**CURSO DE REACT – JOÃO RIBEIRO**

Fonte deste estudo: <https://www.youtube.com/watch?v=ZbCxaaTEkmI&list=PLXik_5Br-zO-JH-3Bxgdj6m0YcHEDeI1e>

DIRETÓRIO ASSOCIADO A ESSE CONTEÚDO:

C:\Users\Luis\Desktop\React\Exercicios\02-curso-de-react-joao-ribeiro\01-projeto-react

[1] Criando projeto de React com Vite

*npm create vite@latest*

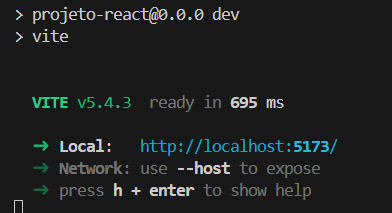
O comando acima verifica se o Vite está instalado. Caso não esteja, instalará.

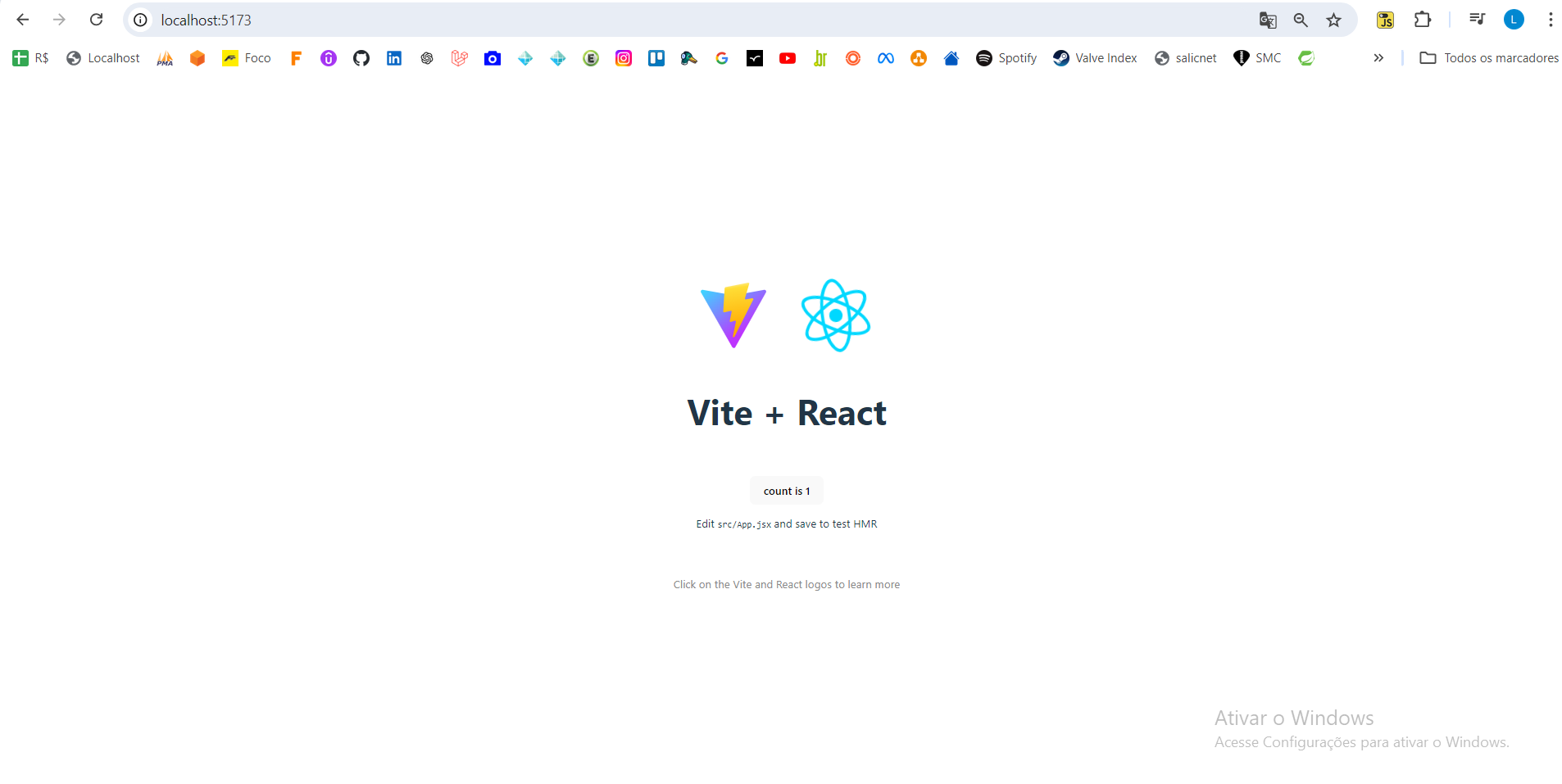
Em seguida iremos escolhendo as opções desejadas para a instalação.

|  |  |
| --- | --- |
|  | A biblioteca que queremos, React. |
|  | A linguagem que queremos utilizar. SWC é Speedy Web Compiler, um otimizador par JS e TS. |

Em seguida, devemos acessar a pasta do projeto e rodar o **npm install** para instalar as dependências que estão listadas no package.json.

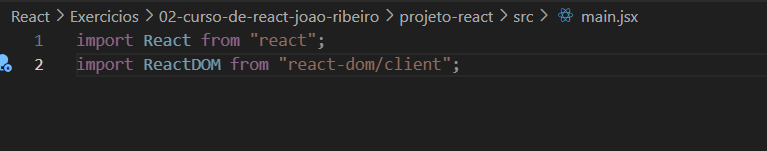
Agora rodamos o comando **npm run dev** para inicializarmos a aplicação.



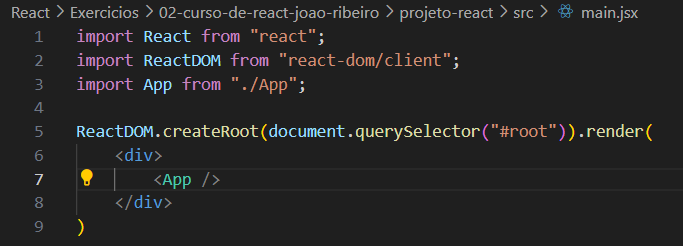


Apagamos todo o conteúdo da pasta src para iniciarmos do zero e criamos o **main.jsx**, visto que no index, porta de entrada da aplicação, ele está sendo referenciado e caso não seja encontrado, retornará erros.

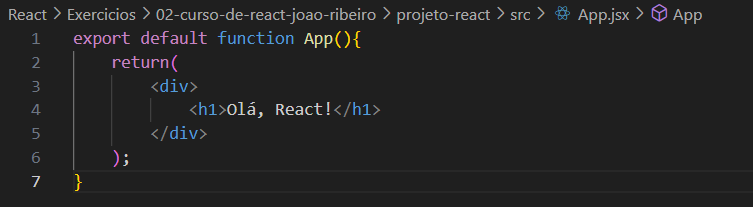
Realizando importações necessárias:



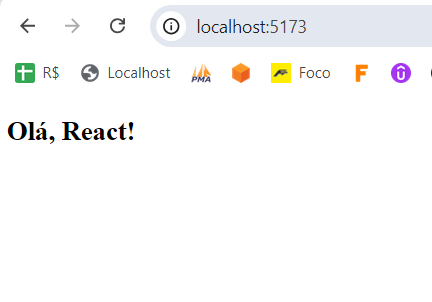
Agora, no main.jsx, nós tratamos de criar o elemento que será renderizado no index.html, que chamará o elemento DOM com id root.



Criaremos um componente App.jsx, exportaremos e importaremos para dentro de main.jsx.

