Day 4 - Building Dynamic Frontend Components For Hekto Ecommerce

Mazz Ather - 476344

Objective

On Day 4, the focus is on designing and developing dynamic frontend components that display marketplace data fetched from Sanity CMS or APIs. This task emphasizes creating modular, reusable components and follows real-world practices to build scalable, responsive web applications.

1 . Build Dynamic Frontend Components To Display Data From Sanity CMS

To build dynamic frontend components for my Hekto E-Commerce project, I connected my Next.js app with Sanity CMS. I configured the Sanity client and

used GROQ queries to fetch essential data, including product ID, image, name, stock, and price.

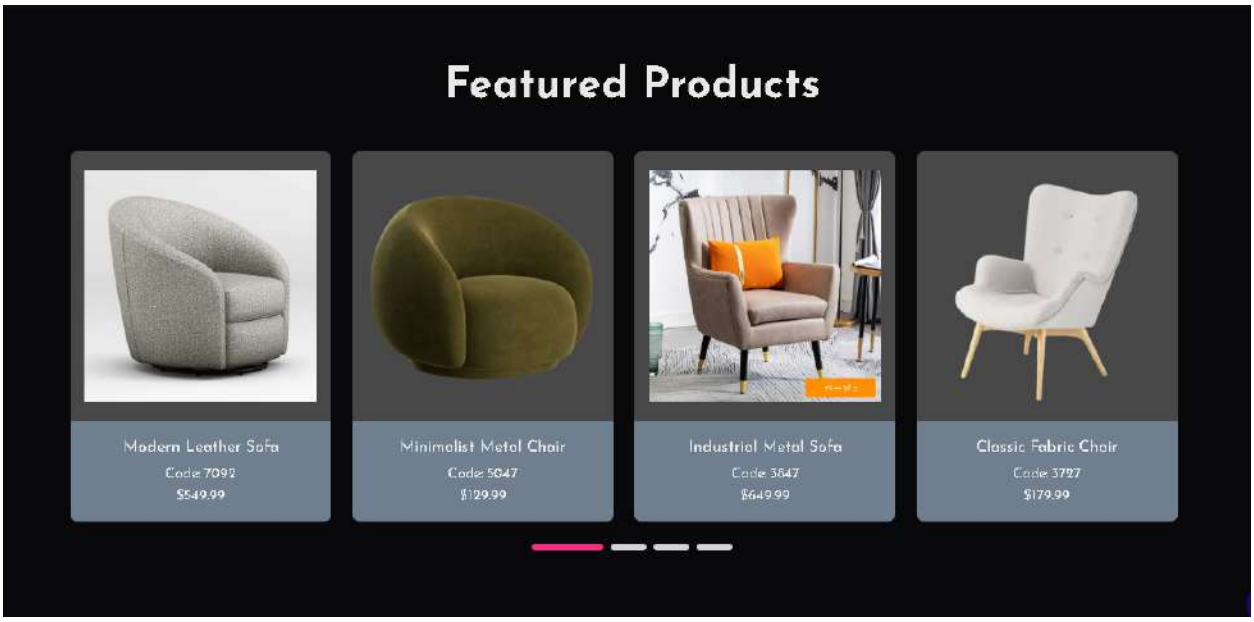
```
const query = `*[_type == "products"]{
  _id,
  productName,
  productDescription,
  price,
  prevPrice,
  stock,
  productImage,
  tag,
  shipmentArray[] {
    trackingId,
    deliveryStatus,
    estimatedDeliveryDate
  }
}`;
```

After fetching data, I mapped it into frontend components, enabling flexible, dynamic displays that automatically reflect the latest information without manual code updates. Utilizing Sanity's real-time data enhanced the user experience with consistently fresh and easily editable content.

```
 {products.map((product) => {
            const selectedImage = product.productImage;
            return (
              <div key={product._id} className="min-w-[250px] sn
                <div className="relative bg-[#e6e6e8] dark:bg-[#
                  {selectedImage && (
                    <Image
                      src={urlFor(selectedImage).url()}
                      alt={product.productName}
                      fill
                      className="object-contain p-4"
                    />
```

```
                )}

                                    <Link href={`/featuredProduct
                  <button
                    className="absolute bottom-4 left-1/2 -tra
                  >
                    View Details
                  </button>
                </Link>
              </div>

              <div className="bg-[#FFFFFF] dark:bg-[#708090] p
                <h3 className="text-[#FB2E86] dark:text-[#EAE
                  {product.productName}
                </h3>
                <p className="text-[#2F1AC4] font-[550] dark:t
                  Code: {Array.from({ length: 1 }, () => Math
                </p>

                <p className="text-[#2F1AC4] font-[550] dark:t
              </div>
            </div>
          );
        })}
```

