greatfrontend / **top-reactjs-interview-questions** Public

Most important React.js interview questions for busy Front End Engineers (updated for 2025)

www.greatfrontend.com/questions/react-interview-questions/quiz?gnrs=github

3.1k stars 83 forks Branches Tags Activity

Star Notifications

Code Issues 1 Pull requests Actions Projects Security Insights

main 2 Branches 0 Tags Go to file Go to file Code

github-actions[bot] [auto] regenerate table of contents a3033be · last month

.github/workflows	Initial commit	last year
__template__/todo-change-me	qns: add answers	last year
data	qns: scaffold questions	last year
questions	fix(typo): Fix typo issues in questions (#8)	3 months ago
scripts	Add www. to greatfrontend.com links	last month
.gitignore	i18n: add langnostic@0.0.5	4 months ago
.prettierrc	Initial commit	last year
README.md	[auto] regenerate table of contents	last month
langnostic.config.ts	i18n: add langnostic@0.0.5	4 months ago
package.json	i18n: langnostic@0.0.8	3 months ago
pnpm-lock.yaml	i18n: langnostic@0.0.8	3 months ago

Top React.js Interview Questions (Updated for 2025)

Curated top React.js interview questions with high quality answers for acing your front end interviews.

Table of Contents

No.	Questions
7	What is the difference between controlled and uncontrolled React Components?
8	What are some pitfalls about using context in React?
9	What are the benefits of using hooks in React?
10	What are the rules of React hooks?
11	What is the difference between `useEffect` and `useLayoutEffect` in React?
12	What is the purpose of callback function argument format of `setState()` in React and when should it be used?
13	What does the dependency array of `useEffect` affect?
14	What is the `useRef` hook in React and when should it be used?
15	What is the `useCallback` hook in React and when should it be used?
16	What is the `useMemo` hook in React and when should it be used?
17	What is the `useReducer` hook in React and when should it be used?
18	What is the `useId` hook in React and when should it be used?
19	What does re-rendering mean in React?
20	What are React Fragments used for?
21	What is `forwardRef()` in React used for?
22	How do you reset a component's state in React?
23	Why does React recommend against mutating state?
24	What are error boundaries in React for?
25	How do you test React applications?
26	Explain what React hydration is
27	What are React Portals used for?
28	How do you debug React applications?
29	What is React strict mode and what are its benefits?
30	How do you localize React applications?
31	What is code splitting in a React application?
32	How would one optimize the performance of React contexts to reduce rerenders?
33	What are higher order components in React?
34	What is the Flux pattern and what are its benefits?
35	Explain one-way data flow of React and its benefits
36	How do you handle asynchronous data loading in React applications?
37	Explain server-side rendering of React applications and its benefits?
38	Explain static generation of React applications and its benefits?
39	Explain the presentational vs container component pattern in React
40	What are some common pitfalls when doing data fetching in React?
41	What are render props in React and what are they for?
42	What are some React anti-patterns?

No.	Questions
43	How do you decide between using React state, context, and external state managers?
44	Explain the composition pattern in React
45	What is virtual DOM in React?
46	How does virtual DOM in React work? What are its benefits and downsides?
47	What is React Fiber and how is it an improvement over the previous approach?
48	What is reconciliation in React?
49	What is React Suspense and what does it enable?
50	Explain what happens when the `useState` setter function is called in React

Questions with answers

1. ### What is React? Describe the benefits of React

   React is a JavaScript library created by Facebook for building user interfaces, primarily for single-page applications. It allows developers to create reusable components that manage their own state. Key benefits of React include a component-based architecture for modular code, the virtual DOM for efficient updates, a declarative UI for more readable code, one-way data binding for predictable data flow, and a strong community and ecosystem with abundant resources and tools.

