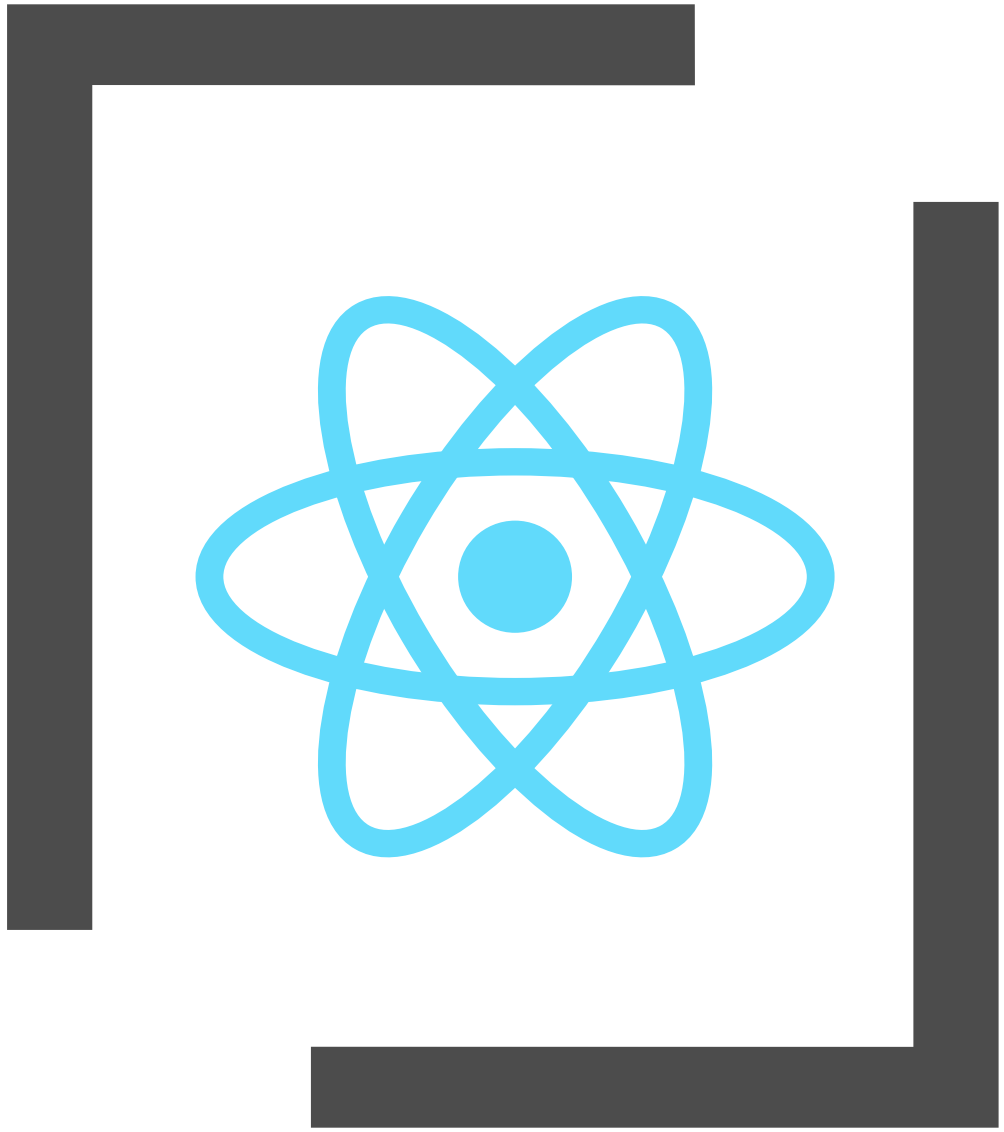
UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Desenvolvedores de softwares (DEVs)

Professor: Helder Morais

helder.morais@posgrad.ufsc.br

CTC • UFSC
Informática e estatística



React – 21/09/2021

1. Hello World
2. Introduzindo JSX
3. Renderizando Elementos
4. Componentes e Props

Conceitos gerais sobre React

- **Ano lançamento:** 2013;
- **Criado por:** Facebook.
- **Extensão do arquivo:** js
- **Especificação:** Escrito em JavaScript.
- **Modelo do desenvolvimento:** Software de código aberto.
- **Paradigma:** funcional.
- **Aplicação:** front-end.

Hello, world!

