEO configurations, and setting up a favicon.
* **manifest.json**:
  + Configures the web app’s manifest, allowing you to control how your app appears when installed on a device (e.g., app icons, startup screen).
* **favicon.ico**:
  + The icon displayed in the browser tab for your application.
* **robots.txt**:
  + Used to instruct web crawlers which URLs they can access on your site.
  1. **src/**:
* **index.js**:
  + The entry point for the React application. It renders the root component (App) into the DOM element in index.html.
  + Responsible for importing and applying global styles and handling the rendering of the app.
* **App.js**:
  + The main component of the application. It’s the first component rendered and typically serves as the root component from which other components are rendered.
* **App.css**:
  + Contains the styles for the App component. Modify this file to style your application.
* **index.css**:
  + Global styles for the entire application. These styles are applied across all components.
* **App.test.js**:
  + A default test file for the App component. Used for writing unit tests with Jest.
* **logo.svg**:
  + A sample SVG logo that is used in the App.js file by default.
* **serviceWorker.js**:
  + Optional service worker script that helps with caching and making the app work offline. (No longer included by default in newer versions of create-react-app).
* **setupTests.js**:
  + Configures the testing environment using Jest and other testing libraries.
  1. **.gitignore**:
* Lists the files and directories that should not be tracked by Git. This typically includes node\_modules/ and environment variables.
  1. **package.json**:
* Contains metadata about the project, including scripts, dependencies, and version information.
* Key for managing project dependencies and scripts (e.g., start, build, test).
  1. **package-lock.json**:
* Automatically generated file that records the exact versions of dependencies installed in node\_modules.
* Ensures consistent installs across different environments.
  1. **README.md**:
* A markdown file containing information about the project. This file typically provides setup instructions, usage guidelines, and additional resources.

**Steps for execution:**

* Open a terminal and navigate to the project directory
* Run npm start to start the development server
* Open a web browser and go to http://localhost:3000
* Interact with the application to see the counter increment and the child component update

**Conclusion:**

In this experiment, we successfully implemented and demonstrated key React fundamentals.

**Postlab questions:**

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

A Single Page Application (SPA) is a web application that loads a single HTML page and dynamically updates the content as the user interacts with the app. Instead of loading entire new pages from the server, SPAs use AJAX and HTML5 to create fluid and responsive web apps. React is often used to build SPAs due to its efficient rendering and state management capabilities.

1. **What is Component Lifecycle?**

The component lifecycle refers to the series of stages a component goes through from its creation to its removal from the DOM. In functional components with hooks, the lifecycle is managed primarily through the useEffect hook. It allows you to perform side effects at different stages, such as when the component mounts, updates, or unmounts.