

Day 4 - Building Dynamic Frontend Components for Ecommerce and Rental Commerce.

Key Components to Build:

1. Product Listing Component:

Included fields are:

- Product title
- product image
- discounted price
- original price
- available for rent badge
- stock quantity
- discount percent badge
- new badge.

Code Snippets:

```
1  const fetchProductData = async () => {
2    const groqQuery = `*[_type == "furniture"]{
3      _id,
4      title,
5      tags,
6      isNew,
7      availableForRental,
8      stock,
9      description,
10     dicountPercentage,
11     "imageUrls": productImage[].asset->url,
12     slug,
13     isStock,
14     price,
15     rentalPricePerDay
16   }`;
17
18   try {
19     const fetchedData = await client.fetch(groqQuery);
20     return fetchedData;
21   } catch (error) {
22     console.error('Error in Fetching product data: ${error}');
23     return [];
24   }
25 };
26
```

```
26
27  const [product, setProduct] = useState<ProductType[]>([]);
28
29  useEffect(() => {
30    const getData = async () => {
31      try {
32        const data = await fetchProductData();
33        setProduct(data);
34      } catch (error) {
35        console.error(`Error fetching data: ${error}`);
36      }
37    };
38    getData();
39  }, []);
```



Modern Side Table

162 ~~180~~

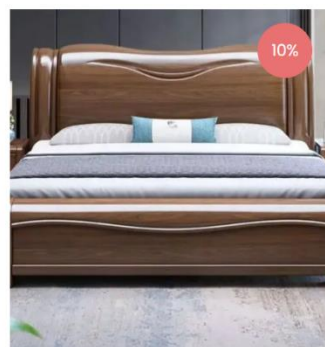
Stock : 20



Modern TV Unit

160 ~~200~~

Stock : 35



King-Size Bed

90 ~~100~~

Available for Rent

Stock : 5



Wooden Desk

340

Out of Stock

2. Product Detail Component:

3. Category Component:

Enable filtering of products by selected categories and Search Bar.

