Seminário de Desenvolvimento Web com ReactJS

Allan Nathan Baron de Souza

Israel Efraim de Oliveira

José Carlos Zancanaro

VaultBox

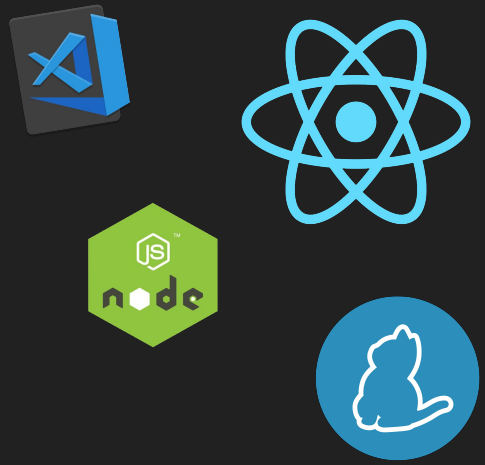
UNIVALI

Allan Nathan Baron de Souza

Israel Efraim de Oliveira

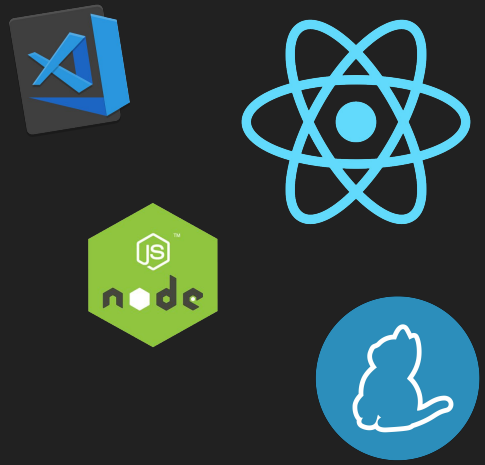
José Carlos Zancanaro

Tópicos



1. **O que é React?**
2. **Instalação**
    a. VS Code
    b. Node.js
    c. Yarn
3. **Inicialização de um projeto React**
4. **Hello World em React**
    a. Projeto simples sem uso de JSX
    b. Criação de componentes
    c. Renderização de componentes

Tópicos



5. **Introdução ao JSX**
6. **Componentes e Propriedades (props)**
    a. Function Component
    b. Class Component
    c. Composição de componentes
    d. Extração de componentes
7. **Estado e Ciclo de vida**
    a. State
    b. Métodos de ciclo de vida

Introdução

O que é React ?

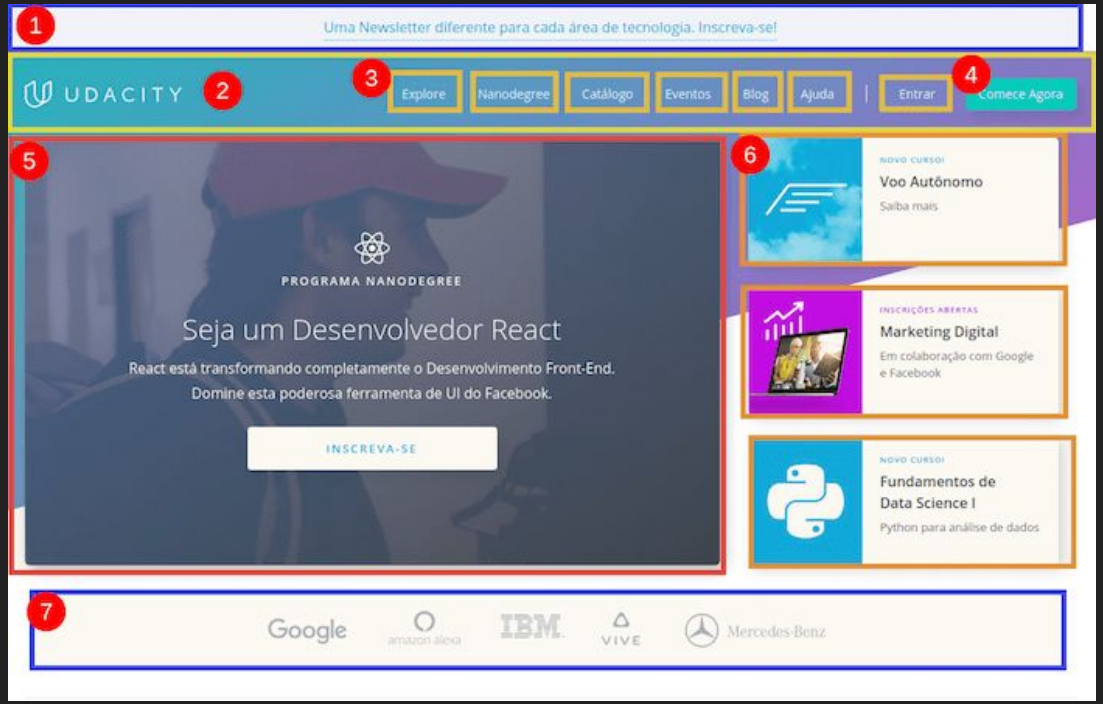
É uma biblioteca Javascript para criação de interfaces de usuário.