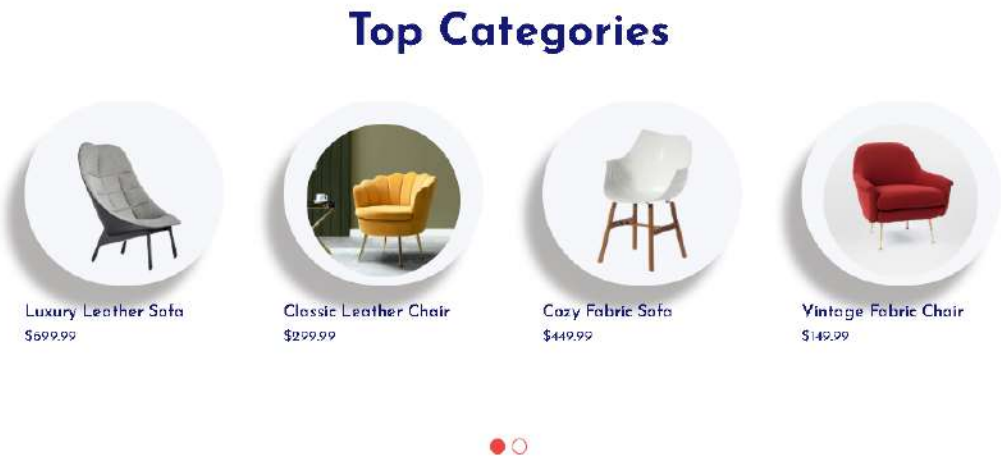
As a result



2. Implement reusable and modular components.

I created components such as Featured Products ,latest Product Trending Products, product grids, and individual product displays, ensuring they can be reused across multiple pages without rewriting the same code.

By creating a reusable `Products` components, I can display individual products anywhere within the app without duplicating the code. This keeps the codebase modular and easier to maintain.



3. Understand and apply state management techniques.

I used **React's** `useState` and `useEffect` hooks to manage the product data.

- `useState` is used to declare a state variable that holds the product data (like name, price etc).

```
const [products, setProducts] = useState<ProductType[]>([]);
```

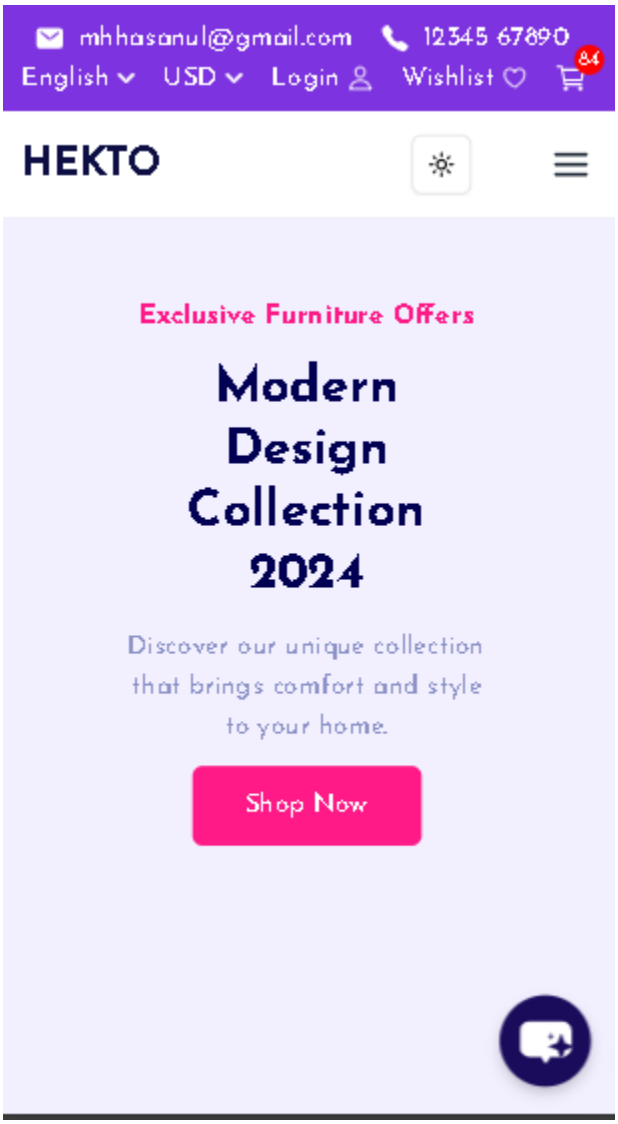
- `useEffect` is used to fetch the product data from the backend (Sanity CMS) when the component mounts.

```
useEffect(() => {
  // Fetch products when the component mounts
  const getProducts = async () => {
    const fetchedProducts = await fetchProducts();
    const filteredProducts = fetchedProducts.filter(product => 

    // Update the state with the fetched products
    setProducts(filteredProducts);


  getProducts();
}, []); // Empty dependency array ensures this runs once when th
```

