Day 04:Building a Dynamic Frontend Components For Marketplace

On Day 4, I focused on designing and developing dynamic frontend components for my marketplace. I worked on creating modular and reusable components to display data fetched from Sanity CMS and APIs. This day helped me learn real-world practices for building scalable and responsive web applications, ensuring my components could be easily maintained and adapted for future needs. I gained hands-on experience in structuring components to handle marketplace data effectively, improving both functionality and user experience.

Key components to Build:

Product Listing Components:

Product Listing Component Documentation

The Newest component fetches and displays a list of products stored in a Sanity CMS. It provides users with a scrollable horizontal carousel view of the latest products and includes features like product details, images, and an "Add to Bag" button for interaction. The component is responsive and styled using TailwindCSS.

Features

1. Dynamic Data Fetching:

Fetches product data using Sanity's GROQ query.

Displays the most recent products sorted by creation date.

2. Horizontal Scrolling:

Users can scroll through the product list using left/right arrow buttons or by swiping.

3. Interactive UI:

Displays product image, name, category, price, and rating.

Includes an "Add to Bag" button for quick actions.

4. Responsive Design:

Optimized for various screen sizes using TailwindCSS classes.

5. Dynamic Routing:

Clicking on a product navigates to its detailed page using a dynamic route (/product/[slug]).

Code Structure

Data Fetching

Function: fetchProducts()

Fetches products from Sanity CMS using the GROQ query.

Retrieves fields like id, name, price, category, slug, and image URL.

State Management

State Variables:

data: Stores the fetched product data.

Effect Hook:

useEffect fetches product data when the component mounts.

Scrolling Functionality

Functions:

scrollLeft: Scrolls the carousel to the left.

scrollRight: Scrolls the carousel to the right.

Implementation:

Uses scrollBy method on a ref pointing to the carousel container.

Rendering

Product Display:

Loops through the fetched data and renders individual product cards.

Each card includes:

Product image.

Name (linked to the product detail page).

Category name.

Price.

"Add to Bag" and "Rating" components.

Carousel Controls:

Left and right buttons for smooth scrolling.

Sanity GROQ Query

Purpose: Fetches all products of type product sorted by their creation date in descending order.

Styling

Libraries Used:

TailwindCSS: For layout, spacing, and responsive design.

Lucide Icons: For left and right navigation arrows.

Component Props

No external props are passed to this component.

Dependencies

1. Next.js: For dynamic routing and server-side rendering.

2. Sanity Client: For fetching data from Sanity CMS.

3. TailwindCSS: For styling.

4. Lucide-React: For icons.

5. React Hooks: For state and lifecycle management.

Sample Product Schema in Sanity

Usage Instructions

1. Add products to Sanity CMS following the provided schema.

2. Integrate the component into your Next.js project.

3. Customize styling or functionality as needed.

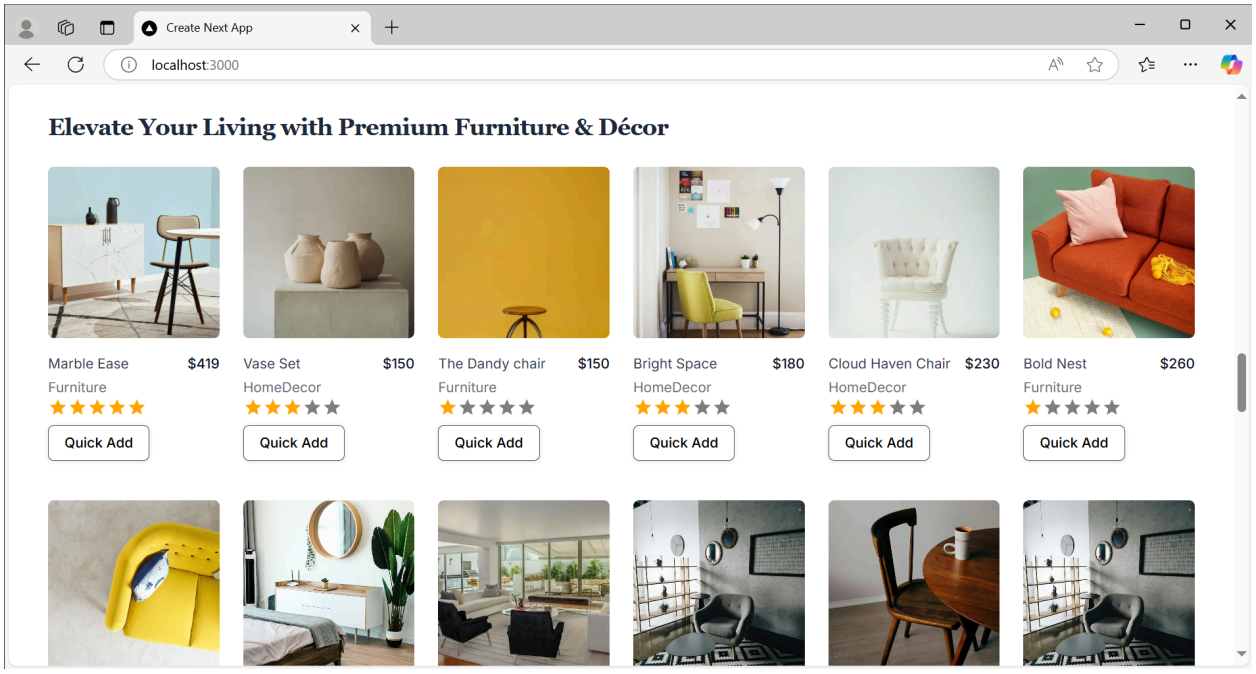
Future Enhancements

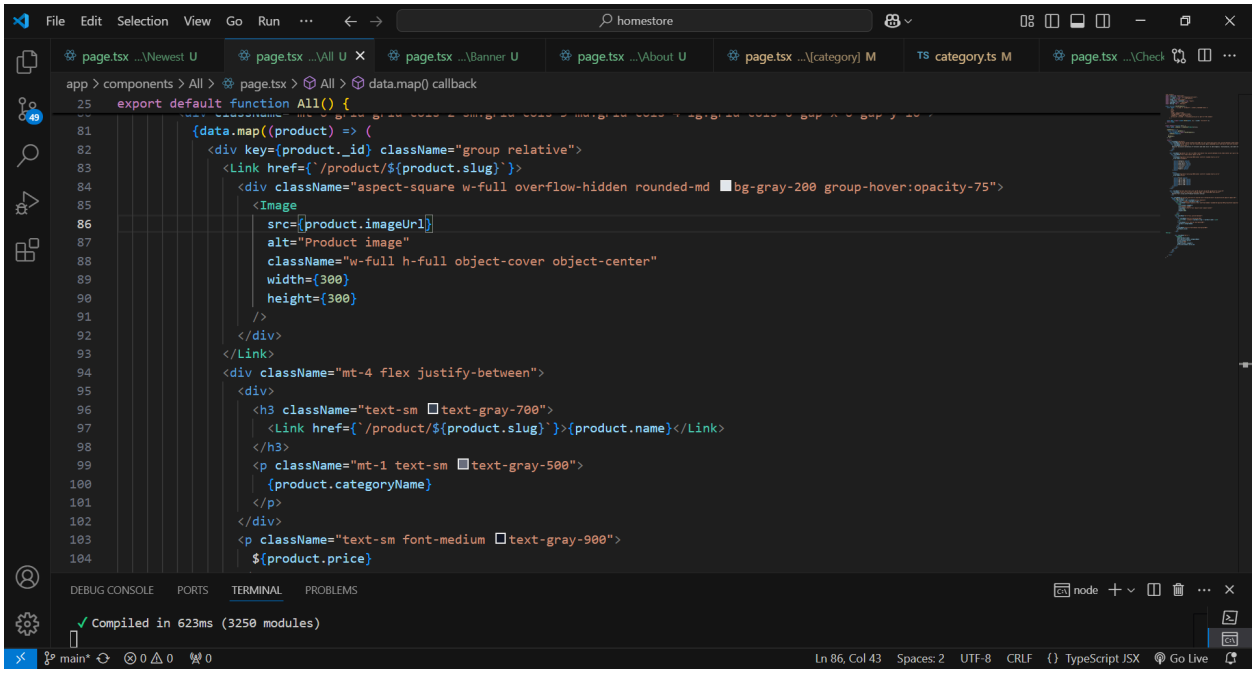
1. Add filtering options (e.g., price range, category).

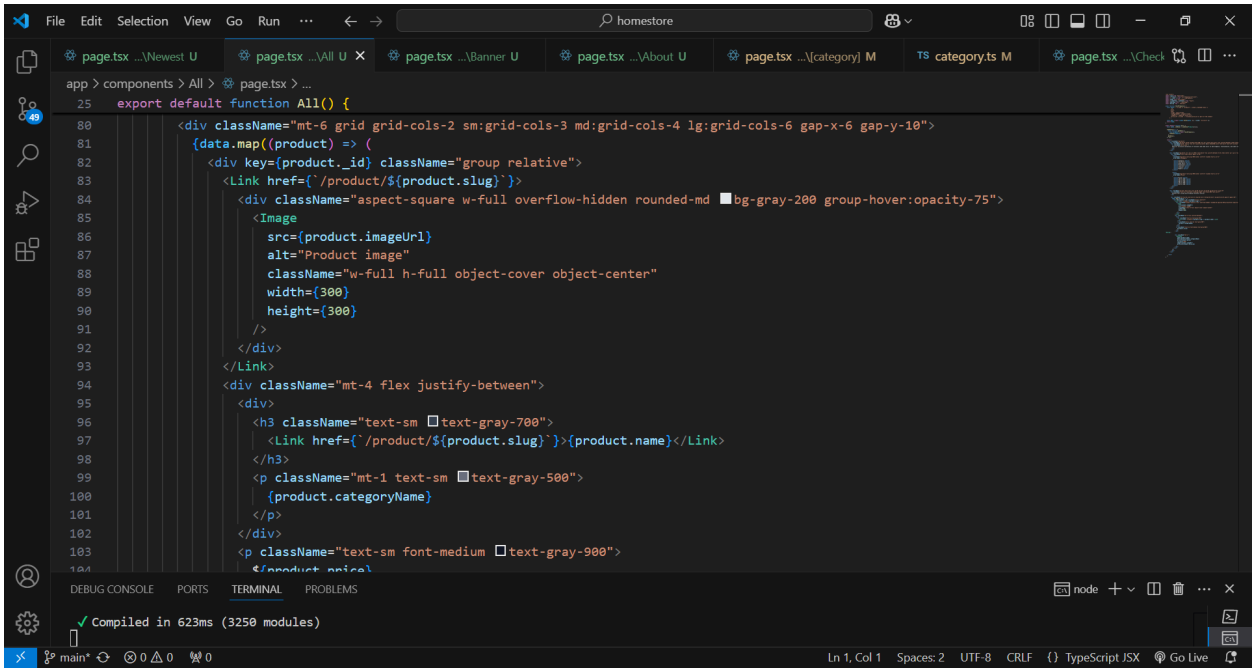
2. Enable infinite scrolling for larger product lists.

3. Include a loading spinner while fetching data.

4. Implement error handling for data fetching failures.







Product Detail Page:

The Product Detail Page allows users to view detailed information about a selected product. This page includes product images, pricing, category, ratings, description, quantity adjustment, reviews, and options to add the product to the cart or proceed directly to checkout. The page is dynamic and fetches product data based on the product slug.

Features and Functionalities

1. Dynamic Data Fetching:

The getData function retrieves product data from Sanity CMS using GROQ queries.

The data includes product images, name, price, description, category name, slug, and price ID.

This ensures each product detail page is unique and dynamically rendered based on the product slug.

2. Image Gallery:

The ImageGallery component displays a carousel of product images.

Users can view multiple images of the product for better understanding.

3. Product Details:

Category Name: Displayed above the product name.

Product Name: Highlighted in a bold font.

Ratings: Integrated using the RatingComponent for customer reviews and ratings.

4. Pricing Section:

Displays the current price prominently.

Includes a strikethrough of the original price to indicate discounts.

Mentions that the price is inclusive of VAT.

5. Quantity Adjustment:

The Quantity component allows users to increase or decrease the product quantity before adding it to the cart.

6. Add to Bag and Checkout:

Users can add the product to their shopping bag using the AddToBag component.

A "Check Out" button redirects users to the checkout page for completing their order.

7. Shipping Details:

Includes an estimated shipping time using the Truck icon for better user clarity.

8. Product Description:

Provides a detailed explanation of the product features and specifications.

Code Structure

1. Imports:

Components such as AddToBag, ImageGallery, Quantity, and RatingComponent are imported for modular functionality.

