BUILDING DYNAMIC FRONTEND COMPONENTS FOR YOUR MARKETPLACE

Key Components:

- Search Bar:
- Product Listing Component:
- Product Detail Component:
- Cart Component:
- Wishlist Component
- Checkout Work Flow Component:
- Product Comparison Component:

1. Search Bar Component:

The search bar uses dynamic routing in Next.js to fetch and display products based on user input. A GROQ query retrieves matching products from Sanity CMS, showing details like title and price. If no results are found, a fallback message appears.

Push query function:

```
const router = useRouter();
const serachResultHandler = () =>{
  router.push(`/search/${query}`)
}
```

```
    <div className="flex items-center gap-4">
      <div className="hidden lg:flex items-center border border-gray-300 rounded-md overflow-hidden">
        <input
          type="text"
          placeholder="Search..."
          className="px-3 py-2 outline-none w-full"
          value={query}
          onChange={(e) => setquery (e.target.value)}
          onKeyDown={(e) => {
            if (e.key === "Enter") {
              handleSearch(); // Call the search function when "Enter" is pressed
            }
          }}
        />
        <button className="bg-gray-200 px-3 py-2 hover:bg-gray-300" onClick={serachResultHandler}>
        <Image
            src={"/icon-search.png"}
            alt="Heart Icon"
            height={26.81}
            width={26.33}
            />
        </button>
```

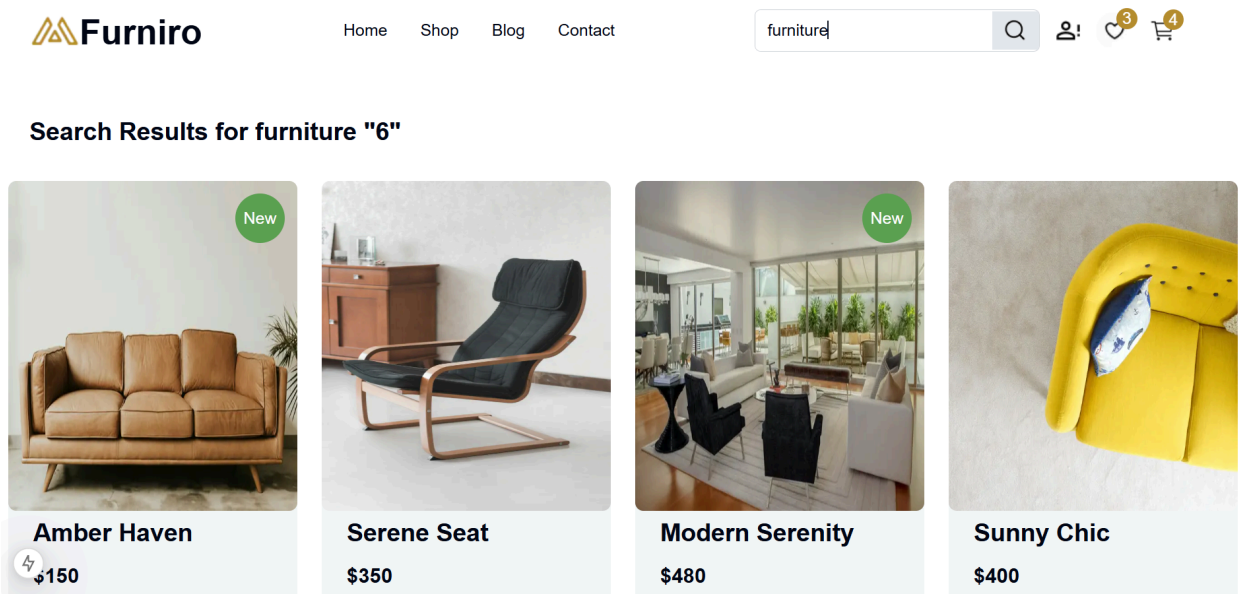
Components > Header.tsx

```
  const { query } = useParams();
  const [products, setProducts] = useState<Product[] | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchProductData = async () => {
      if (!query || typeof query !== "string") {
        console.error("Invalid query parameter:", query);
        return;
      }
      try {
        setLoading(true);
        // Fetch product data using Sanity client
        const productFetchData: Product[] = await client.fetch(
          groq`*[_type == "product" && (title match $query || $query in tags[])]{
            _id,
            title,
            isNew,
            description,
            discountPercentage,
            price,
            productImage,
            tags
            }`,
            { query : query as any}
          );

          if (productFetchData.length > 0) {
            setProducts(productFetchData);
          } else {
            console.error("No products found for the given query.");
          }
        } catch (error) {
          console.error("Error fetching product data:", error);
        } finally {
          setLoading(false);
        }
      };

      fetchProductData();
    }, [query]);
```

