















































































**Section 1: Modern JavaScript & DOM Foundations**

1. **A developer writes const user = { name: 'Alex' }; user.name = 'Bob';. What is the result?**  
   a) A TypeError is thrown because user is a constant.  
   b) The code executes successfully, and user is now { name: 'Bob' }.  
   c) A SyntaxError is thrown.  
   d) The code executes, but the user object remains unchanged.
2. **Regarding the JavaScript Event Loop, what is the execution order of Promise.resolve(), setTimeout(fn, 0), and a regular function call?**  
   a) setTimeout -> Promise -> Regular Function  
   b) Regular Function -> setTimeout -> Promise  
   c) Regular Function -> Promise -> setTimeout  
   d) Promise -> Regular Function -> setTimeout
3. **You have an array const data = [1, 2, 3]; and want to create a new array [1, 2, 3, 4]. Which method is the most idiomatic and avoids mutation?**  
   a) data.push(4);  
   b) const newData = data.concat(4);  
   c) const newData = [...data, 4];  
   d) const newData = data; newData[3] = 4;
4. **What is a key difference between let and var in terms of scoping?**  
   a) let is function-scoped, var is block-scoped.  
   b) let is block-scoped, var is function-scoped.  
   c) Both are block-scoped, but let cannot be reassigned.  
   d) Both are function-scoped, but var can be redeclared.
5. **In HTML, what is the primary purpose of the data-\* attribute?**  
   a) To style elements directly without using CSS classes.  
   b) To store private, custom data for the page or application's JavaScript.  
   c) To tell the browser to ignore the element during rendering.  
   d) It's a deprecated attribute replaced by the <data> tag.

**Section 2: Core React Concepts & Directory Structure**

1. **In a standard Create React App (CRA) project, where does the index.html file, which serves as the application's entry point, reside?**  
   a) src/  
   b) src/components/  
   c) build/  
   d) public/
2. **Why is it crucial to use keys when rendering a list of elements in React?**  
   a) It allows CSS to target specific elements in the list.  
   b) It helps React identify which items have changed, are added, or are removed, optimizing the reconciliation process.  
   c) It's a mandatory prop for all components, and the list will fail to render without it.  
   d) It prevents the use of Array.prototype.map() for rendering.
3. **A useEffect hook with an empty dependency array [] is most analogous to which combination of classic class component lifecycle methods?**  
   a) componentDidMount and componentDidUpdate  
   b) shouldComponentUpdate and componentWillUnmount  
   c) constructor only  
   d) componentDidMount and componentWillUnmount
4. **What is the correct way to pass a function from a parent component to a child to update the parent's state (a pattern known as "Lifting State Up")?**  
   a) Parent: <Child stateUpdater={this.setState} />  
   b) Parent: <Child onUpdate={() => setState(newValue)} />  
   c) Child: this.props.parent.setState(newValue)  
   d) It's impossible; a child component cannot modify its parent's state.
5. **You have a UserProfile component that displays user data fetched from an API. Where should the API call be initiated according to best practices?**  
   a) In the component's render method (or function body).  
   b) In a useEffect hook.  
   c) In the constructor of a class component.  
   d) In a separate utility file, called directly from the JSX.
6. **What is the primary difference between storing a value in useState versus useRef?**  
   a) useRef can only store DOM elements, while useState stores data.  
   b) Updating a useRef value triggers a component re-render, while useState does not.  
   c) Updating a useState value triggers a component re-render, while updating a useRef value does not.  
   d) useRef is asynchronous, while useState is synchronous.
7. **In a typical React project structure, where would you store a reusable Button.jsx component that is used across many different features?**  
   a) Directly inside the src/ folder.  
   b) In src/components/ or src/shared/components/.  
   c) Inside the folder of the first feature that uses it, e.g., src/features/login/.  
   d) In the public/ folder.
8. **What problem does the useCallback hook solve?**  
   a) It prevents a component from re-rendering.  
   b) It memoizes the result of a complex calculation.  
   c) It memoizes a function definition, preventing child components that depend on it from re-rendering unnecessarily.  
   d) It allows you to call functions outside of the React component lifecycle.
9. **When does a component using React.memo re-render?**  
   a) Never, it is memoized forever.  
   b) On every parent re-render, just like a regular component.  
   c) Only if its own state changes or its props have undergone a shallow inequality.  
   d) Only when a Redux state it is connected to changes.
10. **What is the purpose of the children prop?**  
    a) It is an array of all child components defined in the parent.  
    b) It allows you to pass components or JSX directly between the opening and closing tags of a component, e.g., <Card>Here is the content</Card>.  
    c) It is a required prop for any component that has a parent.  
    d) It is a reference to the component's internal state.

