**REACT**

**NAME: SOMYA RANJAN SAHU**

**SUPERSET ID: 6363357**

**HANDS-ON: 1**

## **Objectives**

* Define SPA and its benefits

A Single Page Application (SPA) is a web application architecture that operates within a single web page. The initial request loads the entire application shell—a single HTML file along with the necessary CSS and JavaScript. Subsequent user interactions trigger dynamic content updates through JavaScript APIs, which fetch data from the server and rewrite the current page in place. This model significantly enhances performance and responsiveness by eliminating the need for full page reloads, thereby reducing server round-trips and data transfer.

* Define React and identify its working

React is a declarative JavaScript library developed by Facebook for crafting interactive user interfaces. It employs a component-based architecture, enabling developers to build encapsulated, stateful modules that can be composed to form complex UIs. React's core performance advantage comes from its use of a Virtual DOM. It performs a "diffing" algorithm to compare changes between the current and previous states of this virtual representation, then executes minimal and batched updates to the actual DOM, ensuring high efficiency.

* Identify the differences between SPA and MPA

A **Single Page Application (SPA)** loads a single HTML shell and uses JavaScript to manage all subsequent views and user interactions. This minimizes interruptions by fetching data in the background and updating only specific parts of the page. While this leads to a superior user experience, the initial load can be heavier.

* Explain Pros & Cons of Single-Page Application

While Single Page Applications (SPAs) offer a significantly smoother and faster user experience with quicker navigation and less load on the server, they come with notable trade-offs. The primary challenges include a slower initial page load and increased development complexity due to the need for client-side routing and state management. Furthermore, SPAs can be difficult for search engines to crawl, potentially harming SEO unless techniques like Server-Side Rendering (SSR) are implemented.

* Define virtual DOM

React's declarative approach to UI development is facilitated by its component-based architecture, unidirectional data flow, and an optimized reconciliation process via the Virtual DOM. The Virtual DOM acts as an in-memory abstraction of the real DOM. When a component's state is altered, React generates a new virtual tree. It then performs a "diffing" algorithm to compare this new tree with the previous one, ultimately computing the most economical batch of operations to update the actual DOM, thereby enhancing performance

* Explain about React

React makes UI development easy by using reusable components, unidirectional data flow, and effective DOM updates via the virtual DOM. React is a declarative, lightweight, and versatile JavaScript library created by Facebook that can be used to develop user interfaces. It enables developers to create encapsulated components with their own state, and compose them together to create complex UIs

* Explain Features of React

React's advantages are component-based structure, virtual DOM for performance, JSX (JavaScript XML) for template writing, unidirectional data flow, and rigorous ecosystem support for routing, state management, and server-side rendering.

**Question**: Create a new React Application with the name “myfirstreact”, Run the application to print “welcome to the first session of React” as heading of that page.

**Code:**

**App.js:**

function App() {

  return (

    <div>

      <h1>Welcome to the first session of React</h1>

    </div>

  );

}

export default App;

**OUPUT:**  
A screenshot of a computer

AI-generated content may be incorrect.

**HANDS-ON: 2**

**Objectives**

* Explain React components

In React, components are building units of any React application. They are pieces of the UI and enable developers to divide the interface into separate, reusable units that can be maintained and worked on more effectively. React components can be authored as JavaScript classes or functions and render elements that define what to render on screen.

* Identify the differences between components and JavaScript functions

Though React components and JavaScript functions are defined in the same way, the primary distinction is purpose and behavior. Components are JSX-returning and handle state or lifecycle methods (in class components), while plain JavaScript functions are for logic and calculation. React components are specifically meant to render UI and handle user interaction.

* Identify the types of components

There are two kinds of components in React: class components and function components . Class components are ES6 classes that inherit from `React.Component` and employ methods such as `render()` and lifecycle methods. Function components are more lightweight, expressed as JavaScript functions, and tend to be used with React Hooks (such as `useState` and `useEffect`) to handle state and side effects.

* Explain class component

A class component should have a `render()` method, returning the JSX to be rendered. Class components can have their own local state and can handle lifecycle events and are well-suited for complicated UI logic. A function component is a lightweight form of writing components as functions. Function components were initially stateless, but since React 16.8 introduced hooks, function components can use state and side effects now too.

