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.

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**

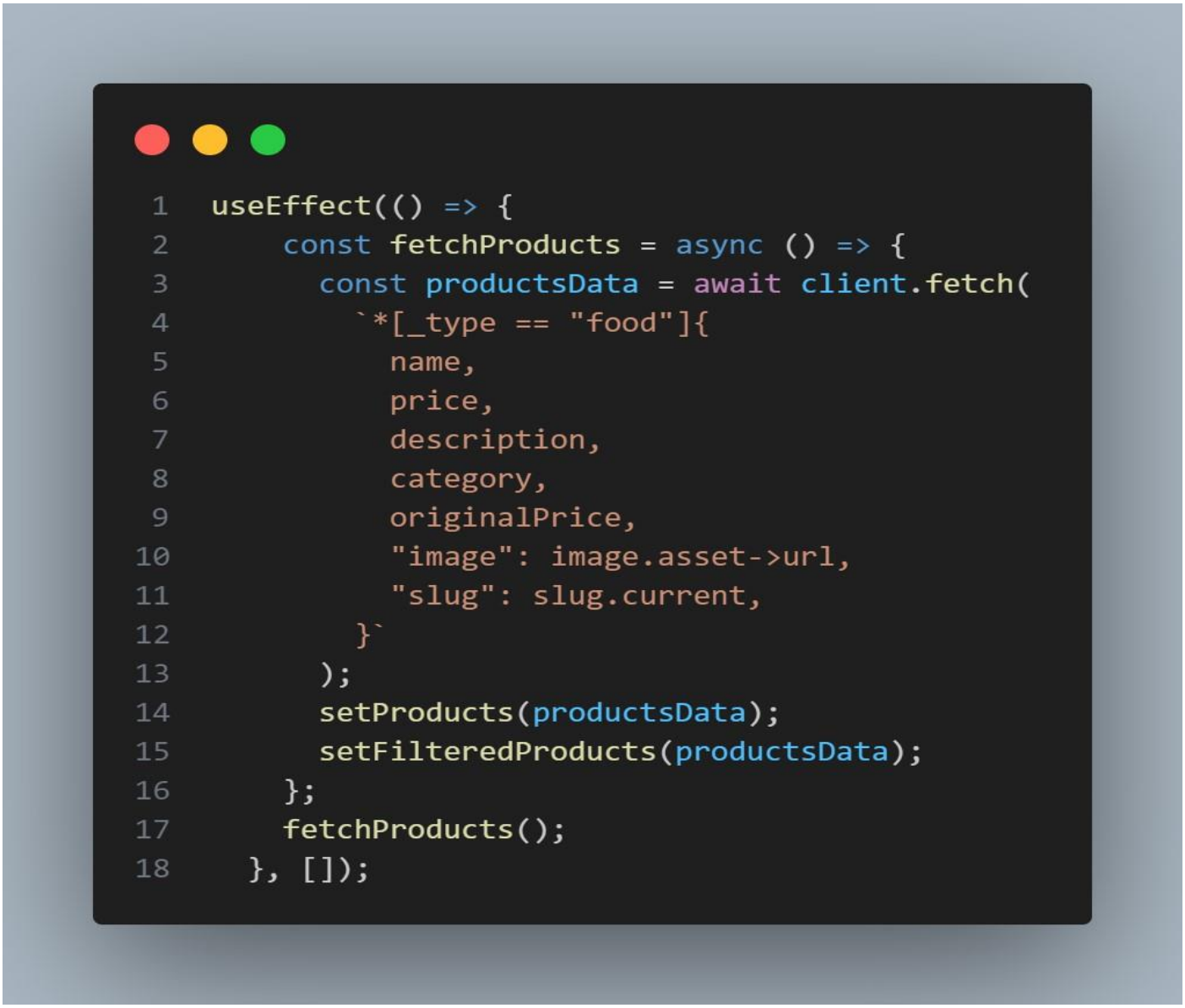
3. Ensure responsiveness across devices.
4. Implement modularity by breaking the component into smaller, reusable parts.

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.



```
useEffect(() => {
    const fetchProducts = async () => {
      const productsData = await client.fetch(
        `*[_type == "food"]{
          name,
          price,
          description,
          category,
          originalPrice,
          "image": image.asset->url,
          "slug": slug.current,
        }`
      );
      setProducts(productsData);
      setFilteredProducts(productsData);
    };
    fetchProducts();
  }, []);
```

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:

