INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

TECNÓLOGO EM ANÁLISE DE DESENVOLVIMENTO
Disciplina de Desenvolvimento Web II
Aula 04: React JS - Hooks / Styled Components / Router

Gil Eduardo de Andrade

Conceitos Preliminares

(https://react.dev/)
(https://pt-br.legacy.reactjs.org/docs/hooks-overview.html)
(https://styled-components.com/)
(https://playcode.io/react) (https://codesandbox.io/p/sandbox/react-new)

Hooks - Definição (https://pt-br.legacy.reactjs.org/docs/hooks-overview.html)
Os Hooks são recursos introduzidos no React, a partir da versão 16.8, que permitem a utilização de estados e outras funcionalidades do React em componentes funcionais. Anteriormente aos Hooks, tais funcionalidades ficavam restritas a componentes de classe. Neste contexto, através da utilização dos Hooks, é possível criar componentes funcionais mais concisos e reutilizáveis, sem a necessidade da escrita das classes.

Por que utilizar Hooks?

- **Componentes mais simples:** tornam os componentes funcionais mais fáceis de entender e manter, eliminando a necessidade de lidar com o ciclo de vida das classes;
- **Reutilização da lógica:** possibilita que o desenvolvedor crie seus próprios Hooks para encapsular lógicas comuns, como o gerenciamento de *fetches, por exemplo,* tornando seu código mais modular.
- **Componentes menores e mais focados:** dividindo a lógica em Hooks, torna-se possível criar componentes mais específicos e com responsabilidades bem definidas.

Principais Hooks:

- **useState:** permite adicionar estado a um componente funcional, retornando um array com duas partes: o valor atual do estado e uma função para atualizar esse valor.
- **useEffect:** usado para executar efeitos colaterais em componentes funcionais, como fazer *fetches* ou manipular o DOM.

- **useContext:** permite acessar o contexto de um componente, uma forma de compartilhar dados entre componentes sem passar props explicitamente.

Outros Hooks:

- **useReducer:** alternativa ao ***useState*** para gerenciar estados mais complexos, especialmente quando você precisa de lógica de atualização mais sofisticada.
- **useMemo:** permite memorizar o resultado de um cálculo custoso, evitando recalcular o valor a cada renderização quando as dependências não mudam.
- **useCallback:** permite memorizar uma callback, evitando a criação de novas funções a cada renderização, o que pode otimizar a performance.
- **useRef:** permite criar uma referência a um elemento DOM ou a um valor que persiste entre as renderizações.

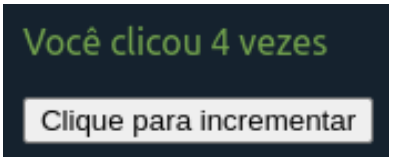
Exemplo Básico

Arquivo: "App.jsx" - React Playground

JavaScript

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

function Counter() {
  // Define o estado "count" e a função que permite
  // modificá-lo "setCount" (Hooks)
  const [count, setCount] = useState(0);
  // Função invocada, como efeito colateral, quando um estado
  // do componente é modificado (Hooks)
  useEffect(() => {
    document.title = `Você clicou ${count} vezes`;
  });

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Clique para incrementar
      </button>
    </div>
```

```
    );
}
```



Styled Components (https://styled-components.com/)
O Styled Components é uma biblioteca popular para React que permite você escrever CSS diretamente dentro dos seus componentes JavaScript. Essa abordagem, conhecida como CSS-in-JS, oferece uma maneira mais intuitiva e eficiente para estilização das aplicações React, promovendo a reutilização de estilos e a organização do código.

Por que usar Styled Components?

- **CSS dentro do JavaScript:** permite a escrita direta dos estilos junto ao JavaScript e a lógica do componente, tornando o código mais coeso e fácil de entender.
- **Componentes estilizados:** permite a criação de componentes estilizados que podem ser reutilizados em toda aplicação, gerando consistência visual.
- **Estilos temáticos:** permite criar temas para personalizar a aparência da sua aplicação de forma fácil e rápida.
- **Nenhum conflito de classes:** possibilita criar estilos específicos para cada componente, evitando conflitos com outros estilos da aplicação.

Instalação
(no terminal de comando)

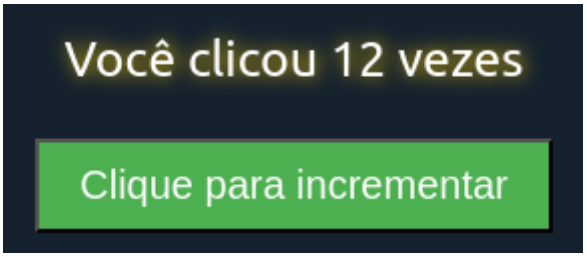
```
None
npm install styled-components
```

Importação e Utilização - Styled Componentes

Arquivo: “App.jsx” - React Playground

JavaScript