   Key characteristics of React:

   - **Declarative**: You describe the desired state of your UI based on data, and React handles updating the actual DOM efficiently.
   - **Component-based**: Build reusable and modular UI elements (components) that manage their own state and logic.
   - **Virtual DOM**: React uses a lightweight in-memory representation of the actual DOM, allowing it to perform updates selectively and efficiently.
   - **JSX**: While not mandatory, JSX provides a syntax extension that allows you to write HTML-like structures within your JavaScript code, making UI development more intuitive.

   > Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

   Back to top ↑

2. ### What is the difference between React Node, React Element, and a React Component?

   A React Node is any renderable unit in React, such as an element, string, number, or `null`. A React Element is an immutable object describing what to render, created using JSX or `React.createElement`. A React Component is a function or class that returns React Elements, enabling the creation of reusable UI pieces.

   > Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

   Back to top ↑

3. ### What is JSX and how does it work?

JSX stands for JavaScript XML. It is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. JSX makes it easier to create React components by allowing you to write what looks like HTML directly in your JavaScript code. Under the hood, JSX is transformed into JavaScript function calls, typically using a tool like Babel. For example, `<div>Hello, world!</div>` in JSX is transformed into `React.createElement('div', null, 'Hello, world!')`.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

4. ### What is the difference between state and props in React?

In React, `state` is a local data storage that is managed within a component and can change over time, while `props` are read-only attributes passed from a parent component to a child component. State is used for data that changes within a component, whereas props are used to pass data and event handlers to child components.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

5. ### What is the purpose of the `key` prop in React?

The `key` prop in React is used to uniquely identify elements in a list. It helps React optimize rendering by efficiently updating and reordering elements. Without a unique `key`, React may re-render elements unnecessarily, leading to performance issues and bugs.

```
{
  items.map((item) => <ListItem key={item.id} value={item.value} />);
}
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

6. ### What is the consequence of using array indices as the value for `key`s in React?

Using array indices as the value for `key`s in React can lead to performance issues and bugs. When the order of items changes, React may not correctly identify which items have changed, leading to unnecessary re-renders or incorrect component updates. It's better to use unique identifiers for `key`s to ensure React can efficiently manage and update the DOM.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

7. ### What is the difference between controlled and uncontrolled React Components?

Controlled components in React are those where the form data is handled by the component's state. The state is the single source of truth, and any changes to the form input are managed through event handlers. Uncontrolled components, on the other hand, store their own state internally and rely on refs to access the form values. Controlled components offer more control and are easier to test, while uncontrolled components can be simpler to implement for basic use cases.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

8. ### What are some pitfalls about using context in React?

Using context in React can lead to performance issues if not managed properly. It can cause unnecessary re-renders of components that consume the context, even if the part of the context they use hasn't changed. Additionally, overusing context for state management can make the code harder to understand and maintain. It's important to use context sparingly and consider other state management solutions like Redux or Zustand for more complex state needs.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

9. ### What are the benefits of using hooks in React?

Hooks in React allow you to use state and other React features without writing a class. They make it easier to reuse stateful logic between components, improve code readability, and simplify the codebase by reducing the need for lifecycle methods. Hooks like `useState` and `useEffect` are commonly used to manage state and side effects in functional components.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

10. ### What are the rules of React hooks?

React hooks have a few essential rules to ensure they work correctly. Always call hooks at the top level of your React function, never inside loops, conditions, or nested functions. Only call hooks from React function components or custom hooks. These rules ensure that hooks maintain the correct state and lifecycle behavior.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

11. ### What is the difference between `useEffect` and `useLayoutEffect` in React?

`useEffect` and `useLayoutEffect` are React hooks used to handle side effects in functional components, but they differ in timing and use cases:

- `useEffect` : Runs asynchronously after the DOM has been painted. It is suitable for tasks like data fetching, subscriptions, or logging.
- `useLayoutEffect` : Runs synchronously after DOM mutations but before the browser paints. Use it for tasks like measuring DOM elements or synchronizing the UI with the DOM.

Code example:

```
import React, { useEffect, useLayoutEffect, useRef } from 'react';

function Example() {
  const ref = useRef();

  useEffect(() => {
    console.log('useEffect: Runs after DOM paint');
  });

  useLayoutEffect(() => {
    console.log('useLayoutEffect: Runs before DOM paint');
    console.log('Element width:', ref.current.offsetWidth);
  });

  return <div ref={ref}>Hello</div>;
}
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

12. ## What is the purpose of callback function argument format of `setState()` in React and when should it be used?

