# **Disclaimer:**

COPYRIGHT NOTICE

©All content, materials, and intellectual property contained within this document, including but not limited to text, images, diagrams, graphics, code, and any other original works, are protected by copyright laws and international treaties. This intellectual property is the sole and exclusive property of SRIKANTH TEKUMUDI

Unauthorized use, reproduction, distribution, modification, or transmission of any portion of this document, in any form or by any means, electronic or mechanical, without the prior written permission of SRIKANTH TEKUMUDI, is strictly prohibited and may result in severe civil and criminal penalties.

By accessing, viewing, or using this document, you acknowledge and agree to abide by all applicable copyright laws and the terms and conditions outlined herein. This copyright notice serves as a legal warning and reminder that the unauthorized use of this intellectual property is strictly prohibited and will be vigorously enforced.

Thank you for respecting the intellectual property rights of SRIKANTH TEKUMUDI

### 

**Once you've completed your interview preparation, it's important to test or revise your knowledge by answering these questions yourself. These are some of the most common theoretical questions you can expect in any frontend interview. If you're comfortable answering these, you've covered half of your preparation.**

**The remaining half involves implementation questions, where you'll need to practice polyfills, machine coding, or JavaScript implementations.**

### **Core JavaScript Concepts**

1. What are closures, and how do they work in JavaScript?
2. Can you explain hoisting and how it affects variables and functions?
3. Describe the different types of scopes in JavaScript: global, function, and block.
4. What is the execution context, and how does the call stack work in JavaScript?
5. How does the event loop work in JavaScript? Can you explain the concept of the task queue and microtasks?

### **Promises & Asynchronous JavaScript**

1. How would you create a polyfill for Promise.all, Promise.race, and Promise.any?
2. How would you implement a retry mechanism for an asynchronous operation with N retries?
3. Can you explain the differences between Promise.all, Promise.race, and Promise.any?
4. How does async/await work in JavaScript, and how does it improve the readability of asynchronous code?
5. What are the differences between microtasks and macrotasks in JavaScript?

### **Advanced Function Concepts**

1. What is currying, and how would you implement it in JavaScript?
2. Can you explain partial application and provide an example?
3. How would you create a polyfill for compose() and pipe() functions in JavaScript?
4. How do you implement a custom debounce function? How about a throttle function?
5. What is the difference between function declaration and function expression?

### **Design Patterns & Architecture**

1. What is the Singleton pattern, and how would you implement it in JavaScript?
2. How does the Factory pattern work? Can you provide a JavaScript implementation?
3. Explain the Publisher-Subscriber (Pub-Sub) pattern and how to implement it in JavaScript.
4. What is the Observer pattern, and how does it differ from the Pub-Sub pattern?
5. Can you explain the Decorator and Strategy patterns and their use cases in JavaScript?

### **Mastering Array Methods**

1. How would you create polyfills for common array methods like map, filter, reduce, and forEach?
2. What is the difference between splice and slice in JavaScript?
3. How would you implement the Array.prototype.flat method in JavaScript?
4. How do higher-order array functions improve the readability and functionality of your code?
5. Can you explain the concept of immutability when working with arrays?

### **Function Prototypes**

1. How would you implement polyfills for Function.prototype.bind, call, and apply?
2. Can you explain prototype inheritance in JavaScript with examples?
3. How does the this keyword work in different contexts (global, object, function)?
4. What is the difference between prototypal and classical inheritance?
5. How does Object.create() work, and how is it used in inheritance?

### **JavaScript in the Browser**

1. Can you explain the rendering pipeline in the browser and how JavaScript affects it?
2. What is event delegation, and how does it improve performance?
3. How do event capturing and bubbling work in the DOM?
4. What is the difference between event.preventDefault() and event.stopPropagation()?
5. How does requestAnimationFrame() work, and when should you use it?

### **Optimization Techniques**

1. How would you implement memoization in JavaScript to optimize performance?
2. What is lazy loading, and how does it help improve performance?
3. Can you explain code splitting and how it can be implemented in a JavaScript project?
4. How do you optimize loops and iterations in JavaScript?
5. What strategies can you use to reduce DOM manipulations for better performance?

