Exercice 5 - Panier avec useReducer

Objectif

Gérer un état complexe avec useReducer au lieu de useState.

Concepts abordés

- useReducer vs useState
- Actions et dispatch
- Reducer function (pure)
- Pattern action avec type et payload

Quand utiliser useReducer ?

useState	useReducer
État simple (booléen, string, number)	État complexe (objet avec plusieurs champs)
Peu de mises à jour différentes	Plusieurs types de mises à jour
Logique simple	Logique complexe, interdépendante

Énoncé

Créer un panier d'achat avec :

1. Liste des produits disponibles
2. Ajouter un produit au panier
3. Retirer un produit du panier
4. Modifier la quantité
5. Vider le panier
6. Afficher le total

Actions à implémenter

- `ADD_ITEM` : Ajouter un produit (ou +1 si déjà présent)
- `REMOVE_ITEM` : Retirer complètement un produit
- `UPDATE_QUANTITY` : Modifier la quantité
- `CLEAR_CART` : Vider le panier

Correction

```
// src/exercices/module-2/ex05-panier/Panier.jsx

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

```jsx
import './Panier.css';

// ═══════════════════════════════════════════════════════════════
// DONNÉES DES PRODUITS (simule un catalogue)
// ═══════════════════════════════════════════════════════════════
const PRODUCTS = [
  { id: 1, name: 'React T-Shirt', price: 25 },
  { id: 2, name: 'JavaScript Mug', price: 12 },
  { id: 3, name: 'Node.js Sticker Pack', price: 5 },
  { id: 4, name: 'TypeScript Cap', price: 18 },
];

// ═══════════════════════════════════════════════════════════════
// ÉTAT INITIAL DU PANIER
// ═══════════════════════════════════════════════════════════════
const initialState = {
  items: [],  // { id, name, price, quantity }
};

// ═══════════════════════════════════════════════════════════════
// REDUCER
//
// Un reducer est une fonction PURE qui :
// - Prend l'état actuel et une action
// - Retourne le NOUVEL état (sans muter l'ancien)
//
// Structure d'une action :
// { type: 'ADD_ITEM', payload: { id: 1, name: '...', price: 25 } }
// ═══════════════════════════════════════════════════════════════
function cartReducer(state, action) {
  switch (action.type) {
    // ─────────────────────────────────────────────────────────
    // AJOUTER UN PRODUIT
    // Si déjà dans le panier : +1 quantité
    // Sinon : ajouter avec quantité 1
    // ─────────────────────────────────────────────────────────
    case 'ADD_ITEM': {
      const existingItem = state.items.find(
        item => item.id === action.payload.id
      );

      if (existingItem) {
        // Produit déjà dans le panier : +1
        return {
          ...state,
          items: state.items.map(item =>
            item.id === action.payload.id
              ? { ...item, quantity: item.quantity + 1 }
              : item
          ),
        };
      } else {
        // Nouveau produit
        return {
```

```
        ...state,
        items: [...state.items, { ...action.payload, quantity: 1 }],
      };
    }
  }

  // ─────────────────────────────────────────────────────────────
  // RETIRER UN PRODUIT
  // ─────────────────────────────────────────────────────────────
  case 'REMOVE_ITEM': {
    return {
      ...state,
      items: state.items.filter(item => item.id !== action.payload.id),
    };
  }

  // ─────────────────────────────────────────────────────────────
  // MODIFIER LA QUANTITÉ
  // Si quantité <= 0 : retirer le produit
  // ─────────────────────────────────────────────────────────────
  case 'UPDATE_QUANTITY': {
    const { id, quantity } = action.payload;

    if (quantity <= 0) {
      // Quantité 0 ou négative : retirer
      return {
        ...state,
        items: state.items.filter(item => item.id !== id),
      };
    }

    return {
      ...state,
      items: state.items.map(item =>
        item.id === id ? { ...item, quantity } : item
      ),
    };
  }

  // ─────────────────────────────────────────────────────────────
  // VIDER LE PANIER
  // ─────────────────────────────────────────────────────────────
  case 'CLEAR_CART': {
    return {
      ...state,
      items: [],
    };
  }

