
🔷 React JS Detailed Notes and Summary

✅ 1. Introduction to React

- **React JS** is an open-source **JavaScript library** developed by **Facebook** for building **user interfaces**, especially **single-page applications (SPAs)**.
- Focused on **UI components** — React breaks the UI into reusable pieces.

✅ 2. JSX (JavaScript XML)

- **JSX** is a **syntax extension** for JavaScript that looks like **HTML**.
- Allows mixing **HTML-like tags** within JavaScript code.
- React transforms JSX into React.createElement() calls behind the scenes.

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

✅ 3. React Core Features

- **Virtual DOM**: A lightweight copy of the real DOM, used for performance optimization.
- **Component-based architecture**: UI is divided into small, reusable components.
- **Unidirectional Data Flow**: Data flows from **parent to child** using props.
- ⚠️ React does **not** support **two-way binding** by default like Angular does.

✅ 4. Component Lifecycle (Class-based)

- React class components have a **lifecycle** with specific methods:
  - componentDidMount(): Called after the component is added to the DOM. Ideal for **data fetching**.
  - componentDidUpdate(): Called after updates to props or state.
  - componentWillUnmount(): Cleanup before the component is removed.
  - componentDidCatch(): Handles errors in child components.

- ❌ componentWillMount(), componentWillUpdate(), and componentWillReceiveProps() are **deprecated**.

✅ 5. State and Props

- **State**:
  - Data **internal** to a component.
  - Modified using this.setState() in class components or useState() in functional components.
- **Props**:
  - Read-only data **passed from parent to child** components.
  - Used to customize components.

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

✅ 6. ReactDOM and Rendering

- The ReactDOM.render() method is used to **render** a React component into the DOM.

```
ReactDOM.render(<App />, document.getElementById('root'));
```

✅ 7. Reusable Components

- React components (either **function** or **class**) are designed to be **reused** across the application.

✅ 8. Keys in Lists

- **Keys** are unique identifiers used in rendering lists.
- Helps React in **reconciliation** to track which items changed or moved.

```
{items.map(item => <li key={item.id}>{item.name}</li>)}
```

✅ 9. Conditional Rendering

- Achieved using:
  - **Ternary operators**
  - **Logical AND (&&)**
  - if-else inside the render function

```
{isLoggedIn ? <Logout /> : <Login />}
```

✅ 10. PropTypes

- PropTypes are used for **runtime type-checking** of props passed to components.

```
ComponentName.propTypes = {
  name: PropTypes.string,
};
```

✅ 11. State Management

- **useState**: Hook to add local state to function components.
- **Redux**: External library used for global state management.
- **Context API**: Built-in React feature to pass data without prop drilling.

✅ 12. Side Effects in React

- Use useEffect() for performing **side effects** like:
  - Data fetching
  - Event listeners
  - Subscriptions

```
useEffect(() => {
  // logic here
}, []);
```

✅ 13. PureComponent and memo

- React.PureComponent performs a **shallow comparison** of props/state for optimization.
- React.memo() is used to **memoize functional components**.

✅ 14. Refs and useRef

- Refs are used to **access DOM elements or component instances** directly.
- createRef() in class components and useRef() in functional components.

```
const inputRef = useRef();
```

✅ 15. React Router

- **React Router** is used to handle **client-side routing** in single-page applications.
- BrowserRouter, Route, Link, and useNavigate are common components/hooks.

✅ 16. React Hooks

- Hooks are functions that let you use state and lifecycle methods in **functional components**:
  - useState() – local state
  - useEffect() – side effects
  - useContext() – access context
  - useReducer() – complex state logic
  - useMemo() – memoizes **expensive computations**
  - useCallback() – memoizes **functions** to prevent unnecessary re-creation
  - useRef() – access DOM nodes or persist values
  - Custom Hooks can also be created

✅ 17. useContext

- Used to access **context values** in a component tree without passing props manually.

```
const theme = useContext(ThemeContext);
```

✅ 18. useMemo vs useCallback

- useMemo(): Returns **memoized result** of a computation.
- useCallback(): Returns **memoized function**.

✅ 19. useReducer

- A hook for **managing complex state transitions**, alternative to useState().

✅ 20. React Fragments

- Used to group **multiple children** without creating extra DOM nodes.

```
<>
 <h1>Hello</h1>
 <p>World</p>
</>
```

✅ 21. React Portals

- Used to **render elements outside** the DOM hierarchy of the parent component.
- Useful for modals, tooltips.

✅ 22. forwardRef

- Used to forward **ref** from a parent component to a child component.

✅ 23. createRef

- Used in class components to **create a reference** to a DOM node.

✅ 24. Styling in React

- **Valid styling methods**:
  - Inline styles: style={{ color: 'red' }}
  - External CSS files
  - CSS Modules
  - styled-components library
- ❌ <style> tags inside JSX are not recommended.

✅ 25. dangerouslySetInnerHTML

- Used to insert raw HTML directly into a component.
- ⚠️ Can lead to **XSS attacks** if not handled carefully.

```
<div dangerouslySetInnerHTML={{ __html: '<h1>Hello</h1>' }} />
```

✅ 26. React Events

- React uses **Synthetic Events** which work identically across all browsers.

```
<button onClick={handleClick}>Click</button>
```

✅ 27. Error Handling

- **componentDidCatch()**: Lifecycle method to catch errors in child components.
- Used with **Error Boundaries**.

✅ 28. Data Flow

- **Parent → Child**: via props.
- **Child → Parent**: by **callback functions** passed down as props.

✅ 29. React Context

- React's **Context API** allows you to pass data deeply without manually passing props at every level.

- Not a complete replacement for Redux in complex apps, but sufficient for light state sharing.

✅ 30. Optimization Techniques

- Use:
  - React.memo()
  - useMemo()
  - useCallback()
  - **Avoid unnecessary re-renders**
  - Use **keys** in lists properly
