Experiment 5 : Study and Implementation of React.js

Problem Statement 1:

Basics of ReactJS

1. What is React and what problem does it solve?

Answer:
React is a JavaScript library developed by Facebook for building user interfaces, especially single-page applications (SPAs). It allows developers to create reusable UI components and efficiently update the UI using a virtual DOM. React solves issues like performance optimization and complex UI state management by enabling declarative programming and component-based architecture.

---

2. What are React components and how are they used?

Answer:
React components are the building blocks of a React application. They are reusable, independent pieces of UI that manage their own logic and rendering. Components can be classified into:

- **Functional Components**: Defined as functions and use hooks for state and lifecycle management.
- **Class Components**: Defined as ES6 classes and use lifecycle methods.

Usage Example (Functional Component):

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

Components are used in JSX like HTML elements:

```
<Greeting name="John" />
```

---

3. What is JSX in React?

Answer:
JSX (JavaScript XML) is a syntax extension for JavaScript that allows writing HTML-like code within JavaScript. JSX makes it easier to describe UI structures while leveraging JavaScript's power.

Example:

```
const element = <h1>Hello, World!</h1>;
```

JSX is compiled into standard JavaScript using Babel:

```
const element = React.createElement("h1", null, "Hello, World!");
```

---

4. What are props in React and how do they differ from state?

Answer:
Props (short for "properties") are read-only data passed from a parent component to a child component. They help make components reusable by allowing dynamic data to be passed.

Example of props:

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

Props vs. State:

- **Props**: Immutable, passed from parent to child, and cannot be modified by the child component.
- **State**: Managed within the component and can be updated using setState (for class components) or hooks like useState (for functional components).

---

5. What is state in React and how does it work?

Answer:
State is an object that stores component-specific data and determines how the component behaves and renders. It allows components to be dynamic and interactive.

Example using useState (Functional Component):

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


function Counter() {
```

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


  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

How it works:

- The useState hook initializes count to 0.
- setCount updates count, causing the component to re-render with the new value.

---

6. What are React lifecycle methods, and why are they important?

Answer:
Lifecycle methods control the behavior of a class component at different stages (mounting, updating, unmounting). They are useful for performing actions like data fetching and event subscriptions.

Key lifecycle methods:

1. **Mounting:** componentDidMount() – Runs after the component is inserted into the DOM.
2. **Updating:** componentDidUpdate() – Runs after a component's state or props update.
3. **Unmounting:** componentWillUnmount() – Runs just before a component is removed from the DOM.

Example (Class Component):

```
class Example extends React.Component {
  componentDidMount() {
    console.log("Component Mounted");
  }
```

```
  componentWillUnmount() {
    console.log("Component Unmounted");
  }
  render() {
    return <h1>Hello, React!</h1>;
  }
}
```

Additional React Concepts

7. Event Handling in React

Answer:
React handles events similarly to DOM events but with JSX syntax and camelCase naming.

Example:

```
function ButtonClick() {
  function handleClick() {
    alert("Button Clicked!");
  }
  return <button onClick={handleClick}>Click Me</button>;
}
```

---

8. Conditional Rendering in React

Answer:
Conditional rendering allows components to render differently based on state or props.

Example using ternary operator:

```
function UserStatus({ isLoggedIn }) {
  return <h1>{isLoggedIn ? "Welcome back!" : "Please log in"}</h1>;
}
```

---

9. Lists and Keys in React

Answer:
React uses keys to identify list items uniquely, optimizing rendering performance.

Example:

```
const names = ["Alice", "Bob", "Charlie"];

return (
  <ul>
    {names.map((name, index) => (
      <li key={index}>{name}</li>
    ))}
  </ul>
);
```

---

10. Forms in React

Answer:
React manages form input using controlled components, where state holds the input value.

Example:

```
function FormExample() {
  const [input, setInput] = useState("");

  return (
    <form>
      <input type="text" value={input} onChange={(e) => setInput(e.target.value)} />
      <p>You typed: {input}</p>
    </form>
  );
}
```

---

11. Hooks in React

Answer:
Hooks allow functional components to use state and lifecycle features.

- **useState:** Manages component state.
- **useEffect:** Handles side effects like data fetching.

Example:

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


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


  useEffect(() => {
    console.log("Component updated!");
  }, [count]);


  return <button onClick={() => setCount(count + 1)}>Count: {count}</button>;
}
```

---

12. React Router

Answer:
React Router enables navigation between different pages in a React app.

Example:

```
import { BrowserRouter, Route, Routes, Link } from "react-router-dom";


function Home() {
  return <h1>Home</h1>;
}


function About() {
  return <h1>About</h1>;
```

```
}


function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}
```

---

13. State Management in React

Answer:
State management refers to handling shared state efficiently. Solutions include:

- **React Context API** – Suitable for small-scale applications.
- **Redux, Recoil, Zustand** – Suitable for large-scale applications.

---

14. React Context API

Answer:
The Context API allows passing data globally without prop drilling.

Example:

```
const ThemeContext = React.createContext();
```

```
function App() {
  return (
    <ThemeContext.Provider value="dark">
      <ThemedComponent />
    </ThemeContext.Provider>
  );
}

function ThemedComponent() {
  const theme = useContext(ThemeContext);
  return <p>Current theme: {theme}</p>;
}
```

---

15. How can you optimize the performance of a React application?

Answer:

1. **Using React.memo** to prevent unnecessary re-renders.
2. **Lazy loading components** using React.lazy.
3. **Optimizing state updates** by minimizing re-renders.
4. **Using useCallback and useMemo** to cache functions and computations.
5. **Code splitting** to load only necessary parts of the app.
6. **Using virtualization** (e.g., react-window) for rendering large lists efficiently.

Problem statement 2: