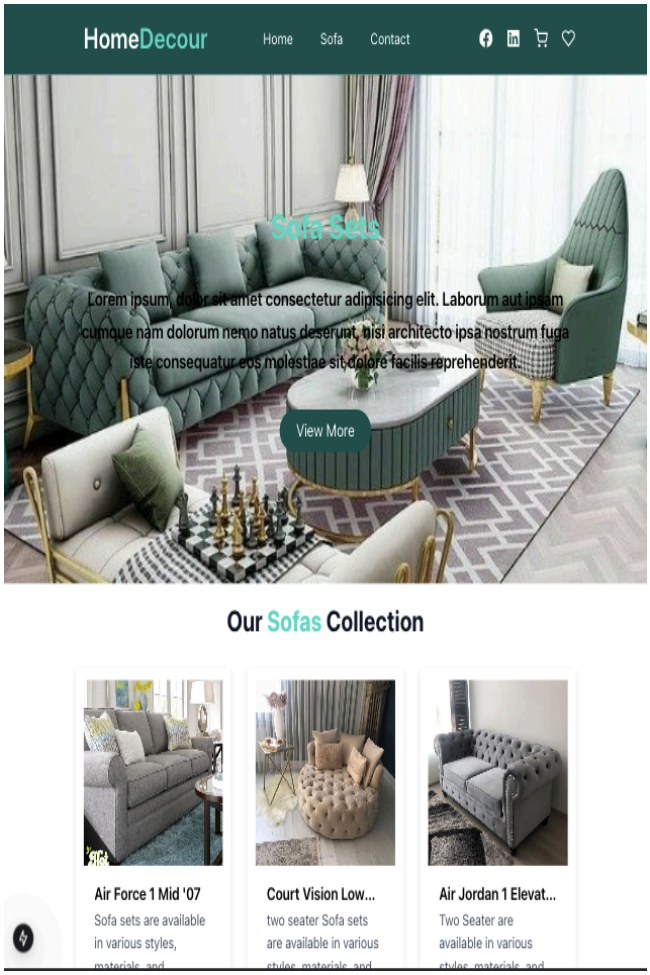


Day 4 - Dynamic Frontend Components “HomeDecour”