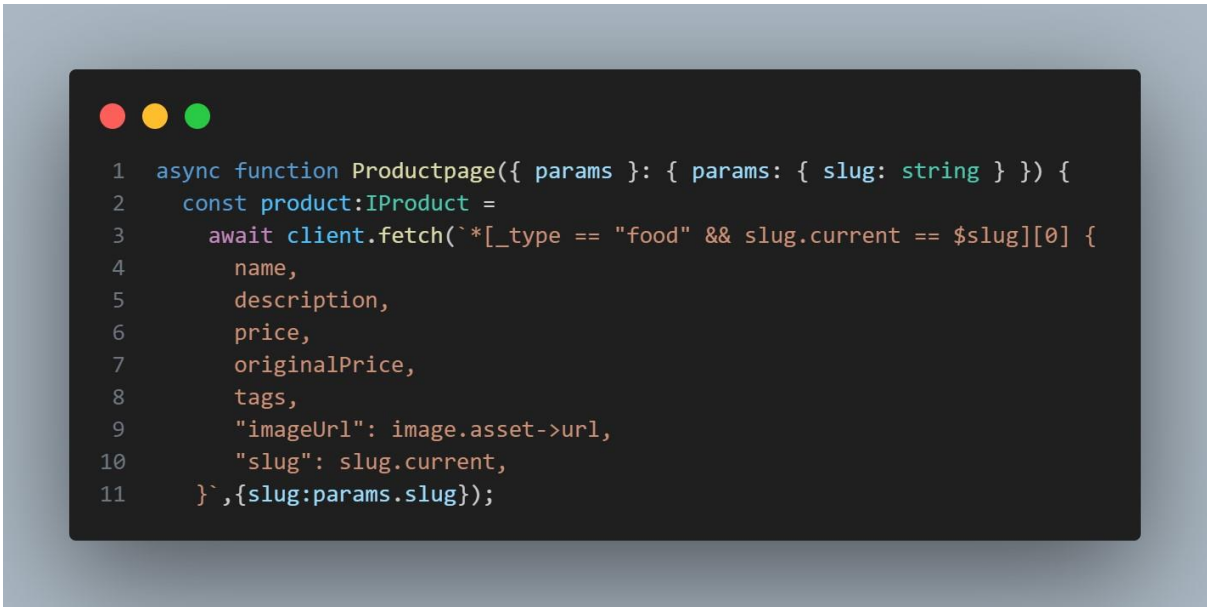
- **Name**
- **Product Description**
- **Price**
- **Category**
- **Stock Availability Implementation Plan:**

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.

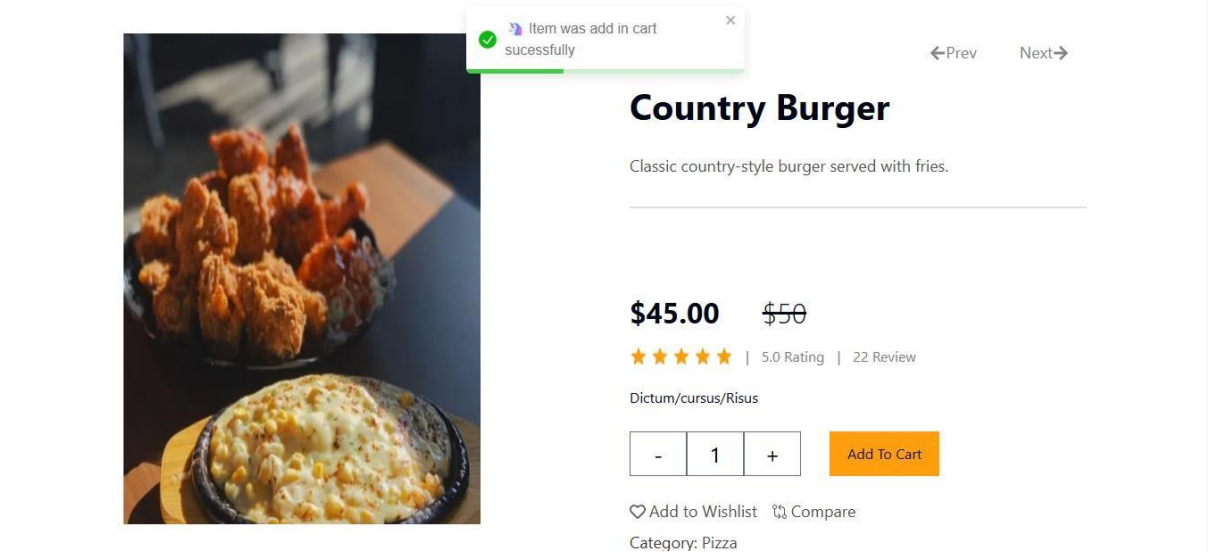
2. **Data Fields:**

   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.



```
async function Productpage({ params }: { params: { slug: string } }) {
  const product:IProduct =
    await client.fetch(`*[_type == "food" && slug.current == $slug][0] {
      name,
      description,
      price,
      originalPrice,
      tags,
      "imageUrl": image.asset->url,
      "slug": slug.current,
    }`,{slug:params.slug});
```