Render

```
ReactDOM.render(
    <h1>Hello, world!</h1>,
    document.getElementById('root')
  );
```

Introduzindo JSX

- **Incorporando Expressões em JSX**
- **JSX Também é uma Expressão**
- **Especificando Atributos com JSX**
- **Especificando Elementos Filhos com JSX**
- **JSX Previne Ataques de Injeção**
- **JSX Representa Objetos**

Introdução ao JSX

JSX

É uma extensão de sintaxe para JavaScript. Recomendamos usar JSX com o React para descrever como a UI deveria parecer.

```
const element = <h1>Hello, world!</h1>;
```

Introduzindo JSX

Por que JSX?

O React separa conceitos com unidades pouco acopladas chamadas “componentes”.

- O React não requer o uso do JSX.
- JSX é prático quando se está trabalhando com uma UI dentro do código em JavaScript.
- Ele permite ao React mostrar mensagens mais úteis de erro e aviso.

Introduzindo JSX

Incorporando Expressões em JSX

Você pode inserir qualquer expressão JavaScript válida dentro das chaves em JSX.

```
const nome = 'Augusto Franceschetto';
const element = <h1>Olá, {nome}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Introduzindo JSX

Incorporando Expressões em JSX

```
function formatName(user) {
    return user.firstName + ' ' + user.lastName;
  }

  const user = {
    firstName: 'Augusto',
    lastName: 'Franceschetto',
  };

  const element = <h1>Hello, {formatName(user)}!</h1>;

  ReactDOM.render(element, document.getElementById('root'));
```

Introduzindo JSX

JSX Também é uma Expressão

- **Depois da compilação**, as expressões em JSX se transformam em chamadas normais de funções que **retornam objetos JavaScript**.

- Isto significa que **você pode usar JSX dentro de condições if e laços for,** atribuí-lo a variáveis, aceitá-lo como argumentos e retorná-los de funções:

```
function getGreeting(user) {
    if (user) {
      return <h1>Hello, {formatName(user)}!</h1>;
    }
    return <h1>Hello, Stranger.</h1>;
  }
```

Introduzindo JSX

Especificando Atributos com JSX

```
const element = <div tabIndex="0"></div>;
```

Você também pode usar chaves para incorporar uma expressão JavaScript em um atributo:

```
const element = <img src={user.avatarUrl}></img>;
```

Nota: o React DOM usa camelCase como convenção para nomes de propriedades ao invés dos nomes de atributos do HTML.
Por exemplo: **class** se transforma em **className** em JSX, e **tabindex** se transforma em **tabIndex**.

Introduzindo JSX

Especificando Elementos Filhos com JSX

Se uma tag está vazia, você pode fechá-la imediatamente com />, como XML:

```
const element = <img src={user.avatarUrl} />;
```

Tags JSX podem conter elementos filhos:

```
const element = (
    <div>
      <h1>Hello!</h1>
      <h2>Good to see you here.</h2>
    </div>
  );
```

Introduzindo JSX

JSX Previne Ataques de Injeção

- Por padrão, o React DOM assegura que você nunca injete algo que não esteja explicitamente escrito na sua aplicação.
- Tudo é convertido para string antes de ser renderizado. Isso ajuda a prevenir ataques XSS (cross-site-scripting).

```
const title = response.potentiallyMaliciousInput;
// This is safe:
const element = <h1>{title}</h1>;
```

Introduzindo JSX

JSX Representa Objetos

O Babel compila JSX para chamadas React.createElement().

Estes dois exemplos são idênticos:

```
const element = (
    <h1 className="saudacao">
      Hello, world!
    </h1>
  );
```

```
const element = React.createElement(
    'h1',
    {className: 'saudacao'},
    'Hello, world!'
  );
```

Introduzindo JSX

JSX Representa Objetos

- Estes objetos são chamados “Elementos React”.
- Você pode imaginá-los como descrições do que você quer ver na tela.
- O React lê esses objetos e os usa para construir o DOM e deixá-lo atualizado.

Introduzindo JSX

JSX Representa Objetos

React.createElement() realiza algumas verificações para ajudar você a criar um código sem bugs, mas, essencialmente, cria um objeto como este:

```
// Nota: esta estrutura está simplificada
const element = {
    type: 'h1',
    props: {
      className: 'saudacao',
      children: 'Hello, world!'
    }
  };
```

Renderizando Elementos

- **Renderizando um Elemento no DOM**
- **Atualizando o Elemento Renderizado**
- **O React Somente Atualiza o Necessário**

Renderizando Elementos

Renderizando um Elemento no DOM

Nós o chamamos de nó raiz do DOM porque tudo dentro dele será gerenciado pelo React DOM.