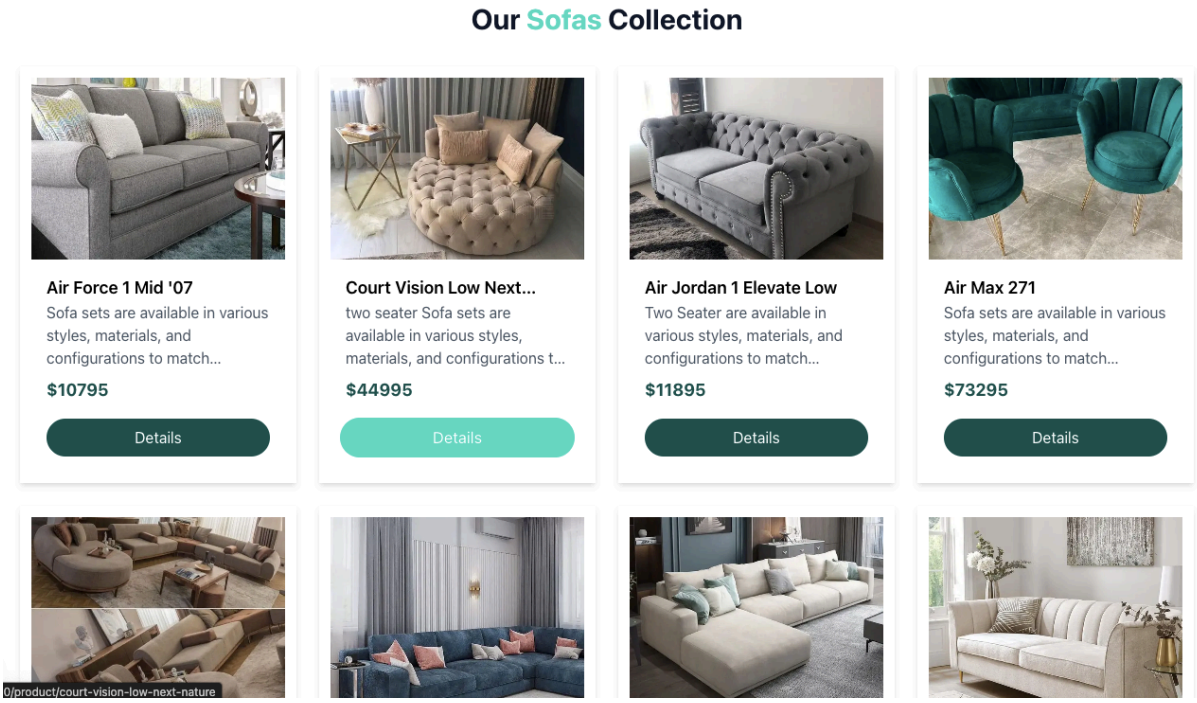
Marketplace_builder_hackathon_2025

ABSTRACT

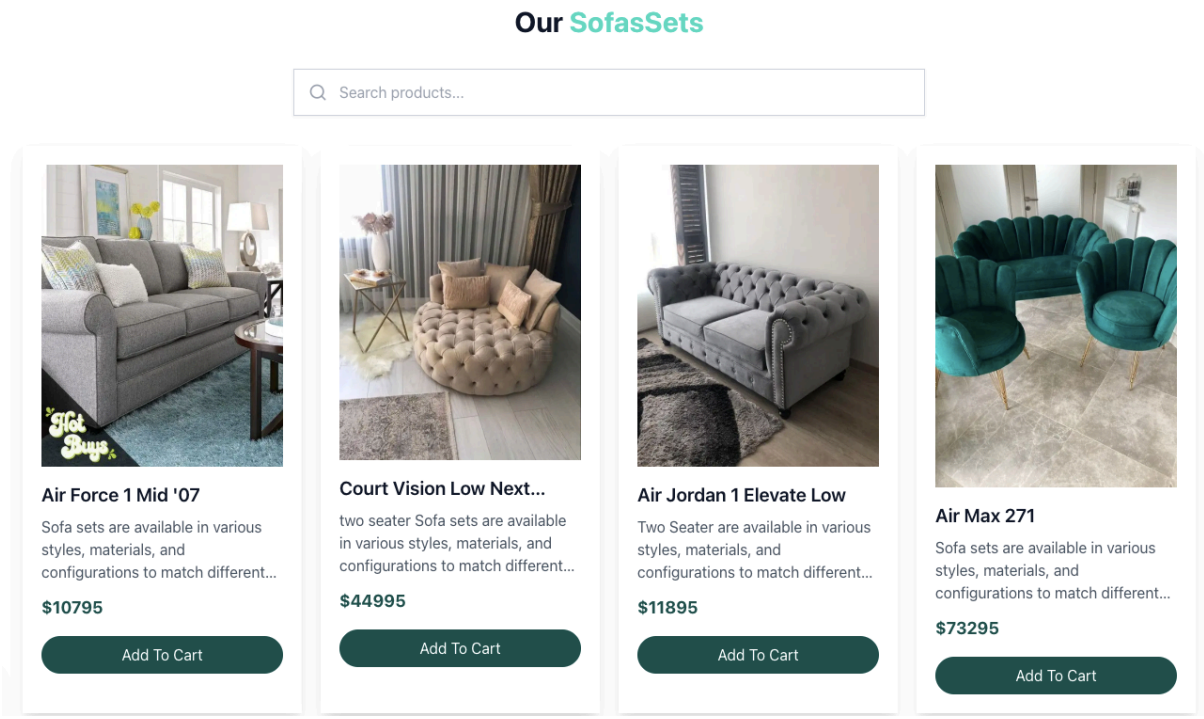
On Day 4, we will focus on designing and developing dynamic frontend components to display marketplace data fetched from Sanity CMS or APIs. This step emphasizes modular, reusable component design and real-world practices for building scalable and responsive web applications.

product listing page with dynamic data.:

product listed component section on front page.

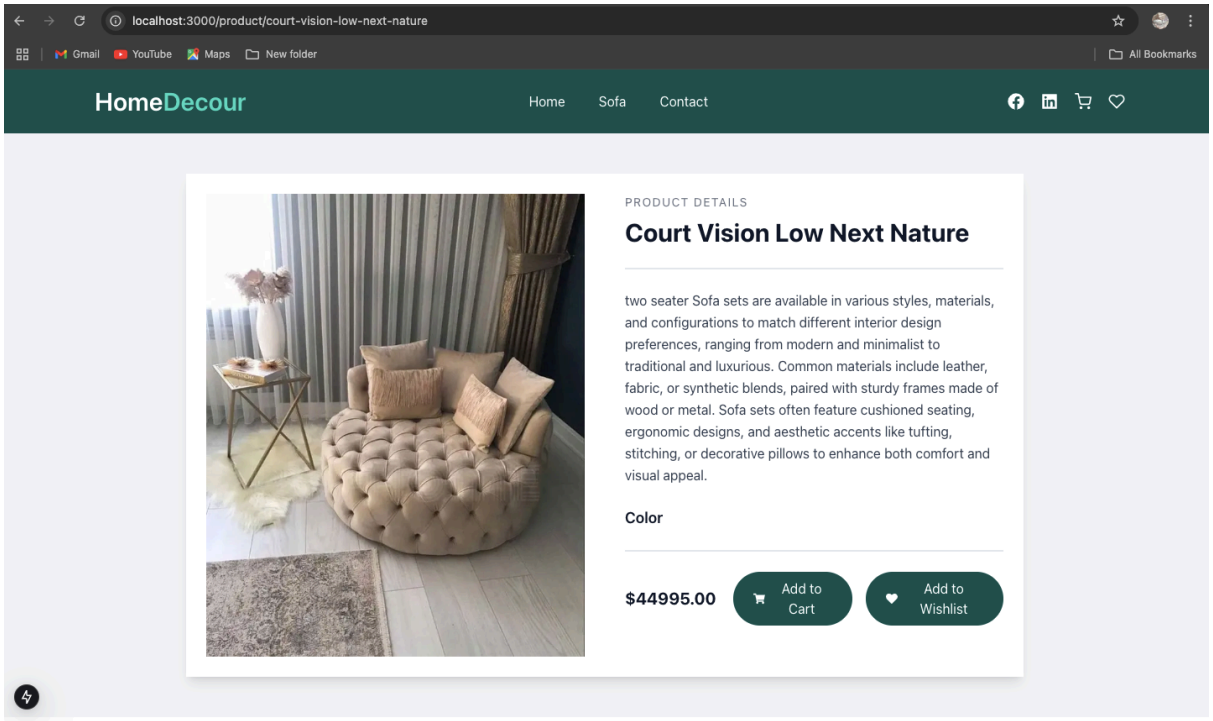


product listed page sofa/page.tsx.

