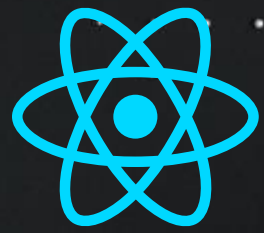
FIAP

FRONT-END DESIG ENGINEERING



REACT PROPS

Prof. Alexandre Carlos	profalexandre.jesus@fiap.com.br	www.linkedin.com/in/alexandre-carlos-de-jesus-57355a34/
Prof. Luís Carlos	lsilva@fiap.com.br	www.linkedin.com/in/luis-c-s-silva
Prof. Wellington Cidade	profwellington.tenorio@fiap.com.br	www.linkedin.com/in/wellingtoncidade

O QUE SÃO PROPS?

REACT PROPS

Em React, props (abreviação de "properties") são os parâmetros que você passa para componentes React, permitindo que você configure ou personalize o comportamento e a aparência desses componentes.

Quando você usa TypeScript com React, você pode definir a estrutura das props utilizando interfaces ou types, o que garante que as propriedades passadas sejam do tipo esperado, proporcionando maior segurança e melhor autocompletar durante o desenvolvimento.

Uma observação muito importante é que, no React, as informações são passadas apenas de componentes pais para componentes filhos, não podendo seguir o fluxo contrário. É o que chamamos de “One way data flow"!

One way data flow
O fluxo de dados só podem seguir em uma única direção.

UTILIZANDO PROPS

PASSANDO UMA VARIÁVEL

Quando props é passada para o componente filho como um único dado, devemos criar um atributo para que seja referenciado na declaração do componente filho que está no componente pai.

src > App.tsx > ...

```tsx
import Cabecalho from "./Cabecalho/Cabecalho";

function App() {

  const titulo:string = "Página Inicial";

  return (
    <>
      <h1>Componente App</h1>
      <Cabecalho titulo={titulo}/>
    </>
  );
}

export default App;
```

```
Type '{ titulo: string; }' is not assignable to type
'IntrinsicAttributes'.
  Property 'titulo' does not exist on type
'IntrinsicAttributes'. ts(2322)

(property) titulo: string

View Problem (Alt+F8)   No quick fixes available
```

UTILIZANDO PROPS

PASSANDO UMA VARIÁVEL

No componente filho devemos adicionar no construtor da função a props seguida da tipagem. Observe que quando passamos os valores do pai para o filho, mesmo que seja apenas uma variável é gerado um objeto que neste caso chamamos de props. Por isso temos que passar a tipagem para o atributo que está dentro do objeto declarado.

src > Cabecalho > Cabecalho.tsx > ...

```tsx
export default function Cabecalho(props:{titulo:string}) {

    document.title = props.titulo;

  return (
    <header>
        <h1>{props.titulo}</h1>
    </header>
  )
}
```

Agora que o dado foi declarado e tipado corretamente, podemos utilizar ele para as funções dentro do componente filho.

UTILIZANDO PROPS

PASSANDO UMA FUNÇÃO

Neste exemplo, estamos passando uma função do componente pai para o componente filho. Repare que temos que declarar o seu tipo “Function”, da mesma forma que fazemos com os atributos.
Outro detalhe importante é que, quando declaramos ele no componente pai, não devemos colocar o construtor na frente, só quando ele é usado no componente filho.

src > App.tsx > ...

```tsx
import Cabecalho from "./Cabecalho/Cabecalho";

function App() {

  const titulo:string= "Página Inicial";

  const aviso = ()=> alert('Aviso vindo do Pai!!!');

  return (
    <>
      <h1>Componente App</h1>
      <Cabecalho titulo={titulo} aviso={aviso}/>
    </>
  );
}

export default App;
```

src > Cabecalho > Cabecalho.tsx > ...

```tsx

export default function Cabecalho(props:{titulo:string, aviso:Function}) {

    document.title = props.titulo;

  return (
    <header>
        <h1>{props.titulo}</h1>
        <button onClick={()=> props.aviso()}>Aviso do App</button>
    </header>
  )
}
```

UTILIZANDO PROPS

DESTRUCTURING PROPS

Como o props é um objeto disponibilizado para o componente filho, podemos usar o destructuring para visualizar melhor os valores recebidos e também deixar o código mais limpo, uma vez que não precisamos mais ficar colocando o props antes do nome do atributo ou função. Lembrando que a palavra “Props” é uma boa prática de mercado, mas não é obrigatória. Poderia ser qualquer outra.

