React Typescript
(Shopping Cart)

By: Fei

Table of contents

React Hooks

useMemo

Hooks dari React yang menyediakan method untuk memoize hasil yang direturn oleh sebuah function. Parameter yang diterima adalah callback function dan array of dependencies.

useCallback

Hooks dari React yang mirip dengan memo tapi melakukan memoize pada function dan bukan hasil dari function tersebut. Parameter yang diterima adalah function dan array of dependencies.

useEffect

Hooks dari React yang akan menjalankan side effect dari perubahan yang terjadi pada variable yang dependent dengan suatu function. Parameter yang diterima adalah function dan array of dependencies. Function hanya akan dieksekusi sekali jika tidak ada perubahan yang terjadi

Prompt: React and TypeScript

React Hooks

useState

Hooks yang memberikan state pada functional components. Memiliki sebuah state variable yang dapat diganti dan function yang dapat mengganti state tersebut.

useReducer

Hook alternative yang dapat digunakan selain useState jika memiliki logic yang kompleks dan terdiri dari beberapa case yang dapat digunakan untuk mengubah state yang dimiliki

useContext

Hook yang digunakan untuk menggunakan sebuah context yang telah dibuat sebelumnya pada functional component lain. Digunakan jika ingin pass function atau variable tanpa melakukan passing dalam bentuk object secara eksplisit

Prompt: React and TypeScript

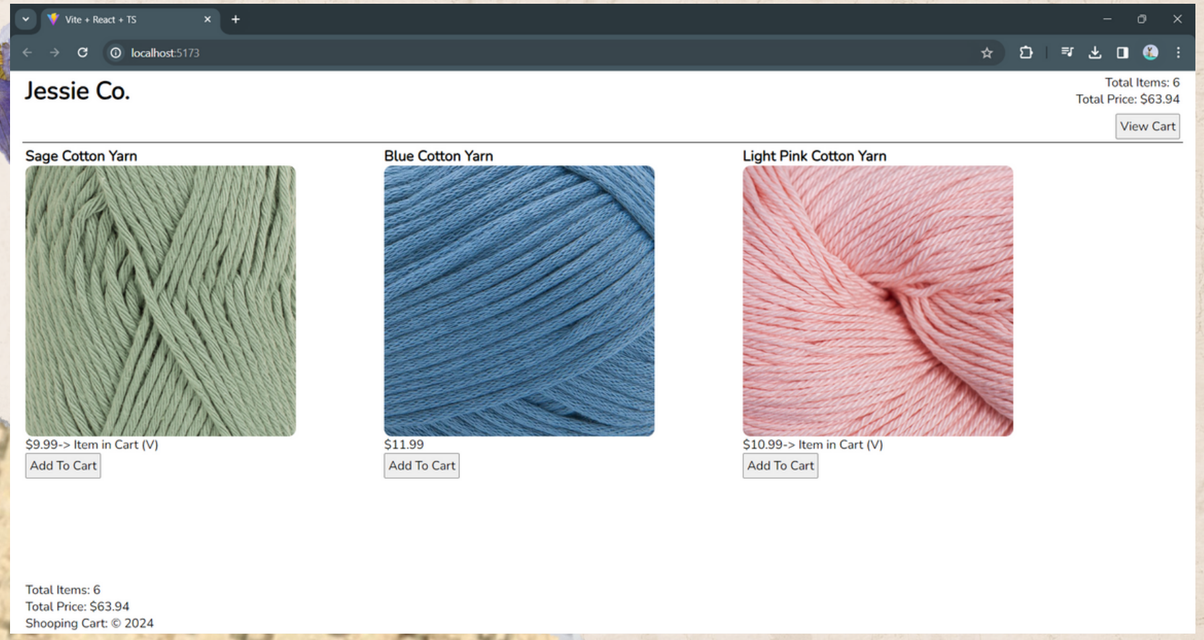
Shopping Cart (1)

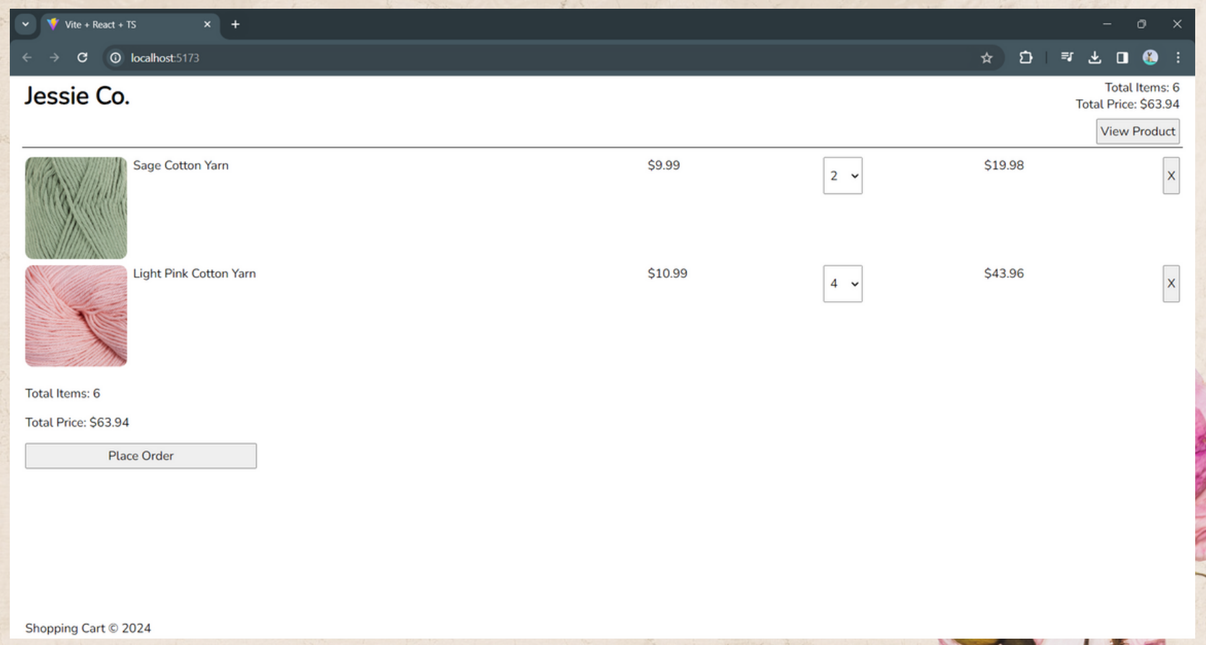
Home | requirements:

- Header dengan navbar
- Content yang berisi list product
- Footer

Cart | requirements:

- Header dan footer sama dengan home page
- List product yang ada di cart
- Detail quantity dan harga
- Select option untuk ubah quantity
- Remove button
- Place order button

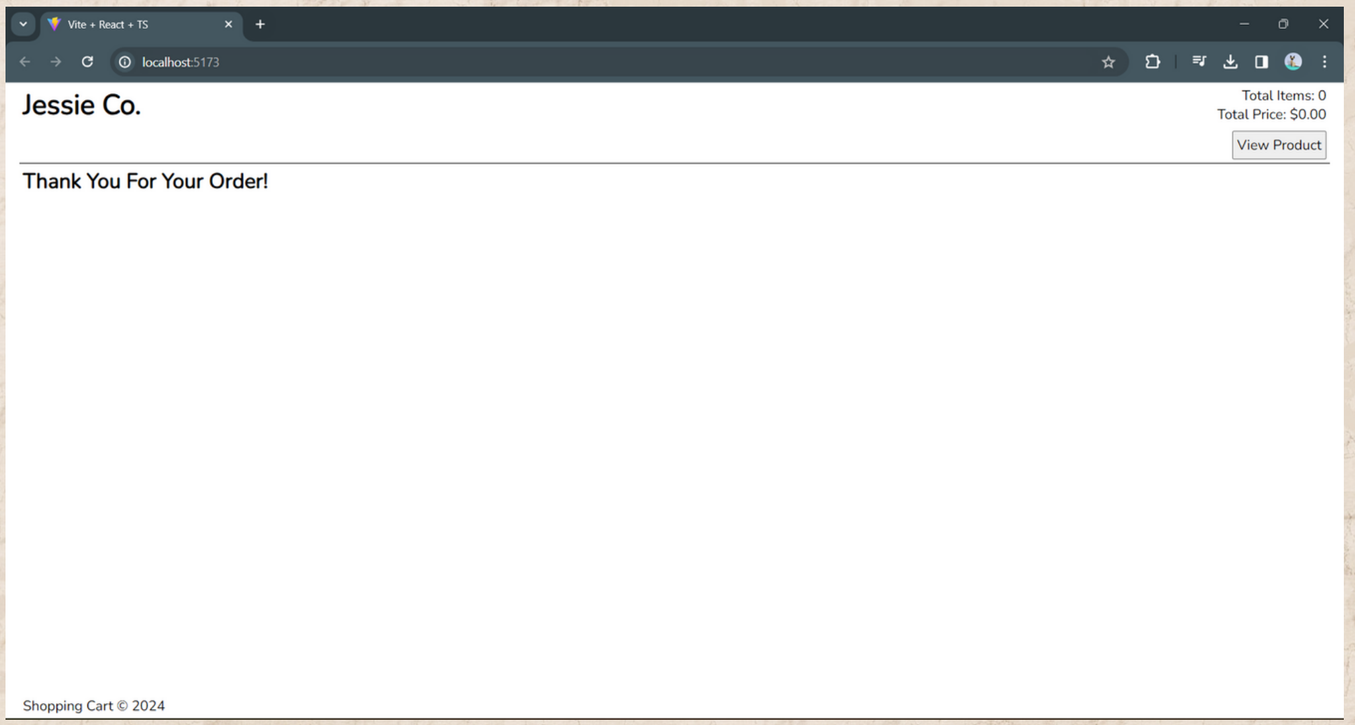




Shopping Cart (2)

Cart | requirements:

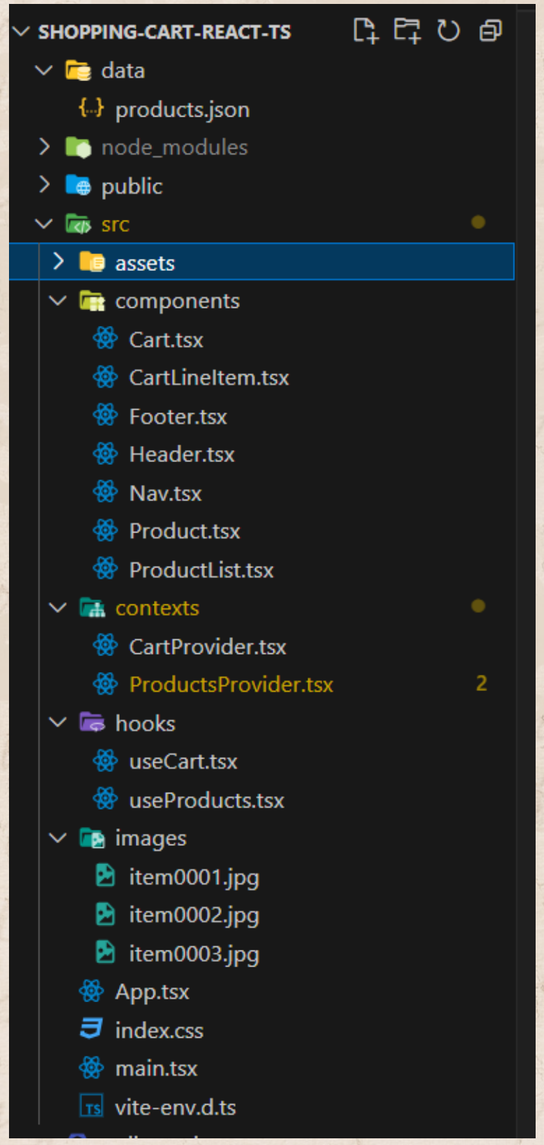
Jika place order, remove semua list item dalam cart dan tampilkan thank you message



Project Explorer

Steps:

1. Tentukan apa saja komponen yang akan menyusun setiap page
2. Tentukan operasi apa saja yang dapat dilakukan oleh setiap komponen
3. Import data dan images yang diperlukan
4. Buat context yang dapat digunakan oleh setiap component
5. Buat folder hook untuk membuat code menjadi lebih bersih



main.tsx & App.tsx

Main.tsx

```tsx
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.tsx";
import "./index.css";

import { CartProvider } from "./contexts/CartProvider.tsx";
import { ProductsProvider } from "./contexts/ProductsProvider.tsx";

ReactDOM.createRoot(document.getElementById("root")!).render(
    <React.StrictMode>
        <ProductsProvider>
            <CartProvider>
                <App />
            </CartProvider>
        </ProductsProvider>
    </React.StrictMode>
);

