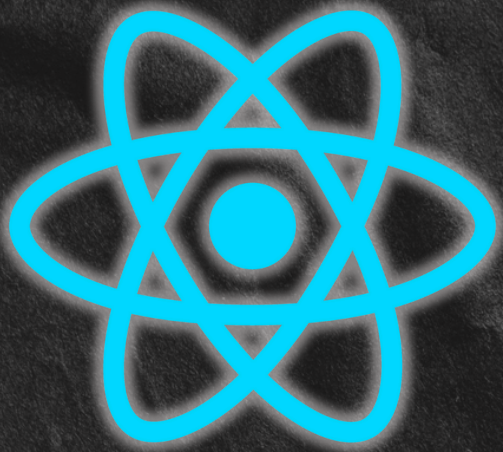


100 QUESTIONS TO ACE THE NEXT REACT INTERVIEW

MOHIT DECODES

@MohitDecodes

The Reacting Point: 100 questions to ace the next React Interview

What is The Reacting Point?

The Reacting Point is a **comprehensive** ebook that is designed to help job seekers and developers brush up on their React knowledge before an interview.

Our goal is to cover the most frequent questions that pop-up on React interviews. In addition to that, we are providing the answers that are deep-enough to understand. We are also providing the resources that will help you dive even deeper.

Ths eBook will evolve as we grow, as technology changes and with your feedback. This is the **first version** of this eBook.

As a **bonus**, at the end of this eBook you will find links to resources where you will be able to find remote React jobs.

We hope that this eBook will bring you value!

Thank you!

1. Advantages and Disadvantages of React

Advantages of React include its ability to handle complex user interfaces, its flexibility and reusability of components, and its performance optimization through the use of a virtual DOM. Disadvantages include a steep learning curve for beginners and the need for additional libraries or tools to complete a full-featured application.

2. What Does DOM Stand For?

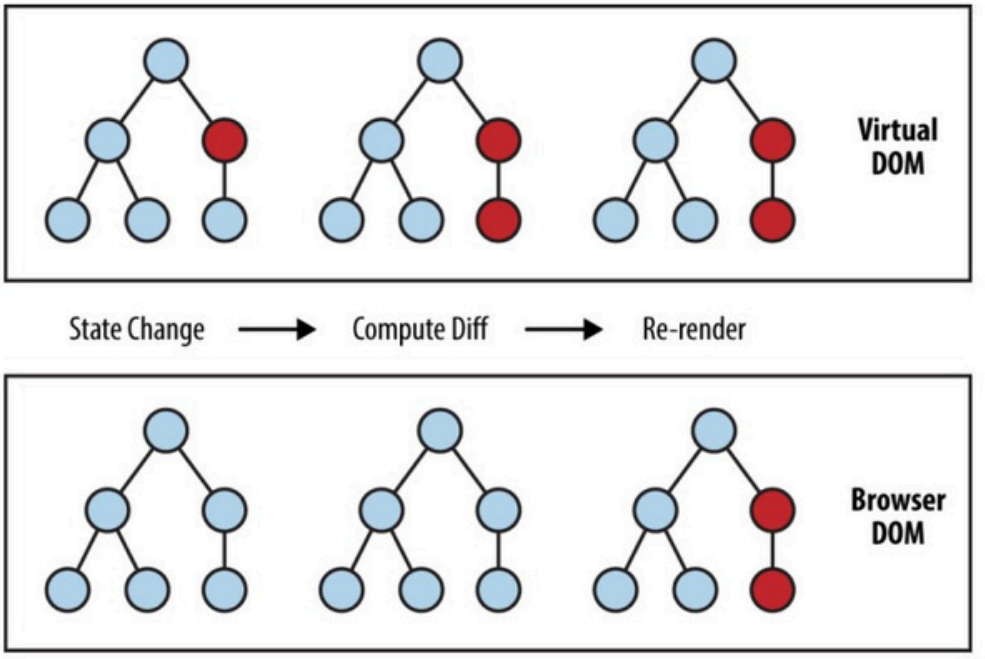
DOM stands for Document Object Model.

