REACT

Curso React + Redux: Fundamentos e 2 Apps do Absoluto ZERO!

Ministrado por Leonardo Leitão.

Sumário

[Fundamentos do React 3](#_Toc130756552)

[Carregar CSS 3](#_Toc130756553)

[Criando Componentes Funcionais 3](#_Toc130756554)

[Adicionando um componente dentro de outro 4](#_Toc130756555)

[Estilizando componente de forma dinâmica 6](#_Toc130756556)

[Renderização condicional 6](#_Toc130756557)

[Comunicação entre componentes 7](#_Toc130756558)

[Direta 7](#_Toc130756559)

[Indireta 7](#_Toc130756560)

[Componente de estado 7](#_Toc130756561)

[Componentes Controlados 8](#_Toc130756562)

[Componente baseado em Classes 8](#_Toc130756563)

[React Router 9](#_Toc130756564)

[Navegação com parâmetro 10](#_Toc130756565)

[**HOOK’s** 11](#_Toc130756566)

[useState 11](#_Toc130756567)

[Atalhos importantes 12](#_Toc130756568)

[Utilitários 12](#_Toc130756569)

# Fundamentos do React

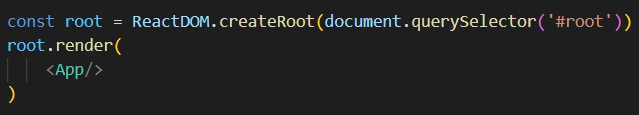
Criando projeto base

|  |  |
| --- | --- |
| npx create-react-app <nome da pasta> | Cria o setup inicial para um projeto em React |
| npm start | Inicializa a aplicação |

1. Excluir todos os arquivos da pasta src.
2. Recriar apenas o index.js
3. Dentro do index fazer os imports



1. ReactDOM – Usado para renderizar elementos na tela.



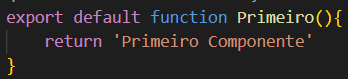
# Carregar CSS

Para carregar um estilo basta fazer o import do arquivo da seguinte forma:



# Criando Componentes Funcionais

Deve ser criado um arquivo que contenha uma função que deve ser exportada.

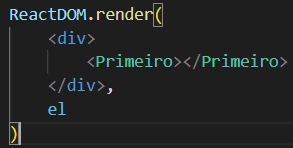


No arquivo que irá receber o componente o index.js primeiro devemos fazer o import dessa função



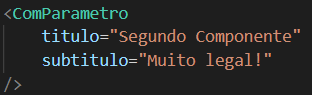
OBS.: O nome usado no import não é necessariamente o nome da função exportada.

Para usar o componente basta que ele seja colocado dentro da parte que será renderizada como uma variável.

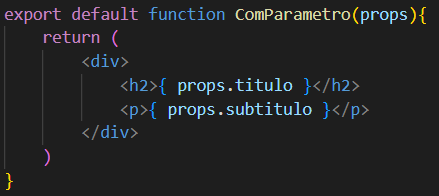


OBS.: Para usar o JSX dentro de um componente é necessário o import do React dentro do componente.

Para passar um parâmetro para um componente. Basta colocar o nome do parâmetro seguido do seu valor:



E uma vez no componente para usar o valor dos parâmetros basta fazer:

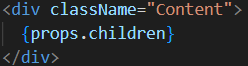


Importante! Cada componente só pode retornar um elemento por vez, para isso muitas vezes são usados os fragmentos <> </>.

Para aplicar uma classe a um elemento a tag usado deve ser className no lugar onde usualmente seria Class.

# Adicionando um componente dentro de outro

Para adicionar um componente que tenha sido passado como uma props para o elemento que estar sendo usado no momento, basta usar o argumento de children. Exemplo:

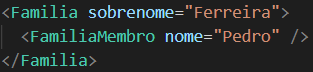


Quando os componentes estão aninhados é possível passar a propriedade de um componente pai diretamente para um componente por meio dos spreads operators:

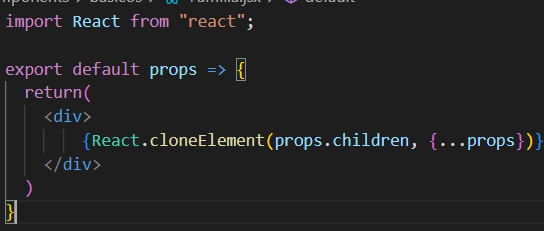


Isso faz com que todas as propriedades do componente pai sejam passadas ao componente filho.

