React Basics

Q1: What is React?  
A1: React is a JavaScript library for building user interfaces, primarily for single-page applications. It allows developers to create reusable UI components.

Q2: What is JSX?  
A2: JSX is a syntax extension for JavaScript that allows you to write HTML-like code inside JavaScript. React uses JSX to describe UI structure.

Q3: What are components in React?  
A3: Components are reusable pieces of UI. They can be functional components or class components.

Q4: What is the difference between functional and class components?  
A4: Functional components are JavaScript functions that return JSX. Class components are ES6 classes that extend React.Component and have lifecycle methods.

Q5: What is the virtual DOM?  
A5: Virtual DOM is a lightweight copy of the real DOM. React uses it to efficiently update only the changed parts of the UI.

Q6: What are props in React?  
A6: Props (short for properties) are read-only inputs passed from parent to child components to make them dynamic and reusable.

Q7: What is state in React?  
A7: State is an object that stores dynamic data in a component. Changing state triggers a re-render of the component.

Q8: What is the difference between state and props?  
A8: Props are read-only and passed from parent components. State is managed within the component and can change over time.

Q9: What is a key in React lists?  
A9: A key is a unique identifier for elements in a list. It helps React efficiently update the DOM when items change.

Q10: What are React events?  
A10: React events are synthetic events that wrap native browser events, providing cross-browser compatibility. Examples: onClick, onChange.

Q11: What is a controlled component in React?  
A11: A controlled component is a form element whose value is controlled by React state. Input changes are handled via onChange events.

Q12: What is an uncontrolled component?  
A12: An uncontrolled component stores its own state internally, typically accessed via ref.

Q13: Can a component return multiple elements?  
A13: Yes, using React Fragments (<></> or <React.Fragment>).

Q14: What is defaultProps?  
A14: defaultProps is used to set default values for props in a component if the parent does not pass them.

Q15: What is prop drilling?  
A15: Prop drilling is passing props through multiple intermediate components to reach a deeply nested child.

Q16: How can you avoid prop drilling?  
A16: Using Context API or state management libraries like Redux.

Q17: Can you update props in a child component?  
A17: No, props are read-only; only the parent can change them.

Q18: What are higher-order components (HOC)?  
A18: HOCs are functions that take a component and return a new enhanced component.

Q19: What is the difference between a presentational and container component?  
A19: Presentational components focus on UI. Container components handle logic and state.

Q20: What is React.memo?  
A20: React.memo is a HOC that prevents unnecessary re-renders of functional components if props haven’t changed.

Q21: What are React lifecycle methods?  
A21: Lifecycle methods are special methods in class components that run during different phases: mounting, updating, and unmounting.

Q22: Name some common lifecycle methods.  
A22: componentDidMount, componentDidUpdate, componentWillUnmount, shouldComponentUpdate, getDerivedStateFromProps.

Q23: What is the difference between componentDidMount and useEffect?  
A23: componentDidMount is for class components, runs once after mount. useEffect can replace it in functional components.

Q24: What is the difference between state and context?  
A24: State is local to a component. Context provides global data accessible by any nested component.

Q25: Can you have multiple state hooks in a single component?  
A25: Yes, you can call useState multiple times for different pieces of state.

Q26: What is the difference between useState and useReducer?  
A26: useState is simpler for basic state. useReducer is better for complex state logic with multiple actions.

Q27: What is the purpose of getDerivedStateFromProps?  
A27: It allows a component to update its state based on props changes before rendering.

Q28: How does React handle state updates asynchronously?  
A28: React batches state updates for performance. The updated state may not be immediately available after setState.

Q29: Can functional components have lifecycle methods?  
A29: Not directly, but hooks like useEffect allow similar lifecycle behavior.

Q30: What is the difference between componentWillUnmount and cleanup in useEffect?  
A30: Both are used to clean up resources, like subscriptions or timers, when a component is removed.

Q31: How do you handle events in React?  
A31: Using camelCase props like onClick, onChange, and passing functions as handlers.

Q32: Why use arrow functions in event handlers?  
A32: To bind this automatically in class components.

Q33: What is a synthetic event?  
A33: React wraps native events into SyntheticEvent, providing cross-browser compatibility.

Q34: How do you prevent default behavior in React events?  
A34: Use event.preventDefault() inside the event handler.

Q35: What is two-way binding in React?  
A35: Controlled components allow two-way binding: input value is synced with state and updates automatically.

Q36: How do you access the value of an input field in uncontrolled components?  
A36: Using a ref to get the DOM element’s value (ref.current.value).

Q37: What is the difference between onSubmit and onClick?  
A37: onSubmit is used for forms to handle submission. onClick is used for buttons or clickable elements.

Q38: How do you handle multiple input fields in a form?  
A38: Use a single state object and update fields dynamically with [e.target.name]: e.target.value.

Q39: How do you prevent excessive re-renders in forms?  
A39: Use useCallback for handlers and React.memo for input components.

Q40: What is the difference between controlled and uncontrolled forms?  
A40: Controlled forms rely on React state for input values. Uncontrolled forms rely on DOM for input values.

Q41: What is the purpose of useEffect?  
A41: To handle side effects like API calls, subscriptions, or timers in functional components.

Q42: What is the dependency array in useEffect?  
A42: It tells React when to run the effect. Empty array → runs once after mount; dependencies → runs when they change.

Q43: What is useRef used for?  
A43: To access DOM elements directly or store mutable values that don’t trigger re-renders.

Q44: What is useMemo?  
A44: It memoizes a computed value to prevent expensive recalculations on every render.

Q45: What is useCallback?  
A45: It memoizes a function to prevent re-creating it on every render, useful for child components.

Q46: What is a custom hook?  
A46: A reusable function that uses hooks to share logic across components.

Q47: Can hooks be used in class components?  
A47: No, hooks only work in functional components.

Q48: What rules must hooks follow?  
A48:

Call hooks at the top level (not inside loops or conditions)

Call hooks only in functional components or custom hooks

Q49: What is useLayoutEffect?  
A49: Similar to useEffect, but runs synchronously after DOM mutations for layout-related calculations.

Q50: What is useImperativeHandle?  
A50: It customizes the instance value exposed to parent components when using ref.

Q51: What is React Router?  
A51: React Router is a library for handling client-side routing in React, allowing navigation between components without page reloads.

Q52: How do you install React Router?  
A52: Using npm:

npm install react-router-dom

Q53: What is the difference between BrowserRouter and HashRouter?  
A53:

BrowserRouter uses HTML5 history API (clean URLs).

HashRouter uses # in URLs, suitable for static servers.

Q54: How do you define a route in React Router v6?  
A54:

<Routes>

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

</Routes>

Q55: What is Link in React Router?  
A55: Link is used for navigation instead of <a> to avoid full page reloads.

Q56: How do you navigate programmatically?  
A56: Using the useNavigate hook:

const navigate = useNavigate();

navigate("/home");

Q57: What is Outlet in React Router?  
A57: Outlet is a placeholder for rendering nested routes.

