Spécification Technique pour la page cart.tsx

1. Introduction

La page `Cart.tsx` est responsable de l'affichage des articles du panier, du calcul du montant total et de la gestion du processus de paiement. Elle utilise plusieurs contextes et hooks pour gérer les états et les interactions avec le backend via WebSocket.

2. Objectifs

- Afficher les articles ajoutés au panier.
- Calculer et afficher le montant total des articles dans le panier.
- Gérer le processus de paiement via WebSocket.
- Afficher des messages de réussite ou d'échec du paiement.
- Permettre à l'utilisateur de vider le panier ou de continuer ses achats.

3. Dépendances

- `react`
- `react-router-dom`
- `../../context/shop-context`
- `./cart-item`
- `./cart.css`
- `../../components/navbar`
- `../../models/item`
- `../../stores/items`
- `../../context/socketContext`

4. Propriétés

Aucune propriété directe n'est passée au composant `Cart`.

5. États

- `cartItems` : Un objet contenant les articles dans le panier et leurs quantités, provenant du `ShopContext`.
- `totalAmount` : Le montant total des articles dans le panier.
- `isProcessing` : Un booléen indiquant si le processus de paiement est en cours.
- `paymentSuccess` : Un booléen indiquant si le paiement a réussi.
- `paymentFailed` : Un booléen indiquant si le paiement a échoué.
- `items` : Une liste d'articles disponibles, obtenue depuis le backend.
- `socket` : La connexion WebSocket, obtenue via le contexte `SocketContext`.
- `isConnected` : Un booléen indiquant si la connexion WebSocket est établie.

6. Méthodes

- `getItems`: Récupère tous les articles depuis le backend et met à jour l'état `items`.
- `onCheckout`: Déclenche le processus de paiement en envoyant une demande de paiement via WebSocket.

7. Hooks

- `useContext(ShopContext)`: Utilisé pour accéder aux articles du panier et à la fonction `clearCart`.
- `useNavigate`: Utilisé pour la navigation entre les pages.
- `useSocket`: Utilisé pour accéder à la connexion WebSocket et à son état de connexion.
- `useEffect`: Utilisé pour gérer les effets secondaires comme la récupération des articles, le calcul du montant total, et la gestion des événements WebSocket.

8. Fonctionnalités

8.1. Récupération des Articles

Les articles disponibles sont récupérés dès que le composant est monté via `getItems`, et l'état `items` est mis à jour avec les données obtenues.

```
useEffect(() => {
    getItems();
}, []);
```

8.2. Calcul du Montant Total

Le montant total est recalculé chaque fois que les articles ou les articles du panier changent.

```
useEffect(() => {
    let temp = 0;
    items.forEach(item => {
        if (cartItems[item.id] !== undefined) {
            temp += (item.price * cartItems[item.id]);
        }
    });
    setTotalAmount(temp);
}, [items, cartItems]);
```

8.3. Gestion du Processus de Paiement

Lorsqu'un paiement est déclenché, un événement WebSocket est envoyé avec le montant total. Les états `isProcessing`, `paymentSuccess`, et `paymentFailed` sont utilisés pour afficher le statut du paiement.

```
const onCheckout = () => {
    if (socket && isConnected) {
        setIsProcessing(true);
        setPaymentSuccess(false);
        setPaymentFailed(false);
        socket.emit("checkout", { totalAmount });
    } else {
        console.error("Socket is not connected");
    }
};
```

8.4. Gestion des Événements WebSocket

Des écouteurs sont ajoutés pour gérer les événements `payement-success` et `payement-failed`, et les états correspondants sont mis à jour.

```
useEffect(() => {
    if (socket) {
        socket.on('payement-success', () => {
            setIsProcessing(false);
            setPaymentSuccess(true);
        });

        socket.on('payement-failed', () => {
            setIsProcessing(false);
            setPaymentFailed(true);
        });

        return () => {
            socket.off('payement-success');
            socket.off('payement-failed');
        };
    }
}, [socket]);
```