```

App.tsx - menggunakan useState

```tsx
import Header from "./components/Header"
import Footer from "./components/Footer"
import Cart from "./components/Cart"
import ProductList from "./components/ProductList"

import { useState } from "react"

function App() {

  const [viewCart, setViewCart] = useState<boolean>(false)

  const pageContent = viewCart ? <Cart></Cart> : <ProductList></ProductList>

  const content = (
    <>
      <Header viewCart={viewCart} setViewCart={setViewCart}></Header>
      {pageContent}
      <Footer viewCart={viewCart}></Footer>
    </>
  )

  return content
}

export default App
```

ProductProvider.tsx

Products Provider

Menyediakan products yang dapat diolah oleh children (jika ada) yang akan mewarisi object products. Dapat diakses dengan context yang sudah dibuat

```tsx
import { ReactElement, createContext, useState, useEffect } from "react";

export type ProductType = {
    sku: string;
    name: string;
    price: number;
};

// fetching form json file, siapkan initial state kosong
// const initState: ProductType[] = [];

// static data, not fetching from json file
const initState: ProductType[] = [
    {
        "sku": "item0001",
        "name": "Sage Cotton Yarn",
        "price": 9.99
    },
    {
        "sku": "item0002",
        "name": "Blue Cotton Yarn",
        "price": 11.99
    },
    {
        "sku": "item0003",
        "name": "Light Pink Cotton Yarn",
        "price": 10.99
    }
]

export type UseProductsContextType = { products: ProductType[] };
```

```tsx
const initContextState: UseProductsContextType = { products: [] };

// buat context dengan parameter initial state (array of ProductType)
const ProductsContext = createContext<UseProductsContextType>(initContextState);

type ChildrenType = { children?: ReactElement | ReactElement[] };

export const ProductsProvider = ({ children }: ChildrenType): ReactElement => {
    // karena tidak menggunakan fetch, maka seharusnya tidak usah pakai useState
    const [products, setProducts] = useState<ProductType[]>(initState)

    // kalau mau fetch productList dari json file
    // useEffect(() => {
    //     const fetchProducts = async (): Promise<ProductType[]> => {
    //         const data = await fetch('http://localhost:3500/products')
    //         .then((res) => res.json()).catch(err => {
    //             if (err instanceof Error) console.log(err.message)
    //         })
    //         return data
    //     }

    //     fetchProducts().then(products => setProducts(products))
    // }, [])

    return (
        <ProductsContext.Provider value={{ products }}>
            {children}
        </ProductsContext.Provider>
    )

}

export default ProductsContext;

```

CartProvider.tsx

Cart Provider

Menyediakan context yang dapat digunakan untuk melakukan perubahan pada state yang dimiliki oleh cart

```tsx
import { ReactElement, createContext, useMemo, useReducer } from "react";

// tipe dari setiap item yang ada di cart
export type CartItemType = {
    sku: string;
    name: string;
    price: number;
    qty: number;
};

// state dari cart yang berisi array of product (cart item)
type CartStateType = { cart: CartItemType[] };

// initial state
const initCartState: CartStateType = { cart: [] };

// aksi yang dapat dilakukan oleh components
const REDUCER_ACTION_TYPE = {
    ADD: "ADD",
    REMOVE: "REMOVE",
    QTY: "QTY",
    SUBMIT: "SUBMIT",
};

export type ReducerActionType = typeof REDUCER_ACTION_TYPE;

// aksi yang dapat dilakukan berdasarkan reducer action type, payload bs diisi product saat add, remove, atau set qty
export type ReducerAction = {
    type: string;
    payload?: CartItemType;
};