Declarativo: **Permite a criação de UI interativas de maneira simples, permitindo a previsibilidade de comportamento e simplicidade na depuração.**

Componentizado: **Baseado em componentes que gerenciam seu próprio estado e facilitam a construção de aplicações complexas e possibilitam que o estado seja mantido fora do DOM*.**

* Document Object Model

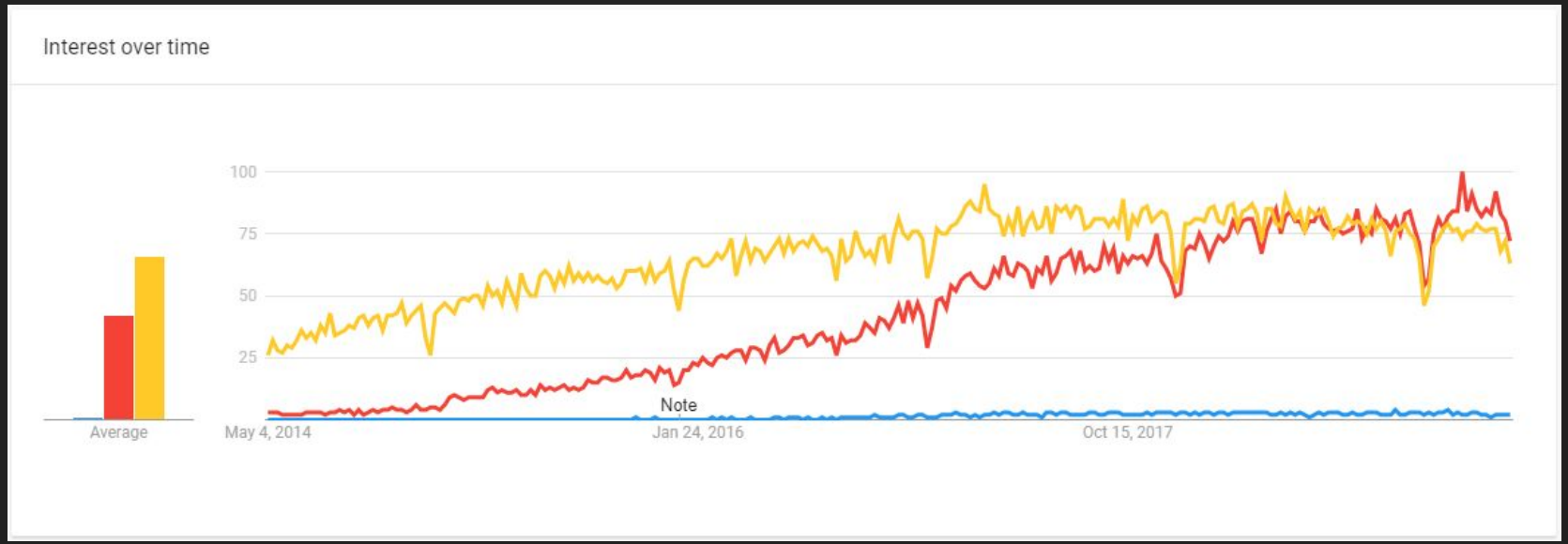
Ideia de componentização



UNIVALI

Contextualização

Interesse em bibliotecas JavaScript



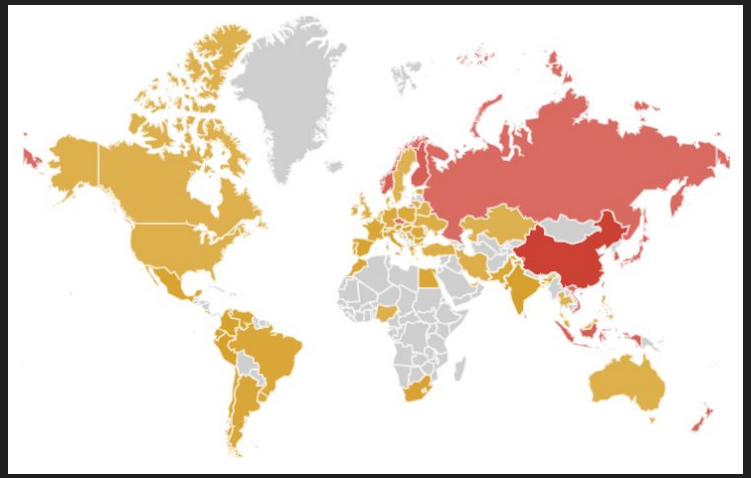
Vue React Angular

UNIVALI

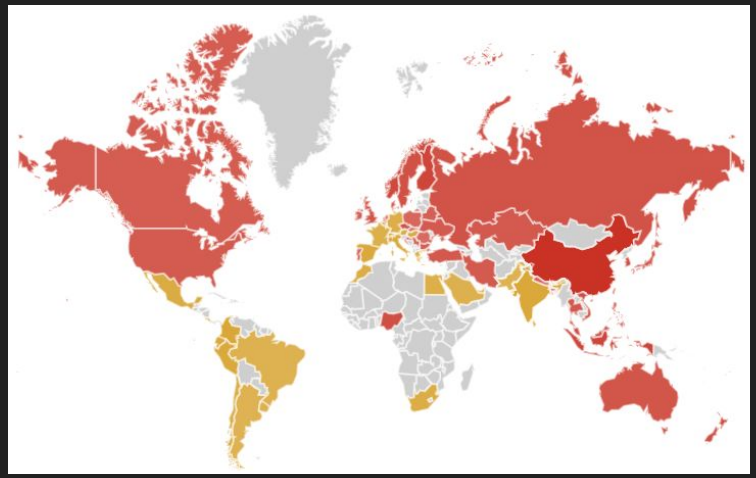
Interesse em bibliotecas JavaScript por países

- Vue
- React
- Angular

Há 5 anos



Hoje



Interesse em bibliotecas JavaScript no Brasil

Instalação

Instalação

Visual Studio Code disponível em *code.visualstudio.com*

NodeJS disponível em *nodejs.org/en/download*

Yarn disponível em *yarnpkg.com/pt-BR/docs/install*

Desenvolvimento

UNIVALI

Iniciando um projeto React

