React

Rodrigo Machado

1 O que é React e quais são suas principais características?

React é uma biblioteca JavaScript de código aberto para criar interfaces de usuário (UI) dinâmicas e reativas. Desenvolvida pelo Facebook, ela foi lançada em 2013 e se tornou uma das ferramentas mais populares para o desenvolvimento de aplicativos web modernos. Sua principal característica é a componentização.

2 Qual é a diferença entre componentes funcionais e de classe em React?

Em resumo, a principal diferença entre componentes funcionais e de classe é a maneira como são definidos e a presença de estado interno e métodos de ciclo de vida. Componentes funcionais são mais simples e leves, enquanto componentes de classe são mais poderosos e podem conter funcionalidades adicionais. Com a introdução dos Hooks, a distinção entre os dois tipos de componentes tornou-se menos clara, e agora é possível usar ambos os tipos de componentes de forma intercambiável em muitos casos.

3 O que são props em React e como eles são usados?

Em React, "props" é uma abreviação para "propriedades" e refere-se aos parâmetros passados para um componente. As props são usadas para passar dados de um componente pai para um componente filho. Elas são somente leitura e não podem ser modificadas pelo componente filho.

As props são passadas para um componente como atributos HTML, e podem conter qualquer tipo de dado, incluindo strings, números, booleanos, objetos, arrays ou até mesmo funções.

4 Explique o estado (state) em React e como ele difere das props.

Em React, "estado" (state) refere-se aos dados internos de um componente que podem ser modificados ao longo do tempo. O estado é gerenciado dentro de um componente e pode ser atualizado usando o método setState(). Cada componente React pode ter seu próprio estado interno, que é independente de outros componentes. Enquanto as props são usadas para passar dados de componentes pais para componentes filhos e são somente leitura, o estado é usado para representar dados mutáveis que afetam a renderização do componente e podem ser modificados ao longo do tempo dentro do próprio componente.

5 Qual é a diferença entre stateful e stateless components em React?

Em React, os componentes são frequentemente classificados como "stateful" (com estado) ou "stateless" (sem estado), com base em como eles gerenciam o estado interno do componente.Também conhecidos como "class components" (componentes de classe), os componentes stateful possuem um estado interno, definido usando o objeto 'state' e podem modificar seu estado interno ao longo do tempo usando o método setState(). Stateless Components (Componentes sem estado) são também conhecidos como "functional components" (componentes funcionais) que não possuem um estado interno e são puramente baseados em suas props, não podem modificar seu estado interno e são geralmente usados para componentes simples que apenas renderizam informações recebidas por props. Ambos os tipos de componentes têm seu lugar em aplicações React e são usados de acordo com a complexidade e os requisitos específicos de cada componente. Com a introdução dos Hooks no React, os componentes funcionais podem agora ter estado interno e são capazes de realizar mais do que no passado, reduzindo assim a distinção entre componentes com estado e sem estado em alguns casos.

6 O que é JSX e por que é comumente usado em React?

JSX é uma extensão de sintaxe JavaScript que permite escrever código HTML dentro de JavaScript. JSX se parece muito com HTML, o que facilita a transição para React para desenvolvedores que já estão familiarizados com o desenvolvimento web, ademais escrever código usando JSX é geralmente mais fácil e mais legível do que manipular diretamente o

DOM usando métodos JavaScript. Com JSX, você pode descrever a estrutura da interface do usuário de forma declarativa, como você faria com HTML.

7 Como você incorporaria JavaScript dentro do JSX?

Para incorporar JavaScript dentro do JSX em um componente React, você pode usar chaves para envolver as expressões JavaScript. Isso permite que você insira dinamicamente valores de variáveis, chame funções e avalie expressões JavaScript diretamente dentro do JSX.