The callback function argument format of `setState()` in React is used to ensure that state updates are based on the most recent state and props. This is particularly important when the new state depends on the previous state. Instead of passing an object directly to `setState()`, you pass a function that takes the previous state and props as arguments and returns the new state.

```
this.setState((prevState, props) => ({
  counter: prevState.counter + props.increment,
}));
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

13. ## What does the dependency array of `useEffect` affect?

The dependency array of `useEffect` determines when the effect should re-run. If the array is empty, the effect runs only once after the initial render. If it contains variables, the effect runs whenever any of those variables change. If omitted, the effect runs after every render.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

14. What is the `useRef` hook in React and when should it be used?

README

and manipulate DOM elements directly, store mutable values that do not cause re-renders when updated, and keep a reference to a value without triggering a re-render. For example, you can use `useRef` to focus an input element:

```jsx
import React, { useRef, useEffect } from 'react';

function TextInputWithFocusButton() {
  const inputEl = useRef(null);

  useEffect(() => {
    inputEl.current.focus();
  }, []);

  return <input ref={inputEl} type="text" />;
}
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

15. What is the `useCallback` hook in React and when should it be used?

The `useCallback` hook in React is used to memoize functions, preventing them from being recreated on every render. This is particularly useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders. You should use `useCallback` when you have a function that is passed as a prop to a child component and you want to avoid the child component re-rendering unnecessarily.

```jsx
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]);
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

16. What is the `useMemo` hook in React and when should it be used?

The `useMemo` hook in React is used to memoize expensive calculations so that they are only recomputed when one of the dependencies has changed. This can improve performance by avoiding unnecessary recalculations. You should use `useMemo` when you have a computationally expensive function that doesn't need to run on every render.

```jsx
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

17. ### What is the `useReducer` hook in React and when should it be used?

The `useReducer` hook in React is used for managing complex state logic in functional components. It is an alternative to `useState` and is particularly useful when the state has multiple sub-values or when the next state depends on the previous one. It takes a reducer function and an initial state as arguments and returns the current state and a dispatch function.

```
const [state, dispatch] = useReducer(reducer, initialState);
```

Use `useReducer` when you have complex state logic that involves multiple sub-values or when the next state depends on the previous state.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

18. ### What is the `useId` hook in React and when should it be used?

The `useId` hook in React is used to generate unique IDs for elements within a component. This is particularly useful for accessibility purposes, such as linking form inputs with their labels. It ensures that IDs are unique across the entire application, even if the component is rendered multiple times.

```
import { useId } from 'react';

function MyComponent() {
  const id = useId();
  return (
    <div>
      <label htmlFor={id}>Name:</label>
      <input id={id} type="text" />
    </div>
  );
}
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

19. ### What does re-rendering mean in React?

Re-rendering in React refers to the process where a component updates its output to the DOM in response to changes in state or props. When a component's state or props change, React triggers a re-render to ensure the UI reflects the latest data. This process involves calling the component's render method again to produce a new virtual DOM, which is then compared to the previous virtual DOM to determine the minimal set of changes needed to update the actual DOM.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

20. ### What are React Fragments used for?

React Fragments are used to group multiple elements without adding extra nodes to the DOM. This is useful when you want to return multiple elements from a component's render method without wrapping them in an additional HTML element. You can use the shorthand syntax `<>...</>` or the `React.Fragment` syntax.

