Web Programming

Lab - 15

React JSX

Name: Sushen Grover
Reg No: 23BCE1728
Slot: L11+L12+L31+L32
Class No: CH2024250502774
Course Code: BCSE203E
Faculty: Dr. L.M. Jenila Livingston

Question 1:

1. You are developing a React application that consists of multiple functional components (Header, Content, and Footer). The main App component organizes these components and displays them on the screen.

   (i) Your task is to define and export an **App** component that contains multiple components:

   a. A Header component that receives a title as a prop.
   b. A Content component that displays a random joke when a button is clicked.
   c. A Footer component that displays a static footer message.

   (ii) Import and render the App component in **index.js** using ReactDOM.render(). Ensure the **index.html file has a root element** where React will mount the application.

Code:

App.jsx

```
import Header from './components/header.jsx'
import Footer from './components/footer.jsx'
import Content from './components/content.jsx'
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
// import './App.css'
function App() {
  const jokes=[
    "I told my suitcase that there will be no vacations this year… now it's full of emotional baggage.",
    "Parallel lines have so much in common—it's a shame they'll never meet.",
    "I used to play piano by ear, but now I use my hands.",
    "I asked the librarian if the library had any books on paranoia… she whispered, They're right behind
you.",
    "I told my wife she should embrace her mistakes… she gave me a hug.",
    "My friend said he didn't understand cloning—I told him, That makes two of us.",
    "I started a band called 999 Megabytes… we still haven't got a gig.",
    "I tried to catch fog yesterday… but I mist.",
    "I told my dog I needed some space… now he won't stop barking at the stars.",
    "Why did the scarecrow win an award? Because he was outstanding in his field!"
  ];
  function fetchJoke(){
    let ind=Math.floor(Math.random()*9);
    return jokes[ind]
  }
  return (
    <>
      <Header titleName="This is a prop title"></Header>
      <Content getJoke={fetchJoke}></Content>
      <Footer></Footer>
    </>
  )
}

export default App
```

Header.jsx

```
import React from 'react'

const Header = (props) => {
  return (
    <div>
        <h2>{props.titleName}</h2>
    </div>
  )
}
```

```
export default Header
```

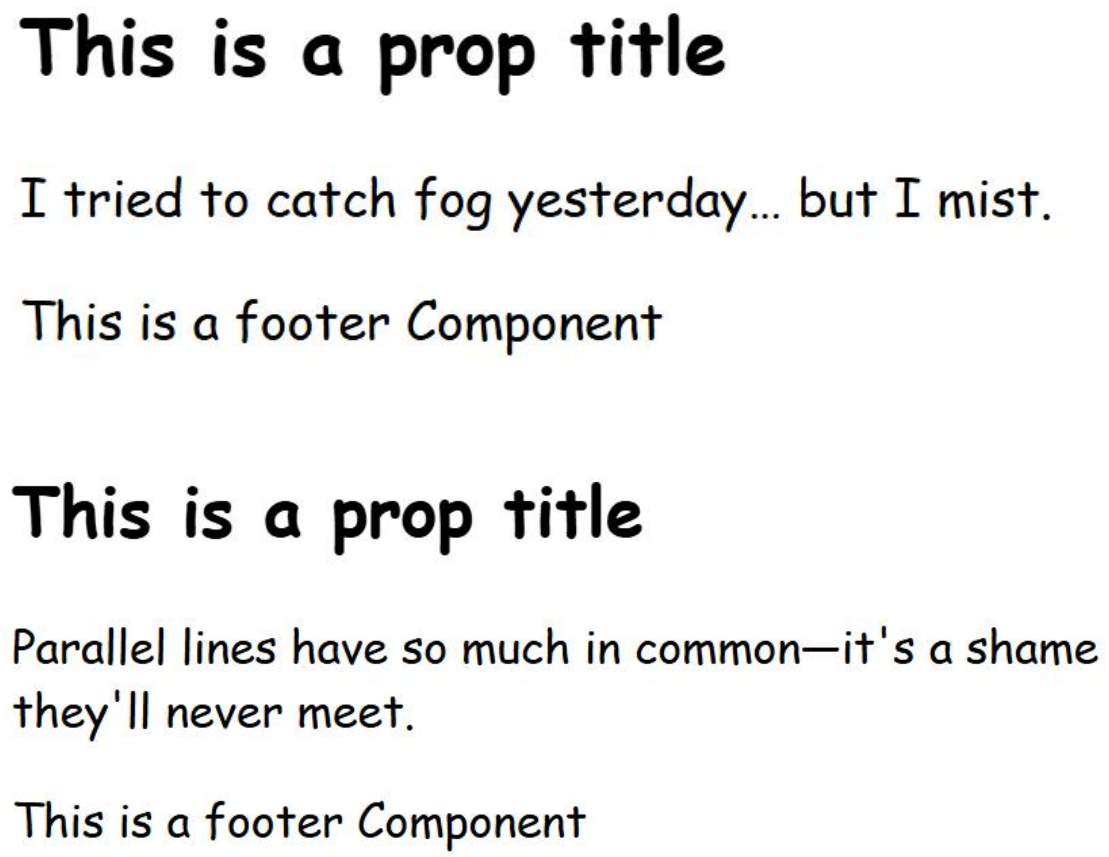
Content.jsx

```
import React from 'react'

const Content = (props) => {
  return (
    <div>
        <p>{props.getJoke()}</p>
    </div>
  )
}
export default Content
```