3. What is Virtual DOM?

Virtual DOM is a lightweight in-memory representation of the actual DOM. It is used to improve the performance of updates and changes to the actual DOM by reducing the number of expensive DOM manipulations.

4. How does Virtual DOM works?

Virtual DOM works by comparing the current virtual DOM tree with a new virtual DOM tree, and then applying the minimal set of changes to the actual DOM. This allows React to efficiently update the user interface without causing unnecessary re-renders or layout changes.



Visual representation of how Virtual DOM works provided by Ayush Verma; source: https://javascript.plainenglish.io/react-the-virtual-dom-comprehensive-guide-acd19c5e327a

5. What is the difference between Shadow DOM and Virtual DOM?

Shadow DOM refers to a feature of web browsers that allows developers to create a separate DOM tree, called a shadow tree, that is attached to a specific element and is hidden from the main document.

Virtual DOM is a concept used in React to optimize the performance of updates to the user interface, whereas Shadow DOM is a feature of web browsers that allows for scoping of CSS and JavaScript within a specific element.

6. What are the differences between Real DOM and Virtual DOM?

Real DOM is the actual tree-like structure of a web page, which can be manipulated directly to change the layout or content of the page. Virtual DOM is a lightweight in-memory representation of the actual DOM, which is used to optimize the performance of updates to the user interface.

7. What is React Fiber?

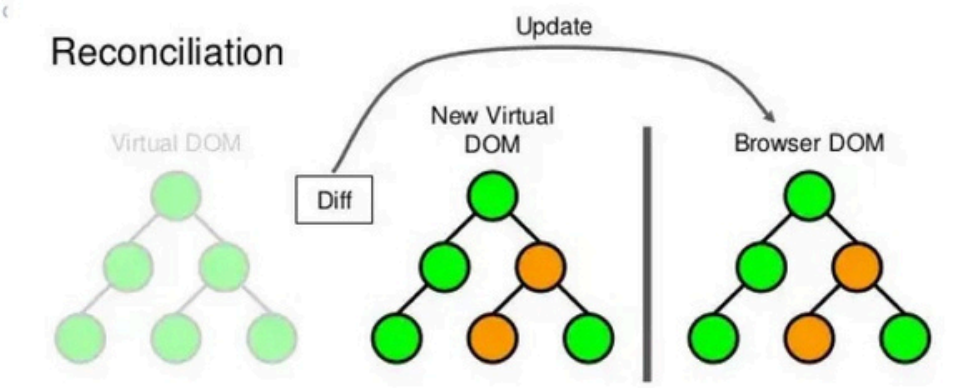
React Fiber is a new reconciliation algorithm that was introduced in React 16. It is designed to improve the performance and flexibility of React by breaking down the render process into smaller, asynchronous chunks.

8. What is the main goal of React Fiber?

The main goal of React Fiber is to improve the performance and responsiveness of web applications by breaking down the render process into smaller, asynchronous chunks. This allows React to more effectively utilize the main thread and to better handle complex user interfaces.

9. What is reconciliation?

Reconciliation is the process that React uses to determine the minimal set of changes to be made to the actual DOM. It compares the current virtual DOM tree with a new virtual DOM tree, and then applies the minimal set of changes to the actual DOM.



Visual representation of how reconciliation works provided by Ayush Verma; source: https://javascript.plainenglish.io/react-the-virtual-dom-comprehensive-guide-acd19c5e327a

10. What is JSX?

JSX is a syntax extension for JavaScript that allows developers to write HTML-like elements in their JavaScript code. It is commonly used in React to describe the structure and content of a component's user interface.

11. Can browsers read a JSX file?

Browsers cannot read JSX files directly. They must be transpiled, or converted, to JavaScript before they can be interpreted by the browser.

12. Explain how VirtualDOM works

Virtual DOM is a lightweight, in-memory representation of the actual DOM. It is used to improve the performance of updates and changes to the actual DOM by reducing the number of expensive DOM manipulations.

