**FSD Lab 5**

**Dev Desai PF – 29**

**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 a JavaScript library for building user interfaces, developed by Facebook. It allows developers to create reusable UI components and efficiently update the user interface in response to changes in data. React follows a component-based architecture, where the UI is broken down into modular components, making it easier to manage and maintain.

To run a React app using create-react-app, follow these steps:

1. Install Node.js and npm (Node Package Manager) if not already installed.
2. Open a terminal and run the following command to install create-react-app globally: npm install -g create-react-app
3. Create a new React app by running:

npx create-react-app my-react-app

1. Change to the app's directory: cd my-react-app
2. Start the development server: npm start

This will open the app in your default browser.

Now, you have a basic React app running locally, and you can start building your components and features.

(2) Passing data through props (Small example)

In React, props (short for properties) allow you to pass data from a parent component to a child component. This is essential for building dynamic and reusable components. Here's a small example:

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

const ParentComponent = () => {

const dataToSend = 'Hello from Parent!';

return (

<div>

<h1>Parent Component</h1>

<ChildComponent data={dataToSend} /> </div>

);

};

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

const ChildComponent = (props) => { return (

<div>

<h2>Child Component</h2>

<p>Data received from parent: {props.data}</p> </div>

);

};

export default ChildComponent;

In this example, `ParentComponent` passes the `dataToSend` variable as a prop to `ChildComponent`, which then displays the received data. Props facilitate communication between components, allowing for a flexible and modular design in React applications.

**FAQ:**

(1) What are React states and hooks?

Ans.

● React States:

In React, a state is a fundamental concept that allows components to manage and store local, mutable data. Unlike props, which are external inputs passed to a component, state is internal and provides a way for components to keep track of dynamic information and respond to user interactions.

In class components, state is typically initialized in the constructor using `this.state`, and updates are managed through the `this.setState()` method. For instance:

import React, { Component } from 'react';

class TemperatureConverter extends Component {

constructor(props) {

super(props); this.state = {

celsius: 0,

fahrenheit: 32

};

}

handleCelsiusChange = (e) => {

const celsiusValue = e.target.value;

const fahrenheitValue = (celsiusValue \* 9) / 5 + 32; this.setState({ celsius: celsiusValue, fahrenheit: fahrenheitValue });

};

render() { return (

<div>

<p>Celsius: {this.state.celsius}</p>

<p>Fahrenheit: {this.state.fahrenheit}</p>

<input type="number" value={this.state.celsius} onChange={this.handleCelsiusChange} /> </div>

);

}

}

export default TemperatureConverter;

With the advent of functional components, React introduced Hooks to allow the use of state and other features in functional components.

● React Hooks:

React Hooks provide a way to use state and other React features in functional components. The most commonly used hooks include `useState` and `useEffect`.

- `useState`: This hook allows functional components to declare state variables. It returns an array containing the current state value and a function to update it.

import React, { useState } from 'react';

function Counter() {

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

return (

<div>

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

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

);

}

export default Counter;

- `useEffect`: This hook is used for side effects in functional components, such as data fetching or subscribing to external events.

React Hooks simplify state management and side effect handling in functional components, enhancing the overall developer experience in building React applications.

Screenshots -



CODE –

import React, { useState } from 'react'

import './calculator.css';

const Calculator = () => {

let [result, setResult] = useState("");

const handleClick = (e) => {

if(result.length >=16){

setResult("!So Much Big Input");

return;

}

if (result.charAt(0) === '0') {

result = result.slice(1, result.length)

}

setResult(result.concat(e.target.name));

}

const clear = () => {

setResult("");

}

const backSpace = () => {

setResult(result.slice(0, result.length - 1))

}

const calculate = () => {

try {

result=eval(result).toString()

if(result.includes('.')){

result=+ eval(result);

result=result.toFixed(4).toString();

setResult(result);

}else{

setResult(eval(result).toString());

}

} catch (err) {

setResult("Error");

}

}

return (

<div className='container'>

<h3 style={{ fontSize: "25px", color: "chocolate", borderBottom: "2px solid whitesmoke" }}>Calculator</h3>

<form action="">

<input type="text" value={result} />

</form>

<div className='keypad'>

<button onClick={clear} className="clear color">Clear</button>

<button onClick={backSpace} className="backspace color">C</button>

<button name="/" className='color' onClick={handleClick}>/</button>

<button name="7" onClick={handleClick}>7</button>

<button name="8" onClick={handleClick}>8</button>

<button name="9" onClick={handleClick}>9</button>

<button name="\*" className='color' onClick={handleClick}>\*</button>

<button name="4" onClick={handleClick}>4</button>

<button name="5" onClick={handleClick}>5</button>

<button name="6" onClick={handleClick}>6</button>

<button name="-" className='color' onClick={handleClick}>-</button>

<button name="1" onClick={handleClick}>1</button>

<button name="2" onClick={handleClick}>2</button>

<button name="3" onClick={handleClick}>3</button>

<button name="+" className='color' onClick={handleClick}>+</button>

<button name="0" onClick={handleClick}>0</button>

<button name="." className='color' onClick={handleClick}>.</button>

<button onClick={calculate} className="equal color">=</button>

</div>

</div>

)

}

export default Calculator