```
return (
  <>
    <ChildComponent1 />
    <ChildComponent2 />
  </>
);
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

21. What is `forwardRef()` in React used for?

`forwardRef()` in React is used to pass a ref through a component to one of its child components. This is useful when you need to access a DOM element or a child component's instance directly from a parent component. You wrap your functional component with `forwardRef()` and use the `ref` parameter to forward the ref to the desired child element.

```
import React, { forwardRef } from 'react';

const MyComponent = forwardRef((props, ref) => <input ref={ref} {...props} />);
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

22. How do you reset a component's state in React?

To reset a component's state in React, you can set the state back to its initial value. This can be done by defining an initial state and then using the `setState` function to reset it. For example, if you have a state object like this:

```
const [state, setState] = useState(initialState);
```

You can reset it by calling:

```
setState(initialState);
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

23. Why does React recommend against mutating state?

React recommends against mutating state because it can lead to unexpected behavior and bugs. React relies on state immutability to efficiently determine when to re-render components. When state is mutated directly, React may not detect the changes, leading to stale or incorrect UI updates. Instead, always create a new state object using methods like `setState` or the `useState` hook.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

24. What are error boundaries in React for?

Error boundaries in React are components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of crashing the whole application. They are implemented using the `componentDidCatch` lifecycle method and the `static getDerivedStateFromError` method. Error boundaries do not catch errors inside event handlers, asynchronous code, or server-side rendering.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

25. How do you test React applications?

To test React applications, you can use tools like Jest and React Testing Library. Jest is a JavaScript testing framework that works well with React, and React Testing Library provides utilities to test React components in a way that resembles how users interact with them. You can write unit tests for individual components, integration tests for component interactions, and end-to-end tests using tools like Cypress.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

26. Explain what React hydration is

React hydration is the process of attaching event listeners and making a server-rendered HTML page interactive on the client side. When a React application is server-side rendered, the HTML is sent to the client, and React takes over to make it dynamic by attaching event handlers and initializing state. This process is called hydration.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

27. What are React Portals used for?

React Portals are used to render children into a DOM node that exists outside the hierarchy of the parent component. This is useful for scenarios like modals, tooltips, and dropdowns where you need to break out of the parent component's overflow or z-index constraints. You can create a portal using `ReactDOM.createPortal(child, container)`.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

28. How do you debug React applications?

To debug React applications, you can use the React Developer Tools browser extension to inspect component hierarchies and state. Additionally, you can use `console.log` statements to log data and errors, and leverage breakpoints in your code using browser developer tools. For more advanced debugging, you can use error boundaries to catch and handle errors in your components.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

29. What is React strict mode and what are its benefits?

React strict mode is a development tool that helps identify potential problems in an application. It activates additional checks and warnings for its descendants. It doesn't render any visible UI and doesn't affect the production build. The benefits include identifying unsafe lifecycle methods, warning about legacy string ref API usage, detecting unexpected side effects, and ensuring that components are resilient to future changes.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

30. How do you localize React applications?

To localize a React application, you typically use a library like `react-i18next` or `react-intl`. First, you set up your translation files for different languages. Then, you configure the localization library in your React app. Finally, you use the provided hooks or components to display localized text in your components.

```
// Example using react-i18next
import { useTranslation } from 'react-i18next';

const MyComponent = () => {
  const { t } = useTranslation();
  return <p>{t('welcome_message')}</p>;
};
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

31. What is code splitting in a React application?

Code splitting in a React application is a technique used to improve performance by splitting the code into smaller chunks that can be loaded on demand. This helps in reducing the initial load time of the application. You can achieve code splitting using dynamic `import()` statements or React's `React.lazy` and `Suspense`.

