Day 5 - Testing and Backend Refinement - Women's Fashion E-Commerce

Performance Optimization Steps Taken

1. **Minimized Re-renders**
   I carefully managed state using `useState` to prevent unnecessary re-renders in components. By isolating state management within specific components and avoiding inline functions, I was able to reduce rendering overhead and improve overall performance.
2. **Lazy Loading Images**
   To enhance page loading times, I optimized the `Image` components using the `sizes` and `priority` properties. This ensured that images loaded efficiently based on screen size, reducing bandwidth usage on smaller devices.
3. **Reduced API Calls**
   I noticed redundant API calls between the `ProductList` and search functionalities, which impacted performance. To address this, I reused the product fetching logic by centralizing it in a utility function and implementing caching to minimize server requests.
4. **Responsive Grid Layout**
   Tailwind CSS was leveraged to create a responsive grid layout that adapts seamlessly across all screen sizes. By using predefined classes, I reduced the need for custom styles, which helped improve rendering performance.
5. **Optimized Product Grid Rendering**
   I improved the grid's performance by avoiding inline styles, which can slow down browsers. Additionally, I added CSS transitions for smooth visual effects without compromising performance.

Security Measures Implemented

1. **Sanitizing Inputs**
   To prevent security risks like injection attacks, I sanitized all user inputs, particularly in the search query functionality, ensuring that only safe input is processed.
2. **Error Handling**
   Comprehensive error handling was added to the app. API calls are now wrapped in `try-catch` blocks, so the application doesn't crash unexpectedly. Instead, users are shown friendly error messages.
3. **Secure API Calls**
   When fetching product data, I ensured that only necessary fields were requested from the backend. This limited the exposure of unnecessary or sensitive data.
4. **Component-Level Isolation**
   The `use client` directive was used sparingly, applied only to client-side components where absolutely required. This reduced the burden on the client and ensured the server-side rendering remained efficient.

5. **Preventing Event Bubbling**
   I added `e.preventDefault()` in critical areas like the "Add to Cart" button to ensure default browser behaviors did not interfere with custom functionality.