### **Object Handling**

1. What is the difference between deep and shallow copies in JavaScript?
2. How would you implement a deep clone function in JavaScript?
3. What are the risks of mutating objects and how can immutability be maintained?
4. How do you handle circular references in deep cloning?
5. What are some best practices for object creation and manipulation?

### **Timers and Events**

1. How would you implement a clearAllTimeout() function in JavaScript?
2. Can you explain the event loop in JavaScript and how it manages asynchronous operations?
3. What is the difference between setTimeout and setInterval, and when would you use each?
4. How does debouncing and throttling work with timers?
5. What is the difference between microtasks and macrotasks in the context of event handling?

### **Security in JavaScript**

1. How do you prevent Cross-Site Scripting (XSS) attacks in JavaScript applications?
2. What are the best practices for consuming APIs securely in JavaScript?
3. How do you sanitize user inputs to prevent security vulnerabilities?
4. What are Content Security Policies (CSP) and how do they protect JavaScript applications?
5. How can you protect your application against CSRF (Cross-Site Request Forgery) attacks?

### **ES6+ Features**

1. What are the differences between let, const, and var?
2. How does destructuring work in JavaScript, and what are its benefits?
3. Can you explain template literals and how they differ from traditional string concatenation?
4. What are arrow functions, and how do they differ from regular functions?
5. How do JavaScript modules work, and what are the benefits of using import and export?

### **Testing in JavaScript**

1. How do you write unit tests using Jest or Mocha in JavaScript?
2. What is the difference between unit testing and integration testing?
3. How does Test-Driven Development (TDD) work, and what are its advantages?
4. What are mocking and stubbing in the context of testing?
5. How do you test asynchronous code in JavaScript?

### **Performance Optimization**

1. What are some techniques for optimizing loops and iterations in JavaScript?
2. How do you reduce DOM manipulations to improve performance?
3. How do efficient event listeners contribute to performance optimization?
4. How do you optimize rendering performance in complex applications?
5. What are the best practices for lazy loading resources?

### **Working with APIs**

1. How does the Fetch API work, and how do you handle errors?
2. What are the advantages of using async/await over traditional promise chaining when handling asynchronous data?
3. How do you parse and handle JSON responses from an API?
4. What are some common issues with CORS (Cross-Origin Resource Sharing), and how do you handle them?
5. How do you manage API rate limits and retries in JavaScript?

### **Package Management**

1. What is the difference between NPM and Yarn, and when should you use each?
2. How does semantic versioning work, and why is it important in package management?
3. What are the best practices for managing dependencies in a JavaScript project?
4. How do you update dependencies while ensuring compatibility?
5. How do you handle peer dependencies in a package?

### **Debugging Techniques**

1. What are the different console methods available in JavaScript, and how do you use them for debugging?
2. How do you debug JavaScript code using Chrome DevTools?
3. What are breakpoints, and how do you use them effectively in debugging?
4. What are some best practices for handling and logging errors in JavaScript?
5. How do you trace asynchronous code execution during debugging?

### **JavaScript and CSS**

1. What are CSS-in-JS libraries, and how do they integrate with JavaScript?
2. How do you manipulate styles using JavaScript?
3. How do you ensure responsive design using JavaScript?
4. How does window.matchMedia work, and how do you use it to handle media queries in JavaScript?
5. What are the pros and cons of using inline styles versus external stylesheets in JavaScript projects?

### **Build Tools**

1. How do you configure Webpack for a JavaScript project?
2. What is Babel, and why is it important in modern JavaScript development?
3. How do you set up ESLint and Prettier for consistent code formatting?
4. What are the benefits of using source maps in debugging?
5. How do you optimize build processes for performance in large projects?

### **Iterators and Generators**

1. How would you implement an iterator in JavaScript?
2. What are generators, and how do they differ from regular functions?
3. How do you use generators for lazy evaluation in JavaScript?
4. What are async generators, and when would you use them?
5. How do you combine iterators and generators for complex data processing tasks?

