Name : T.B.Madhan **Reg No :** 23BCE1087

Web Programming Lab (BCSE203E)
LAB – 15
JSX – Part 3

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:

(i) ***App.jsx***

```
import React from "react";
function Header(props) {
    return (
        <>
            <header>
                <h1>{props.title}</h1>
            </header>
        </>
    );
}
function Main() {
    let arr = [
```

```
        "Why don’t skeletons fight each other? They don’t have the
guts.",
        "Why did the scarecrow win an award? He was outstanding in
his field!",
        "Why did the math book look sad? Because it had too many
problems.",
        "Why don't eggs tell jokes? They'd crack each other up!",
        "Why did the bicycle fall over? It was two-tired!",
    ];

    return (
        <>
            <button
                onClick={() => {
                    var random = Math.floor(Math.random() *
arr.length);
                    document.getElementById("joke").innerHTML =
arr[random];
                }}
            >
                Tell me a joke!
            </button>
            <p id="joke"></p>
        </>
    );
}
function Footer() {
    return (
        <footer>
            <h3>This is the Footer</h3>
        </footer>
    );
```

```
}
function Q1() {
    return (
        <div style={{ textAlign: "center" }}>
            <Header title="Joke Teller" />
            <Main />
            <Footer />
        </div>
    );
}
export default App;
```

Index.html

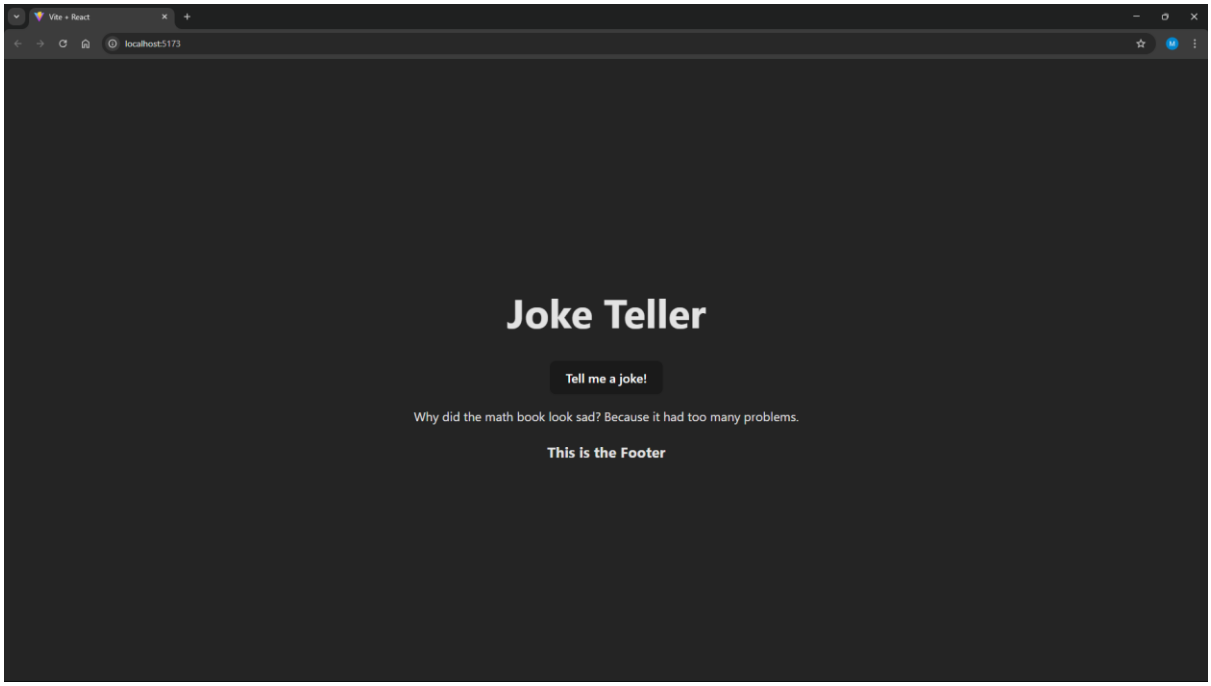
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <link rel="icon" type="image/svg+xml" href="/vite.svg" />
        <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
        <title>Vite + React</title>
    </head>
    <body>
        <div id="root"></div>
        <script type="module" src="/src/index.jsx"></script>
    </body>
</html>
```

Index.jsx

```
import React from "react";
import ReactDOM from "react-dom";
```

```
import "./index.css";
import App from "./App.jsx";

ReactDOM.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>,
    document.getElementById("root")
);
```

Output:

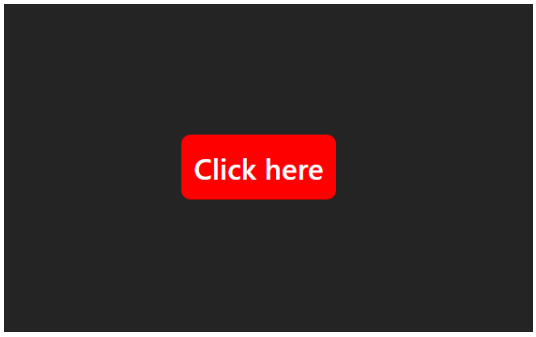


2) Styling in React – Inline CSS:

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

```
import reactLogo from "./assets/react.svg";
import viteLogo from "/vite.svg";
```