Search > [query] > page.tsx

Output:

Users can search for products by entering a name or product tag in the search bar. The dynamic routing fetches and displays matching results, ensuring a seamless search experience.



2. Product Listing Component:

The Product Listing page uses GROQ queries to fetch products from Sanity CMS, displaying titles, prices, images, and descriptions for a seamless browsing experience with up-to-date data.

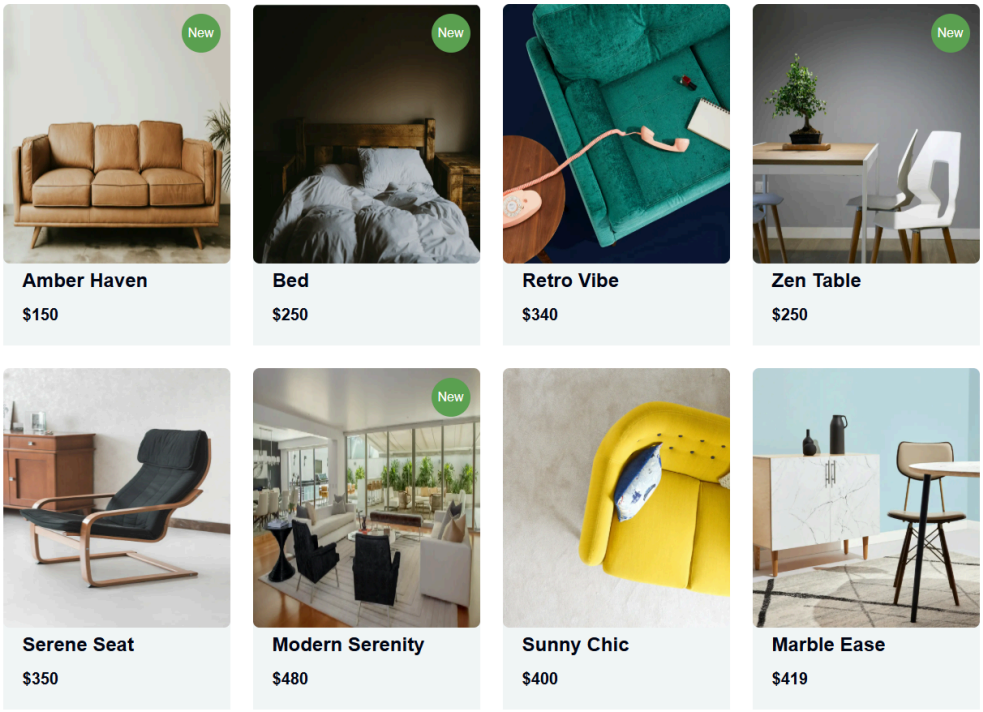
```
const [product, setProduct] = useState<Product[]>([]);

useEffect(() => {
  async function productData() {
    const productFetchData: Product[] = await client.fetch(groq`*[_type == "product"]{
      _id,
      title,
      isNew,
      description,
      discountPercentage,
      price,
      productImage,
      tags
    }`);
    setProduct(productFetchData);
  }
  productData();
}, []);
```

Fetching data from Sanity

```
return (
  <>
    {product.map((element: Product) => (
      <div key={element._id}>
        {element.productImage && (
          <Image
            src={urlFor(element.productImage).url()}
            alt="Single Images"
            className="rounded-lg w-full md:h-[320px] h-[280px]"
            height={391}
            width={481}
          />
        )}

        <h4 className="font-semibold text-lg sm:text-2xl font-poppins">
          {element.title}
        </h4>

        <p className="font-poppins font-semibold text-base sm:text-xl">
          ${element.price}
        </p>
      </div>
    ))}
  </>
);
```

Display Data

Output:


New
Amber Haven
$150
New
Bed
$250
Retro Vibe
$340
New
Zen Table
$250
Serene Seat
$350
New
Modern Serenity
$480
Sunny Chic
$400
Marble Ease
$419

3.Product Detail Component:

The Product Details page leverages dynamic routing to display data for a specific product. Using props, the product ID is retrieved and mapped to fetch and render detailed information such as the title, price, image, and description, ensuring an efficient and dynamic display of the selected product's data.