When the data is fetched, it updates the state, and React automatically re-renders the component with the new data.

4. Learn the importance of responsive design and UX/UI best practices.

Throughout this process, I ensured that the product listing components were responsive on all devices, providing an optimal user experience for mobile and desktop users alike.



5. How I Replicated Professional Workflows for Real-World Client Projects:

1. Modular Components:

I followed the best practice of breaking the project into reusable components, making the code scalable and easier to maintain.

2. State Management with Hooks:

I used React's `useState` and `useEffect` to manage the application state efficiently. For larger projects, I also implemented Redux for state management to handle complex states across different components.

Error Handling and Edge Case Management:

For a production-ready application, I incorporated error handling and managed edge cases

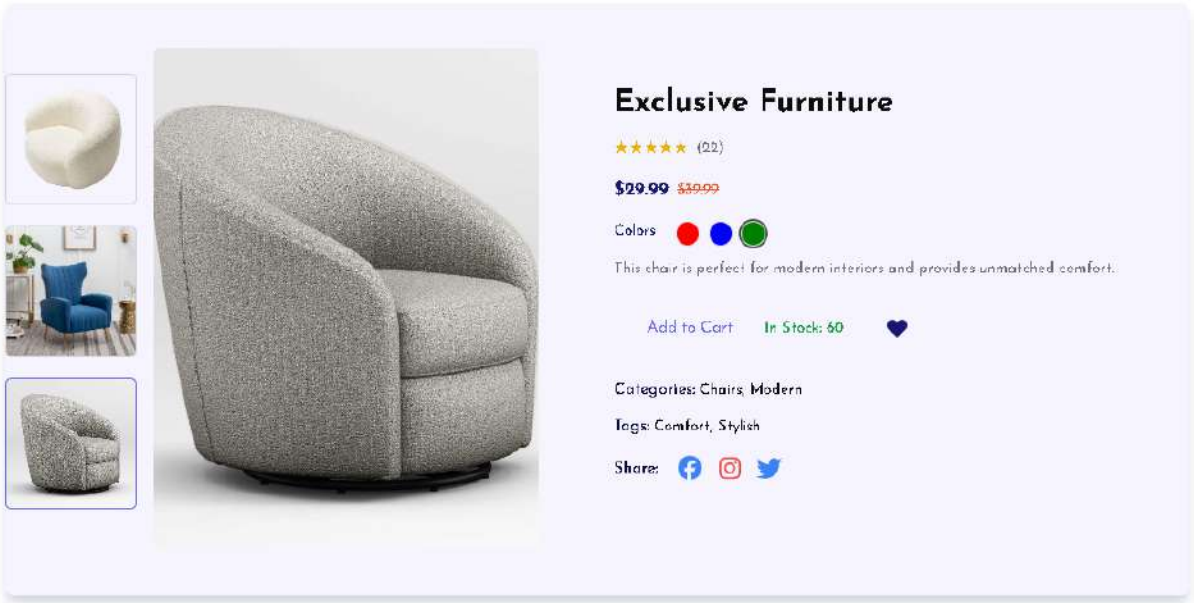
```
if (typeof window !== "undefined") {
   try {
     const serializedWishlist = localStorage.getItem('wishlist
     return serializedWishlist ? JSON.parse(serializedWishlist
   } catch (error) {
     console.error('Error loading wishlist from localStorage:',
     return undefined;
   }
 }
```

6. Create Individual Product Detail Pages Using Dynamic Routing in Next.js

To create individual product detail pages using dynamic routing in Next.js, I created a dynamic route in the `src/app/FeaturedProduct/[id]/page.tsx` file. This structure allows for each product to have its own unique URL based on the product ID, and it dynamically loads the product details from a backend API or CMS (such as Sanity).