src > Cabecalho > Cabecalho.tsx > ...

```tsx

export default function Cabecalho({titulo,aviso}:{titulo:string, aviso:Function}) {

    document.title = titulo;

  return (
    <header>
        <h1>{titulo}</h1>
        <button onClick={()=> aviso()}>Aviso do App</button>
    </header>
  )
}

```

UTILIZANDO PROPS

PROPS UTILIZANDO TYPES

Uma forma de deixarmos nosso código mais organizado e com uma visualização mais limpa é utilizando o “type”. O type é uma palavra chave que usamos para definir tipos personalizados de objetos. Então, depois de criado conforme abaixo, só precisamos colocar ele no lugar da tipagem do nosso props.

src > Cabecalho > Cabecalho.tsx > ...

```tsx

type CabecalhoProps = {
    titulo:string;
    aviso:Function;
}

export default function Cabecalho({titulo,aviso}:CabecalhoProps) {

    document.title = titulo;

  return (
    <header>
        <h1>{titulo}</h1>
        <button onClick={()=> aviso()}>Aviso do App</button>
    </header>
  )
}

```

UTILIZANDO PROPS

UNION-TYPES(|) / ITERSECTION(**VALORES**)

Além de toda a organização que o type traz, podemos também aproveitar os benefícios de utilizar o UNION-TYPES(|) e as INTERSECÇÕES(**VALORES**). Mas como podemos fazer isso, veja os exemplos abaixo:

src > App.tsx > ...

```tsx
import Cabecalho from "./Cabecalho/Cabecalho";

function App() {

  const pagina:string = "Página Inicial";
  const nrPagina:number = 2;
  const status = "loading";
  const aviso = ()=> alert('Aviso vindo do Pai!!!');

  return (
    <>
      <h1>Componente App</h1>
      <Cabecalho status={status} pagina={pagina} nrPagina={nrPagina} aviso={aviso}/>
    </>
  );
}

export default App;
```

src > Cabecalho > Cabecalho.tsx > ...

```tsx
type CabecalhoProps = {
    pagina:string;
    nrPagina:string | number;
    aviso:Function;
    status: "deployed" | "loading";
}

export default function Cabecalho({pagina,nrPagina,aviso,status}:CabecalhoProps) {

    if(typeof nrPagina === "number"){
        document.title = status + " - " + nrPagina;
        nrPagina = "Página nr: " + nrPagina;
    }else{
        document.title = status + " - ??";
        nrPagina = "Página nr: ??";
    }

  return (
    <header>
        <h1>{pagina +"\n"+nrPagina}</h1>
        <button onClick={()=> aviso()}>Aviso do App</button>
    </header>
  );
}
```

UTILIZANDO PROPS

ITERSECTION(&)

Uma outra opção que nos ajuda na recepção de dados é o "&", que nos permite grupar mais de um type no construtor do Componente filho.

src > Cabecalho > Cabecalho.tsx > ...

```tsx
type CabecalhoProps = {
    pagina:string;
    nrPagina:string | number;
}

type Cabecalho2Props = {

    aviso:Function;
    status: "deployed" | "loading";
}

export default function Cabecalho({pagina,nrPagina,aviso,status}:CabecalhoProps & Cabecalho2Props) {

    if(typeof nrPagina === "number"){
        document.title = status + " - " + nrPagina;
        nrPagina = "Página nr: " + nrPagina;
    }else{
        document.title = status + " - ??";
        nrPagina = "Página nr: ??";
    }

  return (
    <header>
        <h1>{pagina +"\n"+nrPagina}</h1>
        <button onClick={()=> aviso()}>Aviso do App</button>
    </header>
  );
}
```

UTILIZANDO PROPS

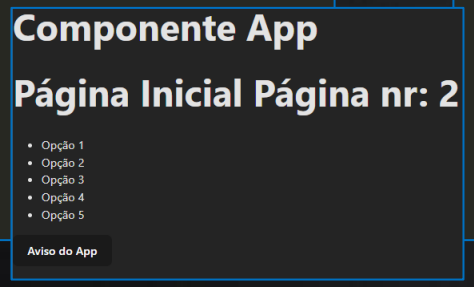
PROPS CHILDREN - REACTNODE

Quando queremos passar elementos ao invés de variáveis ou funções, devemos usar a palavra reservada "children", passando os elementos entre as tags do componente filho declarado no pai, também é muito importante declarar ele como ReactNode.