When a component's state or props change, React will create a new virtual DOM tree, compare it to the previous tree, and then apply only the minimal set of changes necessary to the actual DOM. This improves the performance of the application and reduces the number of unnecessary re-renders.

13. What are React components?

In React, everything is a component. A component is a small, reusable piece of code that represents a part of a user interface.

Components can be defined as either a class or a function and can include a combination of HTML, CSS, and JavaScript code. These components can also be nested and reused throughout the application, which makes it easy to manage and maintain the application's user interface.

14. What is the meaning of the component-based architecture of React?

The component-based architecture of React allows developers to build complex user interfaces by breaking them down into smaller, reusable components. Each component can manage its own state and props, and can be easily reused throughout the application. This structure makes it easy to understand, test, and maintain the application's codebase.

15. What are functional components?

Functional components in React are simply JavaScript functions that return a React element. They do not have state or lifecycle methods and are used for simple, stateless components.

16. What are class components in React?

Class components in React are defined using the ES6 class syntax. They have access to state and lifecycle methods and are used for more complex, stateful components.

17. What is the difference between functional and class components in React?

The main difference between functional and class components in React is that functional components are simpler and do not have state or lifecycle methods, whereas class components have access to state and lifecycle methods and are more powerful.

Additionally, functional components use hooks for state management.

18. How to use CSS in React?

In React, CSS can be added to a component in several ways:

- Using inline styles
- Using a CSS file and importing it into the component
- Using a CSS preprocessor like Sass or Less
- Using a CSS-in-JS library like styled-components

19. How does rendering work in React?

React's rendering process begins when a component's state or props change. React will create a new virtual DOM tree, compare it to the previous tree, and then apply only the minimal set of changes necessary to the actual DOM. This process is called reconciliation.

20. What are states in React?

State in React is an object that holds data that can change, and it is managed by a component. It is used to store and update the component's data and can be passed down to child components as props.

State can be updated using the setState method, which triggers a re-render of the component, updating the user interface.

21. What are props in React?

Props in React are used to pass data from a parent component to a child component. They are essentially a way to pass data and methods down the component tree. Props are read-only, meaning that they cannot be modified by the child component.

22. What is children prop?

The children prop is a special prop that is used to pass children elements to a component. It is used to pass elements between components, such as a list of items or a set of nested components.

23. What is a higher-order component in React?

A higher-order component (HOC) is a function that takes a component as an argument and returns a new component with additional functionality. HOCs are used to reuse component logic, such as authentication or performance optimization.

24. How to create props proxy for HOC component?

To create a props proxy for an HOC component, you can use the `React.forwardRef` function. This function allows you to pass props through to the wrapped component, preserving the original component's props.

25. What are controlled components?

Controlled components are components that are controlled by the state of the parent component. The parent component manages the component's state, and the child component's behavior is determined by the parent component's state.

26. What are uncontrolled components?

Uncontrolled components are components that manage their own state internally, and their behavior is not determined by the parent component's state.

27. How to update state in React class components?

To update state in a React class component, you can use the **`setState`** method. This method takes an object or function as an argument, and it will merge the new state with the existing state.

28. How to update state in React functional component?

To update state in a React functional component, you can use the **`useState`** hook. This hook returns an array containing the current state and a function to update it.

29. How to Differentiate Between State and Props?

State is the internal data of a component that can change and is managed by the component itself. Props are external data passed to a component from its parent component. State can be updated by the component, whereas props cannot be updated by the component.

30. What is Lifting State Up in React?

Lifting state up in React refers to the process of moving state management from a child component to its parent component. This is done to make the state more easily accessible and manageable for the entire component tree.

31. What is an Event in React?

An event in React is a way to respond to user interactions such as clicks, hover, or form submissions.

32. How to Handle Events in React (for both functional and class components)?

In React, events can be handled using the **`on`** keyword, followed by the event name and a callback function. For example, to handle a button click event, you would use **`onClick={handleClick}`**. This can be done in both functional and class components.

33. What is the difference between HTML and React event handling?

In HTML, events are handled using event attributes, such as **`onclick`** or **`onchange`**. In React, events are handled using the **`on`** keyword and a callback function. React's event handling system is more powerful and flexible than the traditional HTML event handling system.