```
import { useState, useEffect } from 'react';
import styled from 'styled-components'; // Importação

// Definição dos Estilos CSS
const Button = styled.button`
    background-color: #4CAF50;
    color: white;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    font-size: 20px;
    cursor: pointer;
`;

const Counter = styled.p`
  color: #FFF;
  text-align: center;
  font-size: 25px;
  text-shadow: #FC0 1px 0 10px;
`;

...

return (
    <div className='App'>
      // Utilização dos Estilos Definidos
      <Counter>Você clicou {count} vezes</Counter>
      <Button onClick={() => setCount(count + 1)}>
        Clique para incrementar
      </Button>
    </div>
  );
}
```



Passando "*props*"
A biblioteca Styled Components permite a passagem de props para os componentes estilizados, possibilitando a criação de estilos dinâmicos.

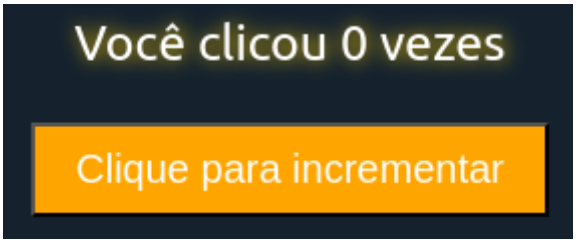
Utilização - "*props*" com Styled Componentes

Arquivo: "App.jsx" - React Playground

```javascript
import { useState, useEffect } from 'react';
import styled from 'styled-components'; // Importação

const Button = styled.button`
    background-color: ${(props) => props.color || '#4CAF50'};
    ...
`;

...

return (
    <div className='App'>
      <Counter>Você clicou {count} vezes</Counter>
      <Button color="orange" onClick={() => setCount(count + 1)}>
        Clique para incrementar
      </Button>
    </div>
  );
}
```

INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação



Boas Práticas - Styled Componentes
(Separando Estilo e Lógica)

Criando o Arquivo: "styled.jsx" - React Playground

JavaScript

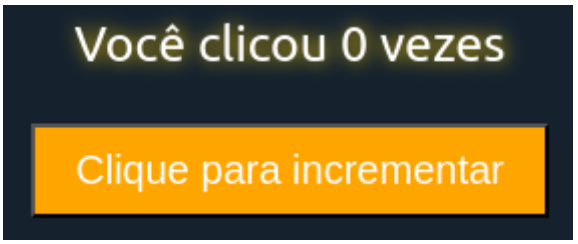
```
import styled from 'styled-components'; // Importação

export const Button = styled.button`
    background-color: ${props => props.color || '#4CAF50'};
    color: white;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    font-size: 20px;
    cursor: pointer;
`;

export const Counter = styled.p`
  color: #FFF;
  text-align: center;
  font-size: 25px;
  text-shadow: #FC0 1px 0 10px;
`;
```

Arquivo: "App.jsx" - React Playground

JavaScript

```
import { useState, useEffect } from 'react';
import { Button, Counter } from './styled';
```

```
...

