**React.js Conceptual Interview Questions & Answers**

**Fundamental Concepts**

**1. What is React and what are its main features?**

**Expected Answer:**

* React is a JavaScript library for building user interfaces
* Key features:
  + Virtual DOM for efficient rendering
  + Component-based architecture
  + Unidirectional data flow
  + Rich ecosystem and community support
  + Server-side rendering capability
  + Mobile development support through React Native

**2. What is the difference between React and other frameworks like Angular or Vue?**

**Expected Answer:**

* React is a library, not a full framework
* Focuses solely on UI development
* More flexible with technology stack choices
* Smaller learning curve
* Better for smaller applications and gradual adoption
* Needs additional libraries for full application architecture

**3. Explain the Virtual DOM and its benefits.**

**Expected Answer:**

* Virtual DOM is a lightweight copy of the actual DOM
* When state changes:
  + Creates a new Virtual DOM tree
  + Compares with previous Virtual DOM
  + Calculates minimal number of updates needed
* Benefits:
  + Better performance
  + Cross-platform compatibility
  + Simplified programming model

**4. What is JSX and why do we use it?**

**Expected Answer:**

* JSX is a syntax extension for JavaScript
* Allows writing HTML-like code in JavaScript
* Benefits:
  + More intuitive for developers
  + Compile-time errors catch
  + Type safety
  + Easier template creation
  + Better developer experience

**Component Concepts**

**5. What is the difference between functional and class components?**

**Expected Answer:**

* Functional Components:
  + Simpler syntax
  + Use hooks for state and lifecycle
  + Easier to test and debug
  + Better performance
  + Modern approach
* Class Components:
  + Use ES6 classes
  + Have lifecycle methods
  + Can use this keyword
  + Traditional approach
  + More verbose

**6. What are Pure Components?**

**Expected Answer:**

* Pure Components implement shouldComponentUpdate automatically
* Only re-render if props or state change
* Perform shallow comparison
* Help optimize performance
* Should be used with immutable data structures

**7. What are Higher-Order Components (HOC)?**

**Expected Answer:**

* Function that takes a component and returns a new component
* Used for code reuse and abstraction
* Common uses:
  + Authentication
  + Styling
  + Data fetching
  + Props manipulation
* Example: withRouter from React Router

**State and Props**

**8. Explain the difference between state and props.**

**Expected Answer:**

* Props:
  + Read-only
  + Passed from parent to child
  + Cannot be modified by component
  + Used for component configuration
* State:
  + Managed within component
  + Can be changed using setState/useState
  + Asynchronous
  + Triggers re-render when modified

**9. What is prop drilling and how can you avoid it?**

**Expected Answer:**

* Prop drilling is passing props through multiple levels
* Solutions:
  + Context API
  + Redux or other state management
  + Component composition
  + Render props pattern

**10. What is the Context API used for?**

**Expected Answer:**

* Provides way to pass data through component tree
* Avoids prop drilling
* Use cases:
  + Theme data
  + User authentication
  + Language preferences
  + Global state

**Hooks**

**11. What are React Hooks and why were they introduced?**

**Expected Answer:**

* Functions that allow using state and lifecycle features in functional components
* Introduced to:
  + Reuse stateful logic
  + Reduce complexity
  + Avoid class confusion
  + Avoid this binding issues
  + Make code more readable

**12. Explain the differences between useEffect and useLayoutEffect.**

**Expected Answer:**

* useEffect:
  + Runs asynchronously after render
  + Doesn't block browser painting
  + Used for most side effects
* useLayoutEffect:
  + Runs synchronously before browser paint
  + Blocks visual updates
  + Used for DOM measurements

**13. What is the purpose of useMemo and useCallback?**

**Expected Answer:**

* useMemo:
  + Memoizes computed values
  + Prevents expensive calculations
  + Performance optimization
* useCallback:
  + Memoizes functions
  + Prevents unnecessary re-renders
  + Useful for performance optimization

**Performance**

**14. How can you optimize React application performance?**

**Expected Answer:**

* Use React.memo for component memoization
* Implement lazy loading
* Use proper key props
* Avoid inline function definitions
* Use production builds
* Implement code splitting
* Use virtual scrolling for long lists

**15. What is code splitting in React?**

**Expected Answer:**

* Splits code into smaller chunks
* Loads components on demand
* Implemented using:
  + React.lazy()
  + Suspense
  + Dynamic imports
* Improves initial load time

**State Management**

**16. When would you choose Redux over Context API?**

**Expected Answer:**

* Redux advantages:
  + Better for complex state
  + Development tools
  + Middleware support
  + Time-travel debugging
  + Large ecosystem
* Context API advantages:
  + Simpler setup
  + Built into React
  + Sufficient for small-medium apps
  + Less boilerplate

**17. What is the flux pattern?**

**Expected Answer:**

* Architectural pattern for managing data flow
* Components:
  + Actions
  + Dispatcher
  + Stores
  + Views
* Unidirectional data flow
* Predictable state management

**Error Handling**

**18. How do you handle errors in React?**

**Expected Answer:**

* Error Boundaries
* try-catch blocks
* Error monitoring services
* Fallback UI
* componentDidCatch lifecycle method

**Testing**

**19. What are the different types of testing in React?**

**Expected Answer:**

* Unit Testing
* Integration Testing
* End-to-End Testing
* Snapshot Testing
* Tools used:
  + Jest
  + React Testing Library
  + Cypress
  + Enzyme

**React Router**

**20. Explain the core concepts of React Router.**

**Expected Answer:**

* BrowserRouter vs HashRouter
* Route matching
* Dynamic routing
* Nested routes
* Route parameters
* Navigation guards
* History API integration

**Interview Assessment Criteria**

Rate candidate's understanding of:

1. Core React concepts
2. Modern React practices
3. Performance implications
4. State management
5. Error handling
6. Best practices
7. Real-world application

Look for:

* Clear explanation ability
* Understanding of tradeoffs
* Real-world experience examples
* Problem-solving approach
* Best practices awareness
* Architecture understanding