8 Qual é a diferença entre JSX e HTML?

Existem algumas diferenças importantes entre JSX e HTML, embora visualmente eles possam parecer semelhantes. A principal diferença é vista na definição de ambas: JSX é uma extensão de sintaxe JavaScript que permite escrever código HTML dentro de JavaScript. Portanto, elementos JSX são definidos usando uma sintaxe semelhante ao HTML, mas são interpretados como expressões JavaScript, possibilitando ainda incorporação de código JavaScript diretamente no código. O HTML, por outro lado, é uma linguagem de marcação usada para criar páginas da web. É uma linguagem independente que não depende de JavaScript.

9 Como você cria um componente em React?

Para criar um componente em React, você pode usar uma sintaxe de classe ou uma função. Componentes Stateful são classes que derivam de React.Component, possuindo seu ciclo de vida, possuindo obrigadoriamente o método render() que retorna o JSX do componente. Componentes Stateless são funções que retornam o JSX, podendo possuir funções em seu corpo, mas sempre retornando o JSX como padrão.

10 O que é renderização condicional em React e como você a implementaria?

A renderização condicional em React refere-se à técnica de renderizar diferentes elementos ou componentes com base em uma condição específica. Isso permite que você controle dinamicamente o que é exibido na interface do usuário com base no estado da sua aplicação.

Existem várias maneiras de implementar a renderização condicional em React, incluindo o uso de instruções if, operador ternário, operador lógico &&, e o método map() em arrays.

11 Como você passaria dados de um componente pai para um componente filho em React?

O ciclo de vida de um componente em React refere-se às diferentes fases pelas quais um componente passa, desde a sua inicialização até a sua remoção do DOM. Cada fase do ciclo de vida oferece pontos de gancho (lifecycle hooks) que permitem que você execute código em momentos específicos do ciclo de vida do componente. Existem três fases principais no ciclo de vida de um componente React: Mounting, Updating e Unmounting.

Durante o Mounting, o componente está sendo criado e inserido no DOM. 'constructor():' é o primeiro método chamado durante a inicialização de um componente de classe e é usado para inicializar o estado e vincular métodos de classe. 'render()' é responsável por retornar os elementos JSX que compõem o componente. 'componentDidMount()' Chamado após o componente ser montado no DOM. É frequentemente usado para realizar chamadas de API, configurar assinaturas de eventos ou realizar outras operações de inicialização que exigem acesso ao DOM.

Durante o updating o componente é atualizado em resposta a alterações em seu estado ou props.

Durante o unmounting o componente é removido do DOM. 'componentWillUnmount()' é o método chamado antes do componente ser removido do DOM. É usado para realizar operações de limpeza, como cancelar assinaturas de eventos ou liberar recursos alocados.

12 Como você lida com eventos em React?

Em React, você pode lidar com eventos usando a sintaxe JSX para adicionar ouvintes de eventos diretamente aos elementos HTML. Por exemplo, no clique de um botão pode-se atribuir uma função a ser executada através do evento onClick.

13 O que é o evento sintético em React?

No React, um evento sintético (SyntheticEvent) é um objeto que encapsula um evento do navegador e fornece uma interface de programação consistente para trabalhar com eventos em diferentes navegadores. Ele é criado para normalizar as diferenças de implementação de eventos entre navegadores e garantir um comportamento consistente em todas as plataformas. Os eventos sintéticos em React fornecem uma camada de abstração sobre os eventos do navegador, garantindo compatibilidade cruzada e consistência de comportamento em diferentes navegadores, ao mesmo tempo em que oferecem recursos adicionais para manipulação de eventos dentro de componentes React.

14 Como você gerencia o estado em React?

Em React, o estado de um componente é uma representação de dados que podem mudar ao longo do tempo, geralmente como resultado de interações do usuário, chamadas de API ou outras fontes de dados externas. O estado é fundamental para criar componentes dinâmicos e reativos em React. Para gerenciar é necessário inicializar o estado (com stateless usando o useState e com o statefull com o this.states =), acessando e atualizando o valor do estado (setState ou this.setState()) e colocando o valor no JSX para apresentação em tela.