**Section 3: React Ecosystem (State Management, Routing, Styling)**

1. **In React Router v6, how do you define a dynamic route to capture a user ID, e.g., /users/123?**  
   a) <Route path="/users/\*" element={<UserProfile />} />  
   b) <Route path="/users/:id" element={<UserProfile />} />  
   c) <Route path="/users?id=" element={<UserProfile />} />  
   d) <Route path="/users/{id}" element={<UserProfile />} />
2. **What is the primary difference between using the Context API and Redux for state management?**  
   a) Redux requires a global store, while Context API does not.  
   b) Context API can cause performance issues in large apps due to re-rendering all consumers, whereas Redux has more refined subscription mechanisms.  
   c) Redux is built into React, while Context API is a third-party library.  
   d) Context API is only for theming, while Redux is for all application state.
3. **In Redux, what is the only way to modify the state?**  
   a) By directly calling reducer functions with the new state.  
   b) By dispatching an action, which is then handled by a reducer.  
   c) By using the store.setState() method.  
   d) By mutating the state object directly inside a component.
4. **You are using the axios library. What is the purpose of an "interceptor"?**  
   a) To block network requests that do not match a specific pattern.  
   b) To cache API responses automatically.  
   c) To globally catch and handle requests or responses before they are processed by .then() or .catch(), e.g., for adding auth tokens.  
   d) To intercept and change the URL of an outgoing request.
5. **Which react-redux hook is used to dispatch actions to the Redux store?**  
   a) useSelector  
   b) useStore  
   c) useDispatch  
   d) useReducer
6. **When using styled-components, how do you pass a React prop to a style rule?**  
   a) background: this.props.primary ? 'blue' : 'white';  
   b) background: ${props => props.primary ? 'blue' : 'white'};  
   c) It's not possible; props are for component logic only.  
   d) background: var(--primary-prop);
7. **What is the core philosophy of the React Testing Library (@testing-library/react)?**  
   a) To test the implementation details of a component, like its state and props.  
   b) To test components from the perspective of an end-user, interacting with the rendered DOM.  
   c) To create snapshot tests that ensure the UI never changes unexpectedly.  
   d) To test only the logic within Redux reducers.
8. **To prevent a user from accessing a route like /dashboard unless they are logged in, what is the common pattern used with React Router?**  
   a) Using a try...catch block around the <Route>.  
   b) Creating a custom ProtectedRoute component that checks for authentication and renders the component or a <Navigate> to the login page.  
   c) Adding an isAuth prop directly to the <Route> component.  
   d) Storing the route definitions on the server and only sending the allowed ones to the client.
9. **In Redux Toolkit, what is the primary purpose of createSlice?**  
   a) To split the main bundle into smaller, lazy-loadable chunks.  
   b) To automatically generate action creators and reducer logic for a specific piece of state, reducing boilerplate.  
   c) To create a memoized selector for performance.  
   d) To define the shape of the Redux store.
10. **What is a key advantage of using CSS Modules over global CSS in a React project?**  
    a) CSS Modules allow you to write CSS in JavaScript.  
    b) They automatically scope class names locally to the component, preventing style conflicts.  
    c) They are faster to load because they are inlined in the <head> tag.  
    d) They support SASS syntax by default without any configuration.

