Day 4 - Dynamic Frontend Components - My Marketplace

Functional Deliverables

Project Overview

This project is an e-commerce marketplace built using Next.js for the frontend and Sanity CMS for backend content management. The goal is to provide a dynamic, user-friendly platform for showcasing products, managing orders, and enabling users to interact with the marketplace seamlessly.

Demonstration Video

You can watch a detailed walkthrough of the project in action by clicking the link below:
Project Demonstration Video

GIAIC Hackathon 3 | Day 4 | Tampelet 2

1. Product Listing Page with Dynamic Data

The product listing page was developed to display dynamic product data fetched from Sanity CMS. Key features include:

- **Grid Layout**: Utilized Tailwind CSS to create a responsive and visually appealing product grid.
- **Dynamic Data Rendering**: Products are fetched via Sanity's GROQ queries and rendered dynamically on the page.

2. Product Detail Pages

Implemented individual product detail pages with dynamic routing. Key functionalities:

- **Dynamic Routes**: Next.js's file-based routing system was used to generate routes for each product based on its slug.
- **Detailed Information**: Fetched and displayed detailed product descriptions, features, images, and pricing from Sanity CMS.

3. Category Filtering and Search Bar

Added functionality to filter products by category and search dynamically:

- **Category Filters**: Users can filter products by selecting categories, which update the product grid in real-time.
- **Search Bar**: Integrated a search bar that allows users to search for products by name or tags.

4. Cart Components

Developed comprehensive cart management features, including:

- **Shipment & Billing Forms**: Multi-step forms for users to input shipping and billing details.
- **Order Success Page**: Displayed after successful order placement with details and a summary.

5. Order Management Page

A dedicated page where users can:

- View their order history with order details.
- Download invoices as PDF using the jsPDF library.

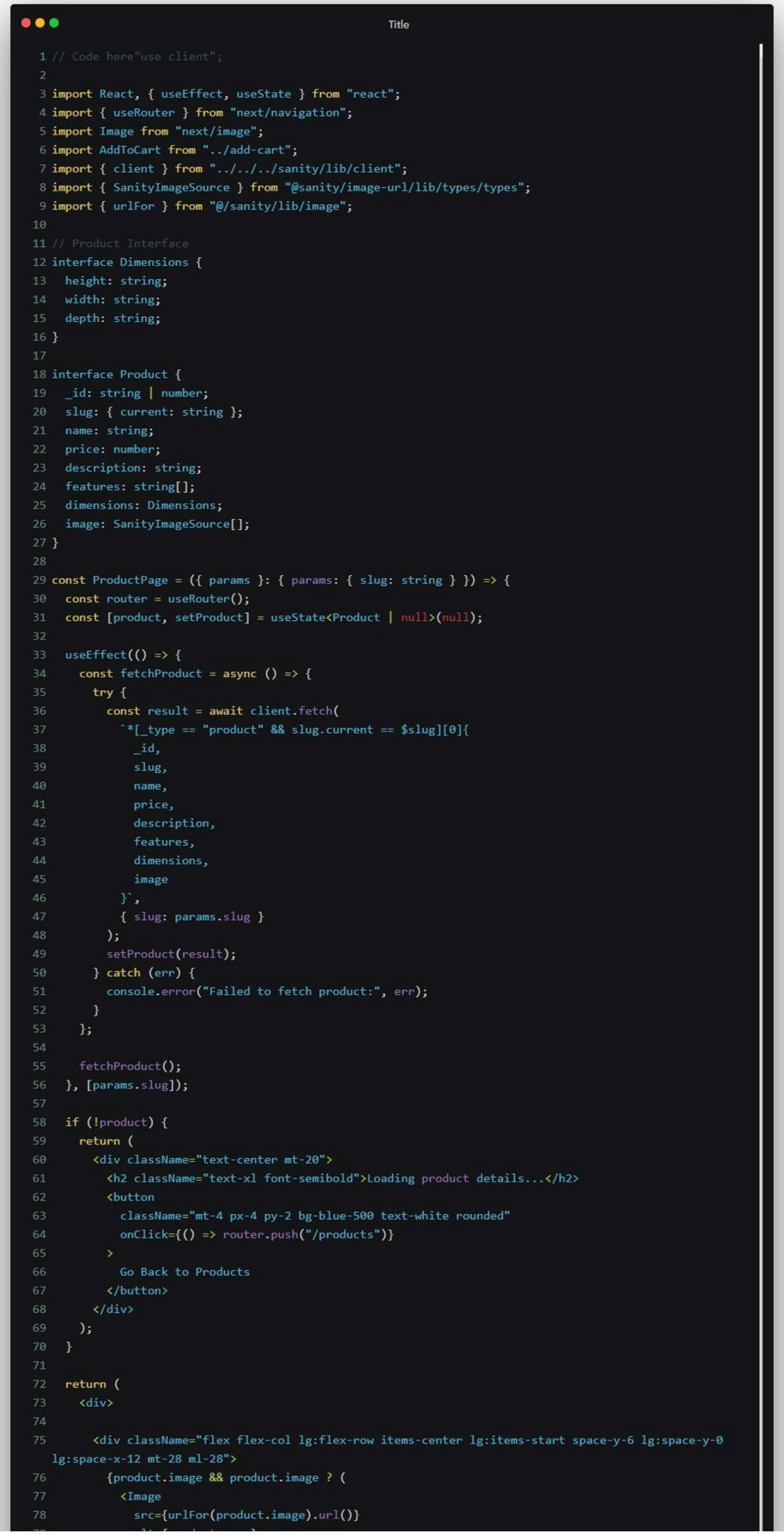
- Track order statuses in real-time.