```
function styledButton() {
    return (
        <div
            style={{
                display: "flex",
                height: "100vh",
                justifyContent: "center",
                alignItems: "center",
            }}
        >
            <button
                style={{
                    backgroundColor: "Red ",
                    padding: "10px",
                    fontSize: "25px",
                }}
            >
                Click here
            </button>
        </div>
    );
}

export default styledButton;
```



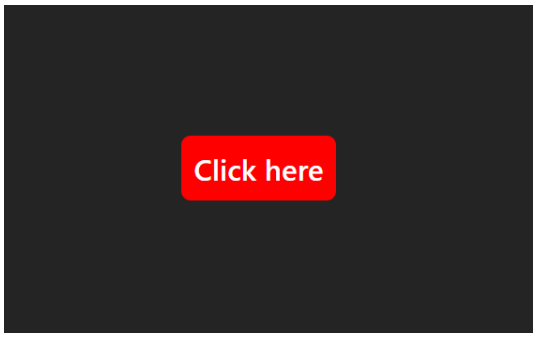
3) **Styling in React – Internal CSS:**
   - **Modify the StyledButton component to include an internal**

Code*:*

StyledButton.jsx

```
function StyledButton() {
    return (
        <>
            <style>
                {`
                    .center-container {
                        display: flex;
                        height: 100vh;
                        justify-content: center;
                        align-items: center;
                    }

                    .red-button {
                        background-color: red;
                        padding: 10px;
                        font-size: 25px;
                    }
                `}
            </style>

            <div className="center-container">
                <button className="red-button">Click here</button>
            </div>
        </>
```

```
    );
}
```



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.

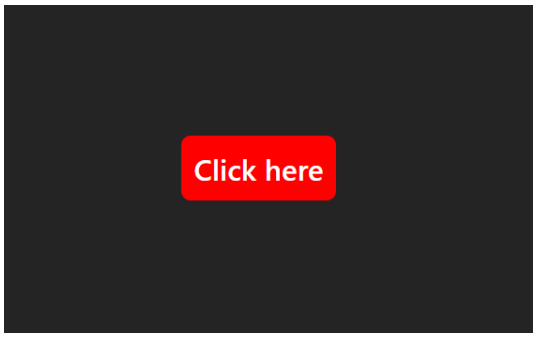
StyledButton.css

```
.center-container {
    display: flex;
    height: 100vh;
    justify-content: center;
    align-items: center;
}

.red-button {
    background-color: red;
    padding: 10px;
    font-size: 25px;
}
```

StyledButton.jsx

```
import './StyledButton.css';

function StyledButton() {
    return (
        <div className="center-container">
            <button className="red-button">Click here</button>
        </div>
    );
}

export default StyledButton;
```

Output:



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

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

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

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

Code:

Q5.jsx

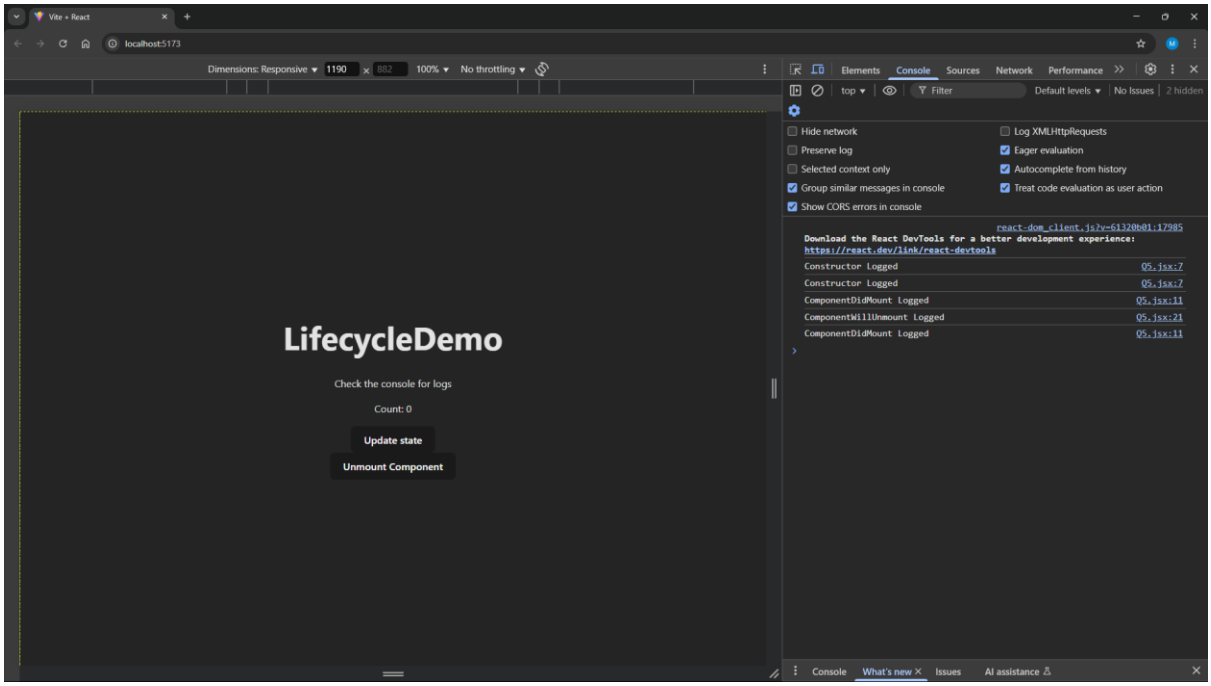
```
import React, { Component } from "react";