Footer.jsx

```
import React from 'react'

const Footer = () => {
  return (
    <div>This is a footer Component</div>
  )
}
export default Footer
```

Output:



Question 2,3,4:

2. **Styling in React – Inline CSS:**

- Create a StyledButton component that applies inline CSS for background color, padding, and font size.

3. **Styling in React – Internal CSS:**

- Modify the StyledButton component to include an internal <style> tag within the component for styling.

4. **Styling in React – External CSS:**

- Create a separate styles.css file and apply external styling to the StyledButton component by importing the CSS file.

Code:

App.jsx

```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
// import './App.css'
import StyledButton1 from './Components/StyledButton1'
import StyledButton2 from './Components/StyledButton2'
import StyledButton3 from './Components/StyledButton3'
function App() {
  const style1=`
  p{
  background-color:grey;
  font-size:5px;
  }
  `
  return(
    <>
    <p>Q2 Inline CSS</p>
    <StyledButton1></StyledButton1>
    <p>Q2 Internal CSS</p>
    <StyledButton2></StyledButton2>
    <p>Q3 External CSS</p>
    <StyledButton3></StyledButton3>
    </>
  )
}

export default App
```

styledButton1.jsx

```
import React from 'react'
const StyledButton1 = () => {
  return (
    <div>
        <button style={{backgroundColor:"yellow",padding:5,fontSize:10}}>Button1</button>
    </div>
  )
}
export default StyledButton1
```

styledButton2.jsx

```
import React from 'react'

const StyledButton2 = () => {
  const style1=`
  button{
  background-color:skyblue;
```

```
  padding:5px;
  font-size:10px;
  }
  `;
    return (
    <div>
        <style>{style1}</style>
        <button>Button2</button>
    </div>
  )
}
export default StyledButton2
```

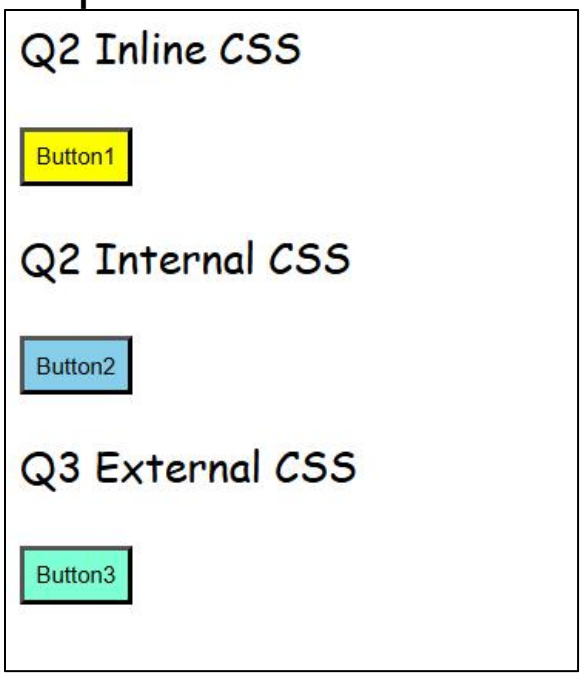
styledButton3.jsx

```
import React from 'react'
import './StyledButton3.css'
const StyledButton3 = () => {
  return (
    <div>
        <button className='styledButton'>Button3</button>
    </div>
  )
}

export default StyledButton3
```