return (
    <div className='App'>
      <Counter>Você clicou {count} vezes</Counter>
      <Button color='orange' onClick={() => setCount(count +
1)}>
        Clique para incrementar
      </Button>
    </div>
  );
}
```



React Router

O React Router é uma biblioteca que permite criar rotas em aplicações React, ou seja, define quais páginas / componentes serão renderizados de acordo com a URL que o usuário digita no navegador. Tal funcionalidade é fundamental para criação de aplicações single-page (SPA) - aplicações web em que o usuário interage com uma única página, onde seu conteúdo é dinamicamente alterado para que as informações exibidas sejam atualizadas.

A seguir é apresentado um exemplo de uso do React-router-DOM, onde um arquivo "router.js" é criado para que todas as rotas da aplicação possam ser especificadas. O padrão adotado busca adoção de boas práticas de programação, possibilitando que a definição de todas as rotas estejam centralizadas num arquivo específico.

Instalação

(no terminal de comando)

```
None
npm install react-router-dom
```

Utilização
(aplicação: https://github.com/Instituto-Federal-PR/Web-EMI/tree/master/Front/react/03%20-%20Axios/app-example)

Arquivo: "./routes.jsx"

```javascript
import { createBrowserRouter, } from "react-router-dom";
// Importa os componentes "Pages" renderizados de acordo com a rota
import Login from './pages/login';
import Home from './pages/home';
import Police from "./pages/police";
import ResetPass from './pages/resetpass'
// Define o Array de Objetos que contém as rotas da aplicação
const router = createBrowserRouter([
    {
        path: "/",            // Rota
        element: <Login />    // Componente/Page que será Invocado
    },

    {
        path: "/home",        // Rota
        element: <Home />     // Componente/Page que será Invocado
    },

    {
        path: "/police",
        element: <Police />
    },

    {
        path: "/reset",
        element: <ResetPass />
    },
])

export default router
```

O ***createBrowserRouter*** é uma função introduzida no ***React Router v6.4***, para substituir o antigo ***BrowserRouter***, como a abordagem recomendada para roteamento em aplicações web. Ele permite definir as rotas como um array de objetos, oferecendo mais flexibilidade e facilitando a utilização das novas APIs de

dados do React Router. O ***createBrowserRouter*** define as rotas da aplicação, especificando qual componente será renderizado para cada uma das URLs.

Arquivo: "./index.jsx"

```JavaScript
import router from './routes';
import { RouterProvider } from "react-router-dom";

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <React.StrictMode>
      <RouterProvider router={router} />
  </React.StrictMode>
);
```

Em aplicações React, o ***RouterProvider*** é um componente fornecido pela biblioteca React Router DOM, ele habilita o roteamento dentro da aplicação. O ***RouterProvider*** atua como um provedor, disponibilizando o contexto de roteamento para toda a árvore de componentes abaixo dele ou seja, permite que os componentes acessem informações sobre a rota atual e naveguem entre as diferentes rotas especificadas. Em outras palavras, o ***RouterProvider*** é o ponto de entrada para o roteamento em aplicações React.

Arquivo: "./src/components/formlogin/index.js"

```JavaScript
import { useNavigate } from 'react-router'
...
export default function FormLogin() {

   const navigate = useNavigate();

   function Authenticate() {

      const user = {
          name: 'Gil Eduardo',
          email: email
      }
      navigate('/home', { state: { user: user }})
   }
```

```
    ...
    <Submit value="Autenticar" onClick={() => Authenticate() }/>
    <LinkForgot onClick={() => navigate('/reset')}>
       Esqueceu sua senha?
    </LinkForgot>
    ...
}
```

React Bootstrap

O React Bootstrap é uma biblioteca que combina a estrutura visual do Bootstrap com a biblioteca de interface do React, permitindo que os desenvolvedores usem componentes Bootstrap dentro do React. Ele substitui a necessidade de uso de outras bibliotecas, como o jQuery, ao mesmo tempo em que garante a compatibilidade com a arquitetura de componentes proposta pelo React. Em outras palavras, o React Bootstrap reescreve os componentes Bootstrap como componentes React, removendo a dependência do JavaScript do Bootstrap.

Instalação

(no terminal de comando)

```
None
npm install react-bootstrap bootstrap
```

Importação

Arquivo: "./index.jsx"

