# Shopping List App

# Shopping List App


Um carrinho de compras é formado por vários itens.

Para isso, iremos criar 2 componentes React.

ShoppingListCart e ShoppingListItem.

## Lista de Compras

- Ovo - 12 unidades
- ~~Peito de Frango - 3 unidades~~
- ~~Manteiga - 1 unidades~~
- Iogurte - 2 unidades
- Farinha d'agua - 1 unidades

# Shopping List App: ShoppingListCart

```
import ShoppingListItem from "./ShoppingListItem";

function ShoppingListCart({shoppingList}) {
  return (
    <>
    <h1>Lista de Compras </h1>
    <ul>
      {shoppingList.map( (i) => (

        <ShoppingListItem id={i.id} name={i.name}
            quantity={i.quantity} isCompleted ={i.isCompleted} />

      ) )}
    </ul>
    </>
  );
}

export default ShoppingListCart;
```

# Shopping List App: ShoppingListItem

```
import "./ShoppingListItem.css"

function ShoppingListItem ({id, name, quantity, isCompleted}){

  let className = isCompleted ? "completed" : "incomplete";

  return (
        <li class={className}>
              {name} - {quantity} unidades
        </li>
      );
}

export default ShoppingListItem;
```

# Shopping List App: ShoppingListItem

- Separamos o CSS para um arquivo e atribuímos as classes CSS de forma dinâmica com o JS.

```
.completed{
    color: grey;
    text-decoration: line-through
}

.incomplete{
    color: red;
    text-decoration: none
}
```

# Shopping List App

- No App invocamos o nosso ShoppingListCart passando um array de itens.

```
const shoppingList = [
  {id: 1, name: "Ovo", quantity: 12, isCompleted: false},
  {id: 2, name: "Peito de Frango", quantity: 3, isCompleted: true},
  {id: 3, name: "Manteiga", quantity: 1, isCompleted: true},
  {id: 4, name: "Iogurte", quantity: 2, isCompleted: false},
  {id: 5, name: "Farinha d'agua", quantity: 1, isCompleted: false}
];

return (
  <>

    <ShoppingListCart shoppingList={shoppingList}  />

  </>
```

# AULA 4/6

# React Events

# React Events

- Similar aos eventos do JS comum (atenção a nova nomenclatura - CamelCase)
  - onclick  onClick
  - dblclick  onDoubleClick
  - onMouseOver
  - etc
- Atenção para não referenciar com os parênteses, isso fará executar a função e não criar uma referência.
  - onClick()

```
<div onClick={nomeFuncao}>
```

```
<div onClick={nomeFuncao()}>
```

# React Events

```
import "./ShoppingListItem.css"

//Pode ser qualquer nome!
function onClickEventHandler(){
    console.log("onClickEventHandler");
}

function onDoubleClickEventHandler(){
  console.log("onDoubleClickEventHandler");
}

function ShoppingListItem ({id, name, quantity, isCompleted}){
  let className = isCompleted ? "completed" : "incomplete";
  return (
        <li class={className} onClick={onClickEventHandler}
onDoubleClick={onDoubleClickEventHandler}>
          {name} - {quantity} unidades
        </li>
      );
}

export default ShoppingListItem;
```

# States

# React States

- Ao alterar os valores de variáveis JS, a interface do usuário não é atualizada de forma automática.
  - Por isso, é preciso informar ao React que o componente deve ser atualizado.
  - Por exemplo, ao clicar no botão e atualizar o valor de uma variável dentro do componente, esse valor não será atualizado na interface do usuário, mas estará atualizado no console.log().

# Counter App sem State

```
//This example will not work as expected! :-(

let num = 0;

function increaseNumber(){
    num += 1;
    console.log(num)
}

export default function CounterStateExample(){

    return (
        <div>
            <h2>Counter: {num}</h2>
            <button onClick={increaseNumber}>Click me</button>
        </div>
    );
}
```

# Counter App sem State

# Counter App COM State

- Agora precisamos deixar o React ciente de que atualizamos um componente para que ele possa renderizar a visualização do usuário, atualizando a tela.

