

Desenvolvimento com React: Importando e Exportando Componentes

Site: [Geração Tech](#)
Curso: Formação em Desenvolvedor Web – Online
Livro: Desenvolvimento com React: Importando e Exportando Componentes

Impresso por: JOÃO VITOR DE MELO FREITAS
Data: quinta-feira, 18 jul. 2024, 22:39

Índice

1. Importando e Exportando Componentes

- 1.1. Configuração do Projeto
- 1.2. Vídeo da Aula
- 1.3. Exportando Vários Componentes
- 1.4. Vídeo da Aula
- 1.5. Trabalhando com Props
- 1.6. Vídeo da Aula
- 1.7. Props e Children
- 1.8. Vídeo da Aula

1. Importando e Exportando Componentes

Introdução ao Import e Export

Olá pessoal, tudo bem com vocês? Vamos continuar nosso curso de [React](#). Hoje vamos explorar os conceitos de import e export em [React](#), criando um novo projeto e entendendo como esses mecanismos funcionam.

1.1. Configuração do Projeto

Para começar, vamos configurar nosso projeto [React](#). Siga os passos abaixo:

1. Criação do Projeto:

- Abra o terminal e navegue até a pasta onde deseja criar o projeto.
- Execute o comando:

```
bash
```

```
 Copiar código
```

```
npm create vite@latest
```

- Siga as instruções para criar um projeto [React](#) com JavaScript. Nomeie o projeto como `react-aula2`.

2. Instalação das Dependências:

- Navegue até a pasta do projeto e instale as dependências:

```
bash
```

```
 Copiar código
```

```
cd react-aula2 npm install
```

3. Abrindo o Projeto no VSCode:

- Abra o projeto no VSCode:

```
bash
```

```
 Copiar código
```

```
code .
```

Conceito de Import e Export

No [React](#), usamos `import` e `export` para compartilhar e reutilizar código entre diferentes arquivos. Vamos explorar isso em detalhe.

Importando e Exportando Componentes

1. Importação:

- Usamos `import` para trazer funcionalidades de um módulo ou arquivo para outro.
- Exemplo:

```
jsx
```


```
 Copiar código
```

```
import React from 'react'; import ReactDOM from 'react-dom'; import App from './App.jsx';
```

2. Exportação:

- Usamos `export` para disponibilizar funções, objetos ou valores de um módulo ou arquivo para serem usados em outros lugares.
- Exemplo:

```
jsx
```

```
 Copiar código
```

```
const App = () => { return ( <div> <h1>Olá, React!</h1> </div> ); }; export default App;
```

Criando Componentes Separados

Vamos criar componentes separados para estruturar melhor nosso projeto.

1. Criando a Estrutura de Pastas:

- Dentro da pasta `src`, crie uma nova pasta chamada `components`.

2. Criando o Componente Header:

- Na pasta `components`, crie um arquivo chamado `Header.jsx`.
- Adicione o seguinte código:

jsx

 Copiar código

```
import React from 'react'; const Header = () => { return ( <header> <h1>Meu Header</h1> </header> ); }; export default Header;
```

3. Criando o Componente Banner:

- Na pasta `components`, crie um arquivo chamado `Banner.jsx`.
- Adicione o seguinte código:

jsx

 Copiar código

```
import React from 'react'; const Banner = () => { return ( <section> <h2>Banner</h2> </section> ); }; export default Banner;
```

Usando os Componentes no Componente Principal

Agora, vamos importar e usar esses componentes no nosso componente principal `App`.

1. Modificando o Arquivo `App.jsx`:

- Abra o arquivo `src/App.jsx` e modifique para:

jsx

 Copiar código

```
import React from 'react'; import Header from './components/Header'; import Banner from './components/Banner'; const App = () => { return ( <div> <Header /> <Banner /> </div> ); }; export default App;
```

2. Executando a Aplicação:

- No terminal, execute o comando:

bash

 Copiar código

```
npm run dev
```

- Abra o navegador e acesse `http://localhost:3000` para ver a aplicação rodando.

Conceitos Importantes

1. Fragmentos `React`:

- Quando um componente retorna múltiplos elementos, eles devem ser envoltos em um único elemento pai. Podemos usar `React.Fragment` ou a sintaxe curta `<></>`:

jsx

 Copiar código

```
return ( <> <Header /> <Banner /> </> );
```

2. Organização de Código:

- Manter componentes em arquivos separados facilita a manutenção e a escalabilidade do projeto.

Conclusão

Hoje aprendemos sobre importação e exportação de componentes em [React](#). Criamos componentes separados e os utilizamos no nosso componente principal. Esses conceitos são fundamentais para construir aplicações [React](#) modulares e reutilizáveis.

Nos vemos na próxima aula! Até lá!