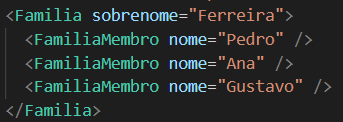
Quando queremos passar componentes filhos de forma dinâmica, ou seja, sem que eles já estejam predefinidos dentro da classe pai da forma:



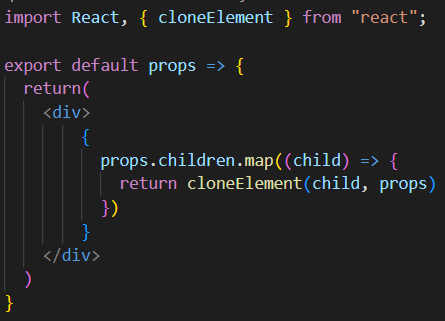
Isso gera um problema pois não conseguimos ter acesso direto as props do componente pai para solucionar isso devemos primeiro dentro da div principal do componente pai abrir um bloco de javaScript e passar o método cloneElement que irá pegar o elemento filho e também as propriedades do elemento pai e montar como elemento único:



Para elementos múltiplos devemos fazer:

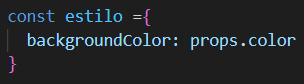


Para isso devemos mapear os elementos e funcionar como no elemento anterior



# Estilizando componente de forma dinâmica

Para passar o estilo de forma dinâmica para um componente devemos criar um objeto que ira armazenar as configurações necessárias e depois passar esse objeto dentro da tag como um atributo de style.



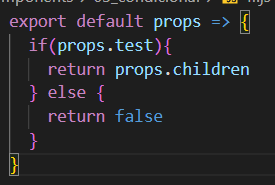


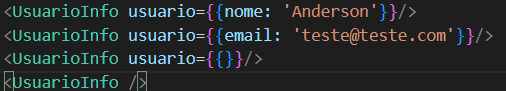
Entretanto é necessário que sejam feitas algumas alterações em atributos que usa hífen como por exemplo background-color pois o js não aceita variáveis com hifens para isso unimos a palavra e passamos a utilizar camel case.

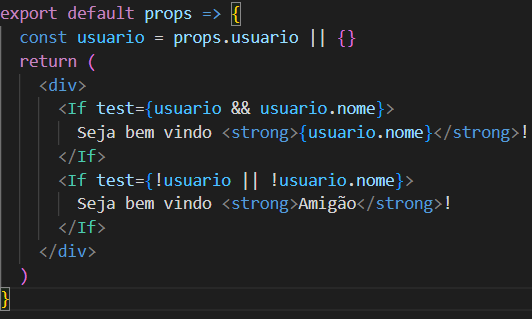
# Renderização condicional

A forma mais usual de fazer a renderização de partes do código é por meio do uso de expressões ternárias.

Ou pode ser criado também um componente filho que seja responsável pela verificação das condições necessárias.







# Comunicação entre componentes

# Direta

É aquela que acontece por meio das props. Como visto na seção “Adicionando um componente dentro de outro”.

# Indireta

Consiste em passar informações do componente filho para o componente pai, logo como não existe uma referência direta como as props a comunicação é feita de forma diferente. Por meio da criação de uma função, de forma que uma função criada dentro de um elemento pai é passado para o elemento filho e quando essa função é chamada devolve os atributos desejados para o componente pai. Essa função é conhecida como CallBack

Componente Pai

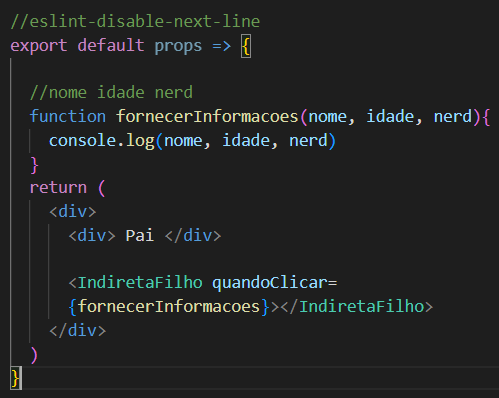
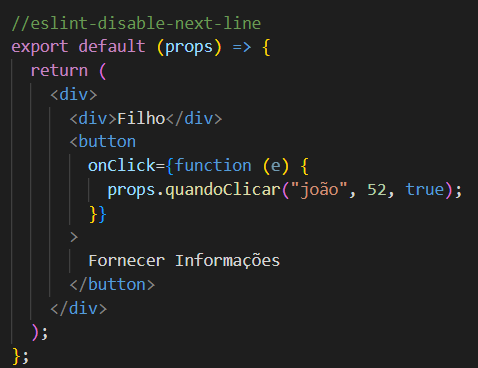


