Props em React

Para que servem e como utilizar pra modularizar seu código

Eduarda Vieira Gonçalves

Índice

O que são props?

- Props ou “propriedades” → dados ou comportamentos que **passamos** para um **componente**.
- Permitem **customizar** instâncias do componente **sem alterar seu código** interno.
- Ele evita a repetição de código e ajuda na reutilização de componentes
- São imutáveis do ponto de vista do componente filho (o componente que recebe props não deve modificar diretamente).
- Em **TypeScript**, podemos definir **tipos** pra props, o que **ajuda a detectar erros** e documentar o que o componente espera.

Eduarda Vieira Gonçalves

Exemplo Básico de Props

Componentes de Função (Function Components)

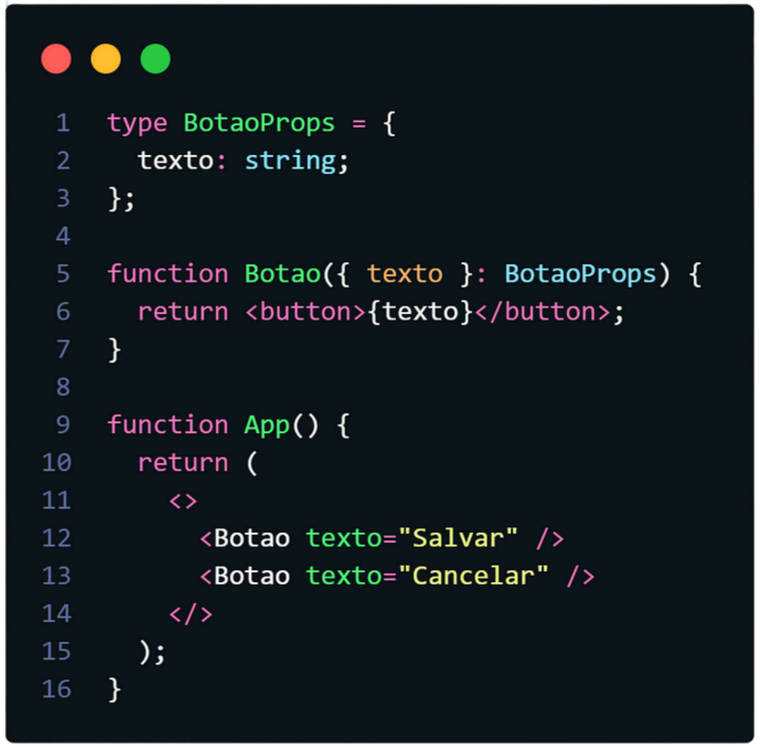
- Forma mais moderna e usada atualmente.
- Escrito como uma função JavaScript.
- Retorna o que será exibido na tela (JSX).
- Pode usar Hooks (como useState, useEffect).

Componentes de Classe (Class Components)

- Mais antigos (antes dos Hooks existirem).
- Escritos como classes JavaScript.
- Possuem métodos de ciclo de vida.
- Ainda funcionam, mas são menos usados hoje.

Eduarda Vieira Gonçalves

Exemplo simples

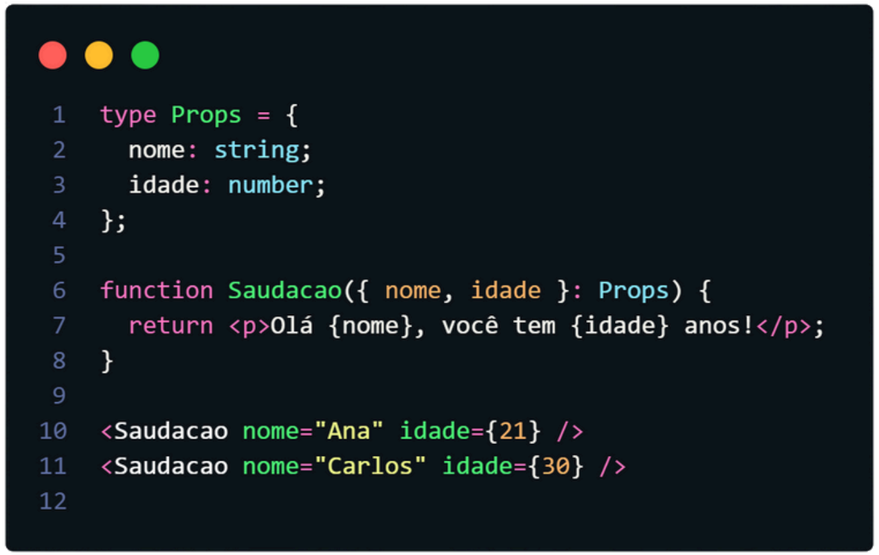


```tsx
type BotaoProps = {
  texto: string;
};

function Botao({ texto }: BotaoProps) {
  return <button>{texto}</button>;
}

function App() {
  return (
    <>
      <Botao texto="Salvar" />
      <Botao texto="Cancelar" />
    </>
  );
}
```