Icons from the lucide-react library, such as ArrowRight, ArrowLeft, and Truck, enhance the visual appeal.

The Link component from Next.js handles navigation to the checkout page.

The ProductPage function uses the params object to access the product slug.

The getData function fetches product details based on the slug.

3. Dynamic Rendering:

The dynamic property is set to force-dynamic to ensure the page re-renders dynamically for each request.

4. Component Composition:

The layout is divided into a two-column grid using TailwindCSS, making the design responsive and visually appealing.

Technologies Used

1. Next.js:

Used for dynamic routing and server-side rendering.

The params object fetches the slug to display product-specific information.

2. Sanity CMS:

Stores and manages product data, including images, pricing, and descriptions.

3. TailwindCSS:

Provides a responsive and modern design for the page layout.

4. Lucide-React Icons:

Adds visually appealing icons for shipping details and navigation.

5. Reusable Components:

ImageGallery, AddToBag, Quantity, and RatingComponent ensure reusability and maintainability.

Page Layout and Design

Responsive Design:

TailwindCSS ensures the layout adapts to various screen sizes, including mobile, tablet, and desktop.

Grid System:

A two-column grid displays product images on the left and details on the right for better usability.

Usage Workflow

1. Users select a product from the product listing page.

2. They are redirected to the product detail page via the dynamic route (/product/[slug]).

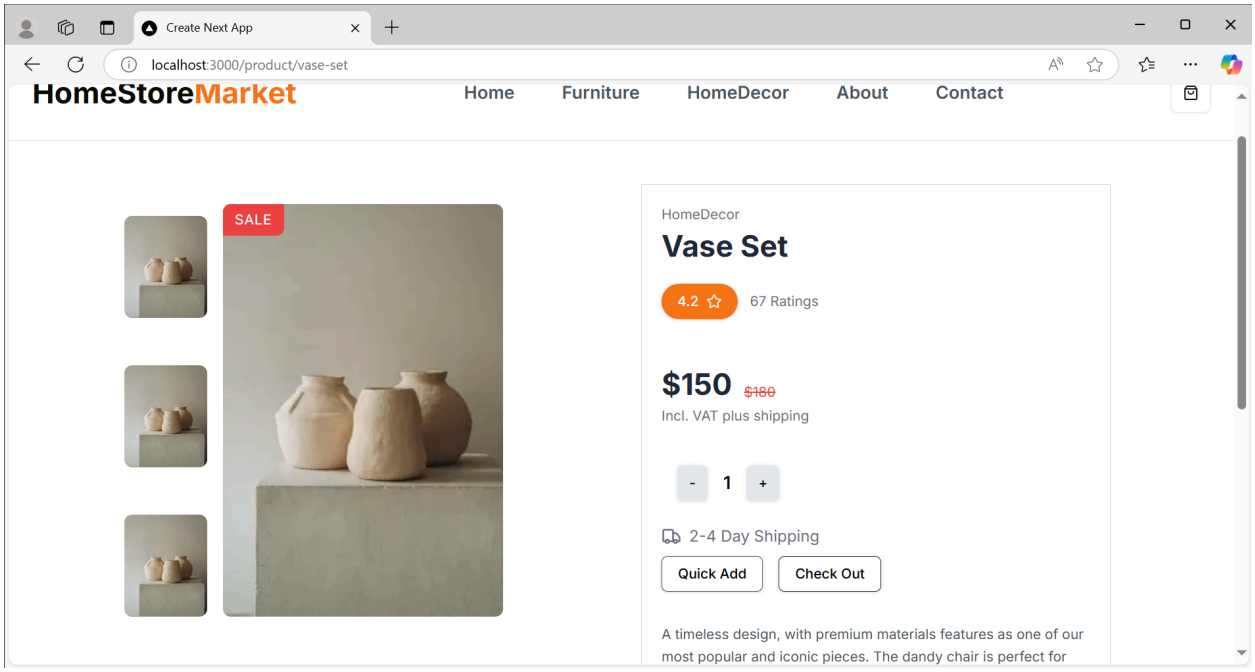
3. On this page, users can:

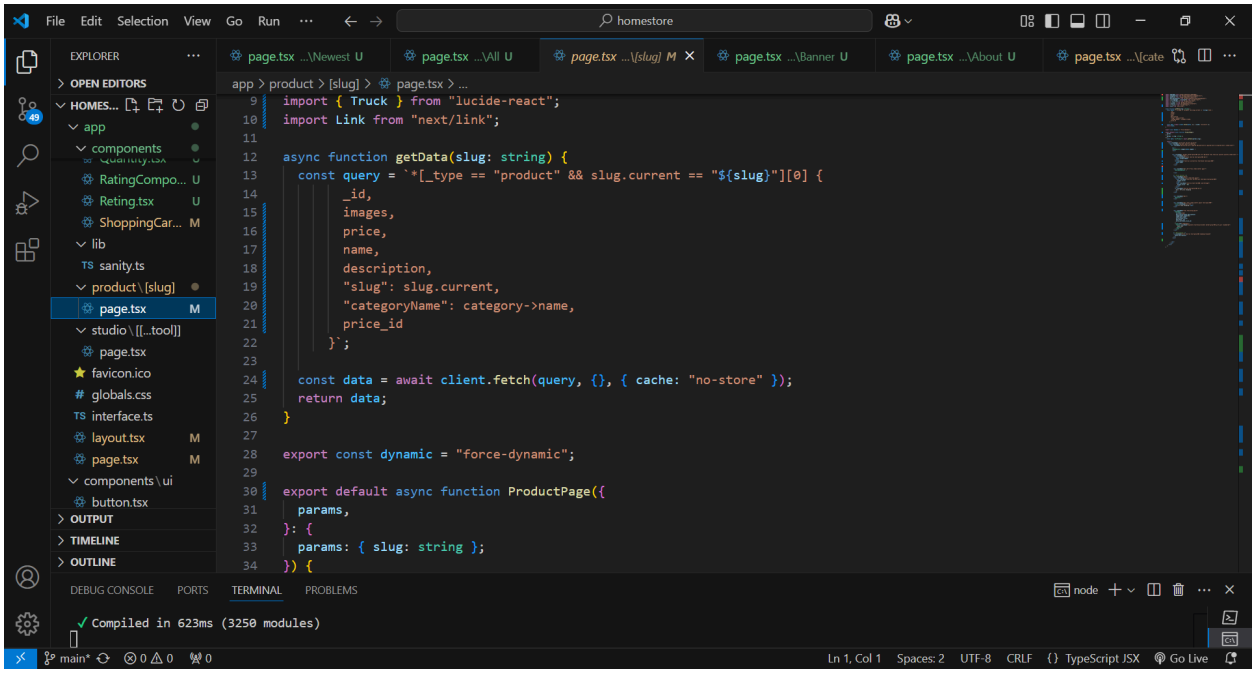
View product images and details.

Adjust the quantity of the product.

Check reviews and ratings.

Add the product to their cart or proceed directly to checkout.





Category-Based Product Listing Page:

This document explains the implementation of a dynamic category-based product listing page in a Next.js project. The page allows users to view

products filtered by category. Categories can be accessed via the navigation bar or category buttons.

Core Features

1. Dynamic Product Listing: Displays products based on the selected category.

2. Sanity Integration: Fetches products and their details from the Sanity CMS.

3. Reusable Components:

AddToBag: Adds products to the shopping bag.

Rating: Displays product ratings.

4. Responsive Design: Adapts to various screen sizes using TailwindCSS.

Code Explanation

1. Fetching Data from Sanity

The getData function fetches products belonging to a specific category using Sanity's GROQ query.

```
async function getData(cateogry: string) {
  const query = `*[_type == "product" && category->name == "${cateogry}"] {
      _id,
      "imageUrl": images[0].asset->url,
      price,
```

```
      name,
      "slug": slug.current,
      "categoryName": category->name
    }`;
  const data = await client.fetch(query, {}, { cache: "no-store" });
  return data;
}
```

2. Dynamic Route Handling

The CategoryPage is a dynamic route-based component that utilizes the params.category parameter to determine the category for which products are fetched and displayed.

3. UI Components

Header Section

Displays the category name dynamically:

```
<h2 className="text-2xl font-bold tracking-tight text-slate-900 font-serif">
  Our Products for {params.category}
</h2>
```

Product Card

Each product is rendered with the following details:

Image: Rendered using the Image component.

Name: Linked to the product's detail page.

Category: Shown below the name.

Price: Displayed alongside the product details.

Rating and Add to Bag: Included as reusable components.

4. Styling

The page uses TailwindCSS for a clean and responsive layout:

Grid layout for product listing:

Mobile-first approach with responsive utilities (sm:, lg:, etc.).

Dynamic Behavior

Dynamic Routing

The file path [category].tsx is used to dynamically fetch and display products based on the category in the URL.

For example:

/category/furniture displays furniture products.

/category/home-decor displays home decor products.

Dynamic Data Fetching

The dynamic export ensures fresh data is fetched whenever the page is loaded:

export const dynamic = "force-dynamic";

Dependencies

1. Sanity Client: For fetching data from the CMS.

2. Next.js: For server-side rendering and dynamic routing.

3. TailwindCSS: For styling and responsive design.

4. Reusable Components: AddToBag and Rating.

How to Use

1. Setup Sanity CMS:

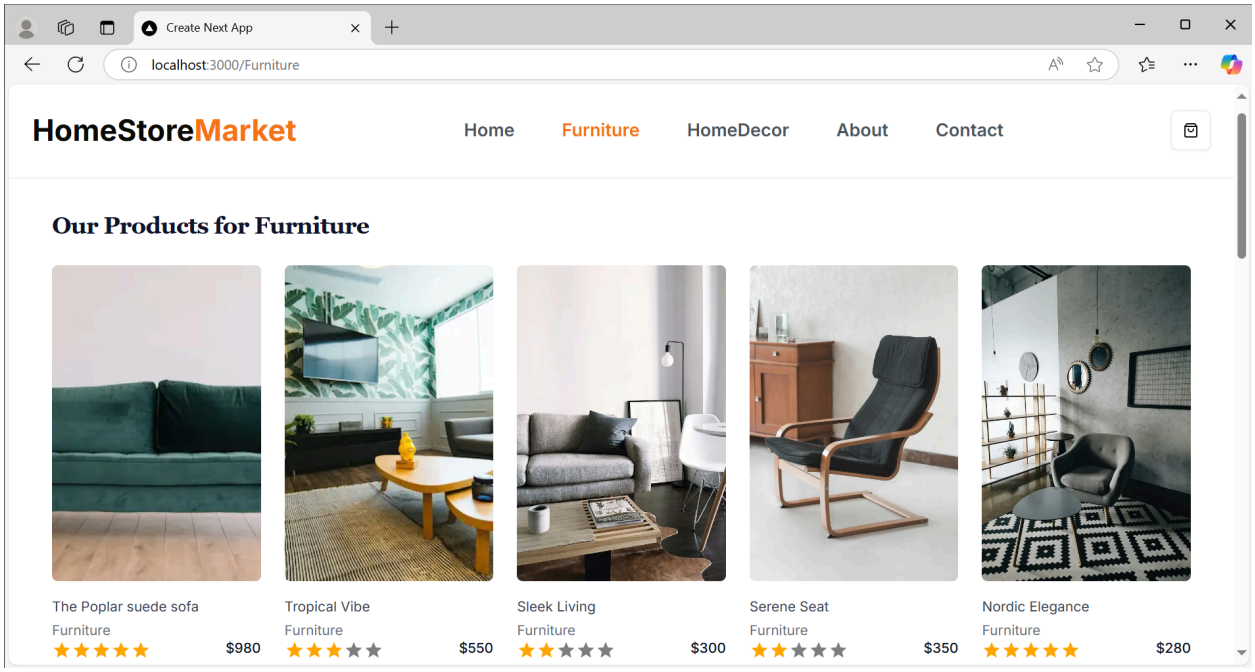
Ensure products are defined with a category field in the schema.

