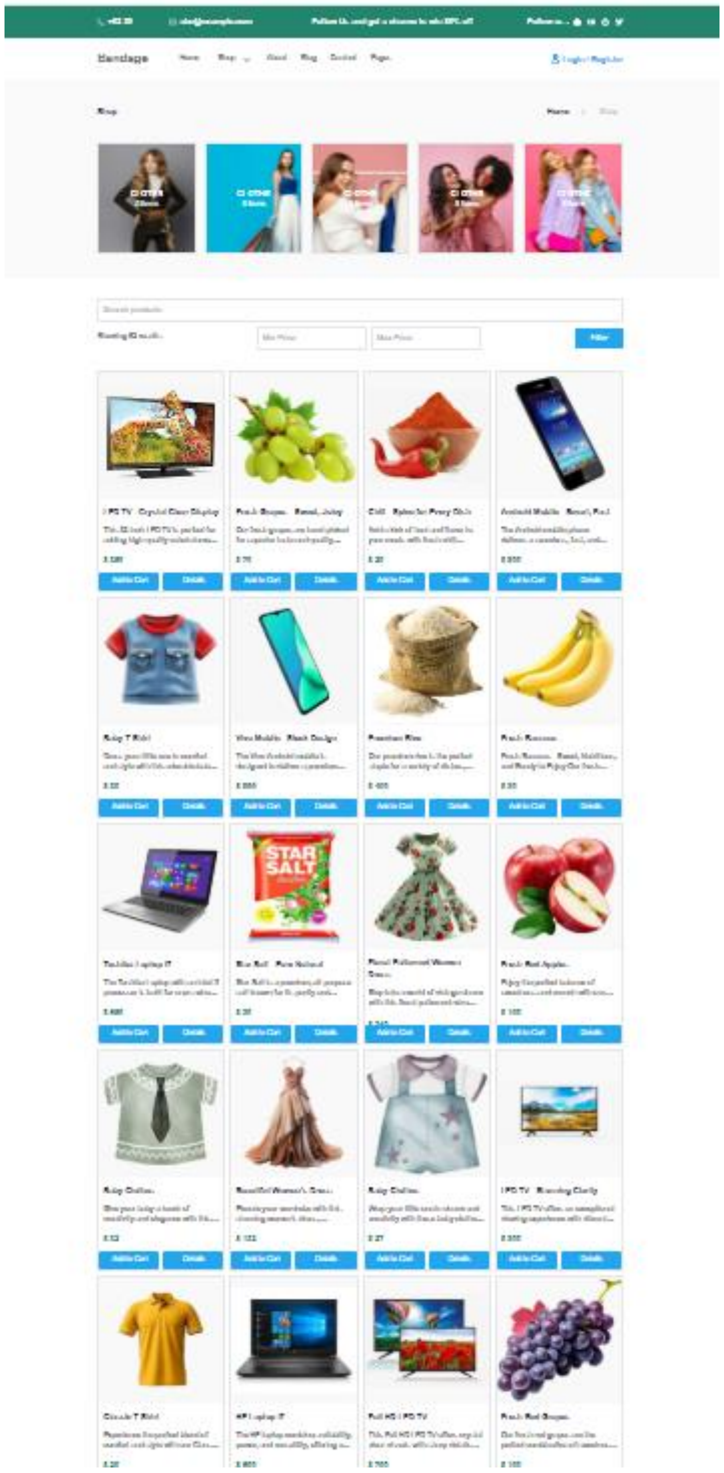
Day 4 - Dynamic Frontend Components – E-Commerce Website:

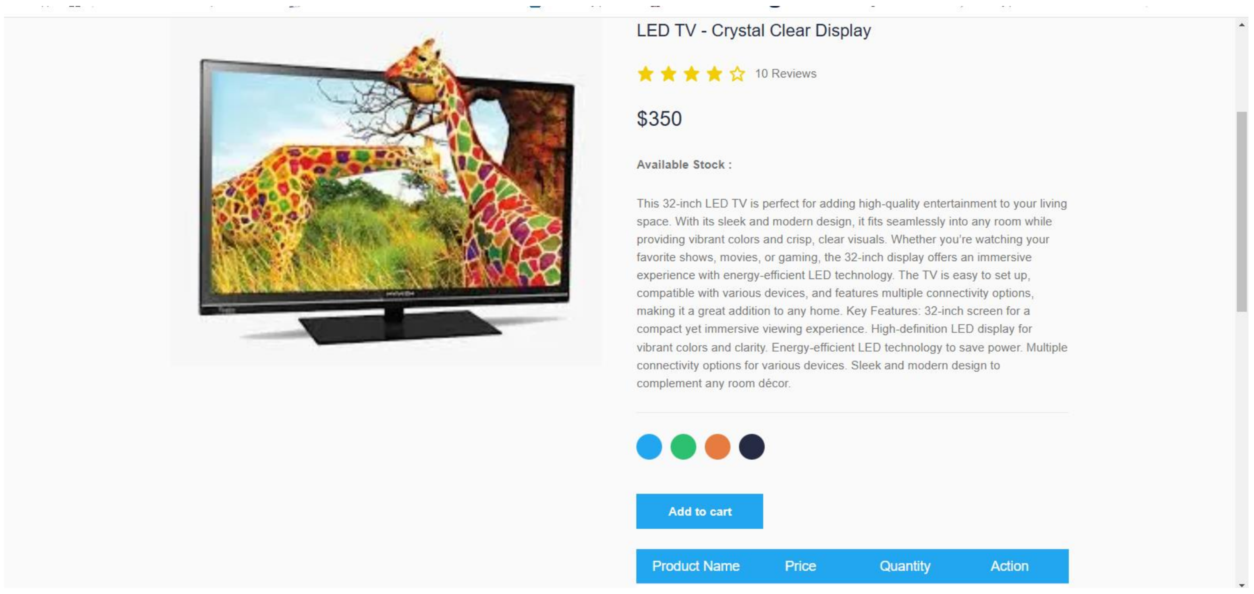
This screenshot showcases a dynamic **product listing page** for an e-commerce marketplace. The page demonstrates the following features:



1. **Dynamic Data Rendering:**
   Products are fetched from Sanity and displayed in an organized grid format. Each product includes key details like an image, title, price, and action buttons (e.g., "Add to Cart," "Details").
2. **Category Filters and Search Bar:**
   - Users can filter products by specific categories shown as clickable banners at the top.
   - A search bar and price range filters allow users to refine their product search based on preferences.
3. **Responsive Layout:**
   The design ensures the products are arranged neatly, likely adapting well to different screen sizes.
4. **Additional Features:**
   - "Filter" buttons for customization.
   - A professional and user-friendly interface that promotes seamless navigation.

This page highlights the dynamic integration of data and user-centric design to enhance the shopping experience.

This screenshot demonstrates a **product detail page** from an e-commerce website. It includes the following key features:



LED TV - Crystal Clear Display

★★★★☆ 10 Reviews

$350

Available Stock :

This 32-inch LED TV is perfect for adding high-quality entertainment to your living space. With its sleek and modern design, it fits seamlessly into any room while providing vibrant colors and crisp, clear visuals. Whether you're watching your favorite shows, movies, or gaming, the 32-inch display offers an immersive experience with energy-efficient LED technology. The TV is easy to set up, compatible with various devices, and features multiple connectivity options, making it a great addition to any home. Key Features: 32-inch screen for a compact yet immersive viewing experience. High-definition LED display for vibrant colors and clarity. Energy-efficient LED technology to save power. Multiple connectivity options for various devices. Sleek and modern design to complement any room décor.