* Explain function component

In React, functional components are JavaScript functions that accept props as input and return a React element (JSX) to describe the UI. They are often favored for their simplicity and readability, especially for components that don't require managing their own state or using lifecycle methods. While traditionally considered "stateless," the introduction of React Hooks like useState and useEffect allows functional components to manage state and side effects, blurring the lines between functional and class components

* Define component constructor

The constructor for a component in class components is a unique method for initializing state and binding functions. It is invoked once when the component is initialized. Within the constructor, `super(props)` must be called to be able to use `this.props` within the constructor.

* Define render() function

The `render()` method is an obligatory method in class components and serves to return the JSX that defines how the UI is supposed to look. It automatically gets invoked when the component's state or the props change, and React has decided whether it should re-render the component or not.

**Question:**

Create a react app for Student Management Portal named StudentApp and create a component named Home which will display the Message “Welcome to the Home page of Student Management Portal”. Create another component named About and display the Message “Welcome to the About page of the Student Management Portal”. Create a third component named Contact and display the Message “Welcome to the Contact page of the Student Management Portal”. Call all the three components.

**CODE:**

**App.js:**

import React from 'react';

import Home from './Components/Home';

import About from './Components/About';

import Contact from './Components/Contact';

import './App.css';

function App() {

  return (

    <div className="App">

      <Home />

      <About />

      <Contact />

    </div>

  );

}

export default App;

**App.css:**

.App {

  text-align: center;

}

.App-logo {

  height: 40vmin;

  pointer-events: none;

}

@media (prefers-reduced-motion: no-preference) {

  .App-logo {

    animation: App-logo-spin infinite 20s linear;

  }

}

.App-header {

  background-color: #282c34;

  min-height: 100vh;

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  font-size: calc(10px + 2vmin);

  color: white;

}

.App-link {

  color: #61dafb;

}

@keyframes App-logo-spin {

  from {

    transform: rotate(0deg);

  }

  to {

    transform: rotate(360deg);

  }

}

.App {

  text-align: center;

  font-family: Arial, sans-serif;

  margin-top: 50px;

}

h1 {

  margin-bottom: 20px;

  color: #333;

}

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**HANDS-ON: 3**

**Objectives**

* Explain React components

React components are fundamental building blocks of a React application. They determine what to render on the UI and encapsulate structure, logic, and behavior. Components provide modular and maintainable code through the separation of the UI into manageable parts. While React components and JavaScript functions share function syntax, React components are distinctive functions that produce JSX and may handle states and side effects, whereas common functions carry out computations and logic.

* Identify the differences between components and JavaScript functions

Though React components and JavaScript functions are defined in the same way, the primary distinction is purpose and behavior. Components are JSX-returning and handle state or lifecycle methods (in class components), while plain JavaScript functions are for logic and calculation. React components are specifically meant to render UI and handle user interaction.

* Identify the types of components

The two primary kinds of components are class components and function components . Class components utilize ES6 class syntax, keep internal state, and offer lifecycle methods, while function components are more concise and simpler. Function components can also handle state and effects using hooks, so they are rapidly becoming the preferred kind of component.

* Explain class component

Class components use a constructor function to set state and bind functions. A constructor is called when the component is instantiated. The `render()` function` in a class component also renders the JSX structure and is called automatically on state or props changes.

* Explain function component

In React, functional components are JavaScript functions that accept props as input and return a React element (JSX) to describe the UI. They are often favored for their simplicity and readability, especially for components that don't require managing their own state or using lifecycle methods. While traditionally considered "stateless," the introduction of React Hooks like useState and useEffect allows functional components to manage state and side effects, blurring the lines between functional and class components

* Define component constructor

The constructor for a component in class components is a unique method for initializing state and binding functions. It is invoked once when the component is initialized. Within the constructor, `super(props)` must be called to be able to use `this.props` within the constructor.

* Define render() function

The `render()` method is an obligatory method in class components and serves to return the JSX that defines how the UI is supposed to look. It automatically gets invoked when the component's state or the props change, and React has decided whether it should re-render the component or not.

**QUESTION:**

Create a react app for Student Management Portal named scorecalculatorapp and create a function component named “CalculateScore” which will accept Name, School, Total and goal in order to calculate the average score of a student and display the same.