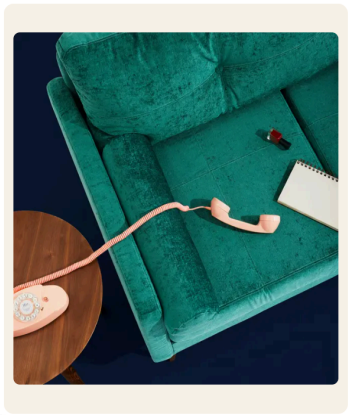
Linking into product Detail page

```
<>
  {product.map((element: Product) => (
    <Link href={`/shop/${element._id}`}>
      <div key={element._id}>
        {element.productImage && (
          <Image
            src={urlFor(element.productImage).url()}
            alt="Single Images"
            className="rounded-lg w-full md:h-[320px] h-[280px]"
            height={391}
            width={481}
          />
        )}

        <h4 className="font-semibold text-lg sm:text-2xl font-poppins">
          {element.title}
        </h4>

        <p className="font-poppins font-semibold text-base sm:text-xl">
          ${element.price}
        </p>
      </div>
    </Link>
  ))}
</>
```

Filter data by id

```
const Page = (props: { params: { id: string } }) => {
  const [product, setProduct] = useState<Product | null>(null);
  useEffect(() => {
    const fetchProductData = async () => {
      try {
        // Unwrap the params promise if needed
        const resolvedParams = await props.params;
        const id = resolvedParams.id;
        console.log("Product ID:", id);

        // Fetch product data using Sanity client
        const productFetchData: Product[] = await client.fetch(
          groq`*[_type == "product" && _id == $id]{
            _id,
            title,
            isNew,
            description,
            discountPercentage,
            price,
            productImage,
            tags
          }`,
          { id } // Pass id as a parameter to avoid string interpolation for security
        );
        if (productFetchData.length > 0) {
          setProduct(productFetchData[0]); // Assuming only one product matches the ID
        } else {
          console.error("No product found for the given ID.");
        }
      } catch (error) {
        console.error("Error fetching product data:", error);
      }
    };
    fetchProductData();
  }, [props.params]);
  console.log(product);
```

Output:



Retro Vibe

$340

5 Customer Review

Introducing RetroVibe—a perfect blend of vintage charm and modern style, designed for those who appreciate timeless aesthetics with a contemporary twist. Inspired by the bold designs and vibrant colors of the past, RetroVibe brings a nostalgic yet fr...

Size

L XL XS

Color

Add To Cart + Compare

Redux Functionality Overview

Redux slice is structured to manage the following features:

1. Cart Management

- Add to Cart: Adds a product to the cart or increments the quantity if it already exists.
- Remove from Cart: Removes a product from the cart.
- Increment/Decrement Quantity: Adjusts the quantity of products in the cart.

```
addToCart: (state, action) => {
  const isPresent = state.cart.find((item: any) => {
    return item._id == action.payload._id;
  });
  if (isPresent) {
    // update the quantity not add one more product
    state.cart = state.cart.map((item: any) => {
      return item._id == action.payload._id
        ? { ...item, quantity: item.quantity + 1 }
        : item;
    });
  } else {
    state.cart.push({ ...action.payload, quantity: 1 });
  }
},
removeFromTheCart: (state, action) => {
  state.cart = state.cart.filter((item: any) => {
    return item._id !== action.payload;
  });
},
incrementQuantity: (state, action) => {
  state.cart = state.cart.map((item: any) => {
    return item._id == action.payload._id
      ? { ...item, quantity: item.quantity + 1 }
      : item;
  });
},
decrementQuantity: (state, action) => {
  state.cart = state.cart.map((item: any) => {
    return item._id == action.payload._id
      ? { ...item, quantity: item.quantity - 1 }
      : item;
  });
},
```