styledButton3.css

```
.styledButton{
    background-color: aquamarine;
    padding: 5px;
    font-size: 10px;
}
```

Output:



Question 5:

5. Develop a LifecycleDemo **class** component that logs messages at each stage of its lifecycle
    - Lifecycle (constructor, componentDidMount, componentDidUpdate, and componentWillUnmount).
    - Implement a button to update the state and trigger componentDidUpdate().
    - Unmount the component dynamically to observe the effect of componentWillUnmount()

Code:

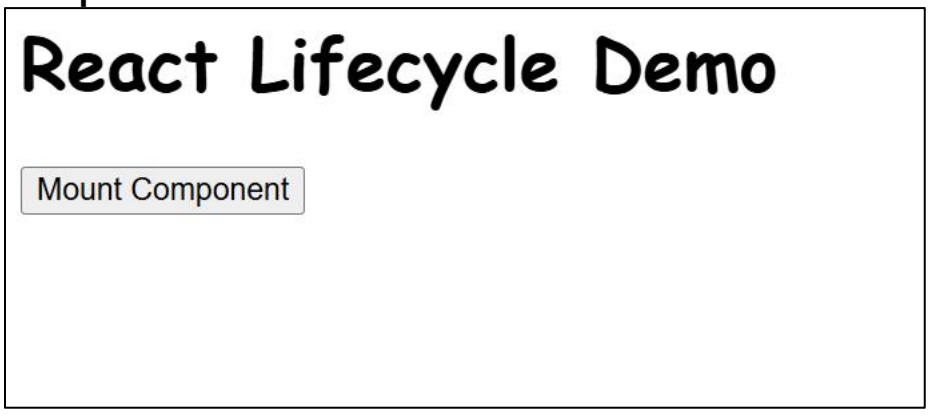
App.jsx

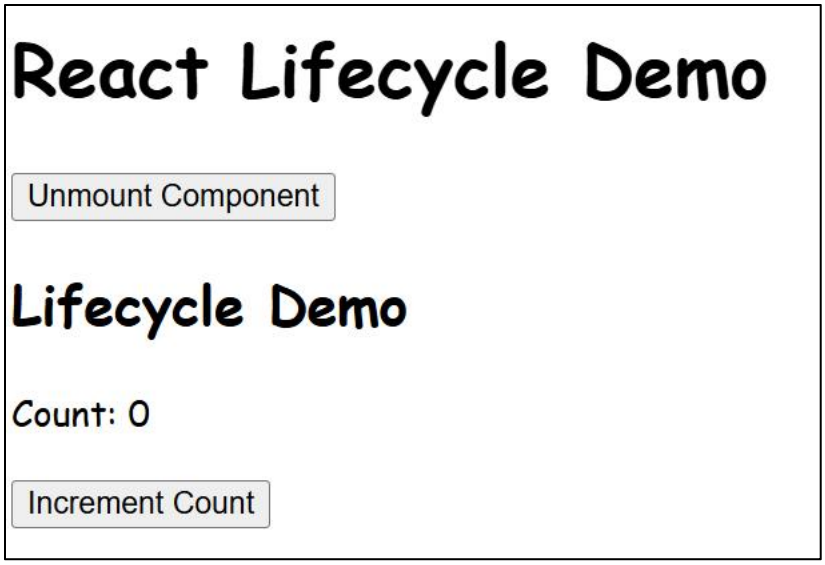
```jsx
import React, { useState } from "react";
import LifecycleDemo from "./Components/LifecycleDemo";

function App() {
  const [showComponent, setShowComponent] = useState(true);
  return (
    <div>
      <h1>React Lifecycle Demo</h1>
      <button onClick={() => setShowComponent(!showComponent)}>
        {showComponent ? "Unmount Component" : "Mount Component"}
      </button>
      {showComponent && <LifecycleDemo />}
    </div>
  );
}
export default App;
```