Code Deliverables | Dynamic Routing

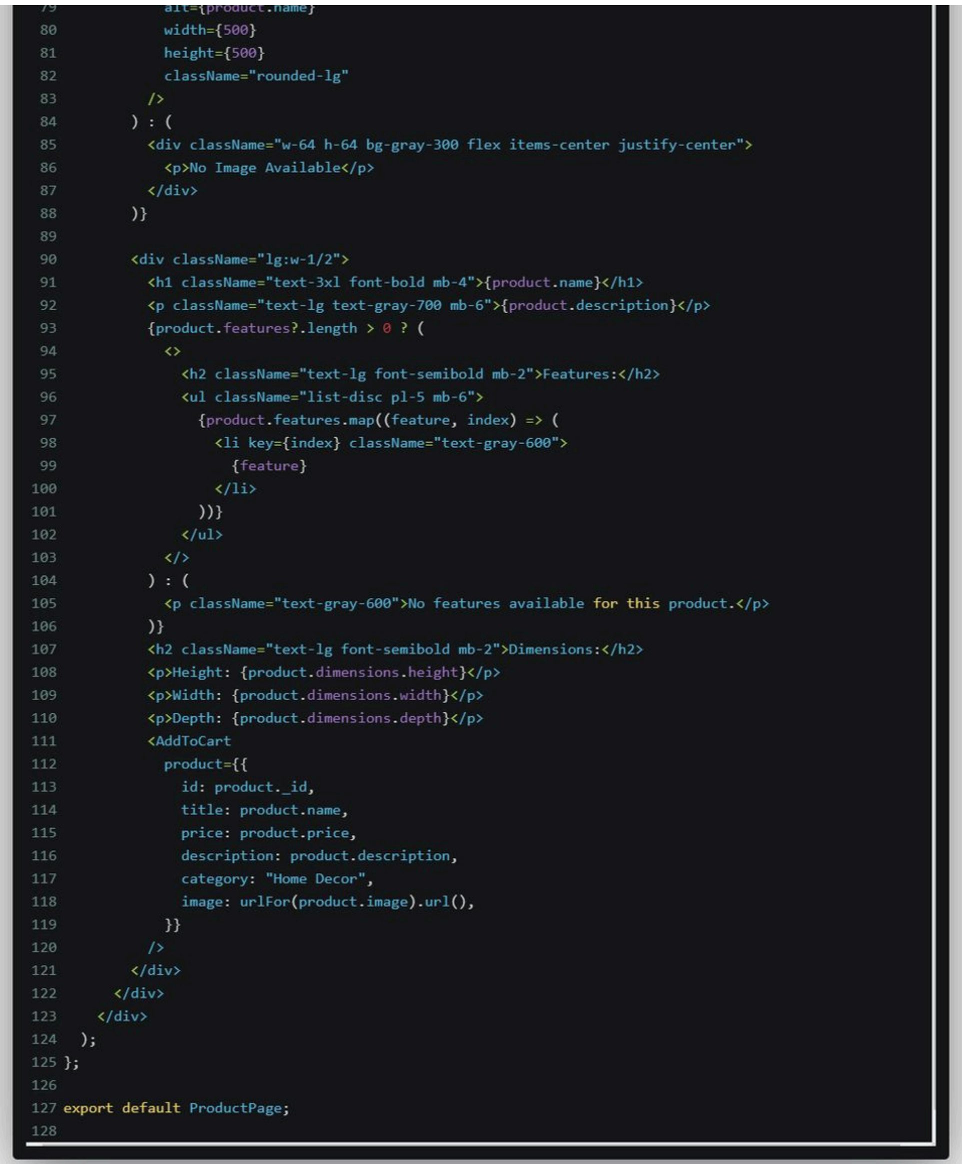
Key Components

ProductCard Component

Handles the rendering of individual product cards in the listing:

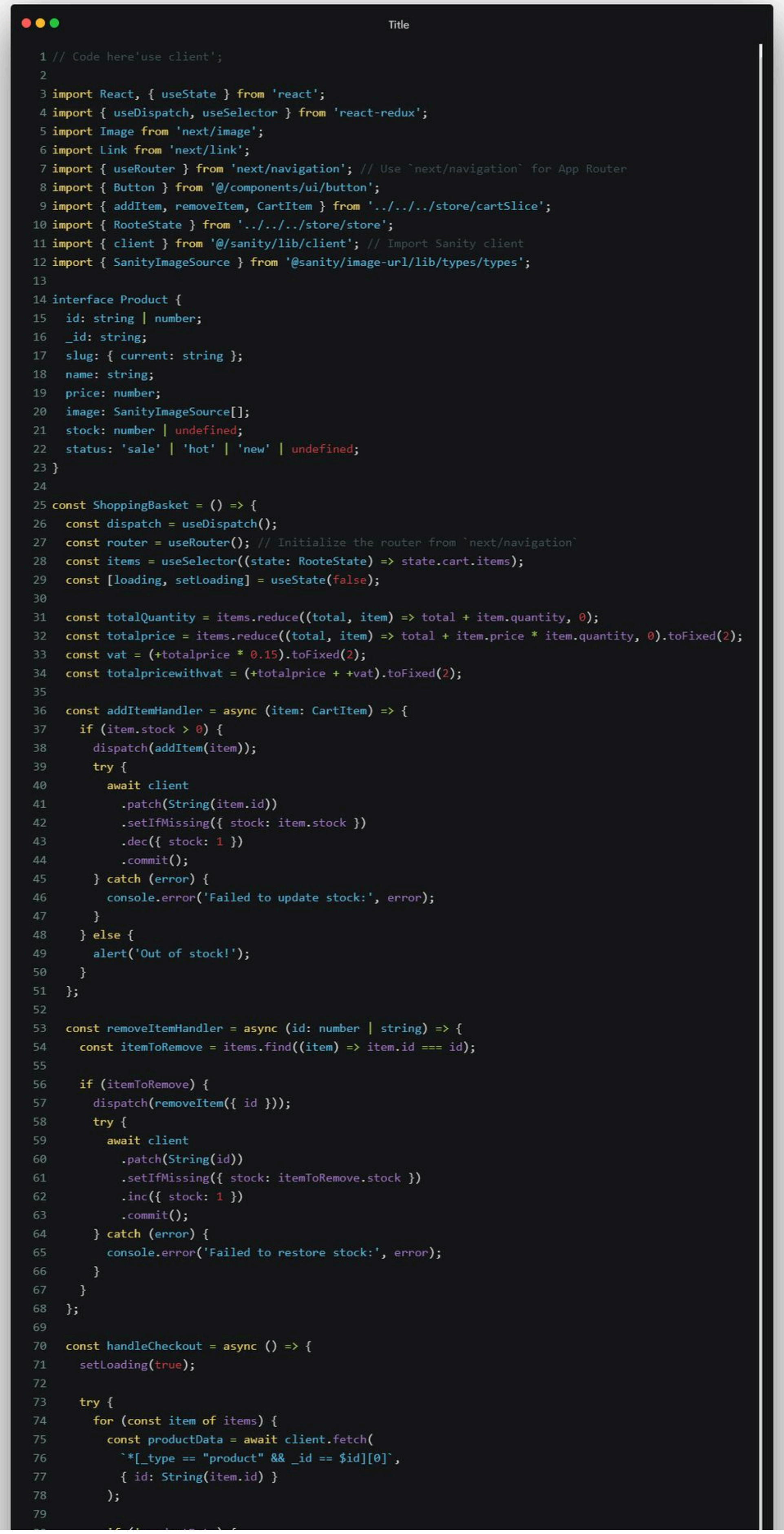

Title

```tsx
// Code here"use client";

import React, { useEffect, useState } from "react";
import { useRouter } from "next/navigation";
import Image from "next/image";
import AddToCart from "../add-cart";
import { client } from "../../../sanity/lib/client";
import { SanityImageSource } from "@sanity/image-url/lib/types/types";
import { urlFor } from "@/sanity/lib/image";

// Product Interface
interface Dimensions {
  height: string;
  width: string;
  depth: string;
}

interface Product {
  _id: string | number;
  slug: { current: string };
  name: string;
  price: number;
  description: string;
  features: string[];
  dimensions: Dimensions;
  image: SanityImageSource[];
}

const ProductPage = ({ params }: { params: { slug: string } }) => {
  const router = useRouter();
  const [product, setProduct] = useState<Product | null>(null);

  useEffect(() => {
    const fetchProduct = async () => {
      try {
        const result = await client.fetch(
          `*[_type == "product" && slug.current == $slug][0]{
            _id,
            slug,
            name,
            price,
            description,
            features,
            dimensions,
            image
          }`,
          { slug: params.slug }
        );
        setProduct(result);
      } catch (err) {
        console.error("Failed to fetch product:", err);
      }
    };

    fetchProduct();
  }, [params.slug]);

  if (!product) {
    return (
      <div className="text-center mt-20">
        <h2 className="text-xl font-semibold">Loading product details...</h2>
        <button
          className="mt-4 px-4 py-2 bg-blue-500 text-white rounded"
          onClick={() => router.push("/products")}
        >
          Go Back to Products
        </button>
      </div>
    );
  }

  return (
    <div>

      <div className="flex flex-col lg:flex-row items-center lg:items-start space-y-6 lg:space-y-0 lg:space-x-12 mt-28 ml-28">
        {product.image && product.image ? (
          <Image
            src={urlFor(product.image).url()}
```