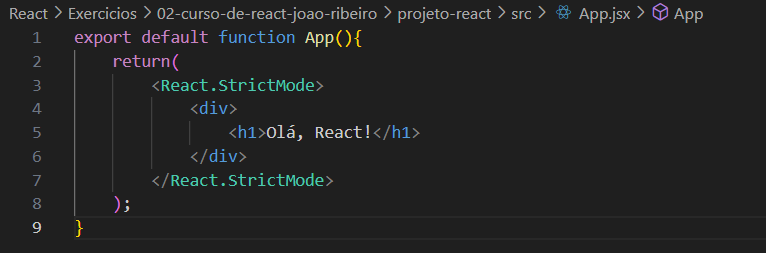


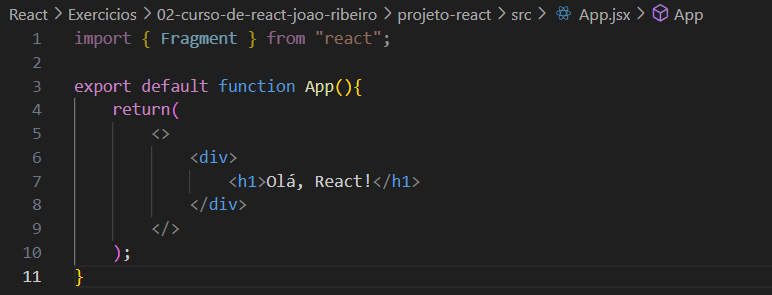
E veremos o seguinte resultado:

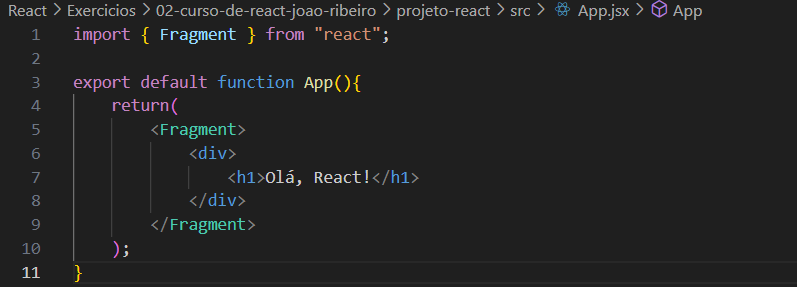


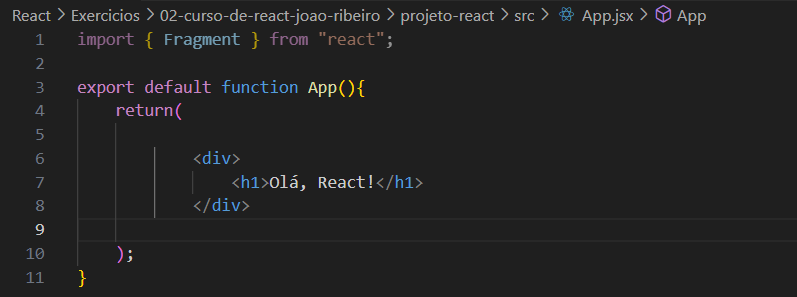
[2] Contentor da aplicação

Sabendo que em **App.jsx** estará o elemento que renderiza a aplicação. Na fase de desenvolvimento é possível utilizar o contentor **React.StrictMode** e quando a aplicação for posta em produção, é possível conter com <></>, <Fragment></Fragment> ou simplesmente com <div></div>.

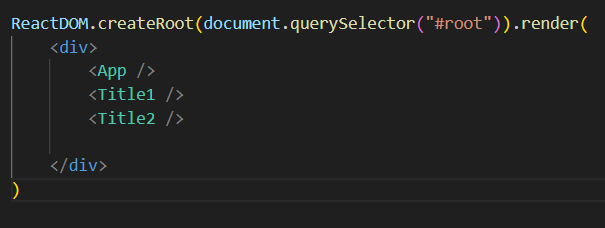








É possível criar um script com vários componentes.



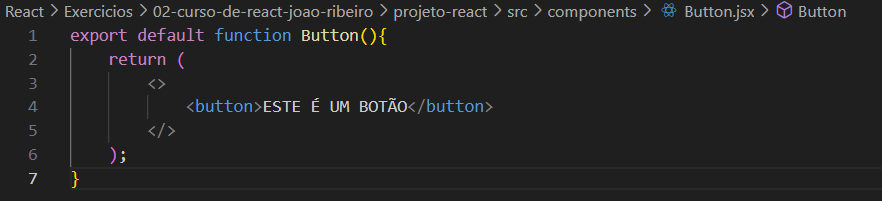
Que poderá ser importado desta maneira:



Mas o comum é que cada componente esteja em um script separado e os componentes organizados em pastas.

[3] Importando componente para dentro de outro componente

Abaixo o componente:

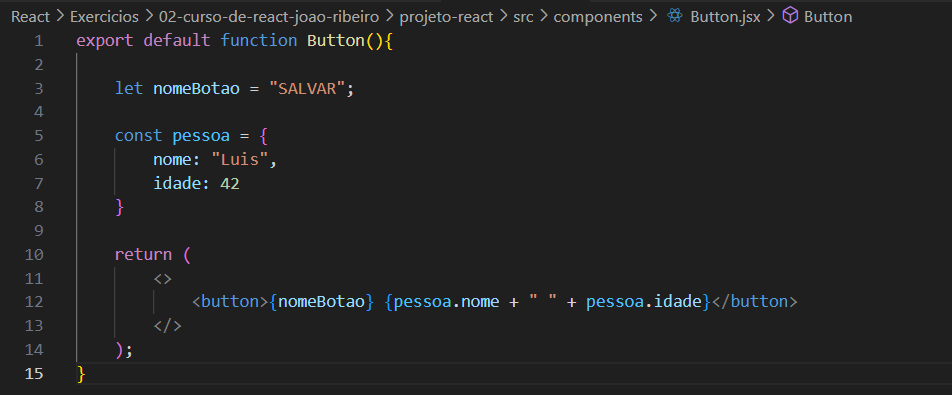


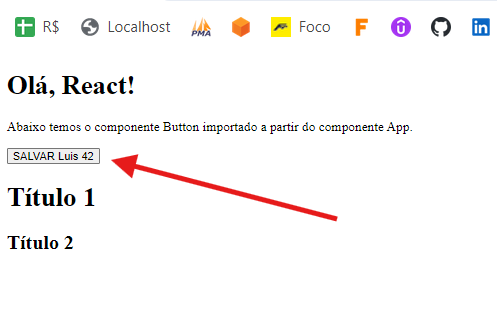
E abaixo a importação do componente para dentro do componente App. Note que é necessário fazer import do React e também do componente que está sendo usado.



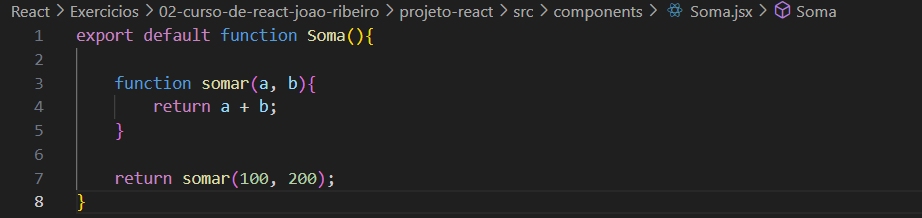
[4] Apresentando dados de variáveis ao carregar um componente

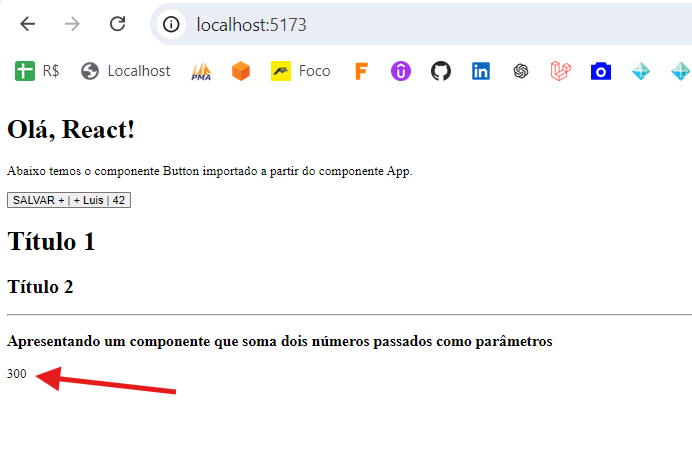
No componente eu criei uma variável e utilizei como uma espécie de label do botão e, desta maneira, carregou o valor atribuído à variável.