I included Necessary Fields

you can also change images



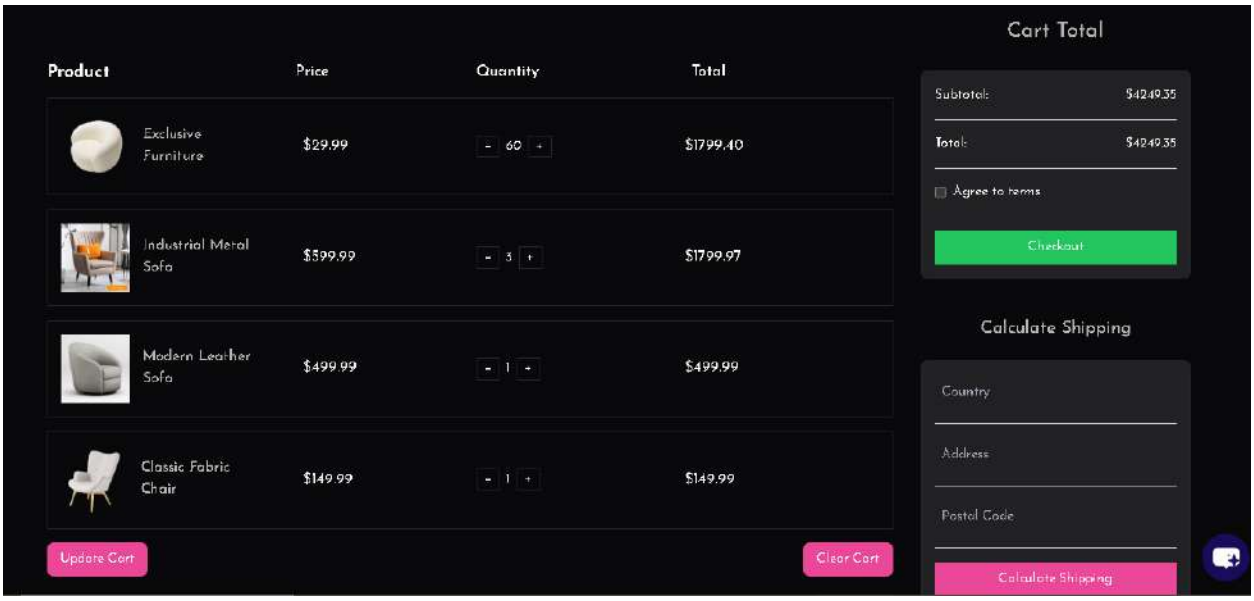
7. Cart Component:

1. **Handling Stock and Cart Button State:**

- The "Add to Cart" button is conditionally enabled or disabled based on the available stock. When the stock is less than or equal to zero, the button becomes disabled to prevent users from adding more items than are available.
- Once a product is added to the cart, the stock count is decremented accordingly, ensuring that the available quantity reflects the cart's current state.

2. **Displaying Added Items, Quantity, and Total Price:**

- The Cart component tracks the products that are added to the cart, showing key details such as the product name, price, quantity, and the total price.
- It displays the products in a list format with options to modify the quantity or remove the product from the cart.
-



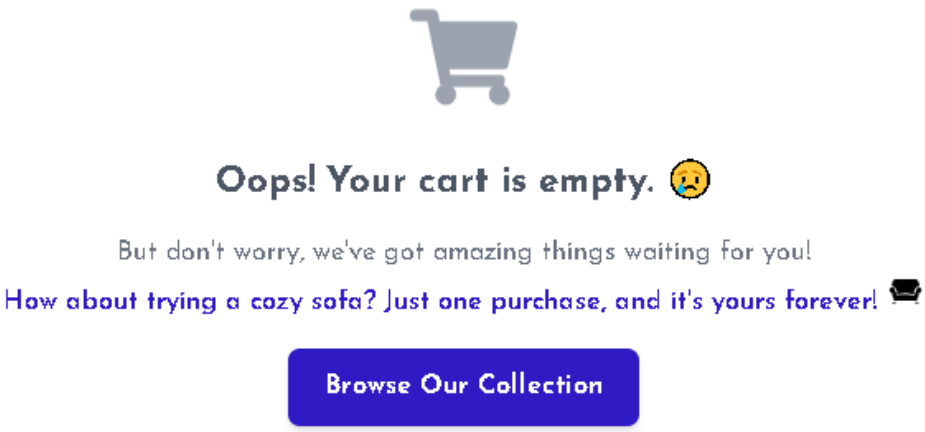
You can increase the product quantity up to the available stock and decrease it to a minimum of one. If you try to reduce the quantity below one, the product will be removed from the cart. Additionally, adjusting the quantity with the "+" or "–" buttons will dynamically update the total price displayed on the right-hand side.