**Section 4: React Native & Mobile Development**

1. **What is the fundamental difference in how React Native renders UI compared to React for web?**  
   a) React Native uses HTML and CSS, but within a WebView.  
   b) React Native uses the same reconciliation algorithm but invokes actual native UI platform widgets (e.g., UIView, android.view.View) instead of a DOM.  
   c) React Native compiles JavaScript into native code (Swift/Kotlin) before runtime.  
   d) There is no fundamental difference; the code is identical.
2. **In React Native, what component is the equivalent of a <div> in web development for grouping content?**  
   a) <Div>  
   b) <Container>  
   c) <View>  
   d) <Section>
3. **How do you write platform-specific code for iOS and Android in React Native?**  
   a) Use if (Platform.OS === 'ios') { ... } else { ... }.  
   b) Create separate files, e.g., MyComponent.ios.js and MyComponent.android.js.  
   c) Use CSS media queries like @media (platform: android).  
   d) Both A and B are common and valid approaches.
4. **You are building an app with both a "bare" React Native CLI project and an Expo project. What is a key limitation of the Expo "managed workflow"?**  
   a) You cannot use JavaScript.  
   b) You cannot write your own custom native modules (Java/Kotlin/Swift/Objective-C) without "ejecting".  
   c) You are limited to using only class components.  
   d) It does not support over-the-air (OTA) updates.
5. **How is styling typically handled in React Native?**  
   a) By linking an external .css stylesheet.  
   b) By using JavaScript objects via the StyleSheet.create API for performance optimization.  
   c) By using inline style attributes on every component, e.g., style={{color: 'red'}}.  
   d) By using Tailwind CSS classes directly in the component.
6. **What component would you use to create a scrollable list of items in React Native?**  
   a) <ScrollView> for short lists, <FlatList> for long, performant lists.  
   b) <ListView> for all list types.  
   c) A <View> with the overflow: 'scroll' style property.  
   d) <ScrollContainer>.
7. **What does the "Bridge" in React Native architecture refer to?**  
   a) The connection between your development machine and the Metro bundler.  
   b) A design pattern for connecting components.  
   c) The asynchronous, serializable, and batched communication layer between the JavaScript thread and the native (UI) thread.  
   d) A UI component for navigating between screens.
8. **To use a device's camera or GPS, what must be configured in a React Native project?**  
   a) Nothing, these APIs are available by default.  
   b) The JavaScript code must be granted special permissions via a permissions.js file.  
   c) The native project configuration files (Info.plist for iOS, AndroidManifest.xml for Android) must be updated with permission requests.  
   d) You must use Expo; it's not possible in a bare React Native project.

**Section 5: Node.js & Express.js Backend**

1. **In a standard Express.js application, what is the purpose of the next function in middleware, e.g., (req, res, next)?**  
   a) To immediately send a response to the client.  
   b) To skip the current middleware and move to the next route handler.  
   c) To pass control to the next middleware function in the stack.  
   d) To refresh the request object.
2. **The statement const app = express(); does what?**  
   a) It starts a web server on a default port.  
   b) It creates an instance of the Express application.  
   c) It connects to a database.  
   d) It imports all middleware required for a web application.
3. **You have an Express route defined as app.post('/user', ...). How would you access the JSON body sent with the request?**  
   a) req.params.body  
   b) req.query  
   c) req.body (assuming express.json() middleware is used)  
   d) req.payload
4. **What is a key characteristic of Node.js's I/O model?**  
   a) Multi-threaded and blocking.  
   b) Single-threaded, non-blocking, and asynchronous.  
   c) Multi-threaded and non-blocking.  
   d) It creates a new process for every incoming request.
5. **What is the purpose of the .env file in a Node.js project?**  
   a) To define the environment, such as 'production' or 'development', directly.  
   b) To store environment-specific variables (like API keys and database URLs) that should not be committed to version control.  
   c) To list all the environment dependencies for npm.  
   d) It is a file that configures the Node.js runtime itself.
6. **In an Express application, the order of middleware is critical. If app.use(express.json()) is placed *after* a route that needs to parse a JSON body, what will happen?**  
   a) Express will automatically reorder the middleware for correct execution.  
   b) The route will work correctly, as middleware is applied globally.  
   c) The route handler will receive req.body as undefined because the parsing middleware has not run yet.  
   d) A 500 Internal Server Error will be thrown.
7. **How do you define an error-handling middleware in Express?**  
   a) It's a regular middleware function placed at the beginning of the stack.  
   b) It's a special middleware function with a signature of (err, req, res, next) placed at the end of the middleware stack.  
   c) By using a try...catch block in every route handler.  
   d) By calling app.onError((err, req, res, next) => { ... }).
8. **What is the difference between fs.readFile and fs.readFileSync in Node.js?**  
   a) readFile is for text files, readFileSync is for binary files.  
   b) readFile is asynchronous and uses a callback/promise; readFileSync is synchronous and blocks the event loop.  
   c) readFileSync is deprecated and should not be used.  
   d) readFile reads the entire file into memory, while readFileSync uses a stream.
9. **You want to serve static files (like images, CSS, and client-side JS) from a directory named public in your Express app. What is the correct way to do this?**  
   a) app.use(express.static('public'))  
   b) app.get('/static/\*', (req, res) => { ... read file manually ... })  
   c) app.static('public')  
   d) app.use('/public', express.static())
10. **What is Nodemon primarily used for in the Node.js development workflow?**  
    a) To monitor application performance and memory leaks.  
    b) To automatically restart the Node.js application when file changes in the directory are detected.  
    c) To manage Node.js versions.  
    d) To bundle the application for production.

