Components

As of right now Meridian furniture is live with a working cart and dynamic product listings and a custom `useCart()` hook. It utilizes the context API to make a cart manipulatable and viewable from anywhere in the site as long as its a child of the root layout where the cart provider is integrated.

Challenges

The major challenge faced was the implementation of the cart. and ensuring that duplicate entries don't get created.

```
"use client";
import { createContext, useState, useContext } from "react";
import { productType } from "@/lib/types";

export interface CartItem {
  product: productType;
  quantity: number;
}



interface CartState {
  cartItems: CartItem[];
  total: number;
  addToCart: (item: CartItem) => void;
  removeFromCart: (item: CartItem) => void;
}

const CartContext = createContext<CartState | null>(null);

export const CartProvider: React.FC<{ children: React.ReactNode }> = ({
  children,
}) => {
  const [cartItems, setCartItems] = useState<CartItem[]>([]);
  const [total, setTotal] = useState(0);

  const addToCart = (item: CartItem) => {
    const existingItemIndex = cartItems.findIndex(
      (i) => i.product._id === item.product._id,
    );

    if (existingItemIndex !== -1) {
      const newCartItems = [ ... cartItems];
```

```
      newCartItems[existingItemIndex] = item;
      setCartItems(newCartItems);
      setTotal(
        total -
          cartItems[existingItemIndex].product.price *
            cartItems[existingItemIndex].quantity +
          item.product.price * item.quantity,
      );
    } else {
      setCartItems([...cartItems, item]);
      setTotal(total + item.product.price * item.quantity);
    }
  };

  const removeFromCart = (item: CartItem) => {
    const updatedCartItems = cartItems.filter(
      (i) => i.product.id !== item.product.id,
    );
    setCartItems(updatedCartItems);
    setTotal(total - item.product.price * item.quantity);
  };

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

export const useCart = () => {
  const context = useContext(CartContext);
  if (!context) {
    throw new Error("useCart must be used within a CartProvider");
  }
  return context;
};
```