If the cart page is empty, I implemented user-friendly code that displays a message encouraging users to browse our collection. Below this explanation, I have included code snippets to illustrate this functionality.

```
<div className="w-full  min:h-[50vh] flex items-center justify
{cart.items.length === 0 ? (
  <div className="h-full flex flex-col justify-center items-ce
    <FaShoppingCart className="text-7xl text-gray-400 mb-4 ani
    <p className="text-2xl text-gray-600 dark:text-white font-
      Oops! Your cart is empty. <span className="text-yellow-5
    </p>
    <p className="text-gray-500 text-lg">
      But don&apos;t worry, we&apos;ve got amazing things wait
      <br />
      <span className="font-semibold text-[#2F1AC4]">
        How about trying a cozy sofa? Just one purchase, and i
        <span className="ml-2 text-2xl text-black inline-flex
          <GiSofa />
```

```
            </span>
          </span>
        </p>
        <Link
          href="/pages/shopLeft"
          className="px-6 py-3 bg-[#2F1AC4] text-white text-lg for
        >
          Browse Our Collection
        </Link>
```

wanna see how it looks like ???



Same As for Wishlist

8. Notification Component

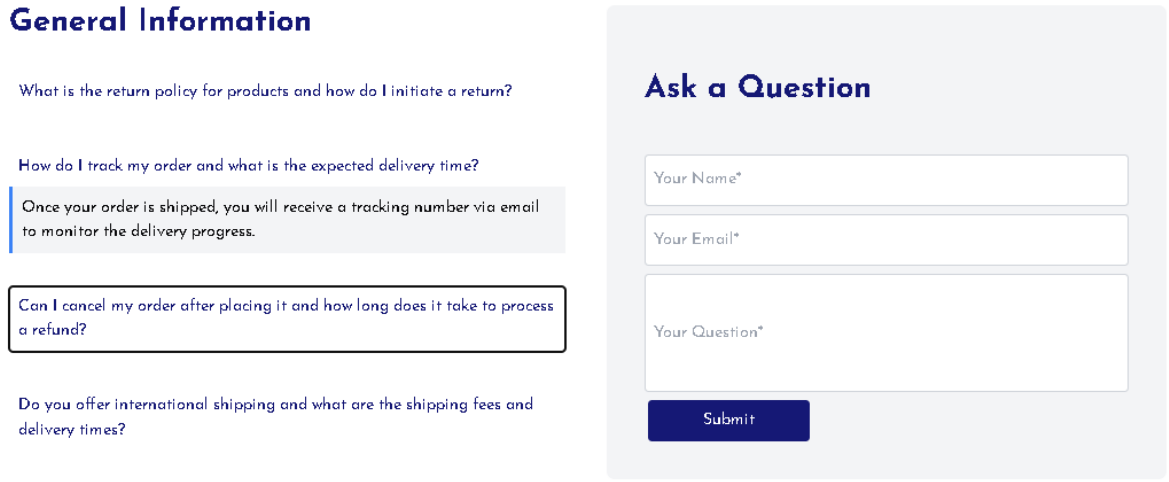
I implemented a beautifully designed, well-structured toast notification system that seamlessly triggers after adding an item to the wishlist or cart. The notifications are elegant, user-friendly, and provide instant feedback with clear messages, enhancing the overall user experience.

https://prod-files-secure.s3.us-west-2.amazonaws.com/4e64d7ab-fcbf-4a0d-925f-c8d42f4a67ed/f62940f7-c7af-4039-bbc6-4bc9e65a81e7/2025-01-20_01-12-44.mp4

```
toast.success(
    <div className="flex items-center space-x-4">
      <FaYahoo size={24} className="text-[#FB2E86]" />
      <div>
        <h4 className="font-semibold text-lg text-green-500">S
        <p className="text-sm text-gray-200">
          You added <strong>{product.productName}</strong> to
        </p>
      </div>
    </div>
```

