

Day 4 - Dynamic Frontend Components

[Hekto: Building the Future of Furniture Shopping]

Author: Areeba Bano

Date: [20 : January : 2025]

Version: 1.0.0

1. Introduction

This documentation provides an overview of the E-Commerce Web Application, developed with Next.js, and Sanity CMS for content management. The application enables users to interact with a variety of components like product listings, shopping carts, checkout processes, and more. The primary goal of the application is to deliver a user-friendly e-commerce experience that integrates seamlessly with a headless CMS and supports dynamic routing.

Technologies Used:

- **Frontend: React, Next.js**
- **State Management: React Context API**
- **Database/Content Management: Sanity CMS**
- **Libraries/Tools: Axios, React Toast Notifications, LocalStorage**
- **Deployment: Vercel / [Add if using a specific platform]**

By Areeba Bano

2. Project Overview

The application features essential e-commerce functionalities that ensure smooth user interaction, including browsing products, adding items to the cart, managing the wishlist, completing checkout, and leaving customer feedback. It also supports dynamic routing, which allows users to visit specific product pages.

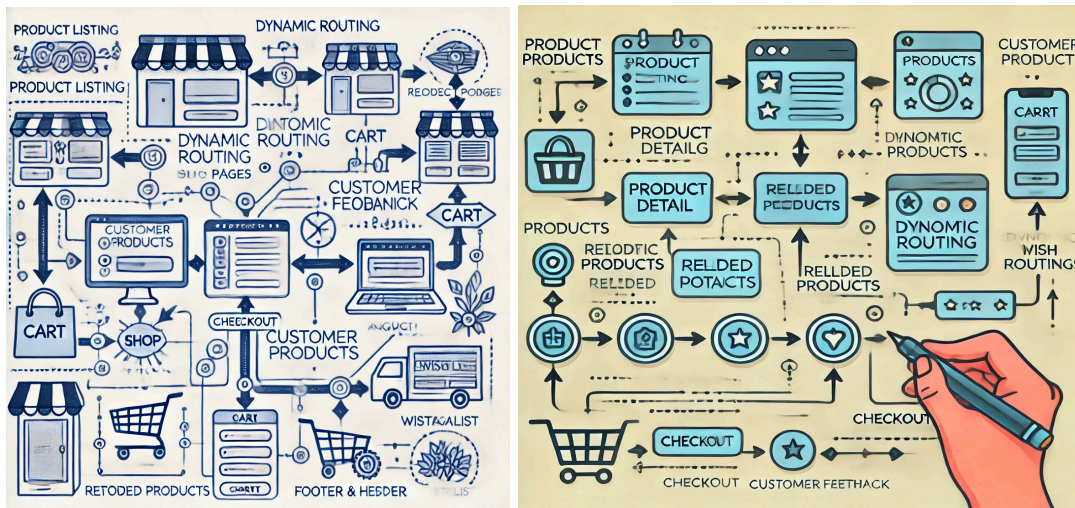
Core Features:

- **Product Listing:** Display products fetched from the CMS.
- **Product Detail Page:** Dynamic display of product information.
- **Search Bar:** Search functionality to find products.
- **Shopping Cart:** View and manage cart items.
- **Wishlist:** Save favorite products for later.
- **Checkout:** Form for order processing.
- **Pagination:** Manage large product datasets efficiently.
- **Filter and Sorting:** Sort products by various criteria.
- **Related Products:** Display related items.
- **Footer and Header:** Common UI components across pages.
- **Notifications:** Real-time user notifications for actions.
- **FAQ and Help Center:** Provide user support and commonly asked questions.

- **Customer Feedback:** Collect product feedback from users.
- **Loading Component:** A reusable component with a spinner animation and loading message, styled using Tailwind CSS for centered display and dynamic visuals.
- **Dynamic Routing:** Shop and product routes for navigation.

3. Architecture Overview

The architecture follows a clean, modular structure with a clear separation of concerns. Below is the diagram representing the overall application architecture:



Note: This is a simplified architecture overview. Each component interacts within this flow to build a cohesive user experience.