2. Liked Products Management

- Like Product: Adds a product to the "Liked" list if it isn't already liked.
- Unlike Product: Removes a product from the "Liked" list.

```
likeProduct: (state, action) => {
  const isLiked = state.likedProducts.find(
    (item: any) => item._id === action.payload._id
  );
  if (!isLiked) {
    state.likedProducts.push(action.payload); // Add to liked products
  }
},
unlikeProduct: (state, action) => {
  state.likedProducts = state.likedProducts.filter(
    (item: any) => item._id !== action.payload
  );
},
```

3. Product Comparison

- Add to Comparison: Adds a product to the comparison list with a maximum limit of two products.
- Remove from Comparison: Removes a product from the comparison list.
- Clear Comparison: Clears the entire comparison list.

```
addToComparison: (state, action) => {
  if (state.comparison.length >= 2) {
    state.notification = "You can only compare 2 products at a time.";
  } else {
    state.comparison.push(action.payload);
    state.notification = ""; // Clear notification if successful
  }
},
removeFromComparison: (state, action) => {
  state.comparison = state.comparison.filter(
    (product) => product._id !== action.payload
  );
  state.notification = ""; // Clear notification on removal
},
clearComparison: (state) => {
  state.comparison = [];
  state.notification = ""; // Clear notification on clear
},
```

4. Selectors

- Access specific slices of the state, such as the cart, liked products, comparison list, and notifications.

4.Cart Component:

The Cart component leverages Redux to manage the add-to-cart functionality efficiently. By wrapping the entire application with the `ReduxProvider`, the cart state becomes accessible throughout the app. This allows seamless addition, removal, and updating of cart items from any component. The Cart component dynamically fetches and displays selected products, showcasing details like title, price, and quantity, ensuring a consistent and responsive shopping experience for users.

Add to Cart Button

```
<button
  onClick={() => {
    dispatch(addToCart(element));
  }}
  type="button"
  className="text-[#D89E00] w-[150px]
>
  Add to cart
</button>
```

Get Cart Data & Calculate Total Price

```
const cart = useAppSelector(getCart);
const dispatch = useAppDispatch();

// calculate total price
let totalPrice = 0;
cart.forEach((item: Product) => {
  totalPrice += item.price * item.quantity;
});
```

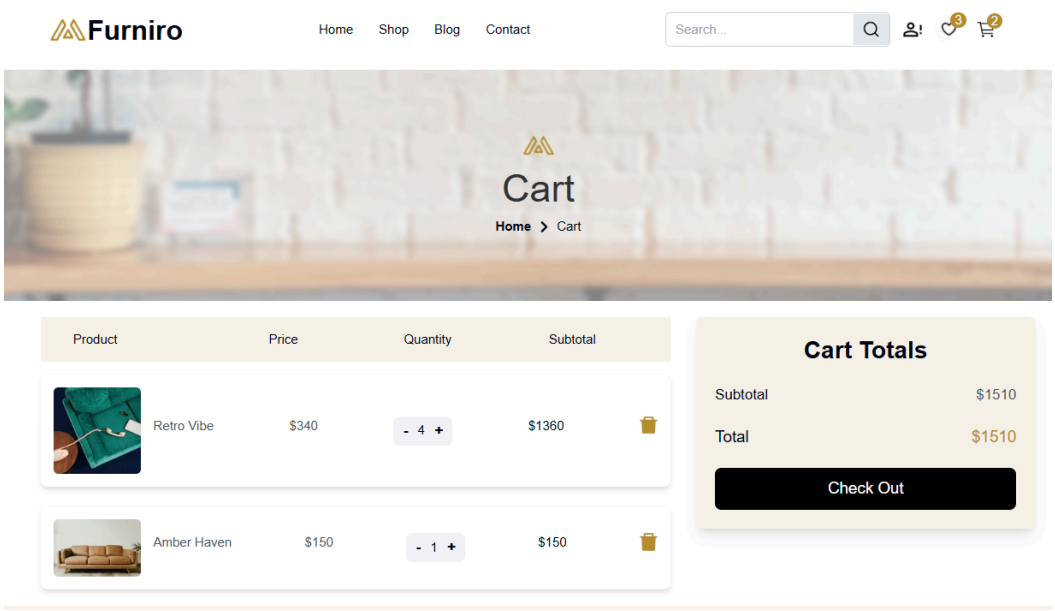
Product Quantity Control

```
{/* Quantity Controls */}
<div className="flex items-center gap-3 bg-gray-1
  <button
    onClick={() => {
      if (val.quantity > 1) {
        dispatch(decrementQuantity(val));
      }
    }}
    className="text-lg font-bold"
  >
    -
  </button>
  <p className="text-base">{val.quantity}</p>
  <button
    onClick={() => {
      dispatch(incrementQuantity(val));
    }}
    className="text-lg font-bold"
  >
    +
  </button>
</div>
```

