React Mastery Path

Phase 1: React Fundamentals

1. JSX (JavaScript XML)

- **What:** A syntax extension for JavaScript that looks like HTML and is used to describe UI structure in React.
- **Why Important:** Enables declarative UI creation and readable component structure.
- **Under the Hood:** JSX is transpiled to `React.createElement()` by Babel, which builds a virtual DOM tree.
- **Used in Real Apps:** Defining component UI layout.
- **Example:**
- `const Welcome = () => <h1>Hello, SafeWeb AI User!</h1>;`
- **Hands-On Task:** Create a `Header`, `Footer`, and `Landing` JSX layout.
- **Interview Question:** What happens to JSX during the build process?
- **SafeWeb AI Use:** Structure landing pages, hero sections, or onboarding screens.

2. Components (Functional & Class)

- **What:** Independent, reusable pieces of UI.
- **Why Important:** Encapsulates logic and markup for reuse and readability.
- **Under the Hood:** Each component maintains its own lifecycle and props/context handling.
- **Used in Real Apps:** Every UI block (Navbar, Card, Form) is a component.
- **Example:**
- `function Card(props) {`
- `  return <div className="card">{props.content}</div>;`
- `}`
- **Hands-On Task:** Build a `FeatureCard` component for highlighting SafeWeb AI features.
- **Interview Question:** Difference between functional and class components?
- **SafeWeb AI Use:** Reusable UI units like user reviews, feature highlights.

3. Props

- **What:** Read-only data passed from parent to child component.
- **Why Important:** Enables communication between components.
- **Under the Hood:** Props are passed as parameters to functions.
- **Used in Real Apps:** Passing user info, config, styles.
- **Example:**
- `const Greeting = ({ name }) => <h2>Hello, {name}</h2>;`
- **Hands-On Task:** Pass site scan results from parent to child.

- **Interview Question:** Can a child modify props?
- **SafeWeb AI Use:** Show user-specific security reports.

4. State

- **What:** A built-in object used to hold dynamic data.
- **Why Important:** Controls component behavior over time.
- **Under the Hood:** React uses `useState` to store and re-render components on update.
- **Used in Real Apps:** Handling form inputs, toggles, modals.
- **Example:**
- `const [email, setEmail] = useState('');`
- **Hands-On Task:** Build a login form managing email/password state.
- **Interview Question:** How does state cause a re-render?
- **SafeWeb AI Use:** Manage user input and scan configuration.

5. Event Handling

- **What:** Listening and responding to user events.
- **Why Important:** Powers interactivity.
- **Under the Hood:** Synthetic event system based on native events.
- **Used in Real Apps:** Form submission, buttons, toggles.
- **Example:**
- `<button onClick={handleScan}>Start Scan</button>`
- **Hands-On Task:** Add a “Start Scan” button with click handler.
- **Interview Question:** What is a synthetic event?
- **SafeWeb AI Use:** Trigger a website scan.

6. Conditional Rendering

- **What:** Rendering different UI based on conditions.
- **Why Important:** Enhances UX by showing relevant data.
- **Under the Hood:** Simple JavaScript logic (ternaries, `&&`, `if`).
- **Used in Real Apps:** Loading screens, empty states, error messages.
- **Example:**
- `{isLoading ? <Loader /> : <ScanResult />}`
- **Hands-On Task:** Show a loader while scanning.
- **Interview Question:** Ways to implement conditional rendering?
- **SafeWeb AI Use:** Show different UI during scan progress.

7. Lists and Keys

- **What:** Rendering multiple elements using `map()` with unique `key` props.
- **Why Important:** Efficient rendering and reconciliation.
- **Under the Hood:** Keys help React identify changed, added, or removed items.
- **Used in Real Apps:** Listing scan results, history logs.
- **Example:**

- `{urls.map((url) => <li key={url}>{url}</li>)}`
- **Hands-On Task:** Display a list of scanned URLs.
- **Interview Question:** Why are keys important in lists?
- **SafeWeb AI Use:** Show scanned website list.

8. useEffect

- **What:** Side effects like data fetching, subscriptions.
- **Why Important:** Handles operations outside render.
- **Under the Hood:** Scheduled after paint, cleanup via return function.
- **Used in Real Apps:** Fetching scan results, triggering alerts.
- **Example:**
- `useEffect(() => { fetchData(); }, []);`
- **Hands-On Task:** Fetch dummy scan data on load.
- **Interview Question:** How does the dependency array work?
- **SafeWeb AI Use:** Load recent scan history on dashboard.

9. Hooks (Intro)

- **What:** Special functions to use React features in functional components.
- **Why Important:** Replaces classes with reusable logic.
- **Under the Hood:** Hooks use internal closures and React's fiber scheduler.
- **Used in Real Apps:** All stateful or side-effect components.
- **Example:** `useState`, `useEffect`
- **Hands-On Task:** Use `useState` and `useEffect` together.
- **Interview Question:** Why can't hooks be called conditionally?
- **SafeWeb AI Use:** State and side effect logic.