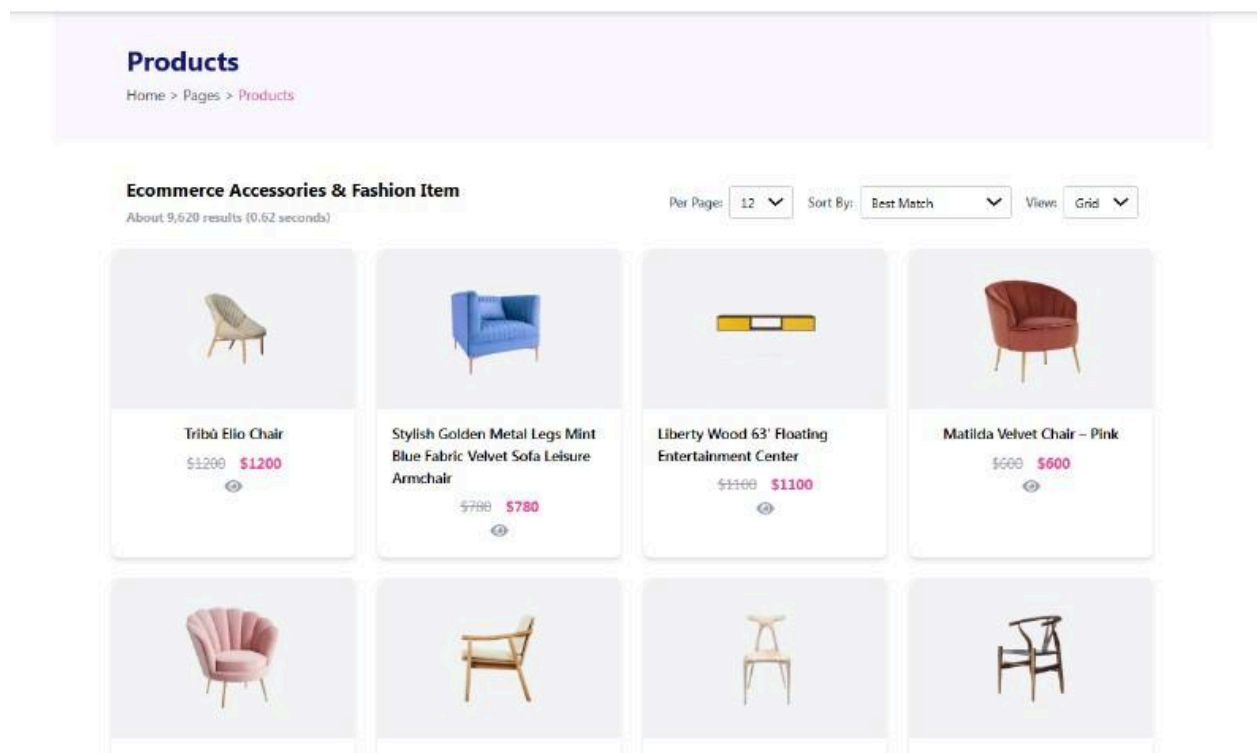
4. Component Breakdown

Below, we will break down each of the major components within the application, including their purpose, implementation, challenges, and solutions.

- **Product Listing Page**

Description:

This page displays products fetched from Sanity CMS in a responsive grid layout. It includes features like product images, names, prices, and options to add products to the cart, wishlist, or view details. The page supports dynamic loading with a "Load More" button to show more products.



Implementation Challenges & Solutions:

By Areeba Bano

1. Data Fetching:

- Challenge: Fetching product data from Sanity.
- Solution: Used **useEffect** to fetch data on component mount and **useState** to store and display it.

2. Pagination:

- Challenge: Dynamically loading products.
- Solution: Implemented a "Load More" button that increases the visible product count by 8 with each click.

3. Responsive Layout:

- Challenge: Ensuring the grid adjusts on different screen sizes.
- Solution: Used Tailwind CSS grid classes for a responsive layout.

4. Hover Effects:

- Challenge: Showing action icons only on hover.
- Solution: Utilized **group-hover** and transition classes in Tailwind CSS for smooth interactivity.

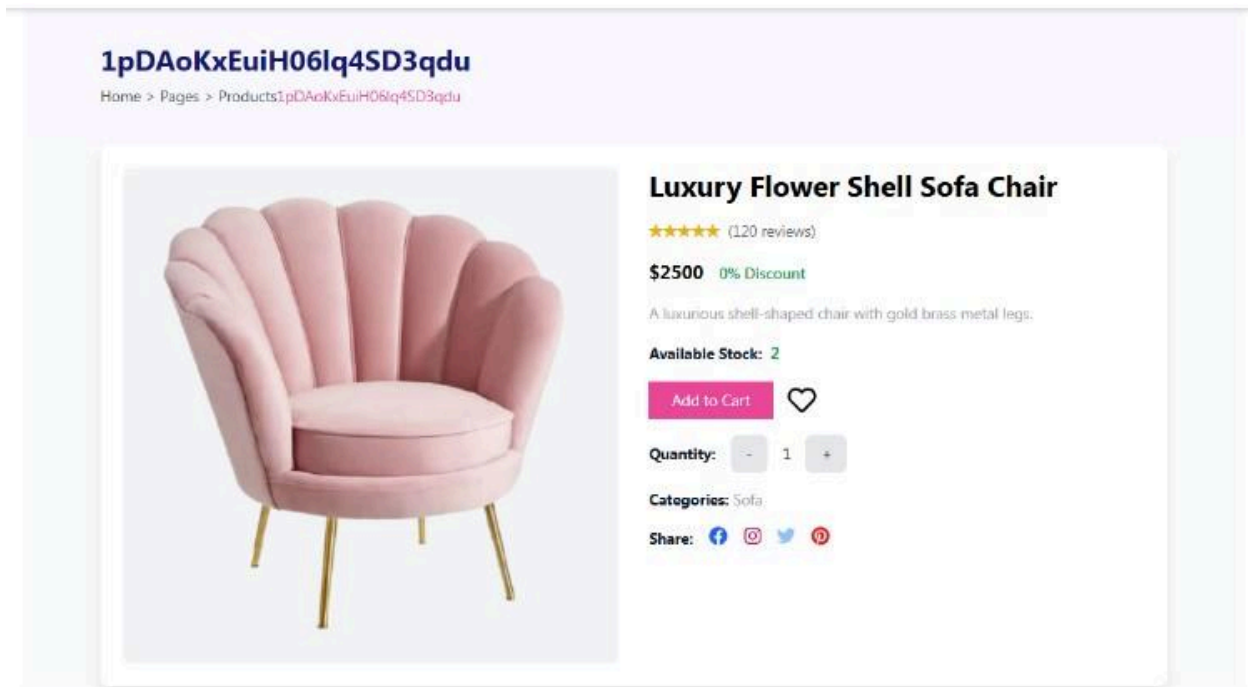
This implementation ensures a dynamic, responsive product listing with interactive actions and smooth user experience.

● Product Detail Page

Description

The product detail page displays comprehensive information about a product, including its image, name, price, discount percentage, description, stock availability, and category. Users can adjust quantity, add

items to their cart, mark them as favorites, and share product details on social media.



Implementation Challenges & Solutions

1. Data Fetching

- **Challenge:** Ensuring data loads correctly for specific product IDs.
- **Solution:** Used a query with **Sanity** to fetch product data and implemented **try-catch** for error handling.

2. Stock Management

- **Challenge:** Preventing users from adding unavailable stock to the cart.
- **Solution:** Disabled the "Add to Cart" button when stock is zero and showed stock-level alerts.

