1. Atividades Práticas - Gerenciamento de Carrinho de Compras

Nesta atividade prática, implementaremos um sistema de carrinho de compras que permite adicionar, remover produtos ao carrinho através do hook **useReducer**.

1.1 Criando o projeto e iniciando o servidor local

Crie um projeto chamado **compras**, digitando o seguinte comando no terminal:

```
npm create vite@latest compras -- --template react
```

Este processo de configuração inicial do projeto leva alguns segundos. Ao terminar, o Vite repassa instruções para que você termine de instalar as dependências do seu projeto. Desta forma, digite o seguinte comando para **entrar no diretório** recém criado do nosso projeto:

```
cd compras
```

Em seguida, **instale as dependências** necessárias do projeto executando no terminal o seguinte comando:

```
npm install
```

Até aqui, você criou um projeto React usando o Vite e adicionou todas as dependências ao projeto. Você pode, agora, **abrir a aplicação no editor Visual Studio Code**. Para isso, dentro do diretório do projeto digite o seguinte comando no terminal:

```
code .
```

Após a execução deste comando, o Visual Studio Code deverá abrir com a pasta raiz do seu projeto sendo acessada. Em seguida, você inicializará um servidor local e executará o projeto em seu navegador. Digite a seguinte linha de comando no terminal:

```
npm run dev
```

Ao executar esse script, você iniciará um servidor local de desenvolvimento, executará o código do projeto, iniciará um observador que detecta alterações no código e abrirá o projeto em um navegador web. Ele irá rodar a aplicação em modo desenvolvimento em http://localhost:5173/.

1.2 Passo 1: Criar a estrutura do projeto

Inicialmente, vamos criar a estrutura do nosso projeto. Neste contexto, dentro do diretório **src**, crie os seguintes diretórios:

- **components:** Contém os componentes visuais e de interação, como o carrinho de compras e os itens dentro do carrinho.
- **reducers:** Contém a lógica do **useReducer**, separando a lógica de estado do componente.
- **actions:** Define as ações disponíveis que podem ser enviadas para o **reducer**. Isso torna o código mais organizado e facilita a adição de novas ações no futuro.

1.3 Passo 2: Criar o Componente para Renderizar Itens do Carrinho

Vamos criar um componente chamado **CartItem** que renderizará individualmente cada item no carrinho. Isso ajuda a separar a lógica de cada item. Dessa forma, dentro do diretório **components**, crie o arquivo **CartItem.jsx** com o seguinte código:

```
function CartItem({ item, removeItem }) {
  return (
    <li>
      {item.name} (Quantidade: {item.quantity}){" "}
      <button onClick={() => removeItem(item.name)}>Remover</button>
    </li>
  );
}
export default CartItem;
```

Observe que o componente **CartItem** recebe um **item** (com nome e quantidade) e a função **removeItem** como **props**. Isso mantém a responsabilidade de renderizar e remover um item fora do componente principal do carrinho de compras.

1.4 Passo 3: Criar a Lógica do useReducer

Aqui vamos isolar a lógica do **reducer** em um arquivo separado. Isso ajuda a manter o código relacionado ao estado mais modular e fácil de testar. Desta forma, dentro do diretório reducers, crie um arquivo chamado **cartReducer.jsx** com o seguinte código abaixo:

```
const initialState = [];

function cartReducer(state, action) {
  switch (action.type) {
    case "ADD_ITEM": {
      const itemIndex = state.findIndex(
        (item) => item.name === action.payload.name
      );
      if (itemIndex > -1) {
        const updatedState = [...state];
        updatedState[itemIndex].quantity += 1;
        return updatedState;
      }
      return [...state, { name: action.payload.name, quantity: 1 }];
    }
    case "REMOVE_ITEM":
      return state.filter((item) => item.name !== action.payload.name);
    case "CLEAR_CART":
      return [];
    default:
      return state;
  }
}
export { cartReducer, initialState };
```

Neste exemplo, observe que:

- **cartReducer:** A função **reducer** recebe o estado atual e uma ação, e retorna o novo estado com base no tipo da ação.
  - **ADD_ITEM:** Verifica se o item já está no carrinho e incrementa sua quantidade, ou adiciona um novo item.
  - **REMOVE_ITEM:** Remove um item do carrinho.
  - **CLEAR_CART:** Limpa completamente o carrinho.
- **initialState**: O estado inicial é uma lista vazia.

1.5 Passo 4: Definir as Ações

Agora podemos isolar as ações que são enviadas para o reducer, mantendo o código ainda mais organizado. Dentro do diretório **actions**, crie o arquivo **cartActions.jsx** que conterá funções que disparam as ações para o reducer.

```
export const addItemAction = (name) => ({
  type: "ADD_ITEM",
  payload: { name },
});
export const removeItemAction = (name) => ({
  type: "REMOVE_ITEM",
  payload: { name },
});
export const clearCartAction = () => ({
  type: "CLEAR_CART",
});
```

Neste exemplo, observe que estamos separando as ações que são enviadas ao **reducer**. Isso torna o código mais legível e facilita a adição de novas ações no futuro, além de manter o componente mais limpo.