34. What are synthetic events in React?

Synthetic events in React are a cross-browser compatible way to handle events. They are used to normalize the behavior of different browser event systems and provide a consistent API for handling events.

35. How to pass parameter to an event handler?

To pass a parameter to an event handler, you can use an arrow function to wrap the event handler. For example: `<button onClick={() => handleClick(parameter)}>`.

36. What are React Hooks?

React Hooks are functions that allow you to use state and other React features in functional components. They were introduced in React 16.8.

37. When were the React Hooks introduced first?

React Hooks were first introduced in React 16.8 in 2019.

38. Can you explain the useState hook with examples?

The useState hook allows you to add state to a functional component. It returns an array with two values: the current state and a function to update it. Here's an example of how to use the useState hook:

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

function MyComponent() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default MyComponent;
```

39. Can you explain the useEffect hook?

The useEffect hook allows you to run side effects, such as fetching data or updating the DOM, in a functional component. It takes a callback function as its first argument, which is called after the component has rendered. Here's an example of how to use the useEffect hook:

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

function MyComponent() {
  const [data, setData] = useState([]);

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

    fetchData();
  }, []);

  return (
    <div>
      {data.map(item => (
        <p key={item.id}>{item.name}</p>
      ))}
    </div>
  );
}

export default MyComponent;
```

The useEffect hook is used to run an async function fetchData that fetches data from an API and updates the component's state with the result. The empty array [] passed as the second argument to useEffect means that the effect will only run once, when the component is first rendered.

The useEffect hook allows the component to update its state and re-render in response to changes in some variable. In this case, when the component is first rendered, the data is fetched and the component re-renders with the updated data.

40. When are we using the useMemo hook and why?

useMemo is a hook that allows you to memoize a value. It is used to optimize the performance of a component by only re-computing a value if its dependencies have changed. This can be useful for avoiding expensive calculations or rendering operations. Here's an example of how to use the useMemo hook:

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

function MyComponent() {
  const [a, setA] = useState(1);
  const [b, setB] = useState(1);

  const result = useMemo(() => {
    // This is a costly calculation
    let sum = 0;
    for (let i = 0; i < 1000000; i++) {
      sum += i;
    }
    return sum;
  }, [a, b]);

  return (
    <div>
      <input type="number" value={a} onChange={e => setA(e.target.value)} />
      <input type="number" value={b} onChange={e => setB(e.target.value)} />
      <p>The result is: {result}</p>
    </div>
  );
}

export default MyComponent;
```

In this example, the **useMemo** hook is used to memoize the result of a costly calculation that depends on the values of a and b. The **useMemo** hook takes two arguments: the first is a function that performs the calculation, and the second is an array of dependencies.

The component has two input fields, where the user can set the values of a and b. When either of the inputs change, the component re-renders, but the result is only recalculated if a or b have changed, which is determined by the dependencies array [a,b].

This prevents the costly calculation from being performed every time the component re-renders, improving the performance of the application.

41. What is useRef being used for?

useRef is a hook that allows you to create a reference to a DOM node or a JavaScript object. It can be used to access a DOM node directly, or to store a value that should not cause a re-render when it changes. Here's an example of how to use the useRef hook:

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

function MyComponent() {
  const inputRef = useRef(null);

  function handleClick() {
    inputRef.current.value = "Hello World!";
  }

  return (
    <div>
      <input type="text" ref={inputRef}/>
      <button onClick={handleClick}>
        Update Input
      </button>
    </div>
  );
}

export default MyComponent;
```

42. How to create refs?

To create a ref, you can use the **`useRef`** hook, which returns a ref object. You can then assign this object to a ref attribute on a JSX element.