O método mais eficiente de criar um projeto React é através do *terminal* (shell).

Através do shell, navegue até uma pasta desejada e crie um projeto React:

```
$ yarn create react-app nome        // criar app React
$ cd nome                           // entrar na pasta criada
```

Onde:

yarn é o gerenciador de pacotes
create é a operação desejada
react-app é o tipo de aplicativo que o yarn deve criar
nome é um substantivo qualquer em letras minúsculas e sem espaços

Executando um projeto React

Para iniciar o aplicativo em modo de desenvolvimento, execute:

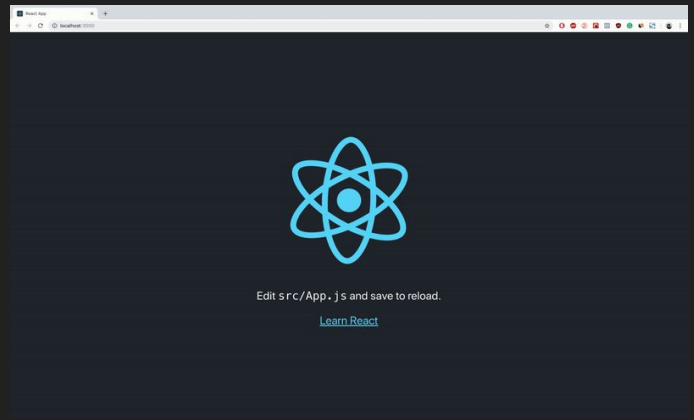
```
$ yarn start
```

Saída:

```
You can now view project in the browser.

  Local: http://localhost:3000/

Note that the development build is not optimized.
To create a production build, use yarn build.
```



http://localhost:3000/

Hello World em React

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';

const raiz = React.createElement('h1', null, 'Olá mundo!');

