Day 4 - Dynamic Frontend Components - Foodtuck Marketplace

1. Introduction

The Foodtuck Marketplace is a dynamic, feature-rich eCommerce platform designed to provide a seamless online shopping experience. The goal of this phase was to build and integrate various frontend components to create a fully functional, interactive, and efficient user interface using **React, Next.js, and TypeScript**.

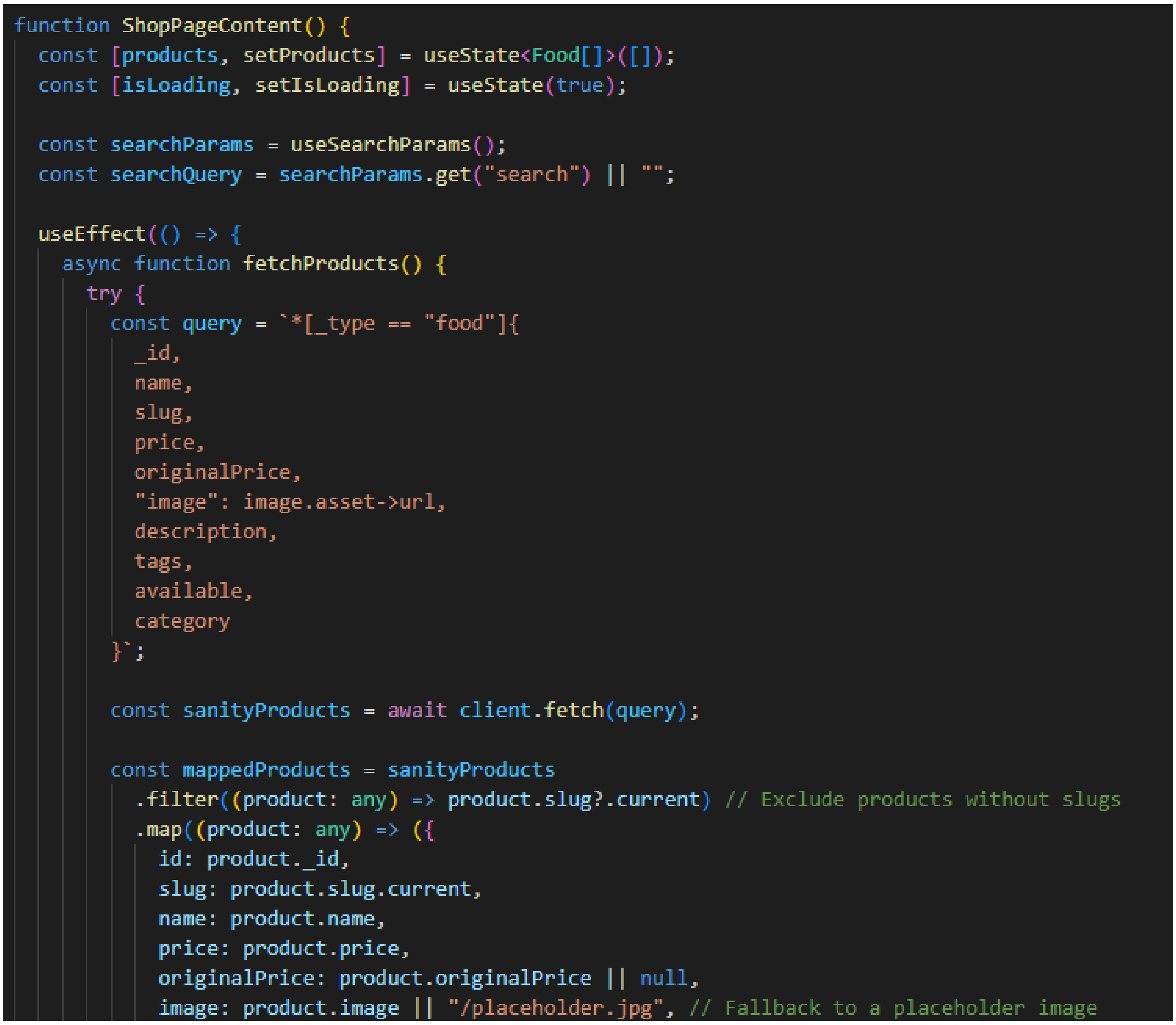
This document outlines the development approach, challenges faced, and best practices followed while implementing core frontend features, including **product listing, product detail pages, search and filtering, shopping cart, checkout, and user interactions**.

2. Functional Components Developed

2.1 Product Listing Component

The **Product Listing Page** dynamically fetches product data from **Sanity CMS**, filters it based on user input, and renders the items in a structured layout. It includes features such as:

- Dynamic product fetching with **API integration**.
- **Search functionality** that updates results in real-time.
- **Category filters** for refining product selection.
- Pagination for browsing through multiple pages.
- Optimized rendering using **useEffect** and **useState**.

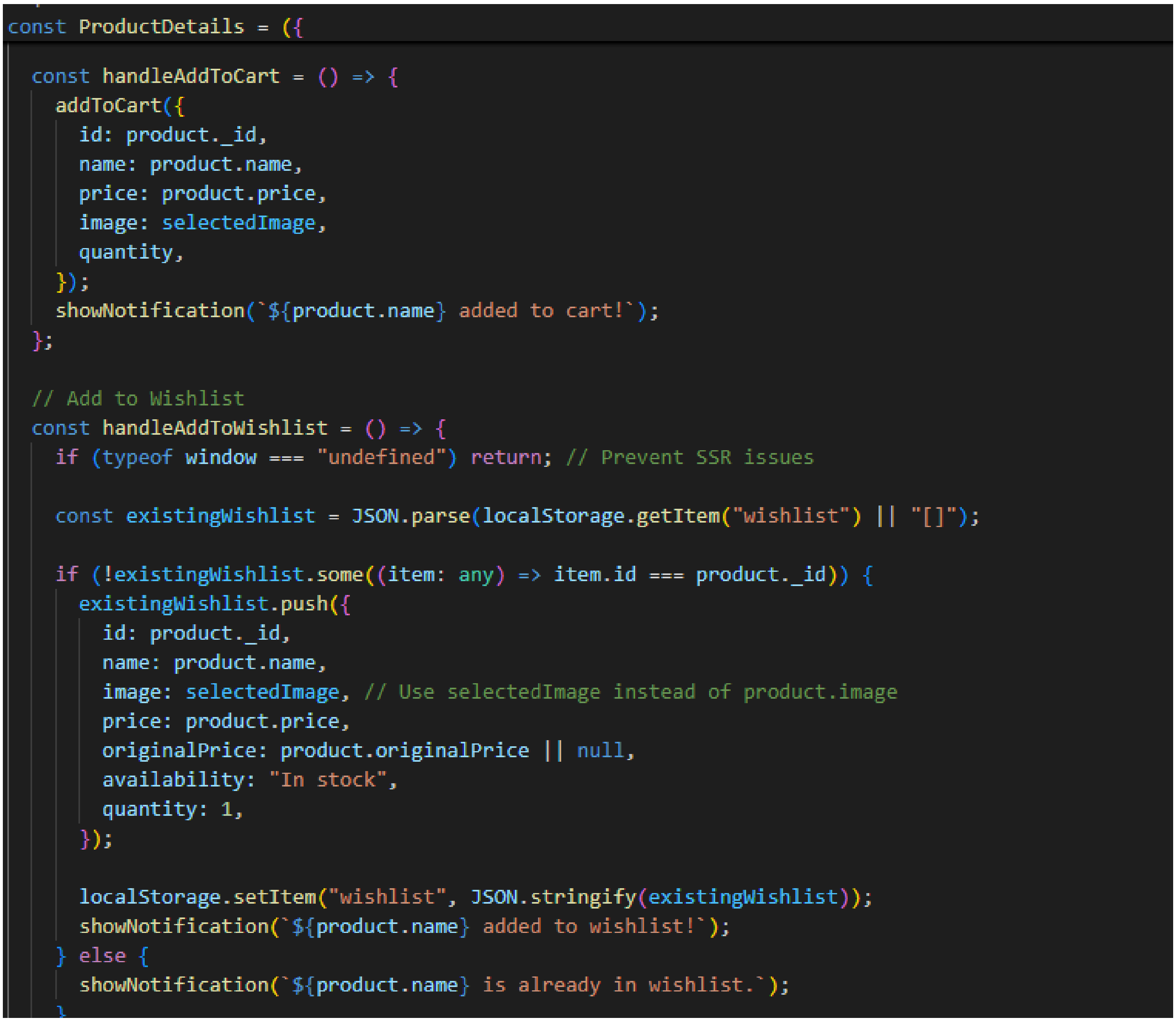


```
function ShopPageContent() {
  const [products, setProducts] = useState<Food[]>([]);
  const [isLoading, setIsLoading] = useState(true);

  const searchParams = useSearchParams();
  const searchQuery = searchParams.get("search") || "";

  useEffect(() => {
    async function fetchProducts() {
      try {
        const query = `*[_type == "food"]{
          _id,
          name,
          slug,
          price,
          originalPrice,
          "image": image.asset->url,
          description,
          tags,
          available,
          category
        }`;

        const sanityProducts = await client.fetch(query);

        const mappedProducts = sanityProducts
          .filter((product: any) => product.slug?.current) // Exclude products without slugs
          .map((product: any) => ({
            id: product._id,
            slug: product.slug.current,
            name: product.name,
            price: product.price,
            originalPrice: product.originalPrice || null,
            image: product.image || "/placeholder.jpg", // Fallback to a placeholder image
```

2.2 Product Detail Component

Each product has a dedicated detail page with:

- Accurate routing using **Next.js dynamic routes** (pages/product/[slug]/page.tsx).
- Display of detailed product information, including pricing, availability, and images.
- **Functionality** to add products to the cart or wishlist.
- A section for **related products** to encourage cross-selling.
- Integration with a **product comparison** feature.

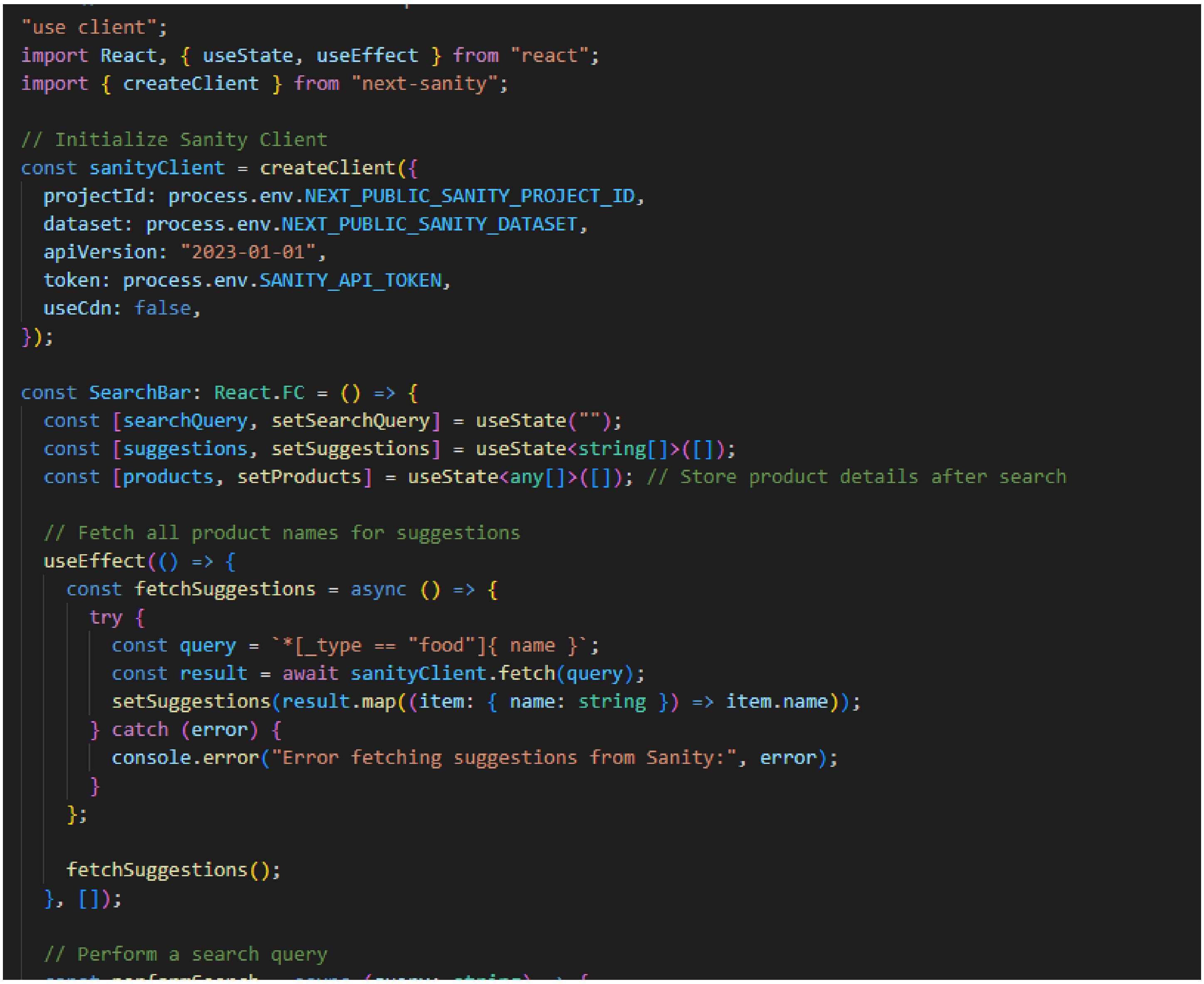


```
const ProductDetails = ({

  const handleAddToCart = () => {
    addToCart({
      id: product._id,
      name: product.name,
      price: product.price,
      image: selectedImage,
      quantity,
    });
    showNotification(`${product.name} added to cart!`);
  };

  // Add to Wishlist
  const handleAddToWishlist = () => {
    if (typeof window === "undefined") return; // Prevent SSR issues

    const existingWishlist = JSON.parse(localStorage.getItem("wishlist") || "[]");

    if (!existingWishlist.some((item: any) => item.id === product._id)) {
      existingWishlist.push({
        id: product._id,
        name: product.name,
        image: selectedImage, // Use selectedImage instead of product.image
        price: product.price,
        originalPrice: product.originalPrice || null,
        availability: "In stock",
        quantity: 1,
      });

      localStorage.setItem("wishlist", JSON.stringify(existingWishlist));
      showNotification(`${product.name} added to wishlist!`);
    } else {
      showNotification(`${product.name} is already in wishlist.`);
```

