Componentes compartilhados - Botão

Introdução

Neste ebook, exploraremos o processo de criação de componentes reutilizáveis em React, com foco especial na implementação de botões customizados. Abordaremos conceitos fundamentais como props, renderização condicional e estilização de componentes. Este guia é ideal para desenvolvedores que desejam aprimorar suas habilidades em React e criar interfaces de usuário mais dinâmicas e flexíveis.
Configuração Inicial do Componente de BotãoEstrutura de ArquivosPara iniciar a criação do nosso componente de botão, seguiremos uma estrutura de arquivos organizada:

- Crie uma pasta chamada button dentro do diretório de componentes
- Dentro desta pasta, crie dois arquivos:button.jsx: para a lógica do componente
- button.css: para os estilos

Configuração Básica do ComponenteNo arquivo button.jsx, começamos com a estrutura básica do componente:

import React from 'react'; import './button.css'; const Button = ({ arrow, buttonStyle, loading, children, ...props }) => { return (<button className={`button ${buttonStyle}`} {...props}>

{children} </button>); }; export default Button;

Pontos importantes:

- Utilizamos desestruturação para receber as props
- O operador spread ...props permite passar propriedades nativas do HTML
- children é utilizado para renderizar o conteúdo interno do botão

Estilização do Componente de BotãoEstilos BaseNo arquivo button.css, definimos os estilos base para todos os botões:
.button { border-radius: 18px; border: none; column-gap: 10px; cursor: pointer; display: flex; font-size: 18px; font-weight: 600; padding: 25px 50px; transition: background-color 0.3s ease; }

Variações de EstiloCriamos diferentes classes para cada variação de botão:

1. **Botão Primário**

.button.primary { background-color: #0cc0f2; color: white; } .button.primary:hover { background-color: #0056B3; }

1. **Botão Secundário**

.button.secondary { background-color: #1D1D1D; color: white; } .button.secondary:hover { background-color: #4D4D4D; }

1. **Botão Outline**

.button.outline { background-color: white; color: #0cc0f2; border: 2px solid #0cc0f2; } .button.outline:hover { background-color: #0cc0f2; color: white; }

1. **Botão Desabilitado**

.button:disabled, .button.disabled { background-color: #CCCCCC; color: #666666; cursor: not-allowed; } .button:disabled:hover, .button.disabled:hover { background-color: #CCCCCC; }

Implementação de Props e Renderização CondicionalUtilizando Props para CustomizaçãoAs props permitem customizar o botão de forma dinâmica. No componente Button, utilizamos as seguintes props:

- buttonStyle: define o estilo do botão (primary, secondary, outline)
- disabled: controla se o botão está desabilitado
- arrow: determina se o ícone de seta deve ser exibido

Renderização Condicional do Ícone de SetaImplementamos a renderização condicional do ícone de seta:

import whiteArrow from '../assets/white-arrow.svg'; // ... dentro do componente Button {arrow && }

Esta lógica exibe o ícone apenas quando a prop arrow é verdadeira.

Utilização do Componente de BotãoExemplo de UsoPara utilizar o componente de botão em outras partes da aplicação:

import Button from './components/button/Button'; // ... dentro de outro componente <Button buttonStyle="primary" arrow> Clique Aqui </Button> <Button buttonStyle="secondary" disabled> Botão Desabilitado </Button> <Button buttonStyle="outline"> Botão Outline </

Button>

- Dicas de UsoUtilize buttonStyle para definir o estilo visual do botão
- Passe disabled como prop para desabilitar o botão
- Use arrow para adicionar o ícone de seta (apenas em botões primários e secundários)

Conclusão

A criação de componentes reutilizáveis, como o botão customizado que desenvolvemos, é fundamental para construir interfaces consistentes e fáceis de manter em React. Através do uso de props e renderização condicional, conseguimos criar um componente flexível que pode ser facilmente adaptado para diferentes contextos na aplicação.
Pontos-chave aprendidos:

- Estruturação de componentes reutilizáveis
- Utilização de props para customização
- Implementação de estilos condicionais
- Renderização condicional de elementos

Ao dominar essas técnicas, você estará bem preparado para criar componentes mais complexos e construir interfaces de usuário