**Overview**

Skill Type

Advance

Skill Category

Front-End

Skill Name

React

Skill Tag

React-Adv #react,#advance

Short Description

Advanced React developers have a deep understanding of React's core concepts and are proficient in building complex applications. They possess expertise in state management, performance optimization, server-side rendering, and advanced architectural patterns. They are familiar with advanced React features, such as portals, context API, error boundaries, and custom hooks. Advanced React developers also have knowledge of advanced testing techniques, security considerations, and deployment strategies.

Objective

After acquiring advanced React skills, the developer will be capable of architecting large-scale applications, optimizing performance, implementing advanced state management patterns, and writing efficient and testable code. They will also have a solid understanding of security best practices and deployment strategies.

Content

|  |  |  |
| --- | --- | --- |
| Topic | Topic Description | Expectation |
| Advanced React Patterns | The developer should be proficient in using advanced React patterns and be familiar with:  - Higher-order components (HOCs) and their usage  - Render props pattern  - Compound components  - Controlled vs. uncontrolled components  - Error boundaries and error handling  - Context API advanced usage  - Using portals for modals or overlays | - Implement higher-order components (HOCs) to reuse component logic and enhance component behaviour  - Utilize the render props pattern to share code between components and pass data or functionality as props  - Design compound components to create a more flexible and cohesive API for component composition  - Understand the difference between controlled and uncontrolled components and use the appropriate approach in different scenarios  - Implement error boundaries to catch and handle errors in React components gracefully  - Leverage advanced features of the Context API to manage state or share data across components effectively  - Use portals to render components outside the normal component hierarchy, such as modals or overlays. |
| Performance Optimization in React | The developer should have a deep understanding of performance optimization techniques in React applications and be familiar with:  - Identifying and optimizing unnecessary re-renders  - Memoization and memo components  - Using React.PureComponent and React.memo  - Virtualization for rendering large lists  - Lazy loading and code splitting  - Performance profiling and optimization tools (e.g., React DevTools, performance timeline) | - Identify and optimize unnecessary re-renders by using shouldComponentUpdate or React.memo to prevent unnecessary component updates  - Apply memoization techniques to memoize expensive computations or data fetching operations  - Utilize React.PureComponent or React.memo to optimize functional components by avoiding unnecessary re-renders  - Implement virtualization techniques like windowing or pagination to efficiently render large lists or tables  - Implement lazy loading and code splitting to improve initial load times and reduce bundle sizes  - Use performance profiling and optimization tools like React DevTools and performance timeline to identify performance bottlenecks and optimize React applications |
| Server-Side Rendering (SSR) with React | The developer should understand the concepts of server-side rendering (SSR) and be familiar with:  - Setting up a server-side rendering environment  - React's server-side rendering API  - Handling data fetching and hydration on the server  - Handling routing and navigation in an SSR application  - Optimizing performance and caching strategies for SSR | - Configure and set up a server-side rendering environment for a React application  - Utilize React's server-side rendering API to render React components on the server  - Implement data fetching and hydration on the server to provide pre-rendered content to clients  - Handle routing and navigation in a server-side rendered application, ensuring proper server-side rendering for each route  - Implement performance optimization and caching strategies specific to server-side rendering to improve application performance |
| Testing React Applications | The developer should be proficient in testing React applications and be familiar with:   * Testing frameworks like Jest and React Testing Library * Writing unit tests for React components * Mocking dependencies and simulating user interactions in tests * Testing asynchronous code (e.g., API calls, promises) * Snapshot testing and component rendering assertions * Integration and end-to-end testing with tools like Cypress | - Set up testing frameworks like Jest and React Testing Library for testing React applications  - Write unit tests for React components, including assertions for rendering, behavior, and state changes  - Mock dependencies and simulate user interactions (e.g., clicks, form submissions) to test component interactions  - Test asynchronous code in React applications, such as API calls or promises, using appropriate testing techniques  - Use snapshot testing to ensure component rendering consistency and prevent unintended UI changes  - Perform integration and end-to-end testing using tools like Cypress to validate the behavior of React applications across multiple components and interactions |
| React State Management Libraries | The developer should be familiar with different state management solutions in React and their usage, including:   * Redux and its core concepts (actions, reducers, store) * React Context API for managing global state * MobX and its observable and computed values * Recoil for managing state with atoms and selectors * Comparison of different state management libraries and their suitability for different use cases | - Implement state management using Redux, including defining actions, reducers, and a central store for managing application state  - Utilize the React Context API to manage global state and share data between components efficiently  - Apply MobX to manage state using observable and computed values, simplifying the development of reactive components  - Utilize Recoil to manage state with atoms and selectors, leveraging a more flexible and granular approach to state management  - Compare different state management libraries and evaluate their suitability for different project requirements and complexity levels |
| React Next.js | The developer should be familiar with Next.js framework for building web applications. - Next.js Project Structure  - Routing Fundamentals  - Rendering  - Data Fetching  - Styling  - Configuring various eg. with TypeScript, ESlint, and more  - Deploying | * Must be able to build application using Next.js * Should understand Next application flow * handle routing in Next.js and know about the routing terminology * Know about the Component-level Client and Server Rendering * Should be aware with styling approaches * Should know about the built-in optimizations designed to improve your application's speed * Should know the integrations with TypeScript, ESlint, and more, as well as internal configuration options such as Absolute Imports and Environment Variables. * Should have the knowledge of Deploying NextJS application |