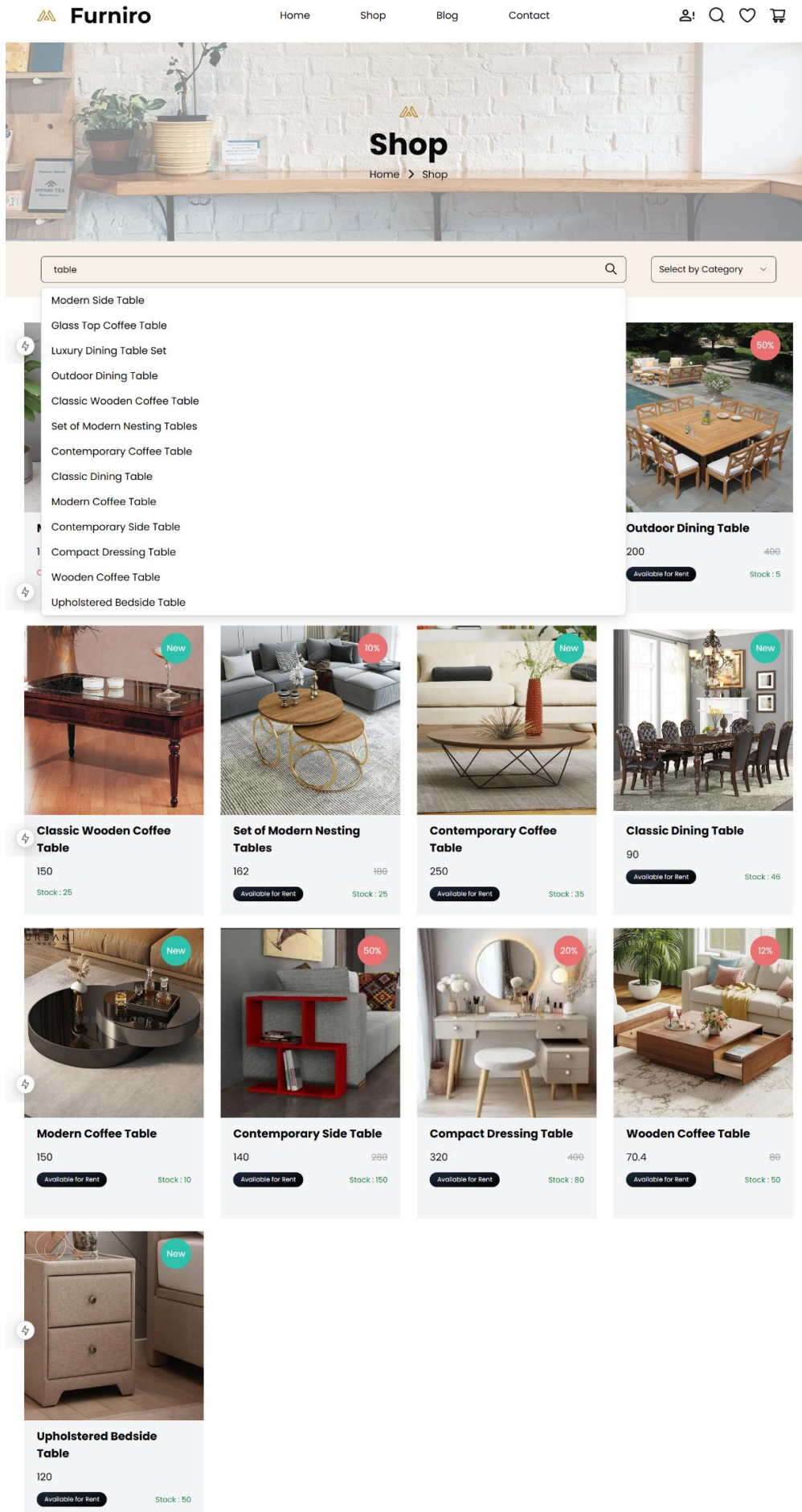
```
1  const [searchQuery, setSearchQuery] = useState("");
2  const handleSearchChange = (e: React.ChangeEvent<HTMLInputElement>) =>
3    setSearchQuery(e.target.value);
4  const handleClearSearchQuery = () => {
5    setSearchQuery("");
6  };
7
8  const [selectedCategory, setSelectedCategory] = useState<string>("");
9  const handleCategoryChange = (category: string) =>
10    setSelectedCategory(category);
11
12  const filteredProducts = product.filter((furniture) => {
13    const matchesSearch = furniture.title
14      .toLowerCase()
15      .includes(searchQuery.toLowerCase());
16    const matchesCategory =
17      selectedCategory === "All" ||
18      furniture.title.toLowerCase().includes(selectedCategory.toLowerCase());
19    return matchesSearch && matchesCategory;
20  });
```

Code Snippets for Search Bar:

```
1  <div className="search relative w-full">
2    <Input
3      placeholder="Search"
4      value={searchQuery}
5      onChange={handleSearchChange}
6      className="border border-slate-800 py-5 px-5 outline-none focus:outline-none w-full"
7    />
8    <IoIosSearch className="w-6 h-6 absolute top-2 right-3" />
9
10    {searchQuery && filteredProducts.length > 0 && (
11      <ul className="absolute left-0 bg-white w-full border border-gray-300 rounded-md shadow-lg mt-2 z-50">
12        {filteredProducts.map((product: ProductType) => (
13          <li
14            key={product._id}
15            className="px-4 py-2 text-black hover:bg-gray-200 cursor-pointer"
16          >
17            <Link
18              href={`~/single-product/${product.slug.current}`}
19              onClick={handleClearSearchQuery}
20            >
21              {product.title}
22            </Link>
23          </li>
24        ))}
25      </ul>
26    )}
27  </div>
```

