# Aula 3 - Criando os próprios Hooks

**Docupedia Export**

Author:Ferro Alisson (CtP/ETS)
Date:25-Aug-2023 13:25

# Table of Contents

# 1 Renderização condicional

Até as aulas anteriores estávamos trabalhando somente com o JSX semelhante ao HTML, sem usar JavaScript.
No projeto que estávamos desenvolvendo da aula anterior, quando o número for 0, o botão para deixar o número zero não pode aparecer, da mesma forma com o 5, para isso vamos usar a renderização condicional. O que isso significa? Significa que temos uma condição, caso for verdadeiro então será exibido um componente, caso seja falso outro componente.

```
import { useState } from "react"

export default function Counter(){
    var [num, setNum] = useState(0);

    function handleFive(){
        setNum(5);
    }

    function handleZero(){
        setNum(0);
    }

    return (
        <>
            {num === 0 ? <></> : <button onClick={handleZero}>Clique aqui</button> }
            {num}
            {num === 5 ? <></> : <button onClick={handleFive}>Clique aqui</button> }
        </>
    )
}
```

Usamos assim um if ternário para realizar essa renderização condicional, de forma semelhante podemos fazer com listas em vários componentes.
Mas ainda assim esta um pouco complicado a leitura do código, podemos melhorar da seguinte forma
Iremos criar uma nova função para controlar a lógica da renderização, como essa função será utilizada apenas dentro desse componente, podemos deixar nesse mesmo arquivo

```
import { useState } from "react"

export default function Counter(){
```

```
    var [num, setNum] = useState(0);

    function handleFive(){
        setNum(5);
    }

    function handleZero(){
        setNum(0);
    }

    const RenderButtonZero = () => {
        if(num === 0)
            return <button disabled onClick={handleZero}>Clique aqui</button>

        return <button onClick={handleZero}>Clique aqui</button>
    }

    const RenderButtonFive = () => {
        if(num === 5)
            return <button disabled onClick={handleFive}>Clique aqui</button>

        return <button onClick={handleFive}>Clique aqui</button>
    }

    return (
        <>
            <RenderButtonZero />
            {num}
            <RenderButtonFive />
        </>
    )
}
```

Também podemos criar na nossa renderização de lista, ou seja, se temos um array de elementos, podemos renderizar dinamicamente da mesma forma

```
import { useState } from 'react';

function App() {
```

```
  var [pessoas, setPessoas] = useState([
    {
      id: 0,
      name: 'Alisson'
    },
    {
      id:1,
      name:'João'
    },
    {
      id: 2,
      name: "Queila"
    }
  ]);

  const RenderPessoas = () => {
    return pessoas.map(pessoa => {
      return(
        <tr id={pessoa.id}>
          <td>{pessoa.id}</td>
          <td>{pessoa.name}</td>
        </tr>
      )
    })
  }

  return (
    <table>
      <tr>
        <th>id</th>
        <th>Nome</th>
      </tr>
      <RenderPessoas />
    </table>
  );
}

export default App;
```

# 2 Styles com Sass

Os componentes podem ser reaproveitados, com isso, podemos ter interferência entre o estilo com css comum, pois ele utiliza somente o mesmo nome da classe para aplicar o estilo.
Pensando nisso, surge o Sass, que uma das funções adiciona identificadores únicos a cada estilo.
Para utilização é simples, basta instalar no projeto com

```
npm install sass
```

Após isso criamos um arquivo na mesma pasta do componente, e importamos no componente com

```
import styles from './styles.module.scss';
```

Após isso onde nós usávamos o className com uma string, agora com sass o className passa a receber um objeto, portanto para usar

```
import { useState } from 'react';
import styles from './styles.module.scss';

function App() {
  var [nome, setNome] = useState(0);
  var [idade, setIdade] = useState(0);

  return (
    <div className={styles.container}>
      <form className={styles.form}>
        <div className={styles.form__card}>
          <input
            className={styles.form__input}
            type='text'
            value={nome}
            onChange={(e) => setNome(e.target.value)}
          />
          <input
            className={styles.form__input}
            type='number'
```