1.2. Vídeo da Aula

dia 17 video 1



1.3. Exportando Vários Componentes

Introdução ao Export de Múltiplos Componentes

Olá pessoal, tudo bem com vocês? Vamos continuar nosso curso de [React](#). Na aula de hoje, vamos explorar como exportar e importar vários componentes de um mesmo arquivo. Isso é útil para organizar melhor nosso código, especialmente quando temos vários componentes relacionados.

Exportando Múltiplos Componentes

Vamos criar um arquivo com múltiplos componentes e aprender a exportá-los corretamente.

1. Criando o Arquivo de Componentes:

- Na pasta `components`, crie um novo arquivo chamado `VariosComponentes.jsx`.

2. Definindo Vários Componentes:

- Adicione o seguinte código no arquivo `VariosComponentes.jsx`:

jsx

 Copiar código

```
import React from 'react'; const Componente1 = () => { return ( <div> <h1>Componente 1</h1> </div> ); }; const Componente2 = () => { return ( <div> <h1>Componente 2</h1> </div> ); }; const Componente3 = () => { return ( <div> <h1>Componente 3</h1> </div> ); }; export { Componente1, Componente2, Componente3 };
```

Importando Múltiplos Componentes

Para usar esses componentes em outro arquivo, precisamos importá-los corretamente.

1. Modificando o Arquivo `App.jsx`:

- Abra o arquivo `src/App.jsx` e modifique-o para importar e usar os componentes que criamos:

jsx

 Copiar código

```
import React from 'react'; import { Componente1, Componente2, Componente3 } from './components/VariosComponentes'; const App = () => { return ( <div> <Componente1 /> <Componente2 /> <Componente3 /> </div> ); }; export default App;
```

Explicação dos Conceitos

1. Exportação de Múltiplos Componentes:

- Utilizamos `export { Componente1, Componente2, Componente3 }` para exportar vários componentes de um único arquivo. Isso permite que outros arquivos importem esses componentes individualmente ou em conjunto.

2. Importação com Desestruturação:

- Usamos a sintaxe de desestruturação para importar múltiplos componentes: `import { Componente1, Componente2, Componente3 }.`
- Isso nos permite importar apenas os componentes que precisamos, mantendo o código organizado e eficiente.

Testando a Aplicação

1. Executando a Aplicação:

- No terminal, execute o comando:

bash

 Copiar código

```
npm run dev
```

- Abra o navegador e acesse `http://localhost:3000` para ver a aplicação rodando com os três componentes exibidos.

Conclusão

Aprendemos como exportar e importar múltiplos componentes em [React](#), uma prática útil para organizar nosso código de maneira eficiente. Utilizamos a exportação com desestruturação para importar apenas os componentes necessários, mantendo nosso projeto modular e limpo.

Nos vemos na próxima aula! Até lá!

1.4. Vídeo da Aula

dia 17 video 2



1.5. Trabalhando com Props

Explicação sobre Componentes e Props

Olá pessoal, tudo bem com vocês? Vamos continuar nosso curso de [React](#). Hoje, vamos explorar mais sobre componentes e como eles podem receber dados dinâmicos usando props.

Criando um Componente com Props

Para explicar o conceito de props, vamos criar um novo componente. Imagine um componente como uma forma de bolo. Dependendo dos ingredientes que você colocar, você terá bolos diferentes. Da mesma forma, você pode passar diferentes propriedades (props) para um componente [React](#) para alterar seu comportamento e exibição.

1. Criando o Componente FormaDeBolo:

- Na pasta `components`, crie um novo arquivo chamado `FormaDeBolo.jsx`.
- Adicione o seguinte código ao arquivo:

jsx

 Copiar código

```
import React from 'react'; const FormaDeBolo = ({ sabor, cobertura }) => { return ( <div> <h1>Bolo de {sabor} com cobertura de {cobertura}</h1> </div> ); }; export default FormaDeBolo;
```

Explicação do Código

1. Props:

- Recebemos `sabor` e `cobertura` como props no componente `FormaDeBolo`.
- Usamos essas props para exibir diferentes tipos de bolos.

2. Desestruturação de Props:

- Usamos a desestruturação para extrair `sabor` e `cobertura` diretamente dos props. Isso torna o código mais limpo e legível.

Usando o Componente com Props

Vamos usar o componente `FormaDeBolo` no nosso componente principal `App` e passar diferentes valores para as props.

1. Modificando o Arquivo `App.jsx`:

- Abra o arquivo `src/App.jsx` e modifique-o para usar o componente `FormaDeBolo`:

jsx

 Copiar código

```
import React from 'react'; import FormaDeBolo from './components/FormaDeBolo'; const App = () => { return ( <div> <FormaDeBolo sabor="laranja" cobertura="chocolate" /> <FormaDeBolo sabor="chocolate" cobertura="chantilly" /> </div> ); }; export default App;
```

Testando a Aplicação

1. Executando a Aplicação:

- No terminal, execute o comando:

bash

 Copiar código

```
npm run dev
```

- Abra o navegador e acesse `http://localhost:3000` para ver a aplicação rodando com os diferentes bolos.

