**REACT TOPICS SO FAR**

**1. Single Page Application (SPA)**

* **Definition**: An SPA is a web application that loads a single HTML page and dynamically updates content based on user interactions without reloading the entire page.
* **Advantages**: Faster user experience, smoother transitions, and reduced server load.
* **Implementation**: React uses a virtual DOM and routing (with libraries like React Router) to achieve SPA behavior.

**Example**:

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import Home from './Home';

import About from './About';

const App = () => (

  <Router>

    <Routes>

      <Route path="/" element={<Home />} />

      <Route path="/about" element={<About />} />

    </Routes>

  </Router>

);

**2. Components**

* **Definition**: Components are reusable building blocks of a React application that encapsulate UI and logic.
* **Types**: Functional and Class components.
* **Usage**: Components can be nested, composed, and reused across the application.

**Example**:

// Functional Component

const MyComponent = () => <div>Hello, World!</div>;

// Class Component

class MyComponent extends React.Component {

  render() {

    return <div>Hello, World!</div>;

  }

}

**3. State**

* **Definition**: State is an object that stores data that changes over time within a component.
* **Usage**: State is mutable and affects component rendering.
* **Management**: In functional components, state is managed using the useState hook. In class components, state is managed using this.state and this.setState.

**Example**:

// Functional Component with useState

const Counter = () => {

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

  return (

    <div>

      <p>Count: {count}</p>

      <button onClick={() => setCount(count + 1)}>Increment</button>

    </div>

  );

};

**4. Props**

* **Definition**: Props (short for properties) are used to pass data from parent components to child components. Props are read-only.
* **Usage**: Props enable component reusability and dynamic rendering based on parent data.

**Example**:

// Parent Component

const Parent = () => <Child message="Hello, World!" />;

// Child Component

const Child = (props) => <div>{props.message}</div>;

**5. Types of Components**

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

**Example**:

// Functional Component

const FunctionalComponent = () => <div>Functional Component</div>;

// Class Component

class ClassComponent extends React.Component {

  render() {

    return <div>Class Component</div>;

  }

}

**6. Hooks**

* **Definition**: Hooks are functions that allow you to use state and other React features in functional components.
* **Common Hooks**: useState, useEffect, useContext, useReducer, useRef.

**Example**:

// useEffect Hook

const ExampleComponent = () => {

  useEffect(() => {

    console.log('Component mounted');

    return () => console.log('Component unmounted');

  }, []);

  return <div>Effect Example</div>;

};

**7. Life of Components (Component Loading and Unloading)**

* **Mounting**: The phase where a component is being created and inserted into the DOM. Lifecycle methods/hooks include componentDidMount (class) and useEffect (functional).
* **Updating**: The phase where a component is being re-rendered due to state/props changes. Lifecycle methods/hooks include componentDidUpdate (class) and useEffect (functional).
* **Unmounting**: The phase where a component is being removed from the DOM. Lifecycle methods/hooks include componentWillUnmount (class) and cleanup function in useEffect (functional).

**Example**:

// Functional Component with useEffect

const LifecycleExample = () => {

  useEffect(() => {

    console.log('Mounted');

    return () => console.log('Unmounted');

  }, []);

  return <div>Lifecycle Example</div>;

};

**8. Form Handling**

* **Definition**: Managing user inputs and form submissions in React. Use controlled components where form data is managed by React state.
* **Usage**: Track and validate form inputs using state and handle form submission events.

**Example**:

const FormComponent = () => {

  const [inputValue, setInputValue] = useState('');

  const handleChange = (e) => setInputValue(e.target.value);

  const handleSubmit = (e) => {

    e.preventDefault();

    console.log('Submitted value:', inputValue);

  };

  return (

    <form onSubmit={handleSubmit}>

      <input type="text" value={inputValue} onChange={handleChange} />

      <button type="submit">Submit</button>

    </form>

  );

};

**9. Events**

* **Definition**: Events are actions or occurrences handled by React to trigger functions or update state.
* **Usage**: Attach event handlers to elements to respond to user actions (e.g., clicks, form submissions).

**Example**:

const EventExample = () => {

  const handleClick = () => alert('Button clicked!');

  return <button onClick={handleClick}>Click Me</button>;

};

**10. List in React**

* **Definition**: Rendering multiple items from an array. React uses map to iterate and generate elements for each item.
* **Usage**: Ensure each item has a unique key prop for efficient updates.

**Example**:

const ListExample = () => {

  const items = ['Item 1', 'Item 2', 'Item 3'];

  return (

    <ul>

      {items.map((item, index) => (

        <li key={index}>{item}</li>

      ))}

    </ul>

  );

};

**11. Conditional Rendering**

* **Definition**: Display different UI elements based on certain conditions using JavaScript operators.
* **Usage**: Use if, &&, or ternary operators to conditionally render components or elements.

**Example**:

const ConditionalRendering = ({ isLoggedIn }) => (

  <div>

    {isLoggedIn ? <p>Welcome back!</p> : <p>Please log in.</p>}

  </div>

);

**12. Routing**

* **Definition**: Managing navigation between different views or pages in a React application. Achieved using React Router.
* **Usage**: Define routes and link navigation between components.

**Example**:

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import Home from './Home';

import About from './About';

const App = () => (

  <Router>

    <Routes>

      <Route path="/" element={<Home />} />

      <Route path="/about" element={<About />} />

    </Routes>

  </Router>

);

**13. Protected Route**

* **Definition**: Restrict access to certain routes based on authentication or authorization. Redirect unauthenticated users to a login page.
* **Usage**: Implement protected routes using conditional rendering and React Router.

**Example**:

javascript

Copy code

import { Navigate } from 'react-router-dom';

const ProtectedRoute = ({ element, isAuthenticated }) => (

  isAuthenticated ? element : <Navigate to="/login" />

);

// Usage in Routes

<Routes>

  <Route path="/protected" element={<ProtectedRoute element={<ProtectedPage />} isAuthenticated={isAuthenticated} />} />

</Routes>