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

Questão

Dada a linha abaixo e a saída no navegador após a execução do projeto, o que é possível concluir sobre os parâmetros do método createElement?

```
const raiz = React.createElement('h1', null, 'Olá mundo!');
```

Questão

Dada a linha abaixo e a saída no navegador após a execução do projeto, o que é possível concluir sobre os parâmetros do método createElement?

```
const raiz = React.createElement('h1', null, 'Olá mundo!');
```

Resposta: O primeiro parâmetro, *h1*, é o tipo de elemento a ser criado; o segundo, *null*, é o objeto de propriedades e o terceiro, “Olá mundo!”, é o conteúdo do elemento (ou filhos).

Criação de Elementos

```
<h1>Olá mundo!</h1>
```

Para criarmos o elemento HTML acima, realizamos a seguinte chamada:

```
const raiz = React.createElement('h1', null, 'Olá mundo!');
```

Onde:

- O primeiro parâmetro é o tipo de elemento (ou tag) a ser criado;
- O segundo parâmetro é o objeto de *propriedades* do elemento; e
- O terceiro são os *filhos* do elemento, ou *conteúdo*.

Criação de Elementos

Desta forma, o elemento HTML abaixo:

```
<h1 id="title">Olá mundo!</h1>
```

Ficaria desse jeito:

```
const raiz = React.createElement('h1', {id: 'title'}, 'Olá mundo!');
```

Renderização de Componentes

```
ReactDOM.render(raiz, elementoDom);
```

ReactDOM é o DOM (Document Object Model) virtual do React, utilizado para renderizar a árvore de componentes React no DOM *físico* da página.

A raiz é o componente base para ser renderizado (um h1 neste exemplo) e o elementoDom é o elemento HTML onde o React deve renderizar a raiz.

Em termos técnicos, o React realiza a conciliação do que está no componente raiz e faz com que o conteúdo do elementoDom fique igual a ele.

Renderização de Componentes

public/index.html

```
<html lang="en">
  ...
 <body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div> <!-- O ReactDOM atualizará este elemento com o conteúdo do componente
                             passado no primeiro parâmetro -->
 </body>
</html>
```

UNIVALI

Renderização de Componentes

public/index.html

```
<html lang="en">
  ...
 <body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root">
    <h1>Olá mundo!</h1>
  </div>
 </body>
</html>
```

UNIVALI

JSX

Introdução ao JSX

A forma tradicional de criarmos elementos no React é o método “createElement”

```
React.createElement(tipoElemento, objetoPropriedades, filhos)
```

Entretanto, há uma forma muito mais intuitiva de fazer isto, com o JSX:

```
<Componente propriedade=valor>
  filhos
</Componente>
```

Introdução ao JSX

E se houver mais de uma propriedade?

```
<Componente propriedade=valor propriedade2=valor2>
  filhos
</Componente>
```

Adiciona-se mais uma tupla de propriedade e valor

Introdução ao JSX

O JSX (Javascript XML) é uma extensão de sintaxe para o Javascript, que possibilita criar componentes com uma sintaxe semelhante ao HTML/XML.

Portanto:

HTML	`<h1>Olá mundo!</h1>`
Sem JSX	`const raiz = React.createElement('h1', null, 'Olá mundo!');`
JSX	`const raiz = <h1>Hello World!</h1>;`

Introdução ao JSX

Algumas considerações:

- Trabalha com o padrão camelCase ao invés do lowercase do HTML.
- Aceita inserção de código Javascript como expressões dentro de chaves.
- Componentes do HTML iniciam com letra minúscula e Componentes definidos por usuário devem iniciar com letra maiúscula.

Exemplo:

```
const botao = <button id="lg-botao" onAction={logar}>Logar</button>
```

Introdução ao JSX

Com expressões (envolvidas de chaves), o JSX resolve o código Javascript e utiliza o valor retornado.

Analise os seguintes elementos, consegue imaginar a diferença?

```
const titulo = <h1 id="soma">5 + 1</h1>
```

```
const titulo = <h1 id="soma">{5 + 1}</h1>
```

Introdução ao JSX

Um botão com id “welcome” que invoque a função mostrarMsg ao receber um clique.

