

## Report: Implementing a Dynamic Product Listing Component

**Prepared By:** Taha Baig

**Prepared For:** Project Day 4 - Building Dynamic Frontend Components

### Objective:

The primary objective of Day 4 is to design and develop **dynamic frontend components** that can display marketplace data fetched from **Sanity CMS** or external APIs. This process focuses on modularity, reusability, and applying real-world development practices to build scalable and responsive web applications.

---

## Task Overview

### Objective:

Build a **Product Listing Component** for a marketplace.

### Requirements:

1. Fetch product data dynamically using Sanity CMS or an external API.
2. Display the data in a **grid layout** of cards with the following details:
  - **Product Name**
  - **Price**
  - **Image**
  - **Stock Status**
3. Ensure responsiveness across devices.
4. Implement modularity by breaking the component into smaller, reusable parts.

### Tools & Technologies:

- **Framework:** React or Next.js
  - **CMS:** Sanity CMS
  - **Styling:** Tailwind CSS or plain CSS
  - **State Management:** React Hooks
-

## Implementation Plan

1. **Set Up Data Fetching:**
  - Integrate Sanity CMS or API endpoints to fetch the product data dynamically.
  - Use React hooks (`useEffect`) for data fetching and (`useState`) to store and manage the data.
2. **Design Reusable Components:**
  - Break down the Product Listing Component into smaller parts:
    - **Product Card Component:** Displays individual product details.
    - **Grid Layout Component:** Arranges the product cards in a responsive grid.
3. **Apply Responsive Design:**
  - Use Tailwind CSS or CSS Grid/Flexbox to ensure the grid layout adapts to all screen sizes.
4. **Enhance User Experience:**
  - Highlight important details like stock status with conditional formatting.
  - Add hover effects for better interactivity.

```
1  useEffect(() => {
2      const fetchProducts = async () => {
3          const productsData = await client.fetch(
4              `*[_type == "food"]{
5                  name,
6                  price,
7                  description,
8                  category,
9                  originalPrice,
10                 "image": image.asset->url,
11                 "slug": slug.current,
12             }`
13         );
14         setProducts(productsData);
15         setFilteredProducts(productsData);
16     };
17     fetchProducts();
18 }, []);
```

## 2. Product Detail Component

### Objective:

Develop individual product detail pages using **dynamic routing in Next.js**. These pages will display detailed information about each product, including:

- **Name**
- **Product Description**
- **Price**
- **Category**
- **Stock Availability**

### Implementation Plan:

#### 1. Dynamic Routing:

- Create dynamic routes using the `[id].tsx` file in the `pages/products` directory.
- Fetch product data based on the product ID from a CMS like Sanity or an API.

#### 2. Data Fields:

Each product detail page should include the following fields:

- **Product Description:** A detailed explanation of the product, fetched from the backend.
- **Price:** Displayed prominently for clear visibility.

#### 3. Integration with Product Listing:

- Link each product card in the **Product Listing Component** to its corresponding detail page using the `Link` component in Next.js.

#### 4. Styling and Layout:

- Use Tailwind CSS or plain CSS for a clean and responsive design.
- Ensure the layout highlights the product description and price for user clarity.

```
1  async function Productpage({ params }: { params: { slug: string } }) {
2    const product:IProduct =
3      await client.fetch(`*[_type == "food" && slug.current == $slug][0] {
4        name,
5        description,
6        price,
7        originalPrice,
8        tags,
9        "imageUrl": image.asset->url,
10       "slug": slug.current,
11     }`,{slug:params.slug});
```

## UI Display OF Product Detail Page:



### Country Burger

Delicious burger with fresh ingredients.

#### Flavors

Original

Spicy

Cheesy

#### Sizes

Single

Double

Family

- 1 +

**Rs 650.00**

Add to Cart

### Step 3: Search Bar with Price Filter

#### Objective:

To implement a **search bar** and **price filters** to enhance the product browsing experience.

#### Implementation Plan:


##### 1. Search Bar Functionality:

- Filter products based on their name or associated tags.
- Update the product list in real-time as the user types.

```
1 // Handle search
2 const handleSearch = (event: React.ChangeEvent<HTMLInputElement>) => {
3   const query = event.target.value.toLowerCase();
4   setSearchQuery(query);
5
6   const filtered = products.filter(
7     (product) =>
8       product.name.toLowerCase().includes(query) ||
9       product.description.toLowerCase().includes(query) ||
10      product.category.toLowerCase().includes(query) ||
11      product.slug.toLowerCase().includes(query)
12   );
13   setFilteredProducts(filtered);
14 };
```

#### UI Display :

Oldest ▾



**Pizza**  
Rs 43.00

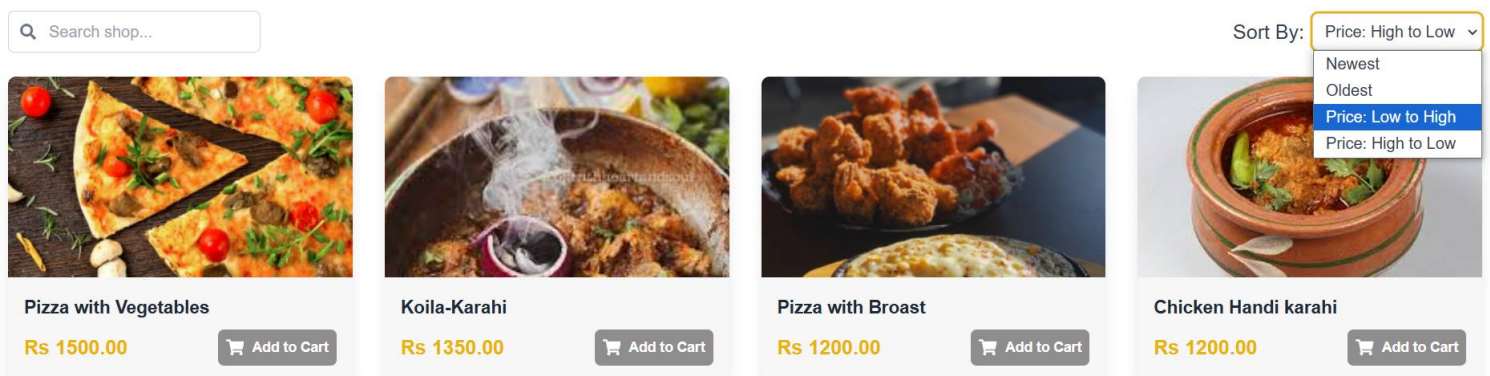
## 2. Price Filtering:

- Add options to sort products by price in **ascending** or **descending** order.
- Combine the price filter with the search bar and category filter for seamless interaction

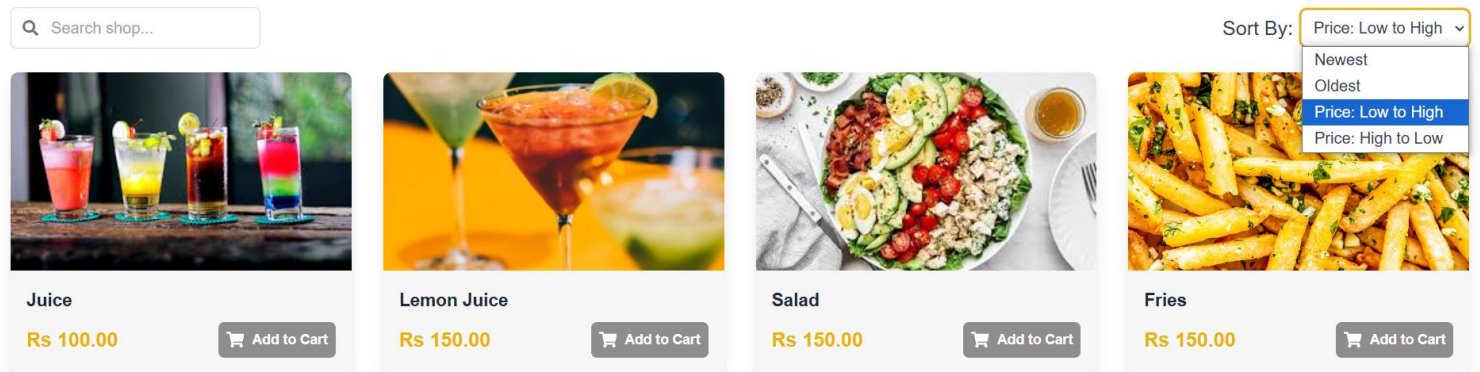
```
1 // Handle sorting
2 const handleSort = (event: React.ChangeEvent<HTMLSelectElement>) => {
3   const sortValue = event.target.value;
4   setSortOrder(sortValue);
5
6   let sortedProduct = [...products];
7   if (sortValue === "lowToHigh") {
8     sortedProduct.sort((a, b) => a.price - b.price);
9   } else if (sortValue === "highToLow") {
10    sortedProduct.sort((a, b) => b.price - a.price);
11  }
12  setFilteredProducts(sortedProduct);
13 };
```