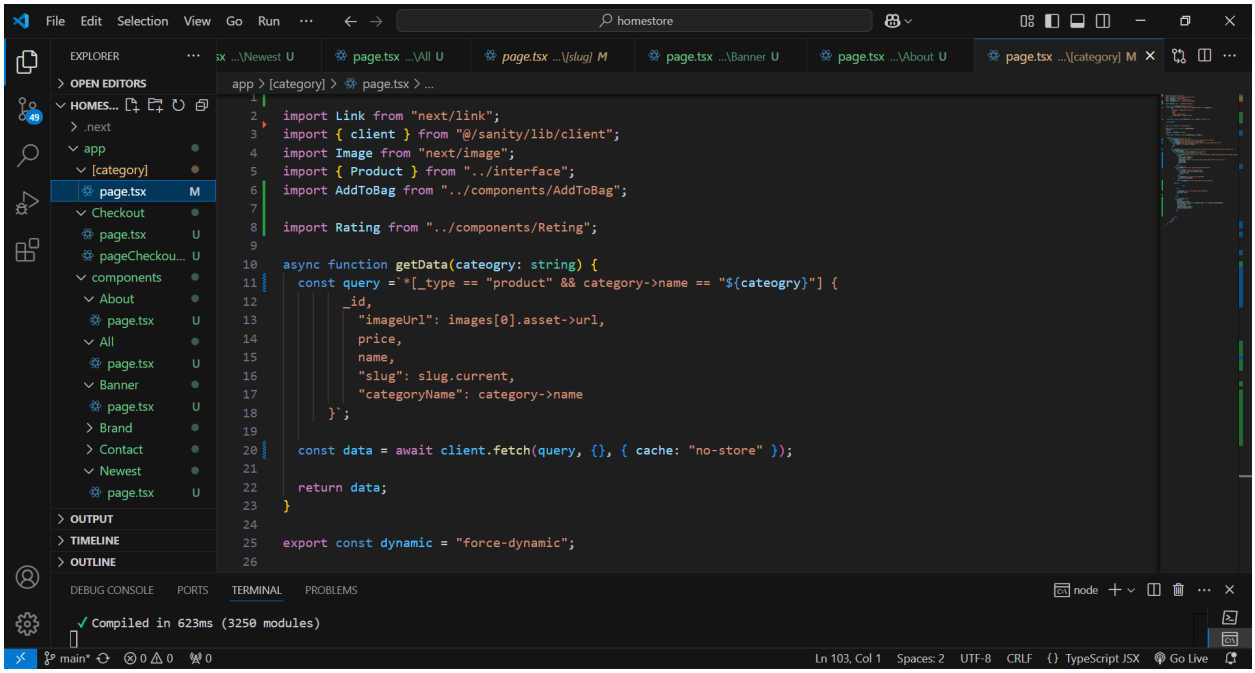
2. Dynamic Route:

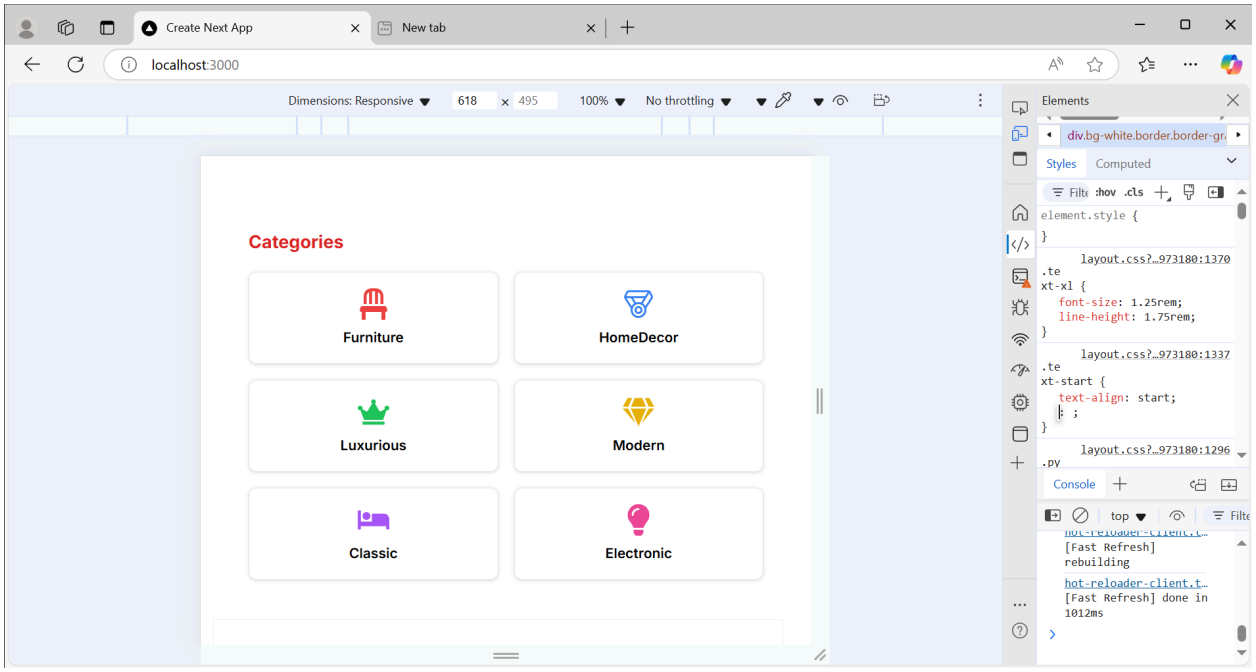
Add a dynamic [category].tsx file under the pages/category folder.

3. Integrate Components:

Import and use AddToBag and Rating components.


Create Next App
localhost:3000/Furniture
HomeStoreMarket
Home
Furniture
HomeDecor
About
Contact
Our Products for Furniture
The Poplar suede sofa
Furniture
$980
Tropical Vibe
Furniture
$550
Sleek Living
Furniture
$300
Serene Seat
Furniture
$350
Nordic Elegance
Furniture
$280


File Edit Selection View Go Run
homestore
EXPLORER
OPEN EDITORS
HOMES...
app > [category] > page.tsx > ...
import Link from "next/link";
import { client } from "@/sanity/lib/client";
import Image from "next/image";
import { Product } from "../interface";
import AddToBag from "../components/AddToBag";
import Rating from "../components/Reting";
async function getData(cateogry: string) {
const query =`*[_type == "product" && category->name == "${cateogry}"] {
_id,
"imageUrl": images[0].asset->url,
price,
name,
"slug": slug.current,
"categoryName": category->name
}`;
const data = await client.fetch(query, {}, { cache: "no-store" });
return data;
}
export const dynamic = "force-dynamic";
DEBUG CONSOLE
PORTS
TERMINAL
PROBLEMS
Compiled in 623ms (3250 modules)
Ln 103, Col 1
Spaces: 2
UTF-8
CRLF
TypeScript JSX
Go Live



Components(Add to Cart):

Documentation for Shopping Cart Functionality

This document outlines the functionality of the Shopping Cart Modal in my application, including features, usage, and behavior.

Overview

The Shopping Cart Modal provides the following functionalities:

1. Add to Cart: Users can quickly add products to the cart from the product listing.

2. View Cart: The cart displays selected products, quantities, prices, and total cost.

3. Remove Items: Users can remove products from the cart.
4. Total Calculation: Displays the subtotal cost of all products in the cart.

5. Checkout Option: Users can proceed to the checkout page.

6. Continue Shopping: Users can close the modal to return to shopping.

Code Features

Components and Dependencies

1. Third-party Libraries:

use-shopping-cart: Handles cart state and related functions.

next/link: For navigation to the checkout page.

next/image: Optimized image rendering.

2. UI Components:

Button: Custom button component.

Sheet, SheetContent, SheetHeader, SheetTitle: For displaying a modal-style cart.

Cart Functionalities

State Management:

cartCount: Total number of items in the cart.

cartDetails: Detailed information about each product in the cart.

totalPrice: Calculated subtotal of all cart items.

shouldDisplayCart: Toggles the visibility of the cart modal.

Actions:

handleCartClick: Opens or closes the cart modal.

removeItem(entry.id): Removes a product from the cart.

redirectToCheckout(): Redirects the user to the checkout page.

Features:

1. Add to Cart

Products are added to the cart dynamically.

Product details (name, price, quantity, image) are retrieved using cartDetails.

2. Cart Display

The modal (SheetContent) shows:

Product name, image, price, description, and quantity.

A "Remove" button for each product.

A "Subtotal" with the total price.

3. Remove Item

The removeItem(entry.id) function allows users to remove an individual product.

Cart updates dynamically to reflect changes.

4. Total Price

Subtotal (totalPrice) is displayed at the bottom of the cart modal.

Shipping and taxes are excluded from this calculation.

5. Checkout

The "Check Out" button navigates to the /Checkout page.

Checkout is triggered via the redirectToCheckout() function.

6. Continue Shopping

Users can close the cart modal using the "Continue Shopping" button.

Component Breakdown

Modal Header

Displays the title "Shopping Cart".

```
<SheetHeader>
  <SheetTitle>Shopping Cart</SheetTitle>
</SheetHeader>
```

Cart Items List

Each product displays:

Thumbnail (Image component).

Name and price.

Quantity.

Description.

Remove button.

Usage:

1. Import and use ShoppingCartModal in your main layout or navbar component.

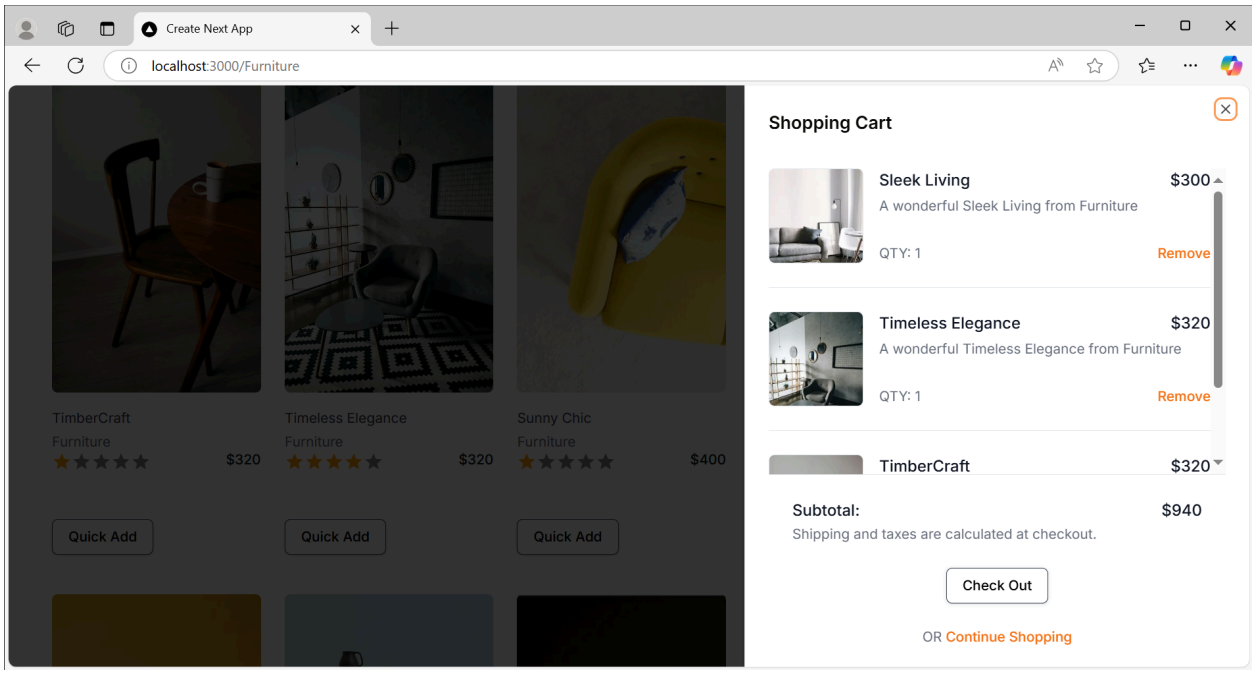
2. Ensure use-shopping-cart is properly configured in your project.

