53 45 52 45 49 20 46 49 45 4c 20
41 4f 53 20 50 52 45 43 45 49 54
4f 53 20 44 41 20 48 4f 4e 52 41
20 45 20 44 41 20 43 49 c3 8a 4e
43 49 41 2c 20 50 52 4f 4d 4f 56
45 4e 44 4f 20 4f 20 55 53 4f 20
45 20 4f 20 44 45 53 45 4e 56 4f
4c 56 49 4d 45 4e 54 4f 20 44 41
20 49 4e 46 4f 52 4d c3 81 54 49
43 41 20 45 4d 20 42 45 4e 45 46
c3 8d 43 49 4f 20 44 4f 20 43 49
44 41 44 c3 83 4f 20 45 20 44 41
20 53 4f 43 49 45 44 41 44 45 2e

RESIDÊNCIA DE

SOFTWARE

CAPACITAR
TREINAR
EMPREGAR

TRANSFORMAR

Firjan SENAI

serratec
Parque Tecnológico Região Serrana

Introdução React

Prof. Diego Leite
diego@domob.me

O que é React?

- Biblioteca para construção de interfaces;
- Utilizado para a construção de Single-Page-Applications (SPA);
- Tudo fica dentro do JavaScript;
- React vs ReactJS vs React Native

Tudo fica dentro do javascript - Exemplo

```
import React from ‘react’;

import ‘./button.css’;
import icon from ‘./button.png’;

function Button() {
      return (
            <button>
                  <img src={icon} />
            </button>
      );
}
```

Vantagens

- Organização do código;
  - Componentização;
- Divisão de responsabilidades;
  - Backend: Regra de Negócio;
  - Frontend: Interface;
- Uma API, múltiplos clientes;
  - Backend (API) serve tanto para web quanto para mobile;

JSX

- Escrever HTML dentro do JavaScript;
- Com React podemos criar nossos próprios elementos;

Sem utilizar JSX

```
function Button() {
    return React.createElement(
        'button',
        { type: 'button' },
        React.createElement(
            'span',
            { class: 'icon' }
        )
    );
}

<button type='button'>
    <span class='icon'></span>
</button>
```

Utilizando JSX

```
function Button() {
    return (
        <button type='button' >
            <span class='icon'></span>
        </button>
    );
}

function Header() {
    return <Button />
}
```

Babel / Webpack

- O browser não entende todo esse código;
- O Babel converte o código JS de uma forma que o browser entenda;
- O Webpack possui várias funções:
  - Criação do bundle, arquivo com todo o código da aplicação;
  - Ensinar ao JavaScript como importar arquivos CSS, imagens e etc;
  - Live reload com Webpack Dev Server;

Ciclo de Vida

constructor

O método constructor é a primeira função executada no componente.

Sempre que definirmos o constructor, precisamos repassar as props recebidas para o componente pai Component.

```
class App extends Component {
    constructor(props) {
        super(props);
    }
}
```

Ciclo de Vida

componentWillMount

Método executado antes do render e pode inclusive realizar alterações no estado.

Ciclo de Vida

componentDidMount

Método chamado após o render indica que a renderização inicial do nosso componente foi finalizada, é o local recomendado para fazer qualquer processo assíncrono ou de efeito colateral como chamadas à API, referenciar componentes criados no render ou inclusive alterar o estado, disparando uma nova atualização no fluxo do componente.

Ciclo de Vida

componentWillReceiveProps

Método executado automaticamente toda vez que alguma propriedade do componente for atualizada.

Ciclo de Vida

shouldComponentUpdate

Método responsável por determinar se o componente deve realizar o render novamente ou não. Lembrando que qualquer alteração de propriedade ou estado do componente faz com que ele gere uma nova renderização

Ciclo de Vida

componentWillUpdate

Se o método shouldComponentUpdate retornar true, o componentWillUpdate realiza a intermediação entre o render. Esse método também recebe as novas propriedades e estado. Após esse método, o render é disparado novamente com as alterações

Ciclo de Vida

componentDidUpdate

Este método é executado após o novo render indicando que o componente foi atualizado com sucesso. Recebe as propriedades e estado antigos como parâmetro.

Ciclo de Vida

componentWillUnmount

Este método chamado antes de um componente ser desmontado, ótimo para cancelar EventListeners ou setIntervals que ainda possam estar sendo executados.

Ciclos de Vida

```
import React, {Component} from ’react’;

class App extends Component {

        componentWillMount () {};

        componentDidMount () {};

        componentWillUnmount () {};

        componentWillReceiveProps (props) {};

        render( ) {
                return (
                        <div className=”container”>
                                <p>Sou um texto</p>
                        </div>
                )
        };
}
```

Componentização

Componentes são conjuntos isolados de:

- lógica
- visualização (JSX)
- estilização

É dividido com o objetivo de encapsular a lógica e estilização do mesmo, assim evitando o compartilhamento desnecessário de código entre outros componentes.

Propriedades (Props)

Props, que é uma abreviação de properties, ou propriedades, são informações que podem ser passadas para um componente. Pode ser uma string, um número, até mesmo uma função.

Este valor pode então ser utilizado pelo componente que a recebe. Primeiro, passamos o dado que desejamos passar ao componente definindo-o como um atributo ou elementos filhos, onde o componente será utilizado.