Q58: How do you pass parameters in routes?  
A58: Use dynamic segments:

<Route path="/user/:id" element={<User />} />

Q59: How do you access route parameters?  
A59: Using useParams hook:

const { id } = useParams();

Q60: How do you handle 404 pages?  
A60: Add a route with path="\*":

<Route path="\*" element={<NotFound />} />

State Management & Context (Q61–70)

Q61: What is the Context API?  
A61: Context API allows sharing state across components without prop drilling.

Q62: How do you create a context?

const ThemeContext = React.createContext();

Q63: How do you provide context to components?  
A63: Using Provider:

<ThemeContext.Provider value={theme}>

<App />

</ThemeContext.Provider>

Q64: How do you consume context?  
A64: Using useContext:

const theme = useContext(ThemeContext);

Q65: What is the difference between Redux and Context API?  
A65: Redux is better for large-scale state management and provides tools like middleware. Context API is simpler and suitable for smaller apps.

Q66: What is a store in Redux?  
A66: Store holds the global state of the app.

Q67: What are actions in Redux?  
A67: Actions are plain objects describing a state change.

Q68: What are reducers in Redux?  
A68: Reducers are pure functions that take current state and action, and return a new state.

Q69: How do you connect a component to Redux?  
A69: Using useSelector (for state) and useDispatch (for actions).

Q70: What is Redux Toolkit?  
A70: Redux Toolkit is the official Redux toolset that simplifies store setup, reducers, and actions.

API Integration & Data Fetching (Q71–80)

Q71: How do you fetch data in React?  
A71: Using fetch or libraries like axios, usually inside useEffect.

Q72: What is the difference between fetch and axios?  
A72: axios provides simpler syntax, automatic JSON parsing, and better error handling.

Q73: How do you handle loading state?  
A73: Use a loading state variable:

const [loading, setLoading] = useState(true);

Q74: How do you handle errors during API calls?  
A74: Use try/catch blocks or .catch() with promises.

Q75: What is optimistic UI update?  
A75: Update UI immediately before the server confirms the change, then rollback if the request fails.

Q76: What is the difference between client-side and server-side data fetching?  
A76: Client-side fetches after rendering; server-side fetches before rendering (e.g., Next.js).

Q77: How do you cancel an API request in React?  
A77: Using AbortController with fetch or cancellation tokens with axios.

Q78: What is SWR or React Query?  
A78: Libraries that simplify data fetching, caching, and background updates.

Q79: How do you handle pagination with APIs?  
A79: Use query parameters like ?page=2&limit=50 and update state on each request.

Q80: How do you debounce API calls?  
A80: Use setTimeout or libraries like lodash.debounce to delay requests during typing.

Performance Optimization (Q81–90)

Q81: What is React.memo used for?  
A81: Prevents unnecessary re-renders of functional components if props haven’t changed.

Q82: What is useMemo?  
A82: Memoizes a computed value to avoid expensive recalculations.

Q83: What is useCallback?  
A83: Memoizes functions to avoid re-creating them on every render.

Q84: What is code splitting?  
A84: Splitting app bundles into smaller chunks to reduce initial load time.

Q85: How do you implement lazy loading in React?  
A85: Using React.lazy and Suspense.

Q86: What is virtualization in React?  
A86: Rendering only visible rows in a long list using libraries like react-window.

Q87: How do you avoid unnecessary re-renders?  
A87: Use memoization (React.memo, useMemo, useCallback) and pure components.

Q88: What is the impact of large state objects on performance?  
A88: Updating large state can cause slow re-renders; split state into smaller pieces.

Q89: What is reconciliation in React?  
A89: React compares virtual DOM with real DOM and updates only what has changed.

Q90: How do you profile a React application?  
A90: Use React DevTools Profiler to analyze component rendering times.

Advanced Hooks (Q101–110)

Q101: What is useLayoutEffect?  
A101: Similar to useEffect, but it runs synchronously after DOM mutations, useful for measuring layout changes.

Q102: What is useImperativeHandle?  
A102: Customizes the instance value exposed to parent components when using ref.

Q103: What is useDebugValue?  
A103: Allows you to display debug information for custom hooks in React DevTools.

Q104: What is a custom hook?  
A104: A reusable function that uses hooks to share logic across multiple components.

Q105: Can hooks be conditional?  
A105: No, hooks must be called at the top level of the component.

Q106: How do you share state logic between components?  
A106: Using custom hooks or context + hooks.

Q107: How do you prevent re-renders when passing functions?  
A107: Wrap functions with useCallback.

Q108: How do you memoize expensive values?  
A108: Use useMemo with dependencies to recompute only when necessary.

Q109: Can you use useEffect for data fetching?  
A109: Yes, it’s the standard hook for side effects like API calls.

Q110: How do you clean up subscriptions in hooks?  
A110: Return a cleanup function from useEffect:

useEffect(() => {

const id = setInterval(() => {}, 1000);

return () => clearInterval(id);

}, []);

Animations & UI Effects (Q111–120)

Q111: How do you add CSS animations in React?  
A111: Use CSS classes with className or libraries like Framer Motion.

Q112: What is Framer Motion?  
A112: A React library for smooth animations and gestures.

Q113: How do you animate a component on mount?  
A113: Using useEffect to add CSS classes or with Framer Motion’s <motion.div> initial/animate props.

Q114: What is CSSTransition?  
A114: Part of react-transition-group, used to apply CSS animations when components enter or leave.

Q115: How do you animate a list of items?  
A115: Use react-transition-group’s TransitionGroup or Framer Motion’s AnimatePresence.

Q116: How do you trigger animations on scroll?  
A116: Use libraries like AOS or detect scroll with window.scrollY and update state.

Q117: How do you pause animations in React?  
A117: Control animation state via state variables or Framer Motion props.

Q118: What is the difference between CSS transitions and JS animations?  
A118: CSS transitions are simpler and GPU-optimized, JS animations offer more control.

Q119: How do you animate route transitions?  
A119: Wrap <Routes> in <AnimatePresence> or use CSSTransition for component mount/unmount animations.

Q120: How do you optimize animations for performance?  
A120: Use transform & opacity, avoid layout thrashing, and leverage requestAnimationFrame.

Form Handling (Q121–130)

Q121: What are controlled forms?  
A121: Forms where input values are managed via React state.

Q122: What are uncontrolled forms?  
A122: Forms where input values are managed by the DOM, accessed via refs.

Q123: How do you handle multiple input fields?  
A123: Store values in an object state and update dynamically:

setForm({ ...form, [name]: value });

Q124: What is Formik?  
A124: A popular React library to manage forms, validation, and submission.

Q125: What is React Hook Form?  
A125: Lightweight library for controlled/uncontrolled forms with minimal re-renders.

Q126: How do you validate forms in React?  
A126: Use libraries like Yup, Formik, or custom validation functions.

Q127: How do you handle file uploads in forms?  
A127: Use input type="file" and handle onChange to access e.target.files.

Q128: How do you prevent form submission?  
A128: Call e.preventDefault() in the submit handler.