2.3 Search Bar Component

The **Search Bar** allows users to search for products dynamically, filtering through the product database and displaying instant results. Key features include:

- **Real-time filtering** of search queries.
- **Optimized API calls** to avoid unnecessary re-renders.
- **Auto-suggestions** using indexed product names.
- Efficient **state management** using useState.

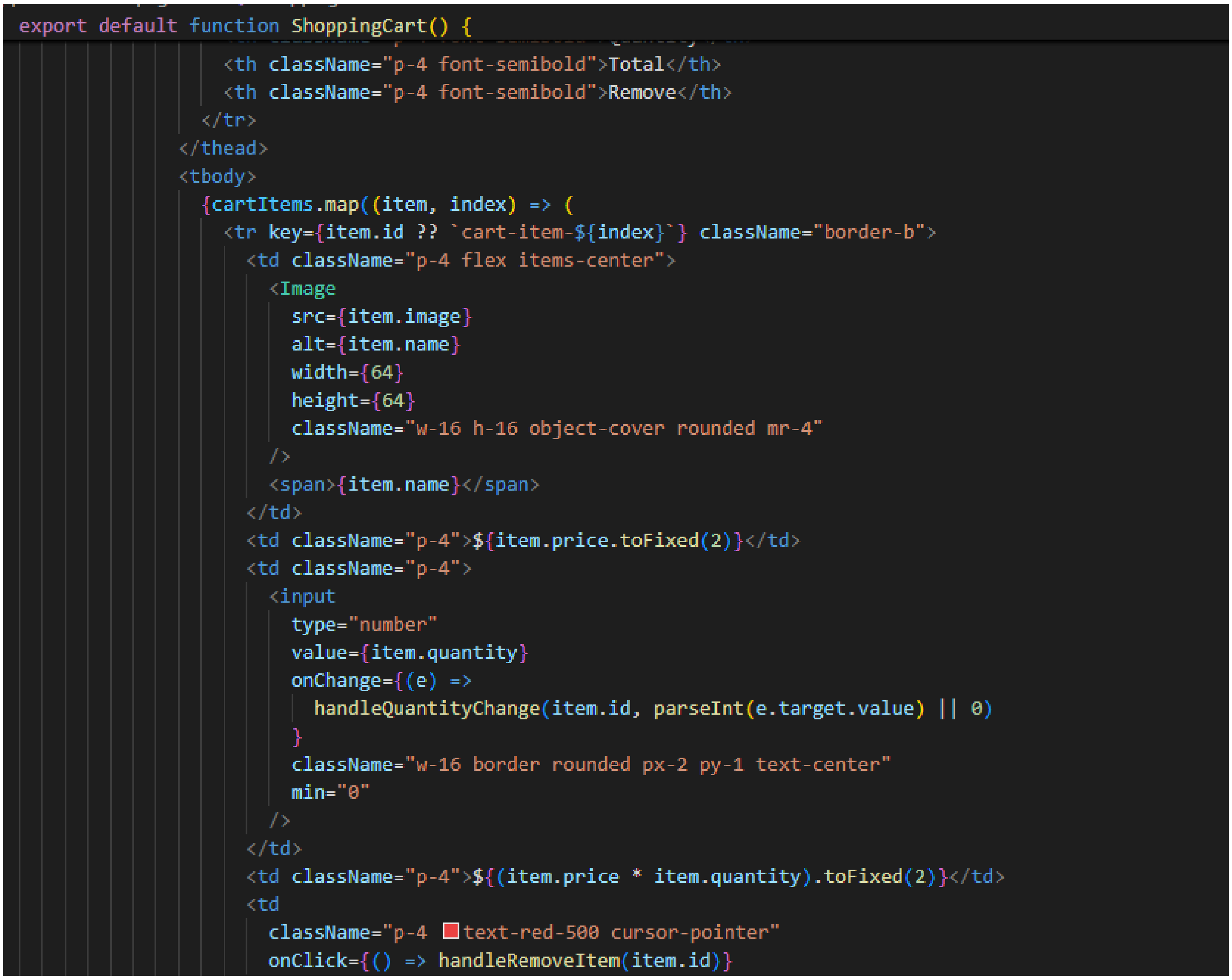


```
"use client";
import React, { useState, useEffect } from "react";
import { createClient } from "next-sanity";

// Initialize Sanity Client
const sanityClient = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  apiVersion: "2023-01-01",
  token: process.env.SANITY_API_TOKEN,
  useCdn: false,
});

const SearchBar: React.FC = () => {
  const [searchQuery, setSearchQuery] = useState("");
  const [suggestions, setSuggestions] = useState<string[]>([]);
  const [products, setProducts] = useState<any[]>([]); // Store product details after search

  // Fetch all product names for suggestions
  useEffect(() => {
    const fetchSuggestions = async () => {
      try {
        const query = `*[_type == "food"]{ name }`;
        const result = await sanityClient.fetch(query);
        setSuggestions(result.map((item: { name: string }) => item.name));
      } catch (error) {
        console.error("Error fetching suggestions from Sanity:", error);
      }
    };

    fetchSuggestions();
  }, []);

  // Perform a search query
```

2.4 Cart Component

The **Cart Component** is essential for managing user-selected products. It ensures a smooth checkout experience with:

- **Persistent cart storage** using localStorage.
- **Quantity management**, allowing users to adjust the number of items.
- **Real-time price calculation** including discounts and shipping costs.
- **Seamless transition** to the checkout page.

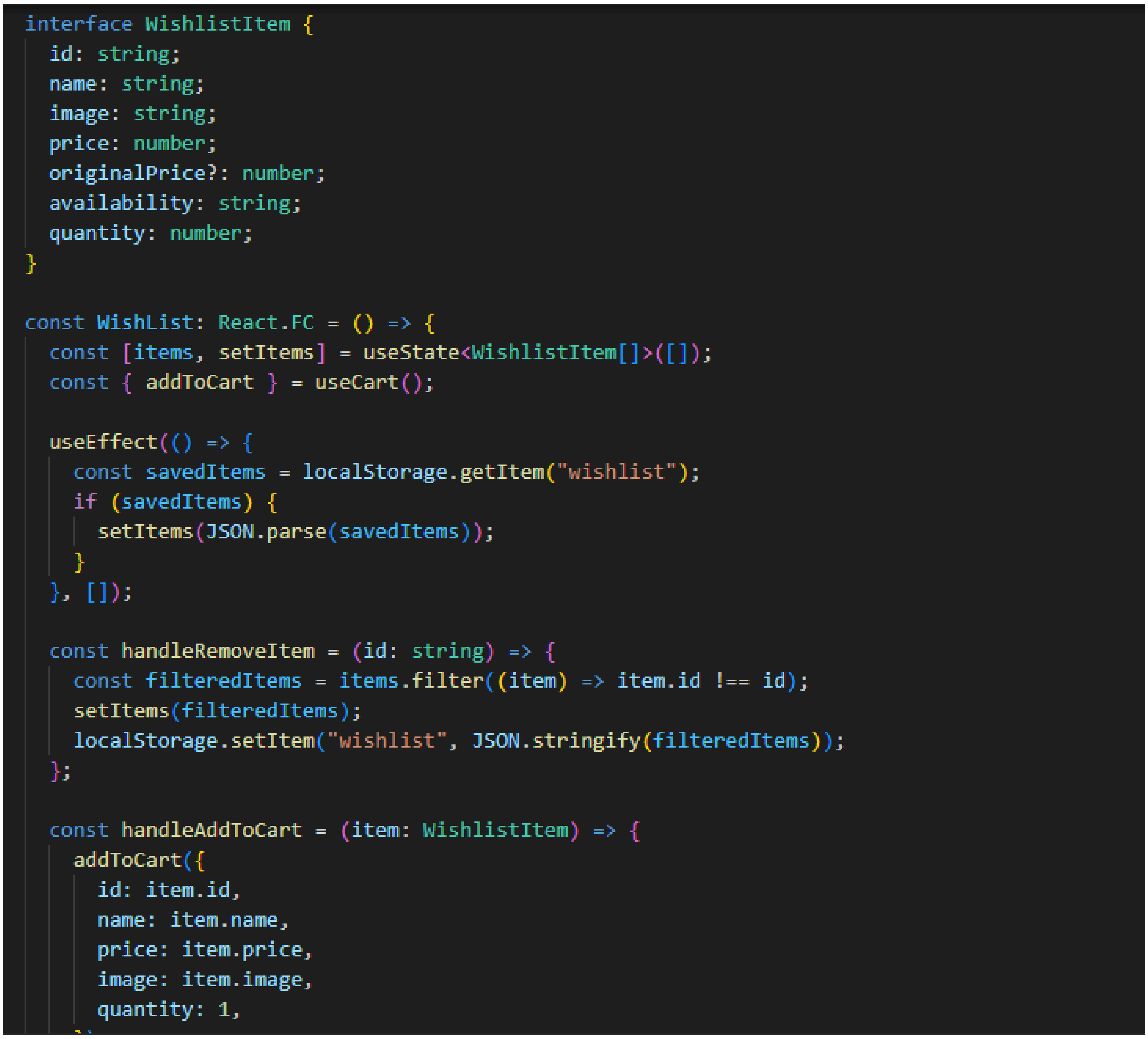


```
export default function ShoppingCart() {
                  <th className="p-4 font-semibold">Total</th>
                  <th className="p-4 font-semibold">Remove</th>
                </tr>
              </thead>
              <tbody>
                {cartItems.map((item, index) => (
                  <tr key={item.id ?? `cart-item-${index}`} className="border-b">
                    <td className="p-4 flex items-center">
                      <Image
                        src={item.image}
                        alt={item.name}
                        width={64}
                        height={64}
                        className="w-16 h-16 object-cover rounded mr-4"
                      />
                      <span>{item.name}</span>
                    </td>
                    <td className="p-4">${item.price.toFixed(2)}</td>
                    <td className="p-4">
                      <input
                        type="number"
                        value={item.quantity}
                        onChange={(e) =>
                          handleQuantityChange(item.id, parseInt(e.target.value) || 0)
                        }
                        className="w-16 border rounded px-2 py-1 text-center"
                        min="0"
                      />
                    </td>
                    <td className="p-4">${(item.price * item.quantity).toFixed(2)}</td>
                    <td
                      className="p-4 text-red-500 cursor-pointer"
                      onClick={() => handleRemoveItem(item.id)}
```

2.5 Wishlist Component

Users can save their favorite products for later using the **Wishlist Component**:

- Items are stored in **localStorage** for persistence.
- Integration with the **Cart Component** for easy addition to checkout.
- A clean UI displaying saved products.