43. What are forward refs and can you give me a code example for it?

A forward ref is a way to pass a ref through a component to a child component. It allows you to access the ref of a child component from the parent component. Here's an example of how to create a forward ref:

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

const ChildComponent = forwardRef((props, ref) => {
  return (
    <input type="text" ref={ref}/>
  );
});

function ParentComponent() {
  const inputRef = React.useRef(null);

  function handleClick() {
    inputRef.current.value = "Hello World!";
  }

  return (
    <div>
      <ChildComponent ref={inputRef} />
      <button onClick={handleClick}>
        Update Input
      </button>
    </div>
  );
}

export default ParentComponent;
```

In this example, the **`ChildComponent`** is defined as a "forward ref" component using the **`forwardRef`** higher-order component. The **`forwardRef`** function takes a component and returns a new component that can accept a ref as a prop.

The `ParentComponent` has a ref object `inputRef` that is passed as a prop to the `ChildComponent` using the `ref` attribute. Inside the child component, the `ref` prop is passed as a second argument to the component function, this way is passed to the input element using the ref attribute.

When the button is clicked, it triggers the `handleClick` function, which updates the value of the input field by accessing its `value` property via the ref object. This way, the ref object can be passed through the component hierarchy to access elements deep down in the tree.

44. Can you create your custom React hooks?

Yes, you can create your custom React hooks by following the naming convention "use" and using state and other hooks inside it.

45. What do you need to keep in mind while creating custom React hooks?

When creating custom React hooks, it's important to keep in mind that they should only call other hooks at the top level and not inside loops or conditions.

Also, it's important to make sure that the hook only performs one specific action.

46. What is the Context API in React?

The Context API in React allows you to share data between components without passing props through every level of the component tree. It provides a way for components to access data that is "global" to the component tree, such as a user's authentication status or a theme.

The Context API consists of a Provider component, which provides the data, and a Consumer component, which accesses the data.

47. What is React Router?

React Router is a library for routing in React apps. It allows you to define the different routes in your application and render the appropriate components for each route. This makes it easy to change the displayed content based on the current URL, without having to refresh the page.

48. What are Pure components and what is their purpose?

Pure components are components that only re-render if their props or state have changed. They are optimized for performance, and they can improve the performance of your application by reducing the number of unnecessary re-renders.

Pure components are also known as "functional components" or "stateless components" and they are defined by a function.

49. Can you update the React state directly?

No, you cannot update the React state directly. You should always use the `setState` method to update the state, which will trigger a re-render of the component.

50. What is the purpose of callback function as an argument of `setState()`?

The callback function passed as an argument to `setState` is called after the state has been updated and the component has re-rendered. It can be used to perform any additional actions, such as sending a network request, that depend on the updated state.

51. What is "key" prop and what is the benefit of using it in arrays of elements?

The "key" prop is used to give a unique identifier to each item in an array of elements. When elements are re-rendered, React uses the key to identify which elements have changed, added, or removed. This allows React to update the DOM efficiently, improving the performance of the application.

52. What are the different phases of component lifecycle in React?

The different phases of the component lifecycle in React are:

- Mounting: When a component is being added to the DOM.
- Updating: When a component's props or state change.
- Unmounting: When a component is being removed from the DOM.

53. What are the lifecycle methods of React?

The lifecycle methods of React are methods that are called at specific points during the lifecycle of a component. They include:

- **`componentDidMount`**: Called after the component has been added to the DOM.
- **`componentDidUpdate`**: Called after the component has been updated.
- **`componentWillUnmount`**: Called before the component is removed from the DOM.
- **`render`**: Called whenever the component needs to update the DOM.
- **`constructor`**: Called before the component is added to the DOM.

54. What is the purpose of using super constructor with props argument?