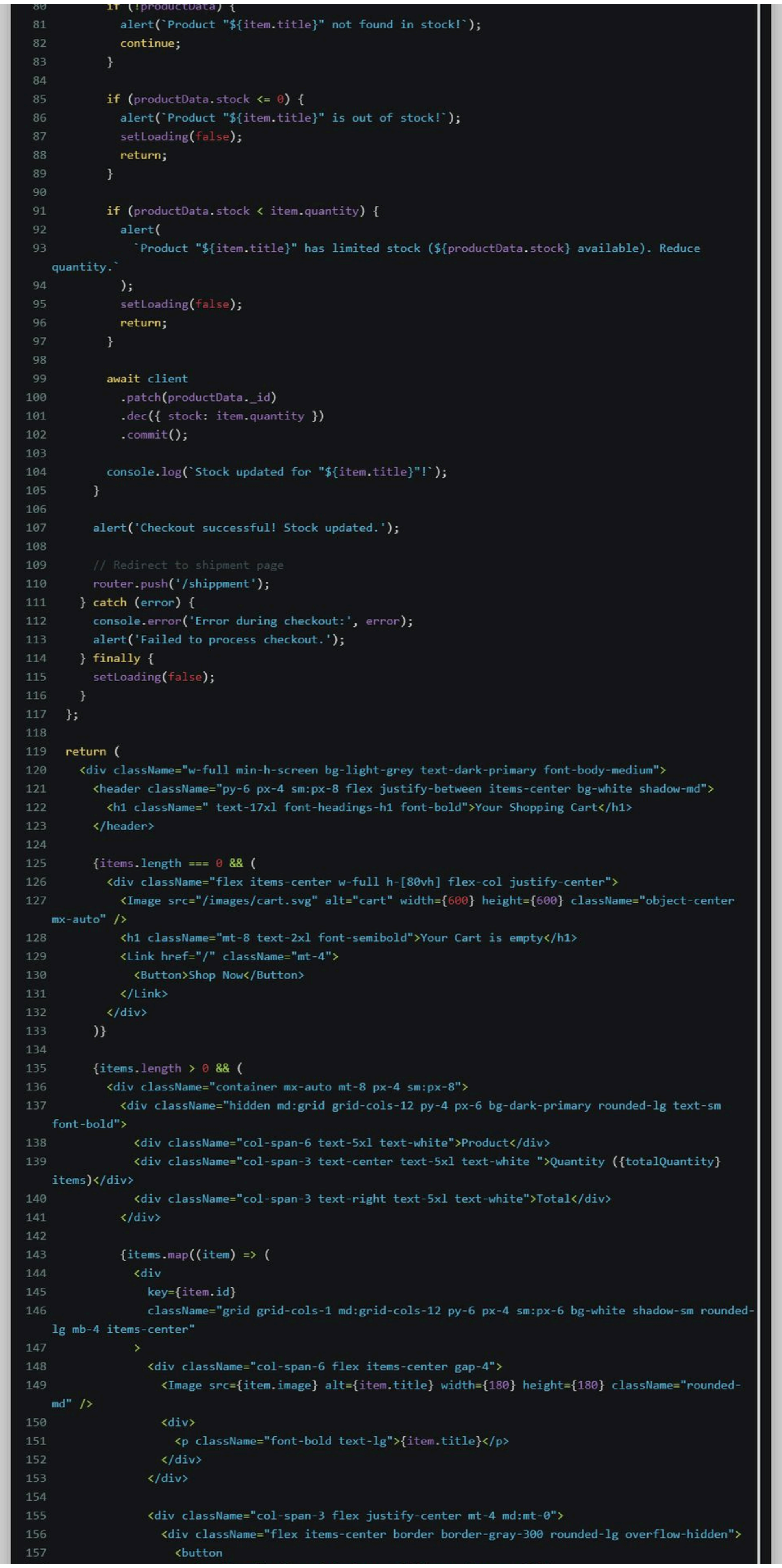


```
            alt={product.name}
            width={500}
            height={500}
            className="rounded-lg"
          />
        ) : (
          <div className="w-64 h-64 bg-gray-300 flex items-center justify-center">
            <p>No Image Available</p>
          </div>
        )}

        <div className="lg:w-1/2">
          <h1 className="text-3xl font-bold mb-4">{product.name}</h1>
          <p className="text-lg text-gray-700 mb-6">{product.description}</p>
          {product.features?.length > 0 ? (
            <>
              <h2 className="text-lg font-semibold mb-2">Features:</h2>
              <ul className="list-disc pl-5 mb-6">
                {product.features.map((feature, index) => (
                  <li key={index} className="text-gray-600">
                    {feature}
                  </li>
                ))}
              </ul>
            </>
          ) : (
            <p className="text-gray-600">No features available for this product.</p>
          )}
          <h2 className="text-lg font-semibold mb-2">Dimensions:</h2>
          <p>Height: {product.dimensions.height}</p>
          <p>Width: {product.dimensions.width}</p>
          <p>Depth: {product.dimensions.depth}</p>
          <AddToCart
            product={{
              id: product._id,
              title: product.name,
              price: product.price,
              description: product.description,
              category: "Home Decor",
              image: urlFor(product.image).url(),
            }}
          />
        </div>
      </div>
    </div>
  );
};

export default ProductPage;