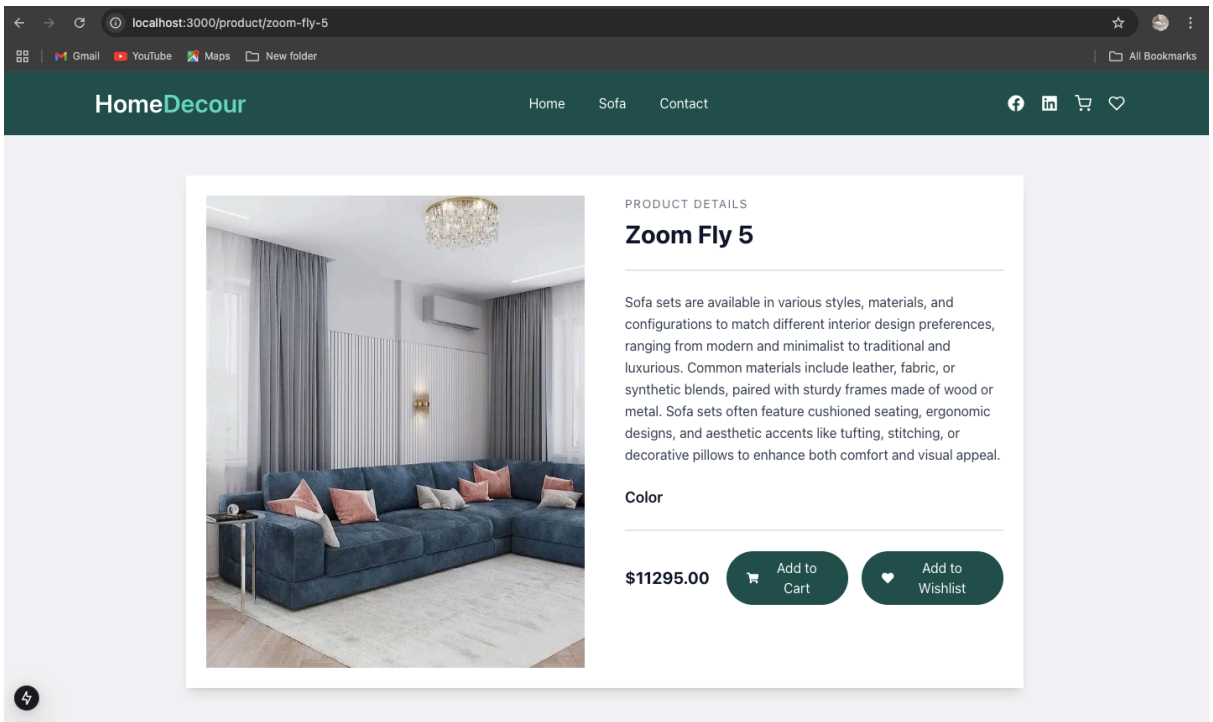


Individual product detail pages with accurate routing and data rendering.

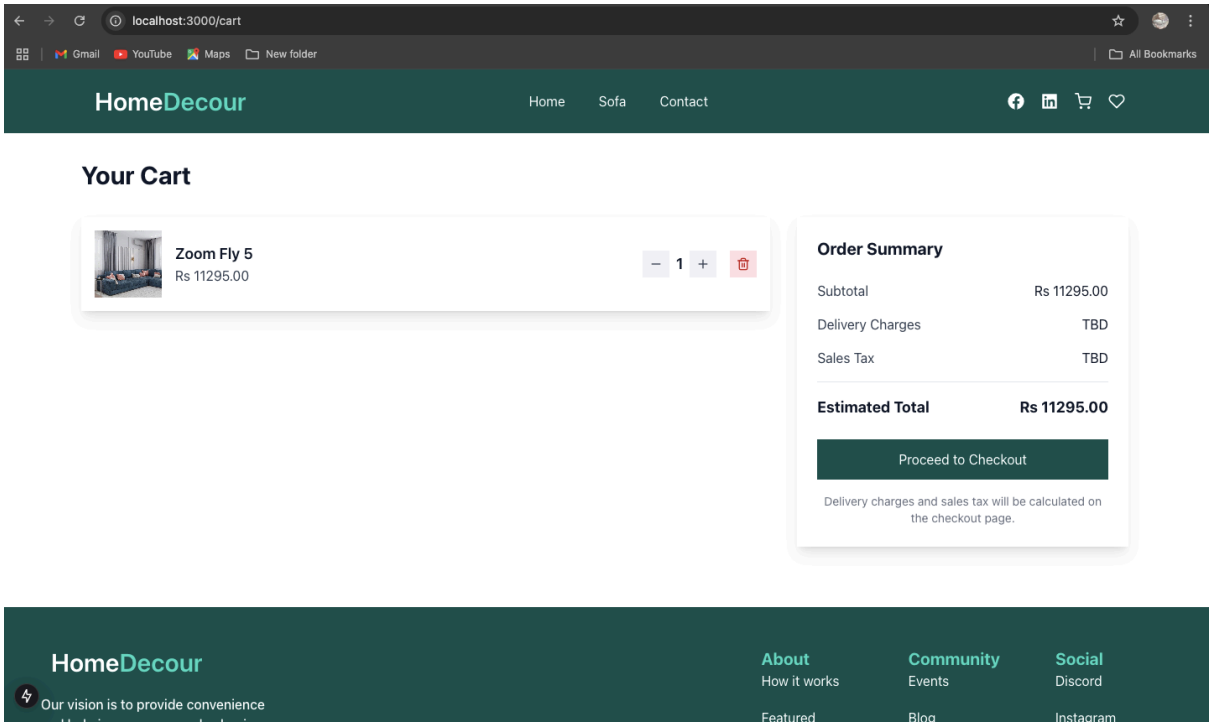
First item.