The purpose of using the **`super(props)`** constructor with the props argument is to call the constructor of the parent class and pass in the props. It is necessary because the parent class's constructor sets up the initial state and props of the component and must be called before the child class's constructor.

55. Why React uses **`className`** over **`class`** attribute?

React uses the **`className`** attribute instead of the **`class`** attribute because **`class`** is a reserved keyword in JavaScript. Using **`className`** avoids any confusion and ensures that the attribute will be interpreted as intended.

56. What are fragments?

A fragment is a way to group a list of children without adding extra nodes to the DOM. It allows you to return multiple elements from a component's render method without wrapping them in an additional DOM node. Fragments are represented by an empty JSX tag: `<>` or `</>`

57. Why fragments are better than container divs?

Fragments are better than container divs because they don't add an extra node to the DOM. This can make the rendered HTML cleaner and more efficient, especially when you have a component that renders a list of items. Additionally, it helps with accessibility because it doesn't create an unnecessary wrapper element around the content.

58. What are stateless components?

Stateless components, also known as functional components, are components that don't have any internal state. They are defined as a function, they receive props and they return JSX to be rendered. They are simpler, easier to reason about and less prone to bugs than stateful components because they don't have lifecycle methods.

59. What are error boundaries in React v16?

Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI. They were introduced in React v16, and they allow you to handle errors gracefully and ensure that your application continues to function even if there is an error.

60. What is the use of `react-dom` package?

The `react-dom` package is a package that provides the renderer for React components. It provides the functions that are used to render React components on the web (DOM).

61. What will happen if you use `setState()` in constructor?

If you use `setState` in the constructor, it will cause the component to re-render before it is added to the DOM. This can cause unexpected behavior and should be avoided. Instead, it is recommended to initialize the state with the constructor's props and use `setState` in `componentDidMount`.

62. What is the impact of indexes as keys?

Using indexes as keys can have a negative impact on performance because they don't provide a stable identity for elements, and React has to rely on the order of elements in the array to determine which elements have changed. This can lead to unnecessary re-renders.

63. How do you implement Server Side Rendering or SSR?

To implement Server Side Rendering (SSR) in a React application, you can use a library such as Next.js or ReactDOMServer. These libraries allow you to render your React components on the server and send the resulting HTML to the browser. This can improve the performance of your application by reducing the initial load time and providing a better experience for search engines.

64. What is the lifecycle methods order in mounting?

The lifecycle methods order in mounting is:

- `constructor()`
- `static getDerivedStateFromProps()`
- `render()`
- `componentDidMount()`

65. What are the lifecycle methods going to be deprecated in React v16?

The lifecycle methods that are going to be deprecated in React v16 are:

- `componentWillMount`
- `componentWillReceiveProps`
- `componentWillUpdate`

66. What is the purpose of `getDerivedStateFromProps()` lifecycle method?

The purpose of `getDerivedStateFromProps()` is to provide a way to synchronize a component's internal state with its props. This lifecycle method is called before `render()` and it allows you to update the internal state based on changes in the props. It is used as an alternative to `componentWillReceiveProps()`

67. What is the purpose of `getSnapshotBeforeUpdate()` lifecycle method?

The purpose of `getSnapshotBeforeUpdate()` is to capture some information from the DOM before it is potentially changed. This lifecycle method is called right before the browser updates the DOM and it allows you to capture the current scroll position or other information that you might want to use later.

68. Why do we need to pass a function to `setState()`?

We need to pass a function to `setState()` instead of an object because `setState()` is asynchronous, it batches multiple calls together and it only updates the component once. By passing a function, we ensure that the latest state is used when the component re-renders.

69. What is strict mode in React?

Strict mode is a development-only feature in React that highlights potential problems in an application. When a component is rendered in strict mode, React will run extra checks and warnings for any potential issues. This can help you identify and fix problems before they become a bigger issue.

70. Can component re-render without calling `setState`?

You can force a component to re-render without calling setState by changing its key prop. This will cause React to treat the component as a new one and it will re-render it. This can be useful in some cases, but it's generally not recommended because it can cause unnecessary re-renders and negatively impact performance.