LifecycleDemo.jsx

```jsx
import React, { Component } from "react";

class LifecycleDemo extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
    console.log("Constructor: Component is initialized");
  }
  componentDidMount() {
    console.log("componentDidMount: Component is mounted");
  }
  componentDidUpdate(prevProps, prevState) {
    console.log("componentDidUpdate: State updated", prevState, "->", this.state);
  }
  componentWillUnmount() {
    console.log("componentWillUnmount: Component is about to be unmounted");
  }
  incrementCount = () => {
    this.setState((prevState) => ({ count: prevState.count + 1 }));
  };
  render() {
    return (
      <div>
        <h2>Lifecycle Demo</h2>
        <p>Count: {this.state.count}</p>
        <button onClick={this.incrementCount}>Increment Count</button>
      </div>
    );
  }
}
export default LifecycleDemo;
```

Output:





Question 6:

6. **State Hooks:**

- Create a React component called Counter using the useState() hook. The component should display a count with two buttons: **Increase** and **Decrease**.
- Modify the component to use the useReducer() hook instead of useState(), handling increment and decrement actions efficiently.

Code:

App.jsx

```jsx
import React from "react";
import Counter1 from "./Components/Counter1";
import Counter2 from "./Components/Counter2";
function App() {
  return (
    <div>
      <h1>React Counter Demo</h1>
      <h2>Using useState</h2>
      <Counter1 />
      <h2>Using useReducer</h2>
      <Counter2 />
    </div>
  );
}

export default App;
```

Counter1.jsx

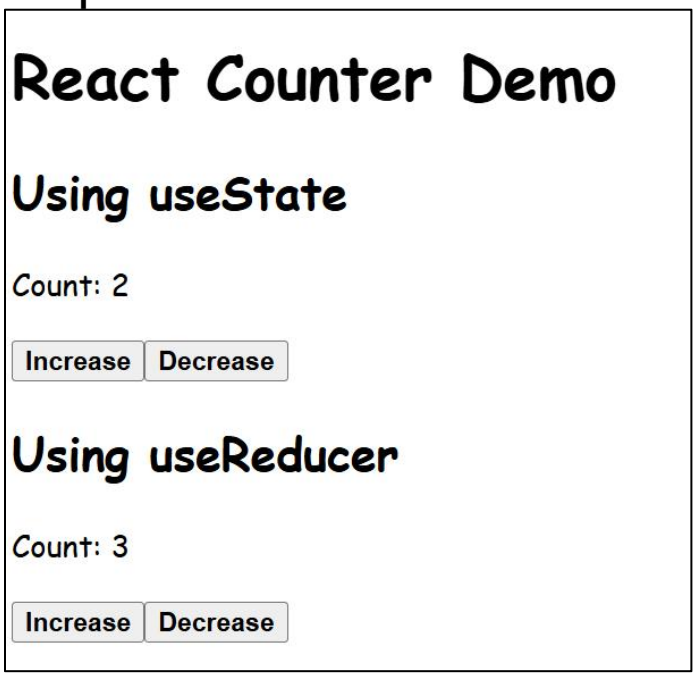
```jsx
import React, { useState } from "react";

function Counter1() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}><strong>Increase</strong></button>
      <button onClick={() => setCount(count - 1)}><strong>Decrease</strong></button>
    </div>
  );
}
export default Counter1;
```

Counter2.jsx

```jsx
import React, { useReducer } from "react";

const reducer = (state, action) => {
```

```
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    default:
      return state;
  }
};
function Counter2() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });
  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: "INCREMENT" })}><strong>Increase</strong></button>
      <button onClick={() => dispatch({ type: "DECREMENT" })}><strong>Decrease</strong></button>
    </div>
  );
}
export default Counter2;
```

Output:



Question 7:

7. **Effect Hooks (useEffect):**

- Develop a React component that fetches and displays a random joke from an API when the component mounts.
- Add functionality to refresh the joke when a button is clicked.