Remove Product From Cart

```
{/* Delete Button */}
<button
  onClick={() => {
    dispatch(removeFromTheCart(val._id));
  }}
>
```

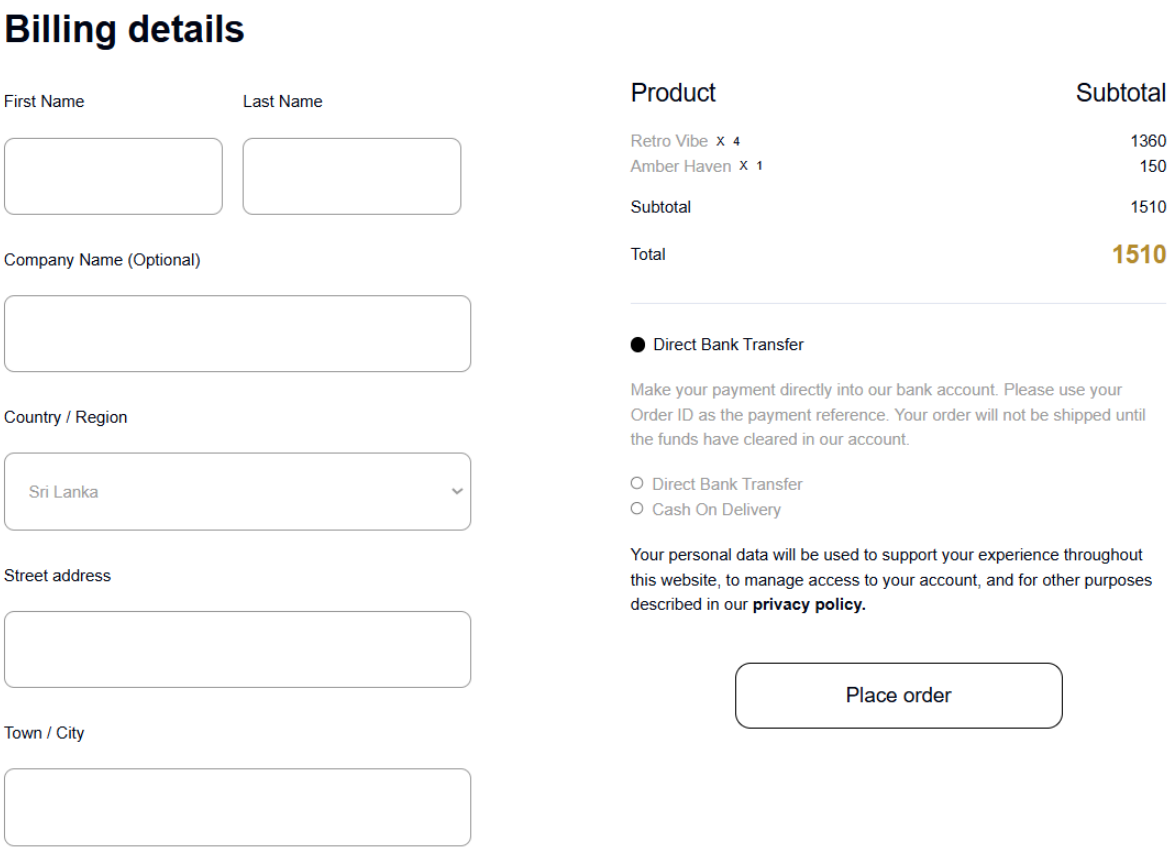
Output:



5.Checkout Work Flow Component:

Our website's checkout process allows customers to review their cart, sign in or check out as a guest, and enter shipping information. Once the details are confirmed, an order summary is shown. Currently, payment, shipment options, and authentication features are not yet integrated but will be added in the future.