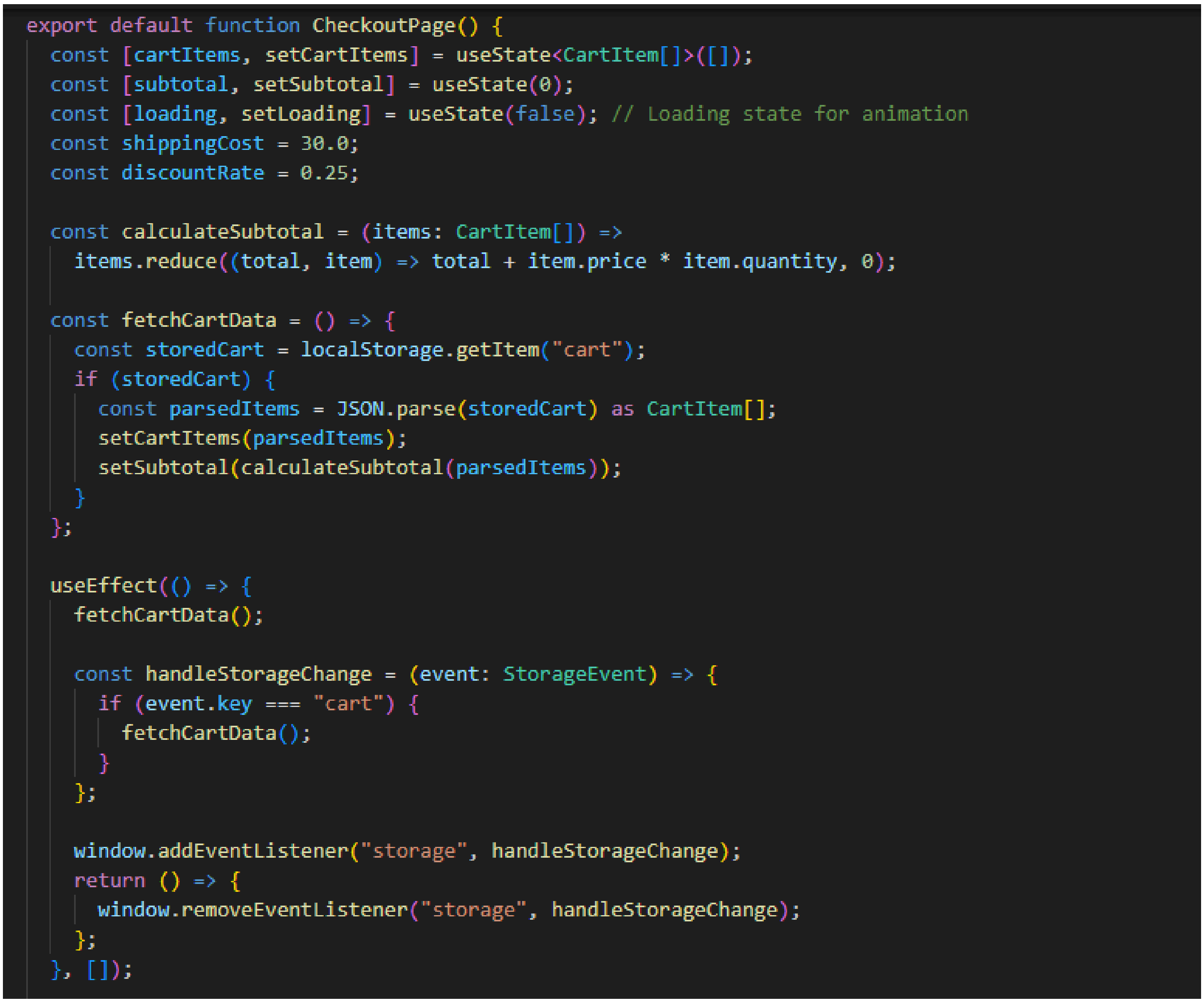


```tsx
interface WishlistItem {
  id: string;
  name: string;
  image: string;
  price: number;
  originalPrice?: number;
  availability: string;
  quantity: number;
}

const WishList: React.FC = () => {
  const [items, setItems] = useState<WishlistItem[]>([]);
  const { addToCart } = useCart();

  useEffect(() => {
    const savedItems = localStorage.getItem("wishlist");
    if (savedItems) {
      setItems(JSON.parse(savedItems));
    }
  }, []);

  const handleRemoveItem = (id: string) => {
    const filteredItems = items.filter((item) => item.id !== id);
    setItems(filteredItems);
    localStorage.setItem("wishlist", JSON.stringify(filteredItems));
  };

  const handleAddToCart = (item: WishlistItem) => {
    addToCart({
      id: item.id,
      name: item.name,
      price: item.price,
      image: item.image,
      quantity: 1,
```

2.6 Checkout Flow Component

The **Checkout Flow** guides users through the final purchase process, ensuring a smooth experience:

- **Shipping details input form** with validation.
- **Real-time order summary** updates based on cart contents.
- **Order processing animation** using Framer Motion.
- **Redirection to Order Confirmation Page** after successful checkout.

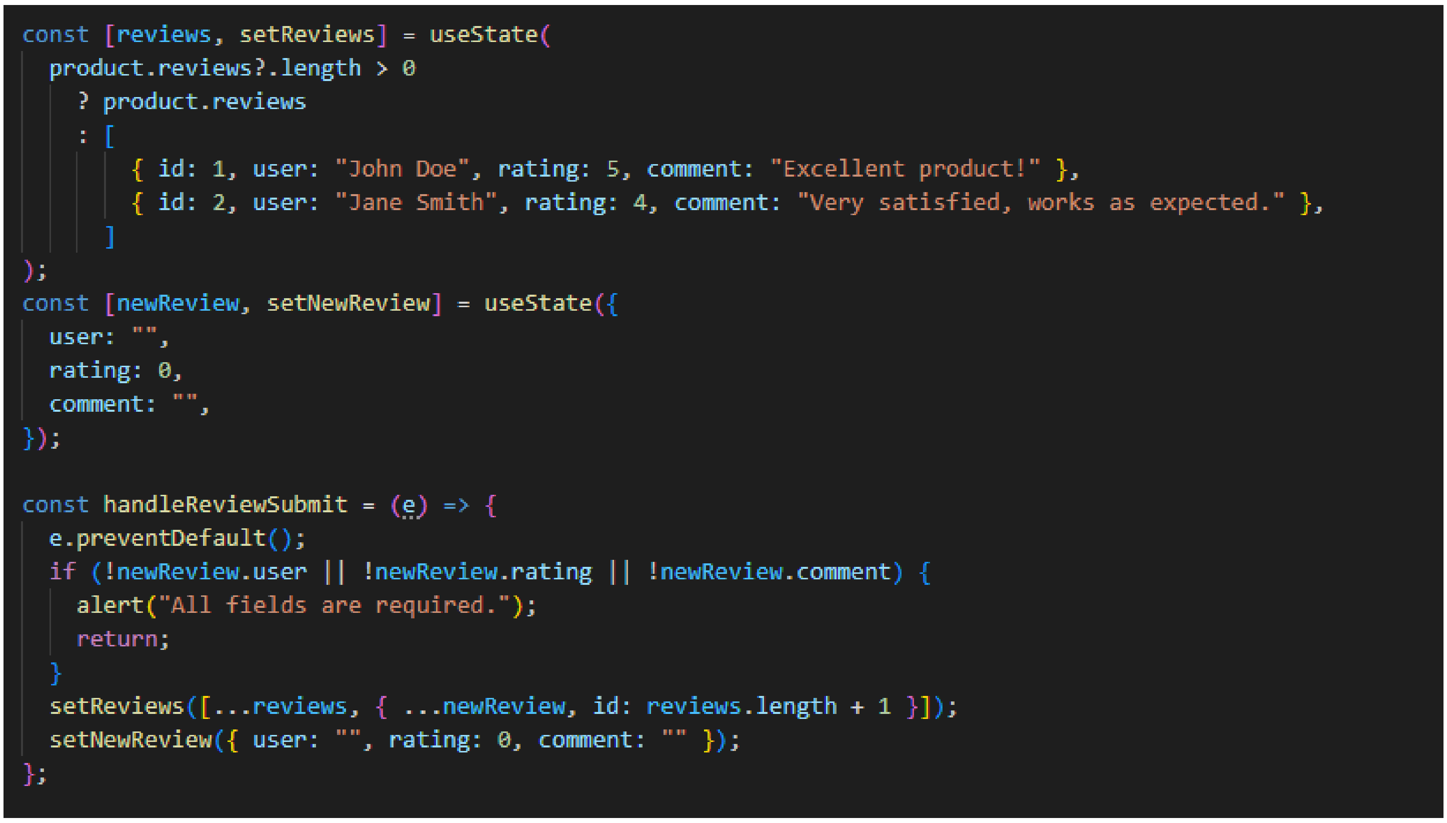


```
export default function CheckoutPage() {
  const [cartItems, setCartItems] = useState<CartItem[]>([]);
  const [subtotal, setSubtotal] = useState(0);
  const [loading, setLoading] = useState(false); // Loading state for animation
  const shippingCost = 30.0;
  const discountRate = 0.25;

  const calculateSubtotal = (items: CartItem[]) =>
    items.reduce((total, item) => total + item.price * item.quantity, 0);

  const fetchCartData = () => {
    const storedCart = localStorage.getItem("cart");
    if (storedCart) {
      const parsedItems = JSON.parse(storedCart) as CartItem[];
      setCartItems(parsedItems);
      setSubtotal(calculateSubtotal(parsedItems));
    }
  };

  useEffect(() => {
    fetchCartData();

    const handleStorageChange = (event: StorageEvent) => {
      if (event.key === "cart") {
        fetchCartData();
      }
    };

    window.addEventListener("storage", handleStorageChange);
    return () => {
      window.removeEventListener("storage", handleStorageChange);
    };
  }, []);
```

2.7 Reviews and Ratings Component

To enhance user engagement, the **Reviews and Ratings** feature allows customers to provide feedback:

- **Star rating system** (1-5 scale).
- Review submission with **form validation**.
- Display of **past customer reviews**.

