Web Programming

Exercise 15

Name: B.Venkatesh
Reg No: 23BCE1012
Course Code and Title: BCSE203E
Slot: TE1

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:

1) **App.js**

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

const Header = ({ title }) => {
  return <h1>{title}</h1>;
};

const Content = () => {
  const [joke, setJoke] = useState("");

  const getJoke = () => {
    fetch("https://official-joke-api.appspot.com/random_joke")
      .then((response) => response.json())
      .then((data) => setJoke(`${data.setup} - ${data.punchline}`));
```

```
  };

  return (
    <div>
       <button onClick={getJoke}>Get Random Joke</button>
      <p>{joke}</p>
    </div>
  );
};

const Footer = () => {
  return <footer>© 2025 My React App</footer>;
};

const App = () => {
  return (
    <div>
      <Header title="Hello World from 23BCE1012" />
      <Content />
      <Footer />
    </div>
  );
};

export default App;
```

2) index.js

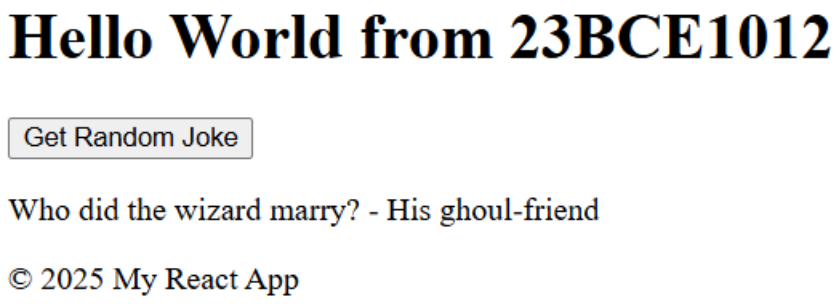
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <App />
);
```

3) index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
```

```
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

Output:



2) Styling in React – Inline CSS:

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

Code:

```
import React from "react";

const StyledButton = ({title, doTask}) => {
  const buttonStyle = {
    backgroundColor: "blue",
    padding: "10px 20px",
    fontSize: "16px",
    color: "white",
    border: "none",
    cursor: "pointer"
  };

  return <button style={buttonStyle} onClick={doTask}>{title}</button>;
};

export default StyledButton;
```

3) Styling in React – Internal CSS:

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

Code:

```
import React from "react";

const StyledButtonInternal = () => {
  return (
    <div>
      <style>
        {`
          .styled-button2{
            background-color: yellow;
            padding: 10px 20px;
            font-size: 16px;
            color: black+;
            border: none;
            cursor: pointer;
          }
        `}
      </style>
      <button className="styled-button2">Styled Button Internal</button>
    </div>
  );
};

export default StyledButtonInternal;
```

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:

1) **StyledButtonExternal.js**

```
import React from "react";
import "./StyledButtonExternal.css";

const StyledButtonExternal = () => {
  return <button className="styled-button">Styled Button External</button>;
};
```

```
export default StyledButtonExternal;
```

2) StyledButtonExternal.css

```
.styled-button {
   background-color: red;
   padding: 10px 20px;
   font-size: 16px;
   color: white;
   border: none;
   cursor: pointer;
 }
```

Output:

5) Develop a LifecycleDemo class component that logs messages at each stage of its lifecycle

i) Lifecycle (constructor, componentDidMount, componentDidUpdate, and componentWillUnmount).

ii) Implement a button to update the state and trigger componentDidUpdate().

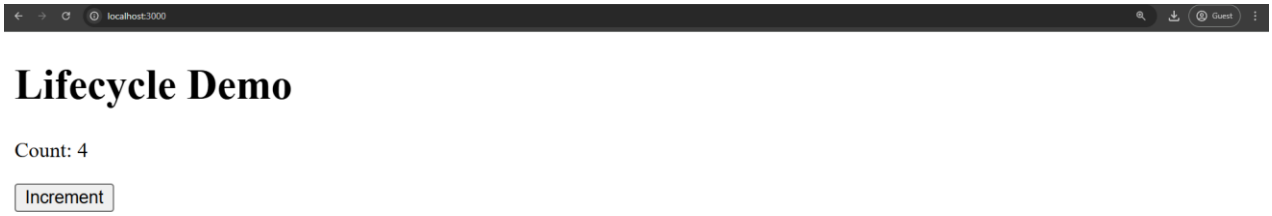
iii) Unmount the component dynamically to observe the effect of componentWillUnmount()

