Day 4 - Dynamic Frontend Components - [F.A Shop]

1. Introduction

This document outlines the work completed on Day 4 of the e-commerce project, focusing on building dynamic frontend components for the marketplace. The components developed allow the user to interact with product listings, view product details, filter by category, search, and navigate through pages.

2. Functional Deliverables

2.1 Product Listing Page with Dynamic Data

The product listing page displays a dynamic list of products fetched from the backend. The data is rendered using the ProductList component, which dynamically loads products based on categories and displays them in a grid layout. The ProductCard component is used to display individual product details, such as name, image, price, rating, and available colors/sizes.

Screenshots/Screen Recordings:

- The product listing page showing dynamically loaded products from the CMS.
- **Example:**
  - *Product listing page with items displayed*
  - *Pagination and filters in action*

2.2 Individual Product Detail Pages

Each product in the product listing page is clickable and redirects the user to a product detail page. The product detail page is dynamically rendered using the ProductDetail component, where product information such as name, description, price, images, and reviews are fetched from the backend and displayed.

Screenshots/Screen Recordings:

- Individual product detail page with accurate data rendering.

2.3 Category Filters, Search Bar, and Pagination

The SearchBar component allows users to search for products by name, and the CategoryFilter component lets users filter products by categories such as electronics, fashion, and groceries. Pagination allows users to navigate through multiple pages of products.

Screenshots/Screen Recordings:

- *Search bar and filters in action on the product listing page*.
- *Pagination working with dynamic data loading*.

2.4 Additional Features

The implementation of related products is achieved by dynamically fetching and displaying products that are similar to the currently viewed item.

Screenshots/Screen Recordings:

- *Related products section displayed on the product detail page*.

3. Code Deliverables

3.1 ProductCard Component

The ProductCard component is used to display individual products on the listing page. It accepts product props and displays product information such as the name, price, image, and rating.

```
const ProductCard = ({ product }: { product: Product }) => {
  const discountedPrice = product.discountPercent
    ? (product.price * (1 - product.discountPercent / 100)).toFixed(2)
    : null;

  return (
    <div className="border rounded-lg p-4 shadow-sm hover:shadow-lg transition">
```

```
    <Image src={product.imageUrl} alt={product.name} width={295} height={298}
/>
    <h2 className="text-lg font-semibold mt-2">{product.name}</h2>
    {/* Other product details */}
   </div>
  );
};
```

3.2 ProductList Component

The ProductList component fetches and renders all products from the API. It includes logic for handling pagination and category filtering.

```
const ProductList = ({ category }: { category: string }) => {
  const [products, setProducts] = useState<Product[]>([]);

  useEffect(() => {
   fetch(`/api/products?category=${category}`)
    .then(res => res.json())
    .then(data => setProducts(data));
  }, [category]);

  return (
   <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">
    {products.map(product => (
     <ProductCard key={product._id} product={product} />
    ))}
   </div>
  );
};
```

3.3 SearchBar and Filters

The SearchBar component provides a text input for searching products by name, while the CategoryFilter component allows the user to filter products by their category.

```
const SearchBar = ({ onSearch }: { onSearch: (query: string) => void }) => {
  return (
    <input
      type="text"
      placeholder="Search for products..."
      onChange={(e) => onSearch(e.target.value)}
    />
  );
};
```

3.4 API Integration and Dynamic Routing

We use Next.js dynamic routing to show individual product pages and fetch the corresponding product details via an API.

```
export async function getServerSideProps({ params }) {
  const res = await fetch(`https://api.example.com/products/${params.id}`);
  const product = await res.json();

  return { props: { product } };
}
```

4. Documentation

4.1 Steps Taken to Build and Integrate Components

1. **Component Structure:**
    - Designed modular components (ProductCard, ProductList, SearchBar, CategoryFilter) to ensure reusability and maintainability.
    - Used dynamic routing for individual product detail pages and fetched product data via API calls.
2. **API Integration:**
    - Integrated with a Sanity CMS backend to fetch product data dynamically using getServerSideProps for SEO benefits.
    - Ensured that products are filtered and searched based on dynamic data fetched from the CMS.
3. **Routing and Pagination:**

- Used Next.js dynamic routing to display individual product details.
- Implemented pagination in ProductList to limit the number of products displayed per page.

4.2 Challenges Faced and Solutions Implemented

1. **Challenge: Handling Dynamic Routing**
   - Initially, there were issues with dynamic product pages not displaying the correct data.
   - Solution: Implemented getServerSideProps to fetch product data on each request, ensuring accurate data display.
2. **Challenge: State Management for Filters and Search**
   - Filters and search inputs were not responsive, causing issues with state updates.
   - Solution: Used React useState hooks to handle filter and search queries efficiently and trigger re-renders on state change.

4.3 Best Practices Followed

1. **Component Reusability:**
   - Ensured components like ProductCard and ProductList were modular, making them easy to reuse across different pages.
2. **Separation of Concerns:**
   - Separated UI logic (components) from data-fetching logic to improve code readability and maintainability.
3. **Responsive Design:**
   - Used Tailwind CSS utility classes to ensure the layout is fully responsive across mobile, tablet, and desktop screens.