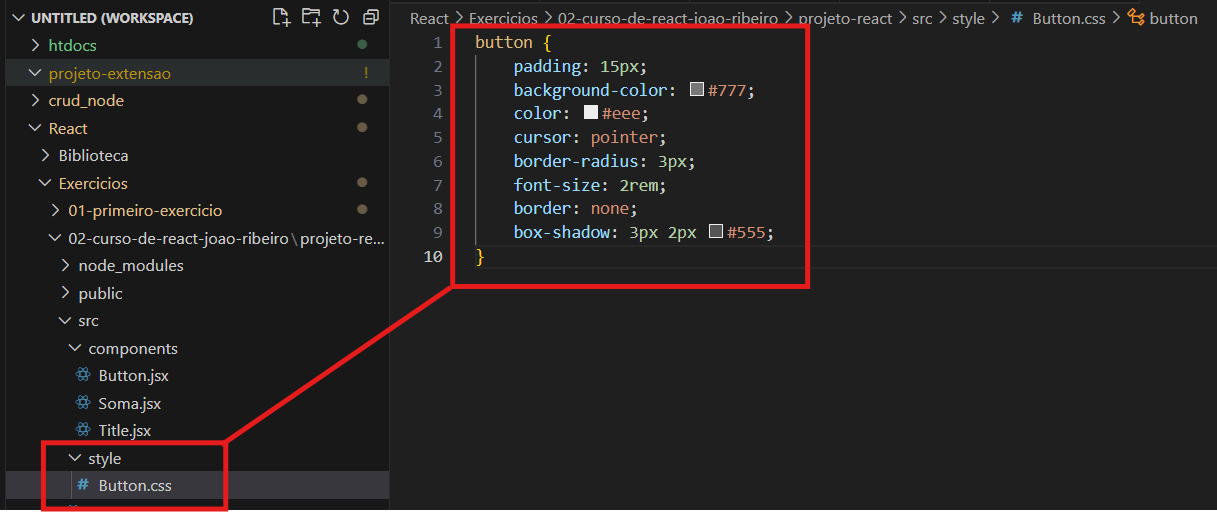
Agora vamos realizar a construção de um componente que possua parâmetros para somar dois números.





[5] Colocando CSS nos elementos

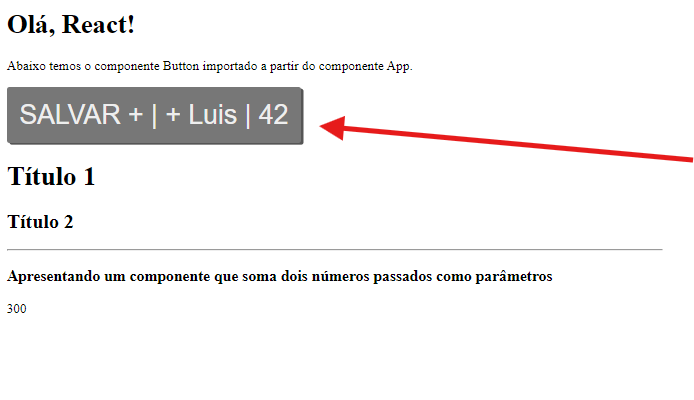
Para inserir CSS, criei uma pasta para guardar os documentos de estilo e criei o estilo para um componente, o botão:



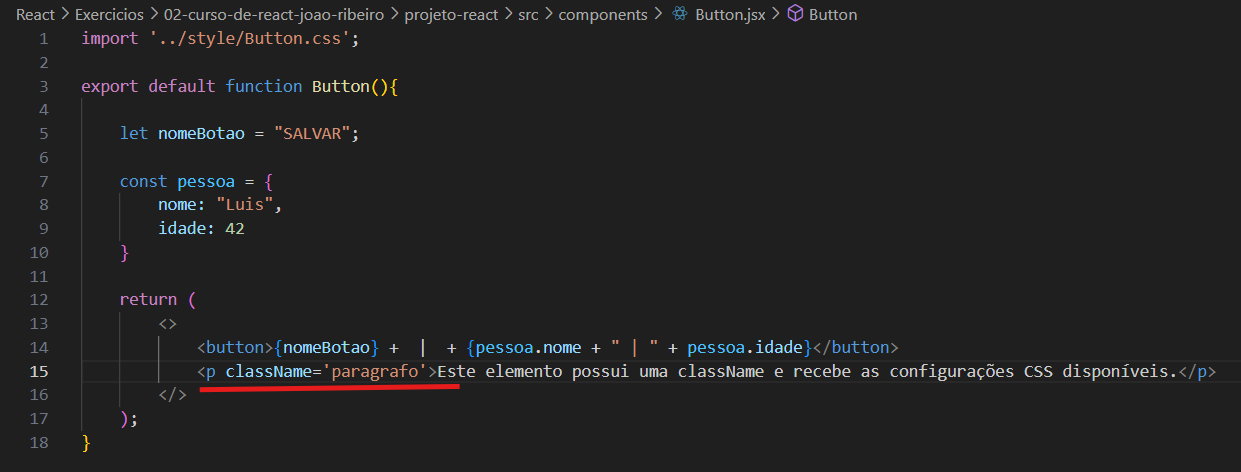
Em seguida importei o arquivo:

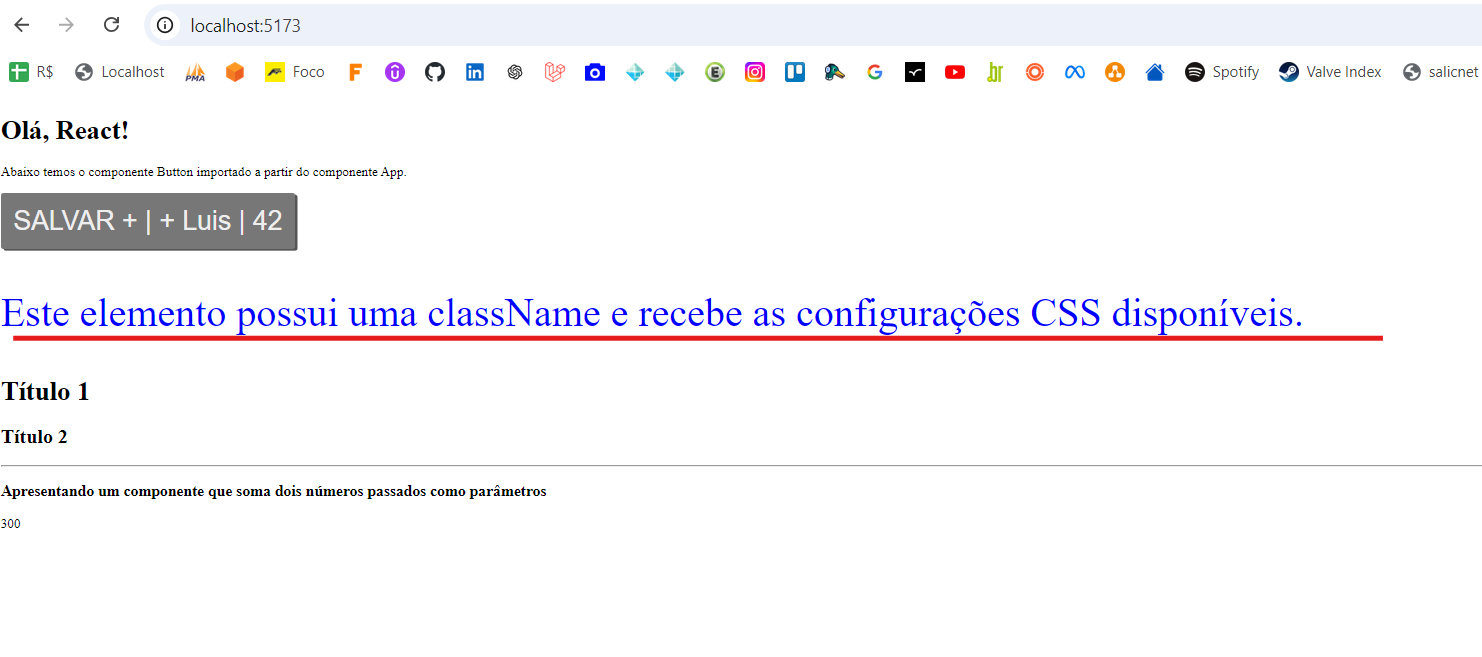


E o resultado foi:



É possível também usar classes em elementos e aplicar regras css, mas ao invés de class (que colidiria com o conceito de classe no JS), utilizamos o atributo className. Veja:





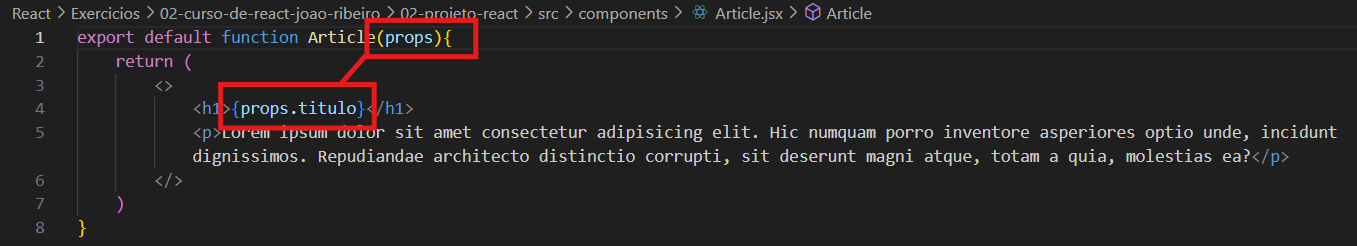
DIRETÓRIO ASSOCIADO A ESSE CONTEÚDO:

C:\Users\Luis\Desktop\React\Exercicios\02-curso-de-react-joao-ribeiro\02-projeto-react

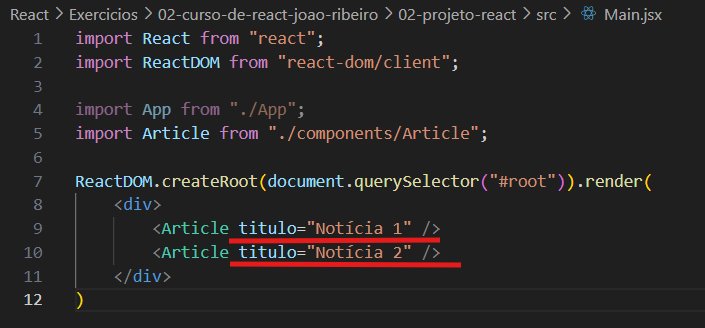
[1] PROPS

Uma maneira de utilizar componentes e personalizando alguma informação, como título, é utilizando as Props.

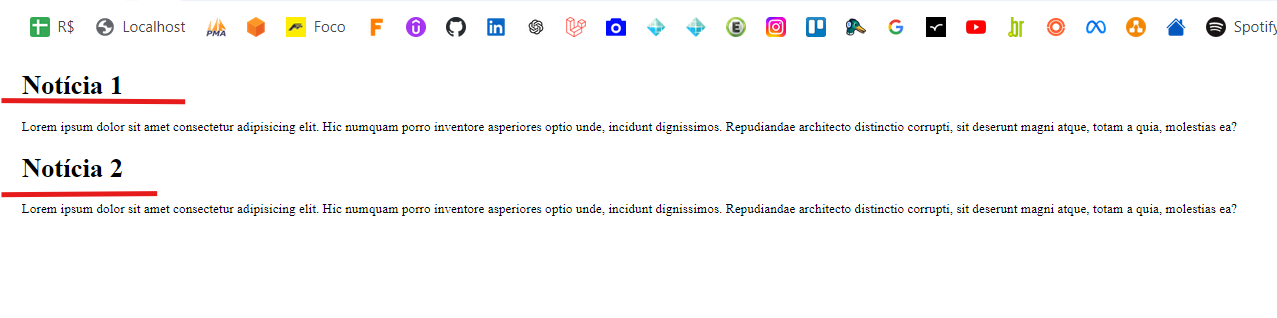
No componente, a sintaxe fica:



Quando chamamos este componente, passamos o valor da propriedade que criamos:

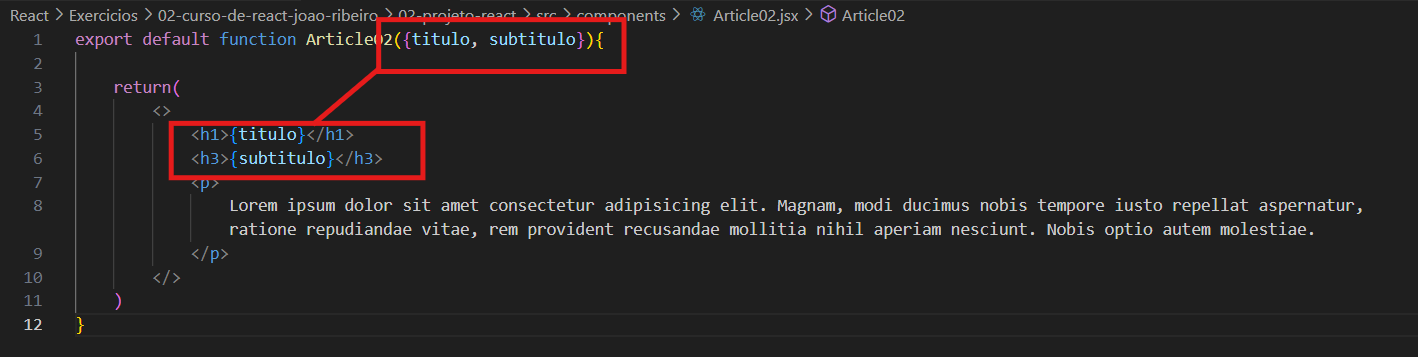


O resultado no navegador é:



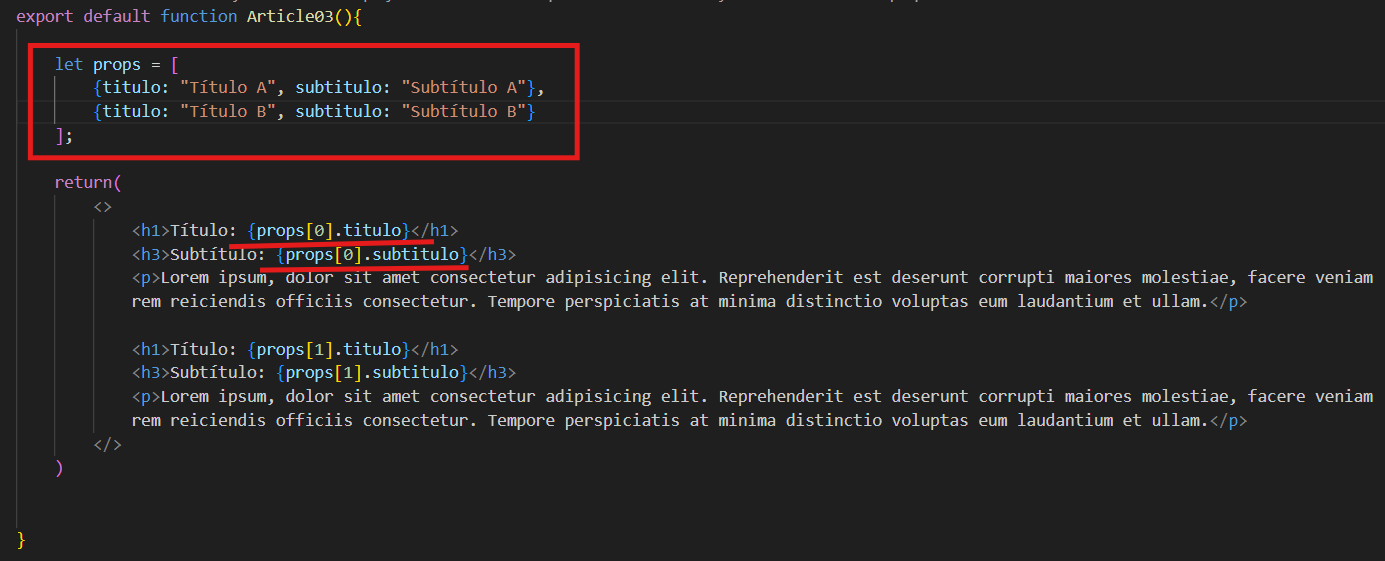
Observação: O nome ‘props’ é uma convenção para denotar propriedade, mas pode ser qualquer outra cadeia de caracteres válida.

A seguir há outra forma de utilizar props, passando sendo o objeto props, mas diretamente como propriedades:

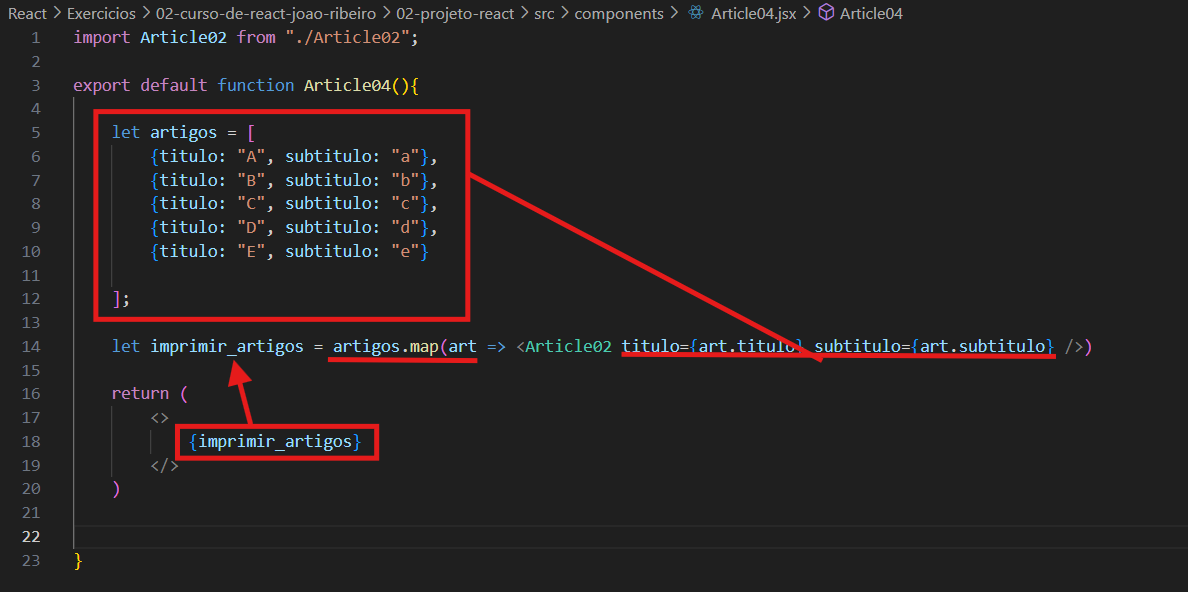


Perceba que não há uso de props.titulo porque as propriedades já estão declaradas no argumento.

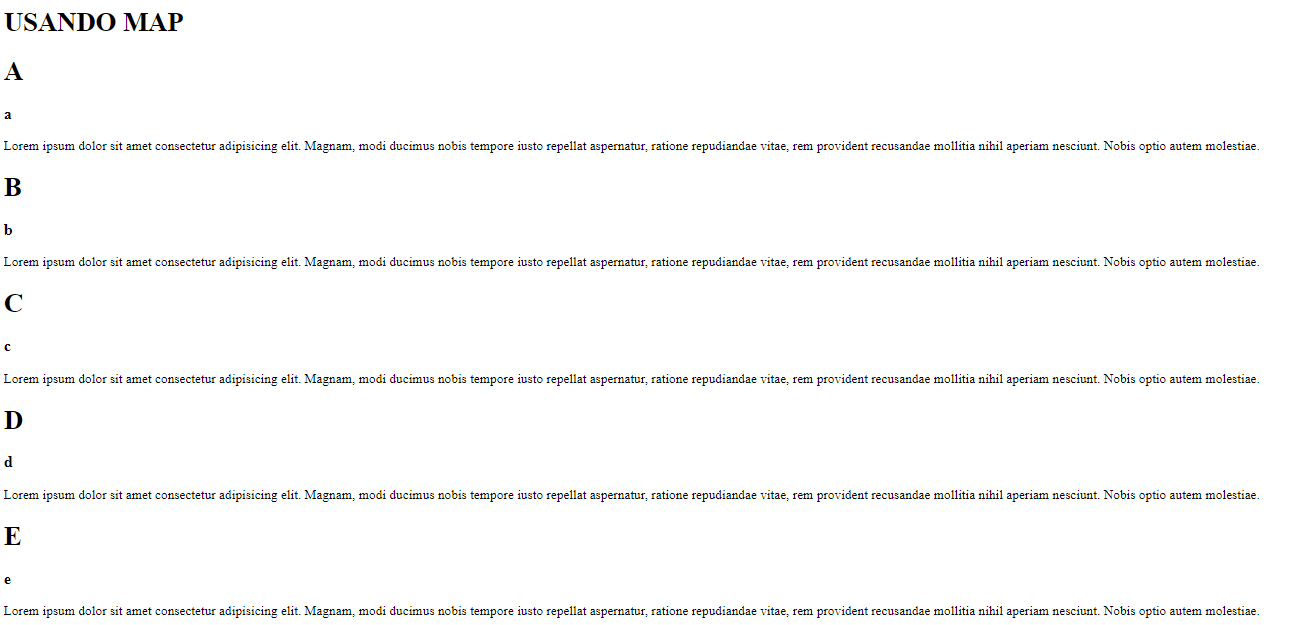
Agora passando props utilizando um objeto:



[2] Usando MAP para iterar



Acompanhe: há um array de objetos. Utilizamos o método map, que itera sobre o array. Chamamos o componente Artigo02, que possui 2 props e atribuímos o resultado da iteração como valores destas propôs (título e subtítulo). Mais abaixo, retornamos o resultado de map, que está em imprimir\_artigos. O resultado no navegador é:



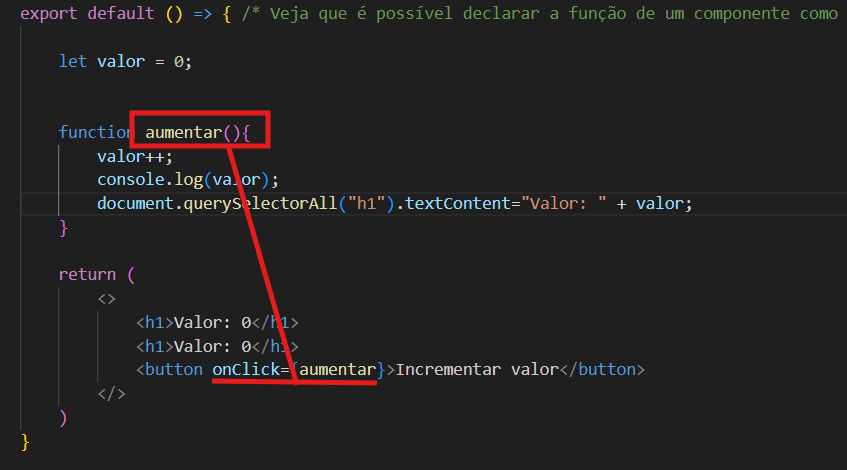
DIRETÓRIO ASSOCIADO A ESSE CONTEÚDO:

C:\Users\Luis\Desktop\React\Exercicios\02-curso-de-react-joao-ribeiro\03-projeto-react

[1] Hooks

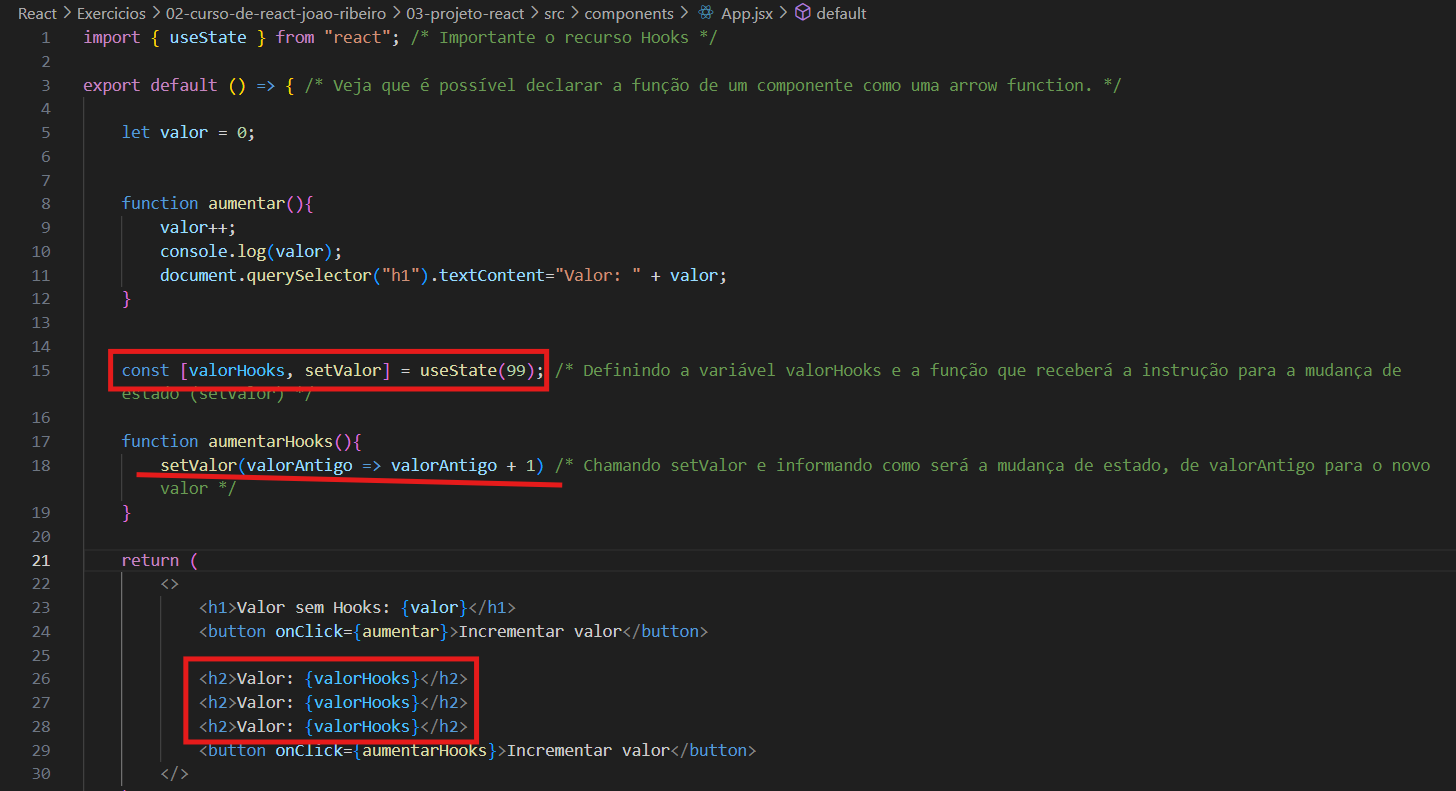
Hooks em React são funções especiais que permitem que você use recursos do React, como estado e ciclo de vida, em componentes funcionais. Antes dos hooks, esses recursos estavam disponíveis apenas em componentes de classe.

Assim, por exemplo, se queremos alterar o estado de um elemento, podemos desenvolver com js puro. Exemplo:



Com o exemplo acima, estamos aumentando o valor em 1 unidade toda vez em que o botão for clicado.

Com Hooks, fica desta forma:



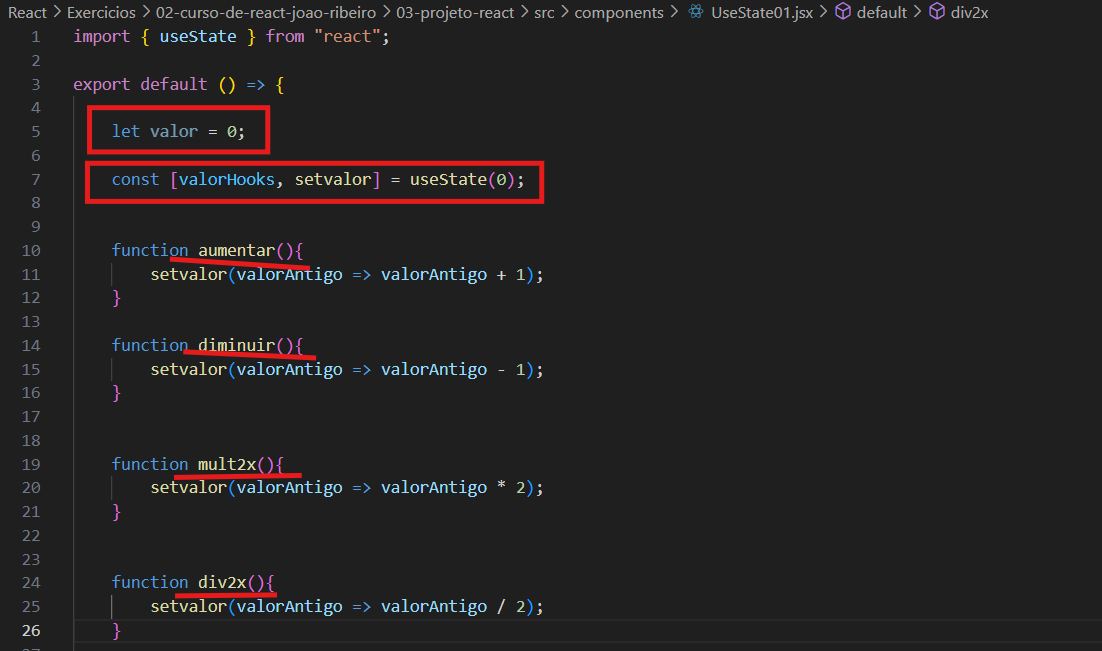
Se quiséssemos aplicar a lógica anterior para modificar o valor várias vezes seguidas, não iria funcionar. Com o Hooks, ao clicar no botão, ele alterará o valor de todos os elementos e não apenas do primeiro.