Q129: How do you reset a form?  
A129: Reset state to initial values or use reset() in React Hook Form.

Q130: How do you display form errors?  
A130: Store error messages in state or use library-provided error objects to render messages.

Fullstack Integration (Q131–150)

Q131: How do you call an API in React?  
A131: Use fetch or axios inside useEffect.

Q132: How do you handle CORS in React?  
A132: Configure the server to allow cross-origin requests or use a proxy.

Q133: How do you handle authentication in React?  
A133: Store tokens in memory, cookies, or localStorage and include them in API headers.

Q134: How do you protect routes in React?  
A134: Use PrivateRoute components that check authentication before rendering.

Q135: How do you manage global state with APIs?  
A135: Use Context API, Redux, or React Query to store API data.

Q136: How do you handle optimistic updates?  
A136: Update UI immediately, send API request, and rollback if the request fails.

Q137: How do you handle real-time data?  
A137: Use WebSockets, Socket.IO, or SSE for live updates.

Q138: How do you fetch data on demand?  
A138: Trigger API calls on button clicks, scroll, or events instead of on mount.

Q139: How do you cache API data?  
A139: Use libraries like React Query or Redux with caching logic.

Q140: How do you handle pagination with APIs?  
A140: Fetch data page by page and append it to existing state.

Q141: What is server-side rendering (SSR)?  
A141: Rendering React components on the server and sending HTML to the client.

Q142: What is static site generation (SSG)?  
A142: Pre-rendering pages at build time, often used in Next.js.

Q143: What is hydration in React?  
A143: Attaching React event listeners to server-rendered HTML on the client.

Q144: How do you handle authentication tokens?  
A144: Store in httpOnly cookies for security or localStorage for convenience.

Q145: How do you implement role-based access in React?  
A145: Check user role in route guards or render components conditionally.

Q146: How do you handle API rate limits?  
A146: Use throttling, debouncing, or caching to reduce requests.

Q147: How do you handle errors globally?  
A147: Use Error Boundaries for UI errors and a global error handler for API errors.

Q148: How do you perform batch API requests?  
A148: Use Promise.all to send multiple requests in parallel.

Q149: How do you secure React API requests?  
A149: Use HTTPS, authentication tokens, and server-side validation.

Q150: How do you optimize large data rendering?  
A150: Use virtualization, pagination, lazy loading, and memoization.

Advanced State Management (Q151–160)

Q151: What is Redux Thunk?  
A151: Middleware that allows writing asynchronous logic in Redux actions.

Q152: What is Redux Saga?  
A152: Middleware for managing side effects in Redux using generator functions.

Q153: What is the difference between Redux Thunk and Redux Saga?  
A153: Thunk uses functions for async actions; Saga uses generator functions and provides more control over complex async flows.

Q154: How do you structure a Redux store?  
A154: Typically: actions, reducers, store, selectors, sometimes split by feature modules.

Q155: What is a selector in Redux?  
A155: A function that extracts specific data from the Redux store for components.

Q156: How do you handle immutable updates in Redux?  
A156: Use spread operators, array methods like map/filter, or libraries like Immer.

Q157: What is the difference between local state and global state?  
A157: Local state is component-specific; global state is shared across the app.

Q158: How do you combine multiple reducers?  
A158: Using combineReducers from Redux.

Q159: What is middleware in Redux?  
A159: Middleware intercepts actions before reaching reducers to add custom logic (e.g., logging, async calls).

Q160: What is the difference between Context API and Redux for global state?  
A160: Context API is simpler, suitable for small apps; Redux provides more features, better for large-scale apps.

Performance Optimization (Q161–170)

Q161: How do you prevent unnecessary re-renders?  
A161: Use React.memo, useMemo, useCallback, and pure components.

Q162: What is lazy loading in React?  
A162: Loading components on demand using React.lazy and Suspense.

Q163: What is code splitting?  
A163: Dividing app into smaller bundles to reduce initial load time.

Q164: How do you optimize long lists?  
A164: Use virtualization libraries like react-window or react-virtualized.

Q165: How do you optimize images in React?  
A165: Use optimized image formats, lazy loading, and responsive sizes.

Q166: What is memoization in React?  
A166: Caching the result of a calculation to avoid recomputing on every render.

Q167: How do you profile a React app?  
A167: Use React DevTools Profiler to identify performance bottlenecks.

Q168: What is the difference between useEffect and useLayoutEffect for performance?  
A168: useEffect runs after painting, useLayoutEffect runs before painting, which can block rendering.

Q169: How do you reduce bundle size?  
A169: Use code splitting, tree shaking, and lazy loading.

Q170: How do you avoid anonymous functions in JSX?  
A170: Use useCallback to memoize handlers and prevent recreating functions every render.

Hooks Deep Dive (Q171–180)

Q171: Can you call hooks inside loops or conditions?  
A171: No, hooks must be called at the top level.

Q172: How do you share logic between hooks?  
A172: By creating custom hooks.

Q173: What is useDebugValue used for?  
A173: Display debug information for custom hooks in React DevTools.

Q174: What is the difference between useRef and createRef?  
A174: useRef persists across renders in functional components; createRef creates a new ref each render in class components.

Q175: What is useImperativeHandle?  
A175: Customize the instance value exposed to parent components via ref.

Q176: What is the difference between useEffect and useLayoutEffect?  
A176: useEffect runs after painting, useLayoutEffect runs synchronously before painting.

Q177: Can you nest hooks inside custom hooks?  
A177: Yes, as long as hooks follow the rules of hooks.

Q178: How do you memoize expensive computations?  
A178: Use useMemo with a dependency array.

Q179: How do you memoize functions?  
A179: Use useCallback with dependencies.

Q180: How do hooks improve functional components?  
A180: They allow functional components to use state, lifecycle, and side effects without classes.

Testing (Q181–190)

Q181: How do you test a component’s rendering?  
A181: Use render(<Component />) and query elements using React Testing Library.

Q182: How do you test events?  
A182: Use fireEvent or userEvent to simulate clicks, typing, or form submission.

Q183: What is snapshot testing?  
A183: Capture component output and compare with future renders to detect unexpected changes.

Q184: How do you mock API calls?  
A184: Use jest.mock or libraries like msw (Mock Service Worker).

Q185: How do you test async components?  
A185: Use waitFor or findBy queries in React Testing Library.

Q186: How do you test hooks?  
A186: Use @testing-library/react-hooks or wrap in a test component.

Q187: How do you test Redux-connected components?  
A187: Wrap the component in <Provider store={store}> during testing.

Q188: What is end-to-end testing?  
A188: Testing the full app flow from user actions to backend responses using tools like Cypress.

Q189: How do you test error boundaries?  
A189: Render a component that throws an error and check if the fallback UI appears.

Q190: Why is testing important in React?  
A190: Ensures UI correctness, prevents regressions, and improves maintainability.

Best Practices (Q191–200)

Q191: Why use functional components over class components?  
A191: Simpler syntax, hooks support, less boilerplate, and easier testing.

Q192: Why keep components small and focused?  
A192: Improves readability, reusability, and testability.