Figura Componente Filho



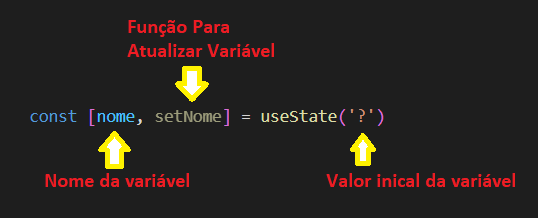
Apesar desse método conseguir enviar informações para o componente pai não é possível simplesmente atribuir esses valores para uma variável e esperar que ela seja renderizada com o novo valor, para que isso aconteça é necessário a criação de um componente de estado.

# Componente de estado

Como explicado logo acima o componente de estado é usado para monitorar quando algo é alterado, isto é, criar um estado para o componente. Para usar esse tipo de componente é necessário fazer o import do useState:

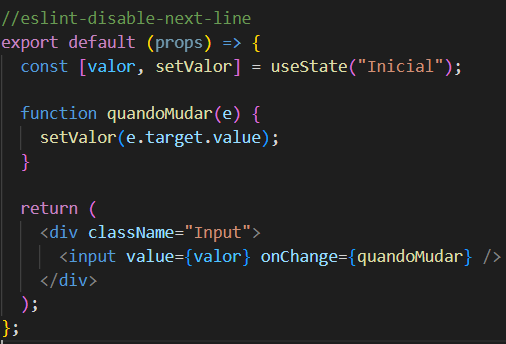


A partir do useState podemos configurar um valor inicial a uma variável e também nomear uma função que será responsável por atualizar o valor dessa variável:



# Componentes Controlados

São componentes que não podem ser alterados de forma direta por meio da interface. De forma que para que qualquer mudança seja exibida em tela primeiro ela deve ter sido feita no componente em si. Isso acontece quando temos por exemplo em um Input algum valor associado a uma variável de estado:



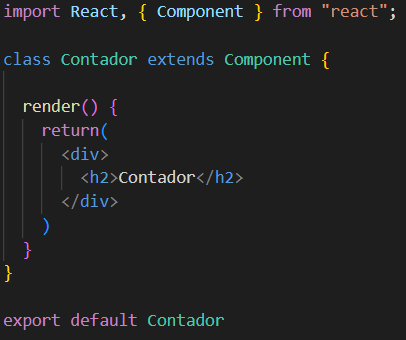
Para que a mudança seja implementada é preciso que a função do onChange seja chamada e altere o valor da própria variável exibida em tela.

Para termos um componente não controlado basta fazer:



# Componente baseado em Classes

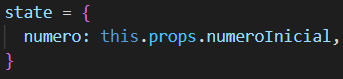
A sintaxe básica para a criação de um componente baseado em classes é a seguinte:



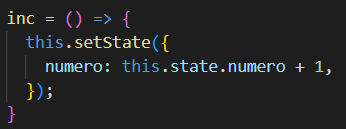
Sendo necessário o uso da library Component e da função render que irá informar aquele que deve ser exibido. Os valores dos componentes Pai (props) ficam disponíveis de uma forma diferente:



Nesse caso os componentes de estado são inicializados de forma diferente:



Já as alterações nos componentes de estado são feitas por meio da função setState recebida de component. Para evitar problemas relacionados ao THIS é importante que seja usada uma arrow function. Dentro da chamada do setState deve ser passado um objeto.



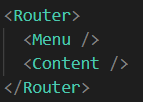
# React Router

Esse pacote do react é responsável por emular rotas de navegação mesmo uma SPA. Para usa-lo é necessário instalar na pasta do projeto o “npm i react-router-dom”.

Já no arquivo App.jsx é necessário que façamos o import do BrowserRoutes:



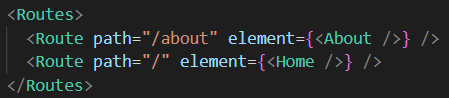
Todos os componentes do App devem estar dentro das tags <Router></Router> para que possam funcionar.