Code:

App.jsx

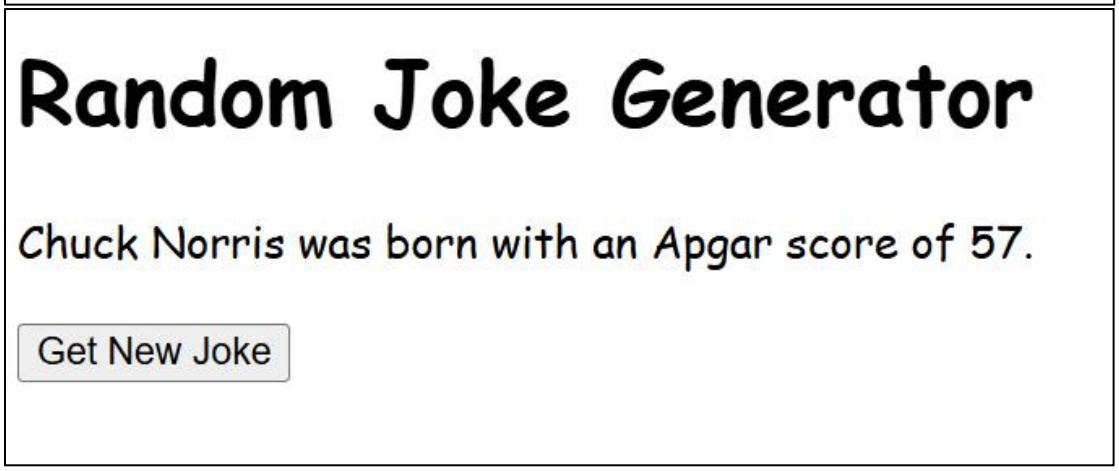
```
import React from "react";
import Joke from "./Components/Joke";
function App() {
  return (
    <div>
      <h1>Random Joke Generator</h1>
      <Joke />
    </div>
  );
}
```

```
export default App;
```

Joke.jsx

```
import React, { useState, useEffect } from "react";

function Joke() {
  const [joke, setJoke] = useState("");
  const fetchJoke = async () => {
    try {
      const response = await fetch("https://api.chucknorris.io/jokes/random");
      const data = await response.json();
      setJoke(data.value);
    } catch (error) {
      console.error("Error fetching joke:", error);
    }
  };
  useEffect(() => {
    fetchJoke();
  }, []);
  return (
    <div>
      <p>{joke || "Loading joke..."}</p>
      <button onClick={fetchJoke}>Get New Joke</button>
    </div>
  );
}
export default Joke;
```

Output:



Question 8:

8. **Ref Hooks (useRef)**:

- Build a simple form with an input field and a button.
- When the button is clicked, the input field should automatically get focused using the useRef() hook.

Code:

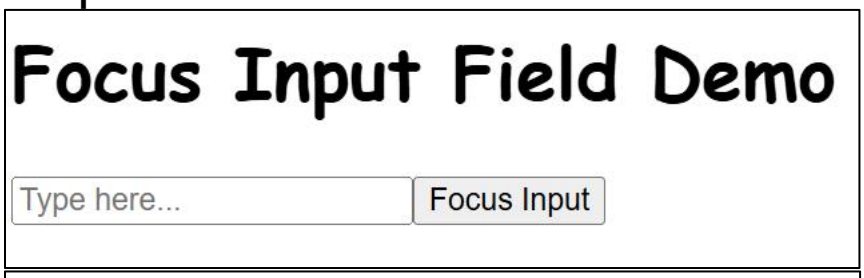
App.jsx

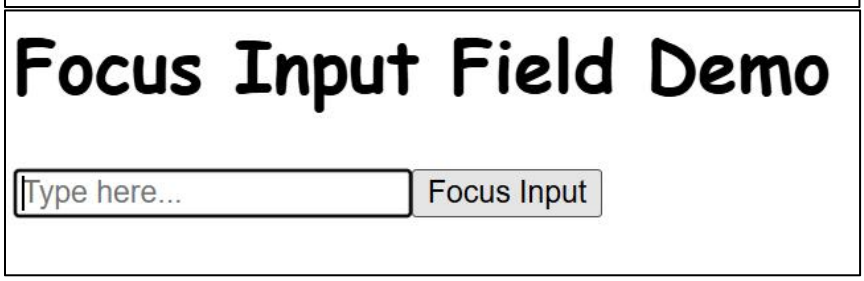
```jsx
import React from "react";
import Form from "./Components/Form";
function App() {
  return (
    <div>
      <h1>Focus Input Field Demo</h1>
      <Form />
    </div>
  );
}

export default App;
```

Form.jsx

```jsx
import React, { useRef } from "react";

function Form() {
  const inputRef = useRef(null);
  const handleFocus = () => {
    if (inputRef.current) {
      inputRef.current.focus();
    }
  };
  return (
    <div>
      <input ref={inputRef} type="text" placeholder="Type here..." />
      <button onClick={handleFocus}>Focus Input</button>
    </div>
  );
}
export default Form;
```

Output:





Question 9:

9. **Context Hooks (useContext)**:

- Create a React application where the theme (dark or light mode) is shared across multiple components using useContext().
- Implement a button to toggle between dark and light themes.

Code:

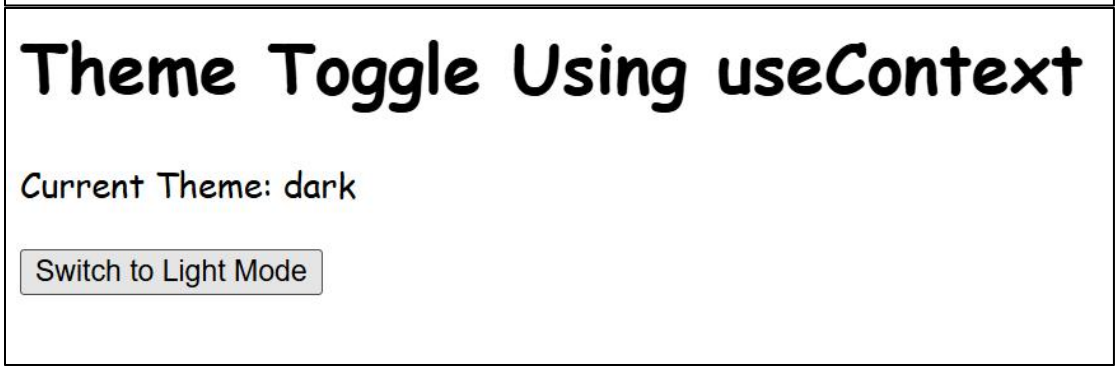
App.jsx

```jsx
import React, { createContext, useState } from "react";
import Mode from './Components/Mode.jsx'
export const ThemeContext = createContext();

function App() {
  const [theme, setTheme] = useState("light");
  const toggleTheme = () => {
    setTheme((prevTheme) => (prevTheme === "light" ? "dark" : "light"));
  };
  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      <div className={theme === "light" ? "light-theme" : "dark-theme"}>
        <h1>Theme Toggle Using useContext</h1>
        <Mode />
      </div>
    </ThemeContext.Provider>
  );
}
export default App;
```

Mode.jsx

```jsx
import React, { useContext } from "react";
import { ThemeContext } from "../App";

function Mode() {
  const { theme, toggleTheme } = useContext(ThemeContext);
  return (
    <div>
      <p>Current Theme: {theme}</p>
      <button onClick={toggleTheme}>
        Switch to {theme === "light" ? "Dark" : "Light"} Mode
      </button>
    </div>
  );
}
export default Mode;
```

Output:



Question 10,11:

10. **React Props:**

- Design a Parent component that sends a message prop to a Child component.
- Ensure the Child component properly receives and displays the message.

