**REACT JS - AULAS RALF LIMA**

**01 - Inicialização do REACT:**

Para começar a utilizar o react, o primeiro passo é criar uma pasta onde será realizado o projeto e utilizar o seguinte comando do terminal do VS CODE:

*npx create-react-app nome\_do\_projeto*

No lugar do nome do projeto, podemos utilizar o ‘ . ’ para que o projeto seja criado diretamente na raiz da pasta, ao invés de criar uma nova para adicionar o react.

**02 – Render**

O render, que fica normalmente no file index.js, tem a função de renderizar a estrutura e enviar para o arquivo index.html o que queremos exibir no site. Para criar um arquivo simples, segue um exemplo:

Primeiro importamos os modulos necessários:

// Importar modulos

import React from 'react';

import { createRoot } from 'react-dom/client';

Depois, por exemplo, teremos uma estrurua que vamos querer renderizar:

// Estrutura

const estrutura =

    <div>

        <h1>Meu primeiro render</h1>

        <p>Testando...</p>

    </div>

Por último, vamos criar a estrutura do render e com o root.render, enviar a estrurura que criamos e pegando o elemento com ID root:

// Estrutura render

const rootElement = document.getElementById('root');

const root = createRoot(rootElement);

root.render(

    estrutura,

    document.getElementById('root')

);

**03 - JSX**

O JSX é uma extensão de arquivo do React que facilita a escrita do código e a utilização do HTML junto com o JS. Segue um exemplo em prática da diferença na aplicação do render:

// Sem JSX

const elemento = *React*.createElement('h1', {}, 'Sem JSX');

root.render(elemento, document.getElementById('root'));

// Com JSX

root.render(<h1>Com JSX</h1>, document.getElementById('root'));

Note que neste exemplo, a diferença está em que utilizando o padrão JSX, não preciamos utilizar a função createElement do JS.

**04 - Componentes**

Os componentes no React são classes que criamos utilizando o React.Component, onde ele vai possuir um método render que retorna o que iremos renderizar. Segue um exemplo:

// Componente

class MeuComponente extends *React*.Component {

    render(){

        return <h1>Meu primeiro componente</h1>

    }

}

\*note que é importante dar o extends no React.Component

Após isso, renderizamos o componente no root.render, da seguinte forma:

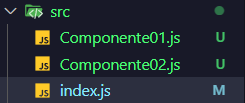
// Render

root.render(<*MeuComponente* />, document.getElementById('root'));

\*para chamar o componente, utilizamos <Componente />

**06 - Criando componentes em arquivos separados e importando**

Em projetos reais, não iremos criar nossos componentes no index.js, mas sim cada componente em um arquivo novo:



Por padrão, no nome do arquivo inserimos o nome da classe do componente que estamos criando (primeira letra em maiúsculo)

No arquivo, vamos realizar a criação do componente:

// Importar o React

import React from 'react';

// Criar componente

class Componente01 extends *React*.Component {

    render() {

        return <h1>Esse é o meu primeiro componente</h1>

    }

}

E no fim vamos exporta-lo:

// Exportar

export default Componente01;

Feito isso, partimos para o arquivo index.js. Nele, vamos importar os componentes que criamos:

// Importar componentes

import Componente01 from './Componente01';

import Componente02 from './Componente02';

E criar um novo componente, englobando os componentes que criamos:

// Componente do arquivo index.js

class Principal extends *React*.Component {

    render(){

        return(

            <div>

                <*Componente01* />

                <*Componente02* />

            </div>

        );

    }

}

Por fim, só renderizamos da forma que já aprendemos o componente criado no arquivo index.js:

// Render

root.render(<*Principal* />, document.getElementById('root'));

**07 - Props**

Props ou propriedades, são características que um componente pode ter acesso. Um props pode ser enviado para um componente como: variável, constante, função, etc.

Formas de enviar um props ao componente:

**-Pré-definida:**

Os props pré-definidos tem seus valores definidos de forma Hard Coded no root.render:

root.render(<*Componente* *nome*="Ralf" />, document.getElementById('root'));

\*neste exemplo, nome é um props

Para exibir esse props, iremos utilizar as chaves { } dentro de componentes. Segue um exemplo:

// Componente

class Componente extends *React*.Component {

    render(){

        return <h1>Olá {*this*.props.nome}</h1>

    }

}

\*utilizamos this para se referir ao objeto, props para chamar os props e então colocamos o nome que foi definido

- Por variável:

Já por variável, conseguimos definir que um props vai receber uma variável:

// JSON

const dados = {nome:"Ralf", idade: 31}

\*criando variável com informações

root.render(<*Componente* *informacoes*={dados} />, document.getElementById('root'));

\*enviando essa variável com os dados para o componente

No return do componente, vamos exibir o props que enviamos, puxando os dados:

return(

<div>

         <p>Olá {*this*.props.informacoes.nome}</p>

          <p>Sua idade é {*this*.props.informacoes.idade}</p>

      </div>

)

**08 - States**

O state do component é similar ao props, mas é privado e por ser alterado, diferentemente do props. States funciona como uma varíavel, sempre que o seu conteúdo for alterado, os elementos que utilizam aquele state sofrem alteração.

// Componente

class Componente extends *React*.Component {

    //Construtor

    constructor(*props*) {

*super*(*props*);

*this*.state = {nome:'Ralf'};

    }

    // Render

    render() {

        return <h1>{*this*.state.nome}</h1>;

    }

}

O state é declarado no componente dentro de um construtor.

09- onClick

No React.js utilizamos a função onClick no render para capturar o evento de clique do mouse e realizar alguma ação. Segue um exemplo onde é chamada uma função de exibir a mensagem ‘Hello World’ na tela:

// Componente

class MeuComponente extends *React*.Component {

    // Função

    mensagem() {

        alert('Hello world');

    }

    // Render

    render() {

        return <button *onClick*={*this*.mensagem}>Clique aqui</button>

    }

}

10 - Bind

O bind é uma das maneiras de enviarmos variáveis para as funções dos componentes. No exemplo, dentro da função onClick, onde chamamos a função mensagem, colocamos um .bind( ), enviando o this e os parametros que desejamos:

// Componente

class MeuComponente extends *React*.Component {

    // Função

    mensagem(*nome*) {

        alert('Olá ' + *nome*);

        console.log(*this*);

    }

    // Render

    render() {

        return <button *onClick*={*this*.mensagem.bind(*this*, 'Guilherme')}>Clique aqui</button>

    }

}

09 - Arrow Function

Outra forma de enviar as váriaveis para as funções dos componentes é utilizando as arrow functions. Com elas, não precisamos enviar o this e nem utilizar o bind:

class MeuComponente extends *React*.Component {

    mensagem(*nome*) {

        alert('Olá ' + *nome*);

        console.log(*this*);

    }

    minhaArrowFunction = (*curso*) => {

        alert(`Estou fazendo o curso de ${*curso*}!`);

    }

    render() {

        return(

            <div>

                <button *onClick*={*this*.mensagem.bind(*this*, 'Guilherme')}>Clique aqui</button>

                <button *onClick*={() => *this*.minhaArrowFunction('React.JS')}>Arrow Function</button>

            </div>

        );

    }

}