Q193: How do you avoid prop drilling?  
A193: Use Context API or state management libraries like Redux.

Q194: Why memoize components?  
A194: To prevent unnecessary re-renders and improve performance.

Q195: Why use keys in lists?  
A195: Keys help React identify which items changed, added, or removed.

Q196: How do you structure a React project?  
A196: Typically: components/, pages/, hooks/, services/, store/, assets/.

Q197: How do you handle errors gracefully?  
A197: Use Error Boundaries and proper try/catch for async operations.

Q198: Why separate UI and logic?  
A198: Makes components clean, reusable, and easier to test.

Q199: What is the difference between controlled and uncontrolled components?  
A199: Controlled → state drives input values. Uncontrolled → DOM manages input values.

Q200: How do you improve React app maintainability?  
A200: Follow component-based architecture, proper state management, hooks, modular code, and testing.

We’ve now completed React Q&A 151–200.

I can continue with questions 201–250 next, covering Advanced Patterns, React Router Deep Dive, Animations, Forms, React Query, SSR, and Next.js basics.

Do you want me to continue with the next 50 questions?

You said:

yes

ChatGPT said:

Great! Here’s React Q&A 201–250, covering Advanced Patterns, React Router Deep Dive, Animations, Forms, React Query, SSR, and Next.js basics.

Advanced Patterns (Q201–210)

Q201: What are Higher-Order Components (HOC)?  
A201: Functions that take a component and return a new enhanced component with additional functionality.

Q202: What is render props?  
A202: A technique where a component accepts a function as a prop to determine what to render.

Q203: What is the difference between HOC and render props?  
A203: HOC wraps a component to extend behavior; render props passes a function to customize rendering.

Q204: What are compound components?  
A204: Components that work together, sharing implicit state via context to make a flexible API.

Q205: What is controlled composition in React?  
A205: Pattern where the parent controls child behavior via props or context.

Q206: What is provider pattern?  
A206: Using a context provider to share state/functions with children components.

Q207: How do you implement error boundaries?  
A207: Create a class component implementing componentDidCatch and getDerivedStateFromError.

Q208: Can functional components be error boundaries?  
A208: No, only class components can be error boundaries currently.

Q209: What are hooks rules in advanced patterns?  
A209: Hooks must be called at the top level and only in functional components or custom hooks.

Q210: What is lazy component loading?  
A210: Loading components on demand using React.lazy and <Suspense> for better performance.

React Router Deep Dive (Q211–220)

Q211: How do you implement nested routes?  
A211: Use <Outlet /> in the parent component and define child routes inside <Routes>.

Q212: How do you navigate programmatically?  
A212: Use the useNavigate hook:

const navigate = useNavigate();

navigate("/path");

Q213: How do you protect routes?  
A213: Use PrivateRoute component to check authentication before rendering children.

Q214: How do you redirect routes?  
A214: Use <Navigate to="/path" /> in React Router v6.

Q215: How do you pass state in navigation?  
A215:

navigate("/path", { state: { key: "value" } });

Q216: How do you read navigation state?  
A216: Use useLocation hook:

const location = useLocation();

console.log(location.state);

Q217: How do you handle 404 routes?  
A217: Use a wildcard route: <Route path="\*" element={<NotFound />} />.

Q218: How do you scroll to top on route change?  
A218: Use useEffect and window.scrollTo(0, 0) on location change.

Q219: How do you lazy load routes?  
A219: Wrap components in React.lazy and Suspense.

Q220: How do you handle query parameters?  
A220: Use useSearchParams to get or set query parameters.

Animations & UI Effects (Q221–230)

Q221: How do you animate components on mount?  
A221: Use useEffect to add CSS classes or Framer Motion’s initial and animate props.

Q222: What is Framer Motion?  
A222: A React library for smooth animations, transitions, and gestures.

Q223: How do you animate lists?  
A223: Use TransitionGroup from react-transition-group or AnimatePresence from Framer Motion.

Q224: What is keyframes animation in React?  
A224: Using CSS @keyframes to define animations applied via className.

Q225: How do you animate route transitions?  
A225: Wrap <Routes> in <AnimatePresence> or use CSSTransition for component enter/exit.

Q226: How do you pause or stop animations?  
A226: Control animation state via React state variables.

Q227: How do you animate using GSAP in React?  
A227: Use useRef to target DOM nodes and GSAP methods in useEffect.

Q228: How do you animate scroll position?  
A228: Use window.scrollTo with smooth behavior or libraries like react-scroll.

Q229: How do you optimize animations?  
A229: Use transform & opacity, avoid layout thrashing, and minimize DOM updates.

Q230: What is the difference between CSS transitions and JS animations?  
A230: CSS transitions are simpler and GPU-optimized, JS animations give more control.

Forms & Validation (Q231–240)

Q231: What is controlled form?  
A231: Input values are managed via React state.

Q232: What is uncontrolled form?  
A232: Input values are managed by DOM using ref.

Q233: How do you handle multiple inputs in a form?  
A233: Use a state object and update dynamically: [name]: value.

Q234: What is Formik?  
A234: A library to manage forms, validation, and submission in React.

Q235: What is React Hook Form?  
A235: Lightweight library for controlled/uncontrolled forms with minimal re-renders.

Q236: How do you validate forms?  
A236: Use custom validation functions, Yup, or library-based validation.

Q237: How do you display error messages?  
A237: Store errors in state or use library-provided error objects.

Q238: How do you reset forms?  
A238: Reset state to initial values or call reset() in React Hook Form.

Q239: How do you handle file uploads?  
A239: Use input type="file" and onChange to access e.target.files.

Q240: How do you prevent default form submission?  
A240: Call e.preventDefault() in the submit handler.

React Query & Data Fetching (Q241–250)

Q241: What is React Query?  
A241: A library for data fetching, caching, and synchronization in React.

Q242: How do you fetch data with React Query?  
A242: Use useQuery hook:

const { data, isLoading } = useQuery("key", fetchFunction);

Q243: How do you perform mutations in React Query?  
A243: Use useMutation for POST, PUT, DELETE requests.

Q244: How does React Query cache data?  
A244: It stores data by query key and invalidates or refetches when needed.

Q245: How do you handle background refetching?  
A245: Set refetchInterval in useQuery.

Q246: How do you invalidate queries?  
A246: Use queryClient.invalidateQueries("key") after mutation.

Q247: How do you prefetch data in React Query?  
A247: Use queryClient.prefetchQuery before component renders.

Q248: How do you handle pagination with React Query?  
A248: Use query keys including page number and fetch accordingly.

Q249: How do you handle errors with React Query?  
A249: Use onError callback or check isError state from useQuery.

Q250: How do you integrate React Query with SSR?  
A250: Use Hydrate and dehydrate from React Query to pass server-fetched data to the client.

We’ve now completed React Q&A 201–250.

I can continue with questions 251–300 next, covering Next.js, SSR/SSG, Performance, Advanced Patterns, Testing, and Deployment Best Practices.

Do you want me to continue with the next 50 questions?

