E-commerce Web Application with Next.js and Sanity CMS

This project is a modern e-commerce web application built with Next.js and Sanity CMS, offering a seamless shopping experience with dynamic content management.

Project Description

This e-commerce application is designed to provide users with a responsive and intuitive shopping platform. It leverages Next.js for server-side rendering and Sanity CMS for flexible content management, allowing for easy updates to product information, categories, and other dynamic content.

Key features include:

- Responsive design for optimal viewing on various devices
- Dynamic product listings with filtering and search capabilities
- User-friendly shopping cart and checkout process
- Integration with Sanity CMS for content management
- Server-side rendering for improved performance and SEO

The application is built with a focus on performance, scalability, and user experience, making it suitable for small to medium-sized e-commerce businesses looking for a modern web presence.

Filter Functionality

The application implements a robust filtering system to help users find products that match their preferences. The `FilterSidebar` component (`src/components/FilterSidebar.tsx`) provides the following filtering options:

1. Categories: Users can select one or more product categories to narrow down their search.
2. Price Range: A slider allows users to set a minimum and maximum price range for products.
3. Sizes: Users can filter products based on available sizes.

The filtering functionality dynamically updates the product list as users adjust their preferences, providing a seamless and responsive shopping experience.

State Management

This application uses Zustand for efficient and scalable state management. Two main stores are implemented:

1. Cart Store (`src/store/useCartStore.ts`): Manages the shopping cart state, including:
    - Adding products to the cart
    - Removing products from the cart
    - Updating product quantities
    - Calculating total quantity and price
2. Customer Data Store (`src/store/useCustomerData.ts`): Manages customer information, including:
    - Setting customer data (ID, phone, name)
    - Clearing customer data

Zustand provides a simple and performant solution for managing global state across the application, ensuring a smooth user experience and efficient data flow.

Repository Structure

```
.
├── src/
│   ├── app/
│   │   ├── api/
│   │   ├── cart/
│   │   ├── category/
│   │   ├── check-out/
│   │   ├── order/
│   │   ├── product-details/
│   │   └── studio/
│   ├── components/
│   │   ├── ui/
│   │   └── FilterSidebar.tsx
│   ├── lib/
│   ├── sanity/
│   │   ├── lib/
│   │   └── schemaTypes/
│   ├── store/
│   │   ├── useCartStore.ts
│   │   └── useCustomerData.ts
│   └── styles/
├── public/
├── components.json
├── next.config.js
├── next.config.mjs
├── package.json
├── postcss.config.mjs
├── README.md
├── sanity.cli.ts
├── sanity.config.ts
├── tailwind.config.js
├── tailwind.config.ts
└── tsconfig.json
```

Key Files:

- `src/app/layout.tsx` : Main layout component for the application
- `src/app/page.tsx` : Home page component
- `src/components/Hero.tsx` : Hero section component for the home page
- `src/components/FilterSidebar.tsx` : Component for product filtering
- `src/sanity/schemaTypes/index.ts` : Sanity CMS schema definitions
- `src/store/useCartStore.ts` : Zustand store for cart management
- `src/store/useCustomerData.ts` : Zustand store for customer data
- `next.config.js` : Next.js configuration file
- `sanity.config.ts` : Sanity Studio configuration
- `tailwind.config.js` : Tailwind CSS configuration

Data Flow

The application follows a typical client-server architecture with Next.js handling server-side rendering and API routes, while Sanity CMS manages the content.

1. User requests a page (e.g., product listing)
2. Next.js server receives the request
3. Server fetches data from Sanity CMS using the Sanity client
4. Server renders the page with the fetched data
5. Rendered page is sent to the user's browser
6. Client-side JavaScript enhances interactivity (e.g., add to cart functionality)

```
[User] <-> [Next.js Server] <-> [Sanity CMS]
   ^               ^
   |               |
   v               v
[Browser]    [API Routes]
```

Customer and Order Data:

- Customer data is stored in Sanity CMS using the `customer` schema
- Order data is stored in Sanity CMS using the `order` schema
- When a user places an order, the order information is sent to Sanity CMS

- The `OrderPage` component (`src/app/order/page.tsx`) fetches and displays the user's orders from Sanity CMS

Creating Customer and Order Data:

- When a user completes the checkout process, the `CheckOutPage` component (`src/components/CheckOutPage.tsx`) creates a new customer and order in Sanity CMS.
- The component first creates a new customer entry in Sanity CMS using the `createCustomer` function.
- After creating the customer, it creates a new order entry using the `createOrder` function, which includes the customer information and the items in the cart.

Fetching and Displaying Order Data:

- The `OrderPage` component (`src/app/order/page.tsx`) is responsible for fetching and displaying the user's orders.
- It uses the `useCustomerData` hook to retrieve the customer's information.
- The component then fetches the customer's orders from Sanity CMS using a GraphQL query.
- The fetched order data is displayed in a table format, showing details such as order date, status, items ordered, and total price.

Note: The application uses server-side rendering for initial page loads and client-side fetching for dynamic updates to improve performance and user experience.

Infrastructure

The project uses Sanity CMS as its content management system. The main configuration for Sanity is defined in `sanity.config.ts` :

- Project ID: Defined by `NEXT_PUBLIC_SANITY_PROJECT_ID` environment variable
- Dataset: Defined by `NEXT_PUBLIC_SANITY_DATASET` environment variable
- Schema: Imported from `./src/sanity/schemaTypes`
- Plugins:
    - structureTool: Configures the Sanity Studio structure
    - visionTool: Enables GROQ querying within the Studio

The Sanity Studio is mounted on the `/studio` route, allowing content editors to manage the e-commerce content directly within the application.