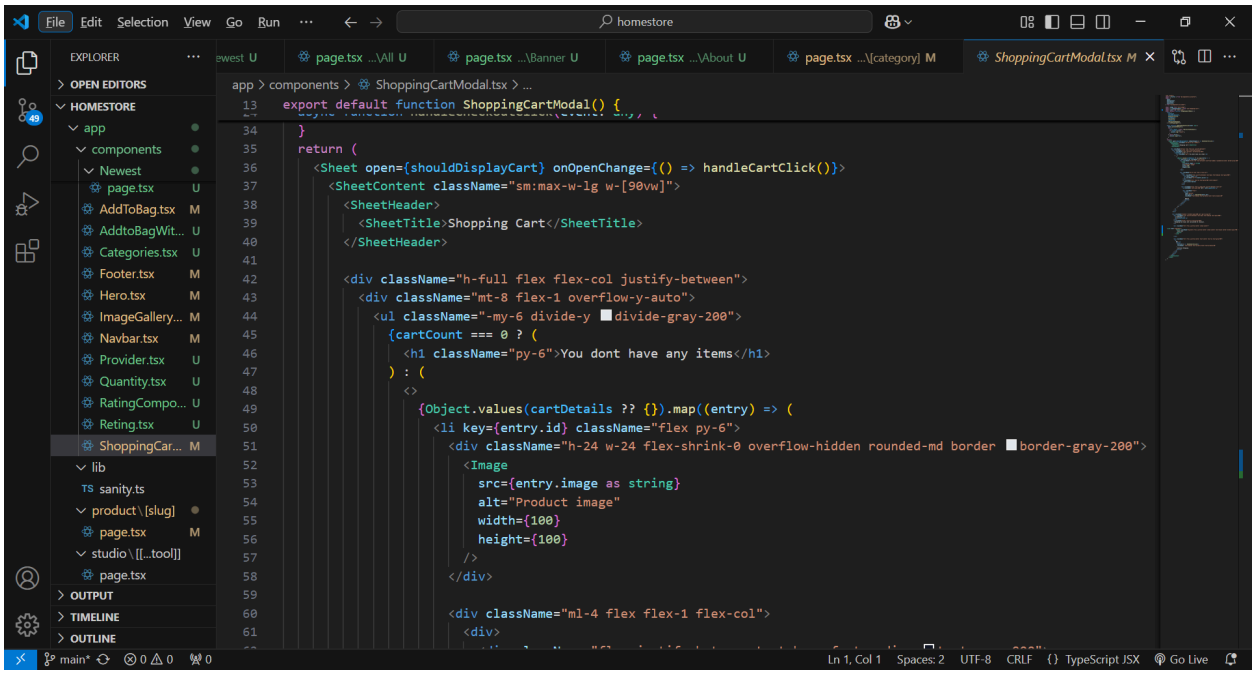
```
<ShoppingCartModal />
```

3. Add a button for quick product addition linked to your cart logic.

Styling:

TailwindCSS classes are used for responsiveness and consistent design.





CheckOut:

This documentation provides an overview of the functionality implemented in the Checkout component. The Checkout page allows users to review their cart items, provide their billing details, and place an order. Upon successful order placement, a confirmation message is displayed with options to navigate to shipping details or return to the previous page.

Features

1. User Billing Details Form:

Users must provide their name, email, phone number, address, city, and postal code.

The form validates that all fields are filled before enabling the "Complete Order" button.

2. Cart Summary:

Displays the list of products in the cart with their names and prices.

Shows the total price of all items in the cart.

3. Order Confirmation:

After the form is successfully submitted, the cart is cleared, and a "Thank You" message is displayed.

Users are provided with two buttons:

Go to Shipping: Redirects to the shipping details page (currently alerts as a placeholder).
Back: Navigates back to the previous page.

4. State Management:

Tracks form input values and validates them.

Manages order completion status and updates the UI accordingly.

Component Structure

Imports

React and useState for managing state.

useShoppingCart from use-shopping-cart for handling cart operations (e.g., clearing the cart and accessing cart details).

State Variables

1. formData: Stores user input for the billing details form.

2. isFormValid: Boolean indicating whether the form is valid.

3. orderCompleted: Boolean indicating whether the order has been successfully placed.

Key Functions

1. handleInputChange:

Updates the formData state when users input data into the form fields.

```
const handleInputChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const { name, value } = e.target;
  setFormData((prev) => ({
    ...prev,
    [name]: value,
  }));
};
```

2. validateForm:

Checks if all required fields in the form are filled and updates isFormValid.

```
const validateForm = () => {
  const isValid =
    !!formData.name && !!formData.email && !!formData.phone && !!formData.address;
  setIsFormValid(isValid);
};
```

3. handleSubmit:

Submits the form, clears the cart, and updates orderCompleted to show the confirmation message.

```
const handleSubmit = (e: React.FormEvent) => {
  e.preventDefault();

  if (isFormValid) {
    setOrderCompleted(true);
    clearCart();
  } else {
    alert("Please fill all the fields.");
  }
};
```

Cart Summary and Total Price

The cart summary iterates through the cartDetails object and displays each item's name and price. The total price is calculated by summing up the prices of all items.

```
{Object.keys(safeCartDetails).map((productKey) => {
  const product = safeCartDetails[productKey];
  return (
    <div key={product.id} className="flex justify-between text-sm">
      <span>{product.name}</span>
      <span>{$${product.price}}</span>
    </div>
  );
})}
```

Order Confirmation UI

When the order is completed:

A "Thank You" message is displayed.

Buttons are provided for navigating to the shipping page or going back to the previous page.

```
{orderCompleted ? (
  <div className="text-center">
    <h2 className="text-3xl font-serif font-semibold text-purple-950 mb-4">
      Thank You!
    </h2>
    <p className="text-green-800 mb-6 text-2xl">
      Your order is confirmed.
    </p>
    <div className="flex justify-center gap-4">
      <button
        onClick={() => alert("Go to Shipping")}
        className="px-6 py-3 text-black bg-white border border-pink-900 font-semibold rounded-md"
      >
        Go to Shipping
      </button>
      <button
```

```
        onClick={() => window.history.back()}
        className="px-6 py-3 bg-gray-500 text-white font-semibold rounded-md"
      >
        Back
      </button>
    </div>
  </div>
) : null}
```

Styling

TailwindCSS is used for responsive and visually appealing styling.

Buttons and input fields are styled with rounded corners and clear feedback for user interactions.

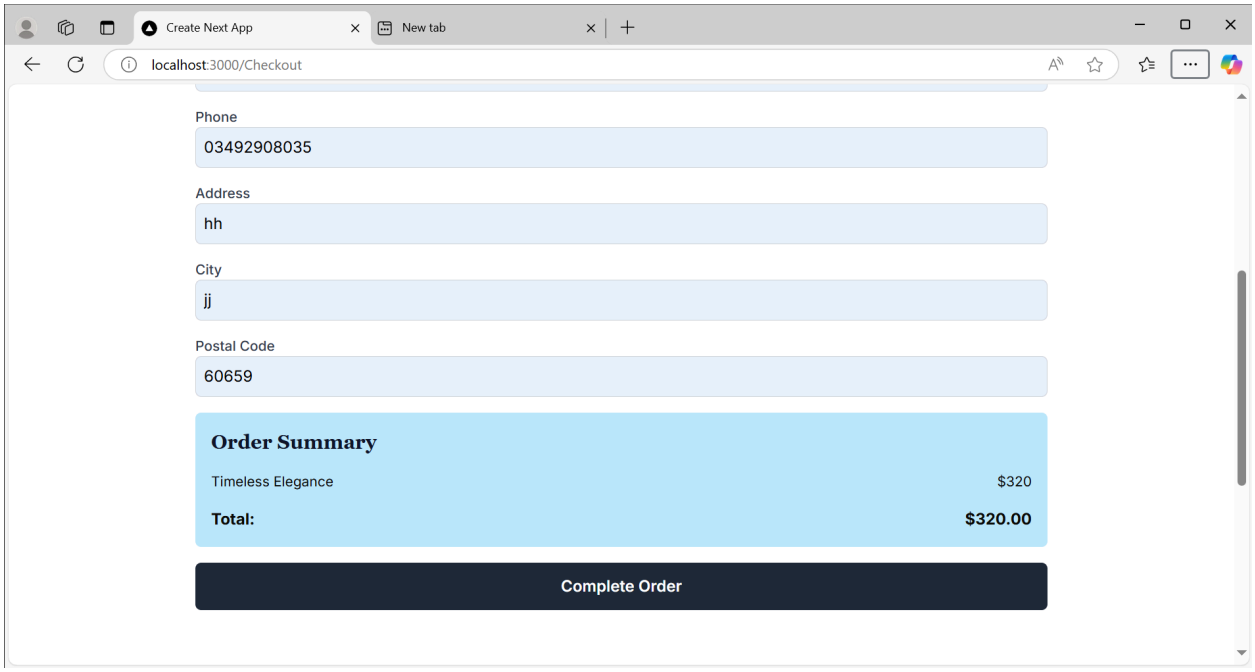
Improvements and Enhancements

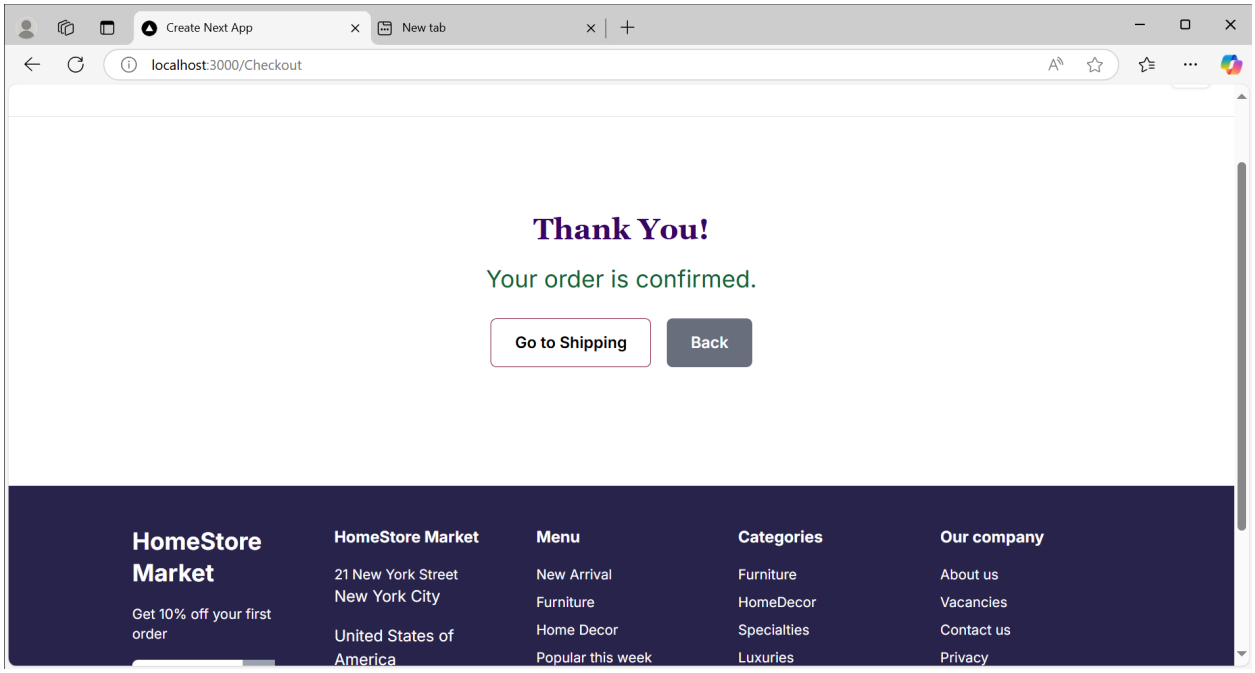
Shipping Page: Replace the alert placeholder with actual navigation logic.

Error Handling: Implement error messages for invalid input fields.

Loading Indicator: Add a spinner or loader when processing the order.

Payment Integration: Integrate a payment gateway for secure transactions.





Contact:

This document explains the code implementation of the "Contact Us" page, enabling users to send inquiries, questions, or feedback. The page

integrates a form for collecting user input and uses the EmailJS service to send messages. Here's an overview of the functionality and structure:

1. Purpose

The "Contact Us" page provides a simple and responsive interface where users can:

Enter their name, email, phone number, and message.

Submit their inquiry via email.

View contact information for alternative communication channels.

2. Features

1. User Input Form:

Fields for name, email, phone (optional), and message.

Input validation for required fields (name, email, and message).

2. Email Sending:

Utilizes EmailJS to send form data to a predefined email address.

Includes alerts for success and failure states.

3. Contact Information Display:

Includes phone number and email addresses for direct communication.
Displays social media links (Facebook and LinkedIn).

4. Styling:

Designed with TailwindCSS for a clean, responsive UI.

Includes hover and transition effects for buttons and social media icons.

3. Code Components

Imports

useState from React for form state management.

emailjs for sending email requests.

react-icons for including social and contact icons.

State Management

```
const [formData, setFormData] = useState({
  name: "",
  email: "",
  phone: "",
  message: "",
});
```

Maintains user input for the form fields.

Event Handlers

1. handleChange: Updates form data dynamically as the user types.

2. sendEmail: Handles form submission by:

Preventing default page refresh.

Sending form data using emailjs.send.

Displaying success or error alerts based on the result.

Resetting the form upon successful submission.

Form Submission Example

```
emailjs
  .send("service_id", "template_id", formData, "user_id")
  .then(
    () => alert("Message sent successfully!"),
    () => alert("Failed to send the message. Please try again.")
  );
```

Responsive Layout

The page uses TailwindCSS classes for:

Grid Layout: Aligns the form and contact details side-by-side on larger screens (md:grid-cols-2).

Borders and Padding: Adds styling to separate sections.

Hover Effects: Enhances user experience for buttons and icons.
4. Key Sections

1. Header:

Displays a message encouraging users to contact support.

Includes a page title styled with a serif font.