3. User Interaction

- **Challenge:** Managing cart updates and favorite toggles.
- **Solution:** Used state management to handle real-time cart and favorite actions efficiently.

4. Dynamic Description

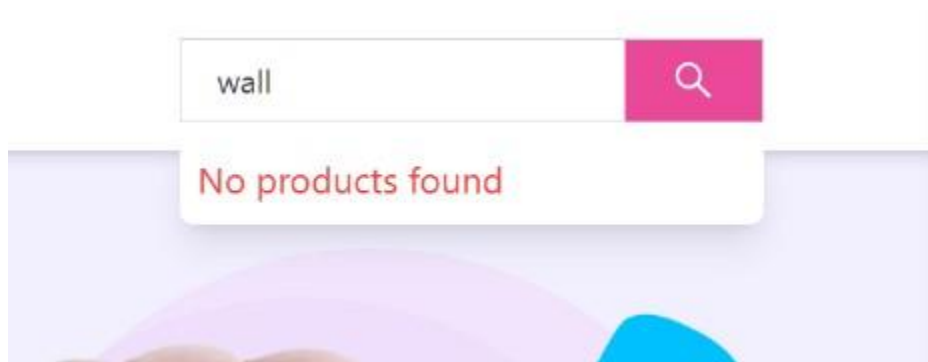
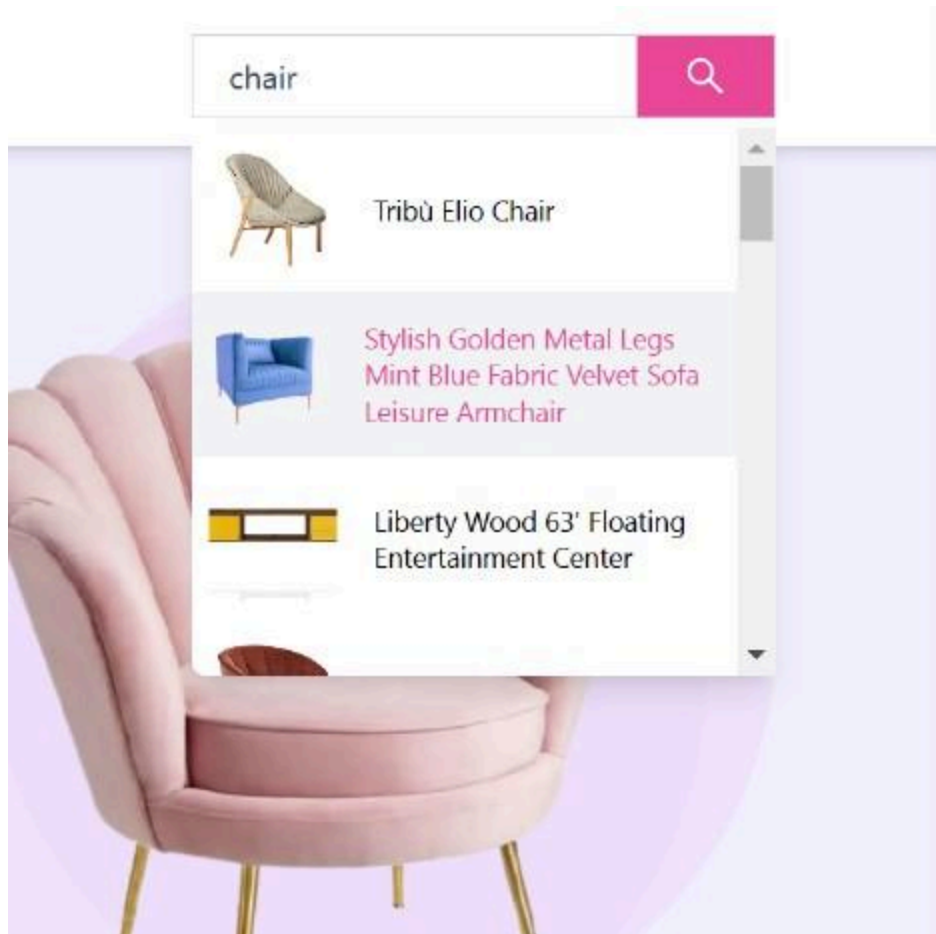
- **Challenge:** Handling missing or lengthy descriptions.
- **Solution:** Added conditional rendering for missing descriptions and a truncation mechanism for lengthy ones.

This implementation ensures a smooth, user-friendly shopping experience.

● Search Bar Component

Description

The search bar in an e-commerce header helps users quickly find products by typing keywords or product names. It improves navigation and user experience.



Implementation

- Prominent placement in the header.
- Real-time query suggestions and results.
- Autocomplete and category filters for usability.