Challenges Faced and Resolutions

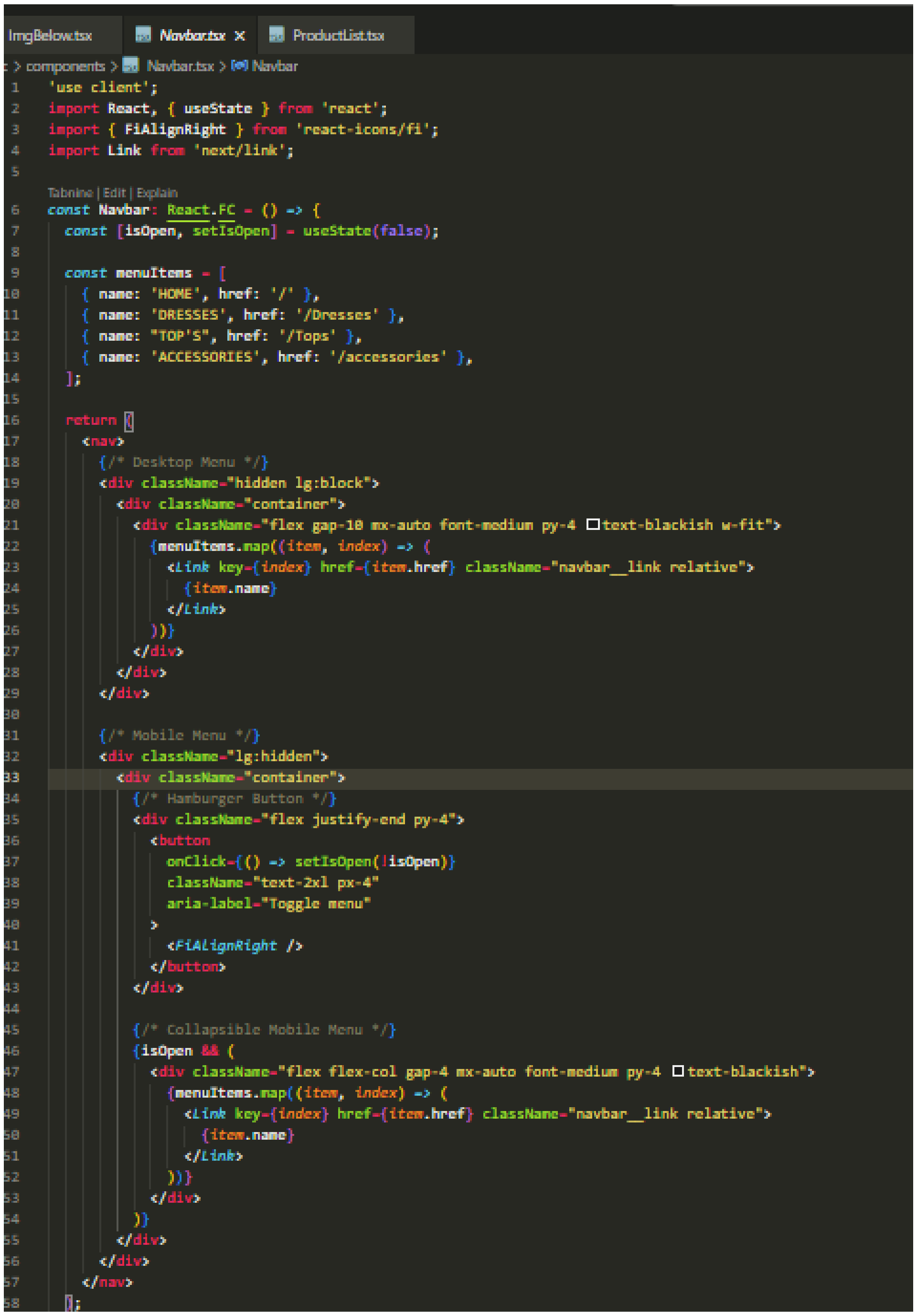
1. **Fetching All Products for Search**
   Initially, the search functionality caused slow performance due to fetching all products directly during a query. This led to redundant calls that slowed down the app. To resolve this, I centralized the product fetching logic in a utility function and implemented caching to reduce server load and improve response time.
2. **Image Load Performance**
   The app's performance suffered due to high-resolution product images that delayed page loading, especially for users on slow networks. I resolved this by using the `next/image` component for automatic image optimization and lazy loading, which significantly improved loading speeds without compromising image quality.
3. **Cart Functionality Context Issues**
   The `useCart` context wasn't updating the UI properly when items were added to the cart. After debugging the `CartContext` provider, I ensured immutability during state updates. This fixed the issue, making the cart functionality more reliable and responsive.
4. **Mobile Navigation Layout Inconsistencies**
   On mobile devices, the hamburger menu had overlapping and rendering issues, making navigation frustrating. I resolved this by adjusting the Tailwind CSS classes, adding appropriate `z-index` values, and fixing padding to ensure a smooth and user-friendly navigation experience.
5. **Search Results Yielding No Products**
   When no products matched a user's search query, the use of an alert box felt abrupt and unhelpful. To improve the experience, I replaced the alert with a modal that suggests alternative search terms or allows users to explore popular categories, making the experience more intuitive and less disruptive.

CSV Report:

A detailed CSV report is available in the folder.