```

```tsx
const reducer = (state: CartStateType, action: ReducerAction): CartStateType => {
    switch (action.type) {
        // menambah item pada cart saat belum ada atau sudah ada (dari product list page)
        case REDUCER_ACTION_TYPE.ADD: {
            if (!action.payload) {
                throw new Error("action.payload missing in ADD action");
            }

            const { sku, name, price } = action.payload;

            const filteredCart: CartItemType[] = state.cart.filter((item) => item.sku !== sku);

            const itemExist: CartItemType | undefined = state.cart.find((item) => item.sku === sku);

            const qty: number = itemExist ? itemExist.qty + 1 : 1;

            return { ...state, cart: [...filteredCart, { sku, name, price, qty }] };
        }
        // menghapus item tertentu dari cart
        case REDUCER_ACTION_TYPE.REMOVE: {
            if (!action.payload) {
                throw new Error("action.payload missing in REMOVE action");
            }

            const { sku } = action.payload;

            const filteredCart: CartItemType[] = state.cart.filter((item) => item.sku !== sku);

            return { ...state, cart: [...filteredCart] };
        }
        // set quantity dari cart berdasarkan qty yg diberikan oleh user
        case REDUCER_ACTION_TYPE.QTY: {
            if (!action.payload) {
                throw new Error("action.payload missing in QUANTITY action");
            }

            const { sku, qty } = action.payload;

            const itemExist: CartItemType | undefined = state.cart.find((item) => item.sku === sku);

            if (!itemExist) {
                throw new Error("Item must exist in order to update quantity");
            }

            const updatedItem: CartItemType = { ...itemExist, qty };

            const filteredCart: CartItemType[] = state.cart.filter((item) => item.sku !== sku);

            return { ...state, cart: [...filteredCart, updatedItem] };
        }
        // clear cart items saat user submit cartnya
        case REDUCER_ACTION_TYPE.SUBMIT: {
            return { ...state, cart: [] };
        }
        default:
            throw new Error("Unidentified reducer action type");
    }
};
```

CartProvider.tsx

Cart Provider

Context mereturn dispatch function, actions yang dapat digunakan untuk menggunakan dispatch function, total item, total harga, dan object cart. Semua di pass pada children yang dimiliki (jika ada).

```tsx
const useCartContext = (initCartState: CartStateType) => {
    const [state, dispatch] = useReducer(reducer, initCartState);

    const REDUCER_ACTIONS = useMemo(() => {
        return REDUCER_ACTION_TYPE;
    }, []);

    const totalItems: number = state.cart.reduce((previousValue, cartItem) => {
        return previousValue + cartItem.qty;
    }, 0);

    const totalPrice = new Intl.NumberFormat("en-US", { style: "currency", currency: "USD" }).format(
        state.cart.reduce((prev, cartItem) => {
            return prev + cartItem.qty * cartItem.price;
        }, 0)
    );

    const cart = state.cart.sort((a, b) => {
        // sort berdasarkan id pada sku
        const itemA = Number(a.sku.slice(-4))
        const itemB = Number(b.sku.slice(-4))

        return itemA - itemB
    })

    return { dispatch, REDUCER_ACTIONS, totalItems, totalPrice, cart }
};
```

```tsx
export type useCartContextType = ReturnType<typeof useCartContext>

const initCartContextState: useCartContextType = {
    dispatch: () => {},
    REDUCER_ACTIONS: REDUCER_ACTION_TYPE,
    totalItems: 0,
    totalPrice: '',
    cart: []
}

export const CartContext = createContext<useCartContextType>(initCartContextState)

type ChildrenType = { children?: ReactElement | ReactElement[] }

export const CartProvider = ({ children }: ChildrenType): ReactElement => {
        return (
            <CartContext.Provider value={useCartContext(initCartState)}>
                {children}
            </CartContext.Provider>
        )
}

export default CartContext
```

Hook Folder

Custom React hook file yang digunakan untuk membantu menghubungkan components yang ingin menggunakan suatu context/hook dan file provider yang memiliki context tersebut

useCart.tsx

```tsx
import { useContext } from "react";
import CartContext from "../contexts/CartProvider";
import { useCartContextType } from "../contexts/CartProvider";

const useCart = (): useCartContextType => {
    return useContext(CartContext)
}

export default useCart
```

useProducts.tsx

```tsx
import { useContext } from "react";
import ProductsContext from "../contexts/ProductsProvider";
import { UseProductsContextType } from "../contexts/ProductsProvider";

const useProducts = (): UseProductsContextType => {
    return useContext(ProductsContext)
}

export default useProducts
```

Components

Building blocks dari UI website yang kita kembangkan. Menggunakan context yang sudah disediakan oleh provider untuk menentukan behavior dari setiap elemen yang akan menyusun website.