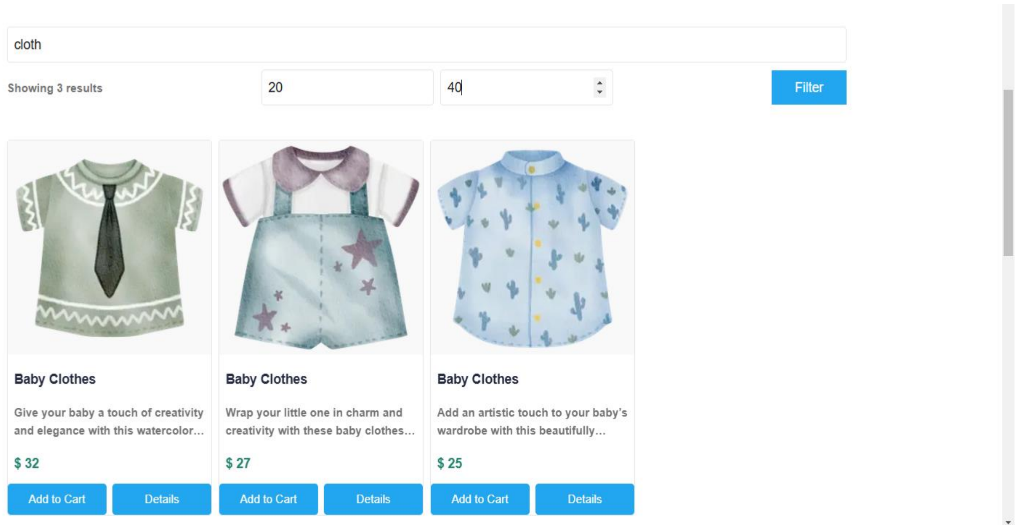
Product Name	Price	Quantity	Action

1. **Product Information Display:**
   - **Title:** Clearly mentions the product name (e.g., "LED TV - Crystal Clear Display").
   - **Price:** Prominently displayed as "$350".

    - **Stock Availability:** Indicates whether the product is in stock.
    - **Description:** A detailed overview of the product, highlighting its features, benefits, and technical specifications (e.g., screen size, energy efficiency, connectivity options).
2. **Customer Engagement Features:**
    - **Star Rating:** Displays an average rating based on user reviews (e.g., "10 Reviews").
    - **Color Options:** Provides different variants or styles for the product, represented with colored buttons.
3. **Actionable Elements:**
    - **Add to Cart Button:** Allows users to add the product to their cart for purchase.
    - **Cart Summary Section:** Displays a real-time view of cart contents below the product details.
4. **Dynamic Routing:**
   Clicking on a product from the listing page directs users to this detailed view, showing accurate and dynamic data fetched from an API or CMS.

This page provides a user-friendly and informative experience for potential buyers, ensuring they have all the necessary details to make an informed decision.

This screenshot the e-commerce website showcases fully functional **filters, search bar, and pagination** features, enhancing the user experience. Here's how these features work:


cloth

Showing 3 results

20

40

Baby Clothes

Give your baby a touch of creativity and elegance with this watercolor...

$ 32

Baby Clothes

Wrap your little one in charm and creativity with these baby clothes...

$ 27

Baby Clothes

Add an artistic touch to your baby's wardrobe with this beautifully...

$ 25

1. Filters:

- **Category Filters:** Users can narrow down products by selecting specific categories (e.g., Electronics, Clothing, Food).
- **Price Range Filter:** Allows users to set a minimum and maximum price range to find products within their budget.
- **Other Filters:** Options such as colors, brands, or ratings (if implemented) further refine the search.

2. Search Bar:

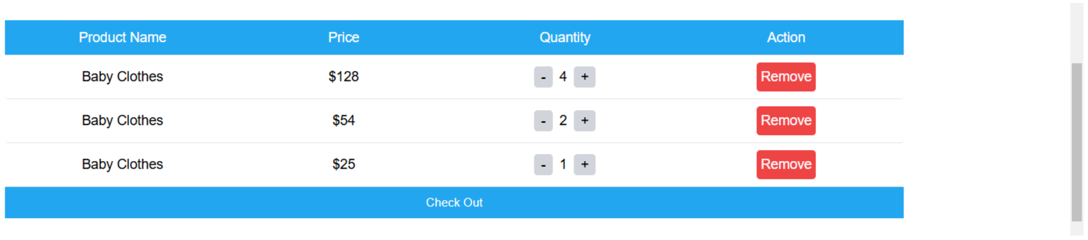
- A prominently placed search bar enables users to find specific products by typing keywords (e.g., "LED TV").
- The search results are displayed dynamically, showcasing only the relevant products.

