

El estado del carrito se tiene que compartir entre las páginas pokemon list y shopping cart

eleva el estado del carrito a un nivel superior, es decir, mover el estado cart y las funciones relacionadas a un componente que se encuentre por encima de PokemonList y ShoppingCart, de tal forma que ambos componentes compartan el mismo estado.

Una manera común de lograr esto es utilizando un contexto de React (React Context). Aquí te dejo los pasos para implementar esto:

1. Crear un contexto para el carrito:

Crea un archivo llamado CartContext.js dentro de la carpeta componentes o context, donde definirás el contexto:

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

export const CartContext = createContext();

export const CartProvider = ({ children }) => {
  const [cart, setCart] = useState([]);

  const addToCart = (pokemon) => {
    setCart((prevCart) => {
      const existingProduct = prevCart.find((item) => item.id === pokemon.id);
      if (existingProduct) {
        return prevCart.map((item) =>
          item.id === pokemon.id ? { ...item, quantity: item.quantity + 1 } : item
        );
      } else {
        return [...prevCart, { ...pokemon, quantity: 1 }];
      }
    });
  };

  const removeFromCart = (indexToRemove) => {
    setCart(cart.filter((_, index) => index !== indexToRemove));
  };

  return (
    <CartContext.Provider value={{ cart, addToCart, removeFromCart }}>
      {children}
    </CartContext.Provider>
  );
};
```

2. Modificar el componente App.js para usar el contexto:

En App.js, envuelve tu aplicación con CartProvider:

```

import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Home from './pages/Home';
import About from './pages/About';
import PokemonList from './pages/PokemonList';
import NotFound from './pages/NotFound';
import Footer from './componentes/Footer';
import NavBar from './componentes/NavBar.js';
import NewUserForm from './pages/NewUserForm.js';
import Cabecera from './componentes/Cabecera.js';
import ShoppingCart from './pages/ShoppingCart';
import { CartProvider } from './componentes/CartContext';

function App() {
  return (

    <Route path="/" element={} />
    <Route path="/about" element={} />
    <Route path="/products" element={} />
    <Route path="/shoppingcart" element={} />
    <Route path="*" element={} />
    <Route path="/user" element={} />

    ); }

export default App;

```

3. Actualizar PokemonList.js y ShoppingCart.js para usar el contexto:

En PokemonList.js, usa el contexto para agregar productos al carrito:

```

import React, { useContext } from "react";
import { CartContext } from "../componentes/CartContext";
import "../App.css";
import Product from "../componentes/Product";
import { useEffect, useState } from "react";

export default function PokemonList() {
  const { addToCart } = useContext(CartContext);
  const [pokemons, setPokemons] = useState([]);

  useEffect(() => {
    fetch(https://pokeapi.co/api/v2/pokemon?limit=10)
      .then((response) => response.json())

```

```

.then((data) => {
const results = data.results;
const promises = results.map((pokemon) =>
fetch(pokemon.url)
.then((res) => res.json())
.then((resData) => ({
id: resData.id,
name: resData.name,
price: `${(resData.id * 10).toFixed(2)}`,
image: resData.sprites.front_default,
}))
);
Promise.all(promises).then((pokemonData) => {
setPokemons(pokemonData);
});
})
.catch((error) => console.error("Error fetching the pokemons: ", error));
}, []);

return (

```

Catálogo de Pokemon

```

{pokemons.map((pokemon) => ( ))}
); }; En ShoppingCart.js, usa el contexto para mostrar el contenido del carrito:

```

javascript

Copiar código

```

import React, { useContext } from "react";
import { CartContext } from "../componentes/CartContext";
import { FaShoppingCart, FaTrashAlt } from "react-icons/fa";
import CalculateTotal from "../componentes/CalculateTotal";

export default function ShoppingCart() {
const { cart, removeFromCart } = useContext(CartContext);

return (


{cart.map((item, index) => (


- {item.name} - {item.quantity} x {item.price} = {item.quantity * parseFloat(item.price.replace("$", ""))}



removeFromCart(index)> Eliminar


```

); } Con estos cambios, el estado del carrito se mantendrá cuando navegues entre las diferentes páginas, permitiendo que los productos seleccionados en la página de catálogo se muestren correctamente en la página del carrito.

Tú dijiste:

¿me puedes explicar este componente? import React, { createContext, useState } from 'react';

```
export const CartContext = createContext();
```

```
export const CartProvider = ({ children }) => {
```

```
  const [cart, setCart] = useState([]);
```

```
  const addToCart = (pokemon) => {
```

```
    setCart((prevCart) => {
```

```
      const existingProduct = prevCart.find((item) => item.id === pokemon.id);
```

```
      if (existingProduct) {
```

```
        return prevCart.map((item) =>
```

```
          item.id === pokemon.id ? { ...item, quantity: item.quantity + 1 } : item
```