You said:

yes

ChatGPT said:

Great! Here’s React Q&A 251–300, focusing on Next.js, SSR/SSG, Performance, Advanced Patterns, Testing, and Deployment Best Practices.

Next.js Basics & SSR (Q251–260)

Q251: What is Next.js?  
A251: A React framework for server-side rendering (SSR), static site generation (SSG), and hybrid apps.

Q252: What is SSR in Next.js?  
A252: Server-Side Rendering renders pages on the server at request time and sends HTML to the client.

Q253: What is SSG in Next.js?  
A253: Static Site Generation pre-renders pages at build time, producing static HTML files.

Q254: What is the difference between SSR and SSG?  
A254: SSR renders on each request; SSG renders once at build time.

Q255: What is ISR (Incremental Static Regeneration)?  
A255: Next.js can rebuild static pages after deployment at runtime for updated content.

Q256: How do you fetch data for SSR in Next.js?  
A256: Use getServerSideProps function in the page component.

Q257: How do you fetch data for SSG in Next.js?  
A257: Use getStaticProps function in the page component.

Q258: How do you define dynamic routes in Next.js?  
A258: Use square brackets in filenames, e.g., pages/user/[id].js.

Q259: How do you access query parameters in Next.js?  
A259: Use useRouter hook or context.params in getServerSideProps/getStaticProps.

Q260: How do you navigate programmatically in Next.js?  
A260: Use useRouter’s router.push("/path") method.

Next.js Advanced (Q261–270)

Q261: What is getStaticPaths?  
A261: Used with dynamic SSG routes to pre-render pages at build time for specific paths.

Q262: What is getInitialProps?  
A262: Fetches data on both server and client for page components, mostly replaced by getStaticProps/getServerSideProps.

Q263: What is Head in Next.js?  
A263: A component from next/head to set page title, meta tags, and SEO elements.

Q264: How do you handle API routes in Next.js?  
A264: Create files under pages/api/ with a default export function to handle requests.

Q265: What is the difference between getStaticProps and getServerSideProps?  
A265: getStaticProps → build-time data fetch; getServerSideProps → per-request server-side fetch.

Q266: How do you implement middleware in Next.js?  
A266: Use middleware.js at root or in /pages to intercept requests.

Q267: How do you handle image optimization?  
A267: Use the <Image> component from next/image for optimized images.

Q268: How do you implement redirects?  
A268: Return redirect object from getServerSideProps or next.config.js.

Q269: How do you implement rewrites?  
A269: Use rewrites in next.config.js to map URLs internally.

Q270: How do you handle environment variables in Next.js?  
A270: Define .env.local variables prefixed with NEXT\_PUBLIC\_ for client-side access.

Performance Optimization (Q271–280)

Q271: How do you reduce bundle size?  
A271: Use dynamic imports, code splitting, and tree shaking.

Q272: What is lazy loading in React/Next.js?  
A272: Loading components on demand using React.lazy or next/dynamic.

Q273: How do you optimize images?  
A273: Use next/image, responsive sizes, and modern formats like WebP.

Q274: How do you memoize components?  
A274: Use React.memo to prevent unnecessary re-renders.

Q275: How do you prevent unnecessary API calls?  
A275: Use caching, useEffect dependency arrays, or React Query.

Q276: How do you optimize long lists?  
A276: Use virtualization libraries like react-window or react-virtualized.

Q277: How do you reduce re-rendering of child components?  
A277: Pass stable props using useMemo and useCallback.

Q278: How do you profile React apps?  
A278: Use React DevTools Profiler to analyze rendering performance.

Q279: What is SSR caching?  
A279: Cache server-rendered HTML for repeated requests to improve response time.

Q280: How do you implement lazy images on scroll?  
A280: Use loading="lazy" in <img> tags or Next.js <Image> component.

Advanced Patterns & State Management (Q281–290)

Q281: What is the compound component pattern?  
A281: Components that share implicit state via context to create flexible APIs.

Q282: What is the provider pattern?  
A282: Use context providers to pass state/functions to nested components.

Q283: What is the container/presentational pattern?  
A283: Containers handle logic/state, Presentational components handle UI.

Q284: What is controlled composition?  
A284: Parent components control child behavior via props or context.

Q285: How do you handle global state in large apps?  
A285: Use Redux, Zustand, or Context API depending on complexity.

Q286: What is the difference between Redux Thunk and Saga?  
A286: Thunk → async actions using functions; Saga → generator functions for side-effects.

Q287: What are selectors in Redux?  
A287: Functions that extract specific data from the store.

Q288: How do you combine multiple reducers?  
A288: Use combineReducers from Redux.

Q289: How do you structure a scalable Redux store?  
A289: Use feature-based modules: actions, reducers, selectors per feature.

Q290: How do you handle optimistic updates?  
A290: Update UI immediately, then rollback if API fails.

Testing & Deployment (Q291–300)

Q291: How do you test React components?  
A291: Use React Testing Library or Enzyme with Jest for unit and integration tests.

Q292: How do you test events?  
A292: Simulate using fireEvent or userEvent in React Testing Library.

Q293: How do you test async components?  
A293: Use waitFor or findBy queries for asynchronous DOM updates.

Q294: How do you mock API calls in tests?  
A294: Use jest.mock or libraries like msw (Mock Service Worker).

Q295: How do you test hooks?  
A295: Use @testing-library/react-hooks or test hooks inside a test component.

Q296: How do you deploy a React app?  
A296: Use Vercel, Netlify, AWS, or Firebase with build output (npm run build).

Q297: How do you deploy a Next.js app?  
A297: Use Vercel (default), or Node.js server hosting with SSR support.

Q298: How do you handle environment variables during deployment?  
A298: Set .env variables in hosting platform or .env.production.

Q299: How do you monitor production performance?  
A299: Use React Profiler, Lighthouse, or analytics tools.

Q300: How do you implement CI/CD for React apps?  
A300: Use GitHub Actions, GitLab CI/CD, or Vercel deployments for automated builds and deployment.

Perfect! Here’s React Q&A 301–350, focusing on Advanced Hooks, Patterns, Performance Optimization, Testing, State Management, and Popular React Libraries.

Advanced Hooks (Q301–310)

Q301: What is useDeferredValue?  
A301: A React hook to defer updating a value to avoid blocking high-priority updates.

Q302: What is useTransition?  
A302: Used to mark non-urgent state updates, allowing React to keep the UI responsive.

Q303: What is useId?  
A303: Generates a unique stable ID for accessibility and component keys.

Q304: What is useSyncExternalStore?  
A304: Allows React to subscribe to external stores and get updates efficiently.

Q305: What is useInsertionEffect?  
A305: Similar to useLayoutEffect, but specifically runs before DOM mutations, used for styling libraries.

Q306: How do you create a custom hook?  
A306: Define a function starting with use and use hooks inside it to share logic.

Q307: Can hooks be conditional?  
A307: No, hooks must be called at the top level of components or custom hooks.

Q308: How do you prevent unnecessary computations in hooks?  
A308: Use useMemo for values and useCallback for functions.