2. Contact Information:

Phone and email details with relevant icons for visual clarity.

Social media links styled with hover and scale effects.

3. User Form:
Includes name, email, phone, and message fields.

Button with hover state for submitting the form.

4. Footer:

Displays placeholder information for the website owner.

5. Dependencies

React: For component creation and state management.

EmailJS: For email functionality.

React Icons: For adding visual elements.

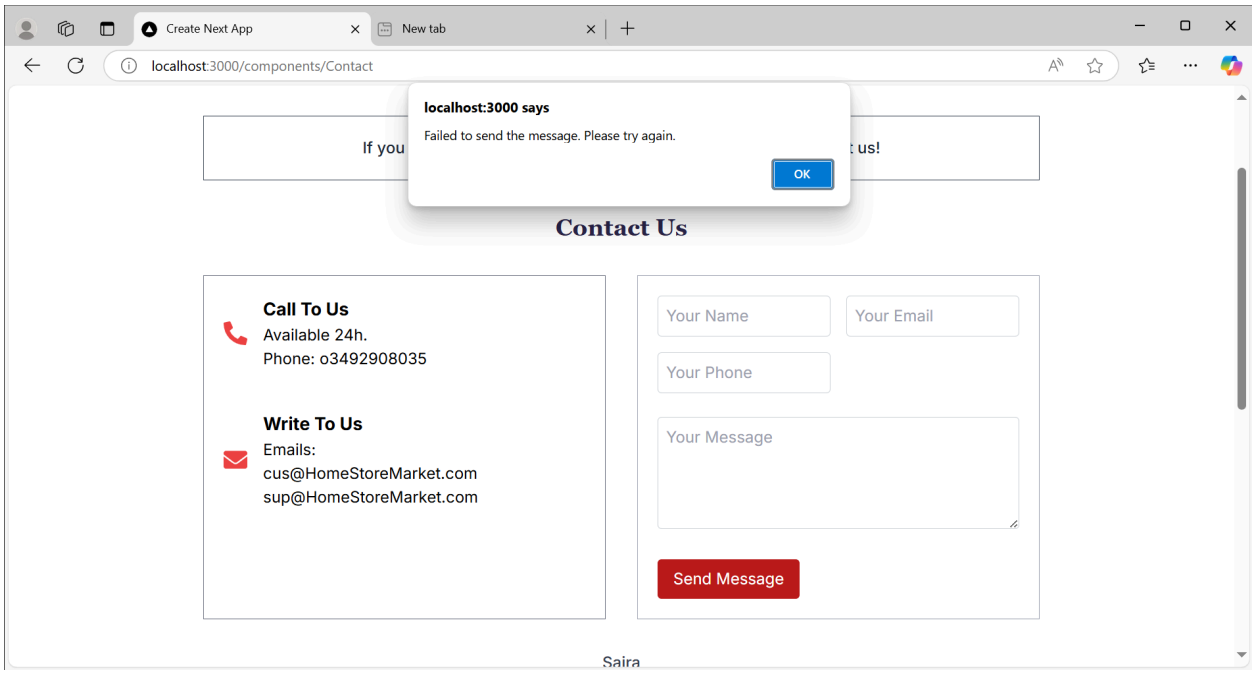
TailwindCSS: For styling the page responsively and efficiently.

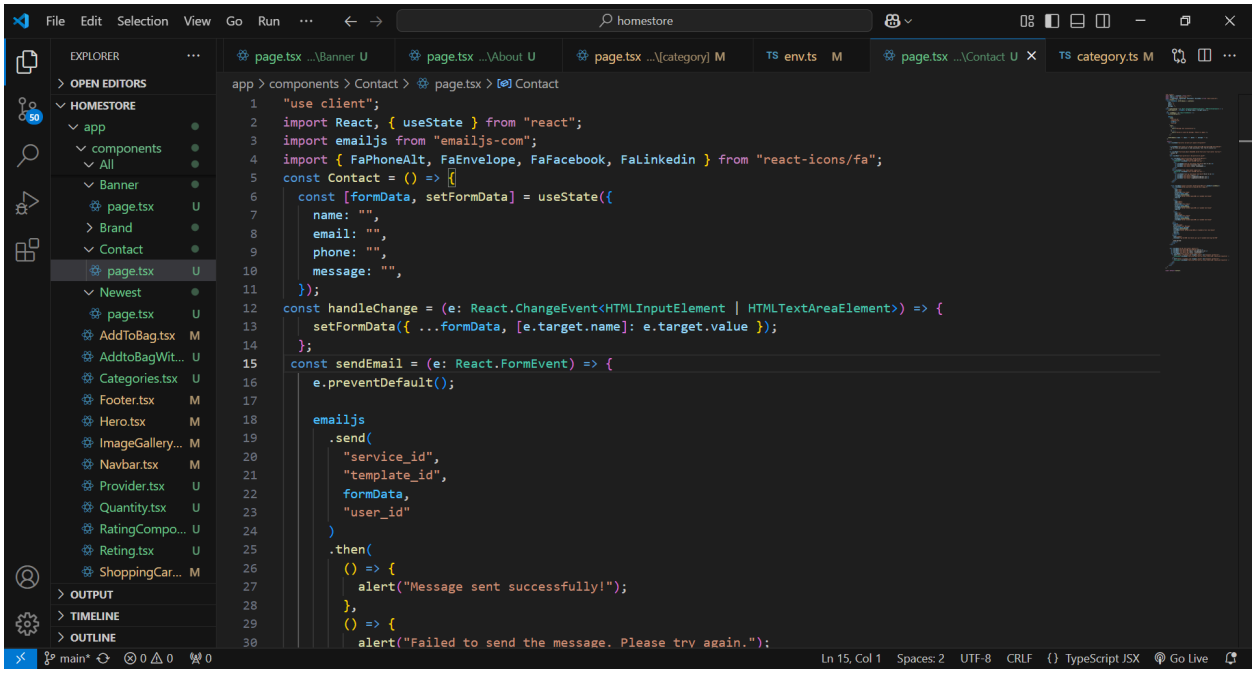
6. Improvements & Customization

Error Handling: Enhance alerts by displaying errors below the form fields.

Accessibility: Add ARIA labels and focus indicators for improved accessibility.

Validation: Use a validation library like yup for robust input validation.





About Page Implementation:

This About page provides detailed insights about the company, including its story, achievements, team members, and core features. The page is designed using Next.js with TailwindCSS for responsive and visually appealing UI and uses SVG icons for enriched content representation.

File Name:

about.tsx

Features

1. Company Overview

Section Name: "Our Story"

Includes a descriptive introduction about the company, its launch year, mission, and statistics.

2. Company Statistics

Displays key metrics such as:

Active sellers

Monthly product sales

Active customers

Annual gross sales

Each metric is represented with an icon and relevant data.

3. Team Members

A grid layout showcasing the core team, including their names, roles, and profile pictures.

Includes links to their social media profiles (Facebook, Twitter, LinkedIn).

4. Core Features

Highlights essential services such as:

Free and fast delivery

24/7 customer service

Money-back guarantee

Each feature is paired with an illustrative icon.

Code Structure

1. Imports

Image: For optimized image handling in Next.js.

Icons (FaTruck, FaPhone, FaMoneyBillAlt, etc.): For visually representing features and metrics.

2. Layout

Main Container: main with a bg-gray-50 background for the entire page.

Sections:

Each section is wrapped with appropriate TailwindCSS utilities (py-12, max-w-6xl, etc.) for responsiveness and spacing.

Statistics Grid:

Dynamically generated using map().

Tailwind classes for styling individual statistic cards.

Team Members Grid:

Dynamically generated using map().

Includes dynamic links to social profiles.

Features Grid:

Dynamically generated using map().

Illustrates core features of the company.

Dependencies:

Next.js: Required for the Image component and project routing.

React Icons: For including prebuilt scalable vector icons.

TailwindCSS: For utility-first styling.

Customization Options

1. Dynamic Data Fetching:

Replace static data for team members, statistics, and features with data fetched from APIs or CMS (e.g., Sanity CMS).

2. Styling Enhancements:

Customize TailwindCSS classes to match branding requirements.

3. Content Updates:

Modify descriptive texts and images to reflect updated company details.

Usage Instructions

1. Save the File: Place this component in the /app or /pages directory as about.tsx.

2. Add Route: Ensure that the pages or app directory includes navigation to /about in the website's header or menu.

3. Images Path: Add images like about1.webp, men1.png, etc., in the /public directory to match the paths used in the component.

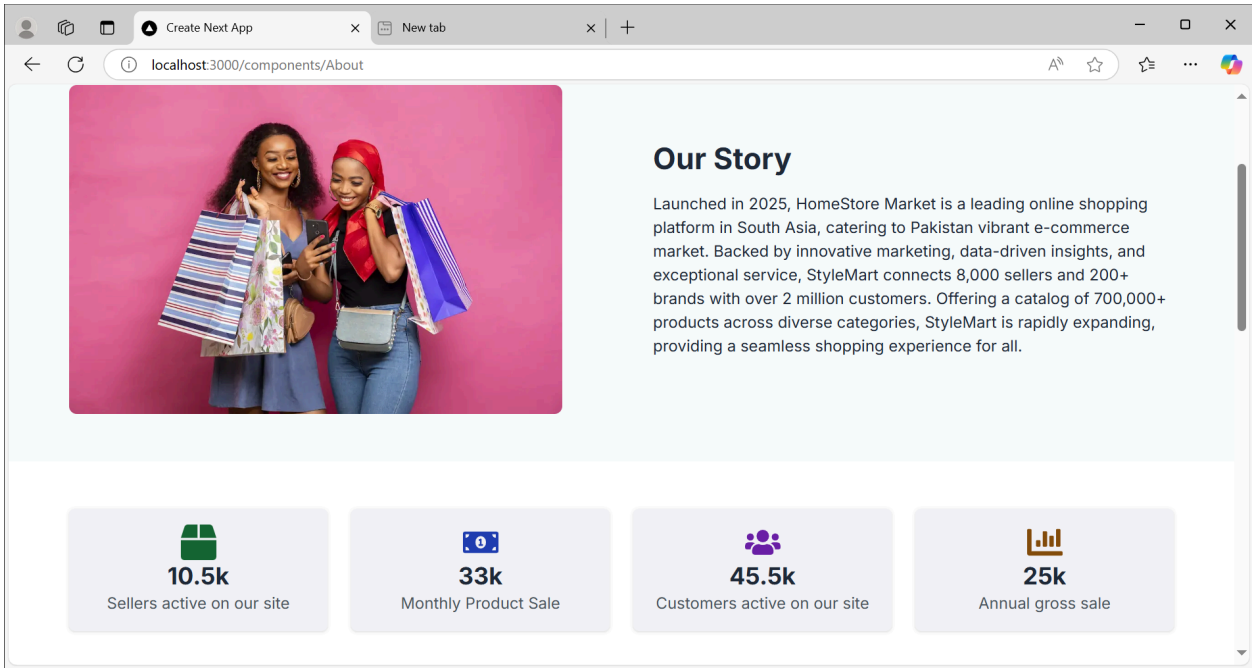
4. Icon Styling: Update or replace icons as per requirements by changing the imported react-icons.

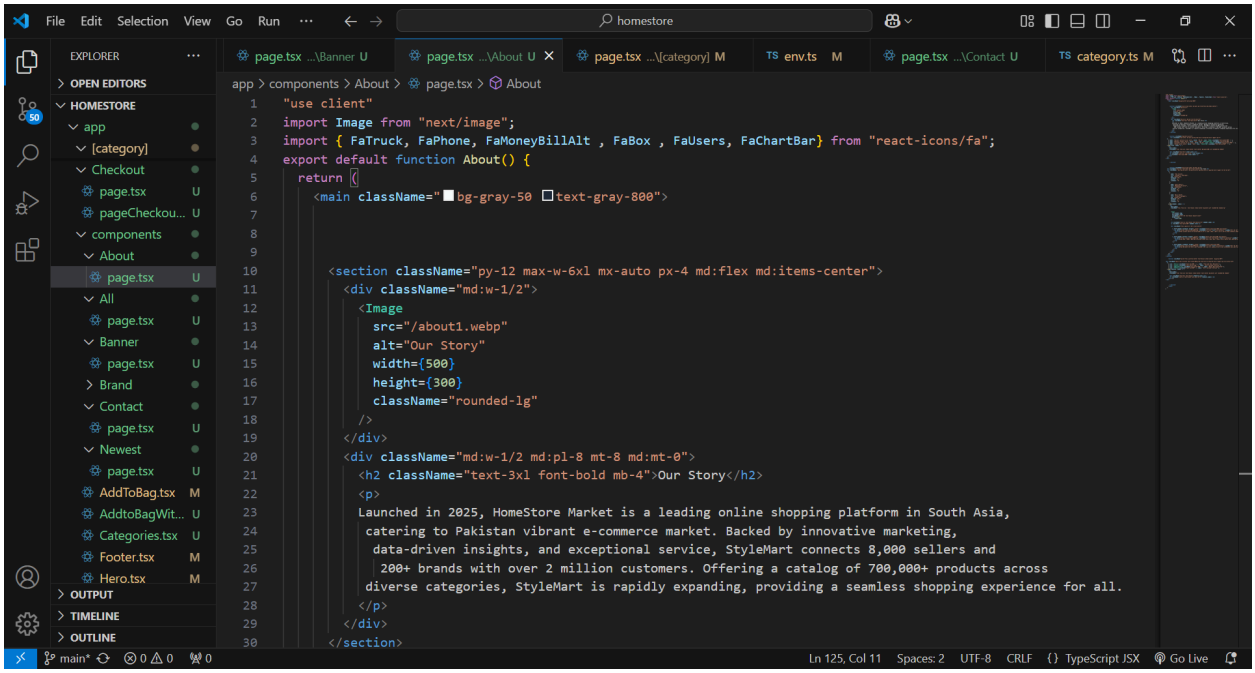
Enhancements

Lazy Loading: Implement lazy loading for images to enhance page performance.

