REACT EFICIENTE

REUTILIZANDO COMPONENTES

JHONATHAN ALVIM

Componentes em React

Boas Práticas para Reutilização de Componentes

Criar componentes reutilizáveis em React é essencial para construir aplicações escaláveis e de fácil manutenção. Neste ebook, vamos explorar as principais boas práticas para parametrizar e reutilizar componentes, utilizando exemplos práticos para facilitar a compreensão.

01

MANTER A SIMPLICIDADE

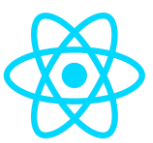
Mantenha seus componentes simples e focados em uma única responsabilidade. Isso facilita a reutilização e a manutenção..

Componentes Simples e Funcionais

Neste exemplo, o componente Button tem uma única responsabilidade: renderizar um botão com um texto e uma ação ao ser clicado.

Button.js

```
const Button = ({ label, onClick }) ⇒ {
  return <button onClick={onClick}>{label}</button>;
};
```



PASSANDO PROPS

Use props para passar dados e comportamentos para seus componentes. As props tornam seus componentes configuráveis e reutilizáveis em diferentes contextos.

Uso Adequado de Props

Neste exemplo, UserProfile recebe name, age e onProfileClick como props, permitindo que o mesmo componente seja utilizado com diferentes usuários e ações.

UserProfile.js

```
const UserProfile = ({ name, age, onProfileClick }) ⇒ {
  return (
    <div onClick={onProfileClick}>
      <h1>{name}</h1>
      <p>{age} anos</p>
    </div>
  );
};
```

DEFAULT PROPS

Definir props padrão é uma boa prática para garantir que seus componentes funcionem mesmo quando algumas props não são fornecidas.

Definindo Props Padrão

Neste exemplo, se a prop alt não for fornecida, o componente Avatar usará o valor padrão "User Avatar".

Avatar.js

```
const Avatar = ({ src, alt = "User Avatar" }) => {
  return <img src={src} alt={alt} />;
};
```

04

ESCOLHENDO O CONTROLE CERTO

Decida se seu componente será controlado (gerencia seu próprio estado) ou não controlado (estado gerenciado pelo pai).

Componentes Controlados e Não Controlados

Exemplo de Componente Controlado:
Nesse primeiro exemplo, o componente TextInput é controlado pelo pai através das props value e onChange.

TextInput.js

```
const TextInput = ({ value, onChange }) => {
  return <input type="text" value={value} onChange={onChange} />;
};
```

Exemplo de Componente Não Controlado:
Nesse segundo exemplo, o componente UncontrolledTextInput gerencia seu próprio estado internamente.

UncontrolledTextInput.js

```
const UncontrolledTextInput = () => {
  return <input type="text" />;
};
```

05

COMPOSIÇAO DE COMPONENTES

Combine componentes pequenos e focados para criar interfaces mais complexas, promovendo a reutilização e a clareza do código.

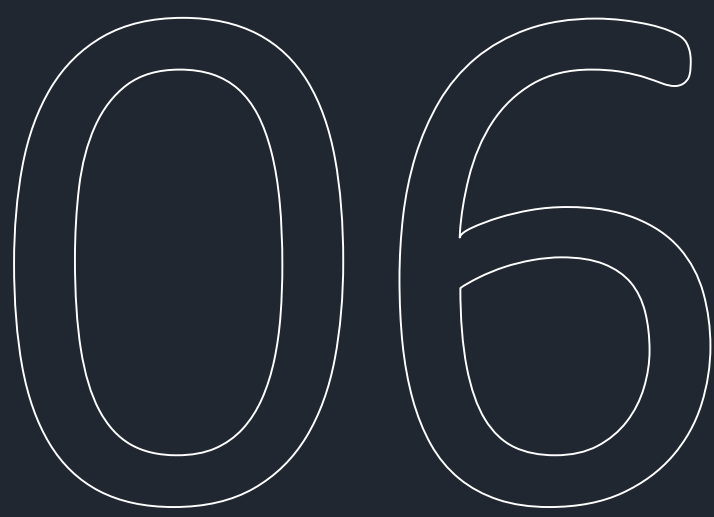
Reutilizando através da Composição

Aqui, App compõe os componentes **Header**, **MainContent** e **Footer**, mantendo cada parte da interface isolada e reutilizável.

App.js

```
const App = () => {
  return (
    <div>
      <Header />
      <MainContent />
      <Footer />
    </div>
  );
};

const Header = () => <header>Header</header>;
const MainContent = () => <main>Main Content</main>;
const Footer = () => <footer>Footer</footer>;
```



USANDO CUSTOM HOOKS

Crie hooks personalizados para reutilizar lógica de estado entre diferentes componentes.

Reutilizando através da Composição

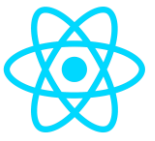
Neste exemplo, useCounter encapsula a lógica do contador, permitindo que diferentes componentes reutilizem essa lógica facilmente..

Counter.js

```
const useCounter = (initialValue = 0) => {
  const [count, setCount] = useState(initialValue);
  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);

  return { count, increment, decrement };
};

const Counter = () => {
  const { count, increment, decrement } = useCounter();
  return (
    <div>
      <button onClick={decrement}>-</button>
      <span>{count}</span>
      <button onClick={increment}>+</button>
    </div>
  );
};
```



Utilize essas dicas

Boas Práticas para Reutilização de Componentes

Seguir **essas boas práticas** ao criar componentes em React não só torna seu código mais limpo e organizado, mas também promove a reutilização e a escalabilidade da sua aplicação. Lembre-se de sempre manter seus componentes simples, validar props, utilizar componentes controlados conforme necessário e aproveitar a composição e hooks personalizados para uma lógica reutilizável.