4. Search Bar:

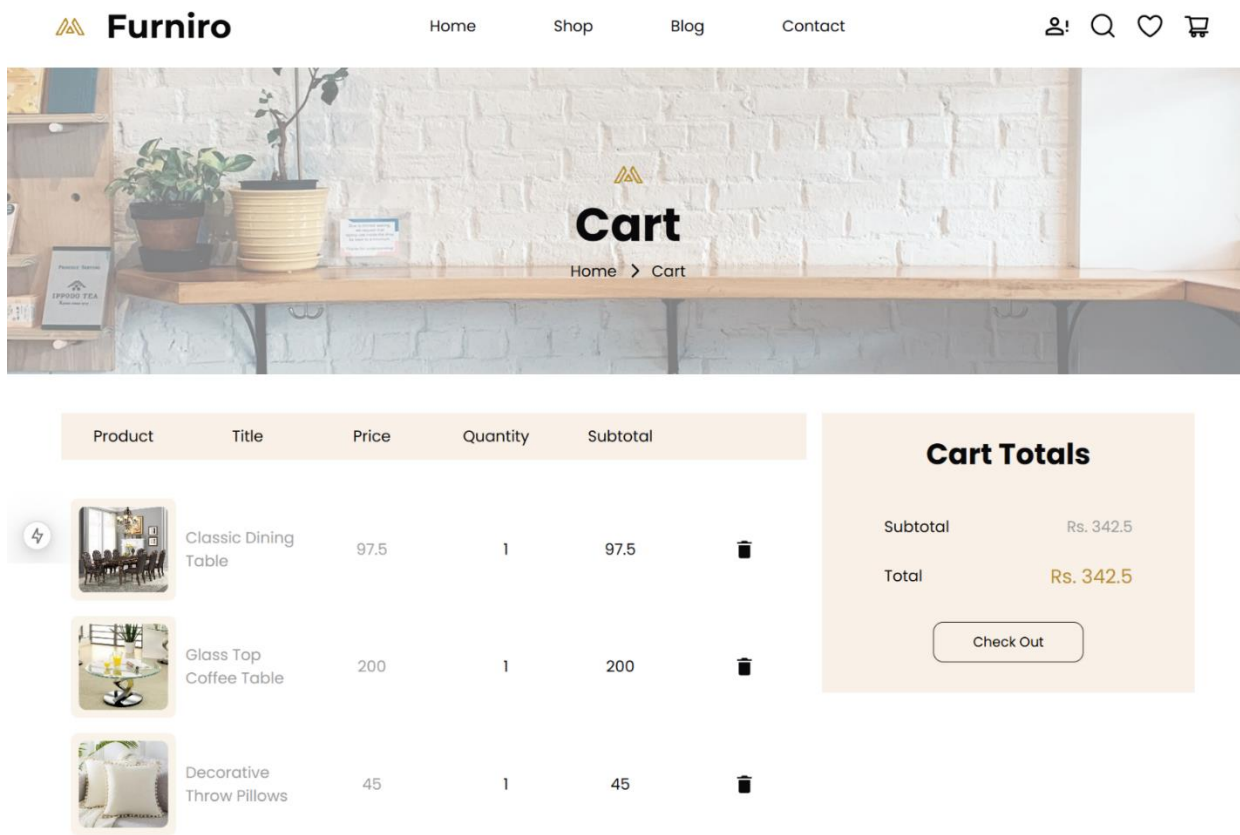
Implement search functionality to filter products by name or tags.



5. Cart Component:

Display added items, quantity, and total price.