10. Forms

- **What:** Collecting and managing user input.
- **Why Important:** Central for user interaction.
- **Under the Hood:** Controlled components store values in state.
- **Used in Real Apps:** Login, feedback, scanning forms.
- **Example:**
- `<input value={email} onChange={(e) => setEmail(e.target.value)} />`
- **Hands-On Task:** Build scan request form.
- **Interview Question:** Difference between controlled and uncontrolled components?
- **SafeWeb AI Use:** Get website input from users.

11. Context API

- **What:** Global state manager for passing data without prop drilling.
- **Why Important:** Simplifies state sharing.
- **Under the Hood:** Uses React context provider/consumer.
- **Used in Real Apps:** Theme, auth, locale.

- **Example:**
- `const UserContext = React.createContext();`
- **Hands-On Task:** Create `AuthContext`.
- **Interview Question:** How does context re-render consumers?
- **SafeWeb AI Use:** Share user data across pages.

12. React Router

- **What:** Routing library to handle navigation.
- **Why Important:** Enables multi-page apps.
- **Under the Hood:** Uses HTML5 history API.
- **Used in Real Apps:** Page navigation.
- **Example:**
- `<Route path="/scan" element={<Scan />} />`
- **Hands-On Task:** Setup `Login`, `Dashboard`, `Scan` routes.
- **Interview Question:** Difference between browser and hash routers?
- **SafeWeb AI Use:** Navigate between sections.

13. Custom Hooks

- **What:** User-defined reusable hook functions.
- **Why Important:** Abstraction and reuse of logic.
- **Under the Hood:** Functions that call other hooks.
- **Used in Real Apps:** Reusable logic blocks.
- **Example:** `useAuth`, `useScanStatus`
- **Hands-On Task:** Create `useAuthGuard`.
- **Interview Question:** Naming rule for hooks?
- **SafeWeb AI Use:** Custom logic management.

14. useMemo & useCallback

- **What:** Memoization hooks to optimize performance.
- **Why Important:** Prevents unnecessary recalculations or re-renders.
- **Under the Hood:** Caches previous return values.
- **Used in Real Apps:** Expensive operations, function refs.
- **Example:**
- `const memoized = useMemo(() => compute(data), [data]);`
- **Hands-On Task:** Optimize result rendering.
- **Interview Question:** When does `useMemo` recalculate?
- **SafeWeb AI Use:** Prevent rerender on scans.

15. Refs and useRef

- **What:** Direct DOM access or persist values across renders.
- **Why Important:** Needed for focus, timers, scroll.
- **Under the Hood:** Maintains mutable `.current` reference.

- **Used in Real Apps:** Form fields, animations.
- **Example:** `inputRef.current.focus()`
- **Hands-On Task:** Focus input on mount.
- **Interview Question:** When to use refs?
- **SafeWeb AI Use:** Scroll to report.

16. Error Boundaries

- **What:** Catch and handle render errors.
- **Why Important:** Prevent crashes.
- **Under the Hood:** Uses lifecycle methods (`componentDidCatch`).
- **Used in Real Apps:** Wrapping entire app/components.
- **Example:** Class-based error catcher.
- **Hands-On Task:** Create `ErrorBoundary` wrapper.
- **Interview Question:** Can hooks catch errors?
- **SafeWeb AI Use:** Catch scan/report failures.

17. Code Splitting

- **What:** Load JS bundles on demand.
- **Why Important:** Reduces initial load.
- **Under the Hood:** Uses `React.lazy` and dynamic import.
- **Used in Real Apps:** Lazy-loading routes/pages.
- **Example:**
- `const Scan = React.lazy(() => import('./Scan'));`
- **Hands-On Task:** Split heavy components.
- **Interview Question:** Difference between lazy and suspense?
- **SafeWeb AI Use:** Speed up first load.

18. Higher Order Components

- **What:** Functions returning components.
- **Why Important:** Reusable logic decorators.
- **Under the Hood:** Functional pattern.
- **Used in Real Apps:** Permissions, tracking.
- **Example:** `withLogging(Component)`
- **Hands-On Task:** Create HOC for authentication check.
- **Interview Question:** Compare HOCs with hooks?
- **SafeWeb AI Use:** Wrap secure dashboard.

19. Render Props

- **What:** Share code by passing render function as prop.
- **Why Important:** Flexible pattern.
- **Under the Hood:** Functions as children.
- **Used in Real Apps:** Sliders, animations.

- **Example:** `<DataProvider render={(data) => <Chart data={data} />}`
- **Hands-On Task:** Create `MouseTracker`.
- **Interview Question:** Downsides of render props?
- **SafeWeb AI Use:** Share UI logic.

