Responda às seguintes perguntas sobre React:

1. O que é o JSX e por que ele é usado?

JSX, ou JavaScript XML, é uma extensão de sintaxe para o JavaScript, frequentemente associada ao React. O JSX permite escrever estruturas de árvore de elementos de maneira mais concisa e legível, misturando HTML/XML com código JavaScript.

2. Como se define um inline style?

Em React, você pode definir estilos inline diretamente nos elementos utilizando o atributo style. O valor desse atributo é um objeto JavaScript onde as chaves são os nomes das propriedades CSS e os valores são os valores dessas propriedades.

3. O que são props?

Em React, "props" é uma abreviação para "propriedades" (do inglês, "properties"). As props são um meio de passar dados de um componente pai para um componente filho. Elas são utilizadas para transmitir informações de um componente superior para um componente inferior na hierarquia de componentes.

Quando você cria um componente React, pode passar dados para esse componente como se estivesse passando atributos para uma tag HTML. Esses dados são então acessíveis dentro do componente como props.

4. Como se passa uma função via props?

Para passar uma função via props em React, você segue um processo semelhante ao de passar qualquer outro dado. A diferença é que, em vez de passar um valor diretamente, você passa uma referência para a função como uma propriedade. Isso permite que a função seja chamada no componente filho, mesmo que a lógica real da função resida no componente pai.

Essa abordagem é útil para estabelecer uma comunicação entre componentes, permitindo que o componente filho invoque a função definida no componente pai. Essa passagem de funções via props é uma técnica poderosa para implementar ações ou manipulações de dados que afetam o estado de componentes pai.

5. Como se lida com o estado de um componente?

O estado em um componente React é um objeto que representa a condição ou os dados mutáveis desse componente. Para lidar com o estado de um componente, você geralmente usa o hook useState (no caso de componentes funcionais) ou o método setState (no caso de componentes de classe).

- useState (Componentes Funcionais): Com o hook useState, você pode declarar variáveis de estado dentro de um componente funcional. O primeiro

elemento retornado pelo useState é o valor atual do estado, e o segundo elemento é uma função para atualizar esse estado.

- setState (Componentes de Classe): Em componentes de classe, o estado é gerenciado usando o método setState. Este método é chamado com um objeto que representa as alterações a serem feitas no estado. O React então mescla esse objeto com o estado atual e re-renderiza o componente.

6. O que é o ciclo de vida de um componente e por que é importante?

O ciclo de vida de um componente em React refere-se às fases pelas quais um componente passa, desde a sua criação até a sua remoção. Existem três fases principais no ciclo de vida de um componente em componentes de classe:

- Montagem (Mounting): Esta fase ocorre quando um componente está sendo criado e inserido no DOM. Os métodos associados incluem constructor, componentWillMount (agora obsoleto), render, componentDidMount. Neste estágio, o componente é inicializado, os elementos são renderizados no DOM e é possível realizar chamadas a API ou configurações iniciais.
- Atualização (Updating): Esta fase ocorre quando o estado do componente ou as props são alteradas. Os métodos associados incluem shouldComponentUpdate, componentWillUpdate (agora obsoleto), render, componentDidUpdate. Aqui, o componente é re-renderizado, permitindo que você ajuste o DOM ou execute ações específicas após uma atualização.
- Desmontagem (Unmounting): Esta fase ocorre quando o componente é removido do DOM. O método associado é componentWillUnmount (agora obsoleto). Neste estágio, você pode realizar tarefas de limpeza, como cancelar assinaturas de eventos ou interromper a execução de temporizadores.

O ciclo de vida é importante porque fornece aos desenvolvedores controle e flexibilidade sobre o comportamento de um componente em diferentes estágios da sua existência. Isso é fundamental para tarefas como inicialização, manipulação de atualizações, gerenciamento de recursos, otimização de desempenho e limpeza.

7. Como se criam componentes funcionais?

Para criar componentes funcionais em React, você define uma função JavaScript que retorna à representação do componente. Essas funções são chamadas de componentes funcionais porque são, literalmente, funções JavaScript. Aqui estão os passos principais para criar um componente funcional:

- Defina a Função:
  - Crie uma função JavaScript que representará o seu componente. Esta função pode ter qualquer nome que você desejar.
- Retorne JSX:
  - Dentro da função, use o return para retornar a descrição do componente em JSX. Isso pode ser tão simples quanto um elemento

 ou algo mais complexo, dependendo da estrutura do componente.

- Exporte o Componente:
    - No final do arquivo, exporte a função que representa o seu componente. Isso permite que outros arquivos importem e usem esse componente.

8. Para que servem os formulários em React?

Os formulários em React são usados para coletar e gerenciar dados de entrada do usuário. Eles são uma parte essencial de muitas aplicações web, pois permitem aos usuários interagir e enviar dados para a aplicação. Em React, os formulários são criados utilizando elementos de formulário HTML tradicionais, como <form>, <input>, <textarea>, etc., mas o gerenciamento de estado e manipulação de eventos são geralmente realizados de maneira especial usando o estado do React.

Podemos citar alguns dos propósitos e funcionalidades dos formulários em React:

1. Coleta de Dados de Entrada:
    1.1. Os formulários são usados para coletar informações inseridas pelos usuários, como texto, seleções, caixas de seleção, botões de rádio, etc.
2. Gerenciamento de Estado:
    2.1. React fornece uma maneira eficiente de gerenciar o estado dos elementos de formulário. Os dados inseridos pelos usuários podem ser armazenados no estado do componente, permitindo a reatividade e atualizações dinâmicas na interface do usuário conforme os usuários interagem com o formulário.
3. Validação de Dados:
    3.1. React possibilita a implementação de lógica de validação de dados de forma controlada. É possível validar os dados de entrada antes de serem enviados para garantir que estejam no formato correto.
4. Manipulação de Eventos:
    4.1. React permite a manipulação de eventos relacionados aos formulários, como eventos de envio (onSubmit), eventos de alteração (onChange), eventos de foco (onFocus), entre outros. Isso facilita a execução de ações específicas em resposta às interações dos usuários.
5. Envio de Dados para o Servidor:
    5.1. Após a coleta e validação dos dados, os formulários em React são usados para enviar esses dados para o servidor. Geralmente, isso é feito por meio de uma solicitação HTTP, como uma solicitação POST.
6. Feedback ao Usuário:
    6.1. Com o gerenciamento de estado, é possível fornecer feedback dinâmico ao usuário à medida que ele preenche o formulário, como mensagens de erro, confirmação de envio ou atualização de contadores de caracteres.

9. O que é e para que serve o React Native?

React Native é um framework de desenvolvimento de aplicativos móveis que permite criar aplicativos nativos para dispositivos iOS e Android usando JavaScript e React. Ele é uma extensão do React, uma biblioteca popular para criar interfaces de usuário (UI) em aplicações web. Com o React Native, é possível reutilizar grande parte do código entre plataformas, mantendo a capacidade de criar aplicativos com desempenho próximo ao de aplicativos nativos.

Principais características e finalidades do React Native:

- Desenvolvimento Cross-Platform: React Native permite que desenvolvedores criem aplicativos para iOS e Android com uma única base de código. Isso significa que grande parte da lógica do aplicativo pode ser compartilhada entre as plataformas, reduzindo a duplicação de esforços e facilitando a manutenção.
- Componentes Nativos Reutilizáveis: React Native permite o uso de componentes nativos da plataforma, o que significa que você pode incorporar facilmente elementos nativos (como barras de navegação, botões e controles) em seus aplicativos React Native. Além disso, é possível criar e reutilizar seus próprios componentes nativos.
- Desempenho: Os aplicativos React Native não são aplicativos web embutidos em uma visualização da web, como em algumas abordagens de desenvolvimento híbrido. Em vez disso, eles utilizam componentes nativos para proporcionar desempenho semelhante ao de aplicativos nativos.
- Hot Reloading: React Native oferece recursos de "hot reloading", permitindo que você veja instantaneamente as mudanças feitas no código refletidas no aplicativo em execução, sem a necessidade de recarregar a aplicação.
- Ecossistema React: Como extensão do React, React Native herda muitos conceitos e padrões de design do React, facilitando a transição para desenvolvedores familiarizados com o ecossistema React.
- Comunidade Ativa e Suporte: React Native possui uma comunidade grande e ativa, o que significa que há uma abundância de bibliotecas, ferramentas e suporte disponíveis para os desenvolvedores.

10. Qual a diferença entre um component e um PureComponent?

- Componentes Comuns (Funcionais ou de Classe):
    - Quando você atualiza o estado ou as props de um componente comum, o React sempre realizará uma nova renderização do componente, mesmo que os dados de entrada (estado ou props) não tenham mudado. Se a renderização do componente for cara em termos de desempenho e não houver alterações nos dados de entrada, essa renderização adicional pode ser considerada desnecessária.
- PureComponent:

- É uma classe especial fornecida pelo React que estende a funcionalidade de uma classe de componente comum. Um PureComponent implementa uma lógica de comparação superficial (shallow comparison) nas props e no estado antes de decidir se deve ou não realizar uma nova renderização. Se as props ou o estado do PureComponent não mudarem (com base em comparação superficial), ele evitará uma nova renderização, o que pode levar a ganhos de desempenho em certos cenários.