71. What is the difference between React and ReactDOM?

React is a JavaScript library for building user interfaces. It provides a way to build reusable UI components and manage their state. ReactDOM is a separate library that provides specific methods for interacting with the DOM, such as `render()` and `unmountComponentAtNode()`. ReactDOM is the glue that connects React with the DOM.

72. Why is ReactDOM separated from React?

React and ReactDOM are separated to allow for better flexibility in different environments. React can be used to build user interfaces on the web, on mobile, or on desktop without having to change the core library. ReactDOM provides the specific methods needed to render components on the web.

73. Is it possible to use React without rendering HTML?

Yes, it is possible to use React without rendering HTML. React can be used to build user interfaces for other environments such as native mobile apps or virtual reality. React Native and React VR are examples of libraries that allow you to use React to build apps for those environments.

74. You can’t update props in React. Why?

You can't update props in React because they are passed down from a parent component and are considered to be immutable. Instead, you should use state to manage data that can change within a component, and pass that data down to child components as props.

75. How to focus an input element on page load?

To focus an input element on page load, you can use the `ref` property to create a reference to the input element, and then call the `focus()` method on that reference in the `componentDidMount()` lifecycle method.

76. How to programmatically trigger a click event in React?

To programmatically trigger a click event in React, you can use the `ref` property to create a reference to the element, and then call the `click()` method on that reference.

77. What is bundling in React and why do we use it?

Bundling is the process of combining multiple JavaScript files into a single file. This is done to reduce the number of network requests needed to load an application and to improve the performance of the application.

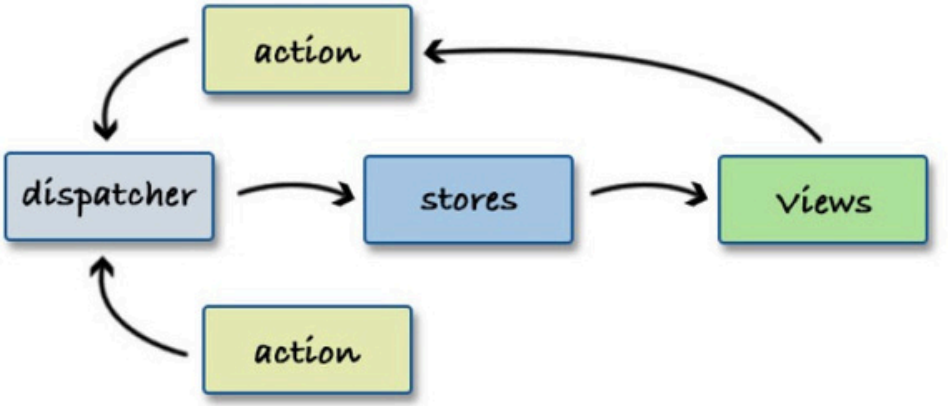
There are various bundlers such as webpack, rollup, parcel, and browserify.

78. What is code-splitting in React and why do we use it?

Code-splitting is a technique that allows you to split your application's JavaScript code into smaller chunks. This can improve the performance of your application by only loading the code that is needed for the current page or component. This technique is usually used in combination with a bundler like webpack.

79. What is flux?

Flux is an architecture for managing the state of an application. It is designed to be used with React and it provides a unidirectional data flow, where data is passed down through components in a hierarchical structure.



A visual representation of flux pattern in React. Source: https://krasimir.gitbooks.io/react-in-patterns/content/chapter-08/

80. What is Redux? Do you know any alternatives?

Redux is a library for managing the state of an application. It provides a centralized store to hold the state, and it uses actions and reducers to manage the state changes. Alternatives to Redux are Mobx, Apollo, and Unstated-next.

81. What are the core principles of Redux?

The core principles of Redux are:

- The state of the application is stored in a single immutable state tree
- The state can only be modified by emitting an action, an object describing the change
- To specify how the state tree is transformed by actions, you write pure reducers

82. What are the downsides of Redux compared to Flux?

The downsides of Redux compared to Flux are:

- The learning curve is steeper as it has a lot more concepts
- It can create unnecessary complexity for small or simple applications
- It can make debugging more difficult because the state is not visible in the components
- It can lead to verbose and repetitive code because actions and action creators are separate files

83. What is the difference between React context and React Redux?

React context is a built-in feature of React that allows you to share data between components without having to pass props down through multiple levels of the component tree. React Redux is a library that allows you to connect your React components to a Redux store. It provides a way to access the state from the store and to dispatch actions to modify the state.

84. Why are Redux state functions called reducers?

Redux state functions are called reducers because they are used to reduce the current state and an action to the next state. The reducer takes the current state and an action as input and returns the next state.

85. What is redux-saga?

redux-saga is a library that allows you to handle side effects in a Redux application by using generator functions. It provides a way to handle async actions and to separate the logic for handling side effects from the rest of the application.

86. What is Redux Thunk?

Redux Thunk is a middleware that allows you to write action creators that return a function instead of an action. This function can then be used to perform async logic and dispatch other actions.