```JavaScript
...
import 'bootstrap/dist/css/bootstrap.min.css'
...
```

Utilização

Arquivo: "./src/components/datatable/index.jsx"

```JavaScript
// Importa os componentes Bootstrap
import Table from 'react-bootstrap/Table';
import Button from 'react-bootstrap/Button';
...
export default function DataTable(props) {
    ...
    // Utiliza o componente Bootstrap -> <Table>
    <Table striped hover>
      ...
      // Utiliza o componente Bootstrap -> <Button>
      <Button variant="success" className="me-1">
            ...
      </Button>
      <Button variant="info" className="me-1">
            ...
      </Button>
      <Button variant="danger" className="me-1">
            ...
      </Button>
      ...
    </Table>
...
}
```

DATA	REGIONAL	POLICIAL	AÇÕES
10-01-2022	Diretivas de Segurança	Janilson Santos Júnior	
10-01-2022	Diretivas de Segurança	Janilson Santos Júnior	

HOOKS PRINCIPAIS

Hook "*useState*" + <input>

Arquivo: "App.jsx" - React Playground

```javascript
import React, { useState } from 'react';

export function App(props) {
  // Inicializa o valor dos campos como string vazia
  const [usuario, setUsuario] = useState({
    nome: '',
    email: '',
  });

  const handleChange = event => {
    // Atualiza o valor do estado com o valor digitado no
input
    setUsuario({
      // Copia propriedades e valores do state 'usuario' autal
      // garantindo que todas as propriedades, que não estão
      // sendo explicitamente alteradas, sejam MANTIDAS no
      // novo estado
      ...usuario,
      // 'event.target' - elemento DOM que disparou o evento
      // 'event.target.name': Obtém o valor do atributo 'name'
      // 'event.target.email': Obtém o valor do atributo 'email'
      [event.target.name]: event.target.value,
    });
  };

  return (
    <div className='App'>
      <input
        type='text'
        name='nome'
```

```
        value={usuario.nome}
        onChange={handleChange}
        placeholder='Digite seu nome'
      />
      <input
        type='email'
        name='email'
        value={usuario.email}
        onChange={handleChange}
        placeholder='Digite seu email'
      />
      <p>Seu nome é: "{usuario.nome}"</p>
      <p>Seu email é: "{usuario.email}"</p>
    </div>
  );
}
```

Hook "*useEffect*"

```JavaScript
useEffect(() => {
    // Códificação executada
  },
  [] // array de dependências
);
```

O Hook "*useEffect*" possui algumas funcionalidades específicas, considerando a forma como é utilizado (configurado) dentro da aplicação React.

- Array de dependências: utilizado para definir quando o ***useEffect*** deve ser executado. Ao passar um array vazio, indica-se que ***useEffect*** será executado apenas uma única vez, quando o componente é montado (*componentDidMount*).
- Função de limpeza: A função retornada pelo ***useEffect*** é executada quando o componente é desmontado (*componentWillUnmount*), sendo utilizada, normalmente, para limpar recursos como timers, event listeners, etc.

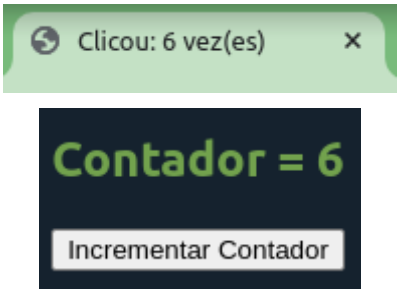
Arquivo: "App.jsx" - React Playground

JavaScript

```
import React, { useEffect, useState } from 'react';

export function App(props) {
  const [width, setWidth] = useState(window.innerWidth);

  useEffect(() => {
      window.addEventListener('resize', () => {
        setWidth(window.innerWidth);
      });

      return () => {
        window.removeEventListener('resize', handleResize);
      };
    },
    [] /*useEffect é executado apenas uma vez*/
  );

  return (
    <div className='App'>
      <h2>Largura da janela: {width}</h2>
    </div>
  );
}
```

Arquivo: "App.jsx" - React Playground