Explicação dos Conceitos

1. Componentes Dinâmicos:

- Usando props, podemos tornar os componentes dinâmicos e reutilizáveis. Passamos diferentes valores para as props e o componente renderiza de acordo com esses valores.

2. Props como Objetos:

- As props são passadas como um objeto para o componente. Podemos desestruturar esse objeto para extrair as propriedades específicas que queremos usar.

Conclusão

Aprendemos como usar props para passar dados para os componentes em [React](#). Isso nos permite criar componentes dinâmicos e reutilizáveis que podem ser configurados de diferentes maneiras. No próximo episódio, vamos explorar mais funcionalidades do [React](#) e como podemos usar props de maneira mais avançada.

Nos vemos na próxima aula! Até lá!

1.6. Vídeo da Aula

dia 17 video 3



1.7. Props e Children

Transformando Componentes Estáticos em Dinâmicos

Olá pessoal, tudo bem com vocês? Vamos continuar nosso curso de [React](#). Na aula de hoje, vamos explorar como tornar componentes mais dinâmicos utilizando props e a propriedade `children`.

Recapitulando o Conceito de Props

Props (propriedades) são usadas para passar dados de um componente pai para um componente filho. No exemplo anterior, vimos como usar props para personalizar o componente `FormaDeBolo`. Agora, vamos criar um novo componente chamado `Card` para ilustrar como as props funcionam.

Criando o Componente `Card`

1. Criando o Arquivo do Componente:

- Na pasta `components`, crie um novo arquivo chamado `Card.jsx`.
- Adicione o seguinte código ao arquivo:

jsx

 Copiar código

```
import React from 'react'; const Card = ({ title, category, content, image }) => { return ( <div style={{ border: '1px solid red', padding: '10px', margin: '10px' }}> {image} && <img src={image} alt={title} style={{ width: '100px' }} /> <h2>{title}</h2> <h6>{category}</h6> <p>{content}</p> </div> ); }; export default Card;
```

Usando o Componente `Card` com Props

Vamos usar o componente `Card` no nosso componente principal `App` e passar diferentes valores para as props.

1. Modificando o Arquivo `App.jsx`:

- Abra o arquivo `src/App.jsx` e modifique-o para usar o componente `Card`:

jsx

 Copiar código

```
import React from 'react'; import Card from './components/Card'; const App = () => { return ( <div> <Card title="Notícia 1" category="Esportes" content="Conteúdo da notícia 1" image="https://via.placeholder.com/100" /> <Card title="Notícia 2" category="Finanças" content="Conteúdo da notícia 2" /> <Card title="Notícia 3" category="Tecnologia" content="Conteúdo da notícia 3" image="https://via.placeholder.com/100" /> </div> ); }; export default App;
```

Introduzindo a Propriedade `children`

A propriedade `children` permite que você aninhe elementos dentro de outros componentes. Vamos criar um componente chamado `CardDinamico` que utiliza `children`.

1. Criando o Componente `CardDinamico`:

- Na pasta `components`, crie um novo arquivo chamado `CardDinamico.jsx`.
- Adicione o seguinte código ao arquivo:

jsx

 Copiar código

```
import React from 'react'; const CardDinamico = ({ children }) => { return ( <div style={{ border: '1px solid blue', padding: '10px', margin: '10px' }}> {children} </div> ); }; export default CardDinamico;
```

2. Usando o Componente `CardDinamico`:

- Abra o arquivo `src/App.jsx` e modifique-o para usar o componente `CardDinamico`:

jsx

 Copiar código

```
import React from 'react'; import Card from './components/Card'; import CardDinamico from
'./components/CardDinamico'; const App = () => { return ( <div> <Card title="Notícia 1" category="Esportes"
content="Conteúdo da notícia 1" image="https://via.placeholder.com/100" /> <Card title="Notícia 2"
category="Finanças" content="Conteúdo da notícia 2" /> <Card title="Notícia 3" category="Tecnologia"
content="Conteúdo da notícia 3" image="https://via.placeholder.com/100" /> <CardDinamico> <h2>Notícia Dinâmica
1</h2>  <p>Conteúdo dinâmico da notícia 1</p>
</CardDinamico> <CardDinamico> <h2>Notícia Dinâmica 2</h2> <p>Conteúdo dinâmico da notícia 2 sem imagem</p>
</CardDinamico> </div> ); }; export default App;
```

Explicação dos Conceitos

1. Props:

- São usadas para passar dados e personalizar componentes. No `Card`, usamos props como `title`, `category`, `content` e `image`.

2. Children:

- Permitem aninhar elementos dentro de outros componentes. No `CardDinamico`, usamos `children` para definir dinamicamente o conteúdo do card.

Conclusão

Aprendemos como usar props e `children` para tornar os componentes mais dinâmicos e flexíveis. Isso nos permite criar componentes reutilizáveis e altamente configuráveis.

Nos vemos na próxima aula! Até lá!

1.8. Vídeo da Aula

dia 17 video 4