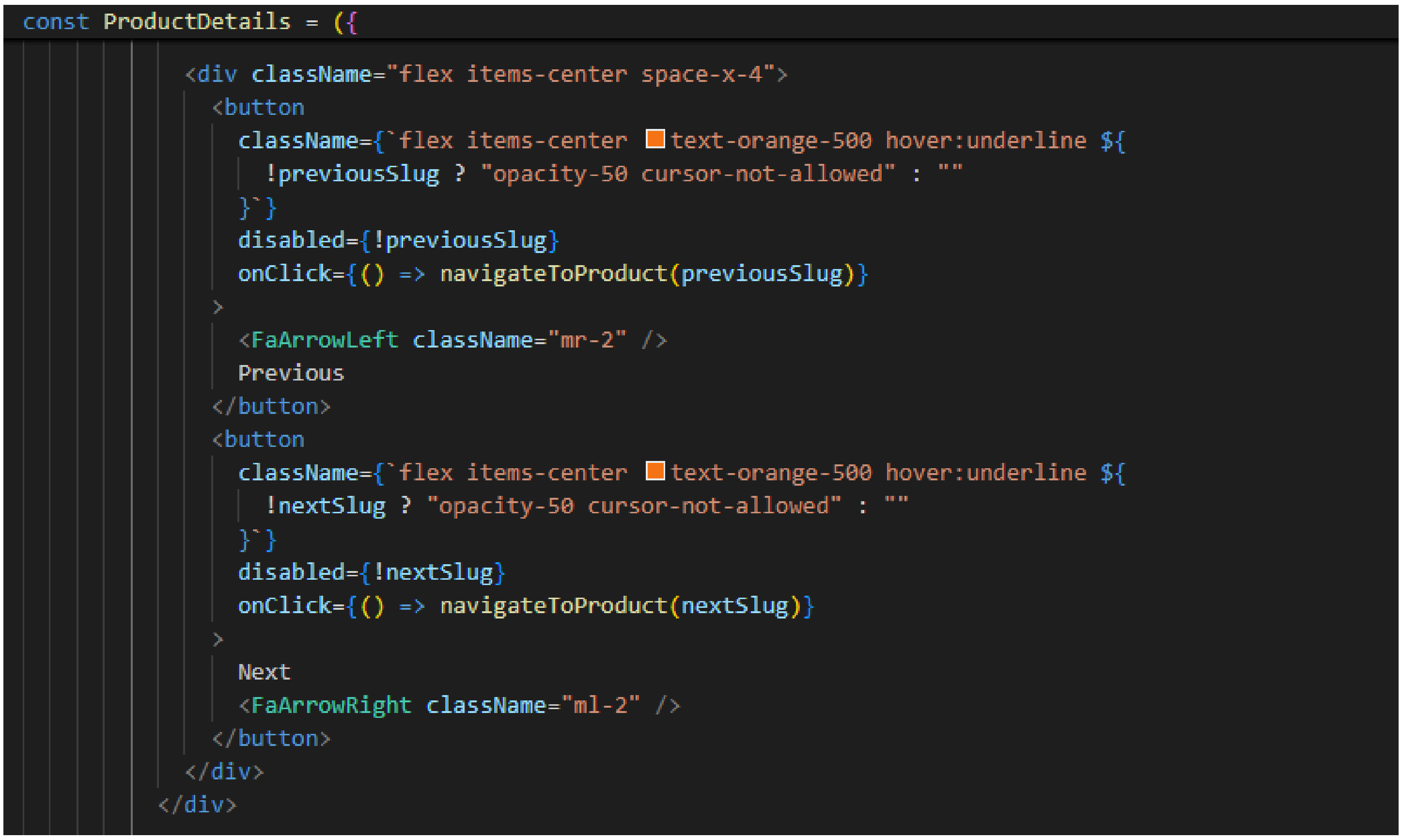


```
const [reviews, setReviews] = useState(
  product.reviews?.length > 0
    ? product.reviews
    : [
        { id: 1, user: "John Doe", rating: 5, comment: "Excellent product!" },
        { id: 2, user: "Jane Smith", rating: 4, comment: "Very satisfied, works as expected." },
      ]
);
const [newReview, setNewReview] = useState({
  user: "",
  rating: 0,
  comment: "",
});

const handleReviewSubmit = (e) => {
  e.preventDefault();
  if (!newReview.user || !newReview.rating || !newReview.comment) {
    alert("All fields are required.");
    return;
  }
  setReviews([...reviews, { ...newReview, id: reviews.length + 1 }]);
  setNewReview({ user: "", rating: 0, comment: "" });
};
```

2.8 Pagination Component

Large product databases require structured navigation. The **Pagination Component** enables:

- **Efficient page navigation** using state-controlled offsets.
- **Dynamic page buttons** based on product count.
- **Performance optimization** for fast switching between products.

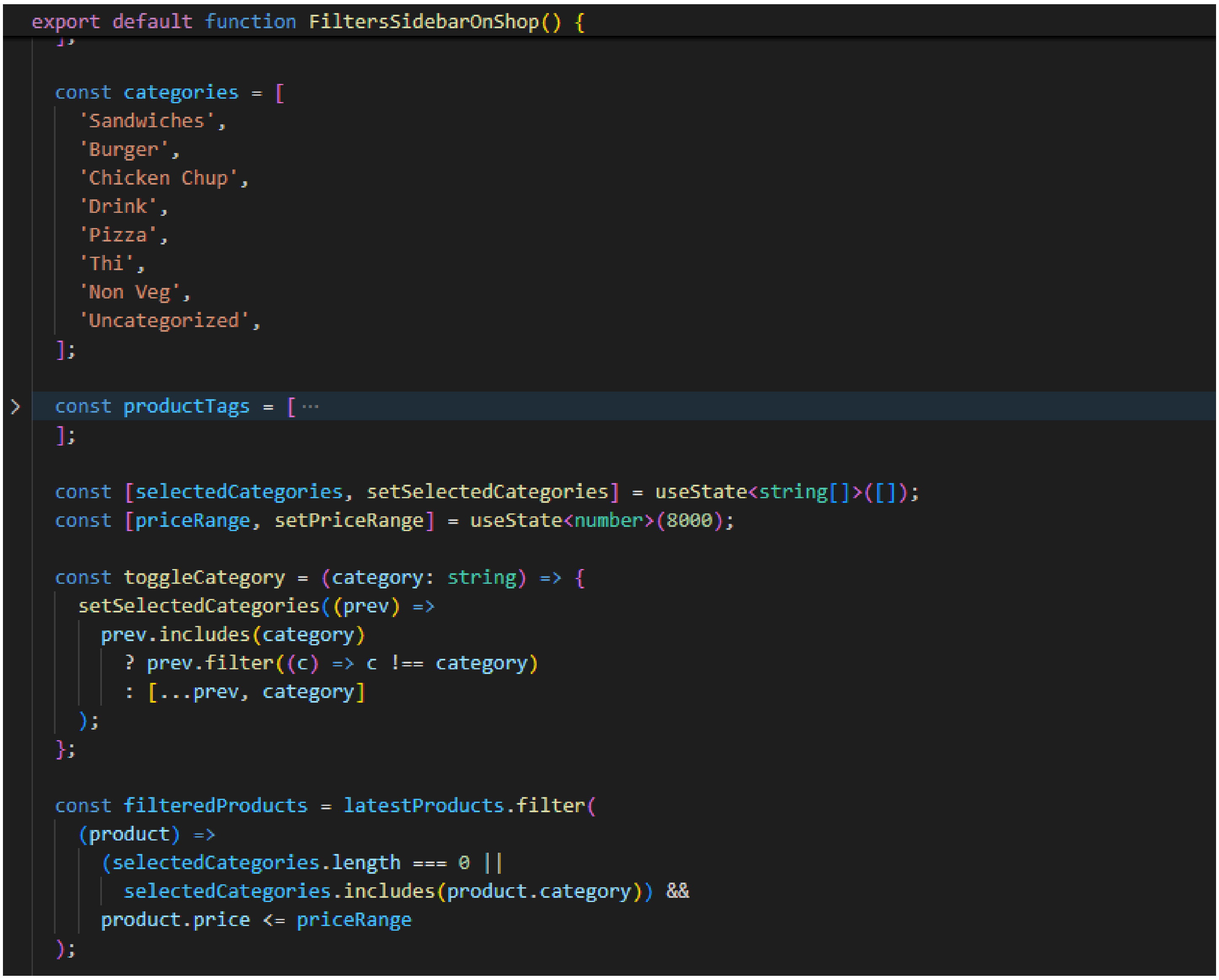


```
const ProductDetails = ({

            <div className="flex items-center space-x-4">
              <button
                className={`flex items-center text-orange-500 hover:underline ${
                  !previousSlug ? "opacity-50 cursor-not-allowed" : ""
                }`}
                disabled={!previousSlug}
                onClick={() => navigateToProduct(previousSlug)}
              >
                <FaArrowLeft className="mr-2" />
                Previous
              </button>
              <button
                className={`flex items-center text-orange-500 hover:underline ${
                  !nextSlug ? "opacity-50 cursor-not-allowed" : ""
                }`}
                disabled={!nextSlug}
                onClick={() => navigateToProduct(nextSlug)}
              >
                Next
                <FaArrowRight className="ml-2" />
              </button>
            </div>
          </div>
```

2.9 Filter Panel Component

The **Filter Panel** refines search results based on category, price, and tags:

- **Multi-category selection** with checkbox inputs.
- **Price range filtering** using an interactive slider.
- **Tag-based product grouping**.

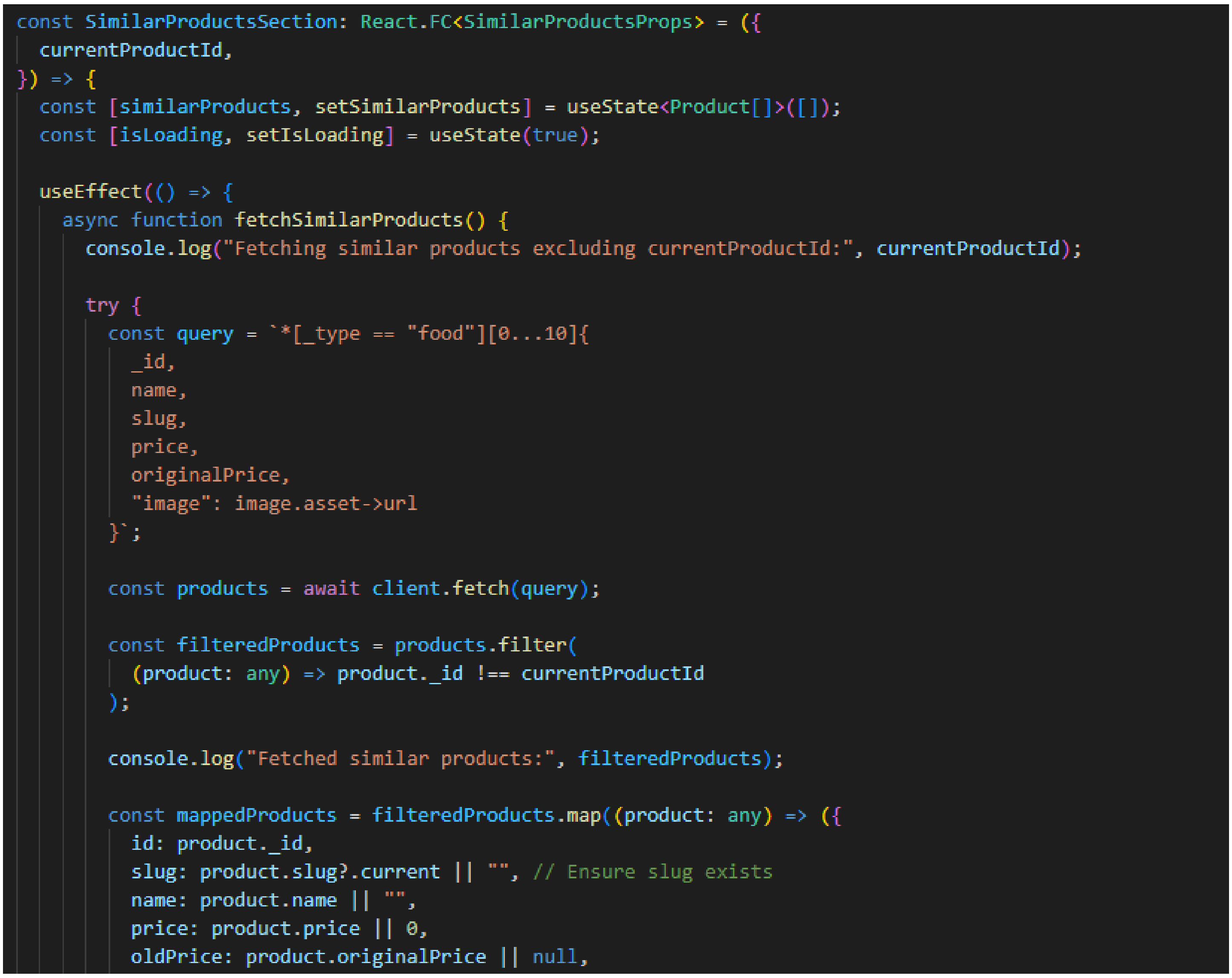


```
export default function FiltersSidebarOnShop() {
  ];

  const categories = [
    'Sandwiches',
    'Burger',
    'Chicken Chup',
    'Drink',
    'Pizza',
    'Thi',
    'Non Veg',
    'Uncategorized',
  ];

  const productTags = [···
  ];

  const [selectedCategories, setSelectedCategories] = useState<string[]>([]);
  const [priceRange, setPriceRange] = useState<number>(8000);

  const toggleCategory = (category: string) => {
    setSelectedCategories((prev) =>
      prev.includes(category)
        ? prev.filter((c) => c !== category)
        : [...prev, category]
    );
  };

  const filteredProducts = latestProducts.filter(
    (product) =>
      (selectedCategories.length === 0 ||
        selectedCategories.includes(product.category)) &&
      product.price <= priceRange
  );
```

2.10 Related Products Component

The **Related Products Section** recommends items based on user preferences:

- Fetches **similar products** from the database.
- Uses **Swiper.js** for a smooth scrolling experience.
- Encourages **cross-selling**.