Q309: How do you share state logic between hooks?  
A309: Use custom hooks that encapsulate shared state logic.

Q310: What is the difference between useEffect and useLayoutEffect?  
A310: useEffect runs after render, useLayoutEffect runs synchronously before painting.

Advanced Patterns (Q311–320)

Q311: What are Higher-Order Components (HOC)?  
A311: Functions that take a component and return an enhanced component.

Q312: What are render props?  
A312: Components that accept a function as a prop to customize rendering.

Q313: What is the container/presentational pattern?  
A313: Containers handle logic/state, Presentational components handle UI.

Q314: What is the compound component pattern?  
A314: Components that share state via context to create flexible APIs.

Q315: What is controlled composition?  
A315: Parent controls child behavior via props or context.

Q316: What is provider pattern?  
A316: Use context providers to share state/functions across nested components.

Q317: How do you implement error boundaries?  
A317: Use a class component implementing componentDidCatch and getDerivedStateFromError.

Q318: Can functional components be error boundaries?  
A318: No, only class components can be error boundaries currently.

Q319: How do you handle cross-cutting concerns in React?  
A319: Use HOC, render props, or custom hooks.

Q320: How do you implement feature-based folder structure?  
A320: Organize folders by features/modules: components, hooks, services per feature.

Performance Optimization (Q321–330)

Q321: How do you avoid unnecessary re-renders?  
A321: Use React.memo, useMemo, useCallback, and pure components.

Q322: What is code splitting?  
A322: Dividing app into smaller bundles to reduce initial load time.

Q323: How do you lazy load components?  
A323: Use React.lazy with <Suspense>.

Q324: How do you optimize long lists?  
A324: Use virtualization libraries like react-window or react-virtualized.

Q325: How do you memoize expensive computations?  
A325: Use useMemo with dependency arrays.

Q326: How do you memoize event handlers?  
A326: Use useCallback to prevent function recreation.

Q327: How do you optimize images in React?  
A327: Use responsive images, modern formats (WebP), and lazy loading.

Q328: How do you prevent unnecessary API calls?  
A328: Use React Query caching, dependency arrays, or throttling/debouncing.

Q329: How do you profile a React app?  
A329: Use React DevTools Profiler to identify slow components.

Q330: How do you optimize route-based code splitting?  
A330: Use React.lazy or Next.js dynamic imports for route components.

Testing (Q331–340)

Q331: How do you test React components?  
A331: Use React Testing Library or Enzyme with Jest.

Q332: How do you test events?  
A332: Simulate events with fireEvent or userEvent.

Q333: How do you test async operations?  
A333: Use waitFor or findBy queries.

Q334: How do you mock API calls?  
A334: Use jest.mock or MSW (Mock Service Worker).

Q335: How do you test hooks?  
A335: Use @testing-library/react-hooks or wrap hooks in a test component.

Q336: How do you test Redux-connected components?  
A336: Wrap the component in <Provider store={store}> during testing.

Q337: How do you test error boundaries?  
A337: Render a component that throws and check fallback UI.

Q338: How do you test route navigation?  
A338: Use MemoryRouter in tests and simulate navigation.

Q339: What is snapshot testing?  
A339: Capture rendered output and compare against future renders.

Q340: Why is testing important in React?  
A340: Ensures UI correctness, prevents regressions, and improves maintainability.

State Management (Q341–350)

Q341: What is Redux?  
A341: A state management library for predictable global state.

Q342: What is Redux Toolkit?  
A342: Official Redux toolset that simplifies store setup, reducers, and actions.

Q343: What are actions in Redux?  
A343: Plain objects describing state changes.

Q344: What are reducers?  
A344: Pure functions that take current state and action, and return new state.

Q345: What is a selector?  
A345: Function that extracts specific data from the Redux store.

Q346: How do you dispatch actions in React?  
A346: Use useDispatch hook and call dispatch(action).

Q347: How do you use slices in Redux Toolkit?  
A347: Use createSlice to define state, reducers, and actions in one place.

Q348: What is the difference between local and global state?  
A348: Local → component-specific; Global → shared across multiple components.

Q349: How do you handle async actions in Redux?  
A349: Use Redux Thunk, Redux Saga, or RTK Query.

Q350: How do you persist Redux state?  
A350: Use redux-persist to save state to localStorage or sessionStorage.

We’ve now completed React Q&A 301–350.

I can continue with questions 351–400 next, covering React Native basics, advanced Next.js features, testing strategies, hooks patterns, and ecosystem libraries.

Do you want me to continue with the next 50 questions?

You said:

yes

ChatGPT said:

Perfect! Here’s React Q&A 351–400, focusing on React Native basics, advanced Next.js features, testing strategies, hooks patterns, and ecosystem libraries.

React Native Basics (Q351–360)

Q351: What is React Native?  
A351: A framework to build cross-platform mobile apps using React.

Q352: How does React Native differ from React?  
A352: Uses native components instead of HTML and CSS; renders to mobile UI.

Q353: What are core components in React Native?  
A353: Examples: <View>, <Text>, <Image>, <ScrollView>, <TextInput>.

Q354: How do you style components in React Native?  
A354: Using StyleSheet.create() or inline styles (similar to CSS-in-JS).

Q355: How do you handle touch events?  
A355: Use TouchableOpacity, TouchableHighlight, or Pressable.

Q356: How do you navigate between screens?  
A356: Use React Navigation or React Native Navigation libraries.

Q357: How do you fetch data in React Native?  
A357: Use fetch, axios, or React Query inside useEffect.

Q358: How do you handle platform-specific code?  
A358: Use Platform.OS or platform-specific file extensions (.ios.js, .android.js).

Q359: How do you handle forms in React Native?  
A359: Use controlled components and libraries like Formik or React Hook Form.

Q360: How do you optimize performance in React Native?  
A360: Use FlatList, memoization, lazy loading, and native modules for heavy operations.

Advanced Next.js Features (Q361–370)

Q361: What is middleware in Next.js?  
A361: Code that runs before requests reach pages; used for authentication, redirects, and rewrites.

Q362: How do you implement API routes in Next.js?  
A362: Create files in pages/api/ and export a default function handling req and res.

Q363: What is ISR (Incremental Static Regeneration)?  
A363: Allows static pages to regenerate after deployment on a timed or on-demand basis.

Q364: How do you prefetch pages in Next.js?  
A364: Use <Link> with prefetch prop (default true in production).

Q365: How do you implement redirects in Next.js?  
A365: Return a redirect object from getServerSideProps or configure in next.config.js.

Q366: How do you implement rewrites?  
A366: Use rewrites in next.config.js to map paths internally without changing URLs.

Q367: How do you optimize images?  
A367: Use <Image> from next/image with width, height, and priority or loading="lazy".

Q368: How do you implement custom server in Next.js?  
A368: Use Node.js/Express server and integrate Next.js handler for SSR pages.

Q369: How do you handle environment variables?  
A369: Use .env.local or .env.production; client variables must be prefixed with NEXT\_PUBLIC\_.