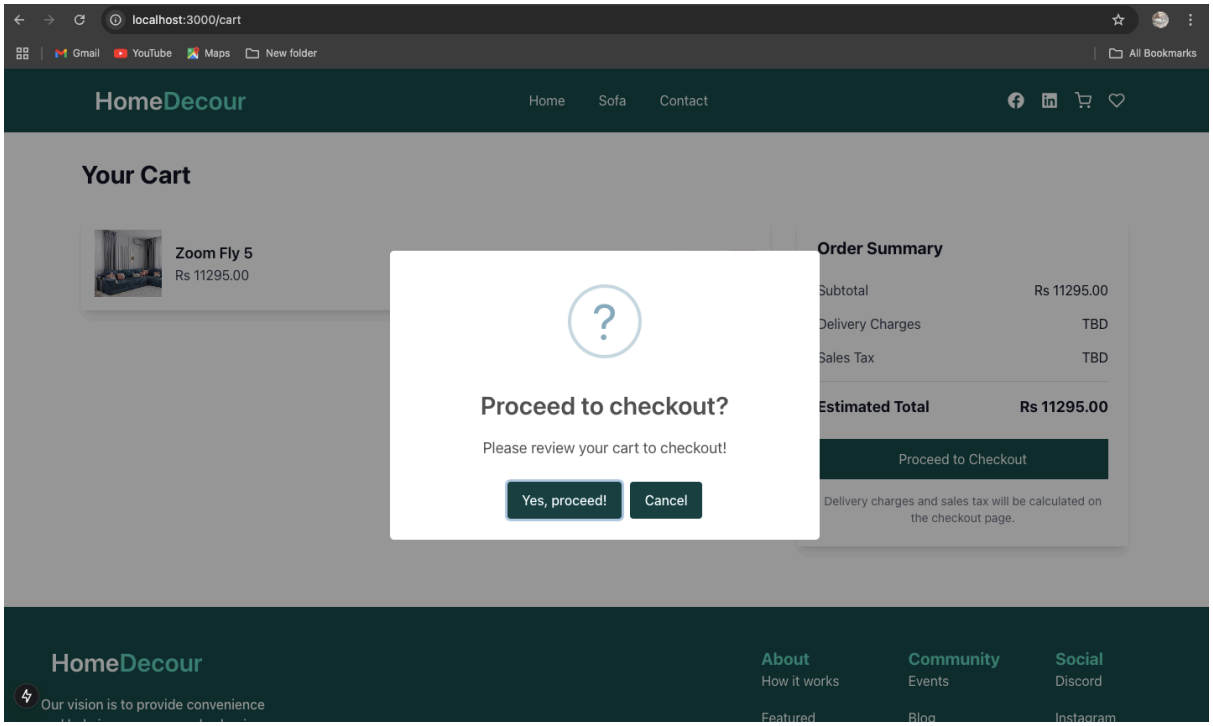
Second item.



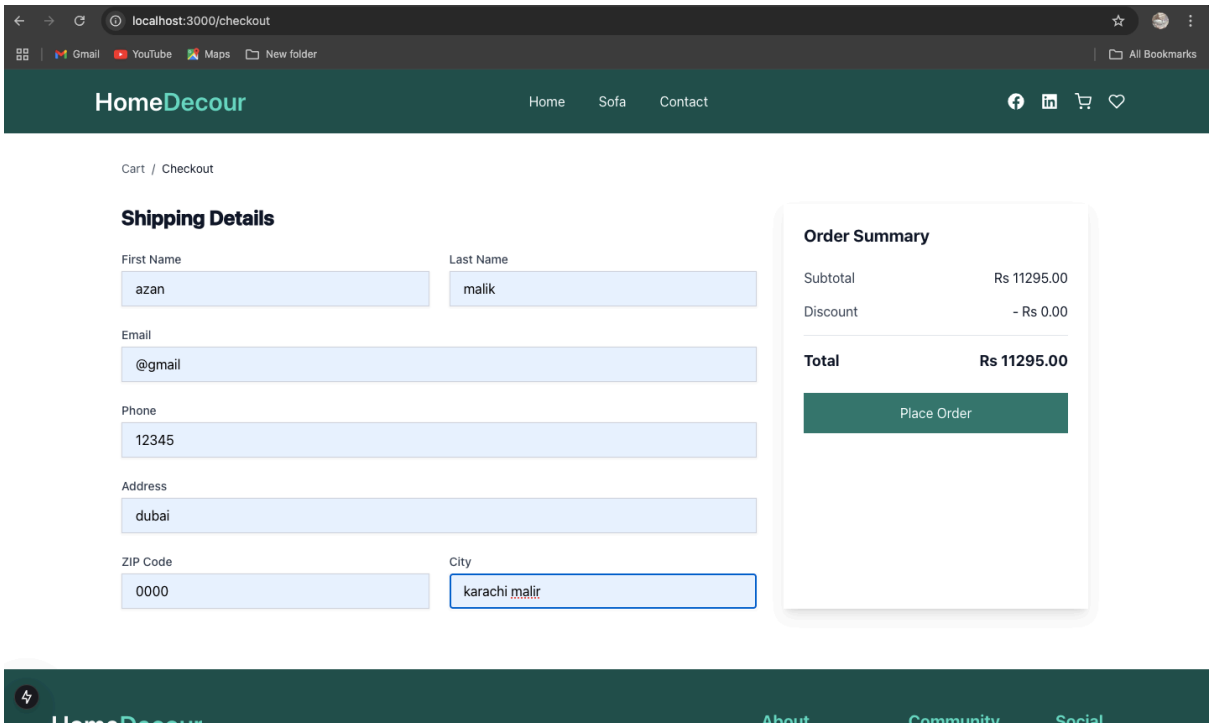
Add TO Cart functionality page:



Alert message for product procced:

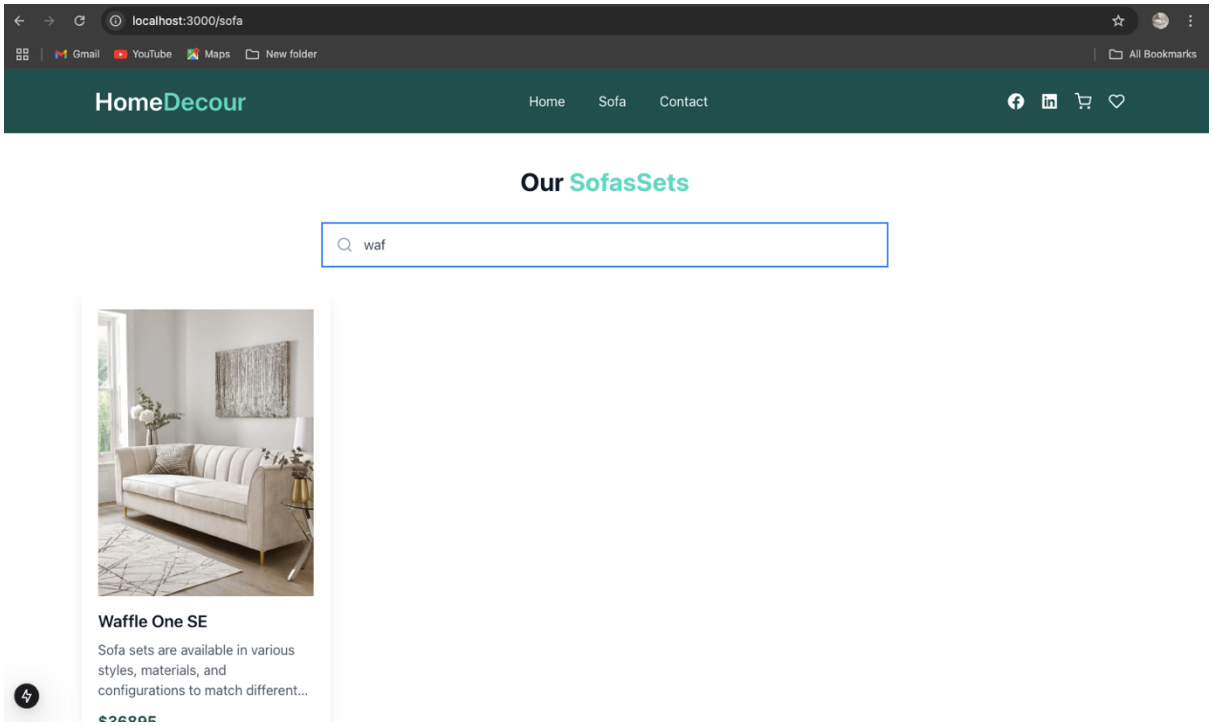


After Procced message checkout page open:

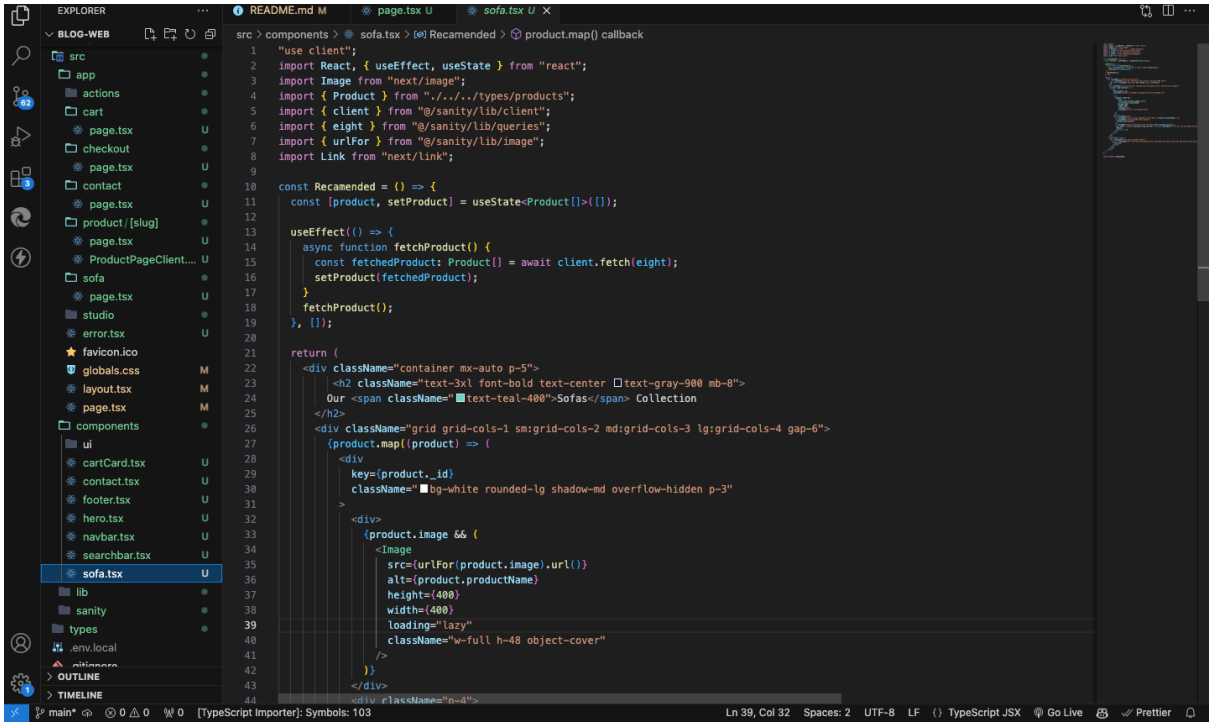


"I am working on placing an order in Sanity to store data. However, I am facing an issue where the data is not being stored in Sanity. I have tried fixing all the bugs, but it is still not working."