20. State Management

- **What:** Tools to manage complex app state.
- **Why Important:** Scalability.
- **Under the Hood:** Central store and dispatch pattern.
- **Used in Real Apps:** Redux, Zustand.
- **Example:** `useReducer`, `Redux` store
- **Hands-On Task:** Manage scan logs in global state.
- **Interview Question:** Redux vs context?
- **SafeWeb AI Use:** Global scan/auth state.

21. Axios / Fetch

- **What:** HTTP libraries to talk to APIs.
- **Why Important:** Core for data interaction.
- **Under the Hood:** Promise-based APIs.
- **Used in Real Apps:** Backend communication.
- **Example:** `axios.get('/api/report')`
- **Hands-On Task:** Fetch scan results.
- **Interview Question:** Axios vs fetch?
- **SafeWeb AI Use:** Talk to scan API.

22. Folder Structure

- **What:** Organizing codebase.
- **Why Important:** Maintainability.
- **Under the Hood:** Logical separation.
- **Used in Real Apps:** Feature/domain-based.
- **Example:** `src/components`, `src/services`
- **Hands-On Task:** Restructure SafeWeb AI project.
- **Interview Question:** Best practices for large apps?
- **SafeWeb AI Use:** Scale codebase.

23. PropTypes / TypeScript

- **What:** Tools to validate types.
- **Why Important:** Prevent runtime bugs.
- **Under the Hood:** Type checking in runtime or compile-time.
- **Used in Real Apps:** Type safety.
- **Example:**
- `Component.propTypes = { name: PropTypes.string }`

- **Hands-On Task:** Add PropTypes to core components.
- **Interview Question:** TS vs PropTypes?
- **SafeWeb AI Use:** Ensure correct data handling.

24. Testing (Jest + RTL)

- **What:** Automated validation of logic and UI.
- **Why Important:** Prevent regressions.
- **Under the Hood:** Simulates components.
- **Used in Real Apps:** CI pipelines.
- **Example:** `expect(getByText('Welcome')).toBeInTheDocument()`
- **Hands-On Task:** Test login and scan page.
- **Interview Question:** Unit vs integration test?
- **SafeWeb AI Use:** Bug-free reliability.

React Interview Questions & Precise Answers

---

1. **What happens to JSX during the build process?**
   - JSX is transpiled by Babel into `React.createElement()` calls that build the virtual DOM.
2. **Difference between functional and class components?**
   - Functional: Simple functions using hooks. Class: ES6 classes using lifecycle methods.
3. **Can a child modify props?**
   - No. Props are read-only; only the parent can change them.
4. **How does state cause a re-render?**
   - Calling `setState` triggers React's scheduler to re-render that component with updated state.
5. **What is a synthetic event?**
   - A cross-browser wrapper around native events, provided by React for consistency.
6. **Ways to implement conditional rendering?**
   - `if/else`, ternary `? :`, logical AND `&&`, IIFE inside JSX.
7. **Why are keys important in lists?**
   - Keys help React identify which items changed, are added, or removed for efficient re-renders.
8. **How does the dependency array work in useEffect?**
   - It determines when the effect runs. Empty array means run once. Include variables to trigger rerun when they change.
9. **Why can't hooks be called conditionally?**

- Because React relies on the order of hooks. Conditional calls break this order, causing bugs.

10. **Difference between controlled and uncontrolled components?**
    - Controlled: state-driven. Uncontrolled: DOM-driven (using refs).
11. **How does context re-render consumers?**
    - When context value changes, all consumers using that context re-render.
12. **Difference between browser and hash routers?**
    - BrowserRouter uses HTML5 history API. HashRouter uses URL hash (`#`) for routing.
13. **Naming rule for custom hooks?**
    - Must start with `use` to let React track them internally.
14. **When does useMemo recalculate?**
    - When any dependency in its dependency array changes.
15. **When to use refs?**
    - When you need direct DOM access or to persist values without triggering re-renders.
16. **Can hooks catch errors?**
    - No. Only class-based error boundaries can catch render-time errors.
17. **Difference between lazy and suspense?**
    - `lazy` dynamically imports components. `Suspense` wraps lazy-loaded components to handle loading fallback.
18. **Compare HOCs with hooks?**
    - HOCs wrap components for reuse. Hooks reuse logic inside functional components.
19. **Downsides of render props?**
    - Can lead to deeply nested structures (callback hell).
20. **Redux vs Context?**
    - Redux is for complex/global state with middleware. Context is for simple global sharing.
21. **Axios vs fetch?**
    - Axios: cleaner syntax, interceptors, error handling. Fetch: native API, more boilerplate.
22. **Best practices for large apps folder structure?**
    - Feature-based or domain-based structure with clear separation of concerns.
23. **TypeScript vs PropTypes?**
    - TypeScript checks types at compile-time. PropTypes check at runtime and are limited.
24. **Unit vs integration test?**
    - Unit: test single component/function. Integration: test interactions between components.