SEO: Add meta tags and alt attributes for better search engine optimization.

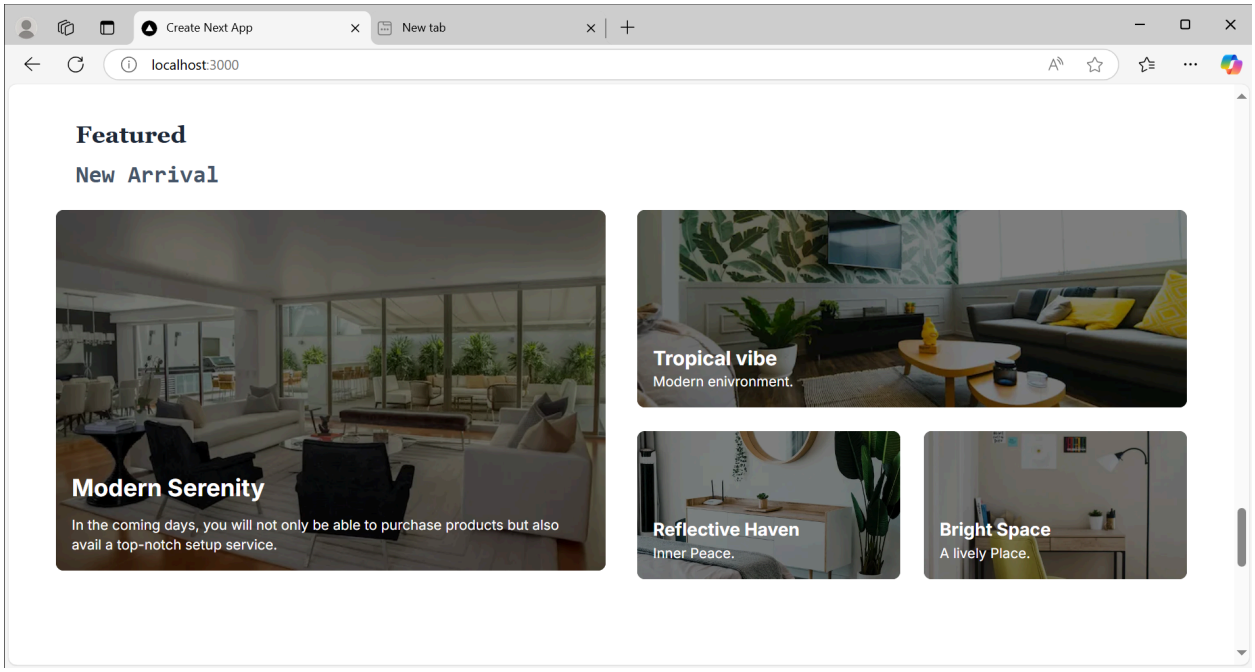
Accessibility: Use ARIA roles and labels for better accessibility.

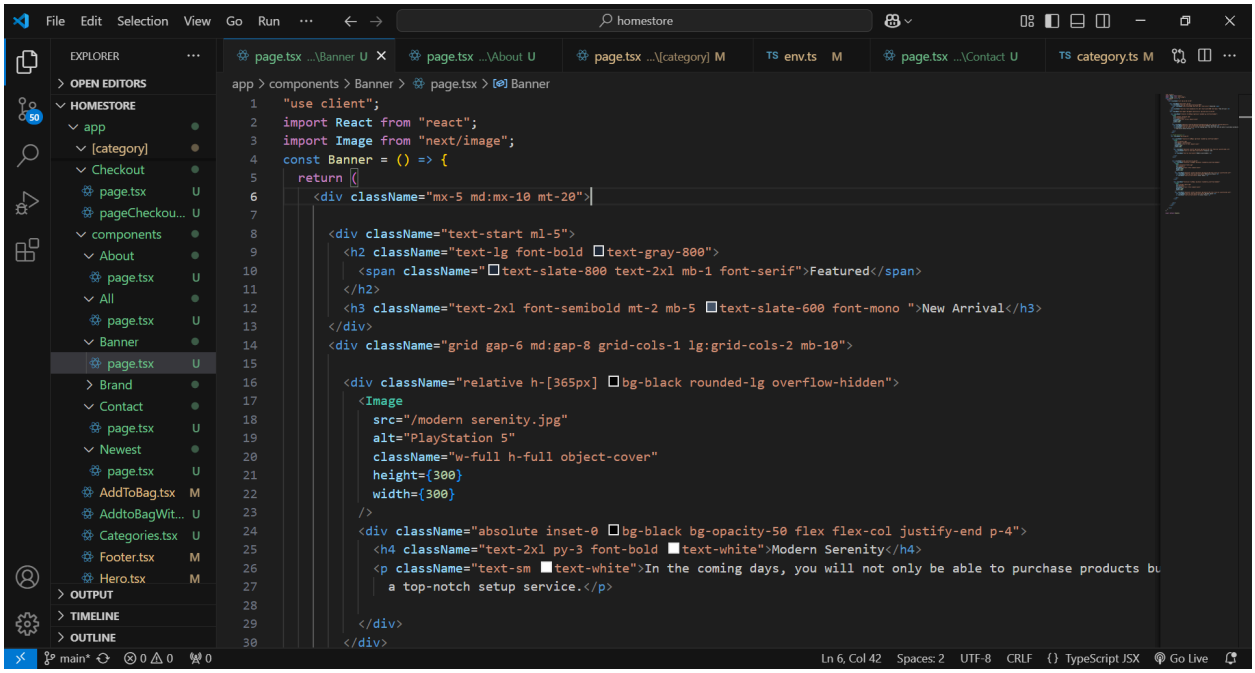




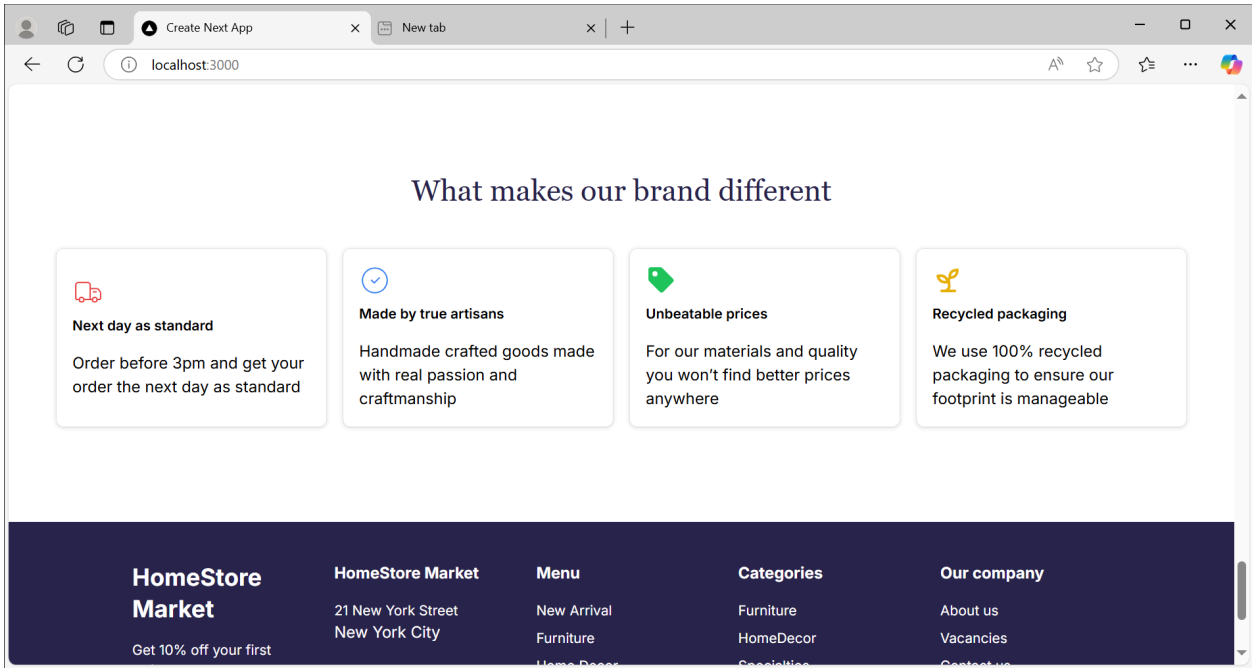
Hero Section:

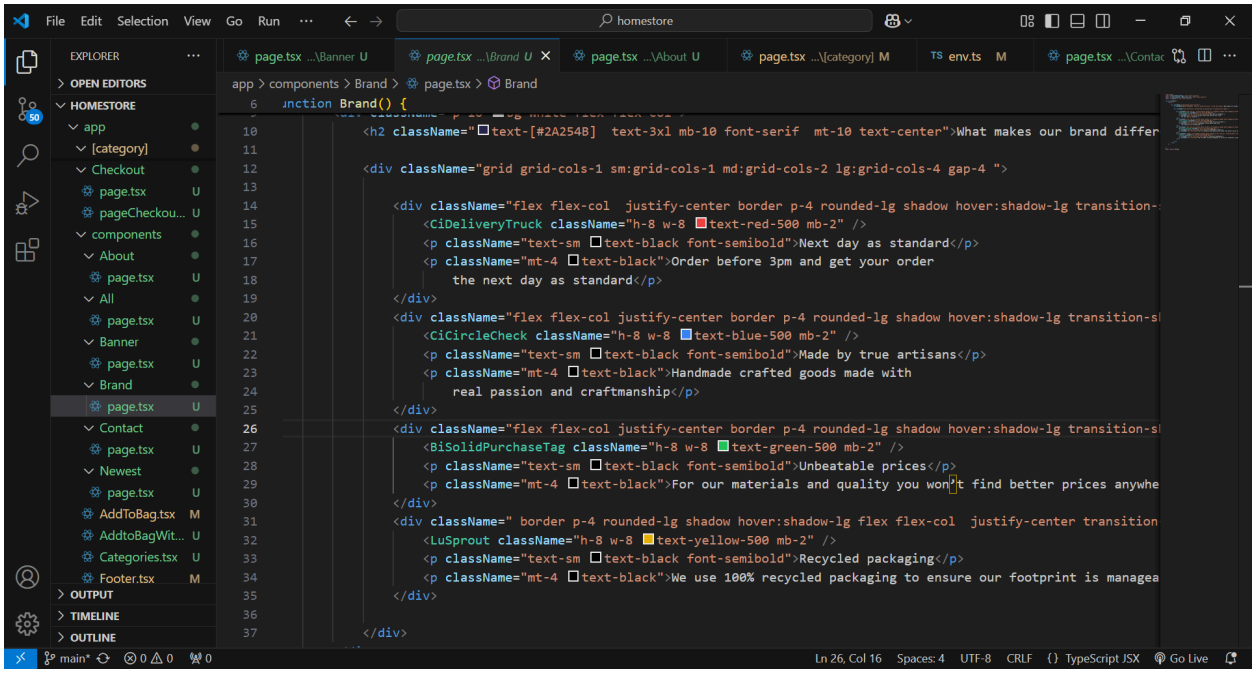
New Arrival:



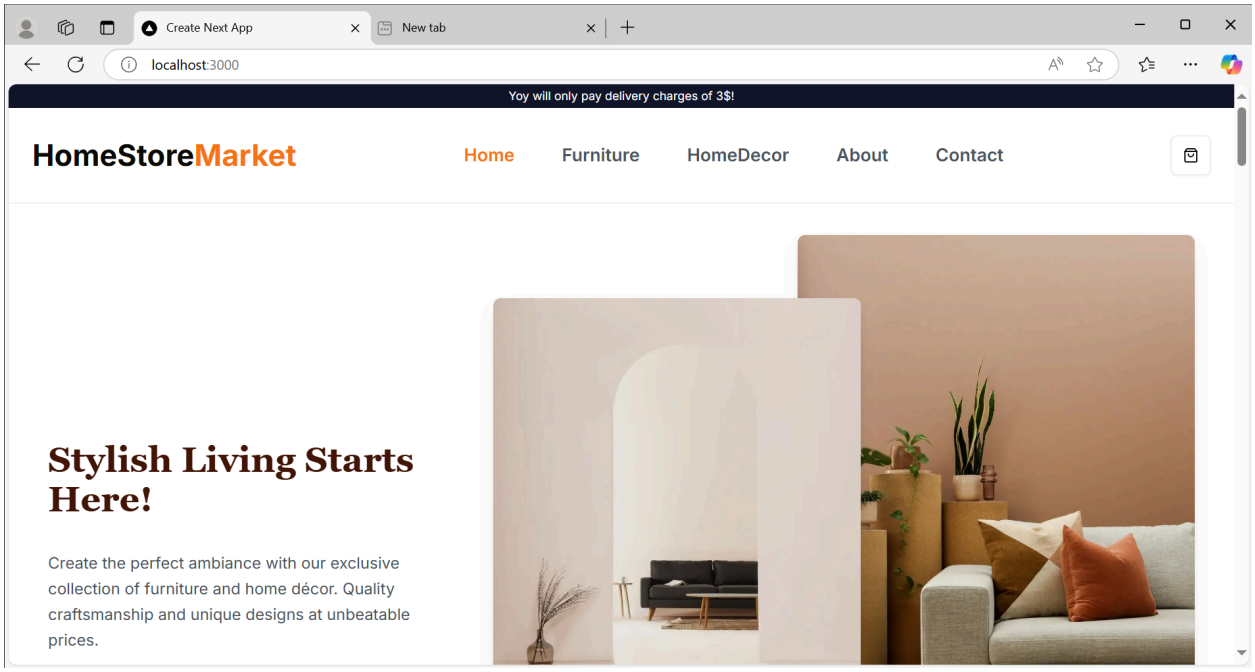


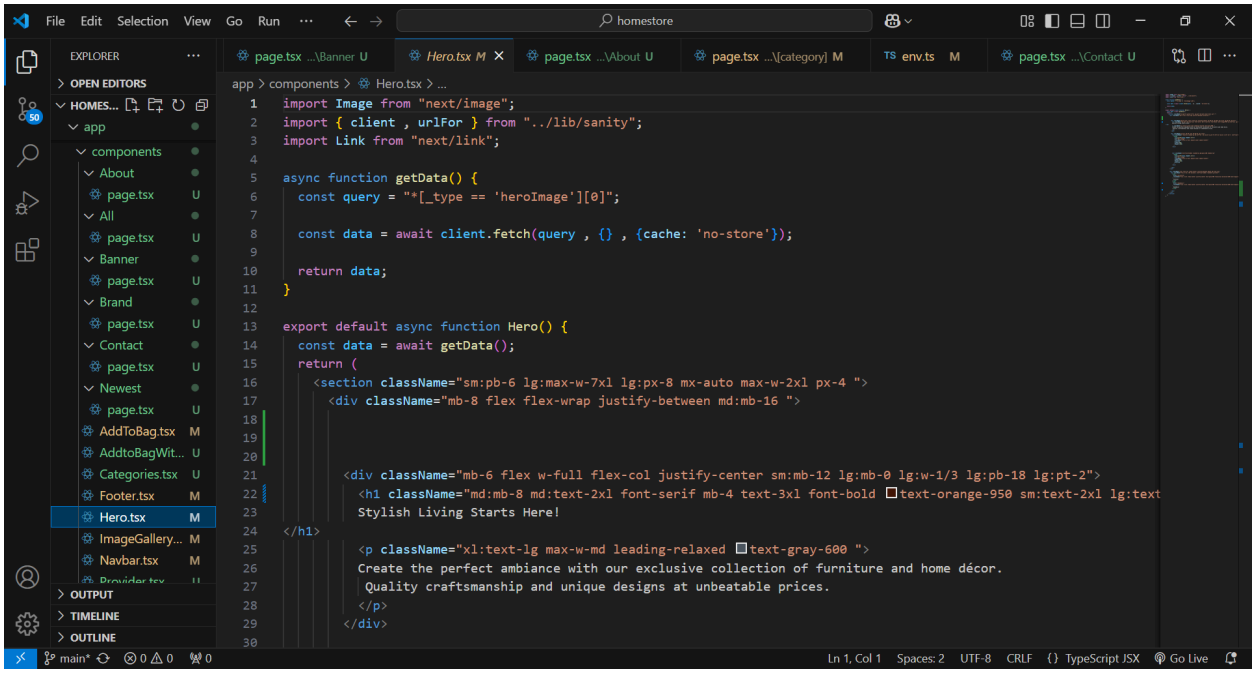
Show brand difference


Create Next App
New tab
localhost:3000
What makes our brand different
Next day as standard
Order before 3pm and get your order the next day as standard
Made by true artisans
Handmade crafted goods made with real passion and craftmanship
Unbeatable prices
For our materials and quality you won't find better prices anywhere
Recycled packaging
We use 100% recycled packaging to ensure our footprint is manageable
HomeStore Market
Get 10% off your first
HomeStore Market
21 New York Street
New York City
Menu
New Arrival
Furniture
Categories
Furniture
HomeDecor
Our company
About us
Vacancies


File Edit Selection View Go Run
homestore
EXPLORER
OPEN EDITORS
HOMESTORE
app
[category]
Checkout
page.tsx
pageCheckou...
components
About
All
Banner
Brand
Contact
Newest
AddToBag.tsx
AddtoBagWit...
Categories.tsx
Footer.tsx
OUTPUT
TIMELINE
OUTLINE
page.tsx ...\Banner U
page.tsx ...\Brand U
page.tsx ...\About U
page.tsx ...\[category] M
env.ts M
app > components > Brand > page.tsx > Brand
<h2 className="text-[#2A254B] text-3xl mb-10 font-serif mt-10 text-center">What makes our brand differ

<div className="flex flex-col justify-center border p-4 rounded-lg shadow hover:shadow-lg transition-
<CiDeliveryTruck className="h-8 w-8 text-red-500 mb-2" />
Next day as standard
Order before 3pm and get your order
the next day as standard
<div className="flex flex-col justify-center border p-4 rounded-lg shadow hover:shadow-lg transition-s
<CiCircleCheck className="h-8 w-8 text-blue-500 mb-2" />
Made by true artisans
Handmade crafted goods made with
real passion and craftmanship
<div className="flex flex-col justify-center border p-4 rounded-lg shadow hover:shadow-lg transition-s
<BiSolidPurchaseTag className="h-8 w-8 text-green-500 mb-2" />
Unbeatable prices
For our materials and quality you won't find better prices anywhe
<div className=" border p-4 rounded-lg shadow hover:shadow-lg flex flex-col justify-center transition
<LuSprout className="h-8 w-8 text-yellow-500 mb-2" />
Recycled packaging
We use 100% recycled packaging to ensure our footprint is manage





Responsive Navbar:

This navbar component is a responsive navigation bar designed for a dynamic e-commerce website. It is optimized for different screen sizes

(mobile, medium, and large devices) and includes key features like navigation links, a shopping cart button, and a collapsible mobile menu.

Code Structure

The component is implemented in React using the following libraries:

Next.js: For routing and navigation.

TailwindCSS: For responsive styling.

React Icons: For menu and close icons.

use-shopping-cart: For shopping cart management.

Key Features

1. Responsive Design:

On large devices (LG): A horizontal navigation bar with visible links.

On mobile devices: A collapsible menu triggered by a hamburger icon.

2. Dynamic Active Link Highlight:

The currently active page is highlighted.

3. Shopping Cart Integration:

Includes a button that triggers the shopping cart modal using use-shopping-cart.

4. Collapsible Menu:

For mobile screens, a collapsible sidebar menu is implemented.

5. Customizable Links:

Links array (links) is dynamically mapped to generate navigation items.

3. Navbar Structure

Header Section:

A top banner with a promotional message.

Main Section:

Includes the site logo, navigation links, and action buttons (e.g., shopping cart and menu toggle).

Mobile Menu:

Appears as a sidebar overlay when toggled.

Breakpoints

Mobile (default):

Displays a hamburger menu (FiMenu) for navigation.

The menu expands into a left-side drawer when opened.

Medium (MD) and Large (LG):

Displays a horizontal navigation bar with all links visible.

The hamburger menu is hidden.

Styling

TailwindCSS classes are used for styling and responsiveness:

hidden lg:flex: Hides elements on mobile and shows them on large screens.

text-gray-600 hover:text-primary: Sets default link color and hover effect.

bg-slate-900: Background color for the top header.

flex justify-between: Aligns elements in a row with spacing.

Usage

1. Import the Component:

import Navbar from "@/components/Navbar";

2. Place in Layout:

```
export default function Layout({ children }) {
  return (
    <>
      <Navbar />
      <main>{children}</main>
    </>
  );
}
```

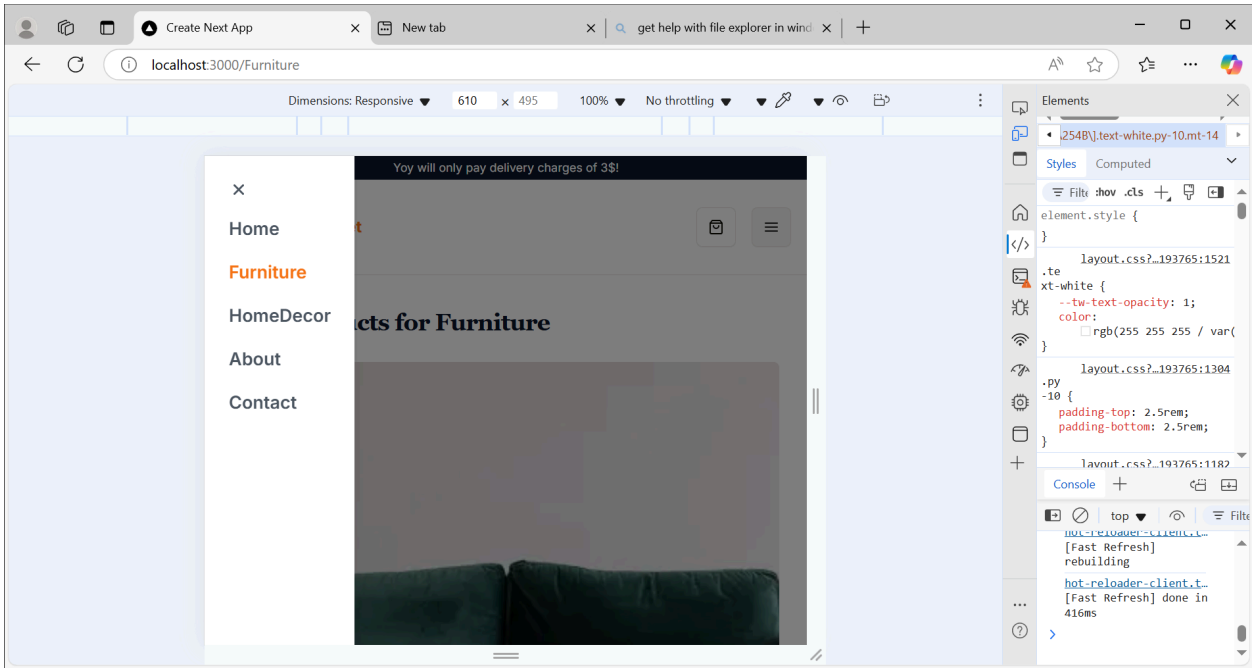
3. Customize Links: Modify the links array to update the navigation links:

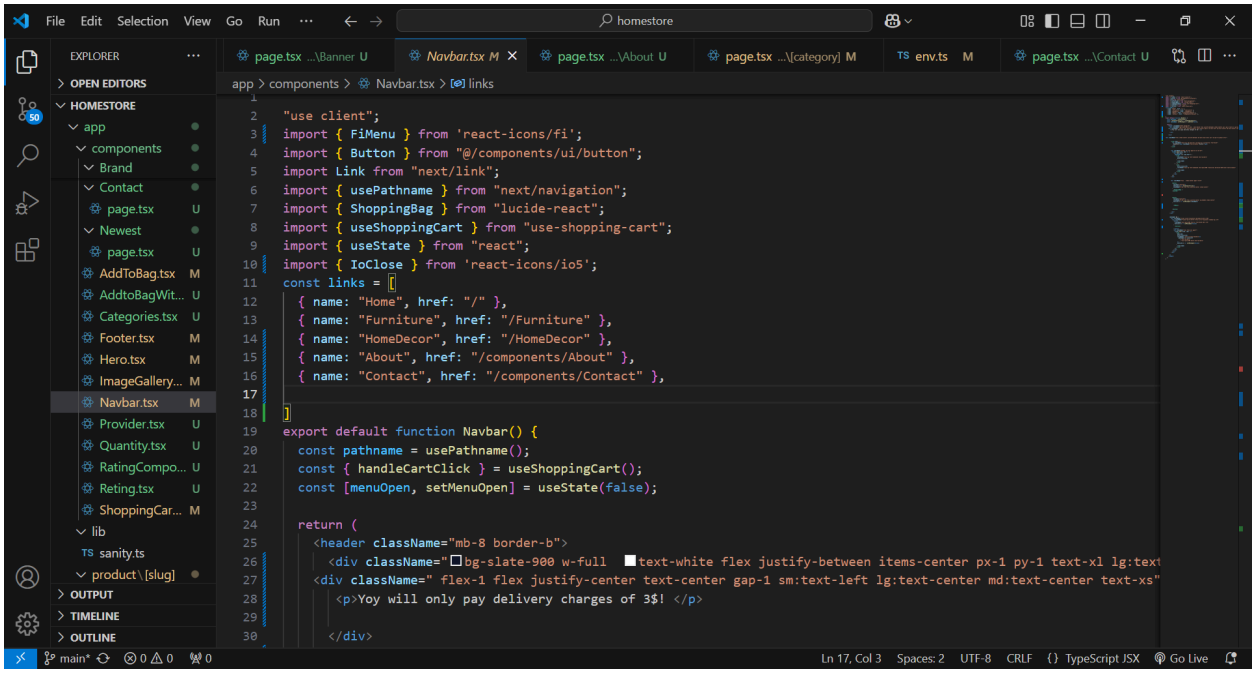
```
const links = [
  { name: "Home", href: "/" },
  { name: "Furniture", href: "/Furniture" },
  { name: "HomeDecor", href: "/HomeDecor" },
  { name: "About", href: "/components/About" },
  { name: "Contact", href: "/components/Contact" },
];
```

Known Issues

1. Shopping Cart Functionality: Ensure use-shopping-cart is correctly configured in your project.

2. Responsive Design: Test on various screen sizes to ensure proper alignment.





Footer Component:

This documentation explains the structure and functionality of the Footer component built in React with Next.js and TailwindCSS. The component serves as the footer for the website and includes various sections like

subscription forms, company information, menus, categories, and social media links.

Component: Footer

File Location: Footer.tsx

Import Requirements:

```
"use client";
import { FaFacebook, FaInstagram, FaLinkedin } from "react-icons/fa";
```

Props

This component does not accept any props.

Structure

The Footer component is divided into the following sections:

1. Main Wrapper

Class Name: bg-[#2A254B] text-white py-10 mt-14

Applies background color, text color, and spacing.

2. Grid Layout

Container Class: container mx-auto grid grid-cols-2 md:grid-cols-5 gap-8 px-4

Responsive grid layout for five sections, adjusting to two columns on smaller screens.

Sections

1. Subscription Section

Purpose: Collect user email addresses for newsletters or promotional offers.

Classes:

Header: font-bold text-2xl sm:text-0.5xl mb-4

Input Field: p-2 w-28 h-10 text-sm text-black rounded-l-md focus:outline-none

Submit Button: bg-gray-400 p-2 hover:bg-blue-500 transition

Behavior:

Users enter an email in the input field.

On clicking the submit button, the form can be connected to a backend for processing.

2. Company Information

Purpose: Display the company's address and provide quick access to social media.

Classes:

Header: font-bold text-1xl mb-4

Address: text-sm

Social Media Links: flex space-x-4 mt-2

Icons:

Facebook (FaFacebook)

LinkedIn (FaLinkedin)

Instagram (FaInstagram)

Behavior:

Clicking on icons redirects to respective social media pages.

3. Menu

Purpose: Provide quick navigation to key website sections.

Classes:

Header: font-bold text-1xl mb-4

List Items: text-sm hover:underline mb-2

4. Categories

Purpose: Navigate through product categories like Furniture, Home Decor, etc.

Classes:

Header: font-bold text-1xl mb-4

List Items: text-sm hover:underline mb-2

Links:

Example: /Furniture, /HomeDecor, /components/Brand, etc.

5. Company Details

Purpose: Provide links for about us, vacancies, contact, and privacy.

Classes:

Header: font-bold text-1xl mb-4

List Items: text-sm hover:underline mb-2

Links:

Example: /components/About (About us), /components/Contact (Contact us), etc.

Footer Bottom Section

Purpose: Display copyright information.

Classes:

Border: border-t border-gray-700 mt-8 pt-4

Text: text-center text-lg text-gray-400

Content: © Copyright 2025.LTD

Responsive Design

The component adjusts seamlessly for different screen sizes using TailwindCSS utilities:

Mobile: Two-column layout.

Desktop: Five-column layout with a gap between sections.

Styling

TailwindCSS is used for styling, making it consistent and responsive.

Colors:

Background: [#2A254B]

Text: white with hover effects.

Typography:

Headers: Bold with varying font sizes.

Links: Underlined on hover.

Icons

The component uses icons from the react-icons library:

FaFacebook

FaInstagram

FaLinkedin

To install:

npm install react-icons

How to Use

1. Import and include the Footer component in your Next.js layout:

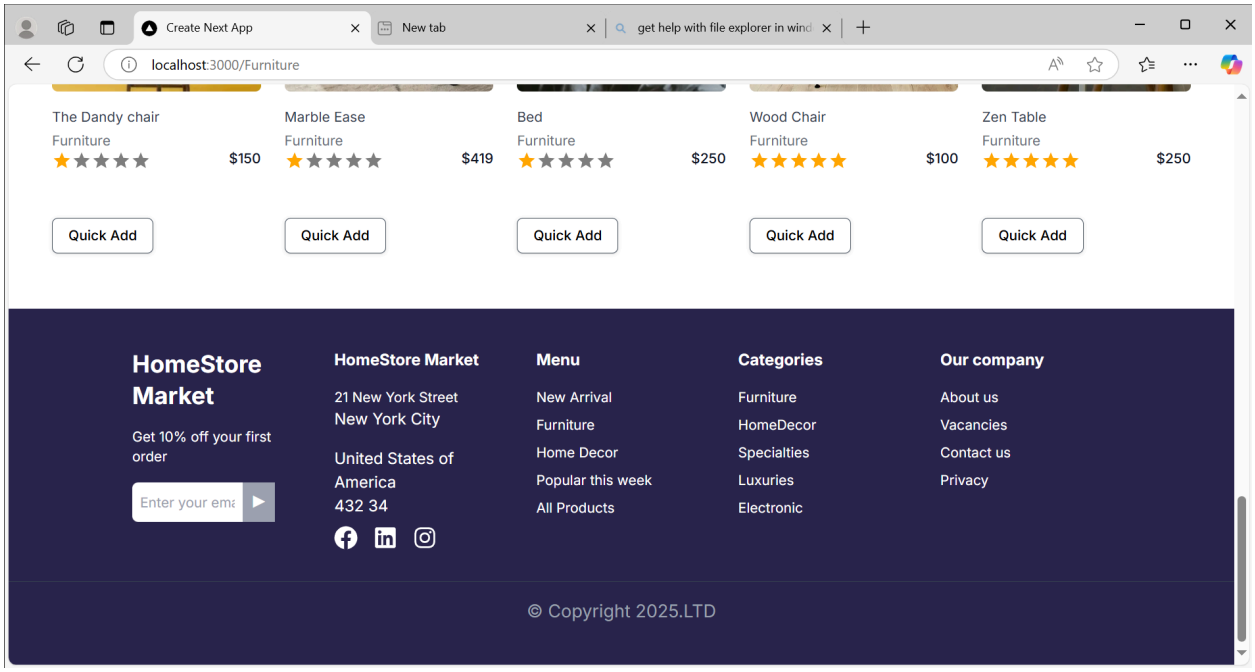
```
import Footer from './components/Footer';

export default function Home() {
   return (
      <>
         <main>/* Main Content */</main>
         <Footer />
      </>
   );
}
```

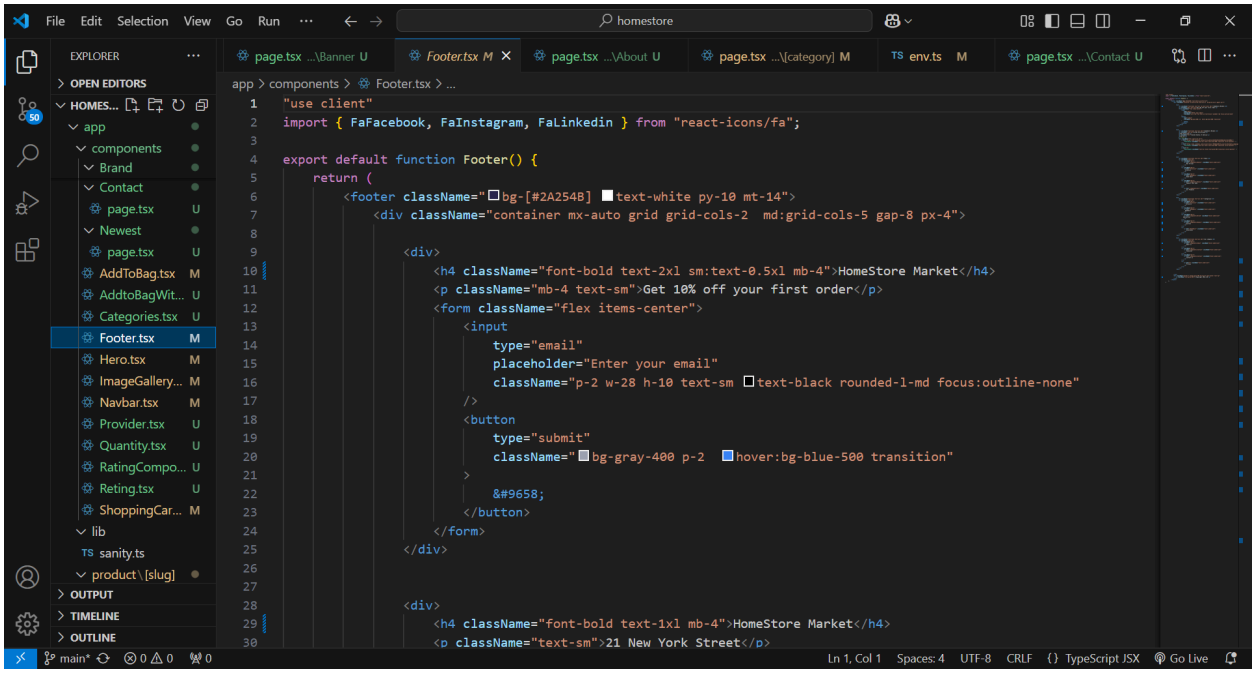
2. Customize as per project requirements (e.g., update links and social media profiles).

Future Enhancements

Add functionality to the subscription form for email validation and backend integration.

Integrate dynamic content for menu and categories using APIs or CMS like Sanity.





Project Summary: Functional Product Listing and Dynamic Features in E-Commerce Platform

I successfully implemented key features for a functional e-commerce platform using Next.js and Sanity CMS. The following tasks were completed to ensure a dynamic and responsive user experience:

1. Product Listing Page:

Developed a fully functional product listing page that dynamically fetches and displays product data from Sanity CMS.

Incorporated real-time data fetching and efficient rendering for optimal performance.

2. Dynamic Product Detail Pages:

Implemented individual product detail pages using dynamic routing based on product slugs.

Displayed comprehensive product information, including images, prices, tags, and related products.

3. Advanced Category Filters:

Built robust category filters to refine product views dynamically.

Ensured seamless interaction by allowing users to filter products based on categories, price range, and other parameters.

4. Pagination and Related Products:

Added pagination to improve user navigation for large product inventories.

Designed a "Related Products" section on product detail pages for cross-selling opportunities.

5. Responsive and Styled Components:

Ensured all components are fully responsive and styled with TailwindCSS to maintain a professional and modern appearance across devices.

6. Modular and Reusable Components:

Created modular and reusable components to facilitate scalability and streamline future development efforts.

This implementation demonstrates the ability to design and develop a robust e-commerce frontend, delivering a dynamic and user-friendly interface.