  // ─────────────────────────────────────────────────────────────
  // ACTION INCONNUE
  // En dev, on peut throw une erreur
  // En prod, on retourne le state inchangé
  // ─────────────────────────────────────────────────────────────
```

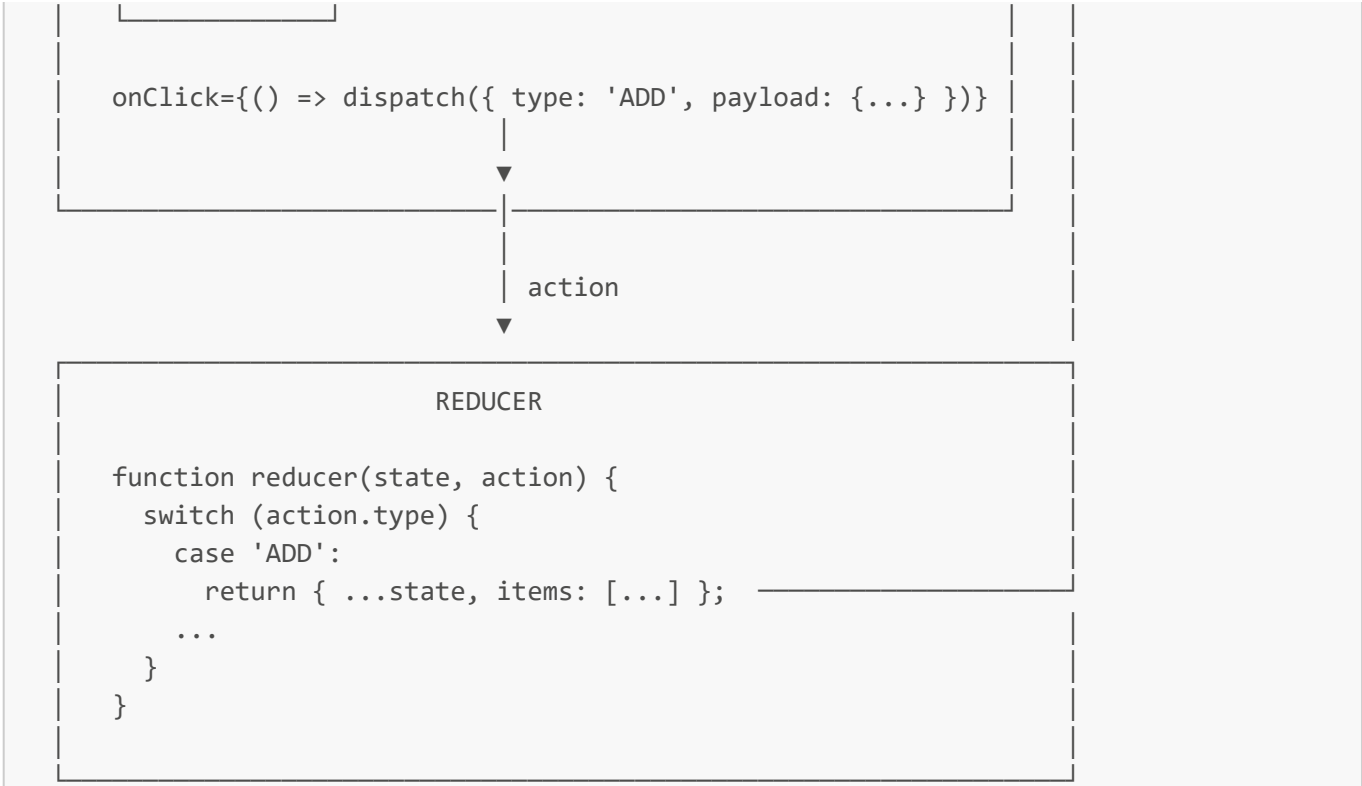
```
      default: {
        console.warn(`Action inconnue : ${action.type}`);
        return state;
      }
    }
  }

  // ═══════════════════════════════════════════════════════════════
  // COMPOSANT PRINCIPAL
  // ═══════════════════════════════════════════════════════════════
  function Panier() {
    // useReducer retourne [state, dispatch]
    // dispatch est une fonction pour envoyer des actions
    const [state, dispatch] = useReducer(cartReducer, initialState);