**CODE:**

**./src/Components/Calculator.js:**

import '../Stylesheets/mystyle.css';

const percentToDecimal = (decimal) => {

  return (decimal \* 100).toFixed(2) + '%';

};

const calcScore = (total, goal) => {

  return percentToDecimal(total / goal);

};

export const CalculateScore = ({ Name, School, total, goal }) => (

  <div className="formatstyle">

    <h1><font color="Brown">Student Details:</font></h1>

    <div className="Name">

      <b><span> Name: </span></b>

      <span>{Name}</span>

    </div>

    <div className="School">

      <b><span> School: </span></b>

      <span>{School}</span>

    </div>

    <div className="Total">

      <b><span>Total:</span></b>

      <span>{total}</span>

      <span> Marks</span>

    </div>

    <div className="Score">

      <b>Score:</b>

      <span>

        {calcScore(total, goal)}

      </span>

    </div>

  </div>

);

**./src/StyleSheets/mystyle.css:**

.Name

{

  font-weight: 300;

  color: blue;

}

.School

{

  color: crimson;

}

.Total

{

  color:darkmagenta;

}

.formatstyle

{

  text-align:center;

  font-size:large;

}

.Score

{

  color: forestgreen;

}

**App.js:**

import { CalculateScore } from './Components/CalculateScore';

function App()

{

  return(

    <div>

      <CalculateScore Name={"Somya Ranjan Sahu"}

      School={"Kalinga Institute of Industrial Technology"}

      total={284}

      goal={300}

      />

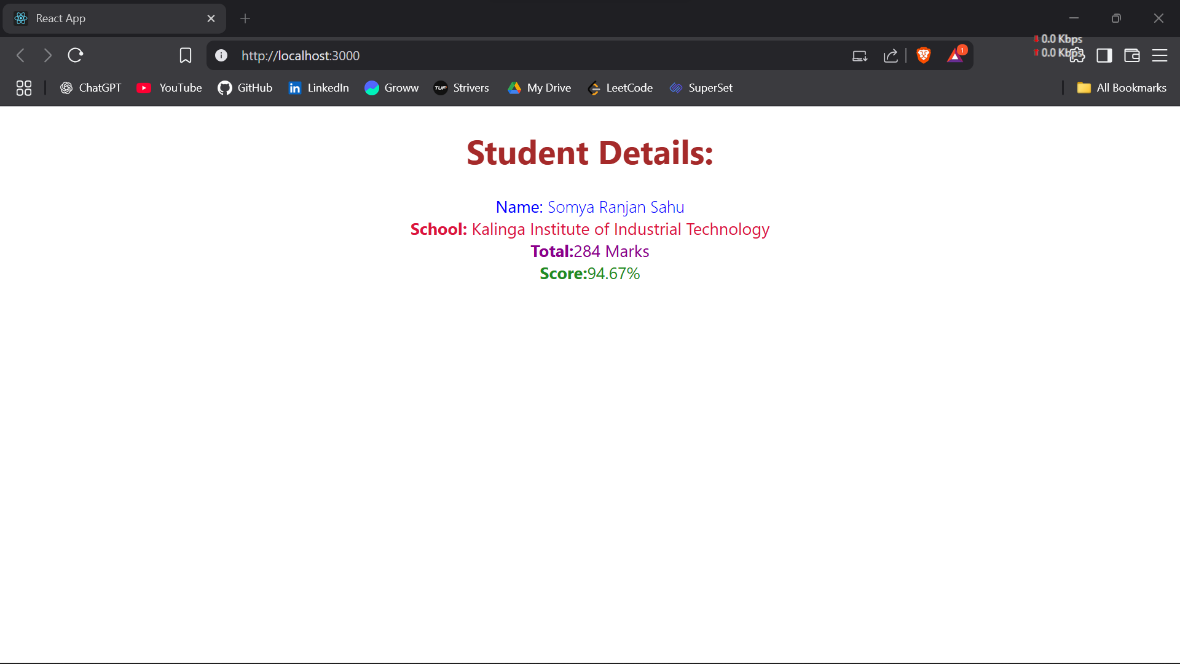
    </div>

  )

}

export default App;

**OUTPUT:**

****

**HANDS-ON: 4**

**Objectives**

* Explain the need and Benefits of component life cycle

Knowing the component lifecycle in React is crucial when dealing with the behavior of components over time. Every component in React has a lifecycle that includes stages such as mounting, updating, and unmounting. React has lifecycle methods (or hooks) that enable developers to execute code at certain places within this cycle—a case in point being loading data after mounting a component or disposing of resources before it gets eliminated from the DOM.

The requirement for lifecycle methods comes into play when a developer wishes to execute activities such as fetching data, setting timers, or DOM manipulation at specific points within the life of a component. This guarantees optimal performance and resource handling. Lifecycle methods bring structure and predictability to the way in which a component will react across its life.