9. Structure du Composant

```
import React, { useContext, useEffect, useState } from "react";
import { ShopContext } from "../../context/shop-context";
import { CartItem } from "./cart-item";
import './cart.css';
import { useNavigate } from "react-router-dom";
import { Navbar } from "../../components/navbar";
import { Item as ItemType } from "../../models/item";
import { getAllItems } from "../../stores/items";
import { useSocket } from "../../context/socketContext";

export const Cart = () => {
    const { cartItems, clearCart } = useContext(ShopContext);
    const [totalAmount, setTotalAmount] = useState<number>(0);
    const [isProcessing, setIsProcessing] =
useState<boolean>(false);
    const [paymentSuccess, setPaymentSuccess] =
useState<boolean>(false);
    const [paymentFailed, setPaymentFailed] =
useState<boolean>(false);
    const navigate = useNavigate();
    const [items, setItems] = useState<ItemType[]>([]);
    const { socket, isConnected } = useSocket();
    const gifUrl =
"https://media.giphy.com/media/QBd2kLB5qDmysEXre9/giphy.gif";

    const getItems = async () => {
        const result = await getAllItems();
        if (result.status === "ok") {
            setItems(result.data as ItemType[]);
        }
    };

    useEffect(() => {
        let temp = 0;
        items.forEach(item => {
            if (cartItems[item.id] !== undefined) {
                temp += (item.price * cartItems[item.id]);
            }
        });
        setTotalAmount(temp);
```

```
}, [items, cartItems]);

useEffect(() => {
    getItems();
}, []);

useEffect(() => {
    if (socket) {
        socket.on('payement-success', () => {
            setIsProcessing(false);
            setPaymentSuccess(true);
        });

        socket.on('payement-failed', () => {
            setIsProcessing(false);
            setPaymentFailed(true);
        });

        return () => {
            socket.off('payement-success');
            socket.off('payement-failed');
        };
    }
}, [socket]);

const onCheckout = () => {
    if (socket && isConnected) {
        console.log("Subtotal: " + totalAmount);
        console.log(socket);
        setIsProcessing(true);
        setPaymentSuccess(false);
        setPaymentFailed(false);
        socket.emit("checkout", { totalAmount });
    } else {
        console.error("Socket is not connected");
    }
};

return (
    <div className="cart">
        <Navbar />
```

```
            {paymentSuccess ? (
                <div className="success">
                    <p>Paiement effectué !</p>
                </div>
            ) : paymentFailed ? (
                <div className="failed">
                    <p>Échec du paiement. Veuillez réessayer.</p>
                </div>
            ) : isProcessing ? (
                <div className="processing">
                    <p>Processing to payment...</p>
                    <img src={gifUrl} alt="Processing..." />
                </div>
            ) : (
                <>
                    <div>
                        <h1>Your Cart Items</h1>
                    </div>
                    <div className="cartItems">
                        {items.map((item) => {
                            if (cartItems[item.id] !== undefined) {
                                return <CartItem key={item.id}
data={item} />;
                            }
                        })}
                    </div>
                    {totalAmount > 0 ? (
                        <div className="checkout">
                            <p>Subtotal: {totalAmount} €</p>
                            <button onClick={() =>
navigate("/shop")}>Continue Shopping</button>
                            <button
onClick={onCheckout}>Checkout</button>
                            <button onClick={clearCart}>Clear
Cart</button>
                        </div>
                    ) : (
                        <h1>Your cart is Empty</h1>
                    )}
                </>
            )}
```

```
        </div>
    );
};
```

10. Conclusion

La page `Cart` gère efficacement l'affichage et la manipulation des articles du panier, le calcul du montant total, et le processus de paiement. Elle utilise des contextes pour accéder aux articles du panier et à la connexion WebSocket, assurant ainsi une gestion centralisée et cohérente des états et des événements.