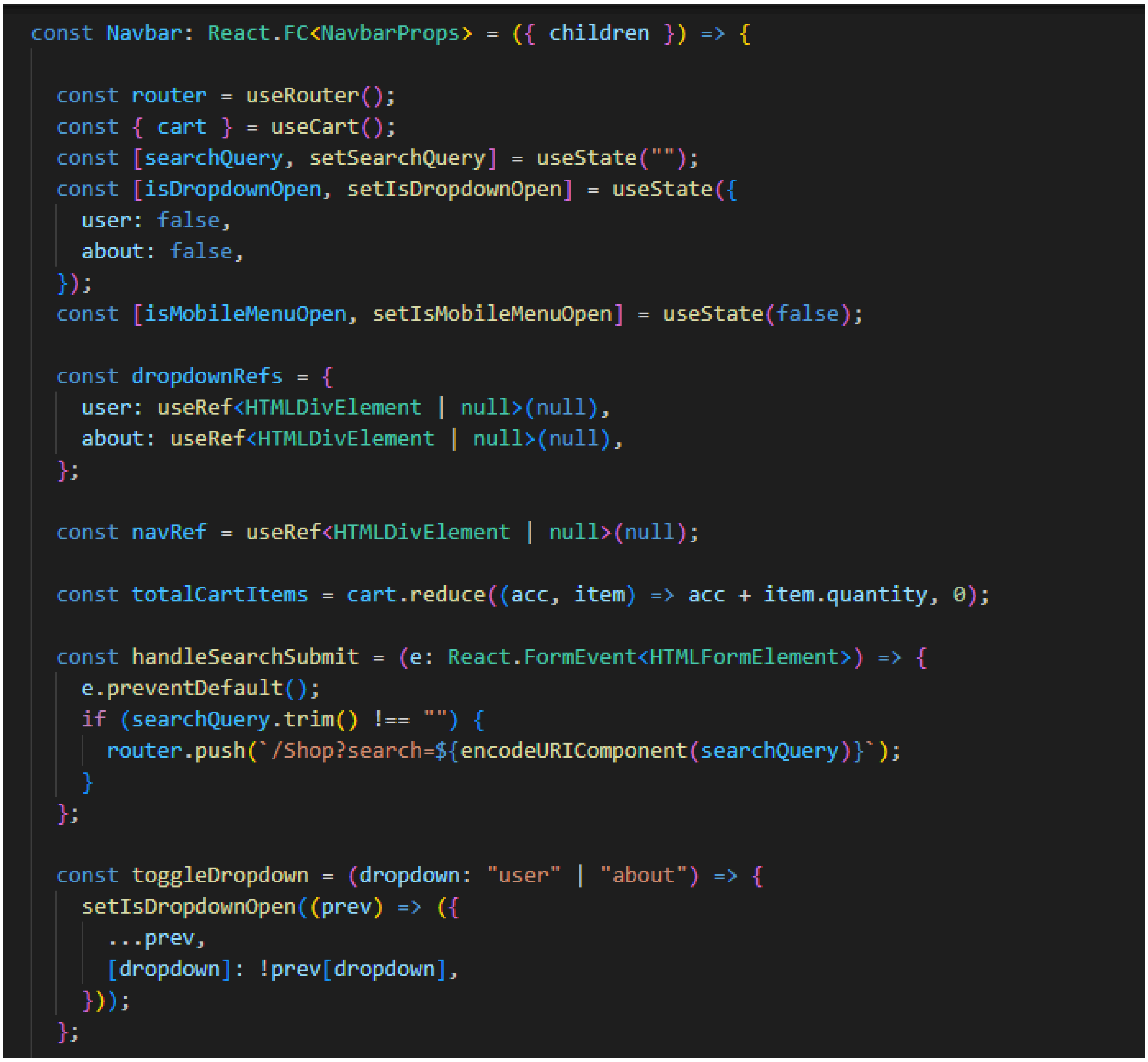


```
const SimilarProductsSection: React.FC<SimilarProductsProps> = ({
  currentProductId,
}) => {
  const [similarProducts, setSimilarProducts] = useState<Product[]>([]);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    async function fetchSimilarProducts() {
      console.log("Fetching similar products excluding currentProductId:", currentProductId);

      try {
        const query = `*[_type == "food"][0...10]{
          _id,
          name,
          slug,
          price,
          originalPrice,
          "image": image.asset->url
        }`;

        const products = await client.fetch(query);

        const filteredProducts = products.filter(
          (product: any) => product._id !== currentProductId
        );

        console.log("Fetched similar products:", filteredProducts);

        const mappedProducts = filteredProducts.map((product: any) => ({
          id: product._id,
          slug: product.slug?.current || "", // Ensure slug exists
          name: product.name || "",
          price: product.price || 0,
          oldPrice: product.originalPrice || null,
```

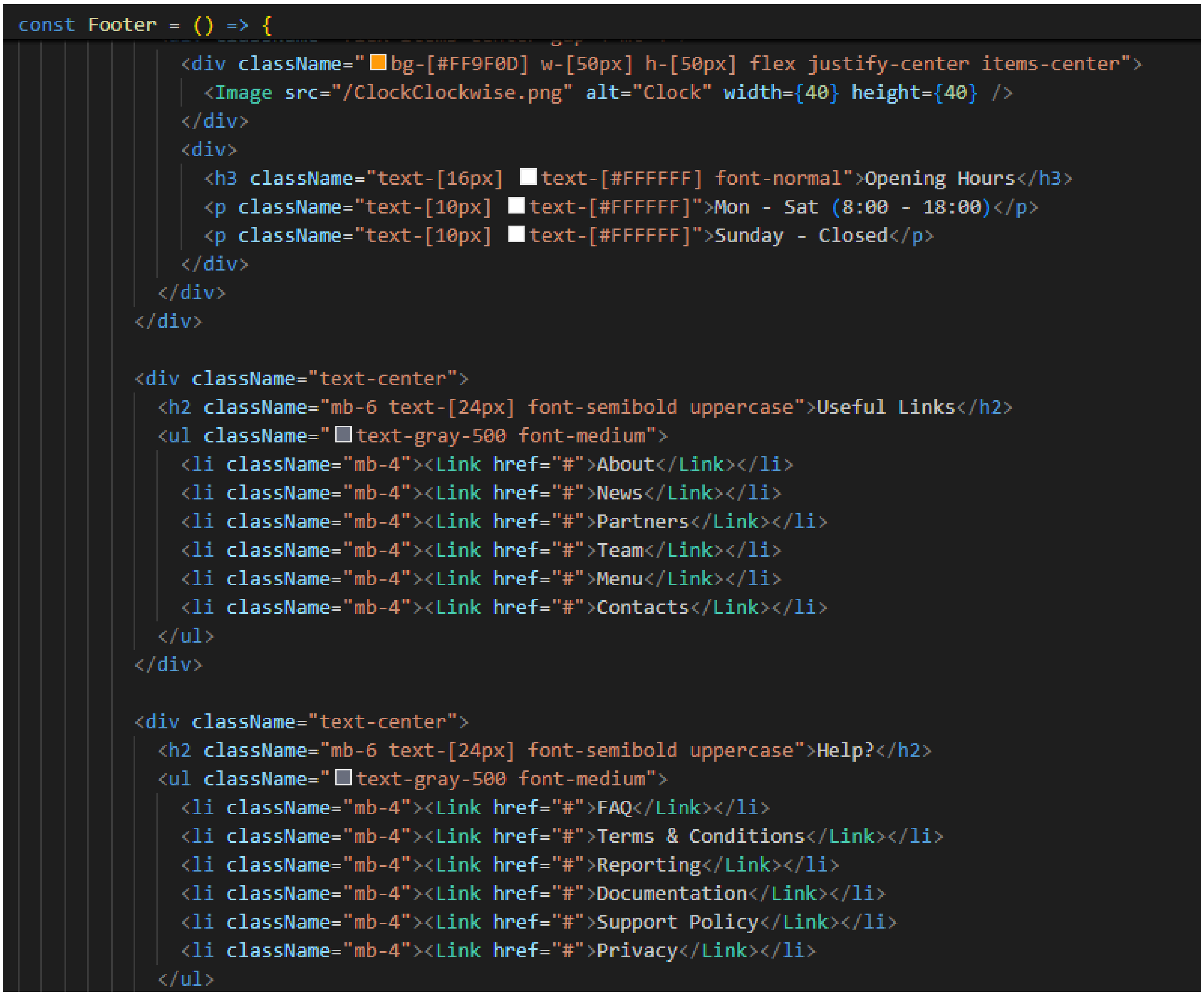
2.11 Footer and Header Components

The **Navbar and Footer** provide intuitive site navigation:

- **Dynamic cart count** in the navbar.
- **Multi-layered dropdowns** for categories and account options.
- **Responsive layout** optimized for mobile and desktop.



```
const Navbar: React.FC<NavbarProps> = ({ children }) => {

  const router = useRouter();
  const { cart } = useCart();
  const [searchQuery, setSearchQuery] = useState("");
  const [isDropdownOpen, setIsDropdownOpen] = useState({
    user: false,
    about: false,
  });
  const [isMobileMenuOpen, setIsMobileMenuOpen] = useState(false);

  const dropdownRefs = {
    user: useRef<HTMLDivElement | null>(null),
    about: useRef<HTMLDivElement | null>(null),
  };

  const navRef = useRef<HTMLDivElement | null>(null);

  const totalCartItems = cart.reduce((acc, item) => acc + item.quantity, 0);

  const handleSearchSubmit = (e: React.FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    if (searchQuery.trim() !== "") {
      router.push(`/Shop?search=${encodeURIComponent(searchQuery)}`);
    }
  };

  const toggleDropdown = (dropdown: "user" | "about") => {
    setIsDropdownOpen((prev) => ({
      ...prev,
      [dropdown]: !prev[dropdown],
    }));
  };
```

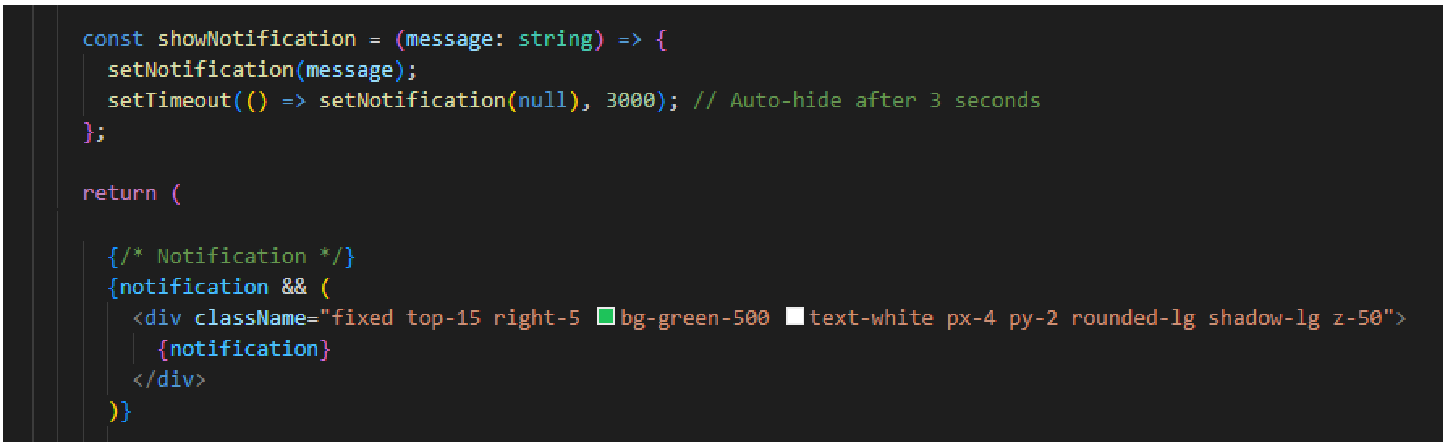


```
const Footer = () => {
              <div className="bg-[#FF9F0D] w-[50px] h-[50px] flex justify-center items-center">
                <Image src="/ClockClockwise.png" alt="Clock" width={40} height={40} />
              </div>
              <div>
                <h3 className="text-[16px] text-[#FFFFFF] font-normal">Opening Hours</h3>
                <p className="text-[10px] text-[#FFFFFF]">Mon - Sat (8:00 - 18:00)</p>
                <p className="text-[10px] text-[#FFFFFF]">Sunday - Closed</p>
              </div>
            </div>
          </div>

          <div className="text-center">
            <h2 className="mb-6 text-[24px] font-semibold uppercase">Useful Links</h2>
            <ul className="text-gray-500 font-medium">
              <li className="mb-4"><Link href="#">About</Link></li>
              <li className="mb-4"><Link href="#">News</Link></li>
              <li className="mb-4"><Link href="#">Partners</Link></li>
              <li className="mb-4"><Link href="#">Team</Link></li>
              <li className="mb-4"><Link href="#">Menu</Link></li>
              <li className="mb-4"><Link href="#">Contacts</Link></li>
            </ul>
          </div>

          <div className="text-center">
            <h2 className="mb-6 text-[24px] font-semibold uppercase">Help?</h2>
            <ul className="text-gray-500 font-medium">
              <li className="mb-4"><Link href="#">FAQ</Link></li>
              <li className="mb-4"><Link href="#">Terms & Conditions</Link></li>
              <li className="mb-4"><Link href="#">Reporting</Link></li>
              <li className="mb-4"><Link href="#">Documentation</Link></li>
              <li className="mb-4"><Link href="#">Support Policy</Link></li>
              <li className="mb-4"><Link href="#">Privacy</Link></li>
            </ul>
```

2.12 Notifications Component

A **real-time notification system** enhances user experience:

- **Success alerts** for added-to-cart and wishlist actions.
- **Error messages** for invalid inputs or unavailable products.
- **Auto-dismiss animations** for smooth UX.

