Front-end: React

React - Aula 03

Prof. MSc. Kelson Almeida

Agenda

- If / Else no JSX
- Props e Destructuring props
- Reaproveitando Componentes
- Fragments
- Prop Children

If / else + JSX

- No exemplo, a variável é utilizada para verificar se o usuário está logado ou não.
- Em seguida, a variável “greeting” é atribuída ao elemento JSX <h1> com base no valor de “isLoggedIn”.

```javascript
import React from 'react';

function Example(props) {
  const isLoggedIn = props.isLoggedIn;
  let greeting;

  if (isLoggedIn) {
    greeting = <h1>Welcome back!</h1>;
  } else {
    greeting = <h1>Please log in.</h1>;
  }

  return (
    <div>
      {greeting}
    </div>
  );
}
```

If / else + JSX

- Neste exemplo, o operador ternário é usado para criar uma expressão “inline” no elemento JSX.

- Dependendo do valor de “isLoggedIn” o retorno será <h1>Welcome back!</h1> ou <h1>Please log in.</h1>.

javascript

```javascript
import React from 'react';

function Example(props) {
  const isLoggedIn = props.isLoggedIn;

  return (
    <div>
      {isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please log in.</h1>}
    </div>
  );
}
```

Vamos praticar?

- Crie um componente chamado: EstouConseguindoAprenderReact
- Crie um operador ternário para verificar uma props chamada “estouConseguindo” (boolean).
- Se true, o retorno em tela será: <h1>Estou indo bem…</h1>
- Se false, o retorno em tela será: <h1>Preciso estudar mais </h1>

Props

- Abreviação de propriedades.
- Podemos passar valores de componente pai para um componente filho.
- No exemplo ao lado, se quisermos passar um dado para o componente “Filho”, podemos fazer isso através do props.
- Neste caso, foi passada a string “Olá, mundo!” do Pai para o Filho através de “props.mensagem”.
- Note a sintaxe do envio do dado.
- Note a sintaxe do recebimento.

```jsx
function Pai() {
  return <Filho />;
}
```

```jsx
function Pai() {
  return <Filho mensagem="Olá, mundo!" />;
}

function Filho(props) {
  return <p>{props.mensagem}</p>;
}
```

Destructuring em Props

- E se quisermos passar várias props através de um único componente?
- Existe uma solução chamada “Destructuring em Props”, ou Desestruturando Props, que vai nos permitir separar, de forma mais concisa, os dados que vamos passar de um componente pai para um componente filho.
- Não vamos mais precisar usar: props.algumDado

jsx

```jsx
function Filho(props) {
  return <p>{props.nome} tem {props.idade} anos.</p>;
}
```

Desestruturando as props nome e idade

jsx

```jsx
function Filho({ nome, idade }) {
  return <p>{nome} tem {idade} anos.</p>;
}
```

jsx

```jsx
function Pai() {
  return <Filho nome="Ana" idade={30} />;
}
```

Reaproveitamento de Componentes

- Reaproveitamento de Componentes é uma das principais vantagens do React JS.
- Criar componentes que podem ser reutilizados em diferentes partes da aplicação.
- O exemplo ao lado renderiza uma lista de itens. Esse componente pode receber uma propriedade que define qual tipo de item deve ser renderizado (produto) ou postagem.
- Ou seja, pode-se reaproveitar o mesmo componente em diferentes partes da aplicação para renderizar diferentes tipos de itens.

jsx

```jsx
import React from 'react';

function ItemList({ items, itemType }) {
  return (
    <ul>
      {items.map(item => (
        <li key={item.id}>
          {itemType === 'product' ? (
            <p>
              {item.name} - {item.price}
            </p>
          ) : (
            <p>
              {item.title} - {item.author}
            </p>
          )}
        </li>
      ))}
    </ul>
  );
}
```

```jsx
function App() {
  const products = [
    { id: 1, name: 'Product 1', price: 10.0 },
    { id: 2, name: 'Product 2', price: 15.0 },
    { id: 3, name: 'Product 3', price: 20.0 },
  ];

  const posts = [
    { id: 1, title: 'Post 1', author: 'Author 1' },
    { id: 2, title: 'Post 2', author: 'Author 2' },
    { id: 3, title: 'Post 3', author: 'Author 3' },
  ];

  return (
    <div>
      <h2>Products</h2>
      <ItemList items={products} itemType="product" />

      <h2>Posts</h2>
      <ItemList items={posts} itemType="post" />
    </div>
  );
}

export default App;
```

Vamos praticar?

- Crie um componente chamado “Aluno” que vai renderizar informações sobre um aluno, como: nome, email e curso.
- Em vez de acessar cada propriedade individualmente dentro do componente, utilize a técnica de destructuring para extrair as propriedades “nome”, “email” e “curso” de uma vez só
- Renderize o componente “Aluno” em um elemento da página.
- Para isso crie um array com 3 alunos em objetos literais.
- Percorra esse array através de .map para renderizar esses 3 alunos na tela.

Fragments

- Quando precisar ter mais de um elemento pai em um componente.
- Outros elementos como uma div pode exigir que criemos mais regras de CSS desnecessárias…
- E se criarmos um elemento “vazio” que evite elementos pais que não irão ser necessariamente utilizados?

```jsx
function App() {
  return (
    <>
      <h1>Título</h1>
      <p>Parágrafo 1</p>
      <p>Parágrafo 2</p>
    </>
  );
}
```

Vamos praticar?

- **Refaça o exercício anterior utilizando Fragments no componente “Aluno” para evitar o elemento pai “ ”:**
- Crie um componente chamado “Aluno” que vai renderizar informações sobre um aluno, como: nome, email, curso, media e status.
- Em vez de acessar cada propriedade individualmente dentro do componente, utilize a técnica de destructuring para extrair as propriedades “nome”, “email”, “curso”, “media” de uma vez só
- Renderize o componente “Aluno” em um elemento da página.
- Para isso crie um array com 3 alunos em objetos literais.
- Percorra esse array através de .map para renderizar esses 3 alunos na tela.
- **Para o item “status” deve ser exibida a mensagem “APROVADO(A)” ou “REPROVADO(A)”**
    - **Se a média for maior ou igual a 7.00 -> Aprovado**
    - **Se não -> Reprovado.**