SUMÁRIO

1. RECURSOS ESSENCIAIS EXPLICADOS

1.1. SOBRE O ARQUIVO ROOT APP.JS

No arquivo App.js são realizadas as importações dos componentes que serão utilizados na composição geral da aplicação. Não são todas as importações que são realizadas nesse arquivo, mas, digamos assim, as dos componentes de "maior escopo de agregação". Por exemplo, se a página possuir um header, um aside, um main, e um footer, esses serão invocados no App.js, mas os diversos outros possíveis subcomponentes de cada um serão invocados neles próprios.

Também vale citar que o próprio root App.js também é um componente no React, e é invocado no "Entry Point" da aplicação, o index.js.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

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

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

O root da aplicação: index.js

1.2. SOBRE JSX

Ao invés de separar tecnologias artificialmente colocando markup e lógica em arquivos separados, o React separa conceitos com unidades pouco acopladas chamadas "componentes" que contém ambos.

Ou seja, o JSX é uma junção poderosa de um HTML emulado (pois não é HTML, de fato) com Javascript, o que normalmente seria impossível ou evitado ao máximo.

Para saber mais, clique aqui.

1.3. SOBRE COMPONENTES

Componentes, na prática, são criados exatamente como funções, mas que são sempre exportadas e retornam elementos para renderização, e são escritas em **JSX** – a linguagem do React. Essa linguagem é uma mixagem da programação Javascript com uma "emulação" da linguagem de marcação HTML. Segue abaixo o exemplo da estrutura básica de um componente:

```
export default function nomeComponente(Props){ //Parâmetro Props [opcional]
        return ( //Retorno da invocação do componente
                <div> //Elemento pai [opcional]
                        //Conteúdo do componente
                </div>
        )
}
```

O nome do componente, que é uma função com retorno, deve ser o mesmo do arquivo js – sim, para cada componente deve haver um arquivo.

Por sua vez, o retorno do componente, que é o conteúdo que será renderizado ou utilizado na renderização de alguma forma, segue para onde foi invocado na aplicação, podendo ser no arquivo root App.js ou em outro componente.

Muitas vezes o componente terá um elemento pai, como a   do exemplo, mas, podem haver **casos em que um não é necessário**, e para isso basta que seja escrita uma tag sem nome, <> </>, ou, ainda, para uma melhor semântica, <Fragment><Fragment/>. O nome formal para esta prática é "***React Fragment***".

Por último, sobre o parâmetro do componente, "Props", é um objeto que recebe valores, via atributos, a partir da invocação do componente na aplicação, que se dá com tags e o nome dele, assim: <nome_componente />, ou, se houverem atributos para serem enviados, <nome_componente atributo1 = {variavel} atributo2 = "valor" />. Outra possibilidade é utilizar o objeto Props desestruturado, function nomeComponente({atributo1: valor, atributo2: valor}).

1.4. SOBRE PROPS

Comumente nomeados dessa forma, os “Props” são objetos enviados entre componentes, e servem para justamente transmitir dados entre diferentes segmentos da aplicação.

Pense na seguinte situação: existe um componente X, e nele existe uma determinada variável com um valor que é essencial para a programação do componente Y. Ou seja, X produz um valor essencial para Y. Como prosseguir? Na invocação de Y, no componente X, basta também definir um atributo de tag, que será enviado para Y como um argumento, e cujo valor será a própria variável, desta forma: <ComponenteY dado = {variável} />.

Assim, no componente Y, o valor será capturado como um parâmetro da função principal. Existem duas formas comuns: com o Props desestrurado seria function ComponenteY({dado}), e a outra forma, não desestruturada, seria function ComponenteY(Props), com o dado podendo ser acessado por Props.dado.

1.5. SOBRE MANIPULAÇÃO DE EVENTOS

Manipular eventos em elementos React é muito **semelhante à manipulação inline** de eventos em elementos do DOM, possuindo apenas ligeiras diferenças. Eles são escritos na própria tag da estrutura, e em termos de sintaxe mesmo, são idênticos, a não ser pelo camelCase, e uso de JSX para atribuição de valor.

Para fins de exemplo, e considerando que “eventHandler” é uma função criada para trabalhar o evento em questão, um click em um botão será capturado com onClick = { eventHandler } e o envio de um formulário será com onSubmit = {eventHandler}.

Ainda vale citar que esses são manipuladores possíveis e não obrigatórios, introduzidos pelo JSX. Ou seja, os “Listeners” ainda podem fazer presença.

1.6. SOBRE RENDERIZAÇÃO CONDICIONAL

Em React, você pode, então, criar componentes distintos que encapsulam partes da aplicação, e que só são expressos quando retornam conteúdo, e são invocados.

Pois bem: sua própria invocação, e até mesmo o retorno do seu conteúdo, podem ser condicionados. Existem algumas formas de **condicionar a renderização de um componente**.

A primeira forma seria criando condições para o “return()” do componente, isto é, fazer com que ora, se satisfazer uma condição, retorne um conteúdo, e ora, se não, retornar

outro. Sim, podem haver N retornos de um componente se for necessário, cada um como sendo o resultado de uma condição satisfeita pelo estado da aplicação.

A segunda forma seria criar variáveis que, dependendo de um estado da aplicação, recebem um componente ou outro, fazendo com que seu uso no “return()” faça do retorno do componente variável.

A terceira forma seria utilizar operador condicional ternário inline, com JSX, para variar o valor onde for necessário, como no envio de um Props.

A quarta forma seria utilizando o operador if else inline do React, que é realizado com o termo reservado &&. Seu uso se dá da seguinte forma: {condição && output}, sendo output igual a qualquer valor primitivo, componente e até mesmo HTML.

Também é possível evitar a renderização do conteúdo de um componente. Para isso, bastar criar uma estrutura de condições, e no caso em que a condição deve ser evitada, basta retorna null.

1.7. SOBRE HOOKS

A funcionalidade de Hooks trazida a partir da versão 16.7.0 do React visa basicamente oferecer formas de trabalharmos com estado e outras API's sem a necessidade de ter uma classe (stateful component).

2. FONTES

https://pt-br.reactjs.org/docs/getting-started.html