```javascript
import React, { useEffect, useState }  from 'react';

export function App(props) {

  const [count, setCount] = useState(0);

  // Só será reexecutado se o valor de 'count' mudar.
  useEffect(() => {
    document.title = `Clicou: ${count} vez(es)`;
  }, [count]);

  return (
    <div className='App'>
      <h2>Contador = {count}</h2>
      <button onClick={() => setCount(count + 1)}>Incrementar
Contador</button>
    </div>
  );
}
```



Hook "*useContext*"

No exemplo a seguir, para contextualizar, de maneira simplista, o Hook "***useContext***", considere uma aplicação que apresenta o nome de um usuário autenticado. Seu nome será compartilhado entre vários componentes, dentre os quais criaremos um <Header /> e um <Profile />.

Criando Novo Contexto

- ***UserContext*:** cria um novo contexto que será compartilhado.

Arquivo: "context.jsx" - React Playground

```javascript
import React, { createContext } from 'react';

// Primeira etapa para criar um novo "Contexto". Retorna
// um objeto de Contexto, composto por dois componentes
// principais. (a) Provider: envolve os componenentes que
// desejamos dar acesso aos dados do Contexto. Ele aceita
// uma prop value, que é o dado que você deseja compartilhar.
const UserContext = createContext();

export { UserContext };
```

Criando um Fornecedor de Contexto (Provider)

O ***UserProvider*** envolve toda a aplicação, fornecendo os dados para todos os componentes que estiverem dentro dele. Neste caso, estamos usando um estado para armazenar o nome do usuário e fornecê-lo como valor para o contexto.

- ***UserContext.Provider***: envolve a aplicação inteira para fornecer o valor do contexto para todos os componentes.
- ***value***: Objeto que contém os dados a serem compartilhados.

Arquivo: "provider.jsx" - React Playground

```javascript
import React, { useState } from 'react';
import { UserContext } from './context';

export default function UserProvider({ children }) {

  const [user, setUser] = useState({ name: 'Visitante' });

  return (
    // Componente provedor, a partir do UserContext criado,
    // usado para "envelopar" a parte da aplicação que precisa
```

```
    // ter acesso aos dados do contexto.
    <UserContext.Provider value={user}>
      // Componentes que são evelopados e acessam o dado do
      // contexto.
       {children}
    </UserContext.Provider>
  );
}
```

Criando os Componentes que Consomem o Contexto

Arquivo: "header.jsx" - React Playground

JavaScript

```
import React, { useContext } from 'react';
import { UserContext } from './context';

export default function Header() {
  // Obtém o state "name" do contexto criado (UserProvider)
  const { name } = useContext(UserContext);

  return (
    <h1>Olá, {name}!</h1>
  );
}
```

Arquivo: "profile.jsx" - React Playground

JavaScript

```
import React, { useContext } from 'react';
import { UserContext } from './context';

export default function Profile() {
  // Obtém o state "name" do contexto criado (UserProvider)
  const { name } = useContext(UserContext);
```

```
  return (
    <div>Seu nome é {name}</div>
  );
}
```

Utilizando a Codificação Criada

Arquivo: "App.jsx" - React Playground

JavaScript

```
import React from 'react';
import UserProvider from './provider';
import Header from './header';
import Profile from './profile';