1.6 Passo 5: Componente Principal do Carrinho

Vamos implementar o **useReducer** no componente que gerencia o carrinho de compras, permitindo que o usuário adicione, remova e limpe itens do carrinho. Desta forma, dentro do diretório **components**, crie o arquivo **Cart.jsx** com o seguinte código:

```
import { useReducer } from 'react';
import { cartReducer, initialState } from '../reducers/cartReducer';
import { addItemAction, removeItemAction, clearCartAction } from
'../actions/cartActions';
import CartItem from './CartItem';

function Cart() {
  const [cart, dispatch] = useReducer(cartReducer, initialState);
  const addItem = (name) => {
    dispatch(addItemAction(name));
  };
```

```
  const removeItem = (name) => {
    dispatch(removeItemAction(name));
  };
  const clearCart = () => {
    dispatch(clearCartAction());
  };
  return (
    <div>
      <h2>Carrinho de Compras</h2>
      <button onClick={() => addItem('Maçã')}>Adicionar Maçã</button>
      <button onClick={() => addItem('Banana')}>Adicionar Banana</button>
      <button onClick={() => addItem('Laranja')}>Adicionar Laranja</button>
      <ul>
        {cart.length > 0 ? (
          cart.map((item, index) => (
            <CartItem
              key={index}
              item={item}
              removeItem={removeItem}  // Passa a função para remover o item
            />
          ))
        ) : (
          <p>O carrinho está vazio.</p>
        )}
      </ul>

      {cart.length > 0 && <button onClick={clearCart}>Limpar Carrinho</button>}
    </div>
  );
}
export default Cart;
```

Neste exemplo, observe que:

- O **useReducer** é inicializado com o **cartReducer** e o estado inicial (**initialState**). O **useReducer** retorna o estado atual (**cart**) e a função **dispatch**, que é usada para enviar ações.
- A função **dispatch** é chamada para enviar ações ao reducer.
- O **Cart** utiliza o **CartItem** para exibir cada **item** no carrinho. O **map** itera sobre o estado do carrinho (**cart**) e cria um **CartItem** para cada produto.
- Cada **item** e a função **removeItem** são passados como **props** para o **CartItem**.

1.7 Passo 6: Uso no Componente Principal

Vamos integrar o componente **Cart** no componente principal **App** para que possamos usá-lo na aplicação. Desta forma, altere o arquivo **App.jsx** com o seguinte código:

```
import Cart from "./components/Cart";
function App() {
  return (
    <div>
      <h1>Loja Virtual</h1>
      <Cart />
    </div>
  );
}
export default App;
```

1.8 Passo 7: Testando a Aplicação

Agora, vamos testar nossa aplicação. Observe que ao clicar em "Adicionar Maçã", "Adicionar Banana" ou "Adicionar Laranja", o item correspondente será adicionado ao carrinho.

Figura 1: Exemplo da nossa aplicação adicionando 3 produtos

Já, ao clicar em "Remover" ao lado de um item, ele será removido do carrinho.

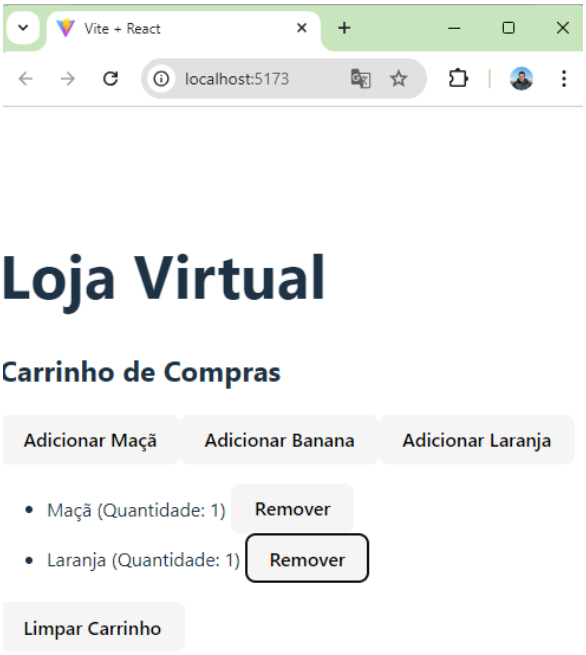

Figura 2: Exemplo da nossa aplicação removendo 1 produto

E, finalmente, ao clicar em "Limpar Carrinho", todos os itens serão removidos.

Figura 3: Exemplo da nossa aplicação limpando o carrinho

Para finalizar, clique em "Logout" para encerrar a sessão. Veja que o carrinho de compras está vazio.