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

    componentDidMount() {
        console.log("ComponentDidMount Logged");
    }

    componentDidUpdate(prevProps, prevState) {
        console.log("ComponentDidUpdate Logged");
        console.log("Previous state:", prevState.count);
        console.log("Current state:", this.state.count);
    }

    componentWillUnmount() {
        console.log("ComponentWillUnmount Logged");
    }

    incrementCount = () => {
        this.setState((prevState) => ({ count: prevState.count + 1
}));
    };

    render() {
```

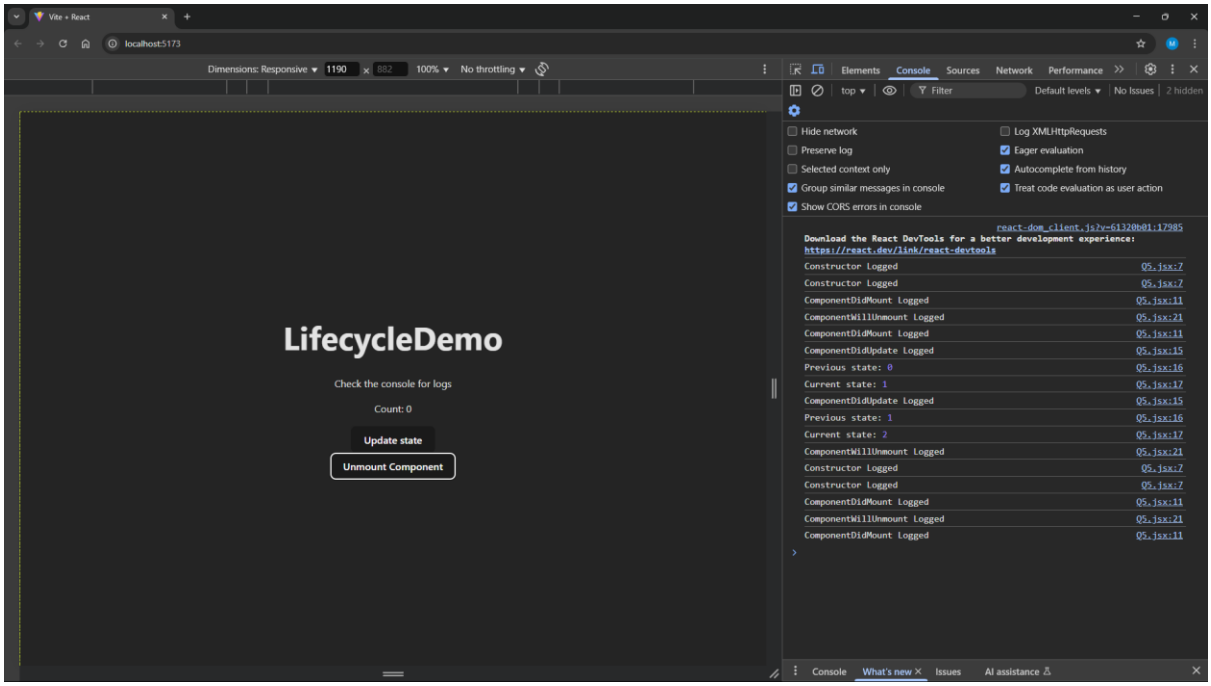
```
        return (
            <div style={{ textAlign: "center" }}>
                <h1>LifecycleDemo</h1>
                <p>Check the console for logs</p>
                <p>Count: {this.state.count}</p>
                <button onClick={this.incrementCount}>Update
state</button>
            </div>
        );
    }
}

export default LifeCycleDemo;
```

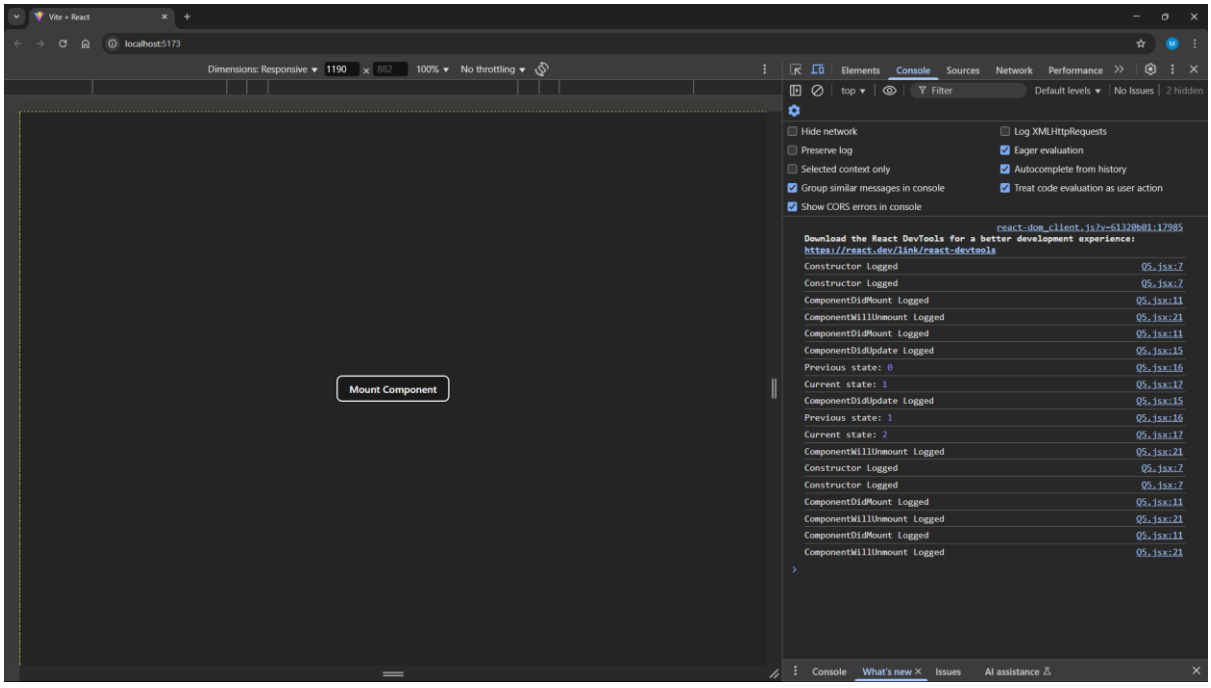
App.jsx