HTML	`<button id="welcome" onClick="mostrarMsg()">Bem vindo</button>`
JSX	`<button id="welcome" onClick={mostrarMsg}>Bem vindo</button>`

Questão

Qual dos códigos abaixo é a sintaxe correta para um manipulador de evento de clique de um botão com JSX?

```
<button onclick={foo()}/>
```
1

```
<button onclick={foo}/>
```
2

```
<button onClick={foo()}/>
```
3

```
<button onClick={foo}/>
```
4

Questão

Qual dos códigos abaixo é a sintaxe correta para um manipulador de evento de clique de um botão com JSX?

```
<button onclick={foo()}/>
```
1

```
<button onclick={foo}/>
```
2

```
<button onClick={foo()}/>
```
3

```
<button onClick={foo}/>
```
4

Componentes e propriedades

Componentes e Propriedades

Um componente no React é uma parte independente e reutilizável de um código de UI.

Todo componente recebe *propriedades* (mesmo que sejam nulas) no momento em que é instanciado (criado).

De acordo com a documentação do React (em *reactjs.org*), “componentes permitem que você quebre a interface em pedaços independentes e reutilizáveis, e permitem vários níveis de isolamento destes componentes”.

Componentes e Propriedades

Existem **dois** meios para construção de componentes personalizados (definidos pelo programador) no React:

- Componentes de Função, ou **Function Components**
- Componentes de Classe, ou **Class Components**

A escolha de um meio geralmente está ligada ao tipo de componente que ele é, mais especificamente, se ele *tem ou não* um *estado* próprio.

Componentes com estado são chamados de *stateful*, enquanto que componentes sem estado são chamados de *stateless*.

Function Components

Componentes de Função são criados a partir de **funções comuns** do Javascript e são escolhidos geralmente quando o componente é **stateless**.

Eles recebem um parâmetro com as propriedades e *retornam* um elemento React.

```
function OlaMundo(props) {
  return <h1>Olá Mundo</h1>
}
```

Function Components

Exemplo:

```
// Definição do function component
function Saudacao(props) {
 return <h1>Olá {props.nome}</h1>
}

// Instanciação
const saudacao = <Saudacao nome="Javascript" />
```

Function Components

Exemplo com mais de uma propriedade:

```
// Definição do function component
function Saudacao(props) {
 return <h1>Olá {props.nome} {props.sobrenome}</h1>
}

// Instanciação
const saudacao = <Saudacao nome="Javascript" sobrenome="ES6"/>
```

Function Components

Um componente que recebe um vetor e o imprime em uma unordered list (ul).

```
class ListaDeBalas extends React.Component {
 render() {
   return (
     <ul>
       { this.props.itens.map(item => <li>{item}</li>) }
     </ul>
   )
 }
}
const listaBalas = <ListaDeBalas itens={['frango', 'carne', 'jujuba']}/>
```

Questão

O que o componente abaixo exibe (qual o seu propósito)?

```
function Texto(props) {
  const texto = props.texto
  return (
    <div>
      {texto.map(paragrafo => <p>{paragrafo}</p>)}
    </div>
  )
}
```

Questão

O que o componente abaixo exibe?

Resposta: O componente é responsável por renderizar uma lista de parágrafos recebido pelo parâmetro de propriedades / props.

Class Components

Componentes de Classe são estruturados a partir de uma **classe** no Javascript e são utilizados quando apresentam um estado interno, isto é, são **stateful**.*

Devem especializar a classe React.Component e implementar o método ***render()***, retornando um elemento React.

```
class OlaMundo extends React.Component {
  render() {
    return <h1>Olá Mundo!</h1>
  }
}
```

* State é apresentado na próxima seção

Class Components

Exemplo:

```
class Saudacao extends React.Component {
 render() {
   return (
     <h1>Olá {this.props.nome}</h1>
   )
 }
}

const saudacao = <Saudacao nome="Javascript"/>
```

Class Components

Exemplo com mais de uma propriedade:

```
class Saudacao extends React.Component {
 render() {
   return (
     <h1>Olá {this.props.nome} {this.props.sobrenome}</h1>
   )
 }
}

const saudacao = <Saudacao nome="Javascript" sobrenome="ES6"/>
```

Class Components

Um componente que recebe um vetor e o imprime em uma unordered list (ul).

```
class ListaDeBalas extends React.Component {
 render() {
   return (
     <ul>
       { this.props.itens.map(item => <li>{item}</li>) }
     </ul>
   )
 }
}
const listaBalas = <ListaDeBalas itens={['frango', 'carne', 'jujuba']}/>
```

Questão

Todo componente recebe propriedades no momento em que é instanciado.