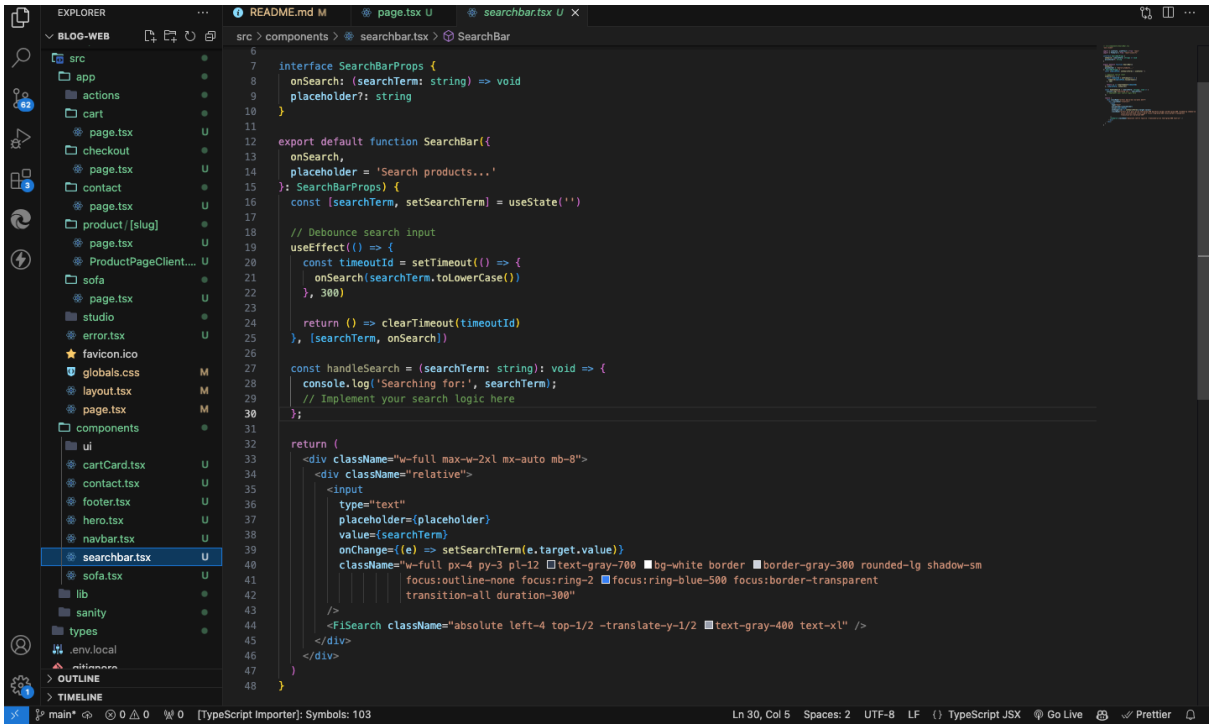
search bar



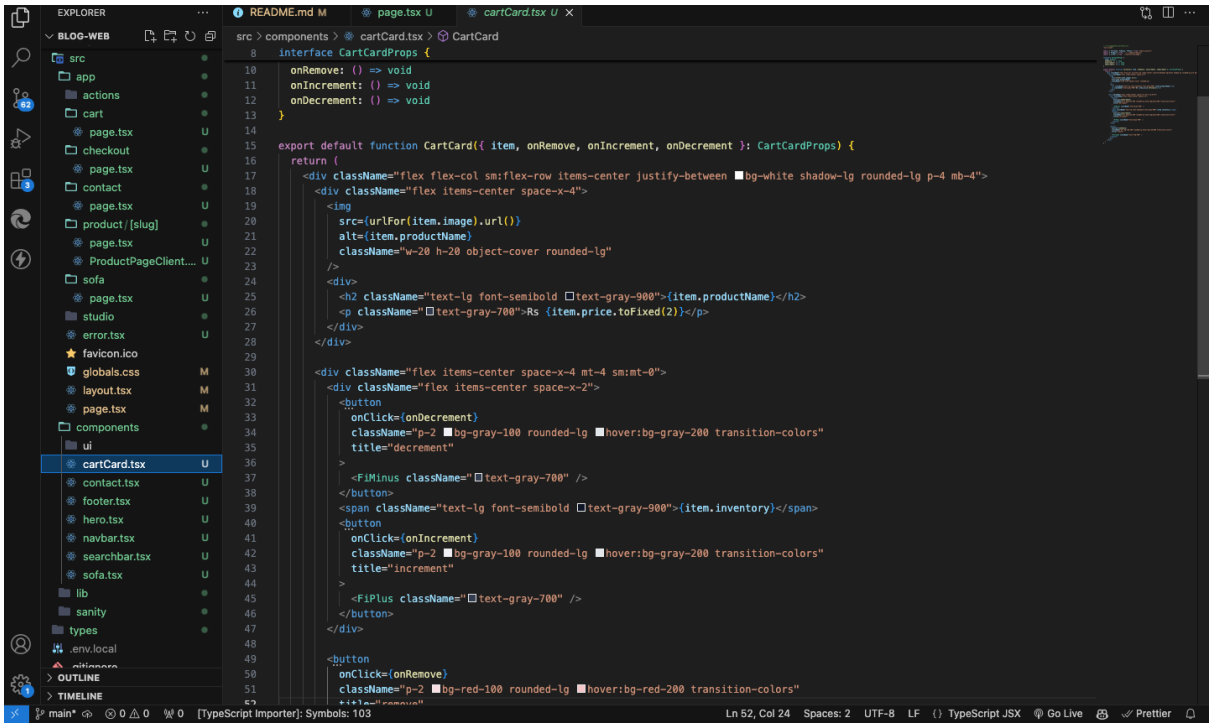
Code snippets for key components (ProductCard,