Já no arquivo onde estarão os links que farão a navegação entre as páginas, os imports necessários são:



E a organização das tags é feita usando o Routes para indicar o conjunto dos caminho e o Route para cada um dos caminhos:



Sendo o path a URL que deve renderizar o que esta dentro de element.

Já para fazermos a navegação o que seria uma tag <a> será substituído por uma tag Link:



# Navegação com parâmetro

Para que possamos deixar a URL dinâmica de modo a receber valores variável basta usar a notação de “:”, observe:



A partir de então os valores passados na URL poderão ser acessados na variável id. Para isso é necessário que seja usado o Hook useParams:



A partir de então podem ser acessadas por meio de:



Para tratar o erro de página não encontrada basta criar uma rota com \* e redirecionar ela para uma de not found:



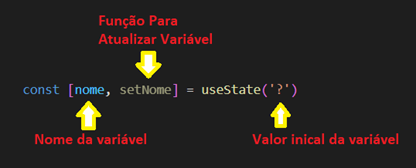
# **HOOK’s**

# useState

É o Hook responsável por criar componentes de estado. Para seu uso é necessário o import de sua biblioteca:



E seu uso acontece da seguinte forma:



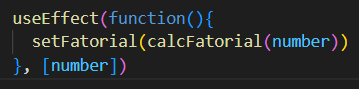
E como estes componentes são controlados sua mudança só pode ocorrer com o uso da função associada para esse fim, no exemplo acima a função setNome. Ver(componentes controlados)

# useEffect

Permite executar efeitos colaterais em componentes funcionais, ou seja, é a capacidade de que ao alterar o estado de um componente outros sejam alterados em decorrência desta mudança. Para usar este Hook o import necessário é:



E seu uso ocorre sendo passados a função que será executada quando houverem mudanças e em seguida aquilo que deve ser observado buscando por mudanças:



No caso acima sempre que o valor de number for modificado , função associada será chamada.

# useRef

Retorna um objeto mutável com a propriedade .current, ou seja,

É necessário fazer o import desse Hook do React:



Ao instanciar uma variável, devemos atribuir o useRef a uma variável, que se comportará como um objeto a partir do qual a chave current sera usada como um componente de estado.

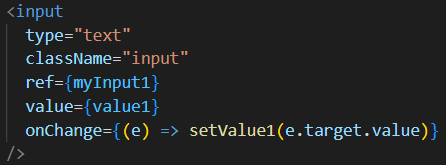


A diferença para o useState é que nesse caso podemos fazer mudanças no valor desse objeto sem que seja gerada uma nova renderização fazendo com que seja possível alterar o seu valor de forma direta:



O useRef pode ser usado paga captura um elemento do HTML, por meio da propriedade ref. Quando criamos um useRef e o atribuímos dentro de um elemento HTML esse elemento passa agora a ser monitorado por esse useRef.





Dessa forma dentro do atributo current existe uma referencia para o elemento input.

# useMemo

Retorna um valor memoizado, em termos práticos quando temos dados que demandam um processamento demorado, esse Hook pode ser usado pra que o programa não fique travado em determinadas partes que não dependem deste processamento. Se comporta de forma bastante semelhante ao useEffect na forma de parâmetros.



# useCallback

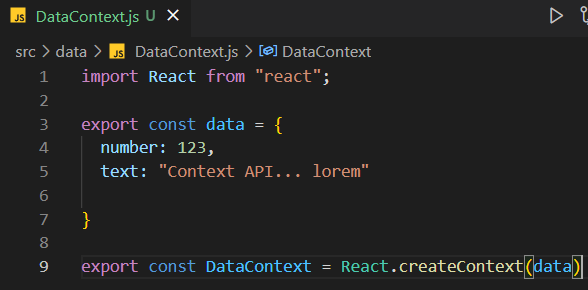
Estudar mais.

# Context API - useContext

Serve para que em aplicações que são estruturadas com o uso de componentes, aqueles componentes que se encontram distantes possam se comunicar de maneira mais simples. Para solucionar esse problema é que existe o useContext. A solução para isso é fazer a criação de uma estrutura que esteja acima de todos os componentes e que por meio do context disponibilize os dados para todos os componentes filhos.