87. What is the difference between React Native and React?

React Native is a framework for building mobile applications using React. It provides a way to use React to build apps for iOS and Android. React, on the other hand, is a JavaScript library for building user interfaces.

React Native uses a subset of React and provides platform-specific components and APIs to access the device's native features.

91. What is NextJS and major features of it?

NextJS is a framework for building web applications with React. It provides a set of features for building server-rendered React applications, including:

- Server-side rendering (SSR) out of the box
- Automatic code splitting for faster load times
- Built-in development server with hot reloading
- Easy setup for custom routes with file-system based routing
- Static site generation (SSG)

92. Is it good to use arrow functions in render methods?

It's generally not good to use arrow functions in the render method because they create a new function on every render, which can negatively impact performance. If a component uses an arrow function in its render method, it will re-create that function on every render and cause the component to re-render even if its props haven't changed.

93. What is route based code splitting?

Route-based code splitting is a technique that allows you to split your application's JavaScript code into smaller chunks based on the routes in your application. This can improve the performance of your application by only loading the code that is needed for the current page or component.

94. Is it possible to use react without JSX?

Yes, it is possible to use React without JSX. React provides a way to use JavaScript to create elements, and you can use JavaScript to create the same elements that you would create with JSX.

88. What is render hijacking in react?

Render hijacking in React is a technique that allows you to modify the rendered output of a component by wrapping it in another component. The wrapped component can then add or remove elements, change the styles, or modify the behavior of the original component.

89. What is React memo function?

React memo is a higher-order component that allows you to optimize the performance of functional components by memoizing the component's output. It prevents the component from re-rendering when its props haven't changed.

90. What is the difference between try catch block and error boundaries?

Try catch block works with imperative code whereas error boundaries are meant for declarative code to render on the screen.

A try-catch block is a language-level construct that allows you to handle exceptions that occur during the execution of the code inside the block.

Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI. Error boundaries are used specifically to handle and handle error in a React application, whereas try-catch block are used to handle exceptions in JavaScript.

95. How to fetch data with React Hooks?

You can use the `useEffect` hook to fetch data in a React functional component. The `useEffect` hook allows you to run a side effect, such as fetching data, after the component has rendered. You can also use `useState` to store the fetched data and update the component when the data is available.

96. What is Concurrent Rendering?

Concurrent rendering is a feature of React that allows the browser to continue rendering the rest of the application while a component is being updated. This can improve the user experience by providing smoother animations and reducing the perceived time to load the application.

97. What is the difference between async mode and concurrent mode?

Async mode is an experimental feature that allows React to schedule updates and render components asynchronously. This can improve the performance of an application by reducing the number of updates that need to be processed at the same time. Concurrent mode is a feature that allows React to work in an asynchronous way and it's not yet released but it's planned to be in the future.

98. What is state mutation and how to prevent it?

State mutation is when the current state is modified directly, instead of creating a new state object. This can make it difficult to track changes and can lead to unexpected behavior. To prevent state mutation, you should use the spread operator or `Object.assign()` to create a new state object and only update the properties that need to be changed.

99. What are the benefits of using typescript with React?

Using TypeScript with React can provide the following benefits:

- Improved code quality and maintainability
- Better type checking and fewer runtime errors
- Improved developer experience with autocompletion and better error messages
- Better documentation and understanding of the codebase

100. What is the purpose of `eslint` plugin for hooks?

The ESLint plugin for hooks is a set of rules that enforces the rules of hooks as specified by the React team. It helps to prevent common mistakes and inconsistencies when using hooks. It also helps to make sure that hooks are only used in functional components and that they are called in the correct order.