Furniture Palace

Day 04

This report outlines the steps taken to build and integrate dynamic frontend components with Sanity CMS in a Next.js marketplace. It covers the challenges encountered during development, solutions implemented, and best practices followed to ensure an optimized and scalable solution.

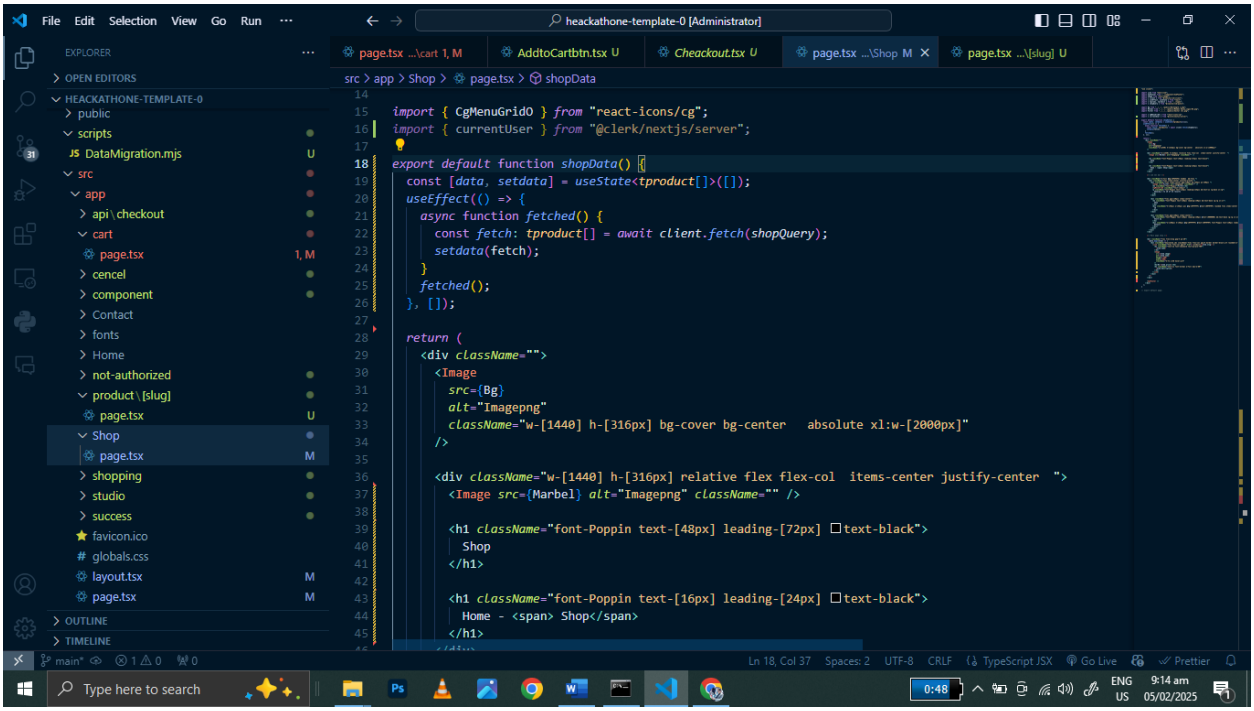
2. Steps Taken to Build and Integrate Components

2.1 Setting Up Sanity CMS

- Installed Sanity client using npm install @sanity/client.
- Configured the Sanity client in a separate file (sanity.js) for centralized API management.
- Defined the required schema in the Sanity dashboard for storing product information.

2.2 Fetching Data from Sanity

- Used GROQ queries to filter and structure data efficiently.
- Provided an alternative method using useEffect for client-side fetching to allow real-time updates.