Header.tsx

```tsx
import Nav from "./Nav";
import useCart from "../hooks/useCart";

type PropsType = {
    viewCart: boolean;
    setViewCart: React.Dispatch<React.SetStateAction<boolean>>;
};

const Header = ({ viewCart, setViewCart }: PropsType) => {

    const { totalItems, totalPrice } = useCart()

    const content = (
        <header className="header">
            <div className="header__title-bar">
                <h1>Jessie Co.</h1>
                <div className="header__price-box">
                    <p>Total Items: {totalItems}</p>
                    <p>Total Price: {totalPrice}</p>
                </div>
            </div>

            <Nav viewCart={viewCart} setViewCart={setViewCart}></Nav>
        </header>
    );
    return content;
};

export default Header;
```

Nav.tsx

```tsx
type PropsType = {
    viewCart: boolean,
    setViewCart: React.Dispatch<React.SetStateAction<boolean>>
}

const Nav = ({viewCart, setViewCart}: PropsType) => {
    const button = viewCart
    ? <button onClick={() => setViewCart(false)}>View Product</button>
    : <button onClick={() => setViewCart(true)}>View Cart</button>

    const content = (
        <nav className="nav">
            {button}
        </nav>
    )
    return content
}

export default Nav
```

Components

Footer.tsx

```tsx
import useCart from "../hooks/useCart"

type PropsType = {
    viewCart: boolean,
}

const Footer = ({viewCart}: PropsType) => {
    const {totalItems, totalPrice} = useCart()

    const year: number = new Date().getFullYear()

    const pageContent = viewCart
    ? <p>Shopping Cart &copy; {year}</p>
    : (
    <>
        <p>Total Items: {totalItems}</p>
        <p>Total Price: {totalPrice}</p>
        <p>Shooping Cart: &copy; {year}</p>
    </>)

    const content = (
        <footer className="footer">
            {pageContent}
        </footer>
    )

    return content
}

export default Footer
```

ProductList.tsx

```tsx
import useCart from "../hooks/useCart"
import useProducts from "../hooks/useProducts"
import Product from "./Product"

import { ReactElement } from "react"

const ProductList = () => {

    const { dispatch, REDUCER_ACTIONS, cart } = useCart()
    const { products } = useProducts()

    let pageContent: ReactElement | ReactElement[] =
    <p>Loading...</p>

    if(products?.length) {
        pageContent = products.map(product => {
            const inCart: boolean = cart.some(item => item.sku === product.sku)

            return (
                <Product
                key={product.sku}
                product={product}
                dispatch={dispatch}
                REDUCER_ACTIONS={REDUCER_ACTIONS}
                inCart={inCart}></Product>
            )
        })
    }

    const content = (
        <main className="main main--product">
            {pageContent}
        </main>
    )

    return content
}

export default ProductList
```

Components

```tsx
import { ProductType } from "../contexts/ProductsProvider";

import { ReducerActionType, ReducerAction } from "../contexts/CartProvider";
import { ReactElement, memo } from "react";

type PropsType = {
    product: ProductType;
    dispatch: React.Dispatch<ReducerAction>;
    REDUCER_ACTIONS: ReducerActionType;
    inCart: boolean;
};

const Product = ({ product, dispatch, REDUCER_ACTIONS, inCart }: PropsType): ReactElement => {
    const img: string = new URL(`../images/${product.sku}.jpg`, import.meta.url).href;
    console.log(img);

    const onAddToCart = () => {
        dispatch({ type: REDUCER_ACTIONS.ADD, payload: { ...product, qty: 1 } });
    };

    const itemInCart = inCart ? "-> Item in Cart (V)" : null;

    const content = (
        <article className="product">
            <h3>{product.name}</h3>
            <img className="product__img" src={img} alt={product.name} />
            <p>
                {new Intl.NumberFormat("en-US", {
                    style: "currency",
                    currency: "USD",
                }).format(product.price)}
                {itemInCart}
            </p>
            <button onClick={onAddToCart}>Add To Cart</button>
        </article>
    );

    return content;
};

```

```tsx
function areProductsEqual(
    { product: prevProduct, inCart: prevInCart }: PropsType,
    { product: nextProduct, inCart: nextInCart }: PropsType
) {
    return Object.keys(prevProduct).every((key) => {
        return (
            prevProduct[key as keyof ProductType] === nextProduct[key as keyof ProductType] && prevInCart === nextInCart
        );
    });
}

const MemoizedProduct = memo<typeof Product>(Product, areProductsEqual);


export default MemoizedProduct;
```

Product.tsx

Isi dari looping yang terjadi pada komponen ProductList. Menggunakan memo untuk melakukan memoize pada product yang sudah pernah diload sebelumnya. Jika tidak ada perubahan maka yang direturn adalah object product yang sudah dimemoize sebelumnya