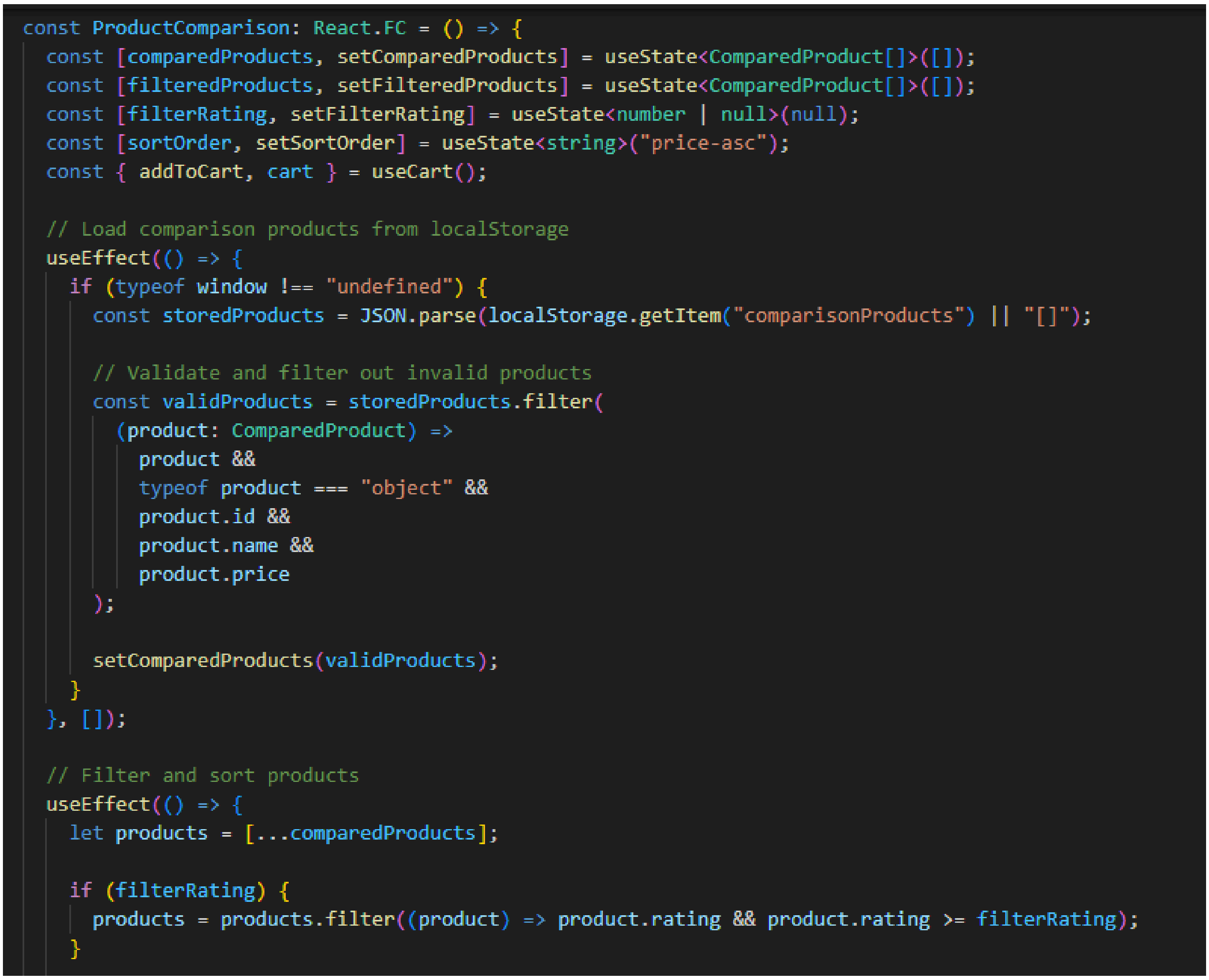


```
const showNotification = (message: string) => {
  setNotification(message);
  setTimeout(() => setNotification(null), 3000); // Auto-hide after 3 seconds
};

return (

  {/* Notification */}
  {notification && (
    <div className="fixed top-15 right-5 bg-green-500 text-white px-4 py-2 rounded-lg shadow-lg z-50">
      {notification}
    </div>
  )}
```

2.13 Product Comparison Component

Users can compare products based on price, features, and reviews:

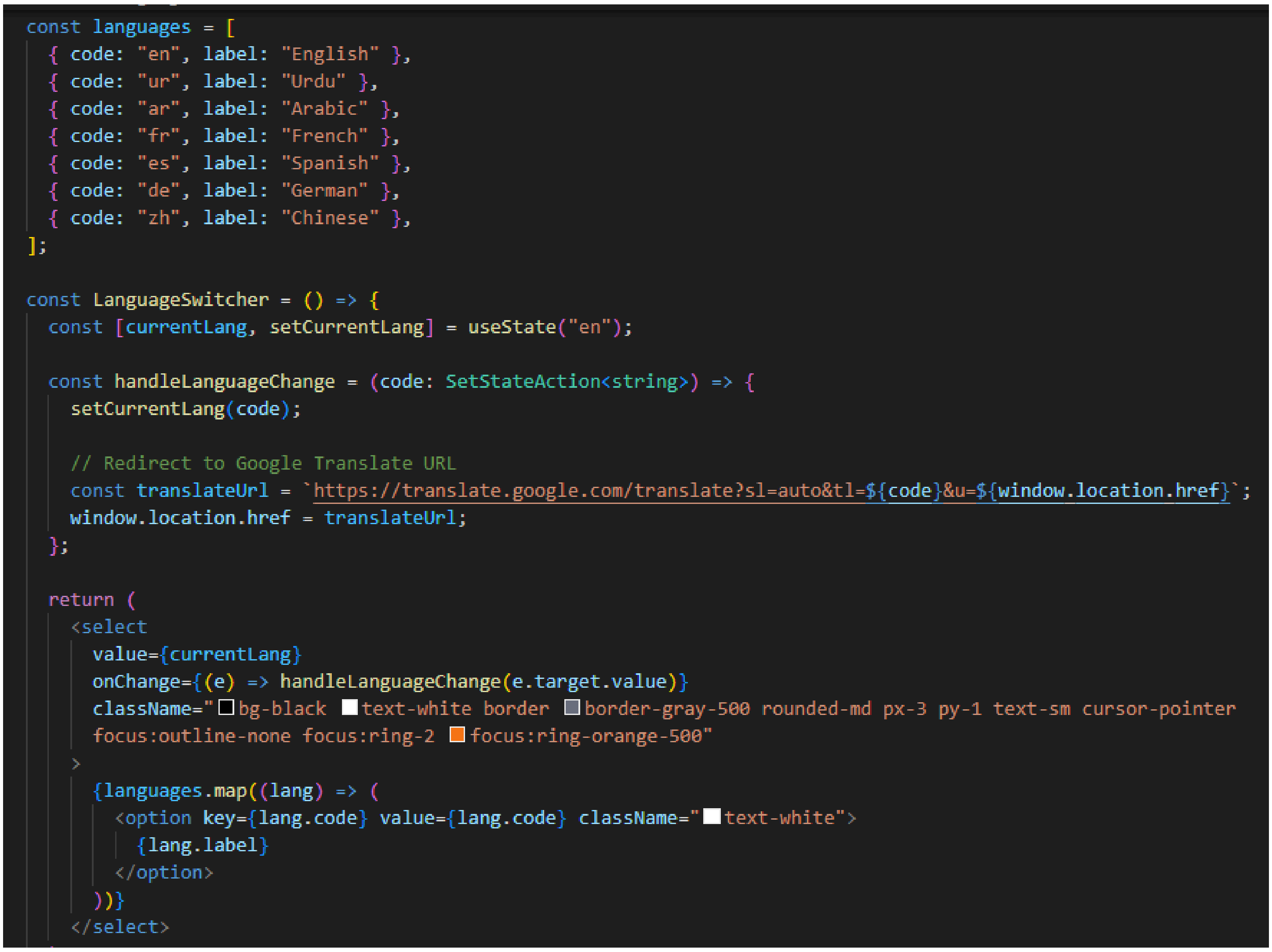
- **Side-by-side comparison UI**.
- **Sorting by attributes** such as price or rating.
- **Remove product from comparison** with a single click.



```
const ProductComparison: React.FC = () => {
  const [comparedProducts, setComparedProducts] = useState<ComparedProduct[]>([]);
  const [filteredProducts, setFilteredProducts] = useState<ComparedProduct[]>([]);
  const [filterRating, setFilterRating] = useState<number | null>(null);
  const [sortOrder, setSortOrder] = useState<string>("price-asc");
  const { addToCart, cart } = useCart();

  // Load comparison products from localStorage
  useEffect(() => {
    if (typeof window !== "undefined") {
      const storedProducts = JSON.parse(localStorage.getItem("comparisonProducts") || "[]");

      // Validate and filter out invalid products
      const validProducts = storedProducts.filter(
        (product: ComparedProduct) =>
          product &&
          typeof product === "object" &&
          product.id &&
          product.name &&
          product.price
      );

      setComparedProducts(validProducts);
    }
  }, []);

  // Filter and sort products
  useEffect(() => {
    let products = [...comparedProducts];

    if (filterRating) {
      products = products.filter((product) => product.rating && product.rating >= filterRating);
    }
```

2.14 Multi-Language Support Component

To cater to a global audience, the **Multi-Language Support Component** allows users to switch between languages dynamically.