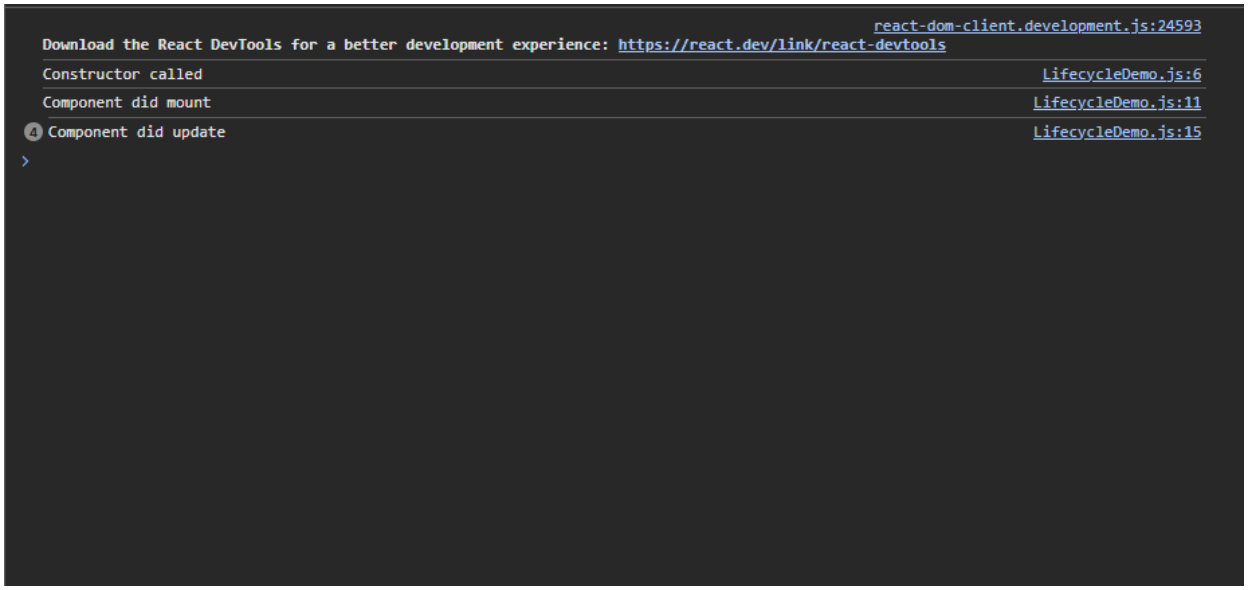
Code:

1) **LifecycleDemo.js**

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

class LifecycleDemo extends Component {
  constructor(props) {
    super(props);
    console.log("Constructor called");
    this.state = { count: 0 };
  }

  componentDidMount() {
    console.log("Component did mount");
  }

  componentDidUpdate(prevProps, prevState) {
    console.log("Component did update");
  }

  componentWillUnmount() {
    console.log("Component will unmount");
  }

  handleClick = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <h1>Lifecycle Demo</h1>
        <p>Count: {this.state.count}</p>
        <button onClick={this.handleClick}>Increment</button>
      </div>
    );
  }
}

export default LifecycleDemo;
```

Output:





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:

1) **Counter.js**

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

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

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increase</button>
      <button onClick={() => setCount(count - 1)}>Decrease</button>
    </div>
  );
};

export default Counter;
```

2) **Counter.js**

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

const counterReducer = (state, action) => {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    default:
      return state;
  }
};

const Counter = () => {
  const [state, dispatch] = useReducer(counterReducer, { count: 0 });

  return (
    <div>
      <h1>Count: {state.count}</h1>
      <button onClick={() => dispatch({ type: "INCREMENT" })}>Increase</button>
      <button onClick={() => dispatch({ type: "DECREMENT"
})}>Decrease</button>
    </div>
  );
};
```

```
export default Counter;
```

Output:

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:

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

const JokeFetcher = () => {
  const [joke, setJoke] = useState("");

  useEffect(() => {
    fetch("https://official-joke-api.appspot.com/random_joke")
      .then((response) => response.json())
      .then((data) => setJoke(`${data.setup} - ${data.punchline}`));
  }, []);

  const refreshJoke = () => {
    fetch("https://official-joke-api.appspot.com/random_joke")
      .then((response) => response.json())
      .then((data) => setJoke(`${data.setup} - ${data.punchline}`));
  };

  return (
    <div>
      <p>{joke}</p>
      <button onClick={refreshJoke}>Get New Joke</button>
    </div>
  );
};

export default JokeFetcher;
```