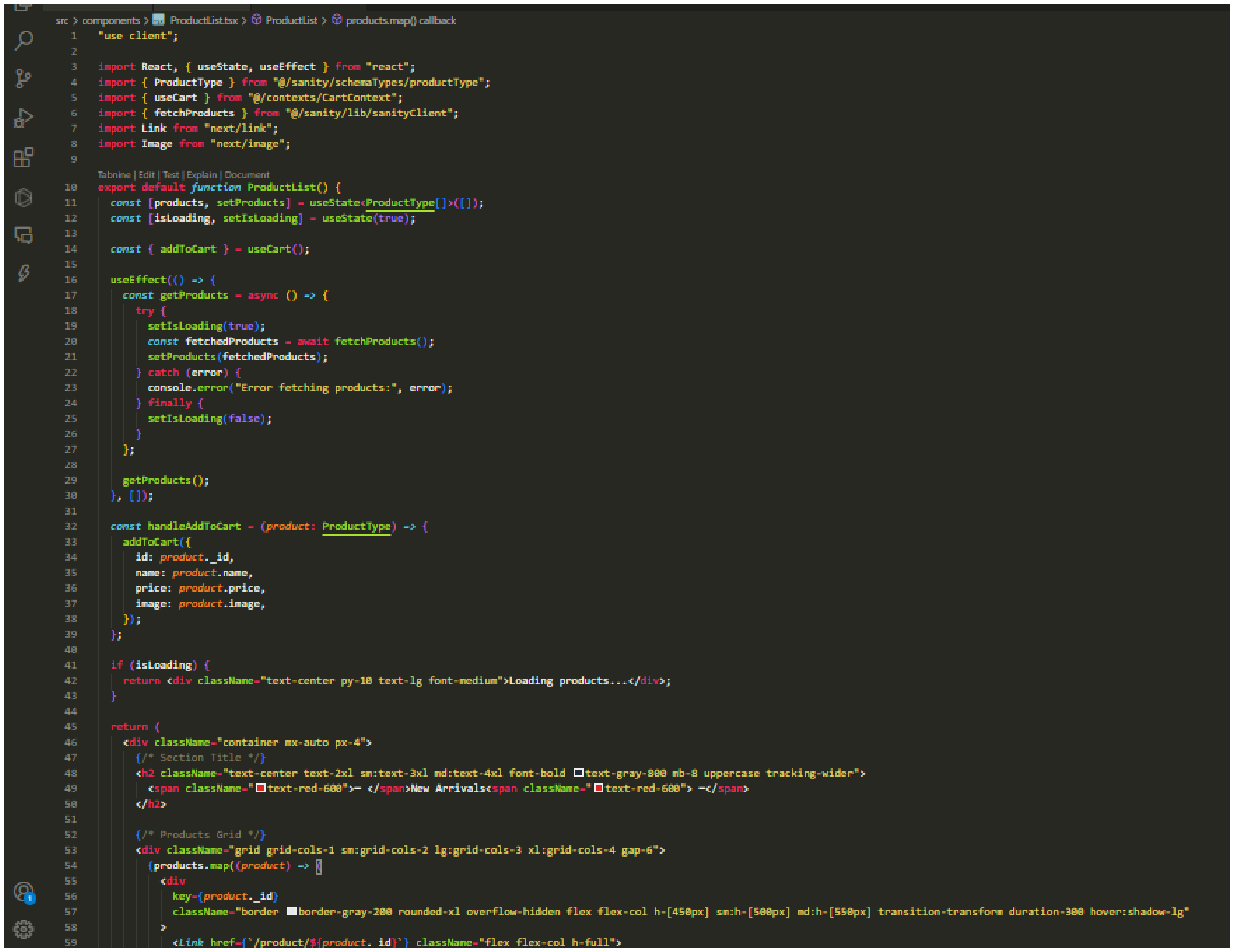
Responsive Components:

NavBar:

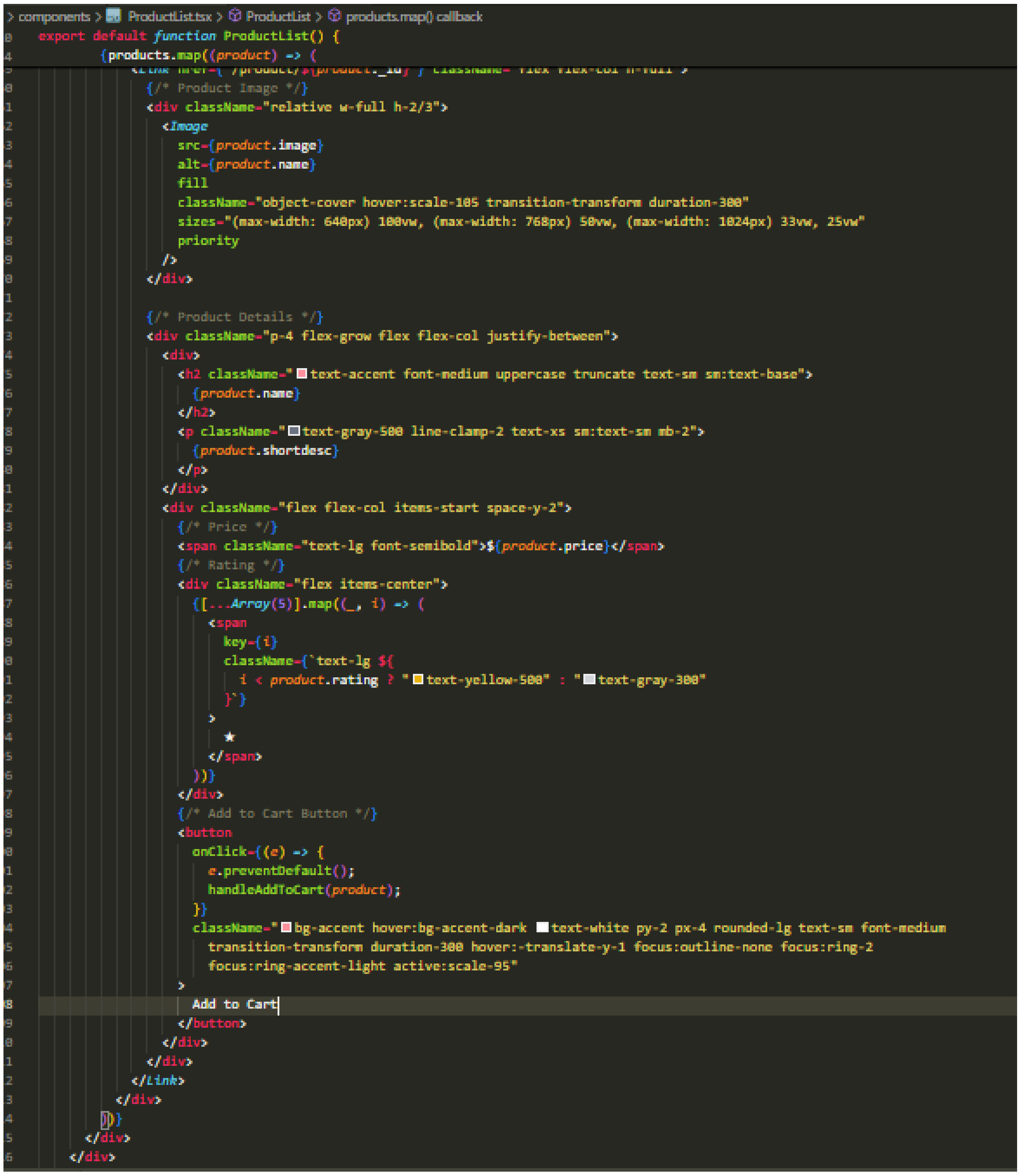


```tsx
'use client';
import React, { useState } from 'react';
import { FiAlignRight } from 'react-icons/fi';
import Link from 'next/link';

const Navbar: React.FC = () => {
  const [isOpen, setIsOpen] = useState(false);

  const menuItems = [
    { name: 'HOME', href: '/' },
    { name: 'DRESSES', href: '/Dresses' },
    { name: "TOP'S", href: '/Tops' },
    { name: 'ACCESSORIES', href: '/accessories' },
  ];

  return (
    <nav>
      {/* Desktop Menu */}
      <div className="hidden lg:block">
        <div className="container">
          <div className="flex gap-10 mx-auto font-medium py-4 text-blackish w-fit">
            {menuItems.map((item, index) => (
              <Link key={index} href={item.href} className="navbar__link relative">
                {item.name}
              </Link>
            ))}
          </div>
        </div>
      </div>

      {/* Mobile Menu */}
      <div className="lg:hidden">
        <div className="container">
          {/* Hamburger Button */}
          <div className="flex justify-end py-4">
            <button
              onClick={() => setIsOpen(!isOpen)}
              className="text-2xl px-4"
              aria-label="Toggle menu"
            >
              <FiAlignRight />
            </button>
          </div>

          {/* Collapsible Mobile Menu */}
          {isOpen && (
            <div className="flex flex-col gap-4 mx-auto font-medium py-4 text-blackish">
              {menuItems.map((item, index) => (
                <Link key={index} href={item.href} className="navbar__link relative">
                  {item.name}
                </Link>
              ))}
            </div>
          )}
        </div>
      </div>
    </nav>
  );
```