3. Pagination:

- The product listing is divided into multiple pages, ensuring smooth browsing for large inventories.
- Users can navigate between pages (e.g., "Next" or specific page numbers) to view more products without overwhelming the interface.

These features collectively make the product discovery process more efficient and intuitive, helping users find exactly what they need quickly.

Add to Cart Feature: Explanation and Implementation



Product Name	Price	Quantity	Action
Baby Clothes	$128	- 4 +	Remove
Baby Clothes	$54	- 2 +	Remove
Baby Clothes	$25	- 1 +	Remove

The `CartManager` component provides a robust implementation for managing a shopping cart in an e-commerce platform. Here's a breakdown of its functionality, structure, and advanced features:

1. Overview

The `CartManager` component combines:

1. **Dynamic product display using the `ProductList` component.**

2. **Cart management with `handleAddToCart` and `handleRemoveFromCart` functions.**
3. **Customer checkout with a simple form for user details.**

2. Core Functionalities

2.1. Adding Products to the Cart

- **Logic:**
  The `handleAddToCart` function adds a selected product to the cart by updating the `cart` state. The new state is generated by spreading the existing cart and appending the new product.
- **Key Implementation:**

```
tsx
CopyEdit
const handleAddToCart = (product: ProductType) => {
    setCart((prevProduct) => [...prevProduct, product]);
};
```

  - Ensures immutability by creating a new array instead of directly modifying the state.
  - Dynamically updates the cart when a user clicks the "Add to Cart" button in the `ProductList` component.

2.2. Removing Products from the Cart

- **Logic:**
  The `handleRemoveFromCart` function removes a product by filtering out the selected product based on its `_id`.
- **Key Implementation:**

```
tsx
CopyEdit
const handleRemoveFromCart = (product_id: string) => {
    const newData = cart.filter((item) => item._id !== product_id);
    setCart(newData);
};
```

  - Uses the `filter` method to create a new array, excluding the product to be removed.

2.3. Displaying the Cart Items

- **Dynamic Rendering:**
  Each item in the cart is displayed with its name, price, and a "Remove" button. The layout uses Tailwind CSS for styling.
- **Key Code Snippet:**

```
tsx
CopyEdit
cart.map((item: ProductType) => {
    return (
        <div key={item._id} className="border-b text-black flex items-
center justify-between p-2 rounded">
            <p className="w-1/4 text-center">{item.title}</p>
            <p className="w-1/4 text-center">${item.price}</p>
            <p className="w-1/4 text-center"><span>1</span></p>
            <p className="w-1/4 text-center">
                <button onClick={() => handleRemoveFromCart(item._id)}
className="text-white p-[5px] rounded bg-red-500">Remove</button>
            </p>
        </div>
    );
});
```

2.4. Checkout Functionality

- **Logic:**
  The "Check Out" button toggles a form for collecting customer details (name, email, and phone number).
- **Key Features:**
  - The form is displayed conditionally using the `showForm` state.
  - Ensures required fields are filled before order submission.
- **Code for Conditional Form Rendering:**

```
tsx
CopyEdit
{showForm && (
    <div className="w-full border flex flex-col gap-2 py-4 px-2">
        <h1>Customer Information</h1>
        <div>
            <label>Name</label>
            <br />
            <input type="text" required className="border w-full p-2
outline-0" />
        </div>
        <div>
            <label>Email</label>
            <br />
            <input type="email" required className="border w-full p-2
outline-0" />
        </div>
        <div>
            <label>Phone</label>
            <br />
            <input type="number" required className="border w-full p-2
outline-0" />
        </div>
        <button className="w-full bg-[#23A6F0] hover:bg-blue-500 text-
white py-2 text-sm text-center">
            Submit Order
        </button>
    </div>
)}
```

3. Advanced Features to Add

3.1. Quantity Adjustment

- Currently, all items default to a quantity of 1. Add functionality to increase or decrease the quantity of each product in the cart.