```

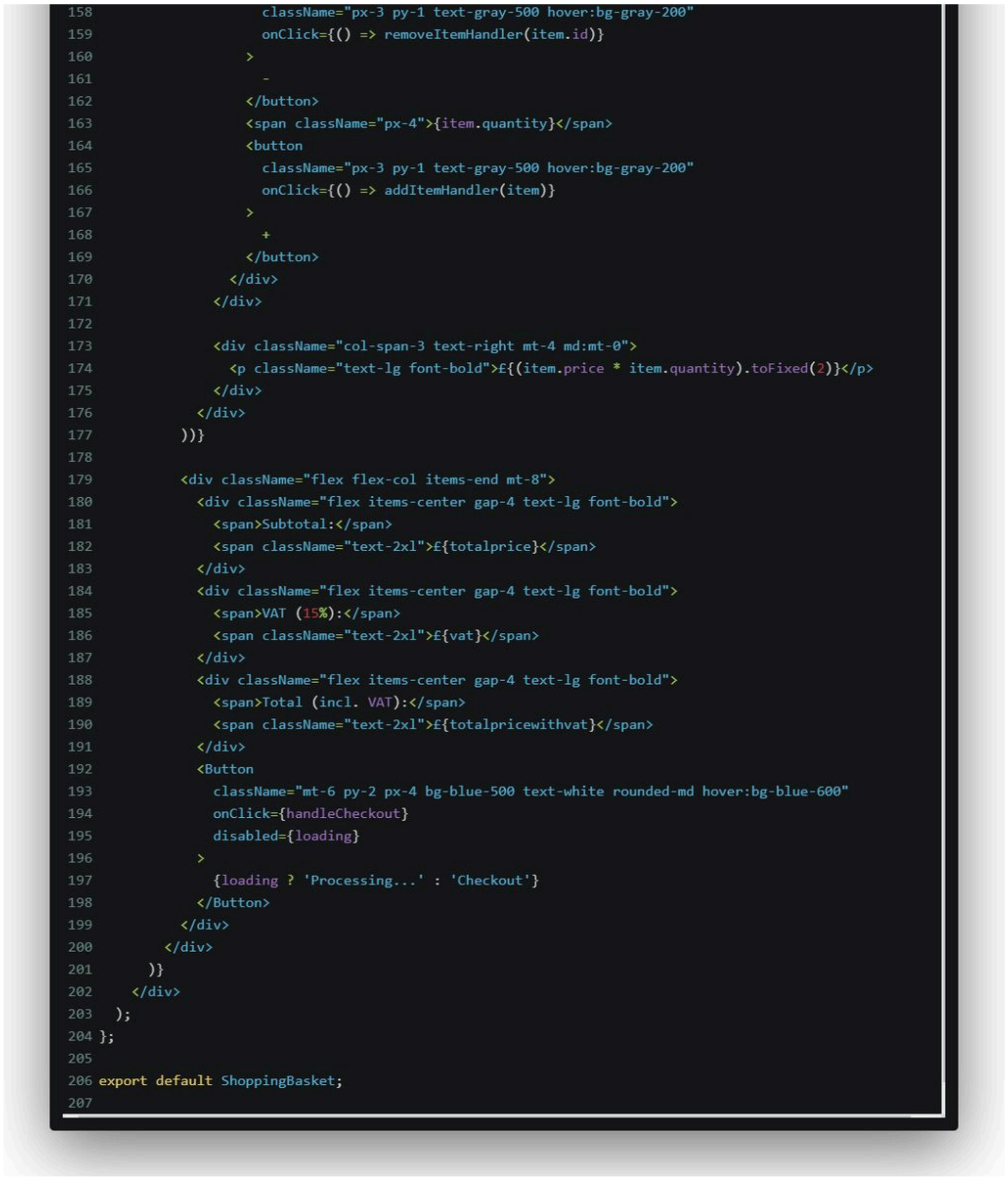
Product Cart page Component



```tsx
// Code here'use client';

import React, { useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import Image from 'next/image';
import Link from 'next/link';
import { useRouter } from 'next/navigation'; // Use `next/navigation` for App Router
import { Button } from '@/components/ui/button';
import { addItem, removeItem, CartItem } from '../../../store/cartSlice';
import { RooteState } from '../../../store/store';
import { client } from '@/sanity/lib/client'; // Import Sanity client
import { SanityImageSource } from '@sanity/image-url/lib/types/types';

interface Product {
  id: string | number;
  _id: string;
  slug: { current: string };
  name: string;
  price: number;
  image: SanityImageSource[];
  stock: number | undefined;
  status: 'sale' | 'hot' | 'new' | undefined;
}

const ShoppingBasket = () => {
  const dispatch = useDispatch();
  const router = useRouter(); // Initialize the router from `next/navigation`
  const items = useSelector((state: RooteState) => state.cart.items);
  const [loading, setLoading] = useState(false);

  const totalQuantity = items.reduce((total, item) => total + item.quantity, 0);
  const totalprice = items.reduce((total, item) => total + item.price * item.quantity, 0).toFixed(2);
  const vat = (+totalprice * 0.15).toFixed(2);
  const totalpricewithvat = (+totalprice + +vat).toFixed(2);

  const addItemHandler = async (item: CartItem) => {
    if (item.stock > 0) {
      dispatch(addItem(item));
      try {
        await client
          .patch(String(item.id))
          .setIfMissing({ stock: item.stock })
          .dec({ stock: 1 })
          .commit();
      } catch (error) {
        console.error('Failed to update stock:', error);
      }
    } else {
      alert('Out of stock!');
    }
  };

  const removeItemHandler = async (id: number | string) => {
    const itemToRemove = items.find((item) => item.id === id);

    if (itemToRemove) {
      dispatch(removeItem({ id }));
      try {
        await client
          .patch(String(id))
          .setIfMissing({ stock: itemToRemove.stock })
          .inc({ stock: 1 })
          .commit();
      } catch (error) {
        console.error('Failed to restore stock:', error);
      }
    }
  };

  const handleCheckout = async () => {
    setLoading(true);

    try {
      for (const item of items) {
        const productData = await client.fetch(
          `*[_type == "product" && _id == $id][0]`,
          { id: String(item.id) }
        );

```

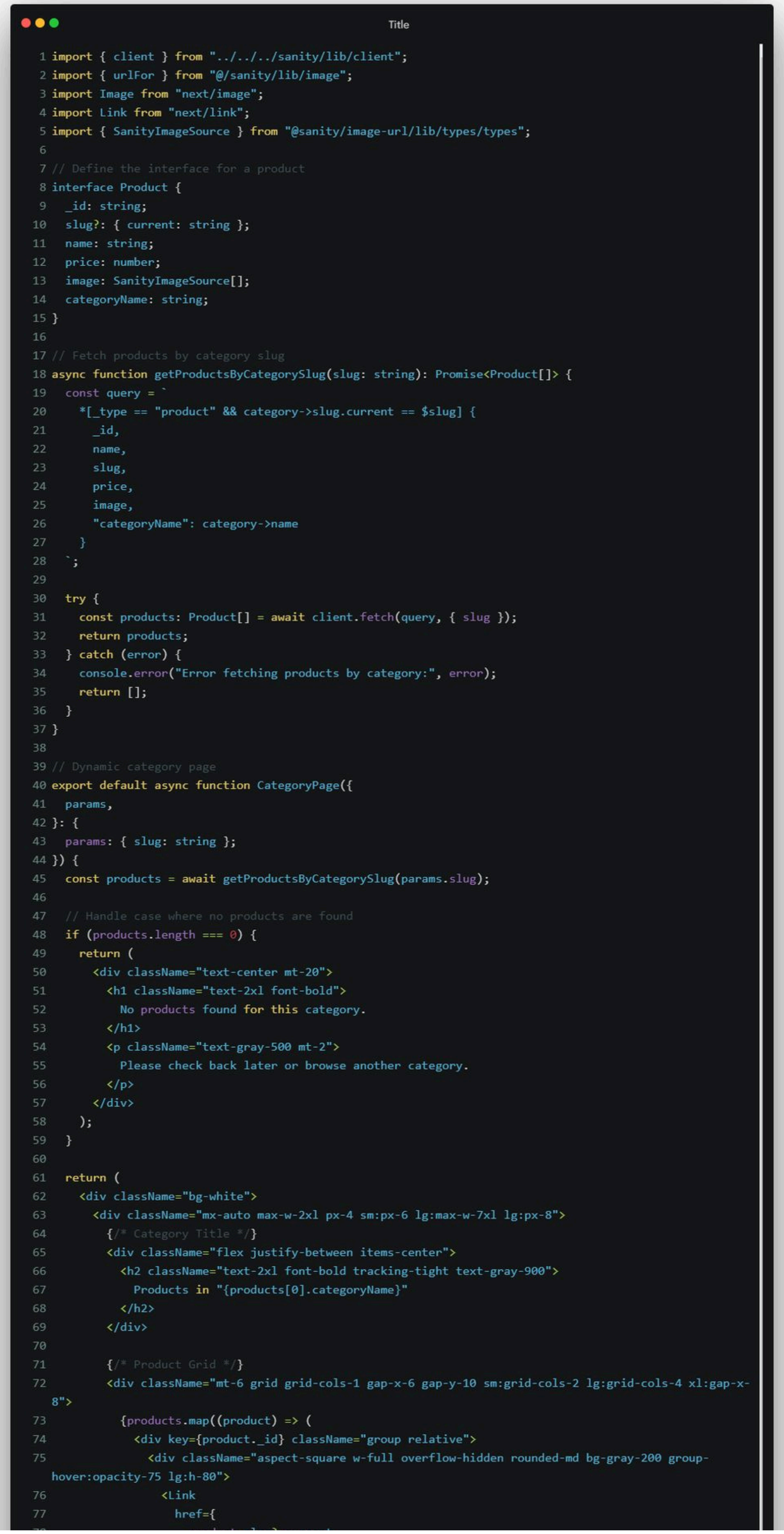


```
        if (!productData) {
          alert(`Product "${item.title}" not found in stock!`);
          continue;
        }

        if (productData.stock <= 0) {
          alert(`Product "${item.title}" is out of stock!`);
          setLoading(false);
          return;
        }

        if (productData.stock < item.quantity) {
          alert(
            `Product "${item.title}" has limited stock (${productData.stock} available). Reduce quantity.`
          );
          setLoading(false);
          return;
        }

        await client
          .patch(productData._id)
          .dec({ stock: item.quantity })
          .commit();

        console.log(`Stock updated for "${item.title}"!`);
      }

      alert('Checkout successful! Stock updated.');

      // Redirect to shipment page
      router.push('/shippment');
    } catch (error) {
      console.error('Error during checkout:', error);
      alert('Failed to process checkout.');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="w-full min-h-screen bg-light-grey text-dark-primary font-body-medium">
      <header className="py-6 px-4 sm:px-8 flex justify-between items-center bg-white shadow-md">
        <h1 className=" text-17xl font-headings-h1 font-bold">Your Shopping Cart</h1>
      </header>

      {items.length === 0 && (
        <div className="flex items-center w-full h-[80vh] flex-col justify-center">
          <Image src="/images/cart.svg" alt="cart" width={600} height={600} className="object-center mx-auto" />
          <h1 className="mt-8 text-2xl font-semibold">Your Cart is empty</h1>
          <Link href="/" className="mt-4">
            <Button>Shop Now</Button>
          </Link>
        </div>
      )}

      {items.length > 0 && (
        <div className="container mx-auto mt-8 px-4 sm:px-8">
          <div className="hidden md:grid grid-cols-12 py-4 px-6 bg-dark-primary rounded-lg text-sm font-bold">
            <div className="col-span-6 text-5xl text-white">Product</div>
            <div className="col-span-3 text-center text-5xl text-white ">Quantity ({totalQuantity} items)</div>
            <div className="col-span-3 text-right text-5xl text-white">Total</div>
          </div>

          {items.map((item) => (
            <div
              key={item.id}
              className="grid grid-cols-1 md:grid-cols-12 py-6 px-4 sm:px-6 bg-white shadow-sm rounded-lg mb-4 items-center"
            >
              <div className="col-span-6 flex items-center gap-4">
                <Image src={item.image} alt={item.title} width={180} height={180} className="rounded-md" />
                <div>
                  <p className="font-bold text-lg">{item.title}</p>
                </div>
              </div>

              <div className="col-span-3 flex justify-center mt-4 md:mt-0">
                <div className="flex items-center border border-gray-300 rounded-lg overflow-hidden">
                  <button
```