UI Display OF Product Detail Page:



Step 3: Search Bar with Price Filter

Objective:

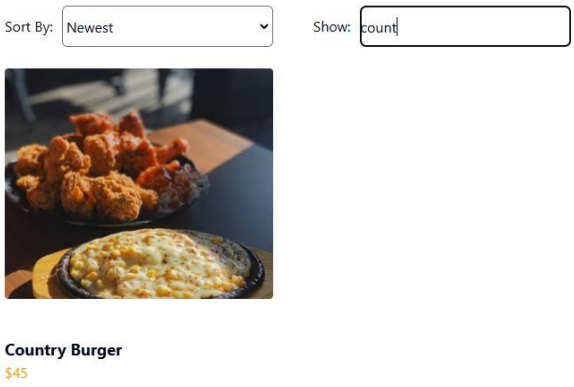
To implement a **search bar** and **price filters** to enhance the product browsing experience.

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.

```tsx
// Handle search
 const handleSearch = (event: React.ChangeEvent<HTMLInputElement>) => {
   const query = event.target.value.toLowerCase();
   setSearchQuery(query);

   const filtered = products.filter(
     (product) =>
       product.name.toLowerCase().includes(query) ||
       product.description.toLowerCase().includes(query) ||
       product.category.toLowerCase().includes(query) ||
       product.slug.toLowerCase().includes(query)
   );
   setFilteredProducts(filtered);
 };
```

UI Display :



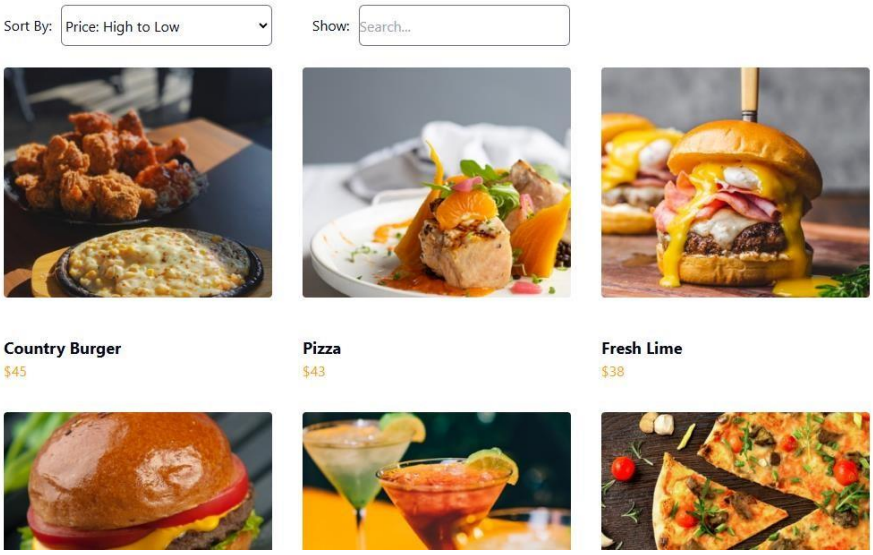
---

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

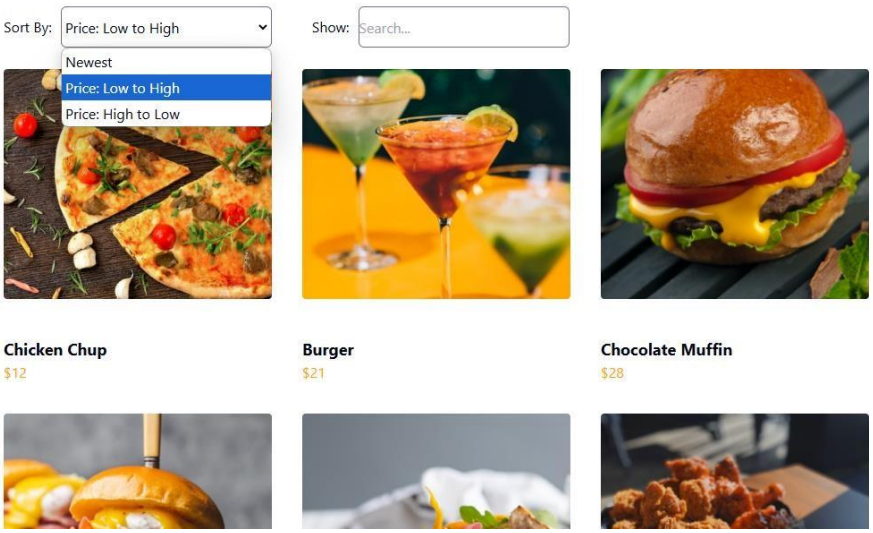
```
// Handle sorting
  const handleSort = (event: React.ChangeEvent<HTMLSelectElement>) => {
    const sortValue = event.target.value;
    setSortOrder(sortValue);

    let sortedProduct = [...products];
    if (sortValue === "lowToHigh") {
      sortedProduct.sort((a, b) => a.price - b.price);
    } else if (sortValue === "highToLow") {
      sortedProduct.sort((a, b) => b.price - a.price);
    }
    setFilteredProducts(sortedProduct);
  };
```

