Day 4 - Dynamic Frontend Components

# Day 4 - Dynamic Frontend Components - [NIKE SHOES WEBSITE]

## 1. Introduction

Day 4 focused on building dynamic frontend components to display marketplace data fetched from Sanity CMS or APIs. The goal was to design modular, reusable components and integrate them seamlessly into the marketplace to enhance functionality and user experience.

This document outlines the components built, their functionality, challenges faced, and the solutions implemented.

## 2. Functional Deliverables

### **2.1 Product Listing Page**

**Component Name**: ProductList.tsx

* **Functionality**: Fetches product data dynamically from Sanity CMS and displays the products in a grid layout. Each product card contains key details such as product name, price, image, and stock status.
* **Key Fields**:
  + Product Name
  + Price
  + Image
  + Stock Status
* **Challenges & Solutions**:
  + **Challenge**: Ensuring proper data fetching for dynamic product listings.
  + **Solution**: Used getServerSideProps to fetch product data in real-time from Sanity CMS.

### **2.2 Product Detail Page**

**Component Name**: ProductDetail.tsx

* **Functionality**: Displays detailed information about a single product, including the product price, quantity, description, and the option to add to the cart.
* **Key Fields**:
  + Product Price
  + Quantity (Input for user to select quantity)
  + Product Description
* **Challenges & Solutions**:
  + **Challenge**: Ensuring the correct product is displayed based on dynamic routing.
  + **Solution**: Implemented dynamic routing using Next.js’s [id].tsx feature to fetch product data by ID.

### **2.3 Add to Cart Component**

**Component Name**: AddToCart.tsx

* **Functionality**: Allows users to add products to the shopping cart. The cart is stored in the application state using useState or useContext for global state management.
* **Challenges & Solutions**:
  + **Challenge**: Managing cart state efficiently and ensuring the product data is added correctly.
  + **Solution**: Used the Context API to manage global cart state, enabling the cart to be accessible across all pages.

### **2.4 Checkout Component**

**Component Name**: Checkout.tsx

* **Functionality**: Displays a multi-step form for checkout, including fields for billing and shipping addresses. The order information is stored in Sanity CMS for future reference.
* **Key Fields**:
  + Billing Address
  + Shipping Address
  + Payment Details (mock implementation)
* **Challenges & Solutions**:
  + **Challenge**: Storing user checkout data in Sanity CMS.
  + **Solution**: Used Sanity's API to store order information in a structured way.

### **2.5 Filter Component**

**Component Name**: CategoryFilter.tsx

* **Functionality**: Filters products dynamically based on selected categories. This helps users quickly find products of their choice.
* **Challenges & Solutions**:
  + **Challenge**: Ensuring that the filtered data updates correctly when a new category is selected.
  + **Solution**: Used useState to track the selected category and updated the product list accordingly.

### **2.6 Search Bar Component**

**Component Name**: SearchBar.tsx

* **Functionality**: Allows users to search for products by name or tags. The search results dynamically update as the user types.
* **Challenges & Solutions**:
  + **Challenge**: Efficiently filtering product data based on search input.
  + **Solution**: Implemented a useState for tracking search input and filtered products based on the input.

### **2.7 Header Component**

**Component Name**: Header.tsx

* **Functionality**: Displays the main navigation links (Home, About, Contact), along with the shopping cart and search bar.
* **Challenges & Solutions**:
  + **Challenge**: Ensuring the header is consistent and responsive across devices.
  + **Solution**: Used Tailwind CSS for responsive design and applied a mobile-first approach.

### **2.8 Footer Component**

**Component Name**: Footer.tsx

* **Functionality**: Provides links to important pages, such as Home, About, Contact, and legal pages. Ensures accessibility and responsiveness.
* **Challenges & Solutions**:
  + **Challenge**: Designing a footer that looks good on both mobile and desktop views.
  + **Solution**: Used Tailwind CSS to ensure proper styling and responsiveness.

## 3. Code Deliverables

### **Key Components Code Snippets**

#### **ProductList.tsx:**

tsx

CopyEdit

import { useEffect, useState } from 'react';

const ProductList = () => {

const [products, setProducts] = useState([]);

useEffect(() => {

const fetchProducts = async () => {

const res = await fetch('/api/products');

const data = await res.json();

setProducts(data);

};

fetchProducts();

}, []);

return (

<div className="grid grid-cols-3 gap-4">

{products.map((product) => (

<div key={product.id} className="card">

<img src={product.image} alt={product.name} />

<h3>{product.name}</h3>

<p>{product.price}</p>

</div>

))}

</div>

);

};

export default ProductList;

#### **ProductDetail.tsx:**

tsx

CopyEdit

import { useRouter } from 'next/router';

const ProductDetail = ({ product }) => {

const { query } = useRouter();

if (!product) return <div>Loading...</div>;

return (

<div>

<h1>{product.name}</h1>

<p>{product.description}</p>

<p>${product.price}</p>

</div>

);

};

export default ProductDetail;

## 4. Documentation

### **Challenges Faced & Solutions**

* **Dynamic Data Rendering**: Ensuring that the data fetched from Sanity CMS renders correctly in the frontend components required debugging and optimization of data fetching functions.
* **State Management**: Managing the state across multiple components (cart, wishlist, filters) was challenging. The solution was implementing React Context API for global state management.

### **Best Practices Followed**

* **Reusable Components**: Components like ProductCard and CategoryFilter are designed to be reused across different pages.
* **Responsive Design**: Used Tailwind CSS to ensure the app is fully responsive on mobile and desktop devices.
* **Modular Code**: Kept components modular to allow easy updates and scalability in future development.

## 5. Submission Requirements

### **Screenshots/Screen Recordings**

* Product listing page with dynamic data.
* Individual product detail pages with accurate routing and data rendering.
* Working category filters and search bar.

### **Code Files**

* Code snippets for key components (ProductList.tsx, ProductDetail.tsx, AddToCart.tsx, etc.).
* Scripts for data fetching, dynamic routing, and state management.

### **Documentation**

* Technical report summarizing development steps, challenges, and best practices.
* Submit the documentation in PDF or Markdown format.

## 6. Conclusion

By the end of Day 4, I successfully developed dynamic frontend components for the marketplace, including product listing, product detail pages, cart functionality, and filters. The components were designed to be reusable, responsive, and modular, following best practices for scalability and performance optimization.