src > App.tsx > ...

```tsx
import Cabecalho from "./Cabecalho/Cabecalho";

function App() {

  const pagina:string = "Página Inicial";
  const nrPagina:number = 2;
  const status = "loading";
  const aviso = ()=> alert('Aviso vindo do Pai!!!');

  return (
    <>
      <h1>Componente App</h1>
      <Cabecalho status={status} pagina={pagina} nrPagina={nrPagina} aviso={aviso}>
        <ul>
          <li>Opção 1</li>
          <li>Opção 2</li>
          <li>Opção 3</li>
          <li>Opção 4</li>
          <li>Opção 5</li>
        </ul>
      </Cabecalho>
    </>
  );
}

export default App;
```

props.children



src > Cabecalho > Cabecalho.tsx > ...

```tsx
type CabecalhoProps = {
    pagina:string;
    nrPagina:string | number;
}

type Cabecalho2Props = {
    children: React.ReactNode
    aviso:Function;
    status: "deployed" | "loading";
}

export default function Cabecalho({pagina,nrPagina,aviso,status,children}:CabecalhoProps & Cabecalho2Props) {

    if(typeof nrPagina === "number"){
        document.title = status + " - " + nrPagina;
        nrPagina = "Página nr: " + nrPagina;
    }else{
        document.title = status + " - ??";
        nrPagina = "Página nr: ??";
    }

  return (
    <header>
        <h1>{pagina +"\n"+nrPagina}</h1>
        {children}
        <button onClick={()=> aviso()}>Aviso do App</button>
    </header>
  );
}
```

UTILIZANDO PROPS

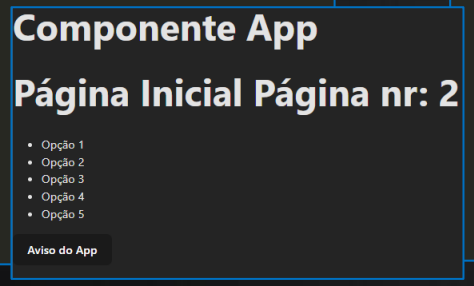
PROPS UTILIZANDO INTERFACE

A utilização de interface além da tipagem possibilita a personalização e padronização de Objetos e valores dentro dos componentes. Ela é comumente utilizada em conjunto com estruturas de classes para amarrar os diversos tipos de método e seus parâmetros principalmente, diferentemente do se busca quando utilizamos o types que é mais difundindo e utilizado atualmente no mercado.

src > App.tsx > ...

```tsx
import Cabecalho from "./Cabecalho/Cabecalho";

function App() {

  const pagina:string = "Página Inicial";
  const nrPagina:number = 2;
  const status = "loading";
  const aviso = ()=> alert('Aviso vindo do Pai!!!');

  return (
    <>
      <h1>Componente App</h1>
      <Cabecalho status={status} pagina={pagina} nrPagina={nrPagina} aviso={aviso}>
        <ul>
          <li>Opção 1</li>
          <li>Opção 2</li>
          <li>Opção 3</li>
          <li>Opção 4</li>
          <li>Opção 5</li>
        </ul>
      </Cabecalho>
    </>
  );
}

export default App;
```

props.children



src > Cabecalho > Cabecalho.tsx > ...

```tsx
type CabecalhoProps = {
    pagina:string;
    nrPagina:string | number;
}

interface Cabecalho2Props  {
    children: React.ReactNode
    aviso:Function;
    status: "deployed" | "loading";
}


export default function Cabecalho({pagina,nrPagina,aviso,status,children}:CabecalhoProps & Cabecalho2Props) {

    if(typeof nrPagina === "number"){
        document.title = status + " - " + nrPagina;
        nrPagina = "Página nr: " + nrPagina;
    }else{
        document.title = status + " - ??";
        nrPagina = "Página nr: ??";
    }

  return (
    <header>
        <h1>{pagina +"\n"+nrPagina}</h1>
        {children}
        <button onClick={()=> aviso()}>Aviso do App</button>
    </header>
  );
}
```