### UI Display:

- **High To Low:**



- **Low To High:**



## Features Implemented:

1. **Search Bar:**
  - Filters products by name or tags in real time.
2. **Price Filter:**
  - Allows sorting products by price (low to high or high to low).

---

## Step 4: Cart Component

### Objective:

To create a **Cart Component** that displays the items added to the cart, their quantity, and the total price of the cart dynamically.

---

## Implementation Plan:

1. **State Management:**
  - Use **React state** or a state management library like Redux for storing cart data.
2. **Cart Data:**
  - Include details for each product in the cart:
    - Product Name
    - Price
    - Quantity
  - Calculate and display the **total price** dynamically based on the items in the cart.



### 3. Cart Interactions:

- Allow users to **increase or decrease the quantity** of items.
- Automatically update the total price when the quantity changes.



```
1
2 // Handle Increment
3 const handleIncrement = () => {
4   const newQuantity = quantity + 1;
5   setQuantity(newQuantity);
6   setCartPrice(newQuantity * product.price); // Update price
7 };
8 // Handle Decrement
9 const handleDecrement = () => {
10  if (quantity > 1) {
11    const newQuantity = quantity - 1;
12    setQuantity(newQuantity);
13    setCartPrice(newQuantity * product.price);
14  }
15 };
16
17 function handleAddToCart() {
18   const cartItem = {
19     slug: product.slug,
20     title: product.name,
21     img: product.imageUrl,
22     price: product.price,
23     quantity: 1,
24   };
25
26   dispatch(addToCart(cartItem));
27 }
```



---

## Your Cart

---



### Salad

Cheese - Large  
Quantity: 1



**Total:** 150 Rs

**Delivery:** 150 Rs

**Grand Total:** 300 Rs

**Proceed to Checkout**

### Features Implemented:

#### 1. Dynamic Item Display:

- Each item in the cart is displayed with its name, price, and quantity.

- Subtotal for each item is dynamically calculated.
  - 2. **Quantity Update:**
    - Buttons to increase (+) or decrease (–) the quantity of an item.
    - Quantity cannot go below 1.
  - 3. **Total Price Calculation:**
    - The total price updates dynamically as items are added or quantities are changed.
  - 4. **Remove Item:**
    - Users can remove individual items from the cart.
-

## Conclusion

On **Day 4** of building dynamic frontend components for a marketplace, the focus was on creating modular, reusable, and responsive components. The following key components were successfully implemented:

1. **Product Listing Component:**
  - Dynamically displayed products in a grid layout with details such as product name, price, image, and stock status.
2. **Product Detail Component:**
  - Built individual product pages using dynamic routing in Next.js, including fields like product description, price, and image.
3. **Search Bar and Filters:**
  - Implemented functionality to filter products by name or tags and added price filters (high to low and low to high).
1. **Cart Component:**
  - Displayed items added to the cart, quantity management, and total price calculation with dynamic updates.