Output:



6. Wishlist Component:

The **Wishlist** component on our website allows users to save their favorite products for future reference. Users can easily add products to their wishlist with a single click and can also remove them at any time. This feature provides a convenient way to keep track of items they're interested in without committing to a purchase immediately. Whether a user wants to compare products later or wait for a sale, the wishlist helps them organize and revisit their preferred items whenever they want.

Display Wishlist Product

```
{like.map((element: Product) => (
  <div
    key={element._id}
    className="relative border rounded-lg overflow-hidden shadow-lg
  >
    {/* Image Section */}
    <div className="relative w-full h-[300px]">
      {element.productImage && (
        <Image
          src={urlFor(element.productImage).url()}
          alt={element.title}
          className="w-full h-full object-cover"
          height={300}
          width={300}
        />
      )}
      {element.isNew && (
        <div className="absolute top-2 right-2 bg-green-500 text
          New
        </div>
      )}
    </div>

    {/* Product Info */}
    <div className="p-4">
      <h4 className="text-lg font-bold mb-2">{element.title}</h4>
      <p className="text-gray-700 mb-2">${element.price}</p>
      <button
        onClick={() => dispatch(addToCart(element))}
        className="w-full bg-yellow-600 hover:bg-yellow-700 te
      >
        Add to Cart
      </button>
      <button
        onClick={() => handleRemove(element._id)}
        className="w-full bg-gray-500 hover:bg-gray-600 text-
      >
        Remove
      </button>
    </div>
  </div>
))}
```

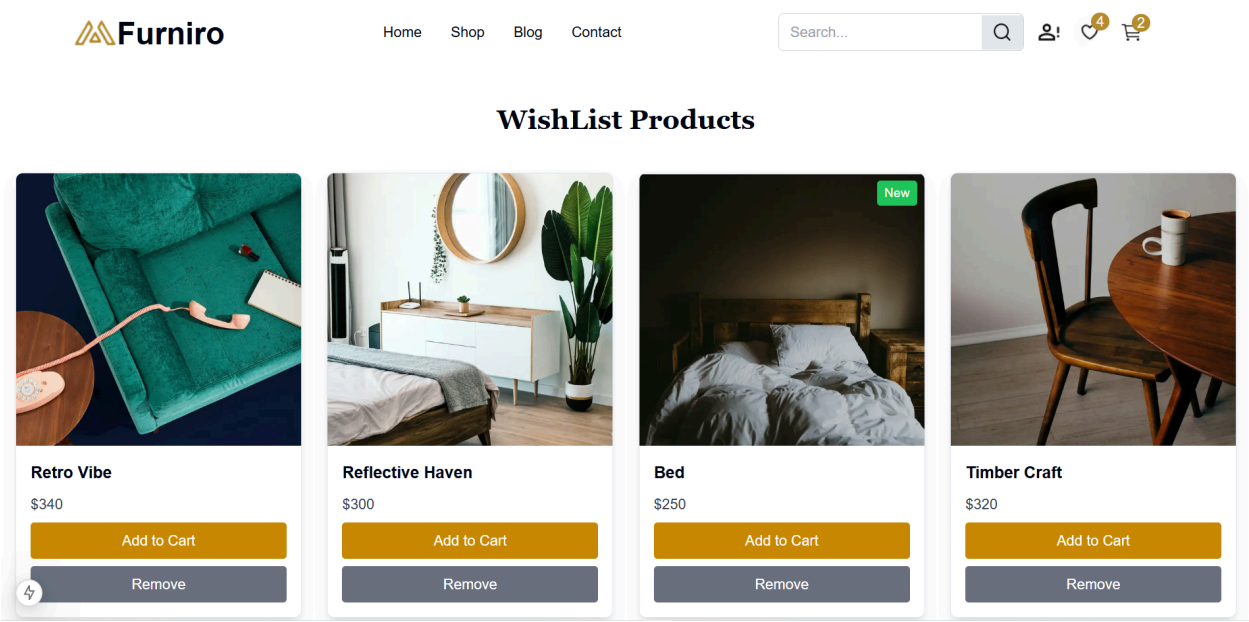
Add Wishlist Product in any page

```
<Link href={""}  onClick={() => {
  dispatch(likeProduct(element));
}}>
  <div className="flex items-center g
    <Image
      src={"/heart-white.png"}
      alt="Like Icon"
      height={16}
      width={16}
    />
    <p className="text-xs sm:text-sm
      Like
    </p>
  </div>
</Link>
```