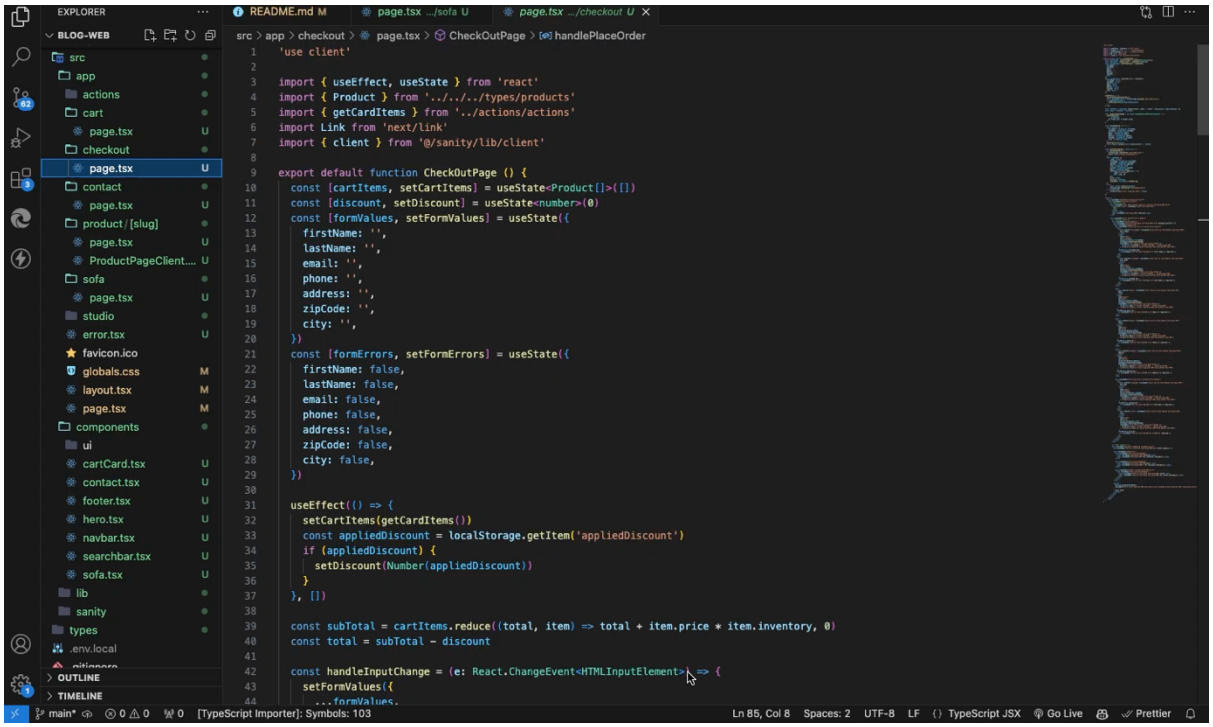
SearchBar



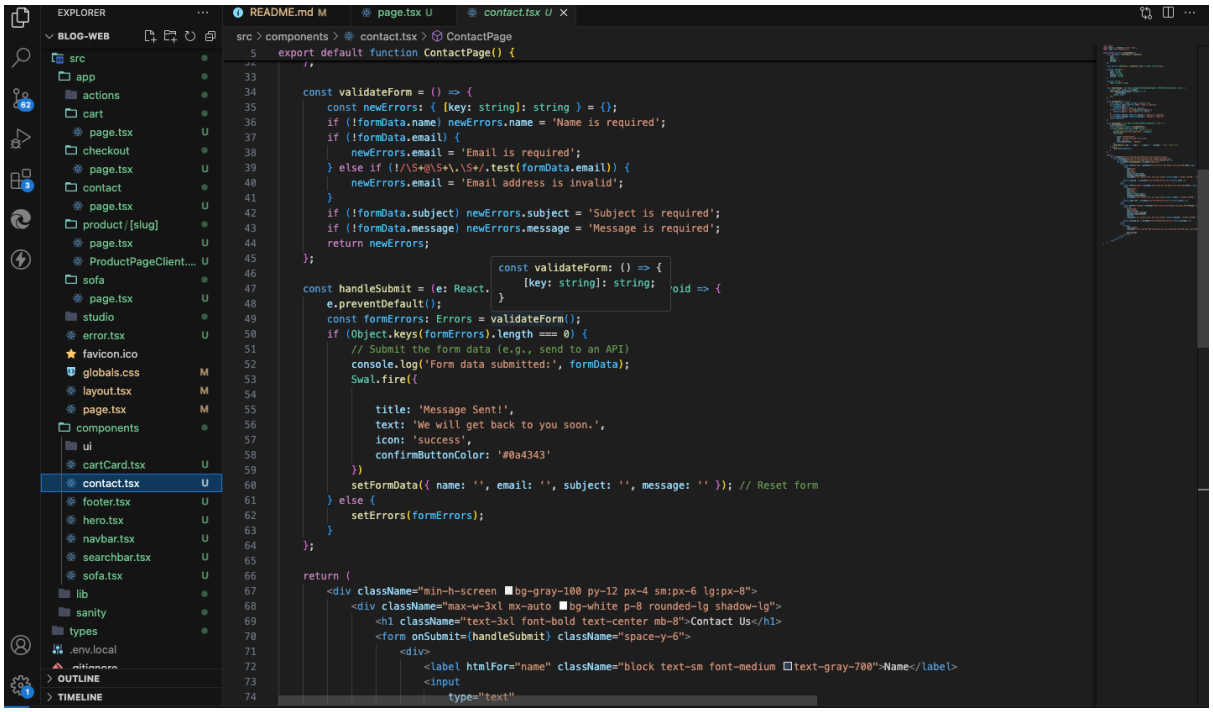
Cart Product Component



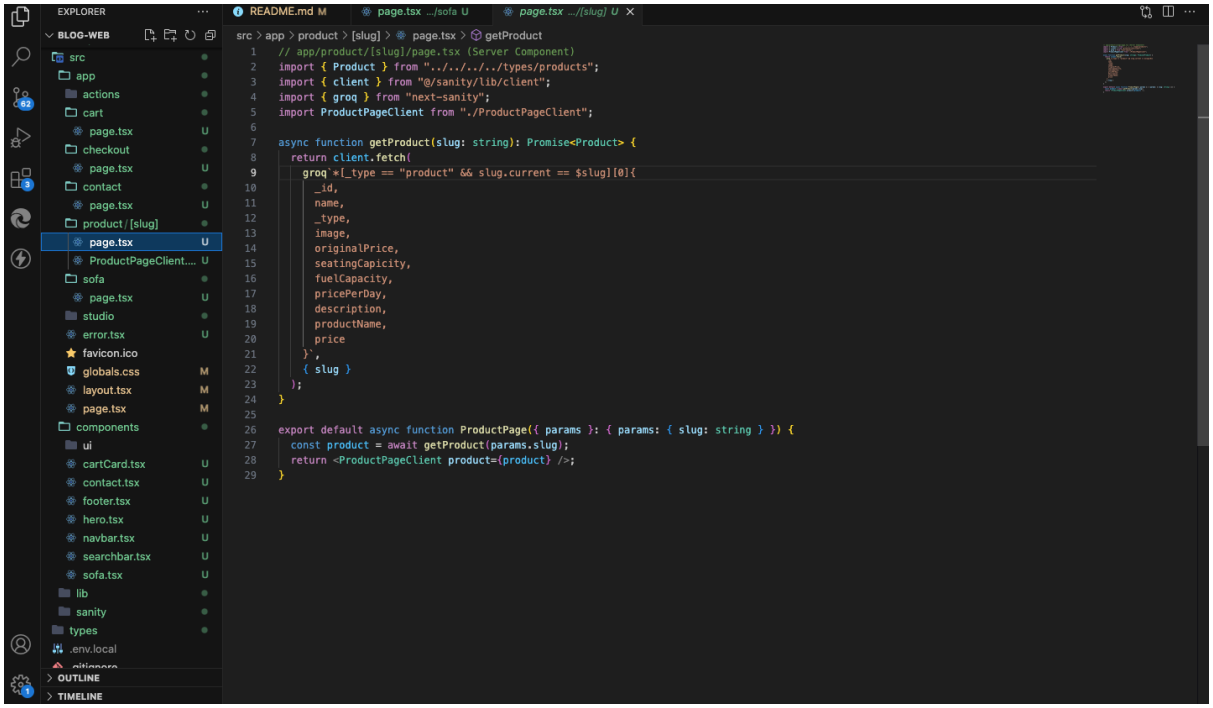
CheckOut Component:



Contact Component:



Dynamic Routing:



SUMMARY:

1. Introduction:

This report summarizes the steps taken to build and integrate key components (e.g., `ProductCard`, `ProductList`, `SearchBar`) for an e-commerce application. It also highlights the challenges faced, solutions implemented, and best practices followed during development.

2. Steps Taken to Build and Integrate Components

Component Breakdown:

The application was built using reusable React components. Below are the steps taken to create and integrate each component:

ProductCard Component:

Purpose: Displays individual product details (e.g., image, name, price, description).

Steps:

- Created a functional React component.
- Used Tailwind CSS for styling.
- Added an "Add to Cart" button for interactivity.

Integration: Used inside the `ProductList` component to render each product.

ProductList Component:

Purpose: Displays a list of products using the `ProductCard` component.

Steps:

- Mapped over an array of products and rendered a `ProductCard` for each item.
- Used a responsive grid layout (Tailwind CSS) to ensure the list looks good on all screen sizes.

Integration: Integrated with the `SearchBar` component to filter products dynamically.

SearchBar Component:

Purpose: Allows users to search for products by name.

Steps:

- Added a controlled input field to capture the search term.
- Implemented a search handler to filter products based on the search term.
- Used Tailwind CSS for responsive design.

Integration:

Connected to the `ProductList` component to update the displayed products dynamically.

Parent Component (App):

Purpose: Acts as the main container for all components.

Steps:

- Managed the state for products and filtered products.
- Passed necessary props to child components (`SearchBar`, `ProductList`).

Integration: Combined all components into a cohesive user interface.

3. Challenges Faced and Solutions Implemented

Challenge: Dynamic Product Filtering

Issue: The `SearchBar` component needed to filter products in real-time without reloading the page.