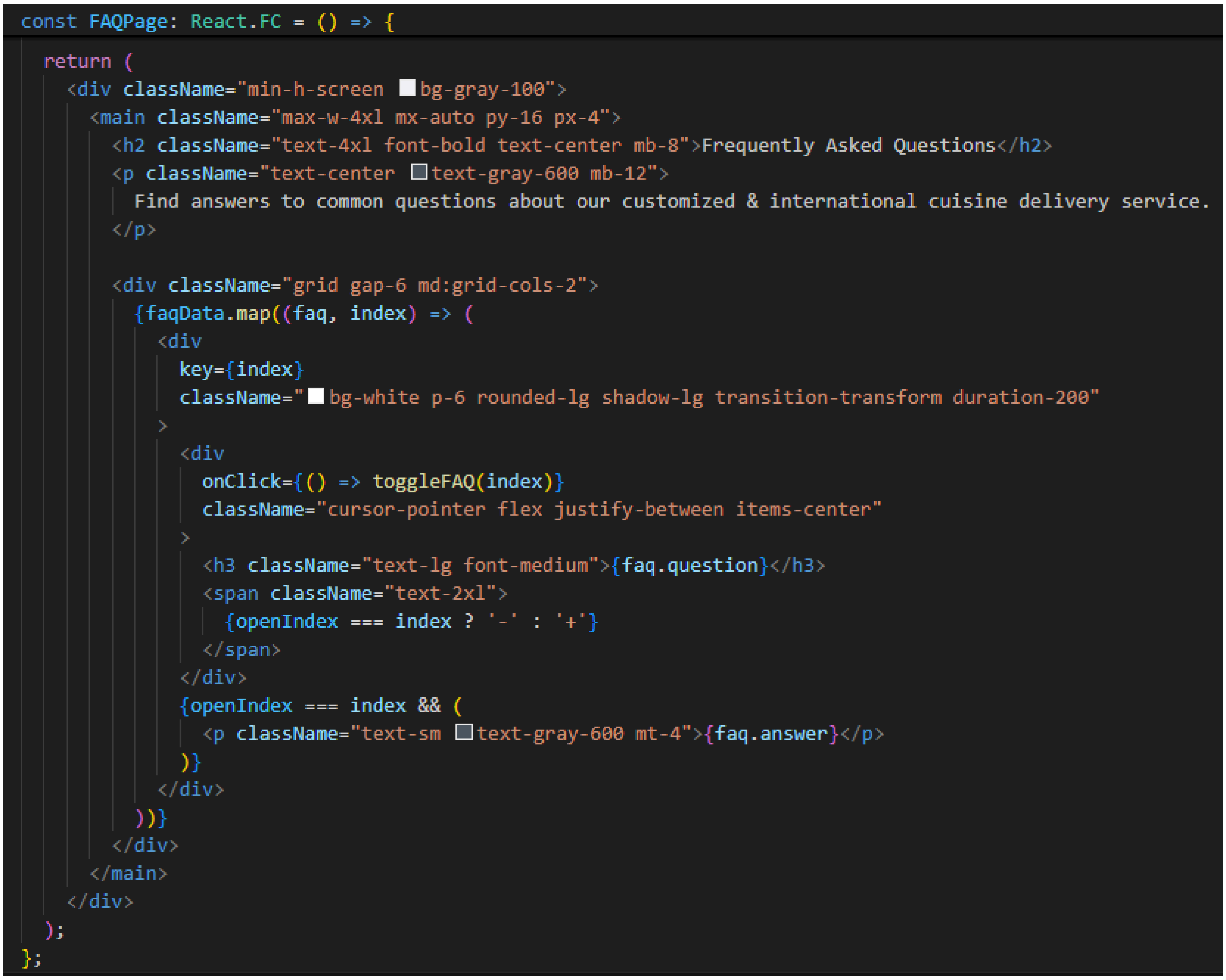


```
const languages = [
  { code: "en", label: "English" },
  { code: "ur", label: "Urdu" },
  { code: "ar", label: "Arabic" },
  { code: "fr", label: "French" },
  { code: "es", label: "Spanish" },
  { code: "de", label: "German" },
  { code: "zh", label: "Chinese" },
];

const LanguageSwitcher = () => {
  const [currentLang, setCurrentLang] = useState("en");

  const handleLanguageChange = (code: SetStateAction<string>) => {
    setCurrentLang(code);

    // Redirect to Google Translate URL
    const translateUrl = `https://translate.google.com/translate?sl=auto&tl=${code}&u=${window.location.href}`;
    window.location.href = translateUrl;
  };

  return (
    <select
      value={currentLang}
      onChange={(e) => handleLanguageChange(e.target.value)}
      className="bg-black text-white border border-gray-500 rounded-md px-3 py-1 text-sm cursor-pointer
      focus:outline-none focus:ring-2 focus:ring-orange-500"
    >
      {languages.map((lang) => (
        <option key={lang.code} value={lang.code} className="text-white">
          {lang.label}
        </option>
      ))}
    </select>
```

2.15 FAQ and Help Center Component

A structured **FAQ section** provides answers to common user queries:

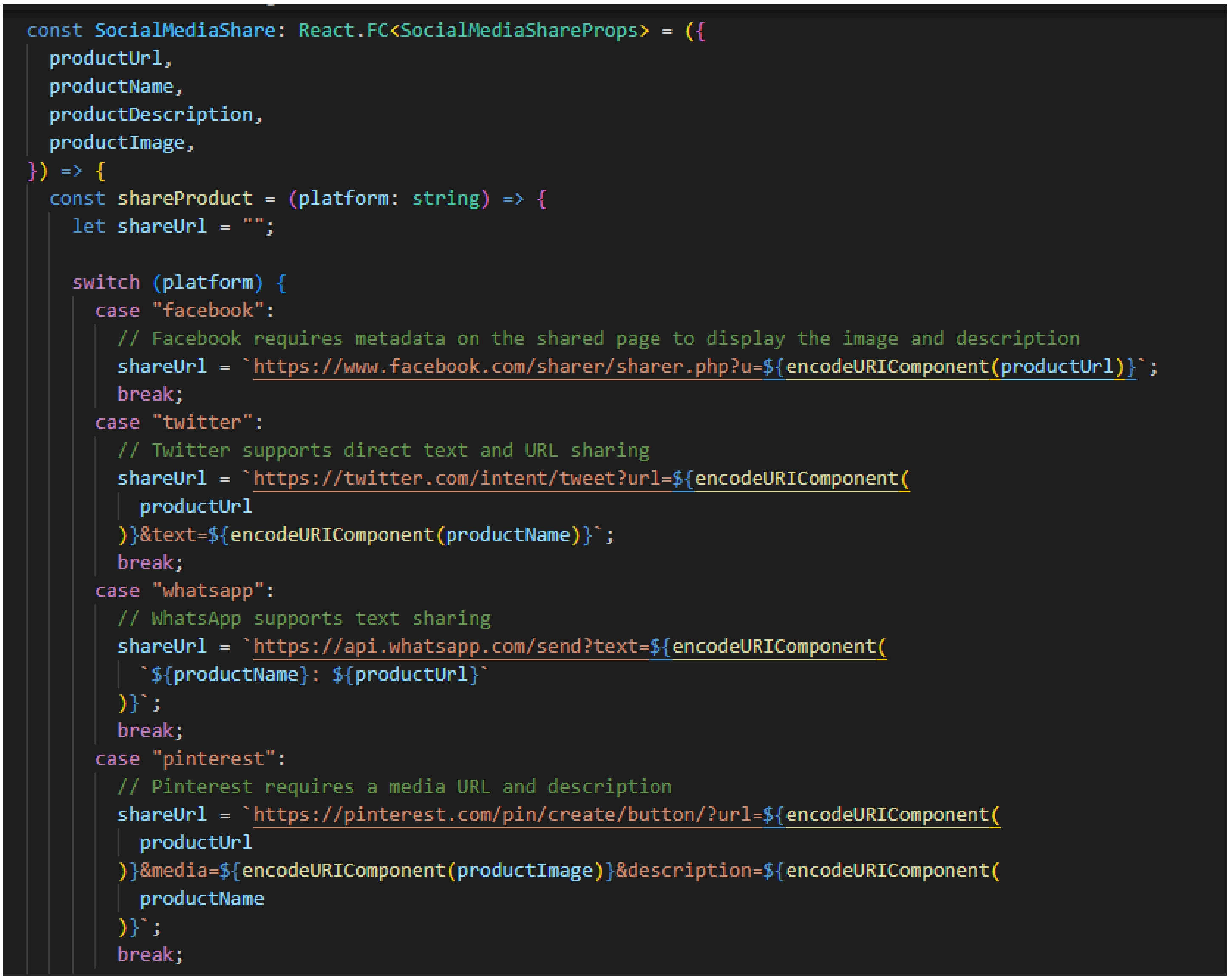
- **Accordion-style UI** for easy navigation.
- **Predefined questions and answers**.



```
const FAQPage: React.FC = () => {
  return (
    <div className="min-h-screen bg-gray-100">
      <main className="max-w-4xl mx-auto py-16 px-4">
        <h2 className="text-4xl font-bold text-center mb-8">Frequently Asked Questions</h2>
        <p className="text-center text-gray-600 mb-12">
          Find answers to common questions about our customized & international cuisine delivery service.
        </p>

        <div className="grid gap-6 md:grid-cols-2">
          {faqData.map((faq, index) => (
            <div
              key={index}
              className="bg-white p-6 rounded-lg shadow-lg transition-transform duration-200"
            >
              <div
                onClick={() => toggleFAQ(index)}
                className="cursor-pointer flex justify-between items-center"
              >
                <h3 className="text-lg font-medium">{faq.question}</h3>
                <span className="text-2xl">
                  {openIndex === index ? '-' : '+'}
                </span>
              </div>
              {openIndex === index && (
                <p className="text-sm text-gray-600 mt-4">{faq.answer}</p>
              )}
            </div>
          ))}
        </div>
      </main>
    </div>
  );
};
```

2.16 Social Media Sharing Component

The **Social Media Sharing Feature** allows users to share products on platforms like Facebook, Twitter, and WhatsApp.



```
const SocialMediaShare: React.FC<SocialMediaShareProps> = ({
  productUrl,
  productName,
  productDescription,
  productImage,
}) => {
  const shareProduct = (platform: string) => {
    let shareUrl = "";

    switch (platform) {
      case "facebook":
        // Facebook requires metadata on the shared page to display the image and description
        shareUrl = `https://www.facebook.com/sharer/sharer.php?u=${encodeURIComponent(productUrl)}`;
        break;
      case "twitter":
        // Twitter supports direct text and URL sharing
        shareUrl = `https://twitter.com/intent/tweet?url=${encodeURIComponent(
          productUrl
        )}&text=${encodeURIComponent(productName)}`;
        break;
      case "whatsapp":
        // WhatsApp supports text sharing
        shareUrl = `https://api.whatsapp.com/send?text=${encodeURIComponent(
          `${productName}: ${productUrl}`
        )}`;
        break;
      case "pinterest":
        // Pinterest requires a media URL and description
        shareUrl = `https://pinterest.com/pin/create/button/?url=${encodeURIComponent(
          productUrl
        )}&media=${encodeURIComponent(productImage)}&description=${encodeURIComponent(
          productName
        )}`;
        break;
```

3. Development Approach

3.1 Tools and Technologies Used

- **Next.js** for dynamic routing and performance optimization.
- **Sanity CMS** for scalable product data management.
- **TypeScript** for type safety and maintainability.
- **Tailwind CSS** for responsive UI design.
- **Framer Motion** for smooth animations.

3.2 Challenges Faced & Solutions

Challenges

- Managing large datasets efficiently.
- Preventing excessive API calls while ensuring real-time updates.
- Handling missing or incomplete product data.
- Creating a smooth user experience with animations and transitions.

- Implementing efficient form validation for checkout and user inputs.
- Preventing duplicate wishlist or cart entries.
- Managing dynamic filter states and search functionality effectively.
- Ensuring pagination does not affect user experience negatively.
- Handling smooth navigation between product detail pages and related products.
- Implementing language support without performance drawbacks.

Solutions

- Used pagination and optimized API queries to handle large datasets.
- Implemented debouncing to limit unnecessary API calls and enhance search efficiency.
- Added fallback placeholders for missing product data to improve UI reliability.
- Integrated Framer Motion for animations to provide smooth transitions.
- Applied robust input validation techniques for checkout and form fields.
- Checked for existing items in wishlist/cart before adding to prevent duplicates.
- Utilized efficient state management strategies to keep filters responsive and user-friendly.
- Developed an offset-based pagination approach to minimize re-renders and enhance performance.
- Optimized routing for seamless navigation between product pages and related items.
- Integrated language translation with minimal overhead to maintain high performance.