export function App(props) {
  return (
    <div className = "App">
      // Define o componente Provider, para o contexto criado
      // (UserContext), que envelopa a parte da aplicação que
     // precisa ter acesso aos dados.
      <UserProvider>
        // Componentes que são evelopados e acessam o dado do
         // contexto.
        <Header />
        <Profile />
      </UserProvider>
    </div>
  );
}
```

OUTROS HOOKS

Hook "*useReducer*"
O **"*useReducer*"** é uma ferramenta poderosa no React para gerenciar estados mais complexos ou que envolvem múltiplas atualizações. Ele funciona de forma similar ao "***useState"***, mas oferece uma estrutura mais organizada e previsível para lidar com mudanças de estado.

Quando utilizá-lo?

- ***Estados complexos***: quando seu estado envolve múltiplas propriedades interdependentes ou quando você precisa realizar várias atualizações em resposta a uma única ação.
- ***Histórico de estados***: se você precisar rastrear o histórico de mudanças de estado para implementar funcionalidades como "*desfazer*" e "*refazer*".
- ***Lógica de atualização complexa***: quando a lógica para atualizar o estado é complexa e envolve várias condições e cálculos.

No exemplo a seguir, para contextualizar, de maneira simplista, o Hook "***useReducer***", considere uma aplicação que mostra um contador e permite incrementá-lo ou decrementá-lo.

- ***Estado inicial***: o estado inicial do contador foi definido como um objeto, contendo a propriedade "*count"* inicializada com valor "*0*".
- ***Função "reducer"***: responsável por atualizar o estado do contador, ela recebe, como parâmetros, o estado atual e uma ação ('*increment*' ou '*decrement*'), e retorna o novo estado.
- ***"useReducer"***: o hook "*useReducer*" recebe, como argumentos, a função reducer e o estado inicial, retornando um array com dois elementos:
  - *state*: valor atual do estado.
  - *dispatch*: função para disparar ações e atualizar o estado.
- ***Interface do usuário***: os botões de incremento e decremento chamam a função "*dispatch*" com uma ação específica, passada para a função "*reducer*", que calcula o novo estado.

Como funciona?

- Quando um botão é clicado, uma ação é disparada.
- A função "*reducer*" é chamada com o estado atual e a ação.

- A função “*reducer*” retorna o novo estado de acordo com a ação.
- O React “*re-renderiza*” o componente com o novo estado.

Arquivo: “App.jsx” - React Playground

JavaScript

```
import React, { useReducer } from 'react';

export function App(props) {
  // Estado inicial
  const initialState = { count: 0 };
  // Função reducer para atualizar o estado
  function reducer(state, action) {
    switch (action.type) {
      case 'increment':
        return { count: state.count + 1 };
      case 'decrement':
        return { count: state.count - 1 };
      default:
        throw new Error();
    }
  }
  // state: representa o estado atual do componente - pode
  // ser de qualquer tipo (objeto, número, string).
  // dispatch: função para "despachar" ações, recebe uma ação,
  // sinaliza o useReducer que o estado deve ser alterado.
  // reducer: função pura, define como o estado deve mudar em
  // resposta a diferentes ações, tem o padrão (state, action)
  // initialState: o valor inicial do estado, com o qual o
  // componente inicia quando é montado.
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div className='App'>
      <h1>Contador: {state.count}</h1>
      // Invoca a função 'reducer', definida pelo 'useReducer'. Passa como
      // parâmetro um objeto contendo a ação 'increment' ou 'decrement'.
      // Por padrão o objeto têm uma propriedade type que descreve o tipo
      // de operação que deve ser realizada no estado.
```

```
        <button onClick={() => dispatch({ type: 'increment' })}>+</button>
        <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
      </div>
    );
}
```

Hook "*useMemo*"
O hook "*useMemo*" é uma ferramenta útil para otimizar o desempenho das aplicações, especialmente quando elas possuem cálculos complexos ou funções que são executadas constantemente, mas cujo resultado não muda com tanta frequência.

Como funciona?

- ***Memoriza valores***: armazena em cache o resultado de uma função.
- ***Evita recalculation***: se os valores de entrada da função não mudarem, o "*useMemo"* retorna o valor armazenado em cache, evitando o recálculo da função a cada nova renderização.
- ***Melhora desempenho***: pode melhorar significativamente o desempenho da aplicação, especialmente em componentes que são renderizados com frequência.

No exemplo a seguir, para contextualizar, de maneira simplista, o Hook "***useMemo***", considere uma aplicação onde existe uma lista grande de produtos e haja a necessidade de filtrá-la com base em uma busca. Filtrar uma lista grande pode ser uma operação custosa, especialmente se for feita a cada renderização.

- ***useState***: utilizado para gerenciar o valor da busca.
- ***useMemo***: a função de filtragem é envolvida pelo "*useMemo*", onde o segundo argumento é um array de dependências. Isso significa que a função de filtragem só será recalculada se o valor de "*products*" ou "*searchTerm*" mudar.

- ***Retorno***: é a lista filtrada, que será renderizada na tela.

Arquivo: "products.jsx" - React Playground

JavaScript