```
import { Component } from "react";
import reactLogo from "./assets/react.svg";
import viteLogo from "/vite.svg";
import "./App.css";
import LifeCycleDemo from "./components/Q5.jsx";

class App extends Component {
    state = { isMounted: true };

    toggleMount = () => {
        this.setState((prevState) => ({ isMounted:
!prevState.isMounted }));
    };

    render() {
        return (
```

```
            <div style={{ textAlign: "center" }}>
                {this.state.isMounted && <LifeCycleDemo />}
                <button onClick={this.toggleMount}>
                    {this.state.isMounted
                        ? "Unmount Component"
                        : "Mount Component"}
                </button>
            </div>
        );
    }
}

export default App;
```




Vite + React
localhost:5173
Dimensions: Responsive 1190 882 100% No throttling
LifecycleDemo
Check the console for logs
Count: 0
Update state
Unmount Component
Elements Console Sources Network Performance
top Filter Default levels No Issues 2 hidden
Hide network
Preserve log
Selected context only
Group similar messages in console
Show CORS errors in console
Log XMLHttpRequests
Eager evaluation
Autocomplete from history
Treat code evaluation as user action
react-dom_client.js?v=61320b01:17985
Download the React DevTools for a better development experience:
https://react.dev/link/react-devtools
Constructor Logged Q5.jsx:7
Constructor Logged Q5.jsx:7
ComponentDidMount Logged Q5.jsx:11
ComponentWillUnmount Logged Q5.jsx:21
ComponentDidMount Logged Q5.jsx:11
ComponentDidUpdate Logged Q5.jsx:15
Previous state: 0 Q5.jsx:16
Current state: 1 Q5.jsx:17
ComponentDidUpdate Logged Q5.jsx:15
Previous state: 1 Q5.jsx:16
Current state: 2 Q5.jsx:17
ComponentWillUnmount Logged Q5.jsx:21
Constructor Logged Q5.jsx:7
Constructor Logged Q5.jsx:7
ComponentDidMount Logged Q5.jsx:11
ComponentWillUnmount Logged Q5.jsx:21
ComponentDidMount Logged Q5.jsx:11
Console What's new Issues AI assistance