15 Quando você usaria o estado local vs. o estado global em um aplicativo React?

Em resumo, o uso de estado local ou global em um aplicativo React depende da natureza dos dados, dos requisitos de compartilhamento e da complexidade da aplicação. Em muitos casos, uma combinação de ambos é usada para alcançar um equilíbrio entre simplicidade, eficiência e gerenciamento adequado do estado.

Normalmente estados globais são utilizados para aplicativos mais complexos com múltiplos componentes interdependentes ou estados complexos que afetam diferentes partes da interface do usuário, um estado global (como o Redux ou o Context API) oferece uma solução mais robusta e escalável, e também quando vários componentes precisam acessar e compartilhar os mesmos dados ou estados, é mais apropriado usar um estado global para evitar a propagação excessiva de props entre os componentes.

Estados locais por sua vez é adequado quando os dados são específicos para um componente e não precisam ser compartilhados com outros componentes, é útil para dados

temporários ou relacionados à interface do usuário, como valores de formulários, estados de exibição de modais ou tooltips, etc.

16 Qual é a diferença entre useState e useContext em React?

O hook useState é usado para adicionar estado a componentes funcionais. Ele retorna um par de valores: o estado atual e uma função para atualizar esse estado. O useState é útil quando você precisa manter e atualizar o estado local dentro de um componente funcional e é local ao componente onde foi definido, não compartilhado com outros componentes.

O hook useContext por sua vez é usado para acessar o contexto em componentes funcionais. Ele permite que você consuma valores fornecidos por um Provider em componentes filhos sem a necessidade de props intermediárias. É útil quando você tem dados ou funcionalidades que precisam ser compartilhados por vários componentes em diferentes níveis da árvore de componentes. O contexto fornecido por useContext é global e pode ser acessado por qualquer componente filho dentro da árvore de componentes envolvida pelo Provider.

Exemplo com o useState:

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

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

  const incrementCount = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={incrementCount}>Increment</button>
    </div>
  );
};

export default MyComponent;
```

Exemplo com o useContext:

```jsx
import React, { useContext } from 'react';

const MyContext = React.createContext();

const MyComponent = () => {
  const data = useContext(MyContext);

  return <p>Data from context: {data}</p>;
};

const ParentComponent = () => {
  return (
    <MyContext.Provider value="Hello from context">
      <MyComponent />
    </MyContext.Provider>
  );
};

export default ParentComponent;
```

17 Como você implementaria o roteamento em um aplicativo React?

Em um aplicativo React, o roteamento é geralmente implementado usando uma biblioteca de roteamento, sendo o React Router a opção mais comum. O React Router é uma biblioteca popular que permite adicionar roteamento de página à sua aplicação React de forma declarativa e fácil de usar.

Exemplo:

```jsx
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Home from './components/Home';
import About from './components/About';
import Contact from './components/Contact';

function App() {
  return (
    <Router>
      <Switch>
        <Route path="/" exact component={Home} />
        <Route path="/about" component={About} />
        <Route path="/contact" component={Contact} />
      </Switch>
    </Router>
  );
}

export default App;
```

18 Qual é a diferença entre BrowserRouter e HashRouter em React?

O BrowserRouter usa o histórico de navegação HTML5, que é fornecido pelo navegador. Ele usa a API pushState() do navegador para manipular as URLs sem recarregar a página. Este método é ideal para aplicativos que serão hospedados em servidores que oferecem suporte à manipulação de URLs do lado do servidor para todas as rotas. O HashRouter usa parte da URL conhecida como "hash", representada pelo caractere #, para controlar o roteamento. Isso significa que todas as rotas serão seguidas por # na URL (por exemplo, http://example.com/#/route). Este método é útil para aplicativos que serão hospedados em servidores que não oferecem suporte à manipulação de URLs do lado do servidor para todas as rotas, como páginas GitHub, onde você não tem acesso ao servidor para configurar redirecionamentos de URL. Em resumo, a principal diferença entre BrowserRouter e HashRouter está na forma como eles gerenciam o roteamento e as URLs. O BrowserRouter oferece URLs limpas e amigáveis sem o caractere #, enquanto o HashRouter usa o caractere # na URL para controlar o roteamento e é compatível com todos os navegadores. A escolha

entre os dois depende das necessidades específicas do seu aplicativo e das configurações de hospedagem.

19 Como você faria uma solicitação HTTP em um componente React?

Para fazer uma solicitação HTTP em um componente React, você pode usar a API fetch do navegador ou bibliotecas de terceiros como Axios. Em sua maioria, as solicitações iram retornar uma promise possibilitando que você programe de forma assíncrona o que deseja que aconteça em acso de sucesso ou erro. Preferencialmente gosto de trabalhar com axios, é possível deixa-lo de forma isolada como um serviço.

Exemplo utilizando fetch:

```jsx
import React, { useState, useEffect } from 'react';
function MyComponent() {
  const [data, setData] = useState(null);
  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch('https://api.example.com/data');
        // Verificando se a solicitação foi bem-sucedida
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        const jsonData = await response.json();
        setData(jsonData);
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };
    fetchData();
  }, []); // O array vazio como segundo argumento garante que o useEffect seja executado a
  return (
    <div>
      {data && (
        <ul>
          {data.map(item => (
            <li key={item.id}>{item.name}</li>
          ))}
        </ul>
      )}
    </div>
  );
}
```

Exmplo utilizando axios:

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

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

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await axios.get('https://api.example.com/data');
        setData(response.data);
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };
    fetchData();
  }, []); // O array vazio como segundo argumento garante que o useEffect seja executado
  return (
    <div>
      {data && (
        <ul>
          {data.map(item => (
            <li key={item.id}>{item.name}</li>
          ))}
        </ul>
      )}
    </div>
  );
}

export default MyComponent;
```

20 Como você lida com a manipulação de dados assíncronos em React?

Em React, a manipulação de dados assíncronos é comumente feita utilizando recursos como Hooks de efeito (useEffect), Promises, e o uso de estados locais ou globais para armazenar os dados.

O hook useEffect é frequentemente utilizado para realizar operações assíncronas, como busca de dados em uma API. Você pode chamar funções assíncronas dentro do useEffect e atualizar o estado com os dados recebidos.

Você pode usar Promises para fazer solicitações assíncronas e lidar com os dados retornados. Isso pode ser feito diretamente nos métodos de ciclo de vida (componentDidMount, componentDidUpdate) em componentes de classe ou dentro de funções assíncronas em componentes funcionais.

Para dados que precisam ser compartilhados entre vários componentes, você pode usar gerenciamento de estado global com Redux ou o Context API do React. Isso permite que você centralize o estado da aplicação e atualize-o de forma assíncrona em resposta a eventos em qualquer parte da aplicação.

21 Qual é a diferença entre Redux e Context API em React?

Redux: O Redux é uma biblioteca independente de React e pode ser usado em qualquer aplicação JavaScript, não apenas em aplicações React. Ele oferece um único store global que contém todo o estado da aplicação. O Redux é uma solução mais robusta e complexa, especialmente para aplicações maiores. Ele possui conceitos como reducers, actions e middlewares, que podem ser complexos para iniciantes, mas oferecem um alto nível de controle e previsibilidade.

Context API: A Context API é uma API nativa do React e é usada especificamente para compartilhar dados entre componentes React. Ela permite criar um contexto que pode ser acessado por componentes aninhados na árvore de componentes do React. A Context API é mais simples e fácil de entender em comparação com o Redux. Ela oferece uma maneira mais direta de compartilhar dados entre componentes, mas pode ser menos escalável para aplicações muito grandes devido à falta de ferramentas para lidar com a complexidade do estado.

