Tecnologias Para Desenvolvimento Web

UNIDADE 02

Componentes e Props

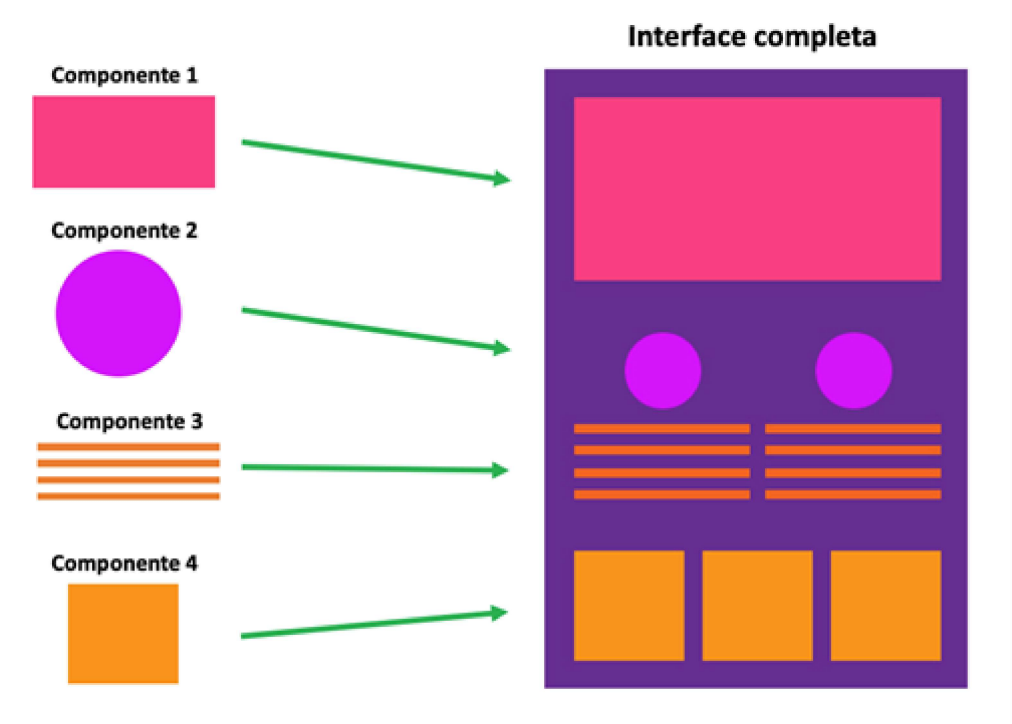
Nesta semana, será abordado o conceito de Componentes e Props em React. É fundamental que se entenda estes dois conceitos, pois serão a base para a construção de nossas aplicações em React. Basicamente todas as aplicações em React apresentam estes dois conceitos inicialmente.

Componentes e Props

Os componentes em React, permitem você dividir a UI (interface do usuário) em partes independentes, reutilizáveis e pensar em cada parte isoladamente. Conceitualmente, componentes são como funções JavaScript, eles aceitam entradas arbitrárias (chamadas “props”) e retornam elementos React que descrevem o que deve aparecer

na tela [1]. Sendo assim, devemos criar diversos componentes de interface diferentes e juntá-los, para assim, formar a interface completa. A Figura 1 demonstra como é a construção de uma interface em React.

Figura 1. Componentes



Fonte: o autor, 2021.

Em React, podemos criar os componentes de 3 formas:

- Function
- Arrow Function
- Class Component

Porém, React não trabalha apenas com Javascript puro, é necessário utilizar o JSX. Em resumo, o JSX é uma sintaxe semelhante ao XML, onde você consegue escrever e compreender de uma melhor forma, como será montado seu componente na UI [1]. A figura 2 mostra um exemplo de JSX.

Figura 2. JSX

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

Fonte: o autor, 2021.

Esta sintaxe “estranha” e nova, não é uma String, nem HTML! É apenas um JSX. Basicamente conseguimos com o uso do JSX, incorporar HTML para dentro de uma variável/constante em Javascript. Mas para que isso? Desta forma, podemos obter o

melhor das duas linguagens, fazendo com as duas (HTML e Javascript) trabalhem juntas. Sendo assim, nesta semana, vamos detalhar os exemplos de componente em **Function** e **Arrow Function** apenas. Class Component ficará para a próxima semana, pois é um pouco mais complexo. A figura 3 mostra um exemplo de componente Function em React.