Se os Function Components recebem suas propriedades por *parâmetro de função*, qual seria o meio pelo qual uma classe receberia suas propriedades?

Questão

Todo componente recebe propriedades no momento em que é instanciado.

Se os Function Components recebem suas propriedades por *parâmetro de função*, qual seria o meio pelo qual uma classe receberia suas propriedades?

Resposta: As propriedades são passadas para o **construtor da classe**.

Class Components

Os componentes de classe recebem suas propriedades pelo construtor padrão, que é criado automaticamente quando não mencionado.

```
class OlaMundo extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <h1>Olá Mundo!</h1>
  }
}
```

Questão

O que é usado para passar dados para um componente?

Questão

O que é usado para passar dados para um componente?

Resposta: Propriedades / props.

Composição de Componentes

Assim como no HTML, componentes podem se referir a outros componentes, sem um limite específico.

Isto permite o uso de componentes como abstração para qualquer nível de detalhe, como uma tela, uma caixa de diálogo, um formulário e etc.

Em aplicativos single-page, é comum que haja apenas um componente no topo, geralmente denominado App, contendo de forma organizada todos os outros componentes da aplicação.

Extração de Componentes (simplificação)

O objetivo da componentização é reduzir a redundância de código, possibilitando que um componente seja reutilizado em diversas áreas do aplicativo. A consequência de um projeto bem organizado são componentes bem isolados com fácil manutenção.

Da documentação do React (em *reactjs.org*):

“Uma boa regra é que se uma parte da UI é utilizada em diversos locais ou for complexa o suficiente por si só, é uma candidata a se tornar um componente reutilizável.”

Extração de Componentes (simplificação)

Considere o componente do próximo slide, e verifique como ele é simplificado.

Extração de Componentes (simplificação)

```
function Comentario(props) {
 return (
   <div>
     <div>
       <img src={props.autor.avatarUrl} alt={props.autor.nome} />
       <p> {props.autor.nome} </p>
     </div>
     <p> {props.texto} </p>
     <p> {props.data.toLocaleDateString()} </p>
   </div>
 )
}
```

Extração de Componentes (simplificação)

O usuário está compondo o componente Comentário, poderíamos torná-lo um componente independente e possibilitar a sua reutilização:

```
function Usuario(props) {
 return (
   <div>
      <img src={props.autor.avatarUrl} alt={props.autor.nome} />
      <p> {props.autor.nome} </p>
    </div>
 )
}
```

State

O state pode ser atualizado através da chamada ao método **setState**() de um Class Component, passando um objeto com as novas propriedades.

As propriedades recebidas pelo método são atualizadas no state e, aquelas que estavam no state e não forem mencionadas, permanecerão com o valor inalterado.

Após a chamada ao método **setState**, o React automaticamente chama o método **render** da chasse.

Extração de Componentes (simplificação)

Desta forma:

```
function Comentario(props) {
 return (
   <div>
     <Usuario autor={props.autor}/>
     <p> {props.texto} </p>
     <p> {props.data.toLocaleDateString()} </p>
   </div>
 );
}
```

Estado e Ciclo de vida

State

É comum um componente necessitar armazenar um dado que não lhe é passado por propriedade, um dado que este mesmo gera.

```
class Comentario extends React.Component {
   constructor(props) {
      super(props);
      this.state = {comentario: 'Exemplo'};
   }
   render() {
      return <h1>{this.state.comentario}</h1>
   }
}
```

State

Existem duas versões do método setState.

Recebe um objeto com os novos valores.

```
this.setState({
        propriedade: novoValor
})
```

Recebe uma função que recebe o state antigo e os props e define novos valores.

```
this.setState((state, props) => {
        propriedade: novoValor
})
```

Questão

O que aconteceria se você chamasse o método setState() dentro do método render()?

Questão

O que aconteceria se você chamasse o método setState() dentro do método render()?

Resposta: O código entraria em loop infinito na maioria dos casos.

Métodos de ciclo de vida

Um componente pode ser informado sobre eventos de *ciclo de vida*, isto é, sobre as circunstâncias que envolvem seu ambiente.