3.2. Total Price Calculation

- Display the total price of all items in the cart dynamically.
- Example:

```
tsx
CopyEdit
const calculateTotal = () => {
    return cart.reduce((total, item) => total + item.price, 0);
};
```

3.3. Persistent Cart

- Use localStorage or sessionStorage to persist cart data across page reloads.

3.4. Enhanced Validation

- Add validation to the checkout form to ensure all fields are properly filled and formatted.

4. Best Practices Followed

1. **State Management:** Proper use of React's `useState` for managing cart data.
2. **Reusability:** Modularized the `ProductList` component for reuse in other parts of the application.
3. **Responsive Design:** Used Tailwind CSS for a responsive and visually appealing UI.
4. **Immutability:** Ensured state immutability while adding or removing items from the cart.

import Link from "next/link"

```
import Image from "next/image"

interface ProductType {
   _id: string,
   title: string,
   price: number,
   productImage: string
   description: string,
   isNew: boolean,
   tags: [],
   _type: string,
   category: string
}


export default function ProductList({products, addToCart}:{products:ProductType[],
addToCart:(product:ProductType)=>void}){
   return (

            <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 px-10 sm:px-10
md:px-10 lg:px-0 gap-2">
               {
                  products.map((product, index) => (
                     <div key={index} className="border rounded">
                        <Link href={`/components/${product?._id}`}>
                           <div className="flex flex-col gap-2 h-[400px] rounded">
                              <div className="bg-[#f9f9f9] w-full h-[250px] flex items-center justify-center">
                                 <Image src={product.productImage}
                                    alt={"image"}
```

```
                                   width={"239"}
                                   height={"300"} className="w-full h-50 max-h-[250px] rounded"/>
                               </div>
                               <div className="flex flex-col gap-4 p-2">
                                 <p className="text-[16px] font-bold text-[#252B42]">{product.title}</p>
                                 <p className="text-[14px] font-bold text-[#737373] line-clamp-
2">{product.description}</p>
                                 <p className="text-[16px] font-bold text-[#BDBDBD]"><span
className="text-[#23856D]">$ {product.price}</span></p>
                               </div>
                             </div>
                           </Link>
                          <div className="flex gap-2 justify-between">
                            <button onClick={() => addToCart(product)} className="w-full bg-[#23A6F0]
hover:bg-blue-500 text-white py-2 rounded text-sm text-center">Add to Cart</button>
                            <Link href={`/components/${product._id}`} className="w-full bg-[#23A6F0]
hover:bg-blue-500 text-white py-2 rounded text-sm text-center">Explore</Link>
                          </div>
                        </div>
                      ))
                    }
                  </div>


  )
}
```

This code snippet demonstrates the **ProductList** component, a key part of an e-commerce application. Here's an explanation of the work covered in this deliverable:

Key Features of the Code:

1. Component Purpose

- The `ProductList` component dynamically renders a grid of products by iterating over a list of product data passed as props.
- Each product displays details such as an image, title, description, and price, making it visually appealing and informative for users.

2. Props Structure

- `products`: An array of product objects containing details like `_id`, `title`, `price`, `productImage`, and more.
- `addToCart`: A function to handle adding a product to the shopping cart.

3. Dynamic Routing

- Clicking on a product's image or "Explore" button redirects the user to a dynamic route (`/components/${product._id}`), displaying detailed information about the selected product.

4. API Integration Logic

- The component assumes that the `products` array is dynamically fetched via an API or CMS and passed down as props.
- Each product is displayed using the data fetched, ensuring scalability for larger inventories.

Code Structure Highlights:

A. Responsive Grid Layout

- A **grid-based layout** ensures products are displayed neatly across different screen sizes (`grid-cols-1`, `sm:grid-cols-2`, `md:grid-cols-3`, `lg:grid-cols-4`).