Q370: How do you deploy Next.js apps?  
A370: Deploy on Vercel, AWS, or custom Node.js servers with SSR support.

Testing Strategies (Q371–380)

Q371: How do you unit test React components?  
A371: Use React Testing Library or Enzyme with Jest for component logic and rendering.

Q372: How do you test hooks?  
A372: Use @testing-library/react-hooks or wrap hooks in a test component.

Q373: How do you test Redux state changes?  
A373: Wrap components in <Provider store={store}> and assert state updates after dispatches.

Q374: How do you test async operations?  
A374: Use waitFor or findBy\* queries in React Testing Library.

Q375: How do you test navigation?  
A375: Use MemoryRouter and simulate route changes.

Q376: How do you mock API requests in tests?  
A376: Use jest.mock or MSW (Mock Service Worker).

Q377: How do you test error boundaries?  
A377: Render a component that throws and check the fallback UI is displayed.

Q378: How do you test forms?  
A378: Simulate input events and submit, then assert expected changes.

Q379: What is snapshot testing?  
A379: Captures component output and compares with future renders to detect changes.

Q380: Why is testing important in React?  
A380: Ensures UI correctness, prevents regressions, and improves maintainability.

Hooks Patterns (Q381–390)

Q381: What is the difference between useEffect and useLayoutEffect?  
A381: useEffect → runs after painting, useLayoutEffect → runs before painting.

Q382: How do you share logic between components?  
A382: Use custom hooks.

Q383: What is useRef used for?  
A383: Access DOM elements or store mutable values that persist across renders.

Q384: What is useCallback?  
A384: Memoizes functions to prevent recreation on each render.

Q385: What is useMemo?  
A385: Memoizes computed values to avoid expensive recalculations.

Q386: What is useTransition?  
A386: Marks non-urgent state updates to keep UI responsive.

Q387: What is useDeferredValue?  
A387: Delays updating a value to improve performance during heavy renders.

Q388: What is useSyncExternalStore?  
A388: Subscribes to external stores efficiently in concurrent mode.

Q389: What is useId?  
A389: Generates a stable unique ID for components or accessibility attributes.

Q390: How do you prevent excessive re-renders with hooks?  
A390: Use useMemo, useCallback, React.memo, and split state properly.

React Ecosystem Libraries (Q391–400)

Q391: What is React Router?  
A391: Library for client-side routing in React apps.

Q392: What is React Query?  
A392: Library for data fetching, caching, and synchronization.

Q393: What is Formik?  
A393: React library for form state, validation, and submission.

Q394: What is React Hook Form?  
A394: Lightweight library for controlled/uncontrolled forms with minimal re-renders.

Q395: What is Redux?  
A395: Global state management library for predictable state updates.

Q396: What is Zustand?  
A396: Lightweight global state library with simpler API than Redux.

Q397: What is Recoil?  
A397: State management library designed for React components and hooks.

Q398: What is Framer Motion?  
A398: Animation library for smooth UI transitions and gestures in React.

Q399: What is React Spring?  
A399: Library for physics-based animations in React.

Q400: What is React Native Paper?  
A400: UI library for Material Design components in React Native.

Performance Optimization (Q401–410)

Q401: How do you reduce unnecessary re-renders?  
A401: Use React.memo, useMemo, useCallback, and split state into smaller components.

Q402: What is virtualization in React?  
A402: Rendering only visible items in a list using libraries like react-window or react-virtualized.

Q403: How do you lazy-load components?  
A403: Use React.lazy and wrap with <Suspense> for fallback rendering.

Q404: How do you lazy-load images?  
A404: Use loading="lazy" in <img> tags or Next.js <Image> component.

Q405: How do you reduce bundle size?  
A405: Use code splitting, dynamic imports, tree shaking, and remove unused packages.

Q406: What is memoization in React?  
A406: Caching results of expensive computations with useMemo or useCallback.

Q407: How do you optimize forms?  
A407: Use uncontrolled components or libraries like React Hook Form to minimize re-renders.

Q408: How do you optimize route-based code splitting?  
A408: Use dynamic imports for route components in React Router or Next.js.

Q409: How do you prevent unnecessary API calls?  
A409: Use React Query caching, dependency arrays, and debouncing.

Q410: How do you optimize CSS in React?  
A410: Use CSS Modules, styled-components, or Tailwind CSS to scope styles and reduce size.

Testing Strategies (Q411–420)

Q411: How do you unit test React components?  
A411: Use React Testing Library or Enzyme with Jest.

Q412: How do you test hooks?  
A412: Use @testing-library/react-hooks or test inside a wrapper component.

Q413: How do you test async operations?  
A413: Use waitFor or findBy\* queries in React Testing Library.

Q414: How do you mock API calls?  
A414: Use jest.mock or MSW (Mock Service Worker).

Q415: How do you test Redux-connected components?  
A415: Wrap in <Provider store={store}> and assert state changes after dispatches.

Q416: How do you test form submissions?  
A416: Simulate input changes and submit events, then assert expected state.

Q417: How do you test navigation?  
A417: Use MemoryRouter or mock router functions to simulate route changes.

Q418: How do you test error boundaries?  
A418: Render a component that throws an error and check fallback UI.

Q419: What is snapshot testing?  
A419: Capture component output and compare with future renders to detect changes.

Q420: Why is testing important in React?  
A420: Ensures UI correctness, prevents regressions, and improves maintainability.

Advanced Patterns (Q421–430)

Q421: What is the container/presentational pattern?  
A421: Containers handle logic/state, Presentational components handle UI.

Q422: What is a Higher-Order Component (HOC)?  
A422: Function that takes a component and returns a new component with added functionality.

Q423: What is a render prop?  
A423: Component that accepts a function as a prop to determine rendering.

Q424: What is the compound component pattern?  
A424: Components that share state via context to provide flexible APIs.

Q425: What is controlled composition?  
A425: Parent component controls child behavior via props or context.

Q426: What is provider pattern?  
A426: Use context providers to share state/functions with nested components.

Q427: How do you handle cross-cutting concerns?  
A427: Use HOC, render props, or custom hooks.

Q428: How do you implement feature-based folder structure?  
A428: Organize by feature/module: components, hooks, services per feature.

Q429: How do you manage reusable hooks?  
A429: Create a custom hook library for shared logic across components.

Q430: How do you handle error boundaries in large apps?  
A430: Use multiple nested error boundaries for critical and non-critical sections.

SSR & Next.js (Q431–440)

Q431: What is SSR?  
A431: Server-Side Rendering: renders pages on the server at request time.

Q432: What is SSG?  
A432: Static Site Generation: pre-renders pages at build time.

Q433: What is ISR?  
A433: Incremental Static Regeneration: updates static pages after deployment.

Q434: How do you fetch data with getServerSideProps?  
A434: Export getServerSideProps in page component, return props.

Q435: How do you fetch data with getStaticProps?  
A435: Export getStaticProps for build-time static data fetching.

Q436: What is getStaticPaths?  
A436: Defines paths for dynamic SSG routes.

Q437: How do you handle API routes?  
A437: Create files in pages/api/ and export functions handling req and res.