By Areeba Bano

Challenges & Solutions

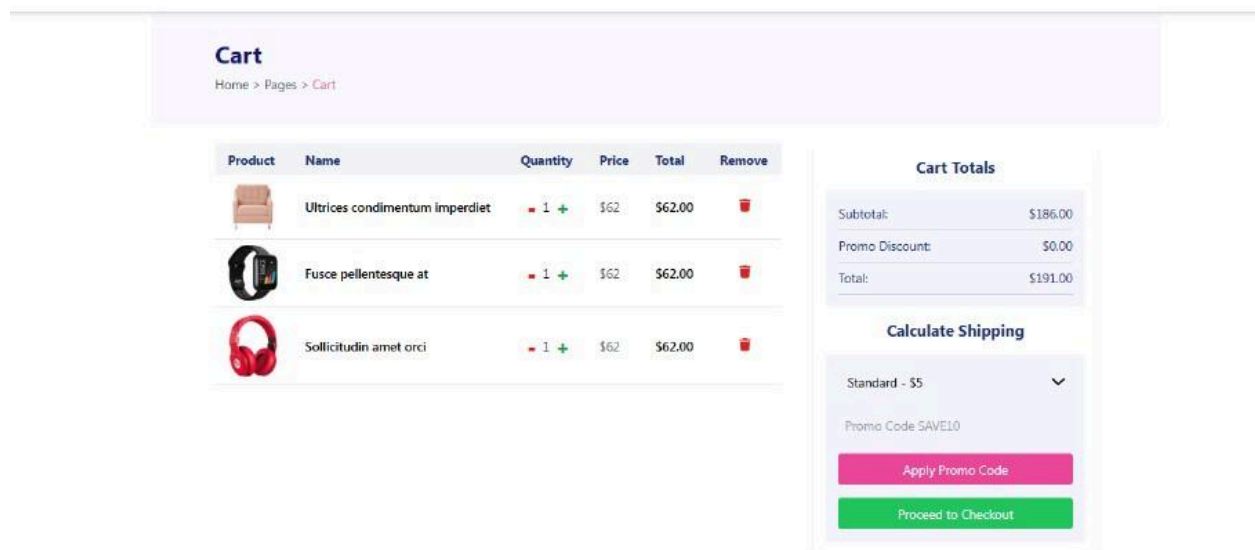
1. **Performance:** Optimize search algorithms with indexing and caching.
2. **Autocomplete:** Incorporate fuzzy matching and typo handling.
3. **Design:** Ensure a responsive, intuitive interface.
4. **Compatibility:** Guarantee smooth functionality across devices.

By tackling these challenges, the e-commerce platform delivers an efficient and user-friendly search experience.

● Cart Page

Description:

The cart system allows users to manage items in their cart, including adding, removing, and updating quantities. It also supports promo code application and shipping method selection.



Implementation:

By Areeba Bano

React Context API is used to manage the cart state globally. The **CartProvider** holds cart data and provides functions like **addToCart**, **removeFromCart**, and **updateQuantity**. **CartPage** displays items and allows interactions.

Challenges:

1. Managing cart state across multiple components.
2. Correctly updating item quantities.
3. Handling promo codes and shipping dynamically.

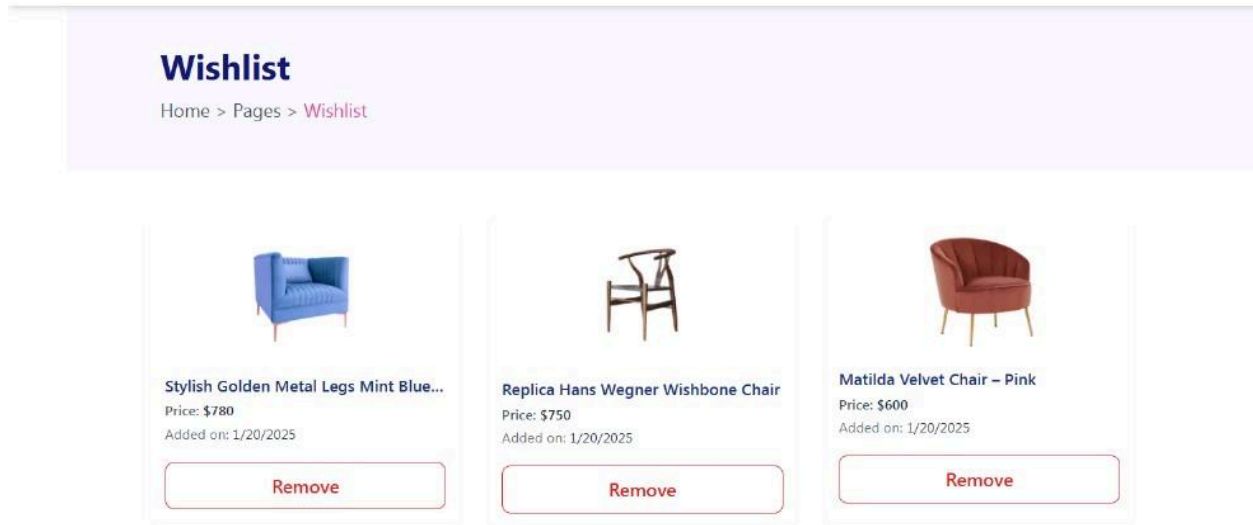
Solution:

1. The Context API provides a centralized state.
2. Local state (**editQuantities**) is used to track individual item quantities.
3. Promo codes and shipping costs are handled with conditional logic and state updates.
- 4.

● Wishlist Page

Description:

The wishlist feature allows users to save their favorite products for easy access. Users can add items to their favorites, view them in a list, and remove them when needed. This is implemented using React Context API to manage the global state of the wishlist.



Implementation:

The **FavouriteContext** provides global state management for the favorites list. It contains methods to add and remove products from the wishlist. The **FavouriteProvider** component wraps the application and makes the context accessible. In the **FavouritePage**, the list of favorite products is displayed, and users can remove items by clicking a button. The state updates in real-time, ensuring changes reflect immediately in the UI.

Challenges:

1. **State management:** Keeping the wishlist state synchronized across components.
2. **UI updates:** Ensuring the UI reflects the changes without page reloads.
3. **Optimized performance:** Handling large datasets of favorite products efficiently.