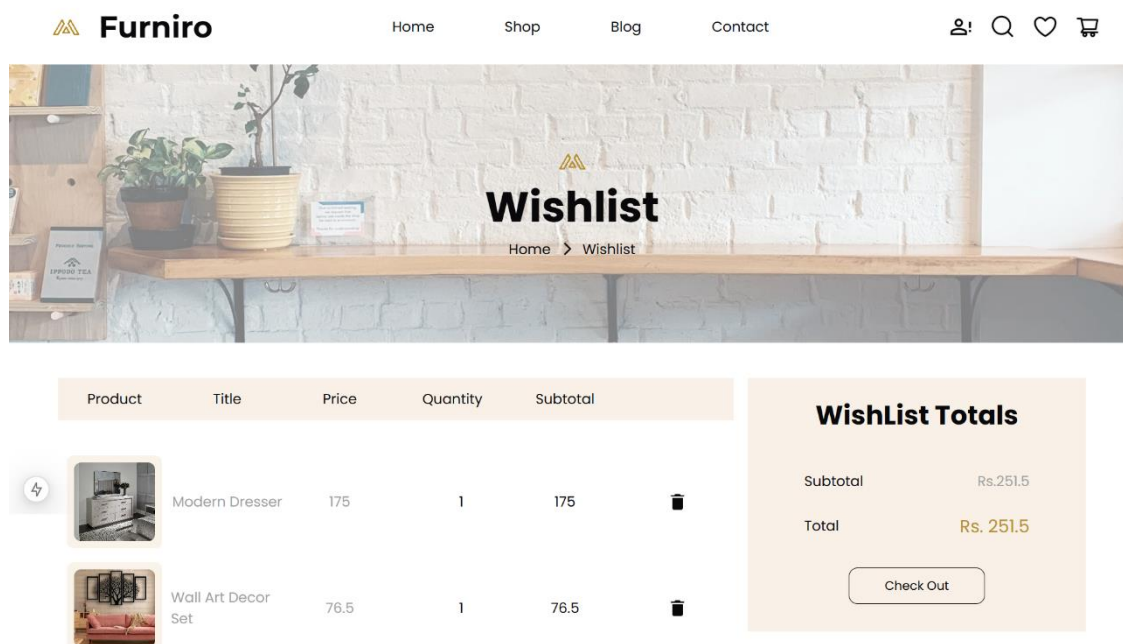
Use state management for tracking cart items.



6. Wishlist Component:


Allow users to save products for future reference.

Use local storage or a global state management tool to persist data.



Additional features: Pagination and Related Products

Related Products




New

110

Available for Rent


Stock : 20



6.7in 7in 9.1in 15.3in

57


Out of Stock



200

Available for Rent

Out of Stock



New

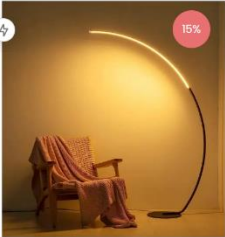
200

Stock : 80

Show More

Search

Select by Category




15%

170

200

Stock : 15




New

110

Available for Rent

Stock : 20




10%

171

100

Available for Rent

Stock : 50




10%

162

100

Out of Stock




20%

160

200

Out of Stock




10%

90

100


Available for Rent

Stock : 5



340

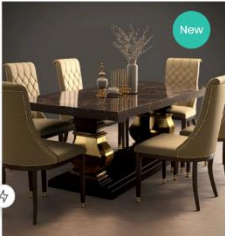
Out of Stock



200

Available for Rent

Out of Stock




New

250

Available for Rent

Stock : 90




60%

140

800

Available for Rent

Stock : 75




20%

224

200

Available for Rent

Stock : 95



30%

210

800

Available for Rent

Stock : 200

Code snippets of Pagination:

```
1  const productsPerPage = 12; // Number of products per page
2  const [currentPage, setCurrentPage] = useState(1);
3  const totalPages = Math.ceil(filteredProducts.length / productsPerPage);
4
5  const paginatedProducts = filteredProducts.slice(
6    (currentPage - 1) * productsPerPage,
7    currentPage * productsPerPage
8  );
9
10 const handlePageChange = (page: number) => {
11   setCurrentPage(page);
12 };
```

```
1  <div className="flex justify-center mt-6 xs:gap-4 xxs:gap-2 gap-1 text-sm">
2    <button
3      className="px-3 py-1 bg-gray-200 rounded-md sm:text-base xs:text-sm text-[0.6rem] md:text-[0.7rem]"
4      onClick={() => handlePageChange(currentPage - 1)}
5      disabled={currentPage === 1}
6    >
7      Prev
8    </button>
9
10   <div className="flex items-center xs:gap-4 xxs:gap-2 gap-1 overflow-x-auto">
11     {Array.from({ length: totalPages }, (_, index) => (
12       <button
13         key={index}
14         onClick={() => handlePageChange(index + 1)}
15         className={`px-3 py-1 sm:text-base xs:text-sm text-[0.7rem] md:text-[0.8rem] ${currentPage === index + 1 ? "bg-[#f1d7b3] text-black" : "bg-gray-200"}`}
16       >
17         {index + 1}
18       </button>
19     ))}
20   </div>
21
22   <button
23     className="px-3 py-1 mx-1 bg-gray-200 rounded-md sm:text-base xs:text-sm text-[0.6rem] md:text-[0.7rem]"
24     onClick={() => handlePageChange(currentPage + 1)}
25     disabled={currentPage === totalPages}
26   >
27     Next
28   </button>
29 </div>
```