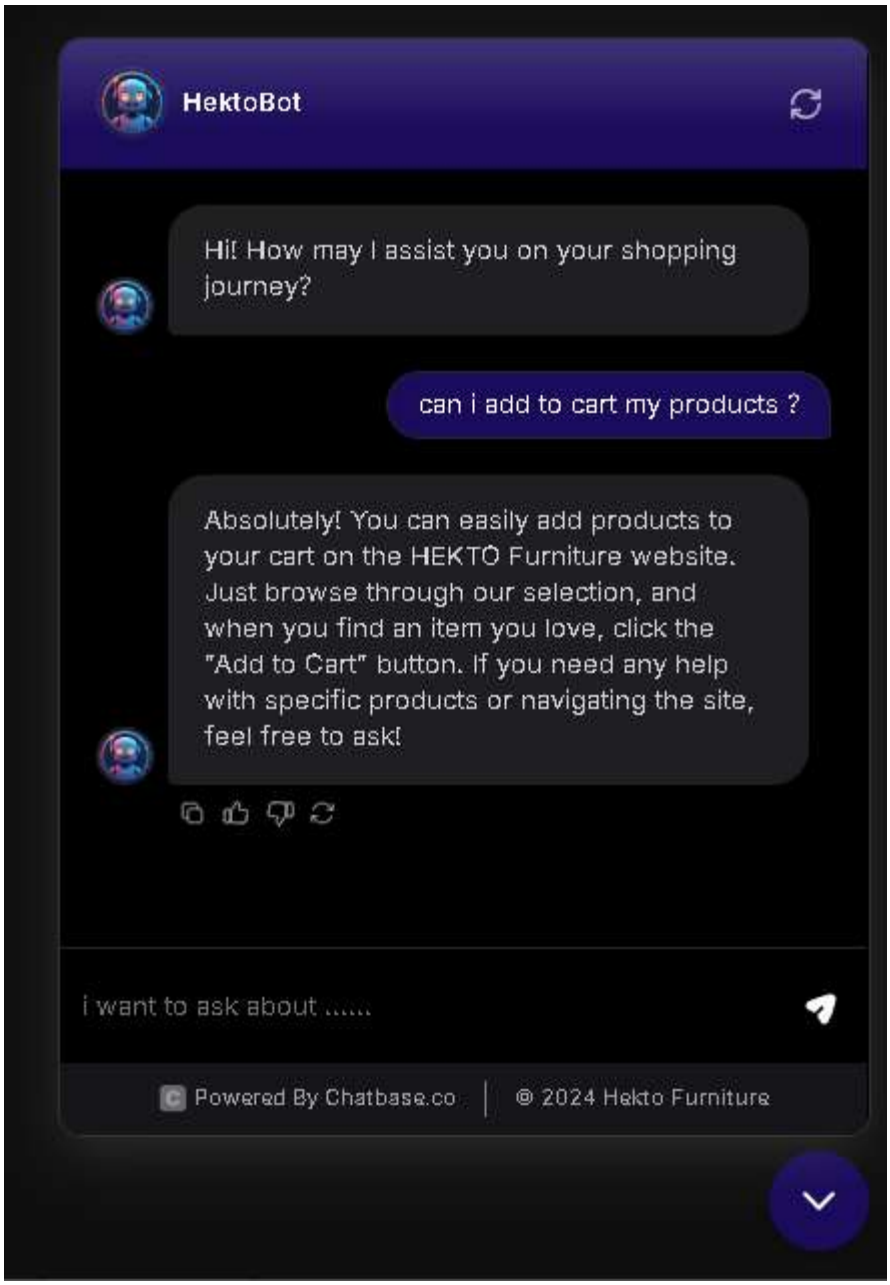
9. FAQ and Help Center Component

I created a comprehensive and beautifully organized FAQ and Help Center to address common questions and provide users with quick, clear, and accessible solutions, ensuring a seamless and supportive experience.