O Hooks está funcionando da seguinte maneira: o ‘valorHooks’ é a variável que mudará de estado; o ‘setValor’ é a função que será chamada para a mudança de estado a partir do onClick do botão, o **useState** atribui o valor inicial da variável ‘valorHooks’ e a função obedece ao princípio de ‘valor antigo => valor antigo + diferença’ que gera o novo estado.

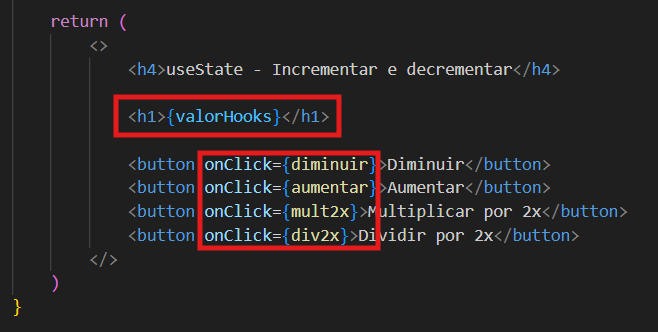
[2] UseState

Vamos utilizar o useState para produzir um exemplo com uma complexidade um pouco maior.

Veja:



Note que há um useState preparado para servir a 4 possíveis funções que realizam operações de incremento, decremento, multiplicação e divisão e abaixo se pode ver a aplicação disto com a impressão do valor e 4 botões, cada um recebendo uma função que realizará uma destas operações:



No navegador, o resultado é:



DIRETÓRIO ASSOCIADO A ESSE CONTEÚDO:

C:\Users\Luis\Desktop\React\Exercicios\02-curso-de-react-joao-ribeiro\ 04-projeto-react-with-branchs-by-steps

ATENÇÃO: A PARTIR DESTE PONTO IREI TRABALHAR COM A CRIAÇÃO DE BRANCHS NO GIT VISANDO QUE CADA TÓPICO/EXERCÍCIO ESTUDADO ESTARÁ NA SUA DEVIDA BRANCH E PODERÁ SER ACESSADO/CONSULTADO AO RETORNAR O ESTADO DO PROJETO AO MOMENTO DESEJADO.

Após criado um novo projeto Vite + React, os comandos utilizados foram:

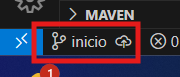
git init

git add .

git commit -m “Projeto Zero Bala”

git checkout -b nomes-da-branch

Ao clicar no nome da Branch atual



Na barra de pesquisa superior surge a listagem com as branchs disponíveis.

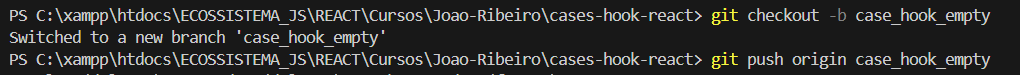
Criado um repositório no git para estes exercícios:

<https://github.com/artisanbahia/cases-hook-react.git>

Rodei o comando para sincronizar com a pasta remota.

git push origin -u master

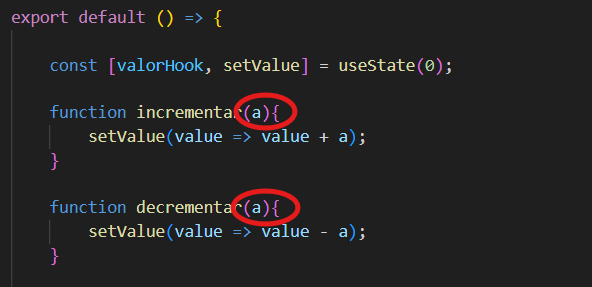
Agora veja que a lógica da criação das branchs e da sincronização delas com o diretório remoto é assim:



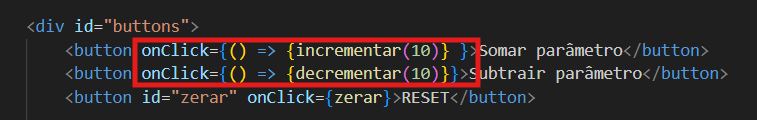
Ou seja, crio a Branch localmente e depois realizo um push criando uma Branch com o nome da criada localmente.

09/09/2024

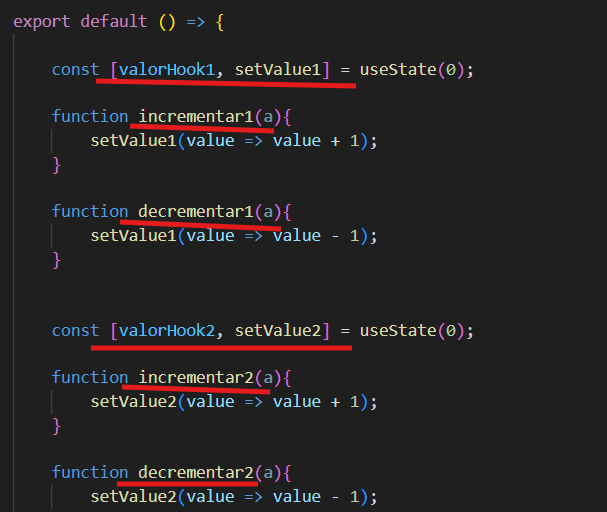
Note que é possível passar parâmetro na função que interage com o useState:



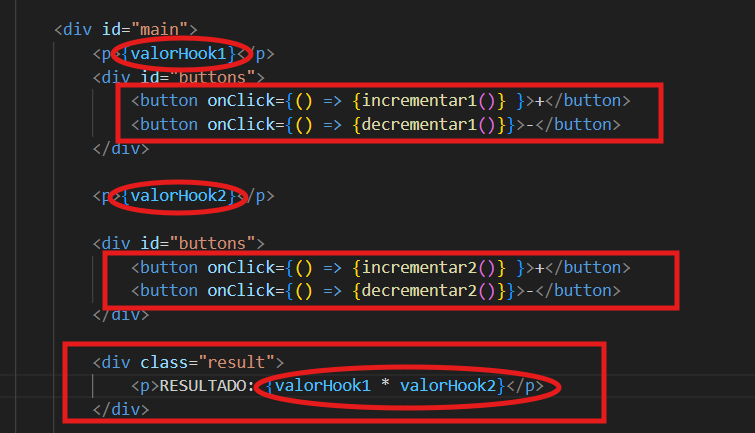
E utilizar uma arrow function para configurar o onClick, visto que não daria certo passando simplesmente a função no valor do onClick:



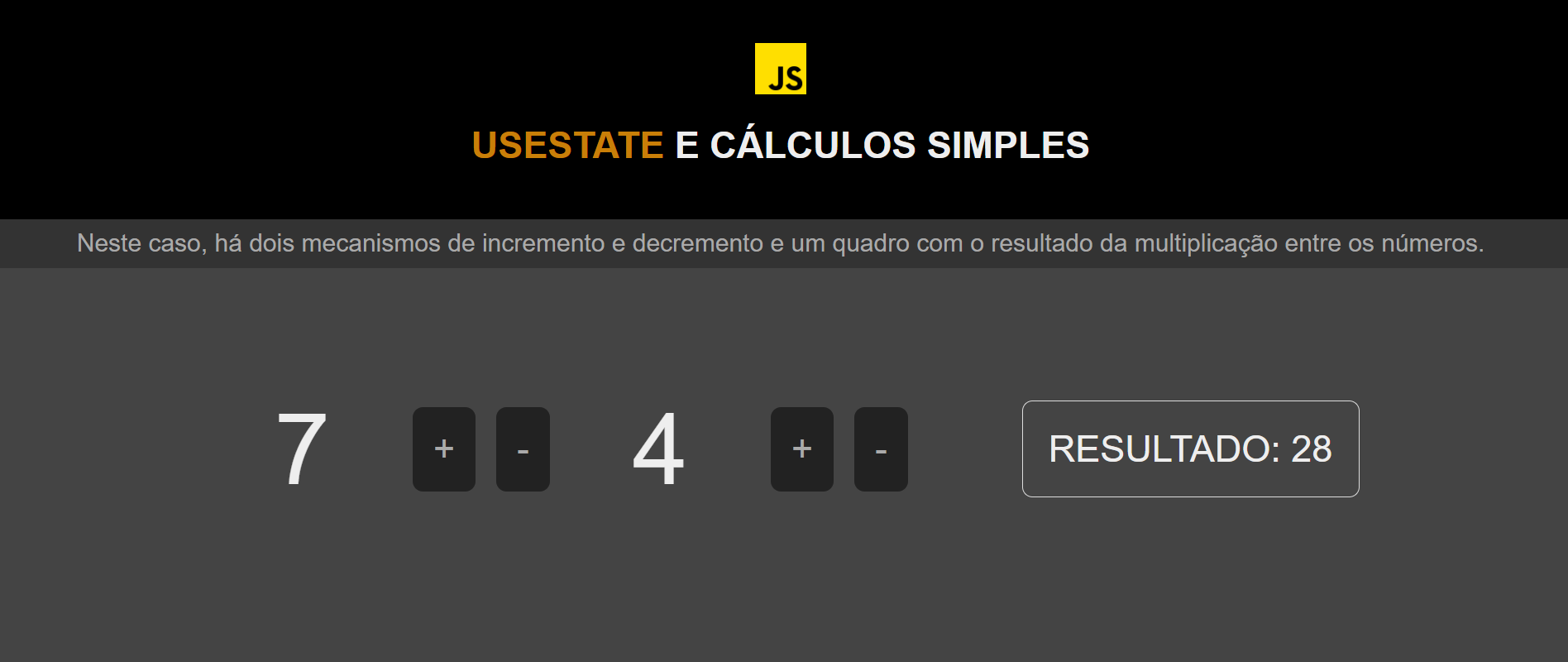
De seguida, uma mudança substancial, com a duplicação dos mecanismos de incremento e decremento:



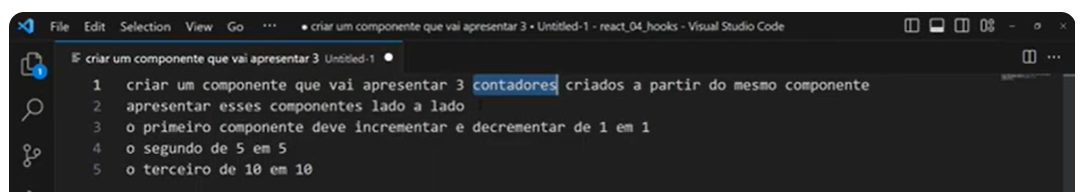
Duplicação dos botões e criação de um box para receber o resultado da multiplicação entre os dois valores:



Veja recorte da view:



Dando sequência, desenvolvo a resposta para a seguinte proposta:

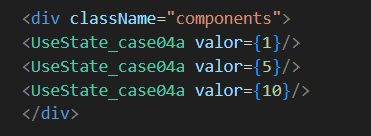


**USESTATE\_CASE04**

Neste novo exercício, criei um contador e reutilizei três vezes dentro de outro componente. Assim, ficaram três contadores na mesma página.

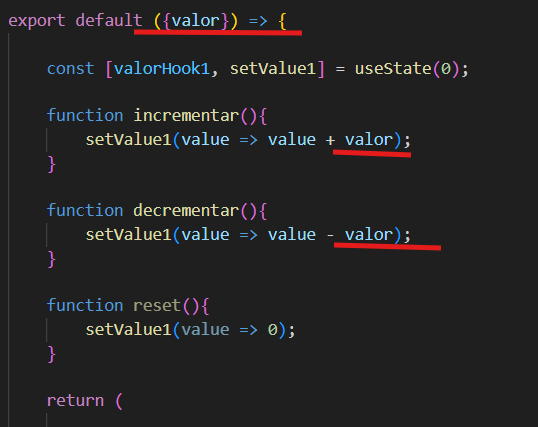


A novidade é que cada contador possui mecanismo de incremento e decremento com valores diferentes. Isso foi possível ao utilizar props. Veja:



Acima temos a prop valor e o valor que ela possui em cada reutilização do componente.

Abaixo temos, no componente em que o contador foi construído, a função com a indicação de prop como parâmetro.

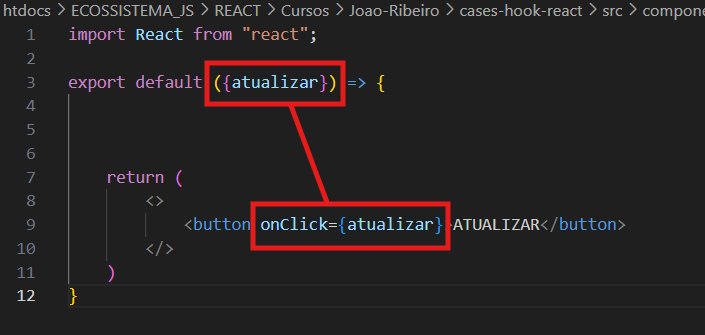


Continua também o mecanismo de reset.

**USESTATE\_CASE05**

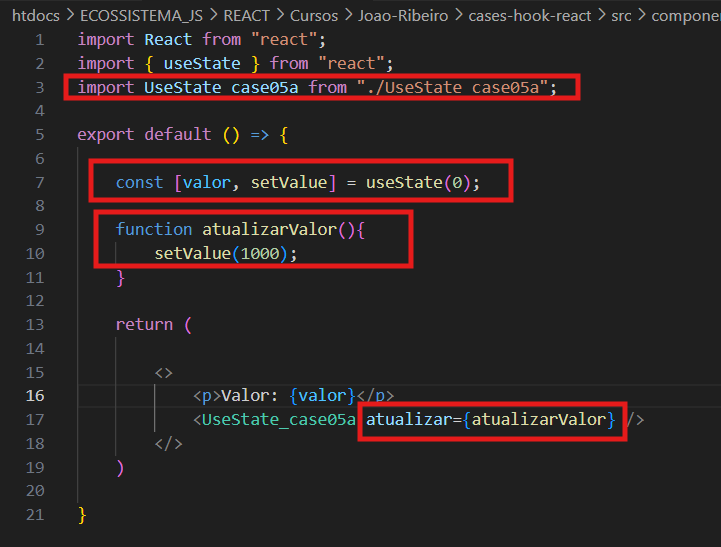
Acompanhe com atenção esse exemplo.

Eu tenho um componente que possui um botão e executa uma função chamada atualizar. Veja:



Acontece que essa função não está definida no corpo do componente, mas sim, no corpo do componente pai, ou seja, o componente que o chama.

Agora veja que lá no componente pai, que chama este componente, há importação do componente, definição do useState, definição da função que atualiza o valor, importação do componente filho com a passagem do prop.



Note ainda, para não se confundir, que lá no componente filho há o nome do prop e aqui no componente pai, o prop recebe como valor o escopo da função declarada no componente pai, por isso que ele funciona.

A view é esta:



11/09/2024