```
const element = <h1>Hello, world</h1>;
```

Nota: Pode-se confundir **elementos** com o conceito mais amplo de "**componentes**".

Renderizando Elementos

Renderizando um Elemento no DOM

Suponhamos que exista um em algum lugar do seu código HTML:

```
<div id="root"></div>
```

Nós o chamamos de nó raiz do DOM porque tudo dentro dele será gerenciado pelo React DOM.

Nota: Aplicações construídas apenas com React geralmente tem apenas um único nó raiz no DOM.

Renderizando Elementos

Renderizando um Elemento no DOM

- Se deseja integrar o React a uma aplicação existente, você pode **ter quantos nós raiz precisar**.

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

Nota: Para renderizar um elemento React em um nó raiz, passe ambos para ReactDOM.render():

Renderizando Elementos

Atualizando o Elemento Renderizado

- **Elementos React são imutáveis.** Uma vez criados, você não pode alterar seus elementos filhos ou atributos.
- Com o que aprendemos até agora, a única forma de atualizar a interface **é criar um novo elemento e passá-lo para ReactDOM.render().**

```
function tick() {
    const element = (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {new Date().toLocaleTimeString()}.</h2>
      </div>
    );
    ReactDOM.render(element, document.getElementById('root'));
  }

  setInterval(tick, 1000);
```

Renderizando Elementos

Atualizando o Elemento Renderizado

Nota: Na prática, a maioria dos aplicativos React usam o ReactDOM.render() apenas uma única vez.

Renderizando Elementos

O React Somente Atualiza o Necessário

O React DOM compara o elemento novo e seus filhos com os anteriores e somente aplica as modificações necessárias no DOM para levá-lo ao estado desejado.

Componentes e Props

- **Renderizando um Elemento no DOM**
- **Atualizando o Elemento Renderizado**
- **O React Somente Atualiza o Necessário**

Componentes e Props

Componentes de Função e Classe

Componentes permitem você dividir a UI em partes **independentes, reutilizáveis** e **pensar em cada parte isoladamente**.

Componentes e Props

Componentes de Função e Classe

Nota: Conceitualmente, componentes são como **funções JavaScript**. Eles aceitam **entradas arbitrárias** (chamadas “props”) e retornam elementos React que descrevem o que deve aparecer na tela.

Componentes e Props

Componentes de Função e Classe

A maneira mais simples de definir um componente é escrever uma função JavaScript:

```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
  }
```

Componentes e Props

Componentes de Função e Classe

Você também pode usar uma classe ES6 para definir um componente:

```
class Welcome extends React.Component {
    render() {
      return <h1>Hello, {this.props.name}</h1>;
    }
  }
```

Componentes e Props

Renderizando um Componente

Anteriormente, nós encontramos apenas elementos React que representam tags do DOM:

```
const element = <div />;
```

No entanto, elementos também podem representar componentes definidos pelo usuário:

```
const element = <Welcome name="Sara" />;
```

Nota: Quando o React vê um elemento representando um componente definido pelo usuário, ele passa atributos JSX e componentes filhos para esse componente como um único objeto. Nós chamamos esse objeto de "**props**".

Componentes e Props

Renderizando um Componente

```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
  }

  const element = <Welcome name="Sara" />;
  ReactDOM.render(
    element,
    document.getElementById('root')
  );
```

Nota: Quando o React vê um elemento representando um componente definido pelo usuário, ele passa atributos JSX e componentes filhos para esse componente como um único objeto. Nós chamamos esse objeto de “**props**”.

Componentes e Props

Renderizando um Componente

Vamos recapitular o que acontece nesse exemplo:

1. Nós chamamos ReactDOM.render() com o elemento <Welcome name="Sara" />.
2. React chama o componente Welcome com {name: 'Sara'} como props.
3. Nosso componente Welcome retorna um elemento <h1>Hello, Sara</h1> como resultado.
4. React DOM atualiza eficientemente o DOM para corresponder <h1>Hello, Sara</h1>.

https://codepen.io/pen?&editors=0010

```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
  }

  const element = <Welcome name="Sara" />;
  ReactDOM.render(
    element,
    document.getElementById('root')
  );
```

Componentes e Props

Renderizando um Componente

Nota: **Sempre inicie os nomes dos componentes com uma letra maiúscula.**

O React trata componentes começando com letras minúsculas como tags do DOM.

Por exemplo, ** ** representa uma tag div do HTML.

No React **<Welcome />** representa um componente e requer que Welcome esteja no escopo.

Componentes e Props

Compondo Componentes

Componentes podem se referir a outros componentes em sua saída.
Isso nos permite usar a mesma abstração de componente para qualquer nível de detalhe.
Um botão, um formulário, uma caixa de diálogo, uma tela:
em aplicativos React, **todos esses são normalmente expressos como componentes.**

Componentes e Props

Compondo Componentes

Nós podemos criar um componente App que renderiza Welcome muitas vezes:

https://codepen.io/pen?&editors=0010

