✫Website Functionality Documentation ✫

Search Functionality Implementation

I have implemented a search functionality on my website where users can search for a product, and the matching products will be displayed. This functionality works by taking user input, comparing it with stored product data, and showing relevant results. If the dataset is small, client-side filtering is used, and for larger datasets, backend queries handle the search. This helps users quickly find their desired products.

Product Detail Page Functionality

The product detail page dynamically displays product information passed through URL parameters. Here's how it works:

1. **Data Passing & Retrieval:**
   - The `Card` component passes product data (image, name, ranking, and price) through URL parameters using Next.js `Link`.
   - The `ProductDetail` page extracts these parameters using `searchParams` and stores them in state (`useState`).
2. **Data Handling with `useEffect`:**
   - `useEffect` fetches the data from `searchParams` and updates the state (`setData`).
   - This ensures that product details are available when the page loads.
3. **Dynamic Rendering:**
   - The product image, name, rating, and price are displayed dynamically.
   - Multiple thumbnail images are shown for preview.
4. **Interactive Features:**
   - Users can select product sizes and colors.
   - The `Add to Cart` button saves the selected product to `localStorage`. If the product already exists, its quantity increases instead of adding a duplicate.
5. **Review System:**
   - The `ReviewCard` component allows users to view and interact with product reviews.

Product Filter Functionality

This document explains the filter functionality implemented in the product display page.

Components Overview

1. **ProductFilterColor**: Fetches products from Sanity CMS and displays them.

2. **FiltersSidebar**: Provides various filter options for the products.
3. **ProductPage**: Combines the filter sidebar with the product display.

Filter Options

The following filter options are available:

1. **Categories**: Users can select from predefined categories like T-shirts, Shorts, Shirts, Hoodie, and Jeans.
2. **Price Range**: A slider allows users to set a minimum and maximum price.
3. **Colors**: Users can select multiple colors from a color palette.
4. **Sizes**: Various size options from XX-Small to 4X-Large are available for selection.
5. **Dress Style**: Options include Casual, Formal, Party, and Gym.

Implementation Details

State Management

- React's `useState` hook is used to manage the state of selected filters and open/closed filter sections.
- The `useEffect` hook is used to fetch initial product data and set up the filtered products state.

Filter Logic :The `applyFilters` function in the `FiltersSidebar` component handles the filtering logic:

1. It filters the products based on the selected criteria:
   - Price range
   - Category
   - Size
   - Dress style
2. The filtered products are then passed back to the parent component (`ProductPage`) via the `onFilterChange` callback.

User Interface

- The sidebar uses the `Collapsible` component to create expandable/collapsible sections for each filter type.
- Checkboxes, buttons, and sliders are used for various filter options, providing an intuitive user interface.

Product Display

- Filtered products are displayed in a grid layout in the `ProductPage` component.

- Each product card shows details like name, price, rating, available colors, and sizes.

Usage : Users can interact with the filter options in the sidebar. As they select or deselect options, the `applyFilters` function is called, which updates the displayed products in real-time.

Reviews Section Component Overview

The Reviews Section is a React component that handles the display and interaction with product reviews. Below is a high-level overview of its structure and functionality:

Component Structure

1. **State Management**
   - Uses React hooks (`useState`, `useEffect`) to manage component state.
   - Tracks reviews, sorting preferences, form data, and UI states.
2. **Data Fetching**
   - push review data to Sanity CMS.
   - Initial data, including user comments, names, and other relevant fields, is provided by the user and then fetched from Sanity to populate the reviews section.
3. **User Interface**
   - Displays a list of reviews using a custom `ReviewCard` component.
   - Provides sorting options via a popover menu.
   - Includes a form for adding/editing reviews.
4. **Interactivity**
   - Allows users to add, edit, and delete reviews.
   - Implements sorting functionality for reviews.
5. **Feedback & Notifications**
   - Uses toast notifications for user feedback on actions.

Key Features

- Dynamic review list with sorting capabilities.
- Add/Edit review form with validation.
- Delete review functionality.
- Responsive design with different layouts for mobile and desktop.

graph TD

A[User Interaction] -->|Add/Edit/Delete| B[Reviews Section Component]

B -->|Update State| C[Review List]

B -->|Update State| D[Review Form]

E[Sanity CMS] <-->|Fetch Data| B

F[Sort Function] -->|Filter Reviews| C

G[Toast Notifications] <-- Feedback --> B

Add to Cart Functionality Documentation

This document outlines the process of adding items to the cart on the product detail page.

Overview

The add-to-cart functionality allows users to select a product variant (size and color) and add it to their shopping cart. The cart data is stored in the browser's `localStorage` for persistence across page reloads. The items added to the cart are then displayed on the checkout page, showing all the selected details.