## React questions

### **1. Components**

1. What is the difference between functional and class components in React?
2. How does JSX differ from regular JavaScript?
3. Can you explain how to use props in a functional component versus a class component?

### **2. Props (Properties)**

1. How do you pass props from a parent to a child component?
2. What are default props, and how are they useful?
3. What are PropTypes, and how do they help in ensuring type safety?

### **3. State**

1. How does the useState hook work in functional components?
2. How do you manage state in class components?
3. What is immutable state, and why is it important in React?

### **4. Lifecycle Methods (Class Components)**

1. When would you use componentDidMount, and why is it important?
2. How does componentDidUpdate differ from componentWillUpdate?
3. What cleanup tasks are typically handled in componentWillUnmount?

### **5. Hooks (Functional Components)**

1. How does useState differ from state management in class components?
2. How do you use useEffect for side effects, and what are some common use cases?
3. What is the purpose of useContext, and how does it simplify state management?
4. How does useReducer compare to useState, and when would you use it?
5. What are useCallback and useMemo, and how do they optimize performance?
6. How does useRef work, and what are its typical use cases?
7. What is useImperativeHandle, and when would you use it?
8. How does useLayoutEffect differ from useEffect?

### **6. Event Handling**

1. How do you handle events in functional components?
2. What are the differences in event handling between functional and class components?

### **7. Conditional Rendering**

1. How do you implement conditional rendering using if statements?
2. What is the difference between using ternary operators and the logical && operator for conditional rendering?

### **8. Lists and Keys**

1. How do you render a list of items in React?
2. Why are keys important in React lists, and how should they be used?

### **9. Component Composition**

1. How do you reuse components effectively in React?
2. What are children props, and how do they enable component composition?
3. What is the difference between composition and inheritance in React?

### **10. Higher-Order Components (HOC)**

1. How do you create a Higher-Order Component (HOC) in React?
2. What are some common use cases for HOCs?

### **11. Render Props**

1. How do you implement the render props pattern in React?
2. What are the advantages of using render props over HOCs?

### **12. React Router**

1. How do you set up routing in a React application using BrowserRouter?
2. What is the purpose of the Route, Link, and Switch components?
3. How do you handle route parameters in React Router?

### **13. Navigation**

1. How do you navigate programmatically using the useHistory hook?
2. How does the useLocation hook help in accessing route information?

### **State Management**

### **14. Context API**

1. How do you create and consume context in React?
2. How does the useContext hook simplify accessing context values?

### **15. Redux**

1. What are actions, reducers, and the store in Redux?
2. How do you connect a React component to Redux using the connect function?

### **16. Forms**

1. How do you handle form data in React?
2. What is the difference between controlled and uncontrolled components in forms?

### **17. Side Effects**

1. How do you use useEffect for data fetching in React?
2. What is the importance of cleanup in useEffect, and how do you implement it?

### **18. AJAX Requests**

1. How do you perform AJAX requests using the Fetch API in React?
2. How does Axios differ from Fetch, and when would you use it?

### **Error Handling**

### **19. Error Boundaries**

1. How do you implement error boundaries using componentDidCatch in class components?
2. How do you create an ErrorBoundary component in functional components?

### **20. Testing**

1. How do you write unit tests for React components using Jest?
2. What is React Testing Library, and how does it differ from Enzyme?

### **21. Optimization**

1. How do you use memoization to optimize React components?
2. What tools and techniques do you use for profiling and performance monitoring in React?

### **22. Build and Deployment**

1. How do you create a React app using Create React App (CRA)?
2. What are some best practices for creating production builds?
3. How do you deploy a React application?

### **Frameworks and Libraries**

### **23. Styling Libraries**

1. How do you style components using Styled-components?
2. What are CSS Modules, and how do they differ from traditional CSS?

### **24. State Management Libraries**

1. How does Redux compare to MobX for state management in React?
2. What are some advantages of using MobX over Redux?

### **26. Routing Libraries**

1. What are the key differences between React Router and Reach Router?
2. How do you decide which routing library to use in a project?