UTILIZANDO PROPS

EXEMPLO PRÁTICO

src > App.tsx > ...

```tsx
import Cabecalho from "./Cabecalho/Cabecalho";
import { dadosAluno } from "./types";
import Cards from "./Cabecalho/Cards";

function App() {

  const pagina:string = "Página Inicial";
  const nrPagina:number = 2;
  const status = "loading";
  const aviso = ()=> alert('Aviso vindo do Pai!!!');

  const alunos: dadosAluno[] =[
    {nome:'João', idade: 25},
    {nome:'Maria', idade: 30},
    {nome:'Pedro', idade: 35}
  ];

  return (
    <>
      <h1>Componente App</h1>
      <Cabecalho status={status} pagina={pagina} nrPagina={nrPagina} aviso={aviso}>
        <ul>
          <li>Opção 1</li>
          <li>Opção 2</li>
          <li>Opção 3</li>
          <li>Opção 4</li>
          <li>Opção 5</li>
        </ul>
      </Cabecalho>
      {
        alunos.map((aluno,i)=>( <Cards key={i} nome={aluno.nome} idade={aluno.idade}/>))
      }
    </>
  );
}

export default App;
```

src > Cabecalho > Cards.tsx > ...

```tsx
import { dadosAluno } from "../types"
import { chaves } from "../types"

export default function Cards({key, nome,idade}:dadosAluno&chaves) {
  return (
    <div key={key}>O aluno {nome} tem {idade}</div>
  )
}
```

src > TS types.ts > ...

```ts
export type dadosAluno = {

    nome:string;
    idade:number
}

export type chaves ={
    key:number
}
```

Componente App

Página Inicial Página nr: 2

- Opção 1
- Opção 2
- Opção 3
- Opção 4
- Opção 5

Aviso do App

O aluno João tem 25
O aluno Maria tem 30
O aluno Pedro tem 35

EXERCÍCIO PROPS

COMPONENTIZAÇÃO E TIPAGEM DE PROPS

Objetivo:

Neste exercício, você irá praticar a componentização e a tipagem de props em um projeto React utilizando TypeScript. O objetivo é construir uma aplicação simples que siga as melhores práticas de desenvolvimento web, respeitando os padrões estabelecidos pelo W3C.

Requisitos:

- Estrutura do Projeto:

- Crie um projeto React utilizando Vite e TypeScript.
- Organize o projeto em componentes separados, conforme descrito abaixo.
- Crie um arquivo de tipos (types.ts) para definir a tipagem das props que serão utilizadas.

- Componentização:

- O projeto deve conter os seguintes componentes:
  - **App**: Componente principal da aplicação que reúne os demais componentes.
  - **Cabecalho**: Representa o cabeçalho da página e deve conter:
    - Um elemento ***header*** contendo:
      - Um título (h1), que será passado como props.
      - Um logo, carregado a partir de uma imagem localizada na pasta public, também passada como props.
  - **Conteudo**: Representa o conteúdo principal da página e deve conter:
    - O componente **Cards**, que receberá um array de objetos como props e exibirá os dados.
  - **Rodape**: Representa o rodapé da página e deve exibir um texto passado como props.

Continua...

EXERCÍCIO PROPS

COMPONENTIZAÇÃO E TIPAGEM DE PROPS

- **Cards**: Este componente receberá uma lista de objetos como props. Cada objeto deve conter as seguintes propriedades:
  - **id** (número único),**nome** (string),**rm** (string),**avatar** (caminho para a imagem do avatar).
  - Um quantidade mínima de 10 objetos.

Dados:

- O componente **App** deve fornecer os dados necessários para os componentes filhos diretos, através de props. Utilize a seguinte estrutura de dados:
- **Título da Página**: Deve ser exibido na aba do navegador e também no componente Cabecalho.
- **Cabeçalho**: Exiba o título e o logo fornecidos.
- **Conteúdo**: Exiba uma lista de cards, onde cada card representará um aluno com um nome, RM, e avatar. Os avatares estarão armazenados em src/assets/img/.
- **Rodapé**: Exiba um texto de rodapé que será passado como prop.

Implementação:

- Defina a tipagem das props no arquivo types.ts.

Padrões W3C:

- Certifique-se de que a estrutura HTML gerada esteja em conformidade com os padrões do W3C.

OBRIGADO

FIAP


FIAP