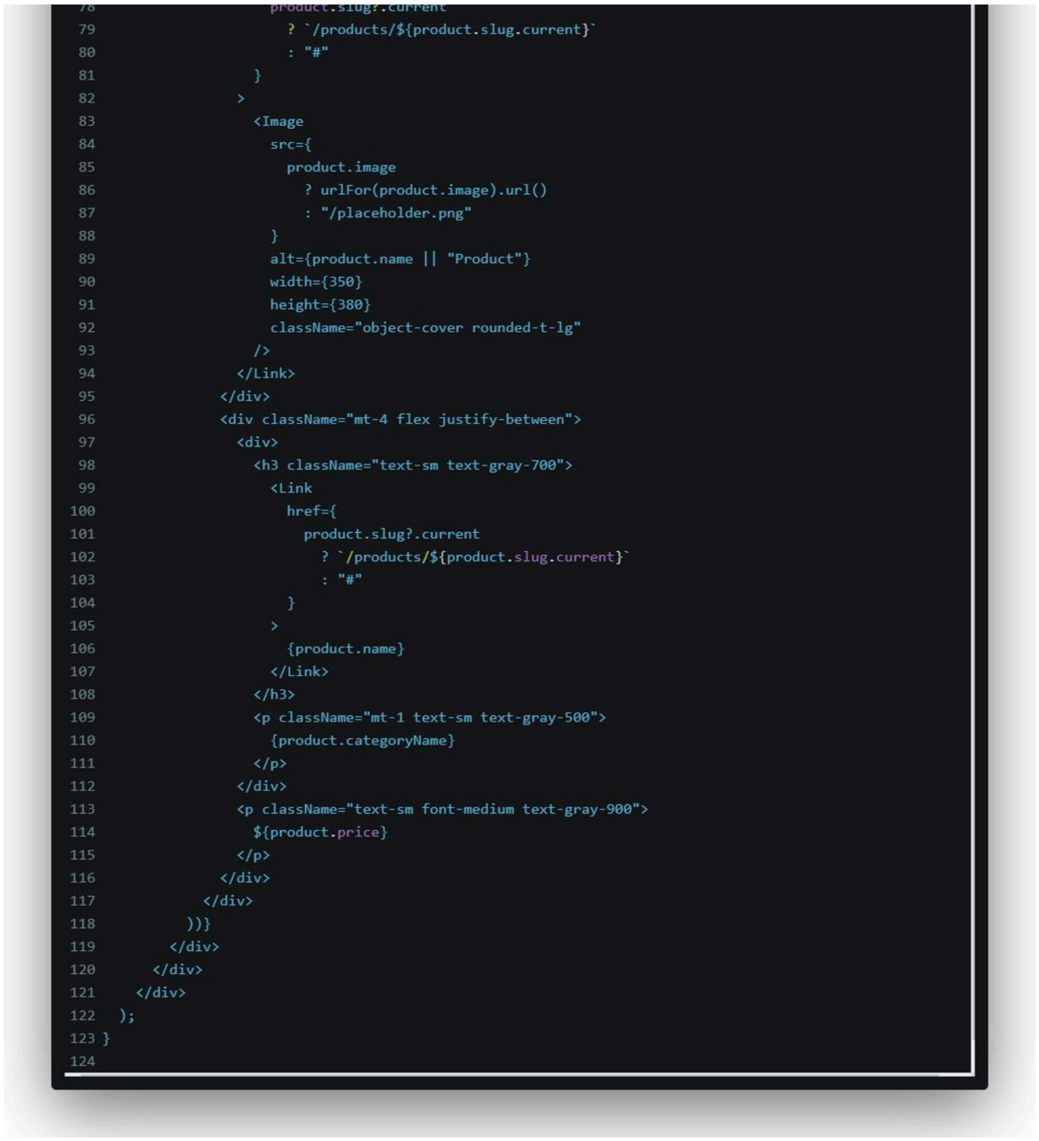


```
                    className="px-3 py-1 text-gray-500 hover:bg-gray-200"
                    onClick={() => removeItemHandler(item.id)}
                  >
                    -
                  </button>
                  <span className="px-4">{item.quantity}</span>
                  <button
                    className="px-3 py-1 text-gray-500 hover:bg-gray-200"
                    onClick={() => addItemHandler(item)}
                  >
                    +
                  </button>
                </div>
              </div>

              <div className="col-span-3 text-right mt-4 md:mt-0">
                <p className="text-lg font-bold">£{(item.price * item.quantity).toFixed(2)}</p>
              </div>
            </div>
          ))}

          <div className="flex flex-col items-end mt-8">
            <div className="flex items-center gap-4 text-lg font-bold">
              <span>Subtotal:</span>
              <span className="text-2xl">£{totalprice}</span>
            </div>
            <div className="flex items-center gap-4 text-lg font-bold">
              <span>VAT (15%):</span>
              <span className="text-2xl">£{vat}</span>
            </div>
            <div className="flex items-center gap-4 text-lg font-bold">
              <span>Total (incl. VAT):</span>
              <span className="text-2xl">£{totalpricewithvat}</span>
            </div>
            <Button
              className="mt-6 py-2 px-4 bg-blue-500 text-white rounded-md hover:bg-blue-600"
              onClick={handleCheckout}
              disabled={loading}
            >
              {loading ? 'Processing...' : 'Checkout'}
            </Button>
          </div>
        </div>
      )}
    </div>
  );
};

export default ShoppingBasket;
```