Solution:

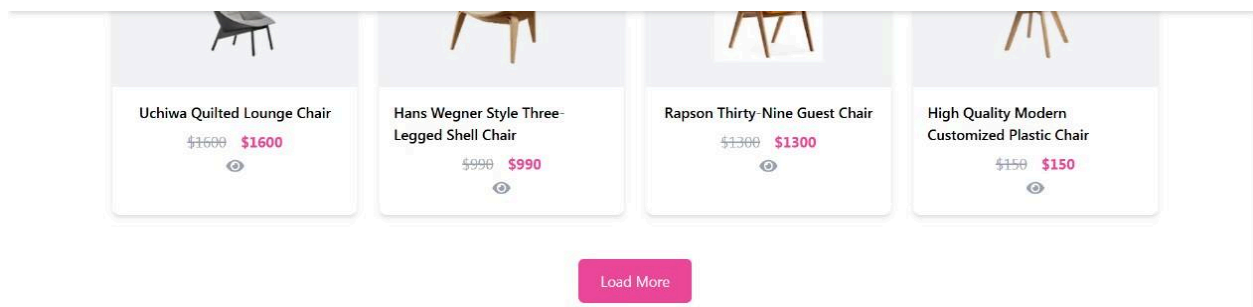
1. The Context API ensures centralized state management, making it easy to access and modify the wishlist globally.
2. Use of React state (**useState**) ensures the UI updates dynamically whenever changes are made to the wishlist.
3. By filtering out items using their unique **id**, the **removeFromFavourites** function efficiently updates the state without unnecessary re-renders.

4.

● Pagination Component

Description:

The **Pagination** component allows navigation between pages, displaying previous, next, and specific page numbers. It adjusts the page numbers based on the current page and total pages, using ellipses for skipped pages.



Implementation:

- Displays page numbers dynamically based on the current page and total pages.

- Shows ellipses when pages are skipped.
- Provides "Previous" and "Next" buttons for navigation.

Challenges:

1. Handling edge cases with few pages or when near the beginning or end.
2. Properly rendering ellipses when necessary.

Solution:

1. Adjusted logic to show the correct pages and handle edge cases.
2. Conditionally rendered ellipses and optimized the page rendering for smooth performance.
- 3.

● Filter and Sorting Component

Description:

This component provides options for filtering and sorting products in an e-commerce platform. It includes options to select how many products to display per page, how to sort the products, and how to view them (grid or list view).



Implementation:

- **Per Page:** A dropdown allows the user to select the number of products shown per page.
- **Sort By:** A dropdown for sorting products by "Best Match", "Price: Low to High", or "Price: High to Low".
- **View:** A dropdown to toggle between grid and list views.

Challenges:

1. **Handling Dynamic Results:** Managing different product counts per page and dynamically updating the view can become complex.
2. **State Management:** Keeping track of selected values across multiple dropdowns and updating the UI accordingly.

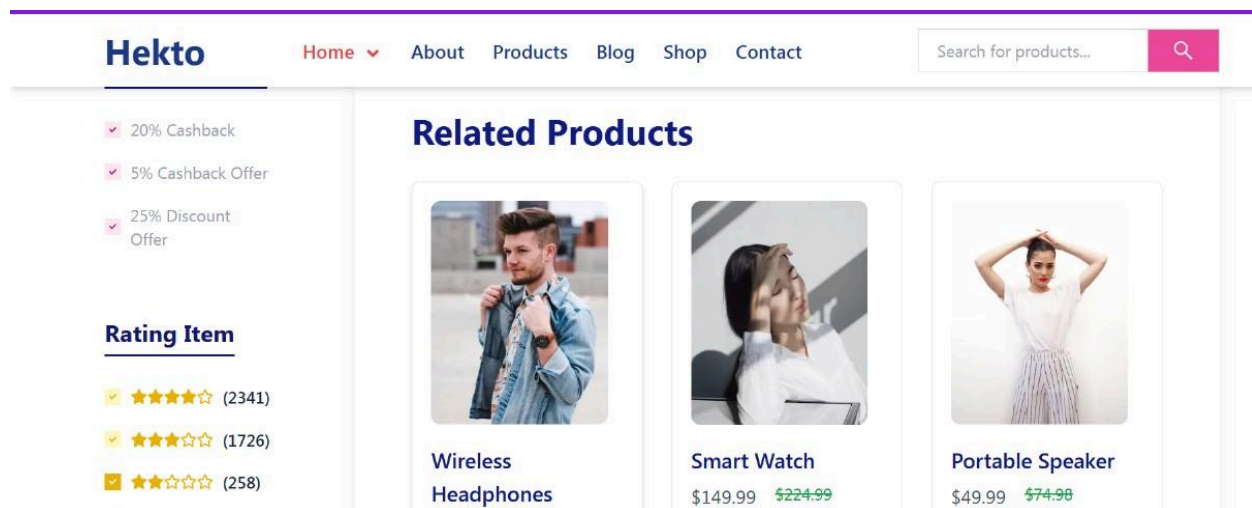
Solution:

1. The component layout adjusts based on the screen size using Flexbox for responsive design.
2. Each dropdown has options, and the component is structured to allow for easy state management and future enhancement.
- 3.

● **Related Products Component**

Description:

This component displays a list of related products. It shows product details like the name, price, rating, and image. The layout is responsive and adjusts to different screen sizes using CSS grid, ensuring an optimal display of products.



Implementation:

- **Grid Layout:** A responsive grid layout (**grid-cols-1**, **sm:grid-cols-2**, **md:grid-cols-4**) is used to display the products in various columns based on the screen size.
- **Product Data:** The component maps through an imported array (**relatedProducts**) to dynamically render the product information, including an image, name, price, and rating.
- **Styling:** The component is styled with Tailwind CSS, applying padding, borders, and shadows to enhance the UI.

Challenges:

1. **Responsiveness:** Ensuring that the grid layout adapts well to various screen sizes.
2. **Performance:** Handling a large number of related products efficiently without causing slow rendering times.

Solution:

1. The component uses Tailwind's utility classes to create a responsive design, ensuring the grid layout adjusts based on screen size.
2. For performance optimization, images are rendered with a fixed width/height, preventing layout shifts and improving load time.

● Header and Footer Components

Description:

- The Header component includes:
 - Contact Info: Email and phone number.
 - Language and Currency Selectors.
 - Navigation: Links to home, about, products, etc.
 - Search Bar with suggestions based on fetched products.
 - Icons for User, Wishlist, and Cart with item counts.
 - Responsive Hamburger Menu for mobile views.



