O que é React.JS?

O ReactJS é uma biblioteca de JavaScript que permite construir interfaces de usuário interativas de forma eficiente e reutilizável. Ele ajuda a criar aplicativos da web mais dinâmicos, flexíveis e rápidos, tornando a experiência do usuário mais agradável.

Em vez de construir cada parte separadamente, o ReactJS permite que você crie componentes reutilizáveis, que são como peças de Lego. Você pode usar esses componentes em diferentes partes do seu aplicativo, o que torna o desenvolvimento mais rápido e fácil.

Uma analogia útil é pensar em um aplicativo como uma casa. O ReactJS seria como a estrutura da casa, onde você pode construir e organizar diferentes quartos e espaços.
Os componentes seriam como os móveis dentro de cada quarto. Você pode ter um componente para uma mesa, outro componente para uma cadeira e assim por diante.
E assim como você pode reorganizar os móveis dentro de uma casa, você pode reutilizar e rearranjar os componentes do ReactJS em diferentes partes do seu aplicativo.

Outro benefício do ReactJS é que ele atualiza apenas as partes necessárias da interface quando os dados mudam. Isso significa que seu aplicativo pode ser muito rápido e responsivo, mesmo quando há muitas coisas acontecendo na tela.

10 tópicos que você deve conhecer antes de começar a programar em ReactJS

HTML e CSS: Embora o ReactJS seja uma biblioteca JavaScript, ele é usado para criar interfaces de usuário, o que envolve trabalhar com HTML e estilizar elementos com CSS. Ter um conhecimento básico dessas tecnologias é importante para criar e estilizar componentes React.

JSX: JSX é uma extensão de sintaxe do JavaScript usada no ReactJS para descrever a estrutura da interface do usuário. É uma mistura de JavaScript e HTML, permitindo que você escreva componentes React de forma declarativa.

JavaScript: O ReactJS é baseado em JavaScript, então é importante ter um bom entendimento dos fundamentos da linguagem, incluindo manipulação de arrays, objetos, funções, escopo, closures e conceitos assíncronos como Promises e async/await.

DOM (Document Object Model): O ReactJS manipula o DOM virtualmente para atualizar eficientemente a interface do usuário. É útil entender o conceito de DOM e como ele funciona para aproveitar ao máximo o ReactJS.

Componentização: O ReactJS é baseado em componentes. É importante entender o conceito de componentização, onde você divide sua interface em componentes reutilizáveis e independentes, cada um com sua própria lógica e renderização.

State e Props: O ReactJS tem o conceito de estado (state) e propriedades (props). O estado é utilizado para controlar dados mutáveis em um componente, enquanto as props são usadas para passar dados de um componente pai para um componente filho. Compreender como trabalhar com state e props é essencial para criar componentes dinâmicos.

Ciclo de vida do componente: Os componentes React têm um ciclo de vida, que consiste em diferentes métodos que são executados em momentos específicos, desde a inicialização até a desmontagem do componente. É importante entender os principais métodos do ciclo de vida e quando usá-los para gerenciar o comportamento do componente.

Eventos: O ReactJS permite lidar com eventos, como cliques de botão, envio de formulários, entre outros. É importante entender como lidar com eventos em componentes React e como atualizar o estado ou invocar ações com base nesses eventos.

Gerenciamento de estado avançado: Além do estado interno de um componente, é importante entender como gerenciar o estado compartilhado entre componentes ou mesmo em toda a aplicação.

Ferramentas de desenvolvimento: Existem várias ferramentas e bibliotecas complementares que podem facilitar o desenvolvimento em ReactJS, como o React Router

para lidar com roteamento, Axios para fazer chamadas de API, ESLint para garantir a qualidade do código, e muitas outras. Familiarizar-se com essas ferramentas pode melhorar sua produtividade e a qualidade do seu código.

Como instalar e iniciar um projeto em ReactJS?

Passo 1: Instale o Node.js
O ReactJS requer o Node.js, então a primeira coisa que você precisa fazer é instalar o Node.js em seu computador. Siga as instruções abaixo:

Acesse o site oficial do Node.js em https://nodejs.org/

Na página inicial, você verá dois botões de download: **LTS** e **Current**.
Recomendo escolher a versão **LTS** (Long-Term Support) para obter a versão mais estável.
Clique no botão de **download** correspondente ao seu sistema operacional (Windows, macOS ou Linux) e aguarde o download ser concluído.
Após o download, **execute o instalador do Node.js** e siga as instruções para concluir a instalação.