Em resumo, o Redux é uma escolha preferencial para aplicações grandes e complexas que exigem um gerenciamento avançado de estado, enquanto a Context API é uma opção mais simples e direta para compartilhar dados entre componentes em aplicações menores ou médias. A escolha entre Redux e Context API depende das necessidades específicas do projeto, da complexidade da aplicação e das preferências da equipe de desenvolvimento.

22 Quais são algumas boas práticas para escrever código eficiente em React?

Escrever código eficiente em React envolve seguir boas práticas de desenvolvimento que promovem a legibilidade, manutenibilidade e desempenho do código. Algumas dicas:

Divida sua interface de usuário em componentes reutilizáveis e pequenos, cada um responsável por uma única funcionalidade.

Mantenha os componentes pequenos e focados em uma única tarefa. Se um componente estiver crescendo demais, considere dividir em componentes menores.

Utilize os Hooks introduzidos no React 16.8 para gerenciar o estado e o ciclo de vida do componente. Prefira o uso de useState, useEffect, useContext e outros Hooks fornecidos pelo React para simplificar o código e evitar a necessidade de classes de componentes.

Mantenha o estado do componente imutável sempre que possível. Evite modificar o estado diretamente, em vez disso, utilize métodos como setState para atualizar o estado de forma imutável.

Evite re-renderizações desnecessárias de componentes. Utilize a otimização de renderização condicional, como React.memo, para evitar a renderização de componentes quando as props não mudaram.

Utilize useMemo para memorizar valores calculados e evitar recálculos em cada renderização.

Utilize a gestão de estado local para dados que são específicos de um único componente ou não precisam ser compartilhados com outros componentes.

Para dados compartilhados entre vários componentes, considere o uso de gerenciadores de estado globais como Redux ou a Context API do React.

Utilize nomes descritivos e significativos para variáveis, funções e componentes.

Siga convenções de nomenclatura consistentes, como camelCase para nomes de variáveis e PascalCase para nomes de componentes.

Implemente tratamento de erros adequado para lidar com exceções e condições inesperadas.

Utilize componentes de erro para renderizar mensagens de erro de forma consistente em toda a aplicação.

Escreva testes unitários e de integração para seus componentes e lógica de aplicativo utilizando bibliotecas como Jest e React Testing Library.

Projete seus componentes com testabilidade em mente, mantendo-os pequenos e focados em uma única responsabilidade.

23 Como você otimizaria o desempenho de um aplicativo React?

Para otimizar o desempenho de um aplicativo React, você pode seguir várias práticas recomendadas e técnicas de otimização. Aqui estão algumas delas:

Renderização Condicional: Utilize a renderização condicional para evitar renderizar componentes desnecessariamente. Isso pode ser feito usando declarações if, operadores ternários ou componentes de renderização condicional, como React.memo ou shouldComponentUpdate.

Memoização de Componentes: Use React.memo para memoizar componentes funcionais e evitar re-renderizações desnecessárias quando as props não mudam.

Otimização de Re-renderizações: Utilize shouldComponentUpdate em componentes de classe ou React.memo em componentes funcionais para evitar re-renderizações desnecessárias quando as props ou estado não mudam.

Uso Eficiente de Estado: Mantenha o estado do componente mínimo e localize-o sempre que possível. Evite armazenar grandes quantidades de dados no estado do componente, especialmente se esses dados não afetarem a renderização do componente.

Lazy Loading e Suspense: Implemente o lazy loading de componentes e o suspense para carregar componentes assincronamente apenas quando necessário, reduzindo o tempo de carregamento inicial e melhorando a experiência do usuário.

Code Splitting: Divida seu código em pacotes menores e carregue-os sob demanda. Isso pode ser feito usando a função import() ou utilizando ferramentas de empacotamento como Webpack.

Otimização de Imagens: Otimize o tamanho das imagens para reduzir o tempo de carregamento da página. Use formatos de imagem eficientes, como WebP, e ferramentas de compressão de imagem, como imagemin.