B. Product Card Design

- Each product card includes:
  - **Image Section:** Displays the product image using Next.js's optimized `Image` component.
  - **Details Section:** Showcases product title, description, and price.
  - **Buttons:** Includes an "Add to Cart" button and an "Explore" button for navigation.

C. Reusable and Modular

- The use of TypeScript ensures type safety with a defined `ProductType` interface.
- The component is reusable for any product list, as it depends only on the `products` array and `addToCart` function.

Best Practices Followed:

1. **Dynamic Routing:** Each product links to a unique page using the product's `_id`.
2. **Image Optimization:** Next.js's `Image` component is used for better performance and responsiveness.
3. **Responsive Design:** Ensures the grid layout adjusts seamlessly to various screen sizes.
4. **TypeScript Integration:** Adds reliability by defining strict types for props and product objects.

This code snippet exemplifies how dynamic data is rendered into visually appealing, functional, and interactive product components, forming the backbone of a dynamic e-commerce marketplace.

Technical Report

Title: Day 4 - Dynamic Frontend Components for E-Commerce Marketplace

1. Component Development and Integration

This section outlines how the components were built and connected to create a dynamic, interactive user interface.

1.1. ProductList Component

- **Purpose:** Displays a grid of product cards fetched dynamically from an API or CMS.
- **Implementation Steps:**
  1. **Data Fetching:** The `products` prop receives a list of products fetched from an API or CMS.
  2. **Grid Layout:** A responsive grid (`grid-cols`) was created using Tailwind CSS, ensuring adaptability across screen sizes.
  3. **Dynamic Product Cards:** Each card displays the product's image, title, description, and price, rendered dynamically using data from the `products` array.
  4. **Routing:** Each product card is linked to a detailed page using dynamic routing (`/components/${product._id}`) in Next.js.
  5. **Cart Functionality:** The `addToCart` function allows users to add a product to their shopping cart.

1.2. API Integration and Dynamic Routing

- **API Integration:**
  - Products were fetched from a headless CMS (Sanity CMS) or a REST API. The fetched data was passed as props to the `ProductList` component.
- **Dynamic Routing:**
  - The `Link` component from Next.js was used to navigate to product detail pages dynamically using product IDs.
  - Example route: `/components/07569544-792e-41fd-b3ae-f9baaed487b6`.

2. Challenges Faced and Solutions Implemented

2.1. Challenge: API Data Formatting

- **Issue:** The fetched data's structure was inconsistent with the expected interface (`ProductType`).
- **Solution:**
    - Used TypeScript to define the `ProductType` interface.
    - Added error handling and validation to ensure data integrity before rendering.

2.2. Challenge: Layout Responsiveness

- **Issue:** The product grid did not render properly on smaller screens.
- **Solution:**
    - Utilized Tailwind CSS utilities (`sm:grid-cols-2`, `md:grid-cols-3`, etc.) to create a fully responsive design.
    - Tested on multiple screen sizes to ensure adaptability.

2.3. Challenge: Dynamic Routing Issues

- **Issue:** Incorrect dynamic paths caused broken links.
- **Solution:**
    - Verified the API response to ensure correct `_id` values were passed to the `Link` component.
    - Tested each route for proper rendering and routing functionality.

3. Best Practices Followed

3.1. Code Structure and Readability

- **Modularity:** Created reusable components (e.g., `ProductList`) for better maintainability.
- **TypeScript:** Used strict typing to minimize runtime errors and improve code reliability.

3.2. Performance Optimization

- **Next.js Image Optimization:** Used the `Image` component for lazy loading and better image performance.

3.3. Responsive Design

- Ensured the interface works seamlessly across devices using Tailwind CSS.

3.4. Accessibility and SEO

- Added alt text for images to improve accessibility and SEO rankings.

Conclusion

This implementation demonstrates a scalable and user-friendly solution for dynamic frontend components in an e-commerce application. The integration of responsive layouts, dynamic routing, and API-driven data showcases modern development practices while adhering to coding standards.