ReactJS Part – 3 – Lab Assignment

1. Create a React Component using JSX that Displays a Greeting Message

✅ **Concepts Covered**: JSX, Props

🔹 **Task**:

- Create a Greeting component that accepts a name prop and displays a greeting message like **"Hello, John!"**.
- Render this component inside App.js with different names.

🔹 **Hint**:

Use JSX to return elements dynamically based on props:

```
function Greeting({ name }) {
  return <h1>Hello, {name}!</h1>;
}
```

---

2. Build a Counter Component using a Constructor in a Class Component

✅ **Concepts Covered**: React Constructors, setState

🔹 **Task**:

- Create a **class component** Counter that initializes count = 0.
- Provide + and - buttons to increase/decrease the count.

🔹 **Hint**:

- Use a **constructor** to initialize the state:

```
constructor(props) {
  super(props);
  this.state = { count: 0 };
}
```

- Use this.setState() to update the count.

---

3. Convert a Class Component into a Functional Component using Hooks

✅ **Concepts Covered**: useState, Functional Components

🔹 **Task**:

- Take an existing **class-based counter component** and rewrite it using a functional component with useState.

🔹 **Hint**:

Replace this.state with:

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

Replace this.setState() with:

setCount(count + 1);

---

4. Create a Dynamic List Rendering Component using .map()

✅ **Concepts Covered**: JSX, Lists, Props

🔹 **Task**:

- Create a UserList component that takes an array of names as a prop and **renders each name as a list item** (<li>).

🔹 **Hint**:

Use .map() inside JSX:

<ul>

{users.map((user, index) => <li key={index}>{user})}

*
*

---

5. Implement a Simple Theme Switcher using useState

✅ **Concepts Covered**: Hooks, State Management

🔹 **Task**:

- Create a **dark/light mode toggle** using a button.
- Store the theme (light or dark) in a state variable and update styles dynamically.

🔹 **Hint**:

Use a **boolean state** and toggle it:

const [theme, setTheme] = useState("light");

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

---

6. Use useEffect to Fetch and Display Data from an API

✅ **Concepts Covered**: useEffect, Fetch API, Lifecycle Hooks

🔹 **Task**:

- Fetch **user data** from https://jsonplaceholder.typicode.com/users when the component mounts.
- Display **name, email, and website** of users.

🔹 **Hint**:

Use useEffect to fetch data only **on mount**:

useEffect(() => {

fetch("https://jsonplaceholder.typicode.com/users")

.then(response => response.json())

.then(data => setUsers(data));

}, []);

---

7. Implement a Simple Form Handling Component using useState

✅ **Concepts Covered**: Form Handling, State Management

🔹 **Task**:

- Create a form with **name and email inputs**.
- On form submission, display the entered data.

🔹 **Hint**:

Use controlled inputs:

const [name, setName] = useState("");

<input type="text" value={name} onChange={(e) => setName(e.target.value)} />

---

8. Create a Component that Uses useEffect to Track Window Resize

✅ **Concepts Covered**: Hooks, Event Listeners

🔹 **Task**:

- Track **window width** and display it in the component.
- Update width dynamically when resized.

🔹 **Hint**:

Use useEffect with an event listener:

useEffect(() => {

const handleResize = () => setWidth(window.innerWidth);

window.addEventListener("resize", handleResize);

return () => window.removeEventListener("resize", handleResize);

}, []);

---

9. Create a Parent-Child Component Communication using Props and Callbacks

✅ **Concepts Covered**: Props, Callback Functions

🔹 **Task**:

- Create a Parent component that has a button.
- The button should **update the state in the Child component**.

🔹 **Hint**:
Pass a function as a prop:

```
function Parent() {
  const [message, setMessage] = useState("Hello");

  return <Child updateMessage={() => setMessage("Updated!")} />;
}

function Child({ updateMessage }) {
  return <button onClick={updateMessage}>Change Parent Message</button>;
}
```

---

10. Implement a StopWatch Using useState and useEffect

✅ **Concepts Covered**: State, useEffect, Timers

🔹 **Task**:

- Create a **StopWatch** with **Start, Stop, and Reset** buttons.
- The stopwatch should count **seconds and minutes**.

🔹 **Hint**:
Use setInterval() inside useEffect:

```
useEffect(() => {
  if (isRunning) {
    const interval = setInterval(() => setTime(time + 1), 1000);
    return () => clearInterval(interval);
  }
}, [isRunning, time]);
```