Otimização de Bundle: Remova dependências não utilizadas e reduza o tamanho do seu bundle JavaScript. Utilize ferramentas como Webpack Bundle Analyzer para identificar e

remover código não utilizado.

Otimização de Redes: Minimize o número de solicitações de rede e reduza o tamanho das respostas do servidor. Utilize técnicas como o caching de dados e a compressão de resposta para melhorar o desempenho da rede.

Monitoramento e Perfis: Use ferramentas de monitoramento de desempenho, como o Chrome DevTools, para identificar gargalos de desempenho e áreas de melhoria em seu aplicativo. Realize perfis de desempenho para identificar componentes ou operações que estão causando lentidão.

24 O que é Code-Splitting em React e como você o implementaria?

O code splitting é quando você permite que sua aplicação carrega componentes conforma necessidade. Ao invés de importar todos os componentes que um componente pai utiliza de uma vez, fazendo a renderização de tudo ao mesmo tempo, o objetivo é fazer isso gradualmente. Então, dado um componente pesado para carregamento, é possível fazer import de novos componentes através de imports e atribuílos a constantes para serem usados no componente pai.

Segue um exemplo prático:

```jsx
// App.js
import React, { useState } from 'react';

const App = () => {
  const [showLazyComponent, setShowLazyComponent] = useState(false);

  // Function to handle loading of the lazy component
  const loadLazyComponent = async () => {
    // Dynamically import the LazyLoadedComponent
    const LazyLoadedComponent = await import('./LazyLoadedComponent');
    setShowLazyComponent(true);
  };

  return (
    <div>
      <h1>My React App</h1>
      <button onClick={loadLazyComponent}>Load Lazy Component</button>
      {/* Render the LazyLoadedComponent conditionally */}
      {showLazyComponent && <LazyLoadedComponent />}
    </div>
  );
};

export default App;
```

25 Como se utiliza react.memo?

React memo é um componente que grava um componente fucional previnindo que o mesmo se renderize novamente várias vezes quando utilizado por outro componente. Segue exmplo

```jsx
import React from 'react';

// Functional component that displays a greeting
const Greeting = ({ name }) => {
  console.log('Rendering Greeting');
  return <div>Hello, {name}!</div>;
};

// Wrap the Greeting component with React.memo
const MemoizedGreeting = React.memo(Greeting);

// Parent component
const ParentComponent = () => {
  const [count, setCount] = React.useState(0);

  // Function to increment the count
  const incrementCount = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <button onClick={incrementCount}>Increment Count</button>
      <MemoizedGreeting name="Alice" />
      {/* The MemoizedGreeting component will not re-render unless the 'name' prop
      <MemoizedGreeting name="Bob" />
    </div>
  );
};

export default ParentComponent;
```

26 Como você testaria componentes em React?

Testar componentes em React é essencial para garantir que eles se comportem conforme o esperado e mantenham a funcionalidade esperada à medida que a aplicação evolui. A abordagem preferida para testar componentes React é usar bibliotecas de teste como Jest junto com React Testing Library ou Enzyme.

Segue um exmplo de como fazer um teste de componente:

```
// Counter.js
import React, { useState } from 'react';

const Counter = () => {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p data-testid="count">Contagem: {count}</p>
      <button onClick={increment}>Incrementar</button>
    </div>
  );
};

export default Counter;
```

Teste:

```
// Counter.test.js
import React from 'react';
import { render, fireEvent } from '@testing-library/react';
import Counter from './Counter';

describe('Componente Counter', () => {
  it('incrementa a contagem quando o botão é clicado', () => {
    const { getByText, getByTestId } = render(<Counter />);
    const elementoContagem = getByTestId('count');
    const elementoBotao = getByText('Incrementar');

    // A contagem inicial deve ser 0
    expect(elementoContagem.textContent).toBe('Contagem: 0');

    // Clique no botão
    fireEvent.click(elementoBotao);

    // Após clicar no botão, a contagem deve ser 1
    expect(elementoContagem.textContent).toBe('Contagem: 1');
  });
});
```