Vite + React
localhost:5173
Dimensions: Responsive 1190 882 100% No throttling
Mount Component
Elements Console Sources Network Performance
top Filter Default levels No Issues 2 hidden
Hide network
Preserve log
Selected context only
Group similar messages in console
Show CORS errors in console
Log XMLHttpRequests
Eager evaluation
Autocomplete from history
Treat code evaluation as user action
react-dom_client.js?v=61320b01:17985
Download the React DevTools for a better development experience:
https://react.dev/link/react-devtools
Constructor Logged Q5.jsx:7
Constructor Logged Q5.jsx:7
ComponentDidMount Logged Q5.jsx:11
ComponentWillUnmount Logged Q5.jsx:21
ComponentDidMount Logged Q5.jsx:11
ComponentDidUpdate Logged Q5.jsx:15
Previous state: 0 Q5.jsx:16
Current state: 1 Q5.jsx:17
ComponentDidUpdate Logged Q5.jsx:15
Previous state: 1 Q5.jsx:16
Current state: 2 Q5.jsx:17
ComponentWillUnmount Logged Q5.jsx:21
Constructor Logged Q5.jsx:7
Constructor Logged Q5.jsx:7
ComponentDidMount Logged Q5.jsx:11
ComponentWillUnmount Logged Q5.jsx:21
ComponentDidMount Logged Q5.jsx:11
ComponentWillUnmount Logged Q5.jsx:21
Console What's new Issues AI assistance

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.

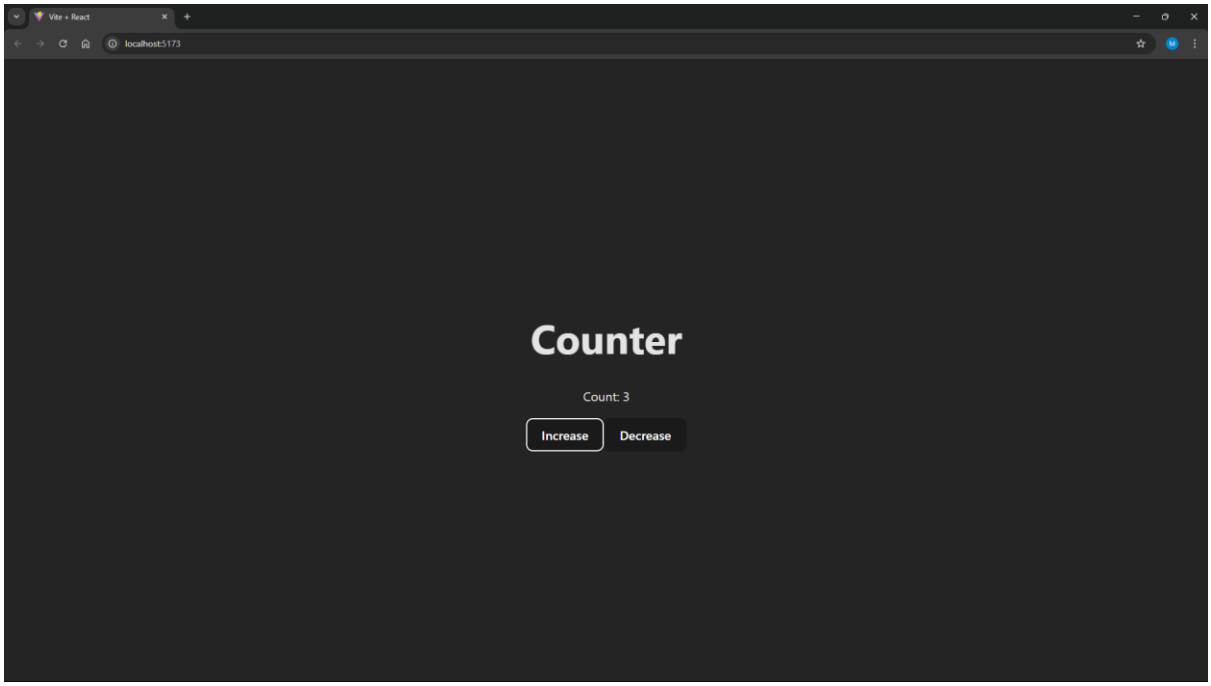
Q6.jsx

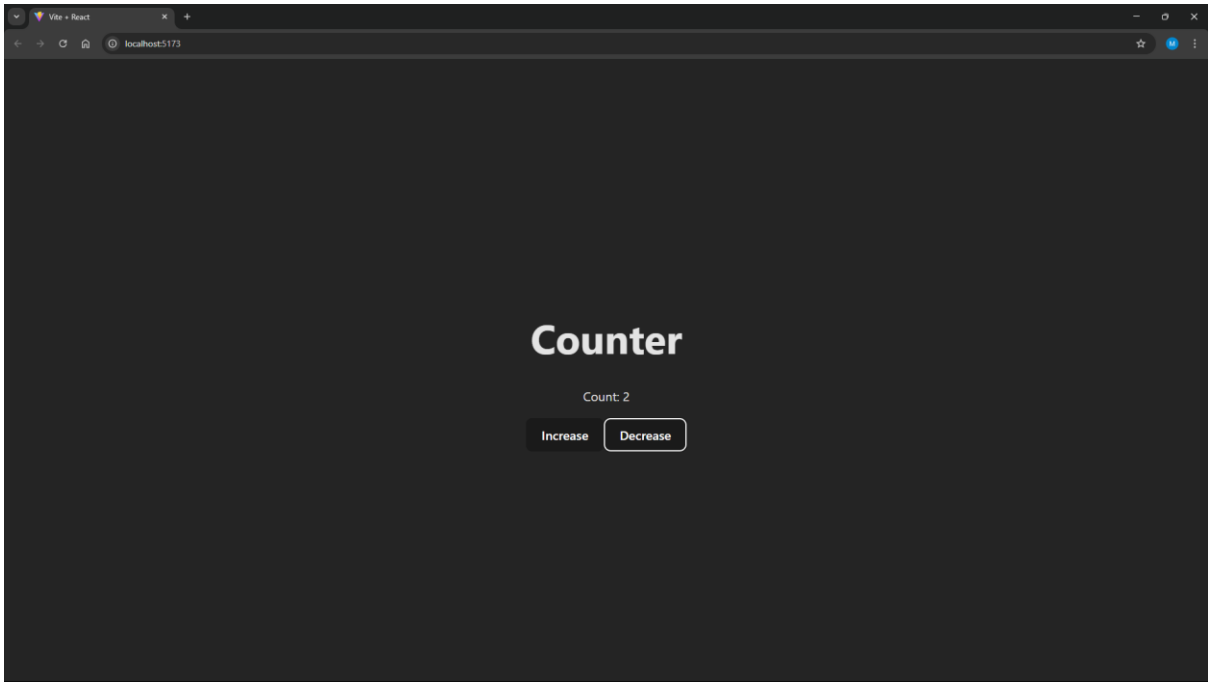
```
import React, { useState } from "react";

function Counter() {
    const [count, setCount] = useState(0);
    return (
        <div style={{ textAlign: "center" }}>
            <h1>Counter</h1>
            <p>Count: {count}</p>
            <button onClick={() => setCount(count +
1)}>Increase</button>
            <button onClick={() => setCount(count -
1)}>Decrease</button>
        </div>
    );
}

export default Counter;
```

Q6Alt.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 Counter() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });

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

export default Counter;
```





7) Effect Hooks (useEffect):