    // ─────────────────────────────────────────────────────────────
    // CALCULS DÉRIVÉS
    // ─────────────────────────────────────────────────────────────
    const totalItems = state.items.reduce(
      (sum, item) => sum + item.quantity,
      0
    );

    const totalPrice = state.items.reduce(
      (sum, item) => sum + item.price * item.quantity,
      0
    );

    // ─────────────────────────────────────────────────────────────
    // HANDLERS
    // Chaque handler dispatch une action
    // ─────────────────────────────────────────────────────────────
    function handleAddToCart(product) {
      dispatch({
        type: 'ADD_ITEM',
        payload: product,
      });
    }

    function handleRemoveFromCart(id) {
      dispatch({
        type: 'REMOVE_ITEM',
        payload: { id },
      });
    }

    function handleUpdateQuantity(id, quantity) {
      dispatch({
        type: 'UPDATE_QUANTITY',
        payload: { id, quantity },
      });
    }

    function handleClearCart() {
```

```
    dispatch({ type: 'CLEAR_CART' });
  }

  // ─────────────────────────────────────────────────────────────
  // RENDU
  // ─────────────────────────────────────────────────────────────
  return (
    <div className="shop">
      {/* Catalogue */}
      <section className="catalog">
        <h2>Produits</h2>
        <div className="products-grid">
          {PRODUCTS.map(product => (
            <div key={product.id} className="product-card">
              <h3>{product.name}</h3>
              <p className="price">{product.price} €</p>
              <button
                onClick={() => handleAddToCart(product)}
                className="add-btn"
              >
                Ajouter au panier
              </button>
            </div>
          ))}
        </div>
      </section>

      {/* Panier */}
      <section className="cart">
        <div className="cart-header">
          <h2>Panier ({totalItems})</h2>
          {state.items.length > 0 && (
            <button onClick={handleClearCart} className="clear-btn">
              Vider
            </button>
          )}
        </div>

        {state.items.length === 0 ? (
          <p className="empty-cart">Votre panier est vide</p>
        ) : (
          <>
            <ul className="cart-items">
              {state.items.map(item => (
                <li key={item.id} className="cart-item">
                  <div className="item-info">
                    <span className="item-name">{item.name}</span>
                    <span className="item-price">{item.price} €</span>
                  </div>

                  <div className="quantity-controls">
                    <button
                      onClick={() =>
                        handleUpdateQuantity(item.id, item.quantity - 1)
```

```
                    }
                  >
                    -
                  </button>
                  <span>{item.quantity}</span>
                  <button
                    onClick={() =>
                      handleUpdateQuantity(item.id, item.quantity + 1)
                    }
                  >
                    +
                  </button>
                </div>

                <span className="item-total">
                  {item.price * item.quantity} €
                </span>

                <button
                  onClick={() => handleRemoveFromCart(item.id)}
                  className="remove-btn"
                >
                  ×
                </button>
              </li>
            ))}
          </ul>

          <div className="cart-total">
            <strong>Total : {totalPrice.toFixed(2)} €</strong>
          </div>
        </>
      )}
    </section>
  </div>
 );
}

export default Panier;
```

Schéma mental : Flux useReducer

```
┌──────────────────────────────────────────────────────────────────┐
│                        COMPOSANT                                 │
│                                                                  │
│   const [state, dispatch] = useReducer(reducer, initialState)    │
│                                                                  │
│   ┌──────────────┐                                               │
│   │    state     │ ◄─────────────────────────────────────┐       │
│   │  { items }   │                                       │       │
```



Comparaison useState vs useReducer

```
// Avec useState : multiple setState, logique dispersée
const [items, setItems] = useState([]);

function addItem(product) {
  const existing = items.find(i => i.id === product.id);
  if (existing) {
    setItems(items.map(i =>
      i.id === product.id ? { ...i, qty: i.qty + 1 } : i
    ));
  } else {
    setItems([...items, { ...product, qty: 1 }]);
  }
}

function removeItem(id) {
  setItems(items.filter(i => i.id !== id));
}

// Avec useReducer : logique centralisée dans le reducer
dispatch({ type: 'ADD_ITEM', payload: product });
dispatch({ type: 'REMOVE_ITEM', payload: { id } });
```

Pour aller plus loin

1. Persister le panier dans localStorage
2. Ajouter un code promo
3. Créer un Context pour le panier global