```
import React, { useState, useMemo } from 'react';

export default function ProductsList({ products }) {

  const [searchTerm, setSearchTerm] = useState('');

  // Função para filtrar os produtos
  const filteredProducts = useMemo(() => {
    // Função que será executada novamente se algum dos
    // valores do array de dependências for alterado.
    return products.filter(product =>
      product.name.toLowerCase().includes(searchTerm.toLowerCase())
    );
  }, [products, searchTerm]);
  // Array de dependências/valores que o useMemo "monitora".
  return (
    <div>
      <input
        type="text"
        value={searchTerm}
        onChange={e => setSearchTerm(e.target.value)}
      />
      <ul>
        {filteredProducts.map(product => (
          <li key={product.id}>{product.name}</li>
        ))}
      </ul>
    </div>
  );
}
```

Arquivo: "App.jsx" - React Playground

JavaScript

```
import React from 'react';
import ProductsList from './products'

export function App(props) {

  const arr = [
      { id: 1, name: 'Geladeira' },
      { id: 2, name: 'Microondas' },
      { id: 3, name: 'Sofá' },
      { id: 4, name: 'Forno Elétrico' },
      { id: 5, name: 'Televisão' },
      { id: 6, name: 'Air Fryer' },
      { id: 7, name: 'Aparador' },
      { id: 8, name: 'Mesa' },
      { id: 9, name: 'Mesanino' },
  ];

  return (
    <div className='App'>
      <ProductsList products={arr}/>
    </div>
  );
}
```

Explicando o Funcionamento do Exemplo:

- Quando o componente é renderizado pela primeira vez, a função de filtragem é executada e o resultado é armazenado em cache.

- A cada renderização subsequente, o "*useMemo*" verifica se os valores de "*products*" ou "*searchTerm*" mudaram.
- Caso não mudem, o "*useMemo*" retorna o valor armazenado em cache, evitando a filtragem da lista novamente.
- Caso tenham mudado, a função de filtragem é executada novamente e o novo resultado é armazenado em cache.

Hook "*useCallback*"

O hook "*useCallback*" é utilizado para otimizar o desempenho de componentes, especialmente quando passamos funções como "*props*" para componentes filhos. Ele memoriza uma função, evitando que ela seja recriada a cada renderização, a menos que suas dependências mudem.

Quando usar?

- ***Funções como props***: em situações onde uma função é passada como "*prop*" para um componente filho e essa função não precisa ser recriada a cada renderização.
- ***Callbacks em otimizações***: Para evitar "*re-renders*" desnecessários em componentes otimizados.

No exemplo a seguir, para contextualizar, de maneira simplista, o Hook "***useCallback***", considere que um componente pai passa uma função para um componente filho, e essa função incrementa um contador.

- ***useCallback***: a função "*increment*" é envolvida pelo "*useCallback*", onde o array de dependências [*count*] indica que a função só será recriada se o valor de "*count*" mudar.
- ***Passagem de props***: a função "*increment*" é passada como "*prop*" para o componente filho "*Child*".
- ***Otimização***: ao usar "*useCallback*", garantimos que a função "*increment*" seja a mesma a cada renderização, a menos que o valor de "*count*" mude. Isso evita que o componente filho seja "*re-renderizado*" desnecessariamente.

Arquivo: "child.jsx" - React Playground

JavaScript

```
import React from 'react';

export default function Child({ increment }) {
```

```
  return (
    <button onClick={increment}>Incrementar</button>
  );
}
```

Arquivo: "parent.jsx" - React Playground

JavaScript

```
import React, { useState, useCallback } from 'react';
import Child from './child';

export default function Parent() {

  const [count, setCount] = useState(0);