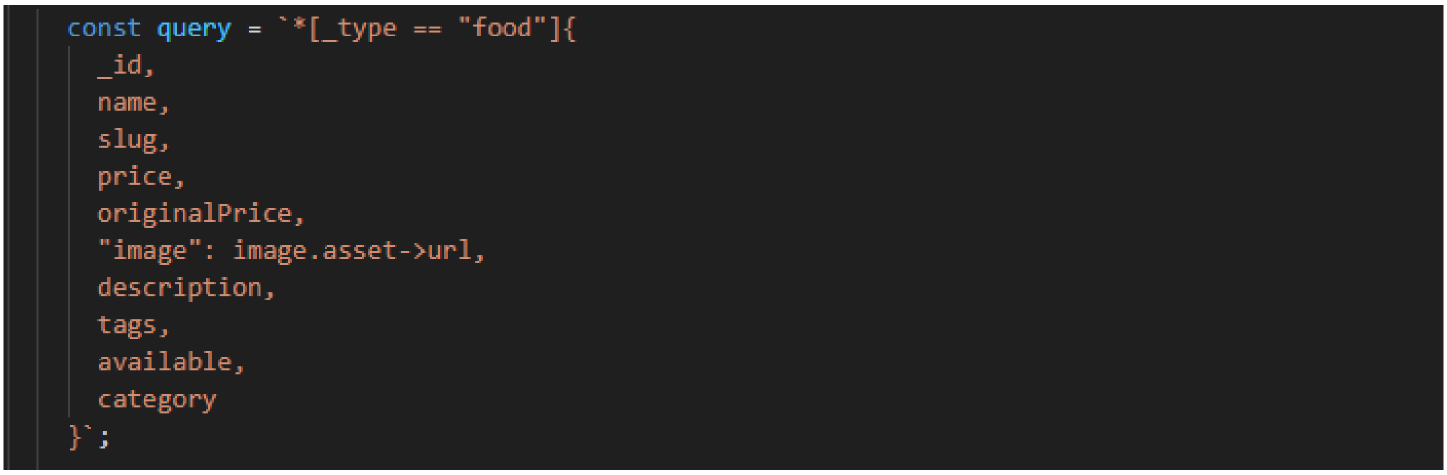
3.3 Best Practices Followed

- **Component Reusability:** Modularized UI elements for scalability.
- **Error Handling:** Implemented try-catch blocks for API requests.

4. Technical Documentation

4.1 Development Approach

Sanity CMS Integration



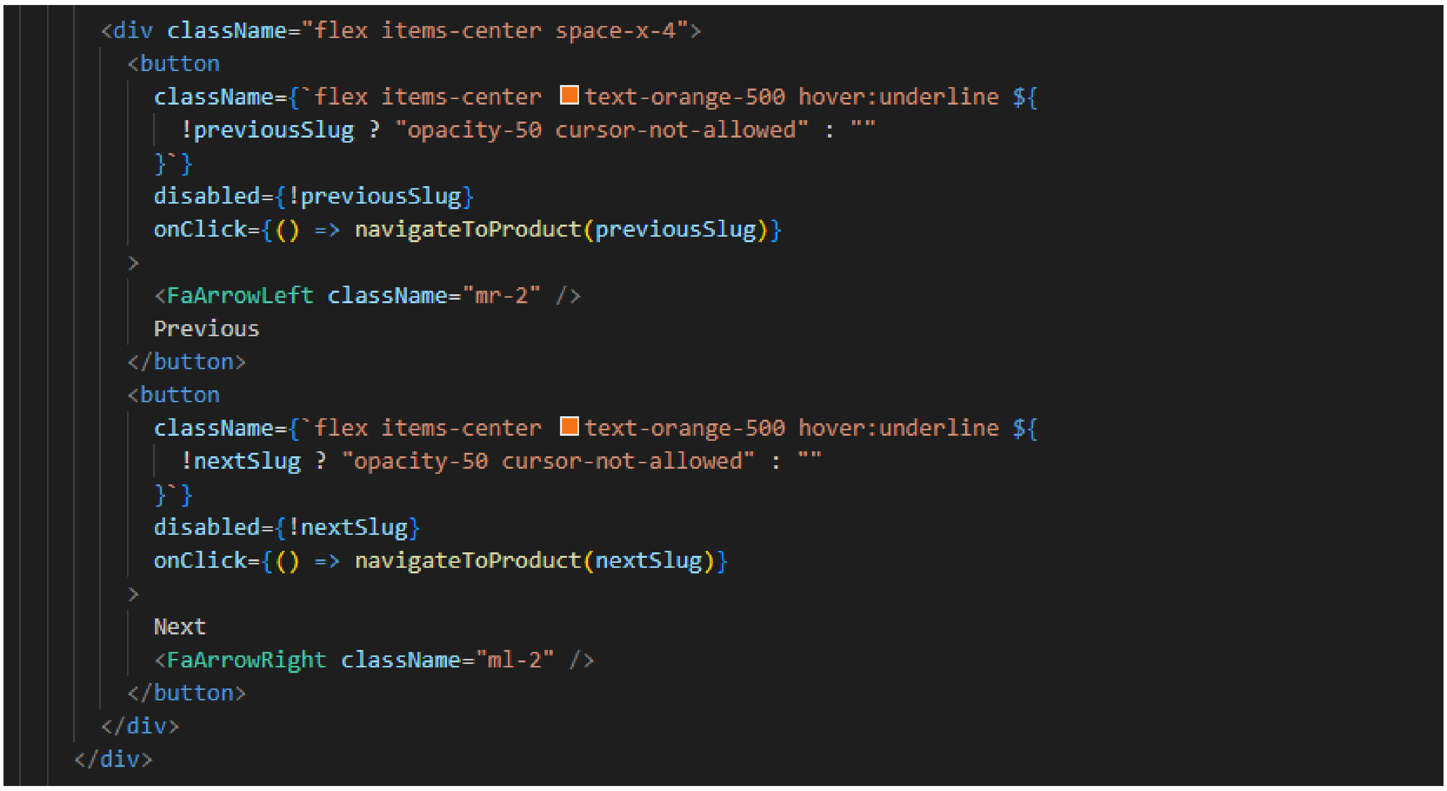
```
const query = `*[_type == "food"]{
  _id,
  name,
  slug,
  price,
  originalPrice,
  "image": image.asset->url,
  description,
  tags,
  available,
  category
}`;
```

- **Mapped Sanity responses to TypeScript interfaces**

- Fallback handling for missing image assets

Dynamic Routing Strategy

- Previous/Next product navigation using slug order

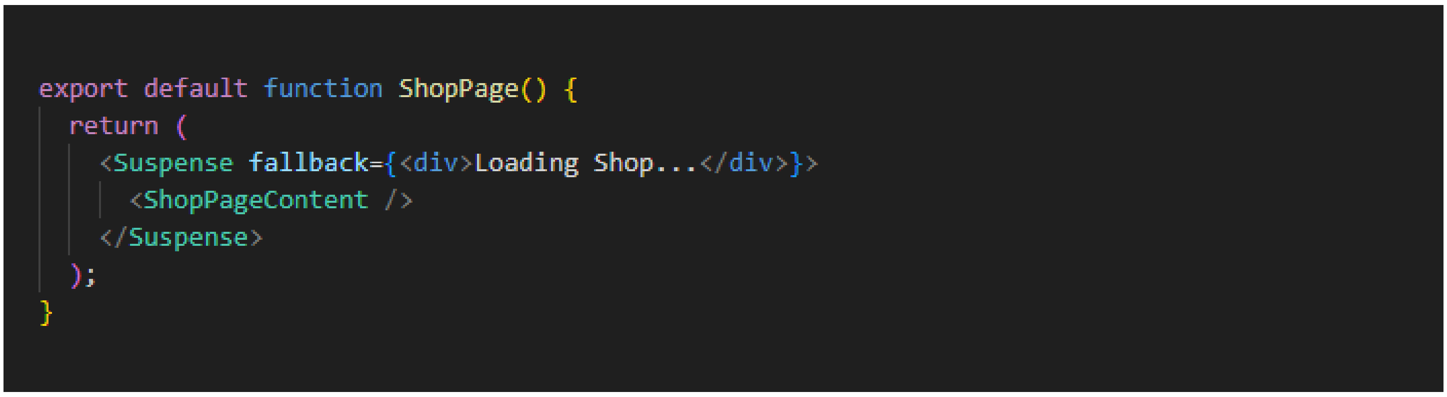


```
    <div className="flex items-center space-x-4">
      <button
        className={`flex items-center text-orange-500 hover:underline ${
          !previousSlug ? "opacity-50 cursor-not-allowed" : ""
        }`}
        disabled={!previousSlug}
        onClick={() => navigateToProduct(previousSlug)}
      >
        <FaArrowLeft className="mr-2" />
        Previous
      </button>
      <button
        className={`flex items-center text-orange-500 hover:underline ${
          !nextSlug ? "opacity-50 cursor-not-allowed" : ""
        }`}
        disabled={!nextSlug}
        onClick={() => navigateToProduct(nextSlug)}
      >
        Next
        <FaArrowRight className="ml-2" />
      </button>
    </div>
  </div>
```

- getStaticProps for product detail pages
- Generated static paths from Sanity data

Performance Optimization

- Suspense boundaries for shop page loading

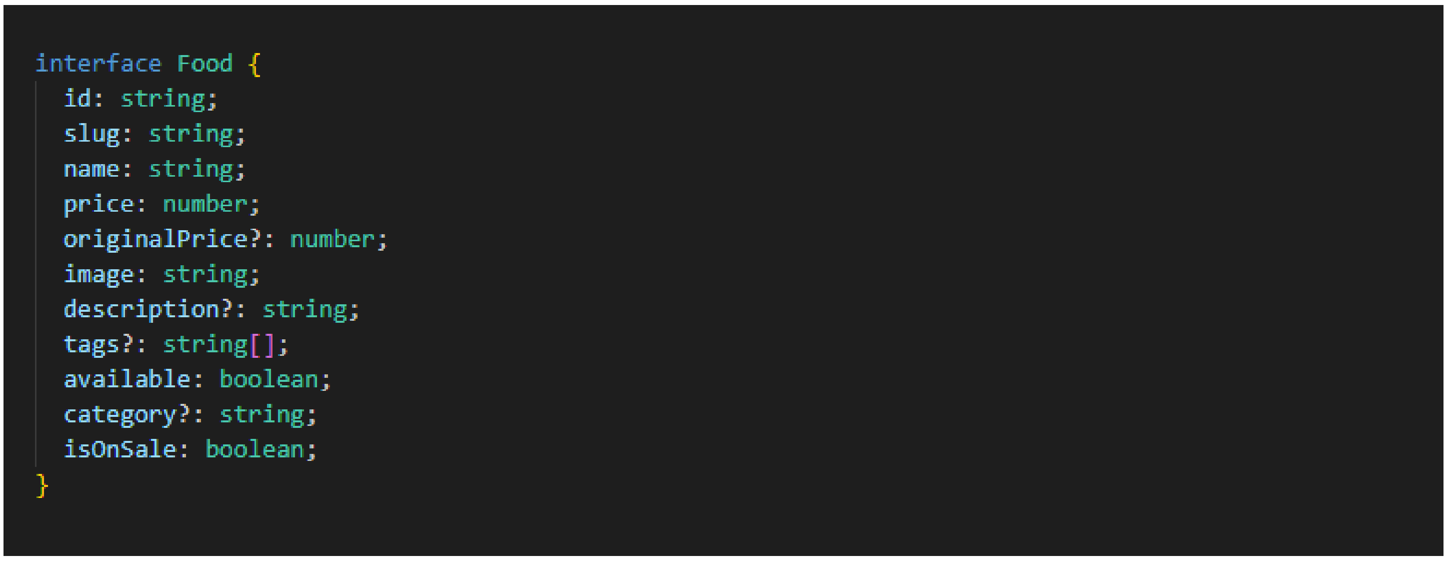


```
export default function ShopPage() {
  return (
    <Suspense fallback={<div>Loading Shop...</div>}>
      <ShopPageContent />
    </Suspense>
  );
}
```

- Image lazy loading with Next.js <Image>
- Client-side caching of CMS data

4.2 Best Practices

Type Safety



```
interface Food {
  id: string;
  slug: string;
  name: string;
  price: number;
  originalPrice?: number;
  image: string;
  description?: string;
  tags?: string[];
  available: boolean;
  category?: string;
  isOnSale: boolean;
}
```

- **Strict TypeScript interfaces for all components**
- **Sanity response validation**

Conclusion:

The Foodtuck Marketplace frontend delivers a dynamic, scalable eCommerce platform using **Next.js**, **Sanity CMS**, and **TypeScript**. Key features include real-time product listing, dynamic filtering, cart/wishlist management, and a seamless checkout flow. Modular components and reusable UI elements ensure scalability, while best practices like **type safety**, **error handling**, and **performance optimization** enhance reliability.

Challenges like handling large datasets and preventing duplicate entries were addressed through **pagination**, **state management**, and **Framer Motion** for smooth animations. The platform is optimized for performance with lazy loading, debounced search, and client-side caching.

This project sets a strong foundation for future enhancements, such as AI recommendations and advanced analytics, positioning Foodtuck as a leading online food delivery solution.