Propriedades (Props)

Tarefa.js

```
import React, {Component} from ‘react’;

export default class Tarefa extends Component {
        render(){
                return (
                        <div>
                                <p>{this.props.title}</p>
                        </div>
                )
        }
}
```

index.js

```
import React, {Component} from ‘react’;
import Tarefa from ‘./Tarefa.js’;

export default class App extends Component {
        render(){
                return (
                        <div>
                                <Tarefa title=’Configurar Ambiente’ />
                                <Tarefa title=’Baixar Material’ />
                                <Tarefa title=’Colocar Playlist’ />
                        </div>
                )
        }
}
```

Estado (state)

O Estado, State, assim como as props, são dados utilizados pelo componente. Novamente, pode ser uma string, um objeto, um array, um número. A diferença, no caso do state, é que ao invés de receber a informação e somente utilizá-la, o state é privado e completamente controlado pelo componente.

O state é imutável a fim de tornar o processo performático, ou seja não pode sofrer alterações, sejam elas feitas pelo usuário ou derivadas de ações terceiras criadas pela aplicação, quando um novo valor é retornado, ele sobrescreve o objeto inicial tornando todo processo mais rápido.

Estado (state)

Tarefa.js

```
import React, {Component} from ‘react’;

export default class Tarefa extends Component {
        this.setState = {
                status: ‘em aberto’
        };

        render(){
                return (
                        <div>
                                <p>{this.props.title}</p>

                                <button type=’button’ onClick={() => this.setState({ status: ‘finalizada’ })}>
                                        Finalizar Tarefa
                                </button>
                        </div>
                )
        }
}
```

Introdução aos Hooks

```
import React, { useState } from 'react';

function Example() {
  // Declare uma nova variável de state, a qual chamaremos de "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Introdução aos Hooks

Hooks são funções que permitem a você "ligar-se" aos recursos de state e ciclo de vida do React a partir de componentes funcionais. Hooks não funcionam dentro de classes — eles permitem que você use React sem classes.

```
function Example() {
    return (
        <div>
            <p>Hello World</p>
        </div>
    );
}
```

Introdução aos Hooks

Hooks são uma nova adição ao React 16.8. Eles permitem que você use o state e outros recursos do React sem escrever uma classe.

```
import React, { useState } from 'react';

function Example() {
    const [count, setCount] = useState(0);

    return (
        <div>
            <p>You clicked {count} times</p>

            <button onClick={() => setCount(count + 1)}>
                Click me
            </button>
        </div>
    );
}
```

Introdução aos Hooks

Sem Quebras de Compatibilidade

Antes de continuar, note que Hooks são:

- Completamente opcionais. Você pode experimentar Hooks em alguns componentes sem reescrever nenhum código existente. Mas você não tem que aprender ou usar Hooks agora se não quiser.
- 100% retrocompatíveis. Hooks não possuem nenhuma quebra de compatibilidade.
- Disponível agora. Hooks estão disponíveis no release v16.8.0.
- Não existem planos de remover classes do React.

Hooks não substituem seu conhecimento dos conceitos de React. Ao invés disso, Hooks provêem uma API mais direta para os conceitos de React que você já conhece: props, state, context, refs e ciclo de vida. Como iremos mostrar em breve, Hooks também oferecem uma poderosa nova forma de combiná-los.

Introdução aos Hooks

Declarando múltiplas variáveis de state

Você pode utilizar o State Hook mais de uma vez em um único componente

```
function ExampleWithManyStates() {
    // Declarando várias variáveis de state!
    const [age, setAge] = useState(42);
    const [fruit, setFruit] = useState('banana');
    const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
    // ...
}
```

Introdução aos Hooks

Hook de Efeito

O Hook de Efeito, useEffect, adiciona a funcionalidade de executar efeitos colaterais através de um componente funcional. Segue a mesma finalidade do componentDidMount, componentDidUpdate, e componentWillUnmount em classes React, mas unificado em uma mesma API.

```
import React, { useState, useEffect } from 'react';

function Example() {
	const [count, setCount] = useState(0);

	useEffect(() => {
		// Atualiza o título do documento utilizando a API do navegador
		document.title = `You clicked ${count} times`;
	},[count]);

	return (
		<div>
			<p>You clicked {count} times</p>
			<button onClick={() => setCount(count + 1)}>
				Click me
			</button>
		</div>
	 );
}
```

Introdução aos Hooks

Hook de Efeito

```
function FriendStatus(props) {
    const [isOnline, setIsOnline] = useState(null);

    function handleStatusChange(status) {
        setIsOnline(status.isOnline);
    }

    useEffect(() => {
        ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);

        return () => {
            ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
        };
    },[ ]);

    if (isOnline === null) {
        return 'Loading...';
    }

    return isOnline ? 'Online' : 'Offline';
}
```

Dúvidas?

Exercício

Criar tela de cadastro de funcionários com as seguintes características:

- Criação de funcionário;
- Edição de funcionário;
- Listar todos os funcionários;
- Acessar dados de um funcionário em específico;
- Remover funcionário;

API: http://residencia-ecommerce.us-east-1.elasticbeanstalk.com/swagger-ui.html

Fontes

https://blog.rocketseat.com.br/react-do-zero-ciclo-de-vida-stateless-components-e-arquitetura-flux/

https://www.codevoila.com/post/57/reactjs-tutorial-react-component-lifecycle

https://medium.com/tableless/o-que-voce-deve-saber-sobre-o-funcionamento-do-rea ct-na tive-7e3c610aa268

http://felipegalvao.com.br/blog/2018/09/24/aprenda-react-componentes-state-e-props/

https://pt-br.reactjs.org/docs/hooks-overview.html