  // Função para incrementar o contador
  const increment = useCallback(() => {
    // Função de callback que você quer memorizar: setCount().
    setCount(count + 1);
  }, [count]);
  // [count]: Array de dependências, se 'count' permanecer o
  // mesmo, a função increment retornada será a mesma
  // instância da função da renderização anterior. Se 'count'
  // for alterado a função 'increment' é recriada.

  return (
    <div>
      <Child increment={increment} />
      <p>Contador: {count}</p>
    </div>
  );
}
```

Arquivo: "App.jsx" - React Playground

```javascript
import React from 'react';
import Parent from './parent';

export function App(props) {
  return (
    <div className='App'>
      <Parent />
    </div>
  );
}
```

Hook "*useRef*"
O *"useRef"* permite criar referências mutáveis que persistem entre as renderizações de um componente. Diferente do *"useState"*, que causa uma nova renderização sempre que seu valor é atualizado, o *"useRef"* não notifica o React sobre mudanças. O *"useRef"* é ideal em situações onde um valor, que não afeta diretamente a UI, precisa ser mantido, ou seja, estar acessível e ser modificável.

Como funciona?
Quando você chama *"useRef()"*, ele retorna um objeto *"ref"* com uma única propriedade mutável chamada *".current"*. O valor inicial que você passa para *"useRef"* é atribuído a .current.

Arquivo: "InputFocus.jsx" - React Playground

```javascript
import React, { useRef } from 'react';

function FocoNoInput() {
  // Cria um useRef - será anexado ao elemento <input>.
```

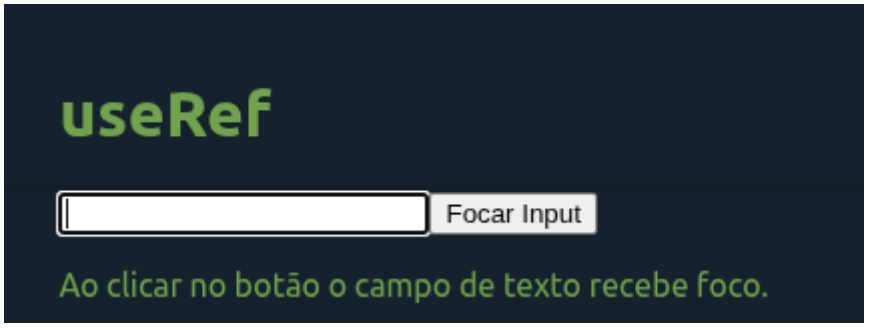
```
  const inputRef = useRef(null);

  // Gerencia o Clique no botão
  const handleClick = () => {
    // Acessa o elemento DOM real através de inputRef.current
    // e invoca o método focus() para ele.
    if (inputRef.current) {
      inputRef.current.focus();
    }
  };

  return (
    <div>
      <input
        type="text"
        // Anexa a ref ao elemento input usando a prop 'ref'.
        // Agora, inputRef.current aponta para o elemento DOM.
        ref={inputRef}
        placeholder="Clique no botão para me focar!"
      />
      <button onClick={handleClick}>
        Focar Input
      </button>
      <p>Ao clicar no botão o campo de texto recebe foco.</p>
    </div>
  );
}

export default FocoNoInput;
```

Arquivo: "App.jsx" - React Playground

JavaScript

```
import React from 'react';
import InputFocus from './InputFocus';
```

```
export function App(props) {
  return (
    <div className='App'>
      <h1>useRef</h1>
      <InputFocus />
    </div>
  );
}
```



Comparação entre *"useRef"* e *"useState"*

Característica	useRef	useState
Re-renderização	Não causa re-renderização ao mudar *".current"*	Causa re-renderização ao mudar o "estate"
Tipo de valor	Objeto ref com propriedade *".current"* mutável	Tupla *"[estate, setState]"*
Uso principal	Armazenar valores mutáveis não visíveis na UI, persistir valores sem re-renderizar	Gerenciar estado que afeta a renderização da UI

Criando Aplicação
(no terminal de comando)

```
None
npx create-react-app project-name
```

Executando Aplicação
(no terminal de comando)

```
None
npm run start
```