- The Footer component includes:
 - Logo and Contact Info with a sign-up form.
 - Categories: Links to product categories.
 - Customer Care: Links for customer support.
 - Social Media: Icons for Facebook, Instagram, and Twitter.



Implementation:

- **React Icons** are used for the navigation and other sections (e.g., user, cart, wishlist).
- **State Management:** **useState** is used for managing states like menu visibility, search query, and filtered products.
- **Sanity Client:** Fetches product data for search functionality.
- **Mobile Responsiveness:** Hamburger menu toggles for smaller screens.

Challenges:

- **Handling dynamic search with filtered product suggestions.**
- **Ensuring mobile responsiveness for the menu and layout.**

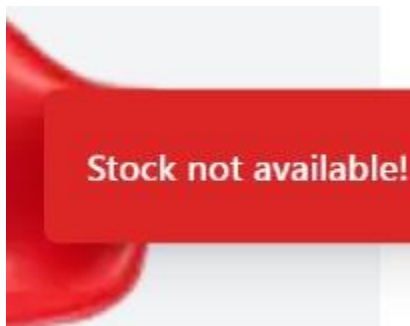
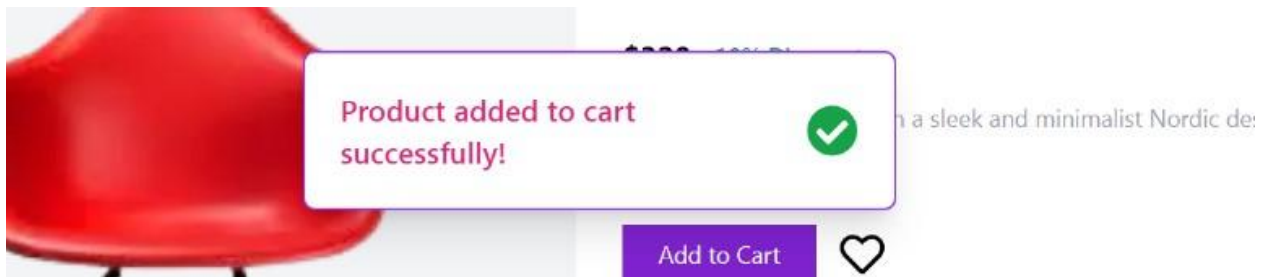
Solution:

- **Search Filter:** Filters products based on the name and description, showing results in a dropdown.
- **Responsive Design:** Uses CSS Grid and Flexbox for layout, and a hamburger menu for mobile screens.

- **Dynamic Fetching:** Fetches product data asynchronously using Sanity client.
- **Notification Component for Success/Error**

Description:

Displays success or error messages with icons. It is triggered by user actions like adding to the cart.



Implementation:

- Accepts **message** and **type** ("success" or "error") as props.
- Uses Tailwind CSS for styling and transitions.
- Centers the notification on the screen.

Challenges:

- Ensuring proper positioning and smooth animation.
- Managing display timing for user clarity.

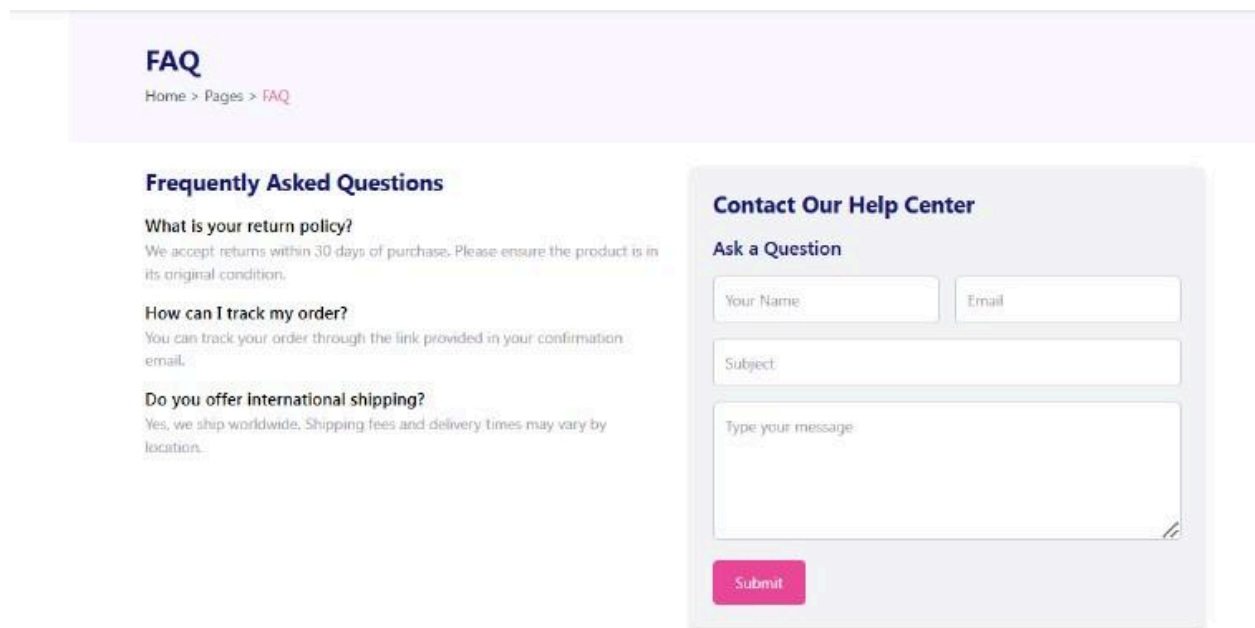
Solution:

Shows a notification after an action, e.g., adding an item to the cart, indicating success or error.

● FAQ and Help Center Component

Description:

This component displays a FAQ section with commonly asked questions and an interactive form for users to contact the help center.