Output:


localhost:3000
Guest
Why couldn't the bicycle stand up by itself? - It was two-tired.
Get New Joke


localhost:3000
Guest
Which song would an exception sing? - Can't catch me - Avicii
Get New Joke

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:

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

const FocusInput = () => {
  const inputRef = useRef();

  const focusInput = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <input ref={inputRef} type="text" placeholder="Focus me!" />
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
};

export default FocusInput;
```

Output:

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:

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

const ThemeContext = createContext();

const ThemeProvider = ({ children }) => {
  const [isDark, setIsDark] = useState(false);
  const toggleTheme = () => setIsDark(!isDark);

  return (
    <ThemeContext.Provider value={{ isDark, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

const ThemeButton = () => {
  const { isDark, toggleTheme } = useContext(ThemeContext);

  return (
    <button
      style={{
        backgroundColor: isDark ? "black" : "white",
        color: isDark ? "white" : "black",
      }}
      onClick={toggleTheme}
    >
      Toggle Theme
    </button>
  );
};

const ThemedApp = () => {
  return (
    <ThemeProvider>
      <ThemeButton />
    </ThemeProvider>
  );
};
```

```
export default ThemedApp;
```

Output:

Toggle Theme

Toggle Theme

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.**

Code:

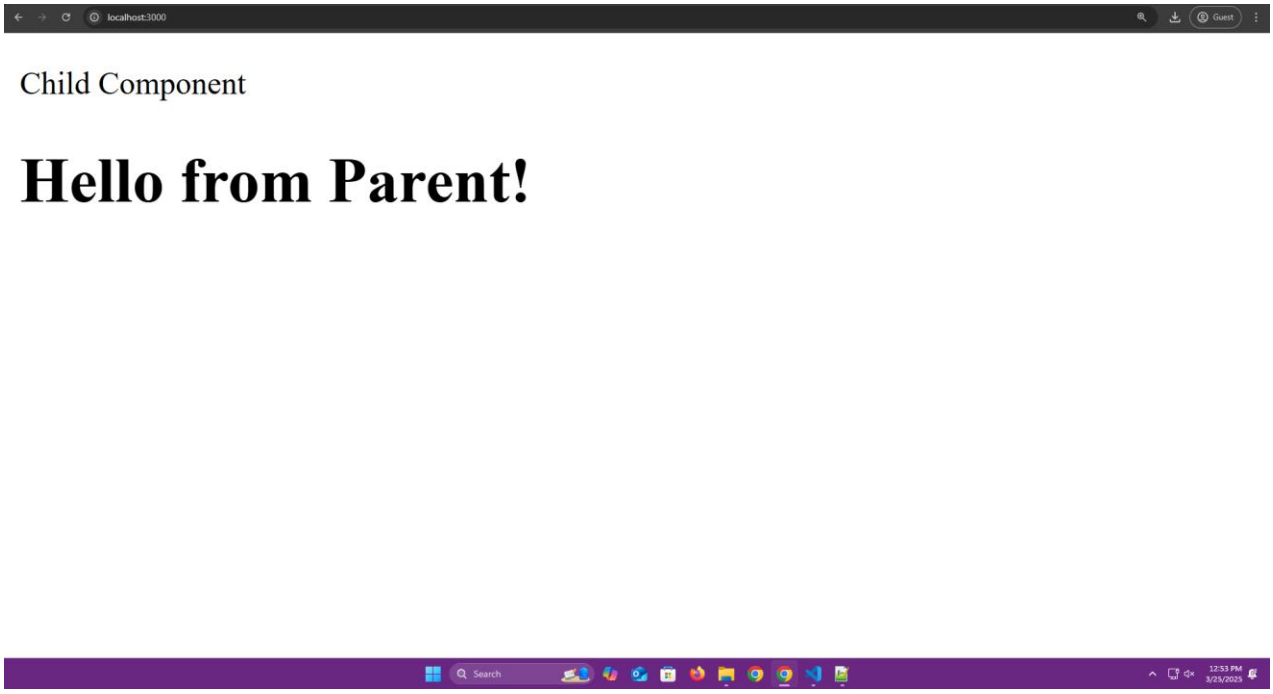
1) **Parent.js**

```
import React from "react";
import Child from "./Child";

const Parent = () => {
  return <Child message="Hello from Parent!" />;
};

export default Parent;
```

2) **Child.js**

```
import React from "react";

const Child = ({ message }) => {
  return (<>
  <p>Child Component</p>
  <h1>{message}</h1>
  </>);
};

export default Child;
```

Output:



11) React Props Validation:

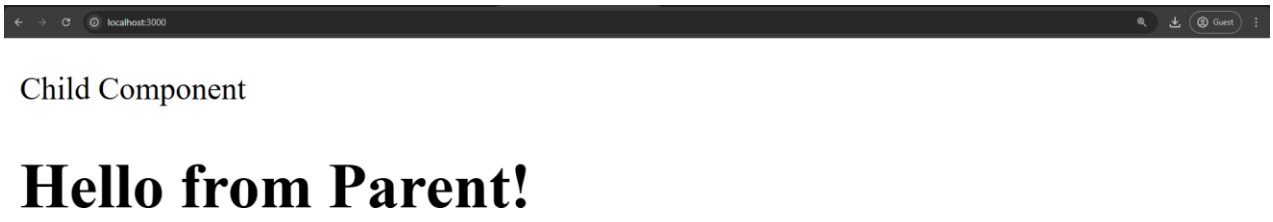
- **Modify the Child component to validate the message prop using prop-types.**
- **Ensure that the prop is required and of type string.**

Code:

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

const Child = ({ message }) => {
  return (<>
  <p>Child Component</p>
  <h1>{message}</h1>
  </>);
};

Child.propTypes = {
  message: PropTypes.string.isRequired,
};
```

```
export default Child;
```

Output:



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:

1) **FormWithState.js**

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

const FormWithState = () => {
  const [formData, setFormData] = useState({ name: "", email: "" });

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log("Submitted Data:", formData);
  };

  return (
    <div>
      <h2>Form Using useState</h2>
      <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>
      </form>
      <h3>Live Preview:</h3>
      <p>Name: {formData.name}</p>
      <p>Email: {formData.email}</p>
    </div>
  );
};

export default FormWithState;
```

2) FormWithRef.js

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

const FormWithRef = () => {
  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 (
    <div>
      <h2>Form Using useRef</h2>
      <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>
      </form>
      {submittedData && (
        <div>
          <h3>Submitted Data:</h3>
          <p>Name: {submittedData.name}</p>
          <p>Email: {submittedData.email}</p>
        </div>
      )}
    </div>
  );
};

export default FormWithRef;
```

Output:

1) **Using State**

Form Using useState

Name: Venkatesh

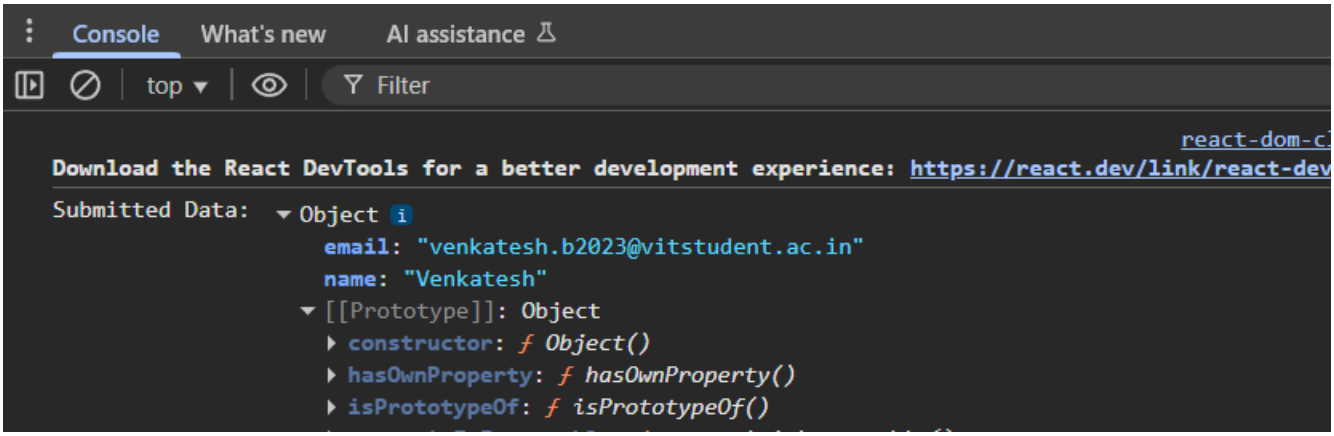
Email: venkatesh.b2023@vitstude

Submit

Live Preview:

Name: Venkatesh

Email: venkatesh.b2023@vitstudent.ac.in



2) **Using Ref**

Form Using useRef

Name: venkatesh
Email: venkatesh.b2023@vitstude
Submit

Submitted Data:

Name: venkatesh

Email: venkatesh.b2023@vitstudent.ac.in