* Identify various life cycle hook methods

In class components, lifecycle hook functions are `constructor()`, `componentDidMount()`, `componentDidUpdate()`, and `componentWillUnmount()`. In functional components, there are hooks such as `useEffect()` that perform similar functions. The common order of steps involved in a component's rendering are: (1) Initialization through constructor, (2) Mounting through `render()` and `componentDidMount()`, (3) Updating through `setState()` or new props invoking `componentDidUpdate()`, and (4) Unmounting through `componentWillUnmount()`.

* List the sequence of steps in rendering a component

React is the waiter who puts in requests from customers and brings them their orders. This process of requesting and serving UI has three steps:

Triggering a render (delivering the guest’s order to the kitchen)

Rendering the component (preparing the order in the kitchen)

Committing to the DOM (placing the order on the table)

**Question:**

Create a new react application using *create-react-app* tool with the name as “blogapp”

**CODE:**

**./src/Post.js:**

import React, { Component } from 'react';

import Post from './Post';

class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: [],

      hasError: false,

    };

  }

  loadPosts = async () => {

    try {

      const response = await fetch('https://jsonplaceholder.typicode.com/posts');

      const data = await response.json();

      const postObjects = data.map(post => new Post(post.userId, post.id, post.title, post.body));

      this.setState({ posts: postObjects });

    } catch (error) {

      this.setState({ hasError: true });

      throw error;

    }

  };

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

    alert('An error occurred while loading posts.');

    console.error('Error caught in component:', error, info);

  }

  render() {

    return (

      <div>

        <h1>Blog Posts</h1>

        {this.state.posts.map(post => (

          <div key={post.id}>

            <h2>{post.title}</h2>

            <p>{post.body}</p>

            <hr />

          </div>

        ))}

      </div>

    );

  }

}

export default Posts;

**./src/Post.js:**

class Post {

    constructor(id, title, body){

        this.id = id;

        this.title = title;

        this.body = body;

    }

}

export default Post;

**App.js:**

import React from 'react';

import './App.css';

import Posts from './Posts';

function App() {

  return (

    <div className="App">

      <Posts />

    </div>

  );

}

export default App;

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**HANDS-ON: 5**

**Objectives**

* Understanding the need for styling react component

Styling is essential in React to create visually appealing, user-friendly applications. In a component-based framework like React, managing styles effectively is crucial to prevent conflicts and keep the codebase maintainable. Traditional global CSS can lead to class name collisions and unpredictable behavior, which is why scoped styling methods are preferred.

* Working with CSS Module and inline styles

A **CSS Module** is a CSS file where all class names and animation names are scoped locally by default. This means a class name you define in one component's CSS module won't conflict with the same class name in another component's module.

**Question:**

In this hands-on lab, you will learn how to:

* Style a react component
* Define styles using the CSS Module
* Apply styles to components using className and style properties

**CODE:**

**./src/CohortDetails.module.css:**

.box {

  width: 300px;

  display: inline-block;

  margin: 10px;

  padding: 10px 20px;

  border: 1px solid black;

  border-radius: 10px;

}

dt {

  font-weight: 500;

}

**CohortDetails.js:**

import React from 'react';

import styles from './CohortDetails.module.css';

function CohortDetails(props) {

    const { cohort } = props;

    const headingColor = cohort.currentStatus.toLowerCase() === 'ongoing' ? 'green' : 'blue';

    return (

        <div className={styles.box}>

            <h3 style={{ color: headingColor }}>

                {cohort.cohortCode} - <span>{cohort.technology}</span>

            </h3>

            <dl>

                <dt>Started On</dt>

                <dd>{cohort.startDate}</dd>

                <dt>Current Status</dt>

                <dd>{cohort.currentStatus}</dd>

                <dt>Coach</dt>

                <dd>{cohort.coachName}</dd>

                <dt>Trainer</dt>

                <dd>{cohort.trainerName}</dd>

            </dl>

        </div>

    );

}

export default CohortDetails;

**App.js:**

import './App.css';

import { CohortsData } from './Cohort';

import CohortDetails from './CohortDetails';

function App() {

  return (

    <div>

      <h1>Cohorts Details</h1>

      {CohortsData.map((cohort, index) => (

        <CohortDetails key={cohort.cohortCode} cohort={cohort} />

      ))}

    </div>

  );

}

export default App;

**OUTPUT:**