Figura 3. Componente Function

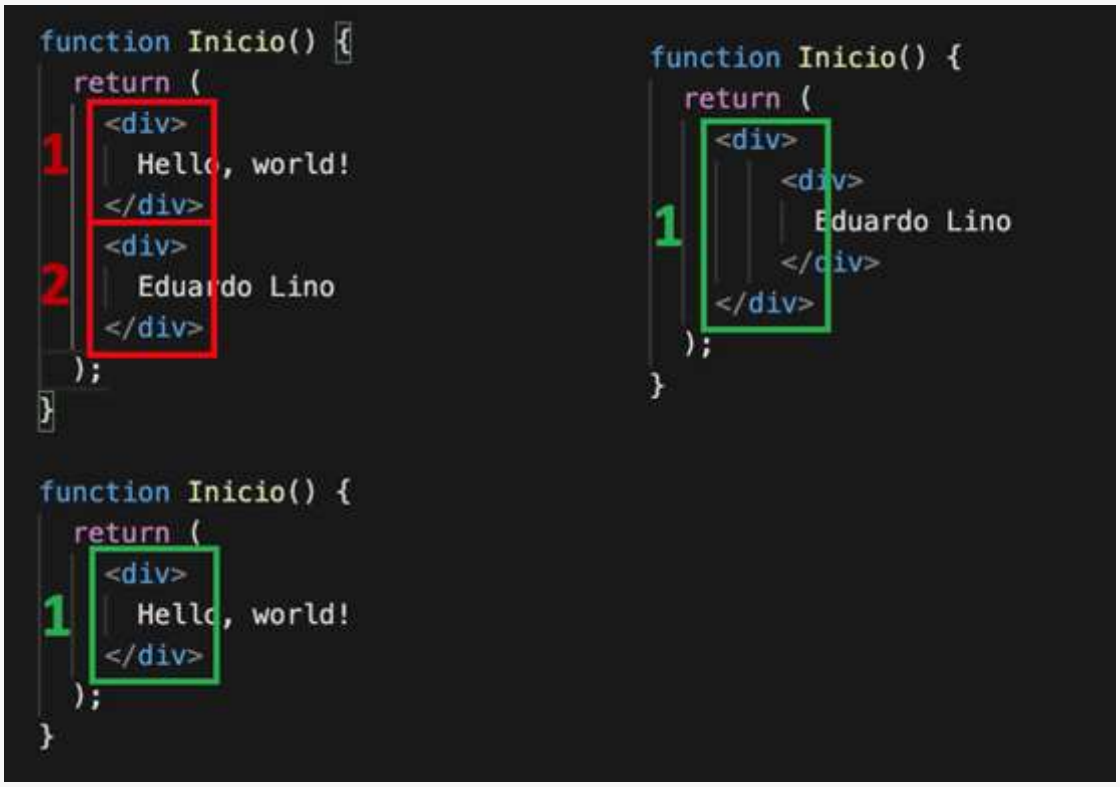
Fonte: o autor, 2021.

Perceba que neste exemplo, temos um pouco mais do que apenas uma **Function**. No início do arquivo, temos a importação do React. Será necessário importar o react seguindo esta sintaxe para trabalharmos com este componente.

Em seguida, temos a **Function** chamada **Inicio**. Desta forma, podemos dizer que este componente se chama **Inicio**. E dentro dele, temos obrigatoriamente uma função chamada **return**. Esta função **return**, irá retornar para a interface um código HTML, que será renderizado na tela, a partir deste componente. Mas para que isso seja possível, temos que exportar o componente, como é demonstrado na linha final. Neste momento, não se preocupe em não entender detalhadamente o significado de cada palavra-chave no código, até porque, no vídeo desta semana, será detalhado um pouco mais.

Perceba também que a função **return** está retornando um código JSX, ou seja, um código similar a um HTML. E **SEMPRE** dentro da função **return** deverá ter um JSX dentro de uma ÚNICA ÚLTIMA tag. Ou seja, sempre o conteúdo JSX que será retornado do **return**, deverá estar dentro de uma tag final, obrigatoriamente. Não pode ter duas tag finais, como é demonstrado no exemplo da figura 4.

Figura 4. Retorno do JSX



Fonte: o autor, 2021.