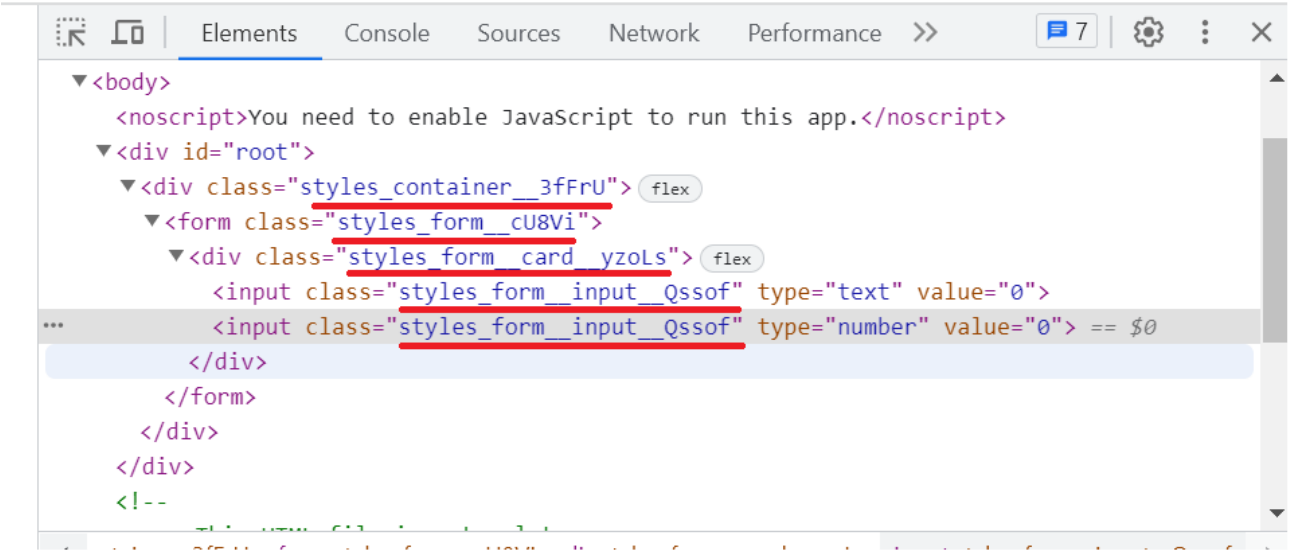
```
            value={idade}
            onChange={(e) => setIdade(Number(e.target.value))}
          />
        </div>
      </form>
    </div>
  );
}

export default App;
```

E no nosso arquivo styles.module.scss criamos o as classes semelhantes ao css convencional

```
.container{
    background-color: white;
    display: flex;
    flex-direction: column;
    margin: 0;
    padding: 0;
}

.form{
    height: 100vh;

    &__card{
        background-color: white;
        display: flex;
        flex-direction: column;
        width: 30%;
        margin: auto;
        padding: 2rem 5rem;
        box-shadow: 0px 0px 5px rgb(225, 225, 225);
        border-radius: 0.5rem;
        position: relative;
        top: 50%;
    }
```

```
    &__input{
        border-radius: 5px;
        border: 1px black
    }
}
```

Observe que ao inspecionar elemento, o class da tag input é um pouco diferente do que foi passado

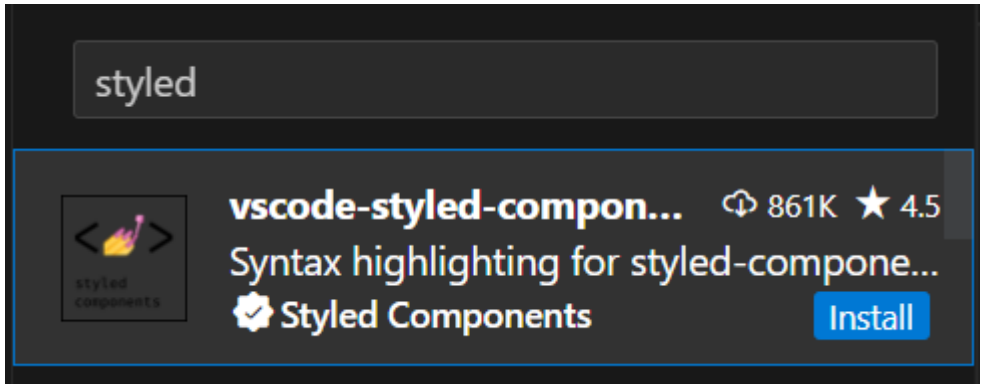



## 2.1 Criando styled components

Styled Components é uma biblioteca popular para estilização de componentes em aplicações React usando JavaScript e CSS-in-JS (CSS no JavaScript). Ela permite que você escreva estilos diretamente no seu código JavaScript, de forma que cada componente tenha seus próprios estilos encapsulados e reutilizáveis.
A ideia central por trás do Styled Components é eliminar a separação entre o JavaScript e o CSS, permitindo que você crie estilos diretamente nos seus componentes React, mantendo a lógica de estilização próxima à lógica do componente.
Para instalar o styled components é bem simples, vamos utilizar o código

```
npm install styled-components
```

e vamos instalar uma extensão para o VSCode chamada vscode-styled-components conforme a imagem abaixo