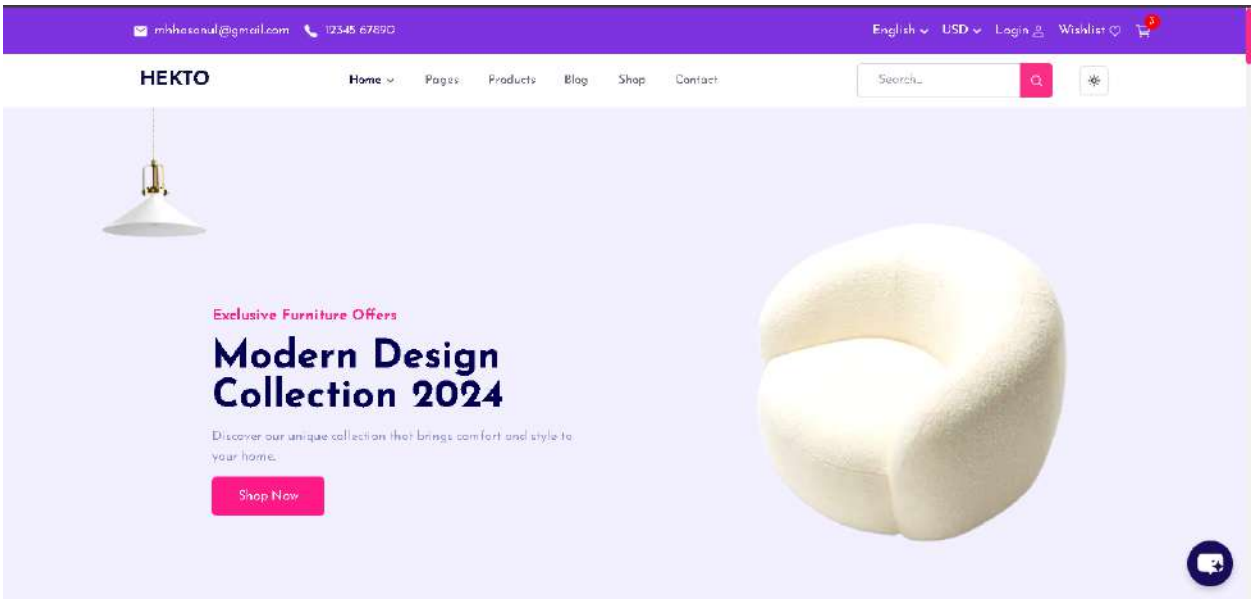
10. Personal AI Chatbot Integration

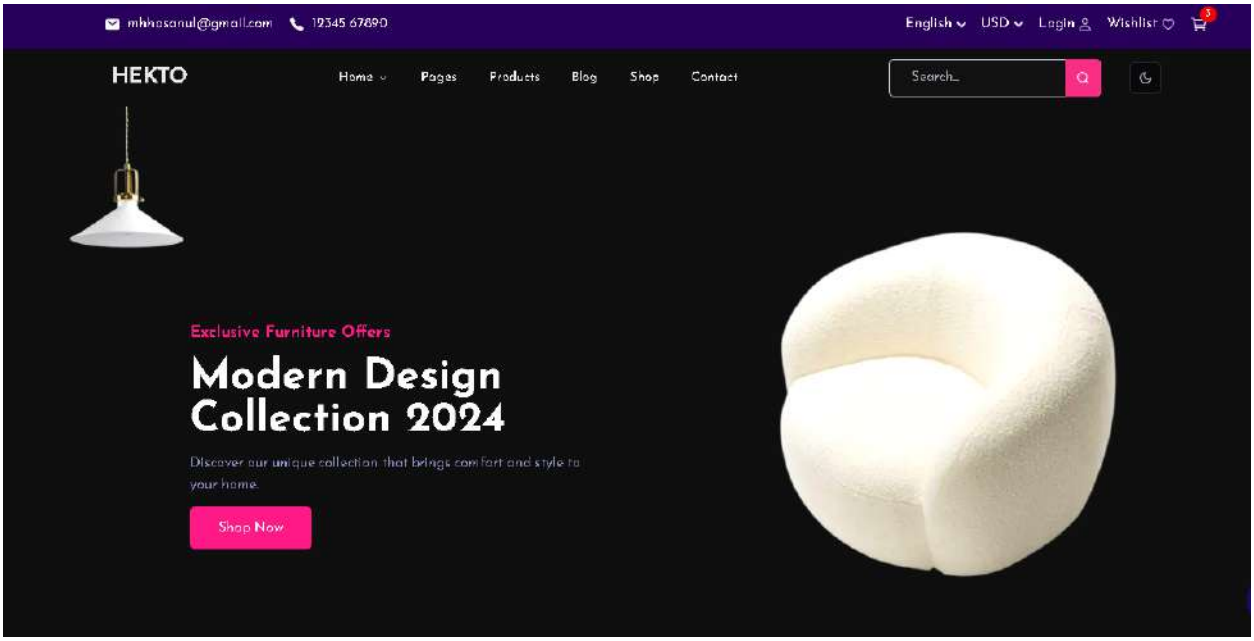
I developed a personalized AI chatbot to enhance user engagement and streamline support. This intelligent assistant is designed to provide instant responses, assist with queries, and deliver a tailored user experience. The chatbot seamlessly integrates with websites or platforms, making interactions efficient and accessible for users.