Get Wishlist Data and Remove product function

```
const like = useAppSelector(getLikedProducts);
const dispatch = useAppDispatch();

const handleRemove = (productId: string) => {
  dispatch(unlikeProduct(productId));
};
```

Output:



7.Product Comparison Component:

The **Product Comparison** page on our website allows users to compare two products side by side. Users can easily select any two products and add them to the comparison tool. Once added, key features, specifications, and prices are displayed together, helping users make an informed decision based on their preferences. This feature simplifies the shopping experience, especially for customers who want to evaluate multiple options before making a final purchase.

Add product for comparison

```
<Link href={"/product-comparison"}>
  <button
  onClick={() => {
    dispatch(addToComparison(product));
  }}
    type="button"
    className="h-16 w-[215px] border gap-5
  >
    + Compare
  </button>
</Link>
```

get comparison product data

```
const dispatch = useDispatch();
const comparison = useSelector(getComparison);

const handleClearComparison = () => {
  dispatch(clearComparison());
};
```

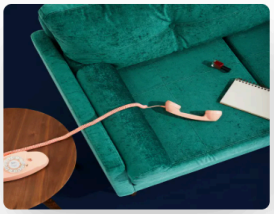
Display Product Comparison products

```
{comparison.map((product: Product) => (
  <div key={product._id}>
    <div className="bg-[#F9F1E7] h-[197px] w-[260px] rounded-lg flex justify-ce
      {product.productImage && (
        <Image
          src={urlFor(product.productImage).url()}
          alt={product.title}
          height={142}
          width={239}
          className="w-[290px] h-[200px] rounded-lg shadow-2xl"
        />
      )}
    </div>

    <div className="text-center w-[260px] sm:text-left">
      <h4 className="text-[20px] font-bold font-poppins">
        {product.title}
      </h4>
      <p className="font-poppins font-medium text-lg/[27px]">
        ${product.price}
      </p>
      <p> {`${product.description.substring(0, 250)}...`}</p>
      <span className="flex gap-2">
        <button
          className="bg-gray-700 w-[160px] h-8 text-white rounded-lg hover:
          type="button"
          onClick={() => {
            dispatch(removeFromComparison(product._id));
          }}
        >
          Romove
        </button>
        <button
          className="bg-yellow-700 w-[160px] h-8 text-white rounded-lg hove
          type="button"
          onClick={() => {
            dispatch(addToCart(product));
          }}
        >
          Add To Cart
        </button>
      </span>
    </div>
  </div>
))}
```

Output:

Go to Product page for more Products

View More

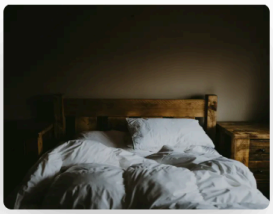


Retro Vibe

$340

Introducing RetroVibe—a perfect blend of vintage charm and modern style, designed for those who appreciate timeless aesthetics with a contemporary twist. Inspired by the bold designs and vibrant colors of the past, RetroVibe brings a nostalgic yet fr...

Romove Add To Cart



Bed

$250

Introducing the Bed—your sanctuary for rest and relaxation, designed with both comfort and style in mind. This timeless piece is crafted to transform your bedroom into a peaceful retreat, offering a perfect balance of support, elegance, and durabilit...

Romove Add To Cart

Add A Product

Choose a Product

Clear Product