Q438: How do you implement redirects?  
A438: Return redirect from getServerSideProps or configure in next.config.js.

Q439: How do you implement rewrites?  
A439: Use rewrites in next.config.js for URL mapping without changing the URL.

Q440: How do you deploy Next.js apps?  
A440: Deploy on Vercel, AWS, or Node.js servers with SSR support.

State Management Deep Dive (Q441–450)

Q441: What is Redux?  
A441: Predictable global state management library.

Q442: What is Redux Toolkit?  
A442: Simplifies Redux setup with slices, reducers, and actions.

Q443: What are actions in Redux?  
A443: Plain objects that describe state changes.

Q444: What are reducers?  
A444: Pure functions returning new state from previous state and action.

Q445: What is a selector?  
A445: Function that extracts specific state from the store.

Q446: How do you dispatch actions?  
A446: Use useDispatch hook and call dispatch(action).

Q447: What is the difference between local and global state?  
A447: Local → component-specific; Global → shared across multiple components.

Q448: How do you handle async actions?  
A448: Use Redux Thunk, Redux Saga, or RTK Query.

Q449: How do you persist Redux state?  
A449: Use redux-persist to save state in localStorage or sessionStorage.

Q450: How do you structure a scalable Redux store?  
A450: Use feature-based folders: actions, reducers, selectors per module.

We’ve now completed React Q&A 401–450.

I can continue with questions 451–500 next, covering React Ecosystem Libraries, React Native Advanced, Performance Patterns, Testing, SSR/Next.js Advanced, and Deployment Best Practices.

Do you want me to continue with the final 50 questions?

You said:

yes

ChatGPT said:

Perfect! Here’s React Q&A 451–500, covering React Ecosystem Libraries, React Native Advanced, Performance Patterns, Testing, SSR/Next.js Advanced, and Deployment Best Practices.

React Ecosystem Libraries (Q451–460)

Q451: What is React Query?  
A451: Library for data fetching, caching, and synchronization in React apps.

Q452: What is SWR?  
A452: React library for stale-while-revalidate data fetching.

Q453: What is Formik?  
A453: Form library for state management, validation, and submission.

Q454: What is React Hook Form?  
A454: Lightweight form library for controlled/uncontrolled components.

Q455: What is Zustand?  
A455: Lightweight state management library with simple API.

Q456: What is Recoil?  
A456: State management library optimized for React hooks.

Q457: What is Framer Motion?  
A457: Animation library for smooth UI transitions and gestures.

Q458: What is React Spring?  
A458: Physics-based animation library for React.

Q459: What is React Native Paper?  
A459: Material Design UI components for React Native.

Q460: What is React Native Elements?  
A460: Cross-platform UI component library for React Native apps.

React Native Advanced (Q461–470)

Q461: How do you optimize React Native performance?  
A461: Use FlatList, memoization, native modules, and lazy loading.

Q462: How do you handle large lists?  
A462: Use FlatList or SectionList with virtualization.

Q463: How do you handle gestures?  
A463: Use React Native Gesture Handler for smooth gestures.

Q464: How do you navigate between screens?  
A464: Use React Navigation or React Native Navigation.

Q465: How do you handle deep linking?  
A465: Configure linking prop in React Navigation.

Q466: How do you access native APIs?  
A466: Use React Native modules or Expo APIs.

Q467: How do you implement push notifications?  
A467: Use Firebase Cloud Messaging (FCM) or Expo Notifications.

Q468: How do you handle offline data?  
A468: Use AsyncStorage, SQLite, or Realm for local persistence.

Q469: How do you debug React Native apps?  
A469: Use React DevTools, Flipper, or console.log.

Q470: How do you test React Native components?  
A470: Use React Native Testing Library with Jest or Detox for E2E.

Performance Patterns (Q471–480)

Q471: How do you prevent unnecessary re-renders?  
A471: Use React.memo, useMemo, useCallback, and proper state splitting.

Q472: What is code splitting?  
A472: Dividing app into smaller bundles to reduce initial load.

Q473: How do you optimize images?  
A473: Use responsive sizes, modern formats, lazy loading, and next/image.

Q474: How do you optimize API calls?  
A474: Use React Query caching, debouncing, and conditional fetching.

Q475: How do you optimize CSS?  
A475: Use CSS Modules, styled-components, Tailwind CSS, or emotion.

Q476: How do you handle virtualization?  
A476: Render only visible list items using react-window or react-virtualized.

Q477: How do you optimize forms?  
A477: Use uncontrolled components or React Hook Form.

Q478: How do you handle expensive computations?  
A478: Use useMemo to memoize computed values.

Q479: How do you handle heavy event handlers?  
A479: Use useCallback to memoize functions and avoid re-creation.

Q480: How do you profile React performance?  
A480: Use React DevTools Profiler to identify slow components and updates.

Testing (Q481–490)

Q481: How do you unit test React components?  
A481: Use React Testing Library or Enzyme with Jest.

Q482: How do you test hooks?  
A482: Use @testing-library/react-hooks or a wrapper component.

Q483: How do you test async operations?  
A483: Use waitFor or findBy\* queries.

Q484: How do you mock API calls?  
A484: Use jest.mock or MSW (Mock Service Worker).

Q485: How do you test Redux-connected components?  
A485: Wrap components in <Provider store={store}> and assert state updates.

Q486: How do you test navigation?  
A486: Use MemoryRouter or mock router methods.

Q487: How do you test forms?  
A487: Simulate input and submit events, then assert state changes.

Q488: How do you test error boundaries?  
A488: Render a component that throws and check fallback UI.

Q489: What is snapshot testing?  
A489: Capture component output and compare with future renders.

Q490: Why is testing important?  
A490: Ensures UI correctness, prevents regressions, and improves maintainability.

SSR / Next.js Advanced (Q491–495)

Q491: How do you implement SSR caching?  
A491: Cache server-rendered HTML using CDNs or server-side caching.

Q492: How do you implement dynamic routes with SSR?  
A492: Use [param].js files and getServerSideProps for data fetching.

Q493: How do you prefetch data for SSR?  
A493: Use getServerSideProps to fetch data before rendering.

Q494: How do you handle authentication in Next.js?  
A494: Use middleware or getServerSideProps to check tokens and redirect unauthenticated users.

Q495: How do you implement on-demand ISR?  
A495: Use revalidate or Next.js on-demand revalidation API.

Deployment Best Practices (Q496–500)

Q496: How do you deploy React apps?  
A496: Use Netlify, Vercel, AWS S3, Firebase Hosting, or similar platforms.

Q497: How do you deploy Next.js apps?  
A497: Use Vercel, AWS, or Node.js servers with SSR support.

Q498: How do you manage environment variables in deployment?  
A498: Use .env.production or platform-specific environment settings.

Q499: How do you monitor production performance?  
A499: Use React DevTools Profiler, Lighthouse, or analytics tools.

Q500: How do you implement CI/CD for React/Next.js apps?  
A500: Use GitHub Actions, GitLab CI/CD, or Vercel’s automatic deployments.