11. **React Props Validation:**

- Modify the Child component to validate the message prop using prop-types.
- Ensure that the prop is required and of type string.

Code:

App.jsx

```jsx
import React from "react";
import ChildComponent from "./Components/ChildComponent";

function App() {
  return (
    <div>
      <h1>React Props & Prop Validation</h1>
      <ChildComponent message="Hello from Parent Component!" />
    </div>
  );
}
export default App;
```

ChildComponent.jsx

```jsx
import React from "react";
import PropTypes from "prop-types";

function ChildComponent({ message }) {
  return (
    <div>
      <p>Message from Parent: {message}</p>
    </div>
  );
}
// Prop validation using prop-types
ChildComponent.propTypes = {
  message: PropTypes.string.isRequired, // Ensures `message` is a required string
};
export default ChildComponent;
```

Output:

Question 12:

12. Passing Values from a Form Using useState and useRef

(i) Create a form with fields for **Name** and **Email**. Use useState to manage input values and display them dynamically.

- Create a new React component.
- Use useState to track form values.
- Display the values dynamically as the user types.
- Submit the form and prevent default page reload.

(ii) Create the same form but use useRef to retrieve values on form submission without managing state updates.

- Create a new React component.
- Use useRef to get form values.
- Display values only when the form is submitted.

Code:

App.jsx

```
import React from "react";
import Form1 from "./Components/Form1";
import Form2 from "./Components/Form2";
function App() {
  return (
    <div>
      <h1>Form Handling with useState and useRef</h1>
      <h2>Using useState:</h2>
      <Form1 />
      <h2>Using useRef:</h2>
      <Form2 />
    </div>
  );
}

export default App;
```

Form1.jsx

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

function Form1() {
  const [formData, setFormData] = useState({ name: "", email: "" });
  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Submitted Data: Name - ${formData.name}, Email - ${formData.email}`);
  };
  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input
          type="text"
          name="name"
          value={formData.name}
          onChange={handleChange}
        />
      </label>
      <br />
      <label>
        Email:
        <input
          type="email"
          name="email"
          value={formData.email}
          onChange={handleChange}
```

```
        />
      </label>
      <br />
      <button type="submit">Submit</button>
      <h3>Live Preview</h3>
      <p>Name: {formData.name}</p>
      <p>Email: {formData.email}</p>
    </form>
  );
}
export default Form1;
```

Form2.jsx

```
import React, { useRef, useState } from "react";

function Form2() {
  const nameRef = useRef();
  const emailRef = useRef();
  const [submittedData, setSubmittedData] = useState(null);
  const handleSubmit = (e) => {
    e.preventDefault();
    setSubmittedData({
      name: nameRef.current.value,
      email: emailRef.current.value,
    });
  };
  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" ref={nameRef} />
      </label>
      <br />
      <label>
        Email:
        <input type="email" ref={emailRef} />
      </label>
      <br />
      <button type="submit">Submit</button>
      {submittedData && (
        <div>
          <h3>Submitted Data</h3>
          <p>Name: {submittedData.name}</p>
          <p>Email: {submittedData.email}</p>
        </div>
      )}
    </form>
  );
}
export default Form2;
```

Output:

Form Handling with useState and useRef

Using useState:

Name:
Email:
Submit

Live Preview

Name:

Email:

Using useRef:

Name:
Email:
Submit

Form Handling with useState and useRef

Using useState:

Name: Sushen
Email: groversushen@gmail.com
Submit

Live Preview

Name: Sushen

Email: groversushen@gmail.com

Using useRef:

Name: Sushen
Email: groversushen@gmail.com
Submit

Submitted Data

Name: Sushen

Email: groversushen@gmail.com