Day 04 Building a Dynamic Frontend Components For Marketplace

On Day 4, I focused on designing and developing dynamic frontend components for my marketplace. I worked on creating modular and reusable components to display data fetched from Sanity CMS and APIs. This day helped me learn real-world practices for building scalable and responsive web applications, ensuring my components could be easily maintained and adapted for future needs. I gained hands-on experience in structuring components to handle marketplace data effectively, improving both functionality and user experience.

Key components to Build:

Product Listing Components:

Product Listing Component Documentation

The Newest component fetches and displays a list of products stored in a Sanity CMS. It provides users with a scrollable horizontal carousel view of the latest products and includes features like product details, images, and an "Add to Bag" button for interaction. The component is responsive and styled using TailwindCSS.

Features

1. Dynamic Data Fetching:

Fetches product data using Sanity's GROQ query.

Displays the most recent products sorted by creation date.

2. Horizontal Scrolling:
   Users can scroll through the product list using left/right arrow buttons or by swiping.

3. Interactive UI:
   Displays product image, name, category, price, and rating.
   Includes an "Add to Bag" button for quick actions.

4. Responsive Design:
   Optimized for various screen sizes using TailwindCSS classes.

5. Dynamic Routing:
   Clicking on a product navigates to its detailed page using a dynamic route (/product/[slug]).

   Code Structure

   Data Fetching
   Function: fetchProducts()

   Fetches products from Sanity CMS using the GROQ query.

   Retrieves fields like id, name, price, category, slug, and image URL.

State Management

State Variables:

data: Stores the fetched product data.

Effect Hook:

useEffect fetches product data when the component mounts.

Scrolling Functionality

Functions:

scrollLeft: Scrolls the carousel to the left.

scrollRight: Scrolls the carousel to the right.

Implementation:

Uses scrollBy method on a ref pointing to the carousel container.

Rendering Product Display: Loops through the fetched data and renders individual product cards.

Each card includes:

Product image.
Name (linked to the product detail page).

Category name.

Price.

"Add to Bag" and "Rating" components.

Carousel Controls:

Left and right buttons for smooth scrolling.

Sanity GROQ Query

Purpose: Fetches all products of type product sorted by their creation date in descending order.

Styling

Libraries Used:

TailwindCSS: For layout, spacing, and responsive design.

Lucide Icons: For left and right navigation arrows.

Component Props

No external props are passed to this component.

Dependencies

1. Next.js: For dynamic routing and server-side rendering.
2. Sanity Client: For fetching data from Sanity CMS.
3. TailwindCSS: For styling.

4. Lucide-React: For icons.

5. React Hooks: For state and lifecycle management.

Sample Product Schema in Sanity

Usage Instructions

1. Add products to Sanity CMS following the provided schema.

2. Integrate the component into your Next.js project.

3. Customize styling or functionality as needed.

Future Enhancements

1. Add filtering options (e.g., price range, category).

2. Enable infinite scrolling for larger product lists.

3. Include a loading spinner while fetching data.

4. Implement error handling for data fetching failures.
Img----------

Product Detail Page:

The Product Detail Page allows users to view detailed information about a selected product. This page includes product images, pricing, category, ratings, description, quantity adjustment, reviews, and options to add the product to the cart or proceed directly to checkout. The page is dynamic and fetches product data based on the product slug.

Features and Functionalities

1. Dynamic Data Fetching:

   The getData function retrieves product data from Sanity CMS using GROQ queries.

   The data includes product images, name, price, description, category name, slug, and price ID.

   This ensures each product detail page is unique and dynamically rendered based on the product slug.

2. Image Gallery:

   The ImageGallery component displays carousel of product images.

   Users can view multiple images of the product for better understanding.

3. Product Details:

Category Name:

Displayed above the product name.

Product Name: Highlighted in a bold font.

Ratings: Integrated using the RatingComponent for customer reviews and ratings

4. Pricing Section:

Displays the current price prominently.

Includes a strikethrough of the original price to indicate discounts.

Mentions that the price is inclusive of VAT.

5.Quantity Adjustment:

The Quantity component allows users to increase or decrease the product quantity before adding it to the cart.

6.Add to Bag and Checkout:

Users can add the product to their shopping bag using the AddToBag component.

A "Check Out" button redirects users to the checkout page for completing their order.

7.Shipping Details:

Includes an estimated shipping time using the Truck icon for better user clarity.

8. Product Description:

Provides a detailed explanation of the product features and specifications.

Code Structure

1. Imports:

   Components such as AddToBag, ImageGallery, Quantity, and RatingComponent are imported for modular functionality.

   Icons from the lucide-react library, such as ArrowRight, ArrowLeft, and Truck, enhance the visual appeal.

   The Link component from Next.js handles navigation to the checkout page.

   The ProductPage function uses the params object to access the product slug.

   The getData function fetches product details based on the slug.

2. Dynamic Rendering:

The dynamic property is set to force-dynamic to ensure the page re-renders dynamically for each request.

4. Component Composition:

The layout is divided into a two-column grid using TailwindCSS, making the design responsive and visually appealing.

Technologies Used

1. Next.js:

   Used for dynamic routing and server-side rendering.

   The params object fetches the slug to display product-specific information.

2. Sanity CMS:

   Stores and manages product data, including images, pricing, and descriptions.

3. TailwindCSS:

   Provides a responsive and modern design for the page layout.

4. Lucide-React Icons:

   Adds visually appealing icons for shipping details and navigation.

5. Reusable Components:

ImageGallery, AddToBag, Quantity, and RatingComponent ensure reusability and maintainability.

Page Layout and Design

Responsive Design:

TailwindCSS ensures the layout adapts to various screen sizes, including mobile, tablet, and desktop.

Grid System:

A two-column grid displays product images on the left and details on the right for better usability.

Usage Workflow

1. Users select a product from the product listing page.

2. They are redirected to the product detail page via the dynamic route (/product/[slug]).

3.On this page, users can:

View product images and details.

Adjust the quantity of the product.

Check reviews and ratings.
Add the product to their cart or proceed directly to checkout.

Category-Based Product Listing Page:

This document explains the implementation of a dynamic category-based product listing page in a Next.js project. The page allows users to view products filtered by category. Categories can be accessed via the navigation bar or category buttons.

Core Features

1. Dynamic Product Listing:

   Displays products based on the selected category.

2. Sanity Integration:

   Fetches products and their details from the Sanity CMS.

3. Reusable Components:

   AddToBag:

   Adds products to the shopping bag. Rating: Displays product ratings.

4. Responsive Design:

   Adapts to various screen sizes using TailwindCSS.

Code Explanation

1. Fetching Data from Sanity

The getData function fetches products belonging to a specific category using Sanity's GROQ query.

```
async function getData(cateogry: string) { const query = `*[_type == "product" && category->name == "${cateogry}"] { _id, "imageUrl": images[0].asset->url, price, name, "slug": slug.current, "categoryName": category->name }`; const data = await client.fetch(query, {}, { cache: "no-store" }); return data;
}
```

2. Dynamic Route Handling

The CategoryPage is a dynamic route-based component that utilizes the params.category parameter to determine the category for which products are fetched and displayed.

3. UI Components

Header Section

Displays the category name dynamically:

Our Products for {params.category}

Product Card Each product is rendered with the following details: Image: Rendered using the Image component. Name: Linked to the product's detail page. Category: Shown below the name.

Price: Displayed alongside the product details. Rating and Add to Bag: Included as reusable component