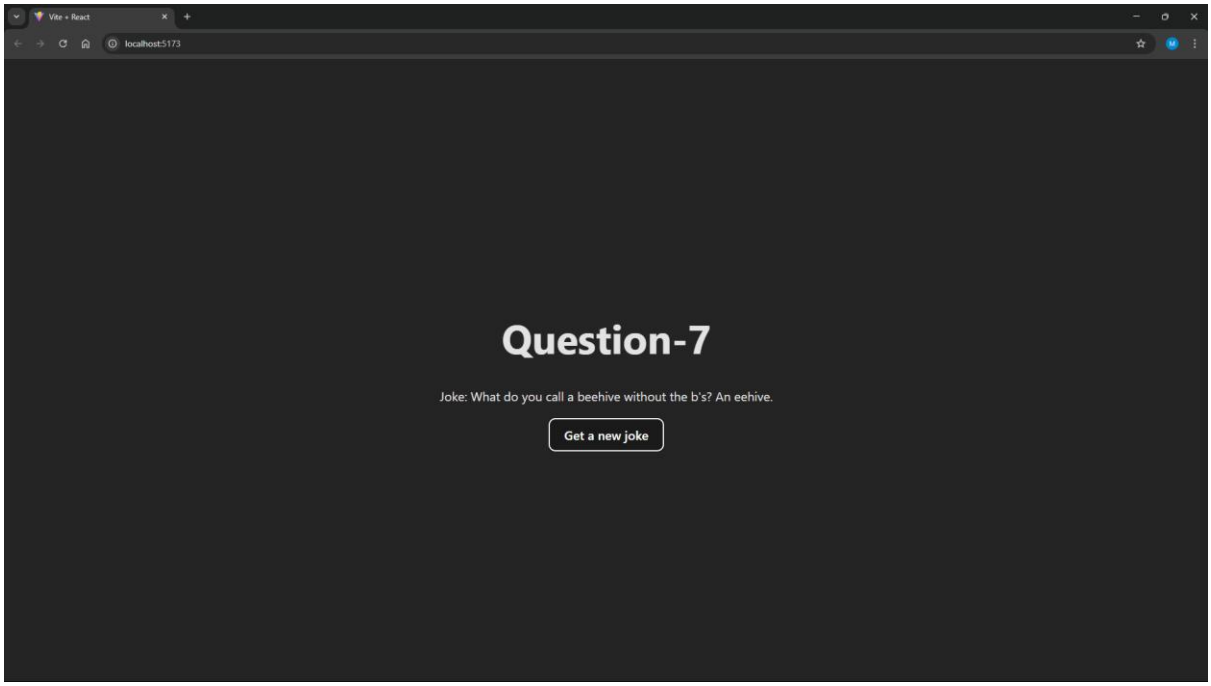
• Develop a React component that fetches and displays a random joke from an API when the component mounts.

• Add functionality to refresh the joke when a button is clicked

Q7.jsx

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

```
function JokeGen() {
    const [joke, setJoke] = useState("");
    const fetchJoke = async () => {
        try {
            const response = await
fetch("https://icanhazdadjoke.com/", {
                headers: { Accept: "application/json" },
            });
            const data = await response.json();
            setJoke(data.joke);
        } catch (error) {
            console.error("Error fetching joke:", error);
        }
    };
    useEffect(() => {
        fetchJoke();
    }, []);

    return (
        <div style={{ textAlign: "center" }}>
            <h1>Question-7</h1>
            <p>Joke: {joke}</p>
            <button onClick={() => fetchJoke()}>Get a new
joke</button>
        </div>
    );
}
export default JokeGen;
```



8. Ref Hooks (useRef):

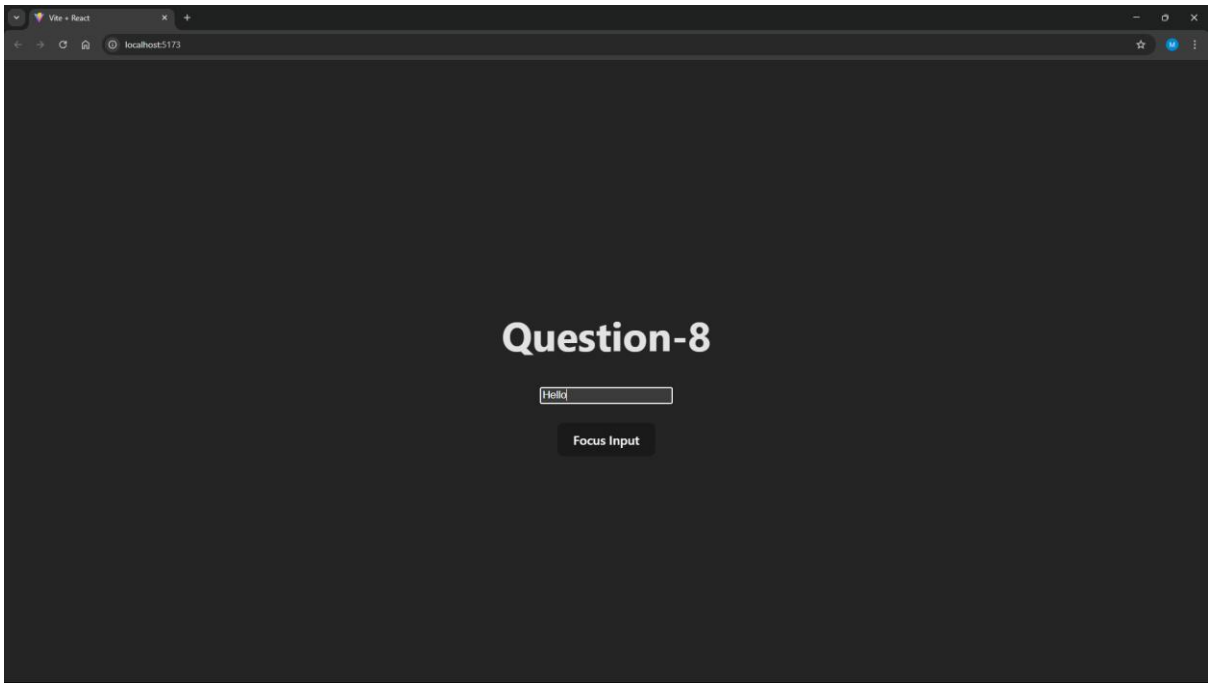
• Build a simple form with an input field and a button.