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:

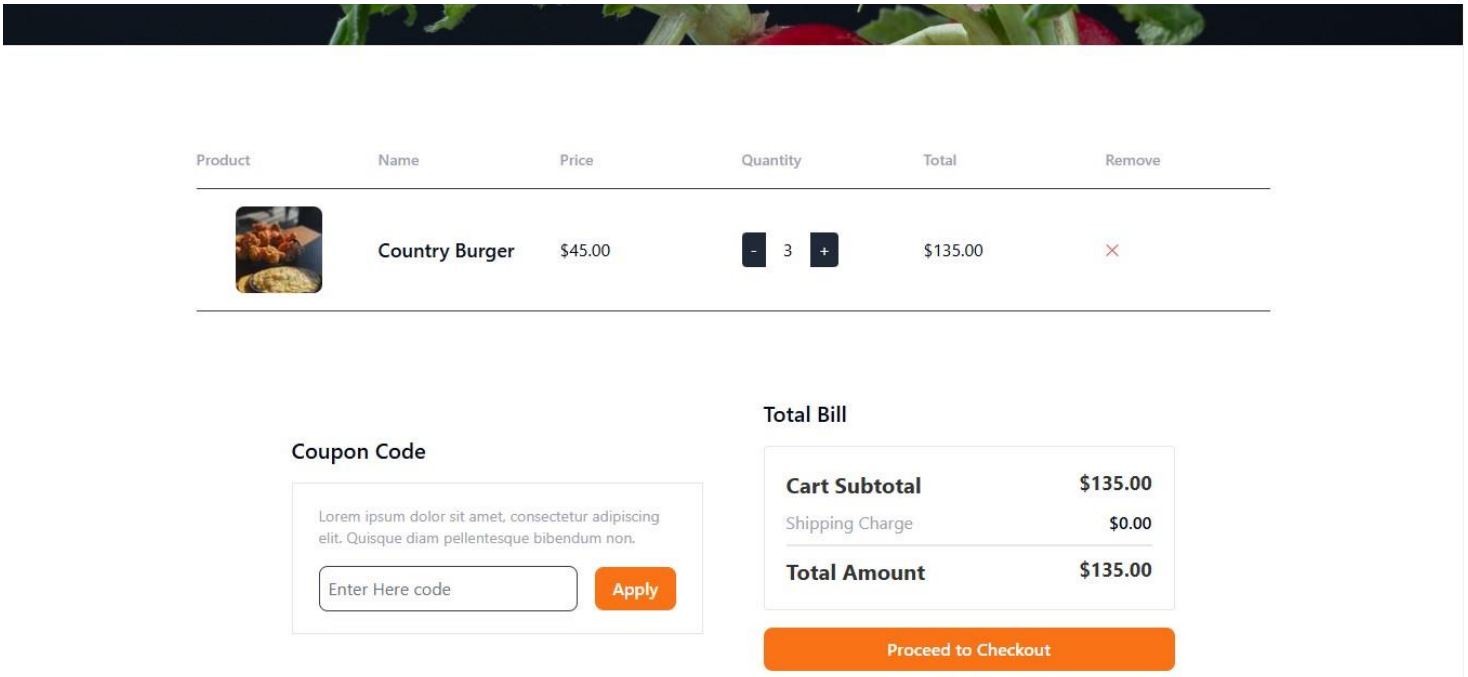
1. **State Management:**
   - Use **React state** or a state management library like Redux for storing cart data.
2. **Cart Data:**
   - Include details for each product in the cart:
     - Product Name
     - Price

- 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.

```
// Handle Increment
const handleIncrement = () => {
  const newQuantity = quantity + 1;
  setQuantity(newQuantity);
  setCartPrice(newQuantity * product.price); // Update price
};
 // Handle Decrement
 const handleDecrement = () => {
  if (quantity > 1) {
    const newQuantity = quantity - 1;
    setQuantity(newQuantity);
    setCartPrice(newQuantity * product.price);
  }
};

function handleAddToCart() {
  const cartItem = {
    slug: product.slugs,
    title: product.name,
    img: product.imageUrl,
    price: product.price,
    quantity: 1,
  };

  dispatch(addToCart(cartItem));
}
```

UI Display On Cart Page:



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).
4. **Cart Component:**
   - Displayed items added to the cart, quantity management, and total price calculation with dynamic updates.