Agora sim podemos criar nossos componentes com styled,
Vamos criar um arquivo em *src* chamado **styled.js**
Agora importamos o styled e exportamos uma constante.

```
import styled from 'styled-components';

export const Header = styled.div`
    display: flex;
    justify-content: space-around;
    background-color: #555;
`;

export const Item = styled.div`
    color:white;
    font-size: 20px;
    font-weight: bold;
`;
```

E no nosso arquivo **App.js**

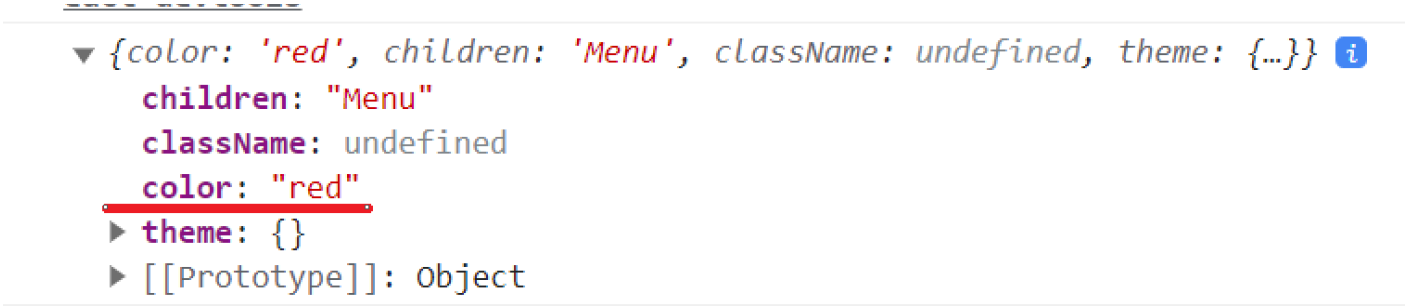
```
import './App.css';
```

```
import { Header, Item } from './styled';

function App() {
  return (
    <>
      <Header>
        <Item>Menu</Item>
        <Item>Home</Item>
        <Item>Sair</Item>
      </Header>
    </>
  );
}
```

Observe que é importado como um componente normal, só que podemos alterar o estilo como um arquivo CSS dentro do arquivo JS.
Podemos passar também props para os componentes.

```
import './App.css';
import { Header, Item } from './styled';

function App() {
  return (
    <>
      <Header>
        <Item color='red'>Menu</Item>
        <Item>Home</Item>
        <Item>Sair</Item>
      </Header>
    </>
  );
}
```

Só com essa alteração não modificou em nada o front-end, mas vamos verificar o que ocorreu.




Agora tem uma propriedade no objeto, com o mesmo nome e o mesmo valor que nós passamos, isso significa que podemos passar informações dessa forma para o nosso style.

Por fim vamos colocar cada cor com uma color diferente recebendo das props;
No arquivo **App.js** vamos adicionar cores nos outros itens.

```
function App() {
  return (
    <>
      <Header>
        <Item color='red'>Menu</Item>
        <Item color='yellow'>Home</Item>
        <Item color='blue'>Sair</Item>
      </Header>
    </>
  );
}
```

e no arquivo **styled.js** vamos receber essa cor.

```
export const Item = styled.div`
    color: ${props=> props.color};
    font-size: 20px;
    font-weight: bold;
`;
```

## 2.2 Criando os próprios Hooks

Podemos criar também Hooks próprios e personalizados, para isso, vamos abstrair a lógica para reaproveitar em mais locais
Vamos criar um arquivo useCounter

```
import { useState } from 'react';

const useCounter = (initialValue, step) => {
  const [count, setCount] = useState(Number(initialValue));

  const increment = () => {
    setCount(prevCount => prevCount + step);
  };

  const decrement = () => {
    setCount(prevCount => prevCount - step);
  };

  return { count, increment, decrement };
};

export default useCounter;
```

E agora para utilizar

```
import useCounter from './useCounter';

function App() {
  const {count, increment, decrement} = useCounter(0, 5)

  return (
    <div>
      <button onClick={decrement}>-</button>
      <p>Count: {count}</p>
      <button onClick={increment}>+</button>
    </div>
```

```
  );
}

export default App;
```

## 2.3 Desafio: Faça uma lista de tarefas, onde podemos adicionar e excluir tarefas