A technical report summarizing:

I learned about **state management**, **props**, **routing**, and **context**.

```
1 import React, { createContext, useEffect, useState } from "react";
2
3 export const ProductContext = createContext<CountContext | undefined>(
4   undefined
5 );
6
7 export const ProductProvider = ({ children }: ProductProviderType) => {
```

Steps taken to build and integrate components.

- Planning and Component Breakdown
- Context and State Management
- API Integration
- Routing
- Error Handling
- Adding to Cart/Wishlist Functionality
- Responsive Design
- Code Formatting and Version Control

Fetch product dynamically from sanity.

Home Page → showing 8 Products fetched from sanity.

Shop Page → showing all products.

- Add search bar for searching products.
- Add filter option that user can filter products through category.

SingleProduct Page → from shop card → on click details button → send to → single Product Page

- Here all the details are showing about product.
- This is dynamic routing I have done through slug.

Fetching and integrating dynamic data from APIs like Sanity.

Managing application state (like using useState, useContext).

Testing and debugging to ensure seamless integration.


```
1  useEffect(() => {
2    const fetchSingleProductData = async () => {
3      try {
4        if (slug) {
5          const groqQuery = `*[_type == "furniture"]{
6            _id,
7            title,
8            tags,
9            isNew,
10           availableForRental,
11           stock,
12           description,
13           dicountPercentage,
14           "imageUrls": productImage[].asset->url,
15           slug,
16           isStock,
17           price,
18           rentalPricePerDay
19         }`;
20         const fetchedData = await client.fetch(groq`${groqQuery}`);
21         const findProduct = fetchedData.find(
22           (singleProductData: ProductType) =>
23             singleProductData.slug.current === slug
24         );
25         if (findProduct) {
26           setSingleProduct([findProduct]);
27         } else {
28           setSingleProduct([]);
29         }
30       }
31     } catch (error) {
32       console.log("Error fetching product data:", error);
33       setError(
34         "We're really sorry, but we couldn't find the product you were looking for."
35       );
36     } finally {
37       setLoading(false);
38     }
39   };
40   fetchSingleProductData();
41 }, [slug]);
42
```

Challenges faced and solutions implemented.

Props dealing is a challenge to handle them in this big project of ecommerce.

I faced challenge when I implementing add to cart functionality. My couple of day incurred in searching solution for this when I clicking on add to cart button it adding double quantity of product. And it is also adding all product on the same product.

Solution:

I found solution that I was sending product to cart through sanity id. But I did not fetch the id. After fetching id multiple product adding successfully done. And for quantity issue, I took help of chatgpt and it also solve the problem. Now product is adding according to the counter quantity.

Best practices followed during development.

My code follows several best practices that contribute to a modular, maintainable, and scalable application. By utilizing the **Context API** for state management, I ensured that product data and related states are easily accessible across my components, which improves code modularity and reusability. Additionally, by working with **TypeScript (TSX)**, I ensured type safety across your app, reducing runtime errors and improving developer experience. My use of **Tailwind CSS** guarantees a responsive design, making the application mobile-first and adaptable to various screen sizes. With **ESLint** and **Prettier**, my code is consistently formatted and follows best practices in terms of readability and maintainability.