**Section 6: Full-Stack Integration & Best Practices**

1. **A React frontend at http://localhost:3000 makes an API request to a Node/Express backend at http://localhost:5000. The browser blocks the request. What is the most likely cause?**  
   a) The server is offline.  
   b) The client does not have an internet connection.  
   c) Cross-Origin Resource Sharing (CORS) policy is preventing the request.  
   d) The React application has a syntax error.
2. **What is the difference between dependencies and devDependencies in a package.json file?**  
   a) There is no functional difference; it's for organization only.  
   b) dependencies are required for production, while devDependencies are only for the development environment (e.g., testing libraries, linters).  
   c) devDependencies are installed globally, while dependencies are local to the project.  
   d) dependencies are for frontend, devDependencies are for backend.
3. **What is the purpose of the proxy key in a Create React App package.json file?**  
   a) To set up a network proxy for npm install.  
   b) To redirect API requests from the React development server to a backend server, avoiding CORS issues in development.  
   c) To define a proxy component for lazy loading.  
   d) To specify a proxy for the production build.
4. **When building a React application with npm run build, what is the primary content of the resulting build directory?**  
   a) The original .jsx and .js source files.  
   b) A Node.js server that serves the application.  
   c) A set of highly optimized, static HTML, CSS, and JavaScript files that can be served by any static host.  
   d) A package.json file listing all production dependencies.
5. **In a MERN stack (MongoDB, Express, React, Node) project, where should the database connection logic typically reside?**  
   a) In a React component, using a useEffect hook.  
   b) In the main App.js file on the server.  
   c) In a dedicated configuration file (e.g., config/db.js) on the Node/Express server.  
   d) In the public/index.html file.
6. **You need to use a value like an API key in your React code, but you don't want to hardcode it. What is the standard CRA-approved method?**  
   a) Store it in localStorage and read it from there.  
   b) Create a .env file and define a variable named REACT\_APP\_API\_KEY. Access it in code via process.env.REACT\_APP\_API\_KEY.  
   c) Fetch it from a config.json file placed in the public folder.  
   d) Just define it as a constant in index.js.
7. **Which HTTP method is idempotent, meaning making the same request multiple times will produce the same result as making it once?**  
   a) POST  
   b) PUT  
   c) PATCH  
   d) Both A and B.