Product Category page Component



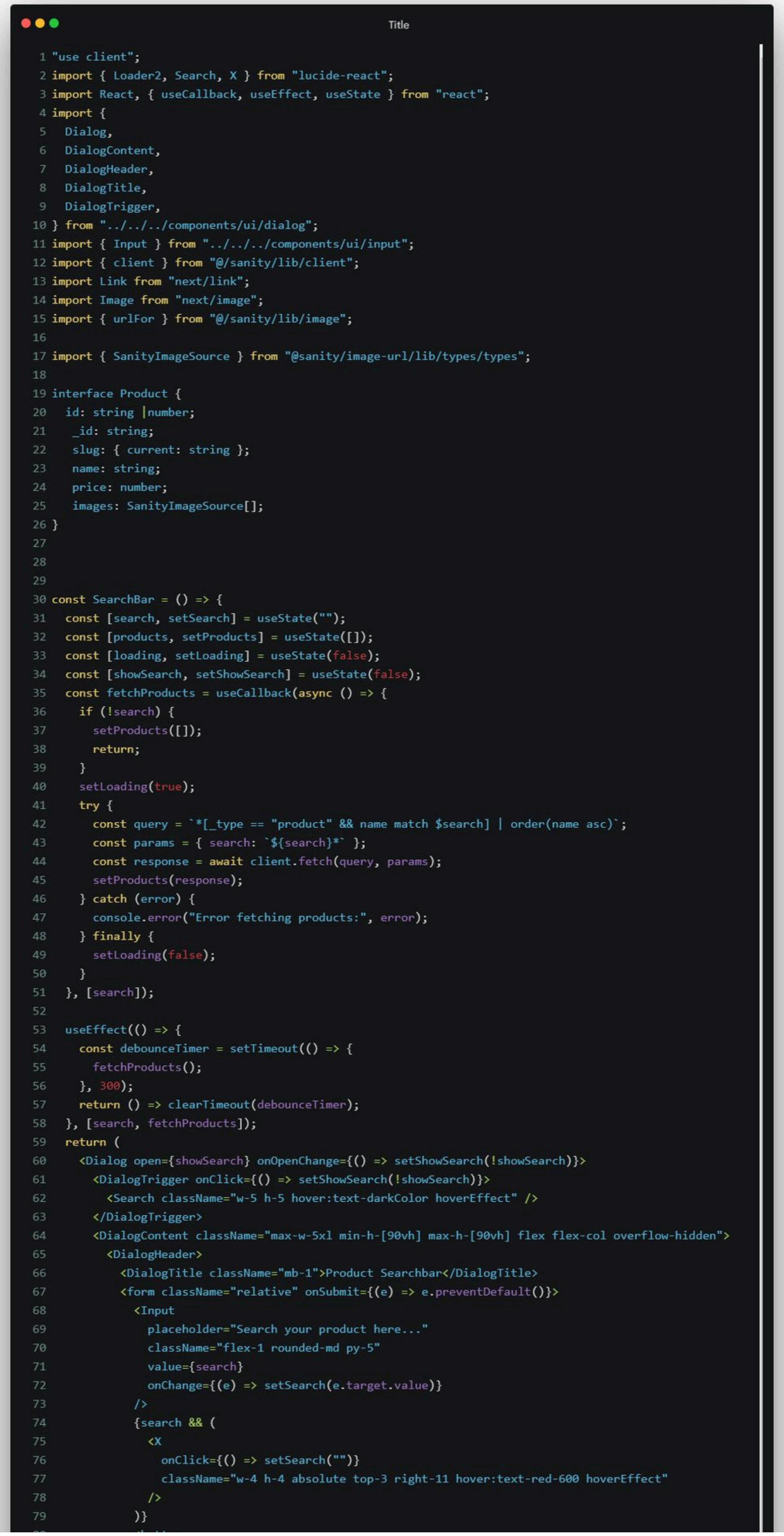
```
import { client } from "../../../sanity/lib/client";
import { urlFor } from "@/sanity/lib/image";
import Image from "next/image";
import Link from "next/link";
import { SanityImageSource } from "@sanity/image-url/lib/types/types";

// Define the interface for a product
interface Product {
  _id: string;
  slug?: { current: string };
  name: string;
  price: number;
  image: SanityImageSource[];
  categoryName: string;
}

// Fetch products by category slug
async function getProductsByCategorySlug(slug: string): Promise<Product[]> {
  const query = `
    *[_type == "product" && category->slug.current == $slug] {
      _id,
      name,
      slug,
      price,
      image,
      "categoryName": category->name
    }
  `;

  try {
    const products: Product[] = await client.fetch(query, { slug });
    return products;
  } catch (error) {
    console.error("Error fetching products by category:", error);
    return [];
  }
}

// Dynamic category page
export default async function CategoryPage({
  params,
}: {
  params: { slug: string };
}) {
  const products = await getProductsByCategorySlug(params.slug);

  // Handle case where no products are found
  if (products.length === 0) {
    return (
      <div className="text-center mt-20">
        <h1 className="text-2xl font-bold">
          No products found for this category.
        </h1>
        <p className="text-gray-500 mt-2">
          Please check back later or browse another category.
        </p>
      </div>
    );
  }

  return (
    <div className="bg-white">
      <div className="mx-auto max-w-2xl px-4 sm:px-6 lg:max-w-7xl lg:px-8">
        {/* Category Title */}
        <div className="flex justify-between items-center">
          <h2 className="text-2xl font-bold tracking-tight text-gray-900">
            Products in "{products[0].categoryName}"
          </h2>
        </div>

        {/* Product Grid */}
        <div className="mt-6 grid grid-cols-1 gap-x-6 gap-y-10 sm:grid-cols-2 lg:grid-cols-4 xl:gap-x-8">
          {products.map((product) => (
            <div key={product._id} className="group relative">
              <div className="aspect-square w-full overflow-hidden rounded-md bg-gray-200 group-hover:opacity-75 lg:h-80">
                <Link
                  href={
```



