Entrega Final

# Explicação do Código ReactJS

**Resalvas: Projeto não estar completo, a parte do projeto funcional é o botão [F9 NOVO] onde o Modal é aberto e assim podendo adicionar produtos a lista e sendo somado ao total que esta no componente Header.**

**ButtonFooter não implementado e nem a exclusão e edição dos itens na lista**

1. Estrutura Básica do Componente:

O componente principal `App` utiliza hooks (`useState` e `useEffect`) e divide sua interface em diversos componentes menores (`Header`, `ButtonRadius`, `CabecalhoItens`, `ItensPedido`, `Modal`, e `ButtonBottom`), promovendo a reutilização e modularidade. Essa estrutura é bastante comum em projetos React para manter o código mais organizado.

2. Gerenciamento de Estado:

- `orders`: Armazena os pedidos que são buscados de uma API externa. A função `setOrders` é usada para atualizar essa lista.

- `openModal`: Controla o estado de abertura do modal, permitindo que seja alternado entre aberto e fechado.

- `totalHeader`: Mantém o total acumulado dos produtos do primeiro pedido retornado. O uso do `setTotalHeader` permite atualizar esse valor dinamicamente.

3. useEffect:

O `useEffect` aqui é utilizado para buscar os dados da API assim que o componente é montado (no ciclo de vida inicial). É uma prática comum para fazer requisições ao carregar a página.

4. Função `fetchOrders`:

- Essa função utiliza `async/await` para fazer uma requisição à API (`http://localhost:3001/orders`), buscando os dados dos pedidos e armazenando-os em `orders`.

- Após a busca, a função calcula o total dos produtos do primeiro pedido utilizando `reduce`, que acumula o valor de `product.total`.

Sugestão de melhoria: A função `fetchOrders` assume que sempre haverá dados no primeiro índice de `data` (`data[0]`). Seria bom adicionar uma verificação de segurança para garantir que os dados existem antes de tentar acessá-los.

5. Renderização dos Pedidos:

O código mapeia os pedidos (`orders.map`) e, para cada pedido, mapeia os produtos associados (`order.products.map`), renderizando o componente `ItensPedido` para cada produto.

6. Modal:

- A exibição do componente `Modal` é controlada por `openModal`. Quando o modal está aberto, ele aparece na tela.

7. Componentes de Botões e Footer:

- Há dois tipos de botões sendo utilizados: `ButtonRadius` (na parte superior) e `ButtonBottom` (na parte inferior).

8. Função `handleOrderUpdate`:

- Esta função é responsável por atualizar os pedidos após uma modificação e fechar o modal.

Sugestões gerais:

- Tratamento de Erros: Adicionar mais tratamentos para quando a API não retorna os dados ou quando `orders` estiver vazio.

- Componentização: Mover a lógica da requisição e o estado para um contexto global, dependendo da necessidade.

- Acessibilidade: Certificar-se de que os botões possuem textos ou rótulos acessíveis para leitores de tela.

# Explicação do Componente Modal

1. Estrutura do Componente Modal:

O componente `Modal` utiliza hooks como `useState` e `useEffect`, e recebe dois props principais: `isOpen` e `onOrderUpdate`.

Ele é responsável por exibir uma lista de produtos disponíveis e permitir que o usuário adicione um produto à ordem ativa.

2. Gerenciamento de Estado:

- `modal`: Controla o estado de abertura/fechamento do modal, usando o valor passado pelo prop `isOpen`.

- `produto`: Armazena a lista de produtos que é buscada da API.

- `orders`: Armazena a ordem atual ('0001'), que é buscada de uma API e atualizada ao adicionar um novo produto.

- `newProduct`: Controla os campos de um novo produto, inicializando-os como um objeto vazio.

3. useEffect:

O `useEffect` é usado para buscar a lista de produtos e a ordem '0001' assim que o modal é aberto (montado). Essa abordagem garante que as informações estejam disponíveis assim que o modal for exibido.

4. Função `fetchProducts`:

- Responsável por buscar a lista de produtos de uma API (http://localhost:3001/products) e armazenar os dados no estado `produto`.

- O tratamento de erros está presente para lidar com falhas na requisição.

5. Função `fetchOrder`:

- Faz uma requisição para obter a ordem '0001' e a armazena no estado `orders`. Isso permite que o modal exiba a ordem corrente e modifique os produtos relacionados.

6. Função `addProductToOrder`:

- Ao clicar no botão `ADD`, essa função cria um objeto de produto com as informações relevantes (`id`, `name`, `un`, `price`).

- Em seguida, o produto é adicionado à lista de `products` da ordem, e uma requisição PUT é feita para atualizar a ordem no servidor.

Sugestão de Melhoria: Seria interessante adicionar validação antes de adicionar o produto à ordem, garantindo que o produto não esteja duplicado.

7. Renderização Condicional:

- O modal só é exibido quando o estado `modal` é `true`. Caso contrário, ele retorna `null`, o que evita a renderização desnecessária.

8. Componente de Fechamento do Modal:

A função `ClosedModal` é responsável por alternar o estado de `modal` e notificar o componente pai sobre a atualização da ordem por meio do prop `onOrderUpdate`.

9. Estilização:

- O componente utiliza a classe `containerModal` para a estrutura do modal e classes como `ulModal` e `ulModalItens` para formatar a exibição dos produtos dentro do modal.

Sugestões gerais:

- Adicionar um feedback visual ao usuário quando um produto é adicionado com sucesso.

- Verificar o tratamento de casos onde a API possa não retornar dados válidos para a ordem ou para os produtos.