Explicação:

- BotaoProps define que o componente Botao **espera uma prop** texto do **tipo string**.
- Botao({ texto }) → usamos desestruturação para pegar texto das props.
- No App, usamos <Botao texto="Salvar" /> ou <Botao texto="Cancelar" /> → instâncias diferentes.

Props com Tipagem (TypeScript)



```tsx
type Props = {
  nome: string;
  idade: number;
};

function Saudacao({ nome, idade }: Props) {
  return <p>Olá {nome}, você tem {idade} anos!</p>;
}

<Saudacao nome="Ana" idade={21} />
<Saudacao nome="Carlos" idade={30} />
```

Por que tipar as props?

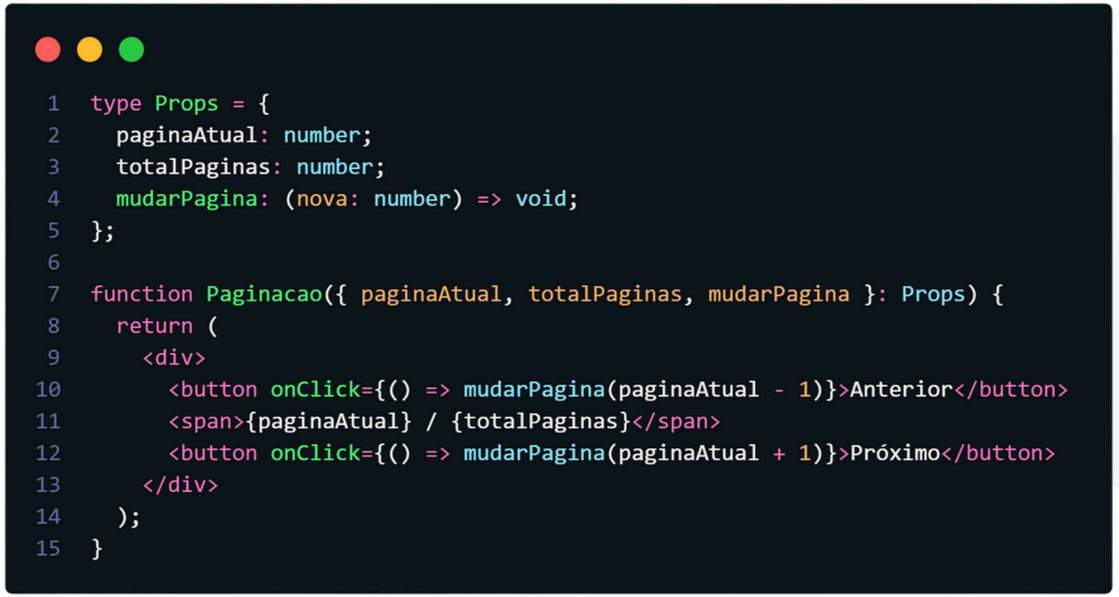
- Organização: deixa claro quais dados o componente espera.
- Segurança: evita passar valores errados (ex.: número no lugar de string).
- Produtividade: editores como VS Code mostram autocompletar das props.
- Escalabilidade: em projetos grandes, facilita manutenção.

