React E-Commerce App Documentation

Description

This React-based e-commerce application is designed to showcase products, allow users to view product details, add items to the cart, and proceed to checkout. The app follows modern React practices including Context API, Redux Toolkit, lazy loading, and Suspense-based routing for performance optimization.

Project Structure

1. Common Data Folder

Purpose: Holds static product data in JSON format.

Usage: Product information is fetched from this folder and used across the app.

2. Context API (Product Context)

Location: context/ProductContext.tsx

Components:

ProductsProvider → Wraps the application and provides product data globally.

useProducts → A custom hook to consume the product context easily in any component.

Purpose: Makes product data available throughout the app without prop drilling.

3. Redux (Cart Management)

Folder: redux/

Files

:cartSlice.ts

Stores cart items.

Handles addToCart action:

```
addToCart:(state,action:PayloadAction<CartItem>)=>{
  const existing=state.items.find(item=>item.id===action.payload.id);
  if(existing){
    existing.quantity+=action.payload.quantity;
  } else {
    state.items.push(action.payload);
  }
}
```

Ensures cart updates on repeated product additions.

store.ts

Defines the Redux store blueprint.

Integrates the cart slice.

4. Pages (Lazy Loaded)

All pages are lazy loaded using React.Suspense for better performance and faster initial load times.

Home.tsx → Displays best sellers and category-based shopping.

Products.tsx → Lists all available products.

ProductDetails.tsx → Shows product-specific details and descriptions.

Checkout.tsx → Displays cart items retrieved from Redux store.

Components

Card.tsx

Displays a product card with image, title, and price.

Accepts props:

- id → product ID (used for navigation to /products/:id).
- image → product image URL.
- title → product title/name.
- price → product price.
- actionButton (optional) → custom button (e.g., Add to Cart).
- Uses Link from react-router-dom to navigate to product details page.
- Lazy loads images for better performance.
- Styled with Tailwind CSS (shadow, hover:shadow-lg, transition).
- Fully reusable across different pages (Products, Home, Recommendations).

FeatureCard

A wrapper component built on top of the Card component.

- Accepts a single prop:
- product → an object containing { id, name, price, image }.
- Extracts product details and passes them to the Card component.
- Provides a cleaner interface for rendering product cards without manually passing each prop.
- Useful for pages where products are mapped from an array (e.g., feature listings, home page).

Example usage:

<FeatureCard product={{ id: 1, name: "Laptop", price: 1200, image: "/images/laptop.jpg" }} />

Footer

a simple point-based documentation for your Footer component:

A stateless functional component that renders the site footer.

Uses Tailwind CSS for layout and styling.

Layout:

- Left: copyright text → © {current year} ShopEase. All rights reserved.
- Right: navigation links → About | Contact | Privacy.
- Responsive design:
- Stacked vertically (flex-col) on small screens.
- Horizontal layout (flex-row) on medium+ screens.
- Links have a hover effect (hover:text-blue-600).
- Fully reusable across all pages.

Example usage:

<Footer />

Navbar

Purpose: Top navigation bar with logo, links, cart, and responsive mobile menu.

Features:

- Logo: Links to the homepage (/).
- Desktop Links (md+):
- Home → /
- Products → /products
- Cart Icon (Checkout link):
- Uses Redux state (cart.items) to show number of items.
- Badge appears when cart has items.

Mobile Menu:

- Hamburger menu (Menu icon).
- Toggles open/close with state menuOpen.
- Displays links stacked vertically.
- Closes automatically when a link is clicked.

Responsive:

- Desktop → horizontal links.
- Mobile → hamburger + collapsible menu.

Styling:

- Tailwind CSS classes for layout and responsiveness.
- Sticky top bar → always visible (sticky top-0 z-50).
- Shadow for separation from content.

Example usage:

<Navbar />

ProductCard

Purpose:

A specialized wrapper around the Card component to display a product with an Add to Cart button.

Props:

- product: Product → Product object containing id, name, price, image.
- onAddToCart: () => void → Callback function triggered when "Add to Cart" button is clicked.

Features:

- Reuses the generic Card component.
- Provides a predefined action button → "Add to Cart".
- Uses React.memo to avoid unnecessary re-renders (only re-renders when product or onAddToCart changes).
- Button styling → bg-gray-500 text-white px-3 py-1 rounded.

Usage Example:

<ProductCard

product={{ id: 1, name: "Headphones", price: 99.99, image: "/images/headphones.jpg" }}

onAddToCart={() => console.log("Added to cart!")}

/>

ScrollCarousel Component

Purpose:

Creates a horizontally scrollable carousel for child elements (e.g., product cards).