Process Flow

graph TD

A[User selects product size] -->B[User selects product color]

B -->C[User clicks 'Add to Cart' button]

C -->D[handleAddToCart function is called]

D -->E[Create new item object]

E -->F[Retrieve existing cart from localStorage]

F -->G{Item already in cart?}

G -->|Yes| H[Increase quantity of existing item]

G -->|No| I[Add new item to cart]

H -->J[Save updated cart to localStorage]

I -->J

J -->K[Display success message]

K -->L[Items are displayed in the checkout page with full details]

Checkout Page Functionality

Overview

In this code, the checkout page handles the display and interactivity of the shopping cart. It performs the following actions:

1. **Displays Cart Items**: Shows the list of items that the user has added to the cart, including the product image, name, size, color, and price.
2. **Manage Quantities**: Users can adjust the quantity of each item using the minus and plus buttons, with the quantity being updated in both the state and `localStorage`.
3. **Remove Items**: Users can remove an item from the cart using the trash button.
4. **Order Summary**: Displays the subtotal, discount, delivery fee, and total cost.
5. **Promo Code Input**: Allows users to add a promo code for discounts.
6. **Proceed to Checkout**: A button that allows users to move to the next page to add address details, leading to the order confirmation page.

Process Flow

```
graph TD

**A**[User adds items to cart] --> **B**[Cart items stored in localStorage]

**B** --> C[Display cart items in the shopping cart]

C --> D[User can update item quantity or remove items]

D --> E[Update cart state and localStorage after quantity changes]

D --> F[Remove item from cart and update state/localStorage]

C --> G[Display order summary including subtotal, discount, delivery fee,
total]

G --> H[User can apply a promo code for a discount]

H --> I[Display the total cost after applying promo code]

G --> J[User proceeds to checkout page]

J --> K[Redirect to address details page (Order_confirm)]
```

Features:

1. **Add to Cart**: Items are added to the cart and stored in the `localStorage`. When the page is reloaded, the cart is fetched from `localStorage` to ensure data persistence.
2. **Remove Item**: Users can remove an item from the cart. This updates both the state and the `localStorage`.
3. **Update Quantity**: Users can update the quantity of an item using the plus/minus buttons. The quantity is updated and saved to the state and `localStorage`.
4. **Order Summary**: This section displays:
   - Subtotal: The total price before any discounts.
   - Discount: A 20% discount applied to the subtotal.
   - Delivery Fee: A fixed delivery fee of $15.
   - Total: The final amount after applying the discount and adding the delivery fee.
5. **Promo Code**: Users can enter a promo code (though the functionality to apply the promo code is not implemented here).
6. **Proceed to Checkout**: The "Add Address Details" button takes the user to the next page where they can enter their address details.

Shipment Process

Overview

This document outlines the shipment process implemented in our application, which includes a shipment form, API interactions, and a response viewer. The system allows users to input shipment details, submit them to a shipping service API, and view the resulting shipment information.

Components and Their Functions

1. **Shipment Form**
2. **API Service**
3. **JSON Response Viewer**

Detailed Process

1. Shipment Form

The shipment form is the primary interface for users to input all necessary details for creating a shipment label. It's divided into several sections:

a) **Shipment Details**

- Carrier ID selection
- Service Code selection (e.g., USPS Priority Mail, UPS Ground)

b) **Ship To Information**

- Recipient's name, phone, address
- City, state, postal code, country code
- Residential/Commercial indicator

c) **Ship From Information**

- Sender's name, company, phone, address
- City, state, postal code, country code
- Residential/Commercial indicator

d) **Package Details**

- Weight (in ounces)
- Dimensions (height, width, length in inches)

The form uses various input types including text fields, number inputs, and dropdown selects to capture all required information accurately.

2. API Service

The API service handles the communication between our application and the shipping service (in this case, ShipEngine). It includes two main functions:

a) **GET Method**

- Fetches carrier information from the ShipEngine API
- Uses the SHIPENGINE_API_KEY for authentication
- Returns a list of available carriers

b) **POST Method**

- Sends the shipment details to the ShipEngine API to create a shipping label
- Constructs the request body using the data from the shipment form
- Handles the API response and returns the shipment data

3. JSON Response Viewer

After successful submission of the shipment form, the JSON Response Viewer displays the shipment details returned by the API. It includes:

- A button to view shipment details
- An alert dialog that shows:
  - Shipment ID
  - Tracking Number

    - Shipment Status
    - Shipping Cost
    - Recipient's Address
- A button to download the shipping label

User Flow

1. The user fills out the shipment form with all necessary details.
2. Upon submission, the form data is sent to the API service.
3. The API service constructs the appropriate request and sends it to the ShipEngine API.
4. The API response is received and processed.
5. The JSON Response Viewer component is updated with the new shipment data.
6. The user can then view the shipment details and download the shipping label.

Key Features

- Real-time form validation and error handling
- Integration with ShipEngine API for live shipping rates and label generation
- Ability to download shipping labels directly from the application
- Comprehensive display of shipment details including costs, tracking information, and addresses

Future Improvements

- Implement address validation to ensure accuracy of shipping addresses
- Add support for multiple packages in a single shipment
- Integrate tracking functionality to allow users to track shipments directly from the application
- Implement a shipment history feature for easy access to past shipments