Os métodos mais utilizados são os seguintes:

```
componentDidMount() {
    // foi carregado na UI e renderizado pela primeira vez
}
componentWillUnmount() {
    // não será mais renderizado
}
```

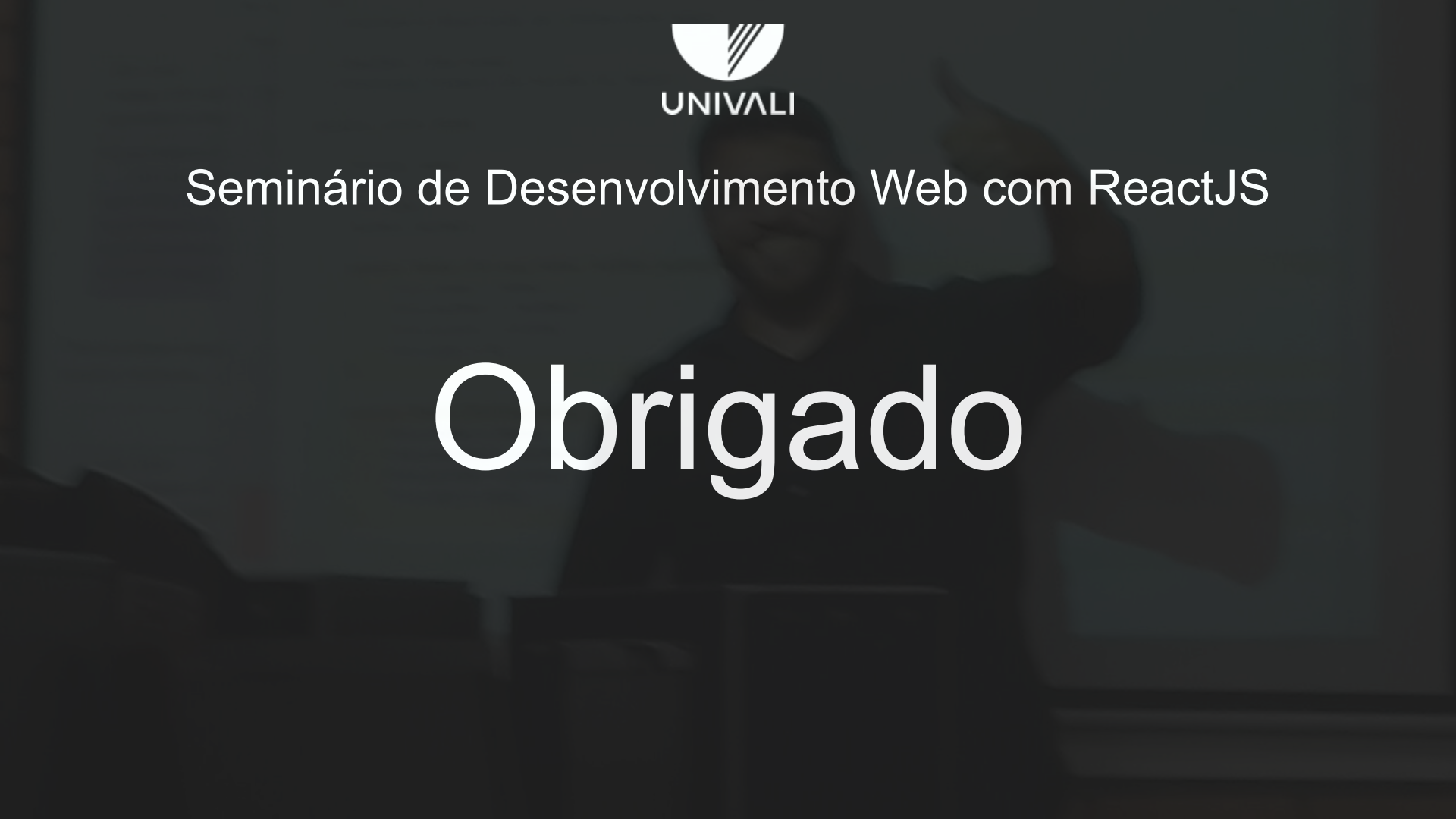
Questão

Qual método de um Componente React é chamado após a renderização do componente pela primeira vez?

Questão

Qual método de um Componente React é chamado após a renderização do componente pela primeira vez?

Resposta: componentDidMount é o método chamado após a primeira renderização de um componente.



Seminário de Desenvolvimento Web com ReactJS

Obrigado