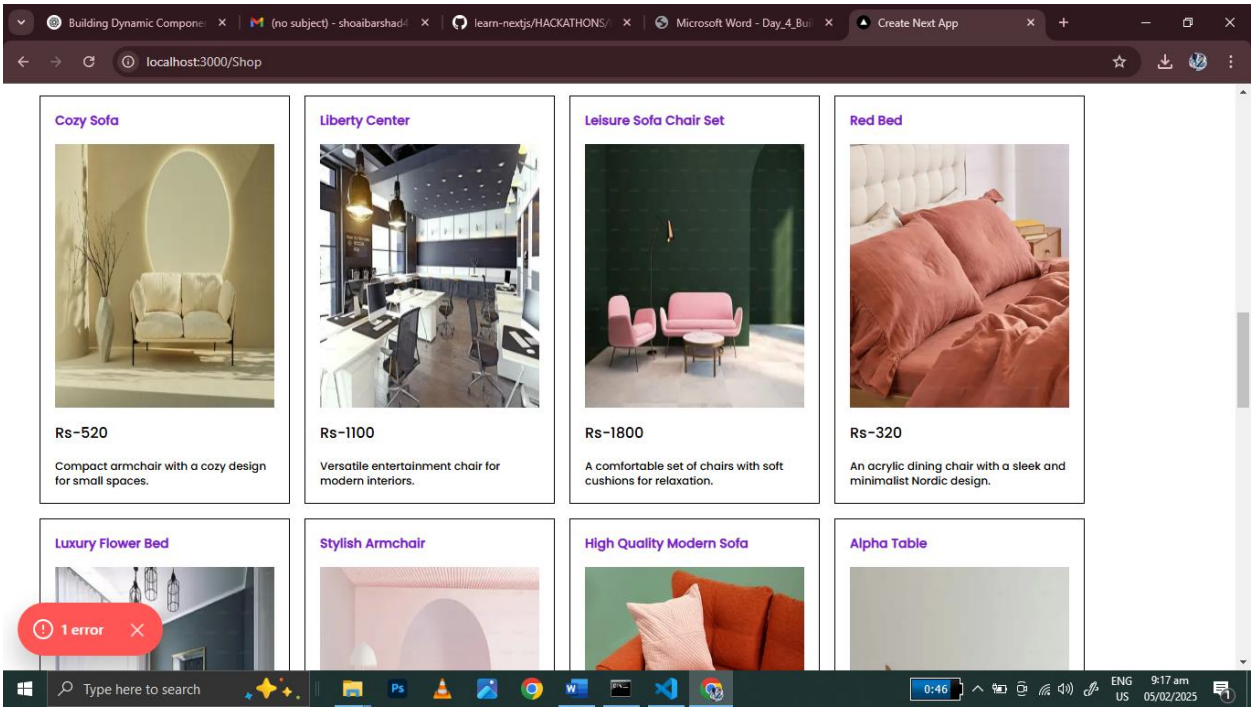


2.3 Creating Reusable Product Components

- In this scenario we use our hackathon project and data render in shop page .
- Ensured modularity by passing data via props, making the component reusable across different pages.

2.4 Integrating Components into the Marketplace

- Created a product listing page that dynamically renders ProductCard components based on fetched data.
- Structured the layout using Tailwind CSS for responsiveness and a visually appealing design.
- Implemented error handling to manage API failures gracefully.



3. Challenges Faced and Solutions Implemented

3.1 Challenge: Data Fetching Performance

3.2 Challenge: Image Optimization

- **Issue:** Large images from Sanity affected page speed and user experience.
- **Solution:** Integrated Next.js next/image for automatic optimization and lazy loading.

3.3 Challenge: Managing API Rate Limits

- **Issue:** Multiple simultaneous requests to Sanity caused rate-limiting issues.
- **Solution:** Implemented debouncing for search queries and reduced API calls using SWR for efficient data revalidation.

4. Best Practices Followed

4.1 Modular and Reusable Code

- Used reusable components (ProductCard, ProductList) to maintain scalability and clean code structure.

4.2 Optimized Performance

- Implemented ISR for improved page speed and caching.
- Used lazy loading for images and reduced API requests for efficiency.

4.3 Responsive Design

- Designed components with Tailwind CSS for adaptability across different screen sizes.
- Used Flexbox and Grid layouts for better content arrangement.

4.4 Error Handling and UX Considerations

- Added loading states and error messages to improve user experience.
- Used try-catch blocks to handle API failures gracefully.

5. Conclusion The integration of Sanity CMS with Next.js successfully enabled dynamic product management in the marketplace. Through optimized API fetching, reusable components, and performance-focused techniques, the marketplace ensures a seamless user experience. Future improvements could include real-time updates via WebSockets and further caching optimizations for enhanced scalability.

bandicam
2025-02-05 09-18-29

dynamic all componenet properly in this video