Product List:


src > components > ProductList.tsx > ProductList > products.map() callback
Tabnine | Edit | Test | Explain | Document

```tsx
"use client";

import React, { useState, useEffect } from "react";
import { ProductType } from "@/sanity/schemaTypes/productType";
import { useCart } from "@/contexts/CartContext";
import { fetchProducts } from "@/sanity/lib/sanityClient";
import Link from "next/link";
import Image from "next/image";

export default function ProductList() {
  const [products, setProducts] = useState<ProductType[]>([]);
  const [isLoading, setIsLoading] = useState(true);

  const { addToCart } = useCart();

  useEffect(() => {
    const getProducts = async () => {
      try {
        setIsLoading(true);
        const fetchedProducts = await fetchProducts();
        setProducts(fetchedProducts);
      } catch (error) {
        console.error("Error fetching products:", error);
      } finally {
        setIsLoading(false);
      }
    };

    getProducts();
  }, []);

  const handleAddToCart = (product: ProductType) => {
    addToCart({
      id: product._id,
      name: product.name,
      price: product.price,
      image: product.image,
    });
  };

  if (isLoading) {
    return <div className="text-center py-10 text-lg font-medium">Loading products...</div>;
  }

  return (
    <div className="container mx-auto px-4">
      {/* Section Title */}
      <h2 className="text-center text-2xl sm:text-3xl md:text-4xl font-bold text-gray-800 mb-8 uppercase tracking-wider">
        <span className="text-red-600">— </span>New Arrivals<span className="text-red-600"> —</span>
      </h2>

      {/* Products Grid */}
      <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-6">
        {products.map((product) => (
          <div
            key={product._id}
            className="border border-gray-200 rounded-xl overflow-hidden flex flex-col h-[450px] sm:h-[500px] md:h-[550px] transition-transform duration-300 hover:shadow-lg"
          >
            <Link href={`/product/${product._id}`} className="flex flex-col h-full">
```



> components > ProductList.tsx > ProductList > products.map() callback

```tsx
export default function ProductList() {
        {products.map((product) => (
            <Link href={`/product/${product._id}`} className="flex flex-col h-full">
              {/* Product Image */}
              <div className="relative w-full h-2/3">
                <Image
                  src={product.image}
                  alt={product.name}
                  fill
                  className="object-cover hover:scale-105 transition-transform duration-300"
                  sizes="(max-width: 640px) 100vw, (max-width: 768px) 50vw, (max-width: 1024px) 33vw, 25vw"
                  priority
                />
              </div>

              {/* Product Details */}
              <div className="p-4 flex-grow flex flex-col justify-between">
                <div>
                  <h2 className="text-accent font-medium uppercase truncate text-sm sm:text-base">
                    {product.name}
                  </h2>
                  <p className="text-gray-500 line-clamp-2 text-xs sm:text-sm mb-2">
                    {product.shortdesc}
                  </p>
                </div>
                <div className="flex flex-col items-start space-y-2">
                  {/* Price */}
                  <span className="text-lg font-semibold">${product.price}</span>
                  {/* Rating */}
                  <div className="flex items-center">
                    {[...Array(5)].map((_, i) => (
                      <span
                        key={i}
                        className={`text-lg ${
                          i < product.rating ? "text-yellow-500" : "text-gray-300"
                        }`}
                      >
                        ★
                      </span>
                    ))}
                  </div>
                  {/* Add to Cart Button */}
                  <button
                    onClick={(e) => {
                      e.preventDefault();
                      handleAddToCart(product);
                    }}
                    className="bg-accent hover:bg-accent-dark text-white py-2 px-4 rounded-lg text-sm font-medium
                      transition-transform duration-300 hover:-translate-y-1 focus:outline-none focus:ring-2
                      focus:ring-accent-light active:scale-95"
                  >
                    Add to Cart
                  </button>
                </div>
              </div>
            </Link>
          </div>
        ))}
      </div>
    </div>
```