Passo 2: Verifique a instalação
Após a instalação do Node.js, verifique se foi instalado corretamente. Abra o terminal ou prompt de comando e execute os seguintes comandos:

```
node -v
```

```
npm -v
```

Esses comandos exibirão as versões do Node.js e do npm (gerenciador de pacotes do Node.js), respectivamente. Se você vir as versões, isso significa que a instalação foi bem-sucedida.

Passo 3: Criação de um novo projeto React
Abra o terminal e execute o seguinte comando para criar um novo projeto React:
`` `npx create-react-app meu-app` ``
Isso criará uma nova pasta chamada "**meu-app**" com uma estrutura de projeto React pré-configurada.

*Substitua "**meu-app**" pelo nome que você deseja dar ao seu projeto.*

Passo 4: Execute o aplicativo React
Após a criação do projeto, navegue para a pasta do projeto no terminal ou prompt de comando:

```
cd meu-app
```

Em seguida, execute o seguinte comando para iniciar o aplicativo React:

```
npm start
```

Isso iniciará o servidor de desenvolvimento e abrirá seu aplicativo no navegador. Agora você pode começar a editar o código-fonte do seu aplicativo React e ver as alterações em tempo real no navegador.

E pronto! Agora você tem o ReactJS instalado em seu computador e um novo projeto React pronto para ser desenvolvido.

O que é Componente e JSX?

Em React JS, um componente é uma parte isolada e reutilizável de uma interface de usuário. Você pode pensar em um componente como um bloco de construção que pode ser combinado com outros componentes para criar uma aplicação completa. Os componentes podem conter elementos visuais, lógica e até mesmo estado.

O JSX é uma extensão de sintaxe (ou seja, uma maneira especial de escrever código) que é usada no ReactJS para descrever a estrutura e a aparência das interfaces de usuário.
É uma combinação de JavaScript com HTML.

Pense no JSX como um modo mais fácil e intuitivo de criar componentes e elementos visuais em um aplicativo React. Ele permite que você escreva código que se parece muito com o HTML que você já conhece.

Aqui está um exemplo para ilustrar como o JSX funciona:

```jsx
import React from "react";

function MeuComponente() {
  return (
    <div className="Content">
      <h1>Olá, mundo!</h1>
      <p>Este é um exemplo de JSX.</p>
      <p>Programação é vida</p>
    </div>
  );
}
function Button(props) {
  return <button onClick={props.onClick}>{props.text}</button>;
}

export default { Button, MeuComponente };
```

No exemplo acima, temos um componente chamado **MeuComponente** que retorna uma estrutura de elementos JSX. Você pode ver que estamos escrevendo tags HTML dentro do código JavaScript. O , <h1>, <p> são exemplos de elementos JSX.

E temos um componente chamado "Button". Ele recebe um objeto de props (propriedades) como argumento, que pode conter informações como o texto do botão e uma função de clique.

Uma das coisas legais sobre o JSX é que você pode incluir expressões JavaScript dentro dele. Por exemplo, você pode usar variáveis, operações matemáticas e até mesmo chamar funções. O JSX permite que você insira essas expressões diretamente no meio do código HTML-like.

O que são Props e State?

Props (propriedades) são a principal forma de passar dados de um componente pai para um componente filho em React. Elas são usadas para enviar informações específicas para um componente e permitem que os componentes sejam reutilizáveis e configuráveis.

As props são passadas para um componente como argumentos da função do componente ou através da sintaxe de atributos no JSX. Aqui está um exemplo de como as props podem ser usadas:

```
function BemVindo(props) {
  return <h1>Bem-vindo, {props.name}!</h1>;
}

function App() {
  return <BemVindo name="John" />;
}

export default App;
```

No exemplo acima, temos um componente chamado BemVindo, que exibe uma saudação com base no nome recebido como propriedade (props.name). O componente App utiliza o componente **BemVindo** e passa a propriedade **name** com o valor "**John**". Como resultado, o componente **BemVindo** irá renderizar "Bem-vindo, John!".