O Hook useContext Aceita um objeto de contexto e retorna o valor atual do contexto! Vamos ver em um exemplo de uso:

1. Primeiramente criamos esse elemento que ira ser responsável por “armazenar” os dados de forma global a aplicação.

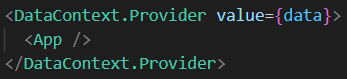


Trata-se de uma constante que nada mais é que um objeto com dados, e que deve ser exportada como um context como vemos acima.

1. Dentro do componente que envolve todos os componentes da minha aplicação, ou no local onde ele é aplicado “index”, devemos fazer o import da constante de contexto que foi exportada, juntamente com os dados que estão presentes nela.



1. Com ele importado devemos criar uma tag <DataContext.Provider> que irá envolver o componente principal da aplicação, com o parâmetro de value com os dados importados:



1. Agora dentro do componente em que queremos fazer uso dos dados que estão armazenados no contexto devemos importar os a constante de contexto que foi criada.



1. Agora criamos uma constante que irá armazenar e usar os dados por meio do useContext.



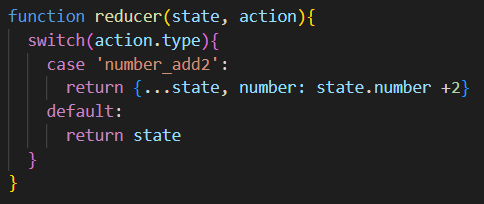
1. Agora basta usar a constante criada como um objeto usual, para acessar os valores presentes no contexto:



# useReducer

É outra forma de ter estado em componentes funcionais.

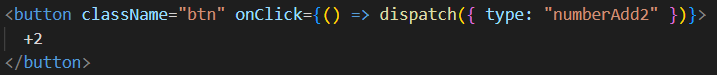
Para usar esse Hook devemos primeiramente criar uma função reducer (responsável por evoluir o estado, pode ter outro nome) que irá receber um estado e uma ação, onde a ação nada mais é do que aquilo que queremos executar. Normalmente as ações são tratadas via switch.



Uma vez criada essa função devemos, construir o useReducer, onde recebemos o estado mais atualizado e uma função que realiza as atualizações, e devemos passar o reducer e o estado inicial:



Agora basta disparar esse evento passando o tipo desejado:



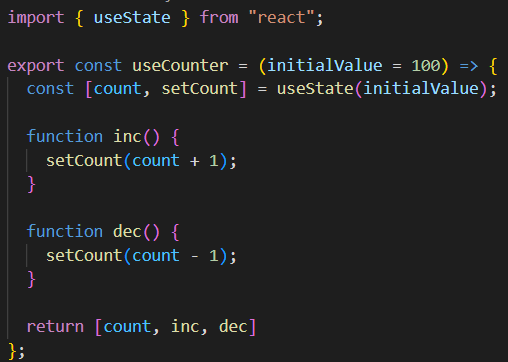
# Criação de Hooks Personalizados

Para fazer a criação de um Hook próprio devemos:

1. Criar uma pasta onde serão armazenados os Hooks personalizados. **(Opcional)**
2. Criar um arquivo .js que será o nosso Hook.



1. Em seguida dentro deste arquivo devemos exportar uma constante com a lógica desse novo Hook, podendo inclusive fazer uso de outros Hooks e funções diversas.



1. Uma vez criado o Hook no componente me que ele for necessário seu uso se dá de forma habitual, faz-se o importe e então passamos valor inicial e desestruturamos os parâmetros e funções recebidas.



# Criando um Projeto com Next.js

O next.js é um framework que é usado para facilitar o uso do React e sua integração com o backend. Para criar uma aplicação com esse framework devemos fazer:

1. Na pasta onde o projeto será executado rodar o comando “npx create-next-app <nome da pasta>”
2. Uma vez criado o projeto, devemos entrar na pasta do projeto e executar o comando “npm run dev” para rodar a aplicação. A partir de então a aplicação pode ser acessada no localhost:3000.

O next possibilita que quando fazemos a criação de uma pagina dentro da pasta pages ela já fique acessível imediatamente por meio daquela rota no navegador.

# Atalhos importantes

|  |  |
| --- | --- |
| Alt + shift + f | Formata todo o código. (quando o pretty esta instalado) |

# Utilitários

* Site com vários exemplos de gradientes: <https://uigradients.com/#WhatliesBeyond>
* A função focus() é responsável por focar em determinado elemento HTML.