```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
  }

  function App() {
    return (
      <div>
        <Welcome name="Sara" />
        <Welcome name="Cahal" />
        <Welcome name="Edite" />
      </div>
    );
  }

  ReactDOM.render(
    <App />,
    document.getElementById('root')
  );
```

Componentes e Props

Compondo Componentes

Tipicamente, novos aplicativos React tem **um único componente App no topo**.

Componentes e Props

Extraindo Componentes

Não tenha medo de dividir componentes em componentes menores.
Por exemplo, considere esse componente Comment:

```
function Comment(props) {
    return (
      <div className="Comment">
        <div className="UserInfo">
          <img className="Avatar"
            src={props.author.avatarUrl}
            alt={props.author.name}
          />
          <div className="UserInfo-name">
            {props.author.name}
          </div>
        </div>
        <div className="Comment-text">
          {props.text}
        </div>
        <div className="Comment-date">
          {formatDate(props.date)}
        </div>
      </div>
    );
  }
```

Componentes e Props

Extraindo Componentes

- Esse componente pode ser complicado de alterar por causa de todo o aninhamento.
- É difícil reutilizar suas partes individuais.

```
function Comment(props) {
    return (
      <div className="Comment">
        <div className="UserInfo">
          <img className="Avatar"
            src={props.author.avatarUrl}
            alt={props.author.name}
          />
          <div className="UserInfo-name">
            {props.author.name}
          </div>
        </div>
        <div className="Comment-text">
          {props.text}
        </div>
        <div className="Comment-date">
          {formatDate(props.date)}
        </div>
      </div>
    );
  }
```

Componentes e Props

Extraindo Componentes

- Vamos extrair alguns componentes dele.
- Primeiro, nós vamos extrair Avatar:

```
function Avatar(props) {
    return (
      <img className="Avatar"
        src={props.user.avatarUrl}
        alt={props.user.name}
      />
    );
  }
```

- O Avatar não precisa saber que está sendo renderizado dentro do Comment. É por isso que nós demos ao seu **prop** um nome mais genérico: **user** em vez de **author**.

- Nós recomendamos nomear props a partir do ponto de vista do próprio componente ao invés do contexto em que ele está sendo usado.

Componentes e Props

Extraindo Componentes

Agora nós podemos simplificar Comment um pouco mais:

```
function Comment(props) {
    return (
      <div className="Comment">
        <div className="UserInfo">
          <Avatar user={props.author} />
          <div className="UserInfo-name">
            {props.author.name}
          </div>
        </div>
        <div className="Comment-text">
          {props.text}
        </div>
        <div className="Comment-date">
          {formatDate(props.date)}
        </div>
      </div>
    );
  }
```

Componentes e Props

Extraindo Componentes

Em seguida, nós vamos extrair o componente UserInfo que renderiza um Avatar ao lado do nome do usuário:

```
function UserInfo(props) {
    return (
      <div className="UserInfo">
        <Avatar user={props.user} />
        <div className="UserInfo-name">
          {props.user.name}
        </div>
      </div>
    );
  }
```

Componentes e Props

Extraindo Componentes

Isso nos permite simplificar Comment ainda mais:

```
function Comment(props) {
    return (
      <div className="Comment">
        <UserInfo user={props.author} />
        <div className="Comment-text">
          {props.text}
        </div>
        <div className="Comment-date">
          {formatDate(props.date)}
        </div>
      </div>
    );
  }
```

Componentes e Props

Extraindo Componentes

Extrair componentes pode parecer um trabalho pesado no começo, mas ter uma paleta de componentes reutilizáveis compensa em aplicativos maiores.

Uma boa regra é que se uma parte da sua UI for usada várias vezes (Button, Panel, Avatar) ou for complexa o suficiente por si só (App, FeedStory, Comment) é uma boa candidata a ser extraída para um componente separado.

Componentes e Props

Props são Somente Leitura

Independente se você declarar um componente como uma função ou uma classe, **ele nunca deve modificar seus próprios props.**

```
function sum(a, b) {
    return a + b;
  }
```

Tais funções são chamadas **“puras”.**
Porque elas **não tentam alterar suas entradas** e **sempre retornam o mesmo resultado** para as mesmas entradas.

Componentes e Props

Props são Somente Leitura

Em contraste, essa função é impura porque altera sua própria entrada:

```
function withdraw(account, amount) {
    account.total -= amount;
  }
```

Componentes e Props

Props são Somente Leitura

Nota: Todos os componentes React tem que agir como funções puras em relação ao seus props.