```
// Using React.lazy and Suspense
const LazyComponent = React.lazy(() => import('./LazyComponent'));

function App() {
  return (
    <React.Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </React.Suspense>
  );
}
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

32. How would one optimize the performance of React contexts to reduce rerenders?

To optimize the performance of React contexts and reduce rerenders, you can use techniques such as memoizing context values, splitting contexts, and using selectors. Memoizing context values with `useMemo` ensures that the context value only changes when its dependencies change. Splitting contexts allows you to isolate state changes to specific parts of your application. Using selectors with libraries like `use-context-selector` can help you only rerender components that actually need the updated context value.

```
const value = useMemo(() => ({ state, dispatch }), [state, dispatch]);
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

33. What are higher order components in React?

Higher order components (HOCs) in React are functions that take a component and return a new component with additional props or behavior. They are used to reuse component logic. For example, if you have a component `MyComponent`, you can create an HOC like this:

```
const withExtraProps = (WrappedComponent) => {
  return (props) => <WrappedComponent {...props} extraProp="value" />;
};

const EnhancedComponent = withExtraProps(MyComponent);
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

34. What is the Flux pattern and what are its benefits?

The Flux pattern is an architectural design used for managing state in applications, particularly in React ecosystems. It enforces a unidirectional data flow, making it easier to manage and debug application state.

- **Core components**:

- **Dispatcher**: Manages actions and dispatches them to stores.
- **Stores**: Hold the state and logic of the application.
- **Actions**: Payloads of information sent from the application to the dispatcher.
- **View**: React components that re-render when stores update.

- **Benefits**:
  - Predictable state management due to unidirectional data flow.
  - Improved debugging and testing.
  - Clear separation of concerns.

Example flow:

i. User interacts with the **View**.
ii. **Actions** are triggered and dispatched by the **Dispatcher**.
iii. **Stores** process the actions and update their state.
iv. **View** re-renders based on the updated state.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

35. Explain one-way data flow of React and its benefits

In React, one-way data flow means that data in a React application flows in a single direction, from parent components to child components. This makes the data flow predictable and easier to debug. The main benefits include improved maintainability, easier debugging, and better performance.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

36. How do you handle asynchronous data loading in React applications?

In React applications, asynchronous data loading is typically handled using `useEffect` and `useState` hooks. You initiate the data fetch inside `useEffect` and update the state with the fetched data. This ensures that the component re-renders with the new data. Here's a simple example:

```
import React, { useState, useEffect } from 'react';

function DataFetchingComponent() {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    async function fetchData() {
      const response = await fetch('https://api.example.com/data');
      const result = await response.json();
      setData(result);
      setLoading(false);
    }

    fetchData();
  }, []);

  if (loading) {
    return <div>Loading...</div>;
  }
```

```
    return <div>{JSON.stringify(data)}</div>;
  }
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

37. Explain server-side rendering of React applications and its benefits?

Server-side rendering (SSR) in React involves rendering React components on the server and sending the fully rendered HTML to the client. This approach improves initial load times and SEO. The server handles the initial rendering, and the client takes over with React's hydration process. Benefits include faster initial page loads, better SEO, and improved performance on slower devices.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

38. Explain static generation of React applications and its benefits?

Static generation in React applications involves pre-rendering the HTML at build time, rather than on each request. This results in faster load times and better performance since the HTML is already generated and can be served directly from a CDN. It also improves SEO and can reduce server load. Tools like Next.js facilitate static generation by allowing developers to generate static pages easily.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

39. Explain the presentational vs container component pattern in React

In React, the presentational vs container component pattern is a design approach where presentational components focus on how things look and container components focus on how things work. Presentational components are concerned with rendering HTML and CSS, while container components handle the logic and state management. This separation helps in maintaining a clean and organized codebase.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

40. What are some common pitfalls when doing data fetching in React?

Common pitfalls when doing data fetching in React include not handling loading and error states, causing memory leaks by not cleaning up subscriptions, and not using the right lifecycle methods or hooks. Always ensure you handle these states properly, clean up after your components, and use `useEffect` for side effects in functional components.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

41. What are render props in React and what are they for?

Render props in React are a technique for sharing code between components using a prop whose value is a function. This function returns a React element and allows you to pass data to the child component. It helps in reusing component logic without using higher-order components or hooks.