The screenshot shows a web interface for a FAQ and Help Center. At the top, there's a purple header with the text "FAQ" and a breadcrumb "Home > Pages > FAQ". Below this, the page is divided into two main sections. The left section, titled "Frequently Asked Questions", contains three questions with answers: "What is your return policy?", "How can I track my order?", and "Do you offer international shipping?". The right section, titled "Contact Our Help Center", contains a form with the heading "Ask a Question". The form has input fields for "Your Name", "Email", and "Subject", followed by a large text area for "Type your message". A pink "Submit" button is at the bottom of the form.

Implementation:

- **FAQ Section:** Displays a list of frequently asked questions and their answers.
- **Help Center Form:** Allows users to ask questions by providing their name, email, subject, and message.

Challenges:

- Ensuring the layout is responsive across different screen sizes (desktop/mobile).
- Managing form state and validation for the help center section.
- Keeping the FAQ content dynamic and easy to maintain.

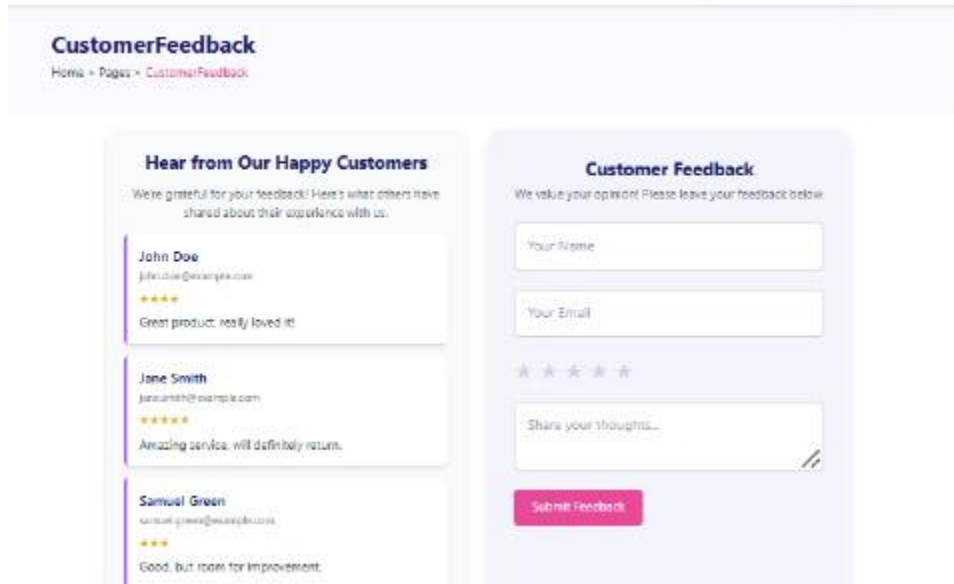
Solution:

Helps users find answers to common queries and offers a way to directly contact customer support for further assistance.

● **Customer Feedback Page**

Description:

This page enables users to submit feedback through a form and view previously submitted feedback. The feedback includes the user's name, email, rating, and comments.



Implementation:

- **Feedback Form:** Users can enter their name, email, rating (from 1 to 5 stars), and comments.
- **Feedback List:** Displays previously submitted feedback from users with their ratings and comments.
- **State Management:** Uses React's **useState** to manage the feedback list and form state.

Challenges:

- **Dynamic Feedback Management:** Ensuring that feedback data is properly managed and displayed.
- **Form Validation:** Ensuring proper validation and clearing of the form after submission.
- **Responsive Layout:** Making the layout responsive for different screen sizes and ensuring proper alignment of feedback sections.

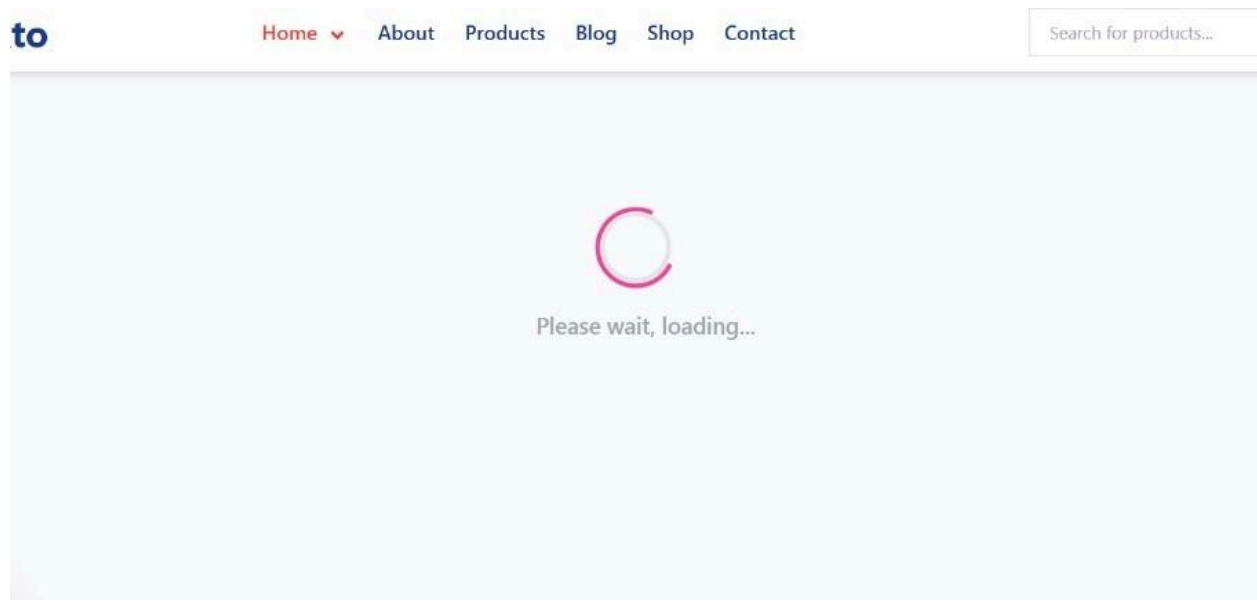
Solution:

The page helps businesses gather customer feedback, improving user engagement and experience. It provides a clean interface for users to submit their reviews while showcasing others' feedback.