Passos principais:

- Criamos um type Props → define quais propriedades o componente recebe.
- Passamos esse Props como tipo do parâmetro do componente.
- Usamos desestruturação ({ nome, idade }) para acessar as props.

Eduarda Vieira Gonçalves

Props Mais Complexas



```tsx
type Props = {
  paginaAtual: number;
  totalPaginas: number;
  mudarPagina: (nova: number) => void;
};

function Paginacao({ paginaAtual, totalPaginas, mudarPagina }: Props) {
  return (
    <div>
      <button onClick={() => mudarPagina(paginaAtual - 1)}>Anterior</button>
      <span>{paginaAtual} / {totalPaginas}</span>
      <button onClick={() => mudarPagina(paginaAtual + 1)}>Próximo</button>
    </div>
  );
}
```

Exemplo: Paginação

- paginaAtual (número) → mostra em qual página estamos.
- totalPaginas (número) → quantas páginas existem no total.
- mudarPagina (função) → muda para a próxima ou anterior.

Nem sempre props são apenas texto

- Podem ser números → ex.: página atual, idade.
- Podem ser booleanos → ex.: isLoading, ativo.
- Podem ser funções → ex.: executar algo quando clicado.
- Podem ser objetos → ex.: dados de um usuário.

Eduarda Vieira Gonçalves

Extensão de Props HTML + Props Opcionais

```tsx
type Props = React.ComponentProps<'input'> & {
  label?: string; // opcional
};

function Input({ label, type = "text", ...rest }: Props) {
  return (
    <label>
      {label}
      <input type={type} {...rest} />
    </label>
  );
}

// Uso
<Input label="Nome" placeholder="Digite seu nome" />
<Input type="password" placeholder="Senha" />
```

Problema comum:

E se quisermos que um componente aceite todas as props padrão do HTML (como placeholder, disabled, onChange)?

Solução:

- Usar React.ComponentProps<'tag'> → herda todas as props da tag HTML.
- Criar props opcionais com ?.(Pode ter ou não)
- Definir valores padrão (ex.: type="text").

- { label, type = "text", ...rest } → aqui estamos pegando:
- label → texto que aparece antes do input.
- type → se não for informado, o padrão é "text".
- ...rest → guarda todas as outras props passadas (placeholder, id, required, etc).

- Dentro do <input {...rest} /> → o React espalha essas props.
- Se você passar <Input label="Nome" placeholder="Digite aqui" />,
- o placeholder="Digite aqui" vai parar dentro de ...rest
- e depois é aplicado no <input>.

Eduarda Vieira Gonçalves

Props Avançadas com Variantes

Props do componente

```tsx
type Props = {
  texto: string;                    // Texto exibido no botão
  variant?: "primary" | "secondary"; // Tipo de estilo do botão
  isLoading?: boolean;              // Estado de carregamento
};
```

Objetivo:

- Mostrar como usar props para mudar o estilo e comportamento de um componente, tornando-o flexível e reutilizável.

- variant → define estilo visual do botão.
- isLoading → controla se o botão mostra carregando e fica desabilitado.
- ? indica que a prop é opcional.

Eduarda Vieira Gonçalves

Props Avançadas com Variantes

Função do componente

```tsx
function Button({ texto, variant = "primary", isLoading = false }: Props) {
  const base = "px-4 py-2 rounded text-white";
  const style =
    variant === "primary" ? "bg-blue-500" : "bg-gray-500";

  return (
    <button className={`${base} ${style}`} disabled={isLoading}>
      {isLoading ? "Carregando..." : texto}
    </button>
  );
}
```

Como usar:

<Button texto="Salvar" />
<Button texto="Cancelar" variant="secondary" />
<Button texto="Entrar" isLoading />

- variant = "primary" → valor padrão se não for passado.
- isLoading = false → botão normal por padrão.
- className={${base} ${style}} → combina classes CSS dinâmicas.
- disabled={isLoading} → desativa botão quando estiver carregando.
- {isLoading ? "Carregando..." : texto} → muda o conteúdo do botão conforme o estado.

Eduarda Vieira Gonçalves

Obrigada!