Props:

children: React.ReactNode[] → Array of React nodes to display inside the carousel.

Features:

- Uses useRef to reference the scrollable container.
- Scroll buttons:
- Left (‹) and Right (›) buttons scroll 4 cards at a time smoothly.
- Responsive card width:

- w-full on mobile, sm:w-1/2, md:w-1/3, lg:w-1/4.
- Hides native scrollbar (hide-scrollbar).
- Smooth scrolling behavior (scroll-smooth).
- Fully reusable for any type of content, not just product cards.

Usage Example:

```
<ScrollCarousel>
  {products.map(product => (
    <ProductCard key={product.id} product={product} onAddToCart={() => addToCart(product)} />
  ))}
</ScrollCarousel>
```

ScrollCarousel Component

Purpose:

Creates a horizontally scrollable carousel for child elements (e.g., product cards).

Props:

children: React.ReactNode[] → Array of React nodes to display inside the carousel.

Features:Uses useRef to reference the scrollable container.

Scroll buttons:

- Left (‹) and Right (›) buttons scroll 4 cards at a time smoothly.
- Responsive card width:
- w-full on mobile, sm:w-1/2, md:w-1/3, lg:w-1/4.
- Hides native scrollbar (hide-scrollbar).
- Smooth scrolling behavior (scroll-smooth).
- Fully reusable for any type of content, not just product cards.

Usage Example:

```
<ScrollCarousel>
  {products.map(product => (
    <ProductCard key={product.id} product={product} onAddToCart={() => addToCart(product)} />
  ))}
</ScrollCarousel>
```

ShopNavBar Component

Purpose:

A horizontal navigation bar for product categories inside the shop pages.

Features:

- Sticky navigation (sticky top-20 z-40) below the main Navbar.
- Background color: light gray (bg-[#e8e8e8]) with shadow.
- Horizontal scrollable layout for categories (overflow-x-auto, scrollbar-hide).
- Categories include:
- Shop All → /products
- Computers → /products/category/laptops
- Audio → /products/category/headphones
- Watches → /products/category/watches
- Home Cinema → /products/category/television
- Camera → /products/category/camera
- Category links have hover effect (hover:text-blue-600).
- Responsive and works on mobile devices with horizontal scroll.

Usage Example:

<ShopNavBar />

ProductsByCategory

Purpose:

Extracts all products marked as best sellers from a grouped category object.

Input:

productsByCategory: { [category: string]: Product[] }

An object where keys are category names and values are arrays of Product objects.

Output:

Product[] → Array of all products with bestSeller: true.

Features / Logic:

- Iterates through all categories.
- Filters products with bestSeller property set to true.
- Combines all best-selling products into a single array.
- Usage Example:
- import { findBestsellers } from "../utils/findBestsellers";
- const bestSellers = findBestsellers(productsByCategory);
- console.log(bestSellers);
- ImagesByCategory

Purpose:

Dynamically imports all product images from the assets folder and groups them by category.

Logic / Features:

Uses Vite's import.meta.glob to import all images (png, jpg, jpeg, svg, avif) under ../assets/products/**.

Organizes images into a nested object:

```
{
  categoryName: {
    fileName: imageURL
  }
}
```

Example:

```
{
  mobiles: {
    "iphone.jpg": "/assets/products/mobiles/iphone.jpg",
    "samsung.jpg": "/assets/products/mobiles/samsung.jpg"
  }
}
```

Returns a promise (async) with the structured object.

Output:

Promise<{ [category: string]: { [fileName: string]: string } }>

Usage Example:

```
import { importAllCategoryImages } from "../utils/importAllCategoryImages";

const imagesByCategory = await importAllCategoryImages();

console.log(imagesByCategory.mobiles["iphone.jpg"]);
```

ProductsWithImagesByCategory

Purpose:

Combines product data (productsByCategory) with actual image URLs imported dynamically, producing a fully usable dataset for the app.

Data Sources:

productsByCategory → Raw product data without image paths.

importAllCategoryImages() → Dynamically imported image URLs grouped by category.

Logic / Features:

Loads all category images asynchronously.

Iterates over each category in productsByCategory.

Maps each product to include the corresponding image URL from categoryImages.

Produces a new object with the structure:

```
{
  categoryName: Product[] // each product now includes image field
}
```

Output:

productsWithImagesByCategory: { [category: string]: Product[] } → Fully populated product objects with images.

Usage Example:

```
import { productsWithImagesByCategory } from "../context/productsWithImagesByCategory";

const mobiles = productsWithImagesByCategory["Mobiles"];

console.log(mobiles[0].image);
```