Solution:

- Used React's `useState` to manage the search term and filtered products.
- Implemented a search handler that filters the product list based on the search term.

Challenge: Responsive Design

Issue: Ensuring the components looked good on all screen sizes (desktop, tablet, mobile).

Solution:

- Used Tailwind CSS utility classes to create responsive layouts (e.g., `grid-cols-1 sm:grid-cols-2 md:grid-cols-3`).
- Tested the design on multiple devices and screen sizes.

Challenge: State Management:

Issue:

Managing the state of products and filtered products without unnecessary re-renders.

Solution:

- Used React's `useState` and `useEffect` hooks effectively.
- Ensured that state updates were optimized to avoid performance issues.

Best Practices Followed During Development:

Component Reusability

- Created modular and reusable components (e.g., `ProductCard`, `SearchBar`).
- Ensured components were decoupled and could be used in different parts of the application.

Responsive Design

- Used Tailwind CSS to create a mobile-first, responsive design.
- Tested the application on multiple devices and screen sizes.

Code Readability and Maintainability

- Followed consistent naming conventions for variables, functions, and components.

- Added comments to explain complex logic.
- Used meaningful prop names for better understanding.

Performance Optimization

- Avoided unnecessary re-renders by using React's state and effect hooks efficiently.
- Implemented debouncing for the search functionality to reduce API calls or filtering operations.

Testing and Debugging

- Tested each component in isolation using tools like Cypress or manual testing.
- Used `console.log` and React DevTools to debug state and props.

Conclusion:

The development process involved building reusable components (`ProductCard`, `ProductList`, `SearchBar`) and integrating them into a cohesive user interface. Challenges such as dynamic filtering, responsive design, and state management were addressed using React hooks and Tailwind CSS. Best practices like component reusability, responsive design, and performance optimization were followed to ensure a high-quality, maintainable application.