11. Seamless Theme Toggle: Light and Dark Modes

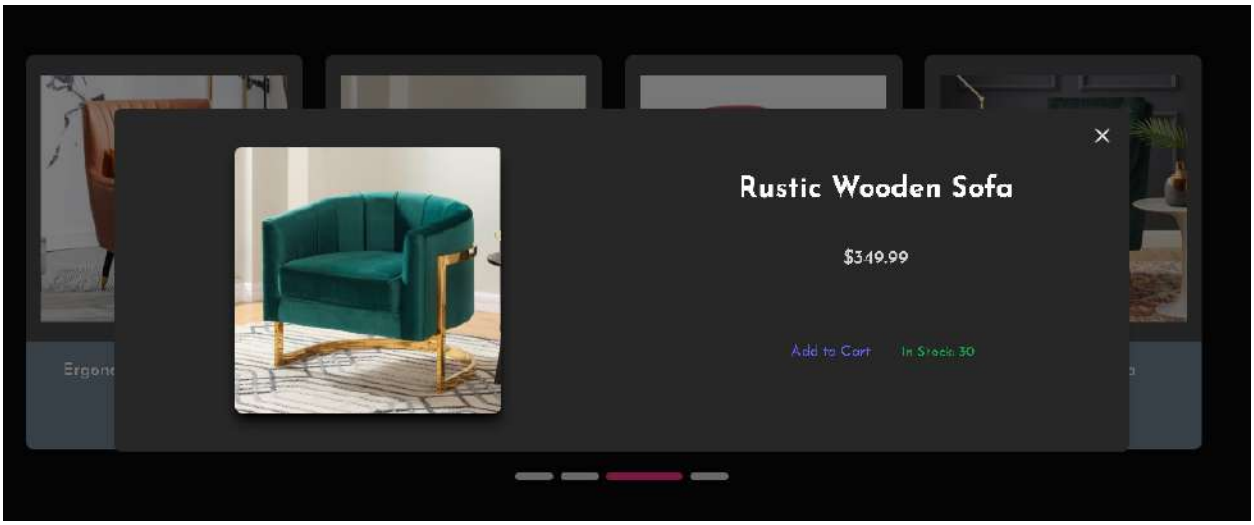
I implemented a smooth and intuitive theme toggle feature, allowing users to switch effortlessly between light and dark modes. This functionality enhances user experience by catering to individual preferences and providing optimal readability in any lighting condition. The transition is visually appealing, ensuring a cohesive and modern design across both themes.





12. Interactive Zoom Feature for Enhanced Viewing

I integrated a dynamic zoom functionality to enhance the product viewing experience. On hover, a sleek zoom icon (🔎) appears, indicating the feature's availability. With a simple click, users can view the product in greater detail without leaving the current page. This interactive feature provides a seamless and immersive way to explore product details, ensuring convenience and clarity.



Conclusion:

Today, I gained a better understanding of how important modular and scalable design is in frontend development. By creating dynamic components and ensuring they're responsive across devices, I've established a strong foundation for building a professional marketplace. Integrating data from APIs and CMSs was a hands-on experience that will be invaluable in real-world projects.

Working with Redux on the add-to-cart feature also gave me a clearer understanding of state management, making my marketplace feel more interactive and user-friendly. I focused on building reusable components, applying UX/UI best practices, and ensuring my design worked seamlessly across different devices. This whole experience has prepared me to tackle real-world client projects, as I've replicated professional workflows and learned essential frontend development practices that will be crucial moving forward.