Já uma **Arrow Function** segue exatamente a mesma ideia da **Function**, a única diferença entre os dois tipos de função é que uma Arrow Function possui uma sintaxe mais curta quando comparada a uma expressão de função do tipo Function. Mas de qualquer forma, internamente, a ideia é a mesma da **Function**. A figura 5 mostra um exemplo de componente **Arrow Function** em React.

Figura 5. Arrow Function

```
1   import React from 'react';
2
3   const Inicio = () => {
4     return (
5       <div>
6         Hello, world!
7       </div>
8     );
9   }
10
11  export default Inicio;
```

Fonte: o autor, 2021.

Além disso, podemos relacionar estes componentes. Ou seja, podemos construir um componente X e executá-lo dentro de um novo componente Y. Por exemplo, podemos criar mais um componente chamado Nome e este componente poderá ser montado

dentro do componente Inicio, como pode ser visualizado na figura 6.

Figura 6. Relacionamento de Componentes

```
1   import React from 'react';
2
3   const Nome = () => {
4     return (
5       <label> Eduardo Lino </label>
6     )
7   }
8
9   function Inicio() {
10    return (
11      <div>
12        <Nome></Nome>
13      </div>
14    );
15  }
16
17  export default Inicio;
```

Fonte: o autor, 2021.

O problema, é que desta forma, caso eu precise ter 3 componentes **Nome** dentro do componente **Inicio**, todos eles vão imprimir na tela o texto “Eduardo Lino”. Desta forma, podemos utilizar as **props** (que significa propriedades), para passar valores como parâmetro para os outros componentes relacionados. A figura 7 mostra o código com o exemplo de props, e a figura 8 mostra com contornos o que foi adicionado para o uso de props.

Figura 7. Componentes e props

```
const Nome = (props) => {
  return (
    <label> {props.texto} </label>
  )
}

function Inicio() {
  return (
    <div>
      <Nome texto="Eduardo Lino"></Nome>
      <Nome texto="Roberto Carlos"></Nome>
      <Nome texto="Ana Maria"></Nome>
    </div>
  );
}
```

Fonte: o autor, 2021.

Figura 8. Componentes e props - contornos

Fonte: o autor, 2021.

Com o uso de **props**, podemos criar novos atributos para as tags que são criadas dentro dos componentes. Desta forma, em **roxo** é demonstrado a criação de um novo atributo chamado **texto** e cada atributo do tipo **texto** está passando um valor diferente em cada componente **Nome**. Assim, para que se consiga usar as props, temos que "dizer" para o componente que vamos utilizá-las! Em **verde**, colocamos a palavra **props** como parâmetro da Arrow Function. Desta forma, já podemos utilizar as **props** neste componente Nome. Para isso, deve colocar um código Javascript dentro da tal <label> e isso somente é possível dentro de { } (abre e fecha chaves), como é demonstrado dentro do contorno em **amarelo**. Dentro de **props**, teremos todas as propriedades do componente, inclusive acesso os valores dos atributos dele.

Em React, também podemos incluir arquivos CSS para que possamos adicionar estilos aos nossos componentes. Para isso, basta criar um arquivo CSS com os estilos e importá-lo para dentro de algum de nossos arquivos JavaScript, conforme é ilustrado na Figura 9. Fique atento em relação ao diretório, deve estar correto para que consiga fazer a importação dos estilos.

Figura 9. Incluindo CSS

```
import "estilos.css";
```

Fonte: o autor, 2021.

A única diferença aqui com CSS, é a sintaxe referente ao nome do atributo atualização para incluir a classe CSS. Aqui com o React, devemos colocar a classe com o estilo no atributo **className**. Com HTML puro, sem a biblioteca React, usamos o atributo **class**,

mas com React, precisamos obrigatoriamente utilizar o **className**, conforme é ilustrado na Figura 10.

Figura 10. Atributo className

```
<button className='estilo-botao'> Gravar </button>
```

Fonte: o autor, 2021.

CONCLUSÃO

Nesta unidade, foi apresentado o conceito de componentes e props que serão a base para a construção de nossas aplicações em React

REFERÊNCIAS

[1] React. **React – Uma biblioteca JavaScript para criar interfaces de usuário**. Disponível em: https://pt-br.reactjs.org/. Acessado em: 7 de outubro de 2021.