**ANSWER KEY**

1. **b)** const prevents reassignment of the variable user, not the mutation of the object it points to.
2. **c)** The call stack executes regular functions first. Promises go to the microtask queue, which runs before the macrotask queue (where setTimeout callbacks go).
3. **c)** The spread operator is the modern, idiomatic way to create a new array with added elements without mutating the original.
4. **b)** This is the fundamental scoping difference between the two keywords. let and const are block-scoped.
5. **b)** data-\* attributes are designed to store custom data for scripts to use, without interfering with other standard attributes.
6. **d)** The public folder contains static assets that are copied to the build folder as-is. index.html is the template.
7. **b)** Keys are a hint for React's diffing algorithm, crucial for performance and preventing bugs with stateful list items.
8. **d)** An empty array dependency means the effect runs once after the initial render (componentDidMount) and the cleanup function runs once on unmount (componentWillUnmount).
9. **b)** The parent defines a function that calls its own state setter and passes that function as a prop to the child.
10. **b)** useEffect is the correct place for side effects like data fetching. Running it in the function body would cause a fetch on every render.
11. **c)** The core distinction: useState updates cause re-renders to display the new value; useRef updates are silent and do not trigger re-renders.
12. **b)** A centralized components or shared folder is standard practice for highly reusable, presentation-only components.
13. **c)** useCallback returns a memoized version of a callback function that only changes if a dependency has changed, optimizing child components that receive it as a prop.
14. **c)** React.memo performs a shallow comparison of props. It is a performance optimization for functional components.
15. **b)** The children prop is a special prop that captures whatever is passed between the component's tags.
16. **b)** The colon (:) syntax is used in React Router to denote a URL parameter.
17. **b)** While Context is simpler, it can lead to performance issues as any state change re-renders all consuming components within that provider. Redux is more optimized for frequent updates.
18. **b)** This is the core principle of Redux: enforcing a one-way data flow through actions and pure reducer functions.
19. **c)** Interceptors are a powerful feature for handling cross-cutting concerns like authentication, logging, or error handling for all axios requests/responses.
20. **c)** useDispatch returns the store's dispatch function, allowing components to trigger state changes.
21. **b)** styled-components uses tagged template literals, and functions can be interpolated to access the component's props for dynamic styling.
22. **b)** Its guiding principle is: "The more your tests resemble the way your software is used, the more confidence they can give you."
23. **b)** This is the standard "auth guard" pattern. The ProtectedRoute acts as a gatekeeper for a specific route.
24. **b)** createSlice is a cornerstone of Redux Toolkit that drastically simplifies Redux by generating actions, action types, and reducers from a single definition.
25. **b)** By generating unique class names like Component\_className\_\_123xyz, CSS Modules ensure styles from one component don't leak and affect another.
26. **b)** This is the core architectural concept of React Native. There is no DOM; React communicates over the "Bridge" to control native UI elements.
27. **c)** <View> is the most fundamental component for building UI, acting as a container that supports layout with Flexbox.
28. **d)** Both the Platform module and platform-specific file extensions are officially supported methods for handling platform differences.
29. **b)** Expo's managed workflow provides a curated set of native APIs. If you need a native library not included in Expo SDK, you must use the bare workflow or eject.
30. **b)** The StyleSheet.create method is used to define styles in a JavaScript object. It sends the styles over the bridge only once and references them by ID, which is more performant than inline objects.
31. **a)** <ScrollView> renders all children at once. <FlatList> virtualizes the list, rendering only the items currently on screen for better performance with long lists.
32. **c)** The Bridge is the message-passing system that allows the JS thread (where your code runs) to communicate with the native UI thread (which manages the UI).
33. **c)** Permissions are a core security feature of mobile OSes and must be declared in the native configuration files.
34. **c)** The next() function is the signal to Express to move on to the next piece of middleware in the chain.
35. **b)** This line instantiates the application. The server itself is started later with app.listen().
36. **c)** The express.json() middleware parses incoming request bodies with a Content-Type of application/json and makes the result available on req.body.
37. **b)** Node.js's event-driven, non-blocking I/O model is its defining feature, allowing it to handle many concurrent connections efficiently with a single thread.
38. **b)** Used in conjunction with a library like dotenv, this file is the standard way to manage sensitive credentials and configuration outside of version control.
39. **c)** Middleware executes sequentially. If the route handler is reached before the JSON parser, req.body will not have been populated.
40. **b)** The unique four-argument signature is how Express identifies a middleware function as being specifically for error handling.
41. **b)** Using the synchronous version (readFileSync) is highly discouraged in a server context as it will block all other requests from being handled while the file is being read.
42. **a)** This is the standard and correct way to use the built-in express.static middleware.
43. **b)** Nodemon is an essential development tool that watches for file changes and automatically restarts the server, speeding up the development cycle.
44. **c)** The browser enforces the Same-Origin Policy. The backend server must explicitly allow requests from the frontend's origin via CORS headers.
45. **b)** This distinction is crucial for optimizing production builds, as it prevents development tools from being packaged and deployed.
46. **b)** The proxy feature is a development-only convenience in CRA to simplify API calls to a different origin and bypass CORS.
47. **c)** The build process transpiles, bundles, and minifies the source code into a handful of static assets ready for deployment.
48. **c)** Server configuration, including database connections, belongs on the server, typically isolated in its own module for clarity and reusability.
49. **b)** CRA has built-in support for environment variables in .env files, requiring the REACT\_APP\_ prefix to avoid accidentally exposing system variables.
50. **b)** PUT is idempotent because calling it multiple times with the same payload to the same resource will result in the same state. POST creates a new resource on each call.