Components

```tsx
import useCart from "../hooks/useCart"
import { useState } from "react"
import CartLineItem from "./CartLineItem"

const Cart = () => {
    const [confirm, setConfirm] = useState<boolean>(false)

    const { dispatch, REDUCER_ACTIONS, totalItems, totalPrice , cart }
    = useCart()

    const onSubmitOrder = () => {
        dispatch({ type: REDUCER_ACTIONS.SUBMIT })
        setConfirm(true)
    }

    const pageContent = confirm
    ? <h2>Thank You For Your Order!</h2>
    : <>
        <h2 className="offscreen">Cart</h2>
        <ul className="cart">
            {cart.map(item => {
                return (
                    <CartLineItem
                    key={item.sku}
                    item={item}
                    dispatch={dispatch}
                    REDUCER_ACTIONS={REDUCER_ACTIONS}
                    ></CartLineItem>
                )
            })}
        </ul>
        <div className="cart__totals">
            <p>Total Items: {totalItems}</p>
            <p>Total Price: {totalPrice}</p>
            <button className="cart__submit" disabled={!totalItems} onClick={onSubmitOrder}>
                Place Order
            </button>
        </div>
    </>

    const content = (
        <main className="main main--cart">
            {pageContent}
        </main>
    )

    return content
}

export default Cart
```

Cart.tsx

Menampilkan semua isi dari object cart dengan melakukan looping dan memanggil CartLineItem untuk menyelesaikan setiap list dari product.

Components

CartLineItem.tsx

Menampilkan setiap detail dari item yang terdapat pada cart beserta dengan select option untuk quantity dan remove button.

```tsx
import { ChangeEvent, ReactElement, memo } from "react";
import { CartItemType } from "../contexts/CartProvider";
import { ReducerAction, ReducerActionType } from "../contexts/CartProvider";

type PropsType = {
    item: CartItemType;
    dispatch: React.Dispatch<ReducerAction>;
    REDUCER_ACTIONS: ReducerActionType;
};

const CartLineItem = ({ item, dispatch, REDUCER_ACTIONS }: PropsType) => {
    const img: string = new URL(`../images/${item.sku}.jpg`, import.meta.url).href;

    const lineTotal: number = (item.qty * item.price)

    const highestQuantity: number = 20 > item.qty ? 20 : item.qty

    const optionValues: number[] = [ ...Array(highestQuantity).keys()]
    .map(i => i + 1)

    const options: ReactElement[] = optionValues.map(val => {
        return <option key={`opt${val}`} value={val}>
            {val}
        </option>
    })

    const onChangeQty = (e: ChangeEvent<HTMLSelectElement>) => {
        dispatch({
            type: REDUCER_ACTIONS.QTY,
            payload: { ...item, qty: Number(e.target.value) }
        })
    }

    const onRemoveFromCart = () => dispatch({
        type: REDUCER_ACTIONS.REMOVE,
        payload: item
    })

    const content = (
        <li className="cart__item">
            <img className="cart__img" src={img} alt={item.name} />
            <div aria-label="Item Name">{item.name}</div>
            <div aria-label="Price Per Item">{
                new Intl.NumberFormat('en-US', { style: 'currency', currency: 'USD'}). format
                (item.price)
            }</div>

            <label className="offscreen" htmlFor="itemQty">
                Item Quantity:
            </label>
            <select className="cart__select" name="itemQty" id="itemQty"
            value={item.qty} aria-label="Item Quantity" onChange={onChangeQty}>
                {options}
            </select>

            <div className="cart__item-subtotal" aria-label="Line Item Subtotal">
                {new Intl.NumberFormat('en-US', { style: 'currency', currency: 'USD'}). format
                (lineTotal)}
            </div>

            <button className="cart__button" aria-label="Remove Item From Cart"
            title="Remove Item From Cart" onClick={onRemoveFromCart}>
                X
            </button>
        </li>
    )

    return content
};

function areItemsEqual ( { item: prevItem }: PropsType, { item: nextItem }: PropsType) {
    return Object.keys(prevItem).every(key => {
        return prevItem[key as keyof CartItemType] === nextItem[key as keyof CartItemType]
    })
}

const MemoizedCartLineItem = memo<typeof CartLineItem>(CartLineItem, areItemsEqual)


export default MemoizedCartLineItem;
```

Thank you

GitHub Repo:

Small projects/typescript/shopping-cart-react-ts