```
        );
```

```
      } else {
```

```
        return [...prevCart, { ...pokemon, quantity: 1 }];
```

```
      }
```

```
    });
```

```
  };
```

```
  const removeFromCart = (indexToRemove) => {
```

```
    setCart(cart.filter((_, index) => index !== indexToRemove));
```

```
  };
```

```
  return (
```

```
    <CartContext.Provider value={{ cart, addToCart, removeFromCart }}>
```

```
      {children}
```

```
    </CartContext.Provider>
```

```
  );
```

```
};
```

ChatGPT Plus

ChatGPT

Claro, te explicaré este componente paso a paso.

1. Importaciones

javascript

Copiar código

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

Aquí se importan React, createContext y useState de la librería react. createContext se usa para crear un nuevo contexto de React, y useState es un hook que permite manejar el estado dentro de un componente funcional.

2. Creación del contexto

javascript

Copiar código

```
export const CartContext = createContext();
```

Aquí se crea un nuevo contexto llamado CartContext. Un contexto en React permite compartir

valores (en este caso, el estado del carrito y las funciones) entre componentes, sin necesidad de pasar explícitamente las props a través de cada nivel del árbol de componentes.

3. Componente CartProvider

javascript

Copiar código

```
export const CartProvider = ({ children }) => {
```

CartProvider es un componente que actúa como el proveedor de este contexto. Recibe una prop llamada children, que representa los componentes hijos que serán envueltos por este proveedor.

4. Estado del carrito

javascript

Copiar código

```
const [cart, setCart] = useState([]);
```

Aquí se define el estado cart utilizando el hook useState. cart es un array que inicialmente está vacío y contiene los productos añadidos al carrito. setCart es la función que se utiliza para actualizar el estado del carrito.

5. Función addToCart

javascript

Copiar código

```
const addToCart = (pokemon) => {  
  setCart((prevCart) => {  
    const existingProduct = prevCart.find((item) => item.id === pokemon.id);  
    if (existingProduct) {  
      return prevCart.map((item) =>  
        item.id === pokemon.id ? { ...item, quantity: item.quantity + 1 } : item  
      );  
    } else {  
      return [...prevCart, { ...pokemon, quantity: 1 }];  
    }  
  });  
};
```

Esta función agrega un producto al carrito.

setCart((prevCart) => {...}): Se usa la función setCart para actualizar el estado del carrito. prevCart es el estado anterior del carrito.

const existingProduct = prevCart.find((item) => item.id === pokemon.id);: Busca si el producto que se quiere agregar ya existe en el carrito comparando el id del producto.

Condicional if (existingProduct):

Si el producto ya existe en el carrito, se actualiza su cantidad sumando 1.

Si el producto no existe, se añade al carrito con una cantidad inicial de 1.

La función retorna el nuevo estado del carrito.

6. Función removeFromCart

javascript

Copiar código

```
const removeFromCart = (indexToRemove) => {
```

```
setCart(cart.filter((, index) => index !== indexToRemove));  
};
```

Esta función elimina un producto del carrito según su índice.

`cart.filter((, index) => index !== indexToRemove)`: Filtra el carrito para remover el producto en el índice especificado por `indexToRemove`.

El nuevo estado del carrito, sin el producto eliminado, se guarda usando `setCart`.

7. Renderizado del proveedor

javascript

Copiar código

```
return (  
  <CartContext.Provider value={{ cart, addToCart, removeFromCart }}>  
    {children}  
  </CartContext.Provider>  
);
```

Aquí, el componente `CartProvider` utiliza `CartContext.Provider` para pasar el estado del carrito (`cart`) y las funciones `addToCart` y `removeFromCart` a los componentes hijos (`children`) que estén envueltos por `CartProvider`.

`value={{ cart, addToCart, removeFromCart }}`: El objeto `value` contiene el estado y las funciones que serán accesibles desde cualquier componente que consuma este contexto.

8. Uso del componente `CartProvider`

Este componente `CartProvider` se usa para envolver cualquier parte de la aplicación donde los componentes necesiten acceder al carrito de compras y las funciones para manipularlo.

Por ejemplo, en el componente `App.js`, podrías envolver tu aplicación con `CartProvider`, de modo que cualquier componente dentro de la aplicación tenga acceso al contexto del carrito.

Resumen

El componente `CartProvider` maneja el estado del carrito de compras en la aplicación. Proporciona funciones para agregar y remover productos del carrito y usa un contexto de React (`CartContext`) para hacer que el estado y las funciones estén disponibles en cualquier parte de la aplicación que consuma este contexto. Esto es útil para mantener un carrito de compras sincronizado entre diferentes páginas o componentes.

En `product list` donde pone

```
{pokemons.map((pokemon) => ( ))}
```

Hay que poner

```
{pokemons.map((pokemon) => ( ))}
```