• When the button is clicked, the input field should automatically get focused using the useRef() hook.

Q8.jsx

```
import React, { useRef } from "react";
function Q8() {
    const inputRef = useRef(0);
    const handleFocus = () => {
        if (inputRef.current) {
            inputRef.current.focus();
        }
    };
    return (
        <div style={{ textAlign: "center" }}>
            <h1>Question-8</h1>
            <input type="text" ref={inputRef} placeholder="Input" />
```

```
            <br />
            <br />
            <button onClick={handleFocus}>Focus Input</button>
        </div>
    );
}
export default Q8;
```



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.

App.jsx

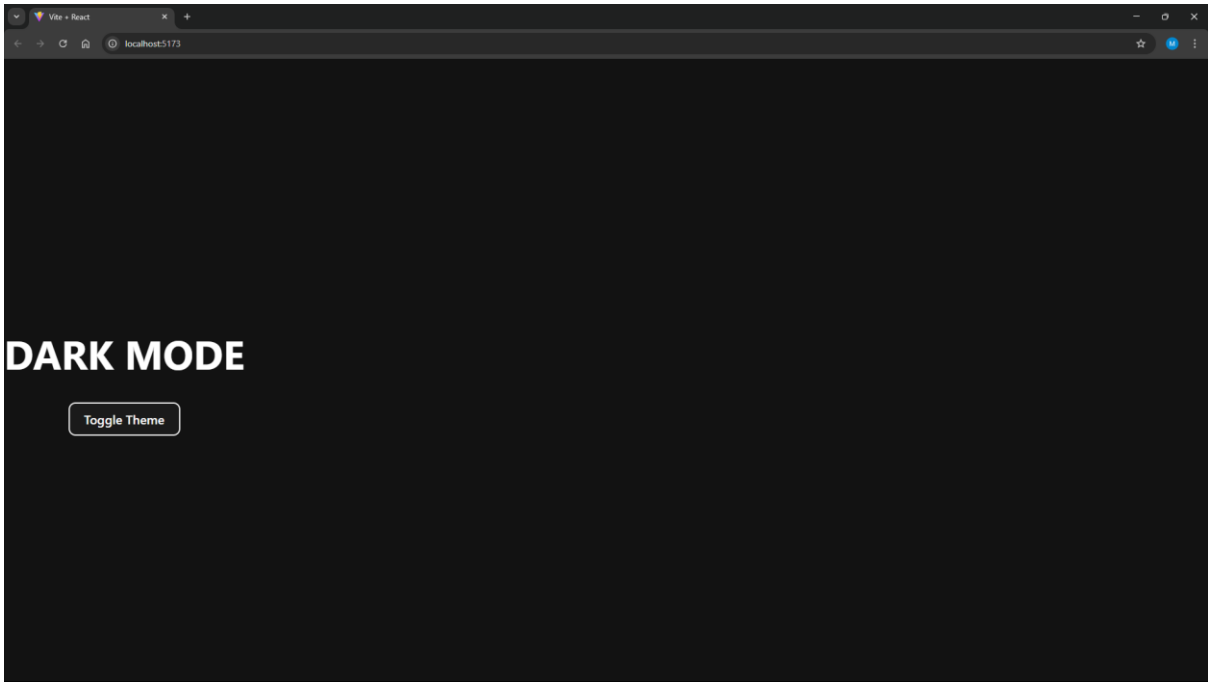
```
import React, { useState, createContext, useContext, useEffect }
from "react";


const ThemeContext = createContext();


function ThemeProvider({ children }) {
    const [theme, setTheme] = useState("light");
```

```
    const toggleTheme = () =>
        setTheme((prev) => (prev === "light" ? "dark" : "light"));

    useEffect(() => {
        document.body.style.backgroundColor =
            theme === "light" ? "#fff" : "#121212";
        document.body.style.color = theme === "light" ? "#000" :
"#fff";
    }, [theme]);

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

const ThemedApp = () => {
    const { theme, toggleTheme } = useContext(ThemeContext);
    return (
        <div style={{ textAlign: "center" }}>
            <h1>{theme.toUpperCase()} MODE</h1>
            <button onClick={toggleTheme}>Toggle Theme</button>
        </div>
    );
};

const App = () => (
    <ThemeProvider>
        <ThemedApp />
    </ThemeProvider>
```

```
);

export default App;
```

LIGHT MODE

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.**

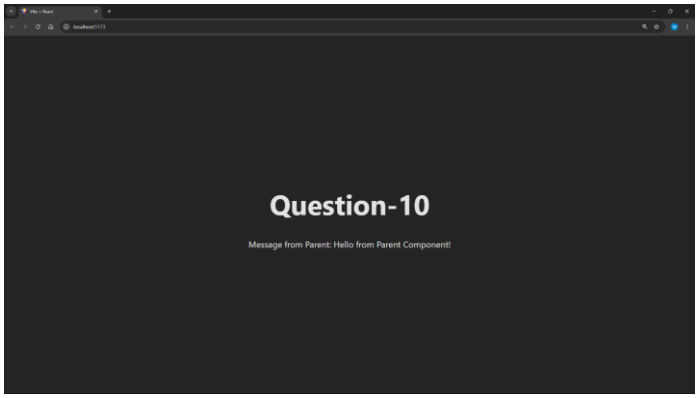
Q10_child.jsx

```
import React from "react";
export default function Q10_Child({ msg }) {
    return (
        <div style={{ textAlign: "center" }}>
            <h1>Question-10</h1>
            <p>Message from Parent: {msg}</p>
        </div>
    );
}
```