As props são imutáveis, o que significa que não devem ser modificadas dentro do componente que as recebe. Elas são apenas para leitura e não devem ser alteradas diretamente. Se você precisar alterar um valor associado a uma **prop**, geralmente é necessário usar um estado (**state**).

O estado (**state**) é um objeto especial no React que permite que os componentes controlem e atualizem suas próprias informações. Ele representa os dados internos de um componente que podem mudar ao longo do tempo, seja em resposta a eventos ou interações do usuário.

Para usar o estado em um componente, você precisa utilizar a função **useState** fornecida pelo React. Aqui está um exemplo básico de como o estado pode ser usado:

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

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

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

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

function App() {
  return <Contar />;
}

export default App;
```

No exemplo acima, temos um componente Contar que exibe um contador e um botão "**Increment**" para incrementar o valor do contador. Usamos a função **useState** para definir o estado inicial do contador como 0 (**const [count, setCount] = useState(0)**).

A função **useState** retorna um array com duas posições: a primeira é o valor atual do estado (count neste caso), e a segunda é uma função para atualizar o valor do estado (**setCount** neste caso). Utilizamos a função **setCount** para atualizar o valor do contador quando o botão é clicado.

Dentro do retorno do componente **Counter**, exibimos o valor do contador (**count**) e adicionamos um evento **onClick** do botão para chamar a função **increment**, que atualiza o estado.

Quando o estado é atualizado usando a função **setCount**, o React renderizar novamente o componente, atualizando automaticamente o valor do contador exibido na interface.

Espero que isso tenha esclarecido as diferenças entre props e state! As props são usadas para passar dados de um componente pai para um filho, enquanto o estado é usado para controlar e atualizar dados internos de um componente.

O que é renderização condicional?

A renderização condicional é um conceito importante em React, pois permite exibir diferentes elementos ou componentes com base em condições específicas. Você pode decidir o que renderizar com base no estado do componente, nas props recebidas ou em outras variáveis relevantes.

Existem várias maneiras de implementar a renderização condicional em React:

Condicional usando operadores lógicos:
Você pode usar operadores lógicos, como **&&** e **||**, dentro da função render() do componente para controlar quais elementos serão renderizados. Por exemplo:

```
render() {
  const isLoggedIn = this.state.isLoggedIn;

  return (
    <div>
      {isLoggedIn && <p>Usuário logado!</p>}
      {!isLoggedIn && <p>Por favor, faça login.</p>}
    </div>
  );
}
```

Neste exemplo, a renderização condicional é baseada no valor da variável isLoggedIn. Se isLoggedIn for true, o parágrafo "Usuário logado!" será renderizado. Caso contrário, se isLoggedIn for false, o parágrafo "Por favor, faça login." será renderizado.

Condicional usando o operador ternário:
O operador ternário **(condition ? trueExpression : falseExpression)** também pode ser usado para fazer renderização condicional. Veja o exemplo:

```
render() {
  const isAdmin = this.props.isAdmin;

  return (
    <div>
      {isAdmin ? <p>Usuário é administrador.</p> : <p>Usuário não é administrador.</p>}
    </div>
  );
}
```

Neste caso, a renderização condicional é baseada na prop **isAdmin**. Se **isAdmin** for **true**, o parágrafo **"Usuário é administrador."** será renderizado. Caso contrário, se **isAdmin** for **false**, o parágrafo "**Usuário não é administrador**." será renderizado.

Condicional usando componentes condicionais:
Você também pode criar componentes separados para representar diferentes estados ou condições e, em seguida, renderizá-los com base nas condições. Por exemplo:

```
function AdminMessage() {
  return <p>Usuário é administrador.</p>;
}

function UserMessage() {
  return <p>Usuário não é administrador.</p>;
}

render() {
  const isAdmin = this.props.isAdmin;

  return (
    <div>
      {isAdmin ? <AdminMessage /> : <UserMessage />}
    </div>
  );
}
```

Nesse exemplo, são criados dois componentes funcionais, **AdminMessage** e **UserMessage**, para representar as mensagens diferentes com base na condição **isAdmin**. Em seguida, o componente apropriado é renderizado com base nessa condição.

Essas são apenas algumas maneiras de implementar a renderização condicional em React. É importante lembrar que o React permite uma abordagem declarativa para a renderização condicional, facilitando a exibição de diferentes elementos com base em diferentes estados ou props.