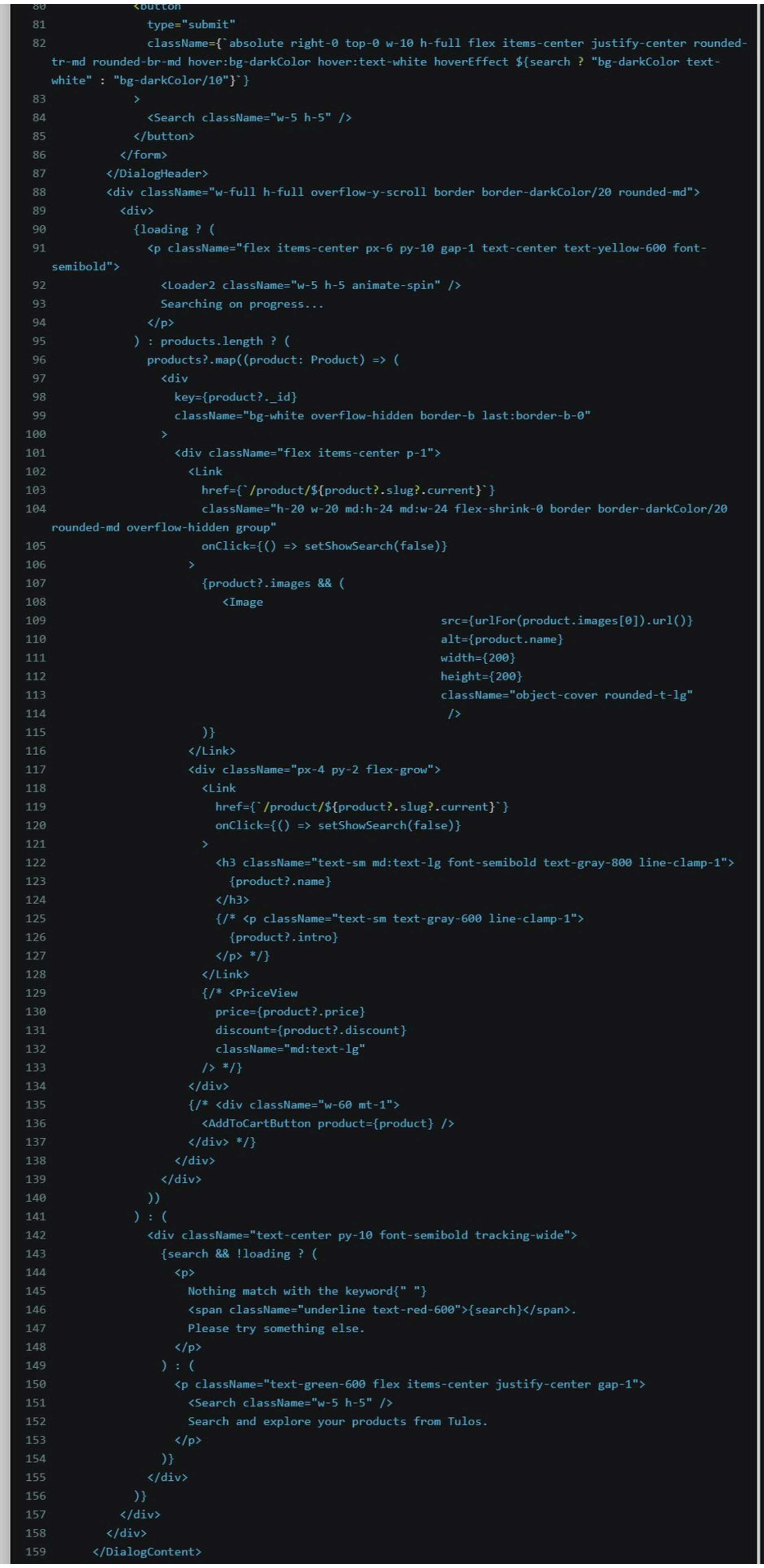
```
                    product.slug?.current
                      ? `/products/${product.slug.current}`
                      : "#"
                  }
                >
                  <Image
                    src={
                      product.image
                        ? urlFor(product.image).url()
                        : "/placeholder.png"
                    }
                    alt={product.name || "Product"}
                    width={350}
                    height={380}
                    className="object-cover rounded-t-lg"
                  />
                </Link>
              </div>
              <div className="mt-4 flex justify-between">
                <div>
                  <h3 className="text-sm text-gray-700">
                    <Link
                      href={
                        product.slug?.current
                          ? `/products/${product.slug.current}`
                          : "#"
                      }
                    >
                      {product.name}
                    </Link>
                  </h3>
                  <p className="mt-1 text-sm text-gray-500">
                    {product.categoryName}
                  </p>
                </div>
                <p className="text-sm font-medium text-gray-900">
                  ${product.price}
                </p>
              </div>
            </div>
          ))}
        </div>
      </div>
    </div>
  );
}
```

SearchBar Component

Provides search functionality to filter products dynamically:



```
"use client";
import { Loader2, Search, X } from "lucide-react";
import React, { useCallback, useEffect, useState } from "react";
import {
  Dialog,
  DialogContent,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from "../../../components/ui/dialog";
import { Input } from "../../../components/ui/input";
import { client } from "@/sanity/lib/client";
import Link from "next/link";
import Image from "next/image";
import { urlFor } from "@/sanity/lib/image";

import { SanityImageSource } from "@sanity/image-url/lib/types/types";

interface Product {
  id: string |number;
   _id: string;
   slug: { current: string };
   name: string;
   price: number;
   images: SanityImageSource[];
}



const SearchBar = () => {
  const [search, setSearch] = useState("");
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(false);
  const [showSearch, setShowSearch] = useState(false);
  const fetchProducts = useCallback(async () => {
    if (!search) {
      setProducts([]);
      return;
    }
    setLoading(true);
    try {
      const query = `*[_type == "product" && name match $search] | order(name asc)`;
      const params = { search: `${search}*` };
      const response = await client.fetch(query, params);
      setProducts(response);
    } catch (error) {
      console.error("Error fetching products:", error);
    } finally {
      setLoading(false);
    }
  }, [search]);

  useEffect(() => {
    const debounceTimer = setTimeout(() => {
      fetchProducts();
    }, 300);
    return () => clearTimeout(debounceTimer);
  }, [search, fetchProducts]);
  return (
    <Dialog open={showSearch} onOpenChange={() => setShowSearch(!showSearch)}>
      <DialogTrigger onClick={() => setShowSearch(!showSearch)}>
        <Search className="w-5 h-5 hover:text-darkColor hoverEffect" />
      </DialogTrigger>
      <DialogContent className="max-w-5xl min-h-[90vh] max-h-[90vh] flex flex-col overflow-hidden">
        <DialogHeader>
          <DialogTitle className="mb-1">Product Searchbar</DialogTitle>
          <form className="relative" onSubmit={(e) => e.preventDefault()}>
            <Input
              placeholder="Search your product here..."
              className="flex-1 rounded-md py-5"
              value={search}
              onChange={(e) => setSearch(e.target.value)}
            />
            {search && (
              <X
                onClick={() => setSearch("")}
                className="w-4 h-4 absolute top-3 right-11 hover:text-red-600 hoverEffect"
              />
            )}
```



```
            <button
              type="submit"
              className={`absolute right-0 top-0 w-10 h-full flex items-center justify-center rounded-tr-md rounded-br-md hover:bg-darkColor hover:text-white hoverEffect ${search ? "bg-darkColor text-white" : "bg-darkColor/10"}`}
            >
              <Search className="w-5 h-5" />
            </button>
          </form>
        </DialogHeader>
        <div className="w-full h-full overflow-y-scroll border border-darkColor/20 rounded-md">
          <div>
            {loading ? (
              <p className="flex items-center px-6 py-10 gap-1 text-center text-yellow-600 font-semibold">
                <Loader2 className="w-5 h-5 animate-spin" />
                Searching on progress...
              </p>
            ) : products.length ? (
              products?.map((product: Product) => (
                <div
                  key={product?._id}
                  className="bg-white overflow-hidden border-b last:border-b-0"
                >
                  <div className="flex items-center p-1">
                    <Link
                      href={`/product/${product?.slug?.current}`}
                      className="h-20 w-20 md:h-24 md:w-24 flex-shrink-0 border border-darkColor/20 rounded-md overflow-hidden group"
                      onClick={() => setShowSearch(false)}
                    >
                      {product?.images && (
                         <Image
                                                         src={urlFor(product.images[0]).url()}
                                                         alt={product.name}
                                                         width={200}
                                                         height={200}
                                                         className="object-cover rounded-t-lg"
                                                          />
                      )}
                    </Link>
                    <div className="px-4 py-2 flex-grow">
                      <Link
                        href={`/product/${product?.slug?.current}`}
                        onClick={() => setShowSearch(false)}
                      >
                        <h3 className="text-sm md:text-lg font-semibold text-gray-800 line-clamp-1">
                          {product?.name}
                        </h3>
                        {/* <p className="text-sm text-gray-600 line-clamp-1">
                          {product?.intro}
                        </p> */}
                      </Link>
                      {/* <PriceView
                        price={product?.price}
                        discount={product?.discount}
                        className="md:text-lg"
                      /> */}
                    </div>
                    {/* <div className="w-60 mt-1">
                      <AddToCartButton product={product} />
                    </div> */}
                  </div>
                </div>
              ))
            ) : (
              <div className="text-center py-10 font-semibold tracking-wide">
                {search && !loading ? (
                  <p>
                    Nothing match with the keyword{" "}
                    <span className="underline text-red-600">{search}</span>.
                    Please try something else.
                  </p>
                ) : (
                  <p className="text-green-600 flex items-center justify-center gap-1">
                    <Search className="w-5 h-5" />
                    Search and explore your products from Tulos.
                  </p>
                )}
              </div>
            )}
          </div>
        </div>
      </DialogContent>
```

```
    </Dialog>
  );
};

export default SearchBar;
```