```
class DataFetcher extends React.Component {
  state = { data: null };

  componentDidMount() {
    fetch(this.props.url)
      .then((response) => response.json())
      .then((data) => this.setState({ data }));
  }

  render() {
    return this.props.render(this.state.data);
  }
}

// Usage
<DataFetcher
  url="/api/data"
  render={(data) => <div>{data ? data.name : 'Loading...'}</div>}
/>;
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

42. What are some React anti-patterns?

React anti-patterns are practices that can lead to inefficient, hard-to-maintain, or buggy code. Some common anti-patterns include:

- Mutating state directly instead of using `setState`
- Using `componentWillMount` for data fetching
- Overusing `componentWillReceiveProps`
- Not using keys in lists
- Overusing inline functions in render
- Deeply nested state

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

43. How do you decide between using React state, context, and external state managers?

Choosing between React state, context, and external state managers depends on the complexity and scope of your application's state management needs. Use React state for local component state, React context for global state that needs to be shared across multiple components, and external state managers like Redux or MobX for complex state management that requires advanced features like middleware, time-travel debugging, or when the state needs to be shared across a large application.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

44. Explain the composition pattern in React

The composition pattern in React is a way to build components by combining smaller, reusable components. Instead of using inheritance, React encourages composition to create complex UIs. You can pass components as children or props to other components to achieve this. For example:

```
function WelcomeDialog() {
  return (
    <Dialog>
      <h1>Welcome</h1>
      <p>Thank you for visiting our spacecraft!</p>
    </Dialog>
  );
}

function Dialog(props) {
  return <div className="dialog">{props.children}</div>;
}
```

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

45. What is virtual DOM in React?

The virtual DOM in React is a lightweight copy of the actual DOM. It allows React to efficiently update the UI by comparing the virtual DOM with the real DOM and only making necessary changes. This process is called reconciliation. By using the virtual DOM, React minimizes direct manipulation of the real DOM, which can be slow and inefficient.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

46. How does virtual DOM in React work? What are its benefits and downsides?

The virtual DOM in React is a lightweight copy of the actual DOM. When the state of a component changes, React creates a new virtual DOM tree and compares it with the previous one using a process called "reconciliation." Only the differences are then updated in the actual DOM, making updates more efficient. The benefits include improved performance and a more declarative way to manage UI. However, it can add complexity and may not be as performant for very simple applications.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

47. What is React Fiber and how is it an improvement over the previous approach?

React Fiber is a complete rewrite of React's reconciliation algorithm, introduced in React 16. It improves the rendering process by breaking down rendering work into smaller units, allowing React to pause and resume work, which makes the UI more responsive. This approach enables features like time slicing and suspense, which were not possible with the previous stack-based algorithm.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

48. What is reconciliation in React?

Reconciliation in React is the process through which React updates the DOM to match the virtual DOM. When a component's state or props change, React creates a new virtual DOM tree and compares it with the previous one. This comparison process is called "diffing." React then updates only the parts of the actual DOM that have changed, making the updates efficient and fast.

> Read the detailed answer on GreatFrontEnd which allows progress tracking, contains more code samples, and useful resources.

Back to top ↑

49. What is React Suspense and what does it enable?

React Suspense is a feature that allows you to handle asynchronous operations in your React components more gracefully. It enables you to show fallback content while waiting for something to load, such as data fetching or code splitting. You can use it with `React.lazy` for code splitting and with libraries like `react-query` for data fetching.

```
const LazyComponent = React.lazy(() => import('./LazyComponent'));

function MyComponent() {
  return (
    <React.Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </React.Suspense>
  );
}
```

Contributors 9

Languages

- **MDX** 97.7%
- **TypeScript** 2.3%