● Loading Component

Description:

The **Loading** component is a reusable React component designed to display a visually appealing loading indicator with a spinner animation and a message. It serves as a placeholder while data is being loaded or a process is being executed in a Next.js or React application.



Implementation:

Features:

By Areeba Bano

- **Spinner Animation:**
 - The spinner consists of two concentric circles.
 - The outer circle spins using a CSS **animate-spin** animation.
 - The inner circle remains static for a polished visual effect.
- **Responsive Layout:**
 - The **flex** and **min-h-screen** utilities from Tailwind CSS ensure the spinner and text are vertically and horizontally centered.
- **Loading Message:**
 - A pulse animation (**animate-pulse**) is applied to the text for added dynamism.

• Dynamic Routing (Shop & Product Routes)

Description:

Implement dynamic routes for both a shop and a product page. Each route will handle dynamic content based on the **id** parameter in the URL, which can be used to fetch and display details of specific products.

Implementation:

1. Shop Route (**/shop/[id]/page.tsx**):
 - Displays the shop with products.
 - The dynamic route is defined under **/shop/[id]**.
 - URL format: **/shop/[id]** where **[id]** is a dynamic segment for each product.

Shop

Home > Pages > Shop

Ecommerce Accessories & Fashion Item

About 9,620 results (0.62 seconds)

Per Page:

12

Sort By:

Best Match

View:

Grid

Product Brand

- ☒ Coaster Furniture
- ☒ Fusion Dot High Fashion
- ☒ Unique Furniture Restor
- ☒ Dream Furniture Flipping
- ☒ Young Repurposed
- ☒ Green DIY Furniture

Discount Offer

- ☒ 20% Cashback
- ☒ 5% Cashback Offer
- ☒ 25% Discount Offer



Dictum morbi



\$20.99

~~\$29.99~~



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Magna in est adipiscing in phasellus non in justo.



Sodales sit



\$37.99

~~\$49.99~~



2. Product Route (/product/[id]/page.tsx):

- Displays detailed information about a product.
- The dynamic route is defined under /product/[id].
- URL format: /product/[id] where [id] is a dynamic segment for each product's detail.

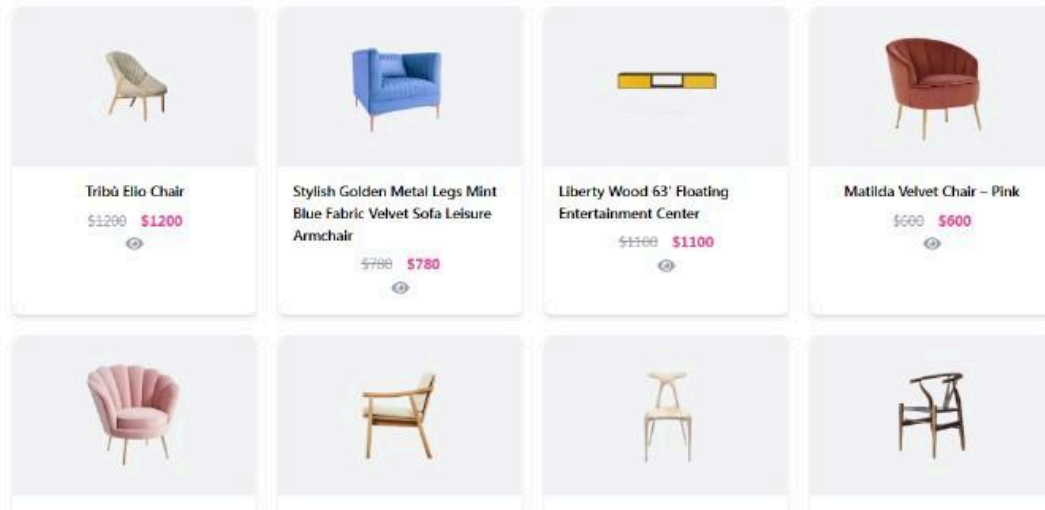
Products

Home > Pages > Products

Ecommerce Accessories & Fashion Item

About 9,620 results (0.62 seconds)

Per Page: 12 Sort By: Best Match View: Grid



5. Data Sanity and Database Management

Since we are using Sanity CMS as our backend, ensuring that the data fetched is correct and up-to-date is crucial. We perform regular sanity checks on the data returned from the CMS API to guarantee its integrity.

Sanity Check Procedures:

1. **Data Validation:** Ensures that all required fields (e.g., product name, price, image) are present.
2. **Data Consistency:** Verifies that the fetched product data is consistent with what is expected (e.g., no missing fields, correct data types).
3. **API Health Checks:** Periodically checks if the API is responsive and returning valid data.

By Areeba Bano

6. Conclusion

The E-Commerce Web Application offers a comprehensive solution for building an online store with essential e-commerce functionality. By leveraging Next.js and React, the application delivers a smooth, responsive user experience while utilizing Sanity CMS for dynamic content management. The system is designed with a focus on scalability, ease of use, and flexibility for future enhancements.

Key Achievements:

- Successfully implemented dynamic routing using Next.js.
- Managed global state with React Context API for seamless navigation across components.
- Integrated localStorage to persist cart and wishlist data.

Future Enhancements:

- Integration of a live payment gateway.
- Advanced filtering and sorting functionalities.
- User authentication and authorization.

This documentation offers an in-depth look at the core components of the E-Commerce Web Application, presenting a clear and structured overview of how each part of the application was built and the challenges faced during development.

Day 4 Checklist

Task	Status
Frontend Component Development	[✓]
Styling and Responsiveness	[✓]
Code Quality	[✓]
Documentation and Submission	[✓]
Final Review:	[✓]