# Counter App COM State

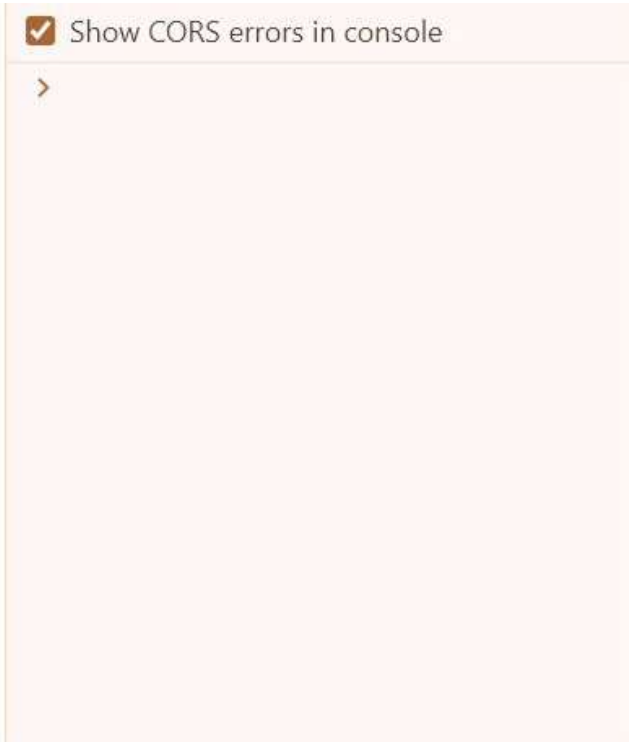
```
//Importar useState Hook (faz parte do React) para trabalhar com os States
import {useState} from "react";

export default function CounterStateExample2(){
//Convenção do React, receber o valor (num) e a função de alterar do valor (setNum)
    const [num, setNum] = useState(0) //valor inicial

//Também precisamos trazer a função para dentro do escopo desta outra função para
poder chamar o setNum().
    function increaseNumber(){
//Ao invés de incrementar a variável, pedimos a função para definir um novo valor.
        setNum(num + 1);
        console.log(num)
    }

    return (
        <div>
            <h2>Counter: {num}</h2>
            <button onClick={increaseNumber}>Click me</button>
        </div>
    );
}
```

# Counter App COM State

# Happy / Sad Face App

- Neste exemplo vamos criar um App React que mostra 2 estados de um componente, podendo exibir uma carinha feliz ou triste.

# Happy / Sad Face App

```
import { useState } from "react";
import "./Toggler.css"

export default function Toggler(){
    const [isHappy, setIsHappy] = useState(true);

    //arrow function
    //Independente do valor de isHappy, retorna a negação
    (true -> false | false -> true).
    const changeHumor = () => setIsHappy(!isHappy);

    return(
        <div>
            <p className="Toggler" onClick={changeHumor}>
                {isHappy ? "😊" : "😢"}
            </p>
        </div>
    )

}
```

# Happy / Sad Face App

```
.Toggler{
    font-size: 15rem;
}
```

```
App
...

  return (
    <>

      <Toggler />

    </>
```

# Happy / Sad Face Multi-States App

- Se precisar ter mais de um estado na sua aplicação, é só chamar o useState quantas vezes precisar.

- Para exemplificar, vamos incrementar o Happy / Sad App para incluir um contador.

- Os States não precisam estar associados, ao alterar um, não precisa alterar o outro!

# Happy / Sad Face Multi-States App

Counter: 0

Click-me

# Happy / Sad Face Multi-States App

```
import { useState } from "react";
import "./Toggler.css"

export default function TogglerMultiState(){
    const [isHappy, setIsHappy] = useState(true);
    const [counter, setCounter] = useState(0);

    const changeHumor = () => {
        setIsHappy(!isHappy);
        setCounter(counter+1);
     }

    return(
        <div>
            <p className="TogglerMultiState">
                {isHappy ? "😊" : "😢"}
            </p>
            <p className="Counter">Counter: {counter}</p>
            <button className="Counter" onClick={changeHumor}>Click-me</button>
        </div>
    )

}
```