Tools and Libraries Used

1. Sanity CMS

- **Command**: `npm create sanity@latest`
- **Purpose**: Sanity serves as the CMS backend, enabling the dynamic management of product data, categories, billing information, and reviews.
- **Usage**: Configured schemas for `product`, `category`, and `billingAddress`. Utilized Sanity's GROQ queries for fetching dynamic content.

2. Tailwind CSS and Tailwind Plugins

- **Command**: `npm install tailwindcss-animate`
- **Purpose**: Used Tailwind CSS for styling and `tailwindcss-animate` to add animations for enhancing user experience.
- **Usage**: Created responsive layouts for components like product listings, modals, and forms.

3. Lucide React

- **Command**: `npm install lucide-react`
- **Purpose**: Icon library providing modern SVG-based icons.
- **Usage**: Icons were used for UI elements such as navigation menus, buttons, and informational tooltips.

4. Form Handling

- **Command**:
  - `npm install react-hook-form @hookform/resolvers`
  - `npm install zod`
- **Purpose**: `react-hook-form` is used for efficient form state management, and `zod` is used for schema-based form validation.
- **Usage**: Implemented in the multi-step billing and payment forms. Ensured form inputs were validated dynamically before submission.

5. Shadcn Components

- **Commands**:
  - `npx shadcn add form input select`
  - `npx shadcn add sheet`
- **Purpose**: Added pre-styled components for consistent design and enhanced UI.
- **Usage**: Styled input fields, select dropdowns, and modular UI elements such as sheets for dialogs or side panels.

6. Toast Notifications

- **Command**: `npm install @radix-ui/react-toast`
- **Purpose**: Display real-time notifications, such as "Order Placed" or "Product Added to Cart."
- **Usage**: Integrated toast notifications for user feedback.

7. Tooltips

- **Commands**:

  - `npm install @radix-ui/react-tooltip`
  - `npm show @radix-ui/react-tooltip version`
- **Purpose**: Enhanced UX by providing additional context on hover for icons or actions.
- **Usage**: Applied to navigation icons and action buttons.

8. Icons

- **Command**: `npm install react-icons`
- **Purpose**: Provided scalable vector icons for aesthetic and functional UI elements.
- **Usage**: Used in headers, footers, and action components.

9. Carousel

- **Command**: `npm install react-slick slick-carousel`
- **Purpose**: Implemented product carousels for showcasing featured or related products.
- **Usage**: Added in the product detail page and homepage sections.

10. Version Control with Git

- **Commands**:
  - `git pull`
  - `git status`
  - `git add .`
  - `git commit -m "Updated responsive styles"`
  - `git push origin main`
- **Purpose**: Managed code changes, tracked progress, and ensured collaboration.
- **Usage**: Maintained a structured repository for version control and deployment.

Dynamic Features Implemented

1. **Dynamic Product Listing**:
   - **Description**: Utilized GROQ queries from Sanity to fetch and display products dynamically.
   - **Components**: `ProductCard`, `ProductList`.
2. **Product Detail Pages**:
   - **Description**: Implemented Next.js dynamic routing for product detail pages.
   - **Key Features**: Displayed product details such as images, price, description, and related products.
3. **Search and Filter**:
   - **Description**: Enabled users to search and filter products by categories, price range, and tags.
   - **Components**: `SearchBar`, `FilterPanel`.
4. **Cart Management**:
   - **Description**: Built features for adding/removing items, updating quantities, and viewing the cart summary.
   - **Components**: `Cart`, `OrderSummary`, `ShipmentForm`.
5. **Billing and Payment Forms**:
   - **Description**: Implemented multi-step forms with validation using `react-hook-form` and `zod`.
   - **Key Features**: Form validation, data submission, and dynamic field rendering.
6. **Notifications and Alerts**:
   - **Description**: Real-time feedback using toast notifications.
   - **Components**: `Toast` from Radix UI.

7. **Admin Dashboard**:
    - **Description**: Designed admin-specific pages for product and user management.
    - **Components**: Charts and graphs displaying key performance indicators.

Challenges Faced and Solutions

1. **Challenge**: Handling complex form validations for billing and payment.
    - **Solution**: Utilized `react-hook-form` with `zod` schemas for robust validation.
2. **Challenge**: Dynamic routing for product detail pages.
    - **Solution**: Leveraged Next.js dynamic routing and Sanity's slug field for seamless navigation.
3. **Challenge**: Ensuring responsiveness across devices.
    - **Solution**: Used Tailwind CSS for building responsive layouts and components.

Best Practices Followed

- **Component Reusability**: Ensured components like `ProductCard` and `SearchBar` were reusable across different pages.
- **State Management**: Kept forms and cart states isolated using `react-hook-form` and local state.
- **Version Control**: Maintained clear commit messages and a structured repository.
- **Dynamic Content**: Relied heavily on Sanity CMS for flexibility in content management.

Project Overview

This e-commerce marketplace project successfully combines modern frontend and backend technologies to deliver a dynamic, user-friendly shopping experience. By leveraging Next.js for robust dynamic routing, Sanity CMS for flexible content management, and a range of powerful libraries for UI components, the platform ensures seamless product browsing, efficient cart management, and intuitive order processing. The implementation of advanced features like real-time notifications, multi-step forms, and admin dashboards highlights the project's focus on scalability and responsiveness. Overall, this marketplace is designed to meet user needs while providing an adaptable foundation for future enhancements.