Q10_parent.jsx

```
import React from "react";
import Child from "./Q10_child";
export default function Q10() {
    const message = "Hello from Parent Component!";
    return <Child msg={message} />;
}
```

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.**

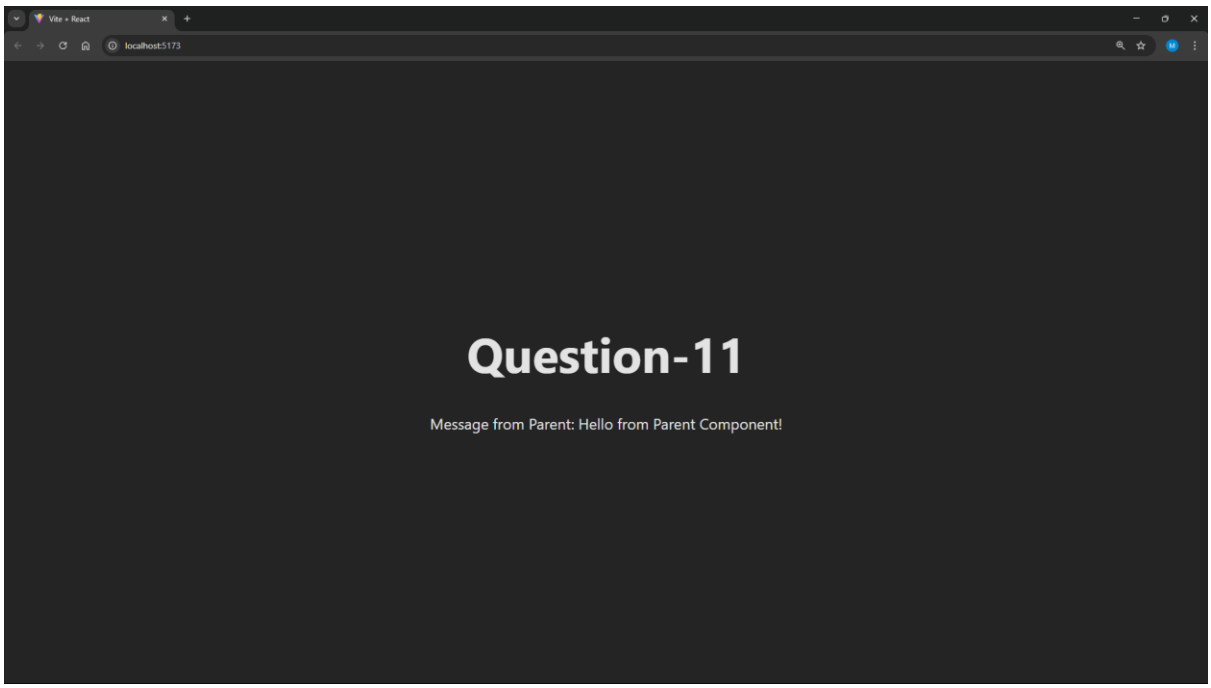
Q11_child.jsx

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

function Q11_Child({ msg }) {
    return (
        <div style={{ textAlign: "center" }}>
            <h1>Question-11</h1>
            <p>Message from Parent: {msg}</p>
        </div>
    );
}

Q11_Child.propTypes = {
    msg: PropTypes.string.isRequired,
};

export default Q11_Child;
```

Q11_parent.jsx

```
import React from "react";
import Child from "./Q10_child";
export default function Q10() {
    const message = "Hello from Parent Component!";
    return <Child msg={message} />;
}
```

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.

Q12.jsx

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


export default function Q12() {
    const [name, setName] = useState("");
```

```
    const [email, setEmail] = useState("");


    const nameRef = useRef();
    const emailRef = useRef();
    const [submittedData, setSubmittedData] = useState({ name: "",
email: "" });


    return (
        <div style={{ textAlign: "center" }}>
            <h2>Form with useState (Live Update)</h2>
            <form
                onSubmit={(e) => {
                    e.preventDefault();
                    console.log("Form Submitted (useState)");
                    console.log("Name:", name);
                    console.log("Email:", email);
                }}
            >
                <input
                    type="text"
                    placeholder="Name"
                    onChange={(e) => setName(e.target.value)}
                />
                <br />
                <br />
                <input
                    type="email"
                    placeholder="Email"
                    onChange={(e) => setEmail(e.target.value)}
                />
                <br />
```

```
                <br />
                <button type="submit">Submit</button>
            </form>
            <p>Name: {name}</p>
            <p>Email: {email}</p>

            <hr style={{ margin: "40px 0" }} />

            <h2>Form with useRef (Values on Submit)</h2>
            <form
                onSubmit={(e) => {
                    e.preventDefault();
                    const nameVal = nameRef.current.value;
                    const emailVal = emailRef.current.value;
                    setSubmittedData({ name: nameVal, email:
emailVal });
                    console.log("Form Submitted (useRef)");
                    console.log("Name:", nameVal);
                    console.log("Email:", emailVal);
                }}
            >
                <input type="text" placeholder="Name" ref={nameRef}
/>
                <br />
                <br />
                <input type="email" placeholder="Email"
ref={emailRef} />
                <br />
                <br />
                <button type="submit">Submit</button>
            </form>
```

```
            <p>Name: {submittedData.name}</p>
            <p>Email: {submittedData.email}</p>
        </div>
    );
}
```

Output: