Day 4 – DYNAMIC FRONTEND COMPONENTS – [AuraBox]

Documentation Author: Umar Ali

Slot: Tuesday(2 to 5)

Task given by: Sir Ameen Alam

Teachers: Sir Ali Aftab Sheikh & Sir Fahad Khan

Overview

Day 4 focused on advancing the functionalities of the project by building on Day 3 tasks. Below is a breakdown of the progress made:

Key Tasks and Steps

1. Reflection on Day 3 Work

- Revisited the work from Day 3, which included API integration, migration, and displaying data on the front end.
- Used this foundation to start new tasks.

---

2. Dynamic Product Detail Page

- **Objective:** Create a dynamic page for product details.
- **Challenges:** Encountered routing errors while implementing dynamic routing.
- **Solution:** Sought help from seniors to resolve the issue.
- **Outcome:** Successfully completed the task and verified the functionality.

```
import NavTwo from "@/components/NavTwo";
import TopHeader from "@/components/TopHeader";
import sanityClient from "@sanity/client";
import imageUrlBuilder from "@sanity/image-url";
import Image from "next/image";
import Link from "next/link";
import { FaHeart } from "react-icons/fa6";

// Initialize Sanity Client
const sanity = sanityClient({
  projectId: "itew173m",
  dataset: "production",
  apiVersion: "2023-01-01",
  useCdn: true,
});

// Configure Image Builder
const builder = imageUrlBuilder(sanity);
Pieces: Comment | Pieces: Explain
function urlFor(source: any) {
  return builder.image(source);
}

Pieces: Comment | Pieces: Explain
export async function generateMetadata({ params }: { params: { id: string } }) {
  const { id } = params;

  // Fetch product data for metadata
  const query = `
    *[_type == "product" && _id == "${id}"][0]{
      title
    }
  `;
  const product = await sanity.fetch(query);

  if (!product) {
    return { title: "Product Not Found" };
  }
```

---

3. Product Filtering

- **Objective:** Implement a filtering mechanism for products.
- **Challenges:** This was a relatively complex task due to multiple conditions.
- **Process:**
  - Leveraged APIs with existing tags.
  - Modified the API to include a "category" field for better filtering.

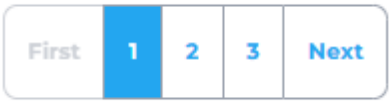
- **Outcome:** Successfully added a filtering system based on the provided tags and categories.

```
const ProductCards: React.FC = () => {
  const [products, setProducts] = useState<Product[]>([]);
  const [categories, setCategories] = useState<string[]>([]); // Category filter state
  const [priceRange, setPriceRange] = useState<string>(""); // Price range filter state
  const [selectedCategory, setSelectedCategory] = useState<string>(""); // Selected category
  const [selectedPriceRange, setSelectedPriceRange] = useState<string>(""); // Selected price range

  // Fetch categories
  const fetchCategories = async () => {
    const query = `*[_type == "category"]{title}`;
    const data = await sanity.fetch(query);
    Pieces: Comment | Pieces: Explain
    setCategories(data.map((category: any) => category.title));
  };

  // Fetch products with filters
  const fetchProducts = async () => {
    try {
      const categoryFilter = selectedCategory ? `&& category == "${selectedCategory}"` : "";
      const priceFilter = selectedPriceRange
        ? `&& price >= ${selectedPriceRange.split("-")[0]} && price <= ${selectedPriceRange.split("-")[1]}`
        : "";
```

---

4. Pagination Implementation

- **Objective**: Implement a seamless pagination system for the product listing page, displaying 8 products per page with navigation controls for "Next" and "Previous".
- **Challenges**: Faced difficulties in slicing the product data dynamically and synchronizing the pagination state with filters.
- **Solution**: Utilized the useState hook to track the current page and calculated the displayed products using array slicing. Ensured compatibility with existing filters by updating the dependency array in useEffect.
- **Outcome**: Successfully added pagination functionality, allowing smooth navigation between pages while preserving filters and maintaining performance. Verified the implementation with various datasets and edge cases.

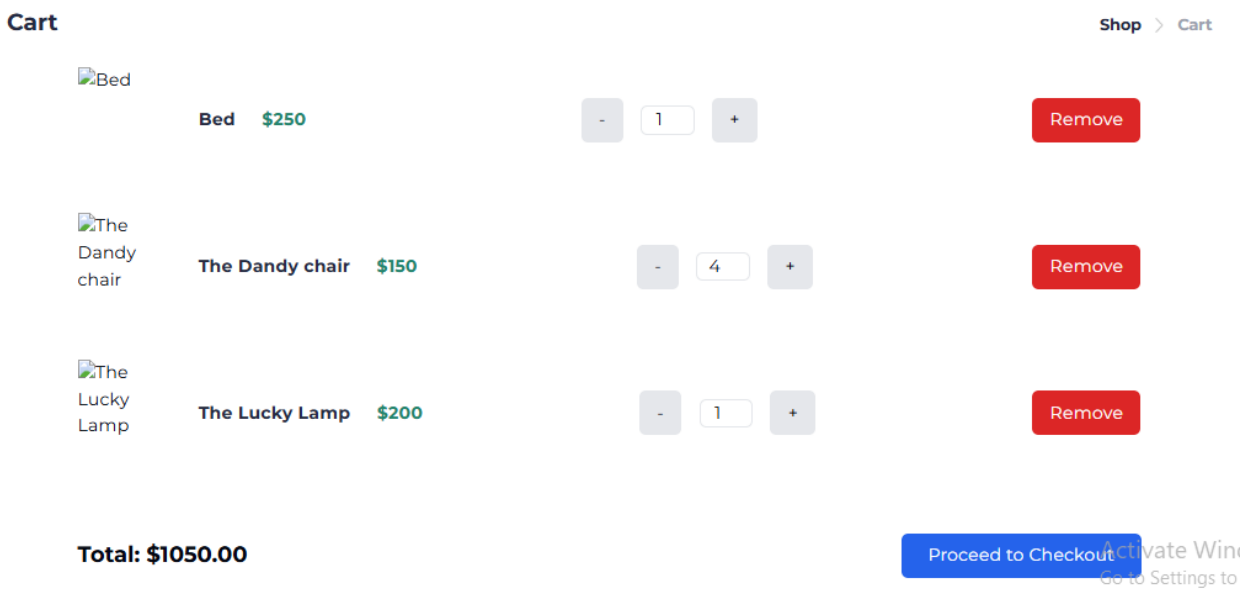



```
// Calculate displayed products based on current page
const displayedProducts = products.slice(
  (currentPage - 1) * itemsPerPage,
  currentPage * itemsPerPage
);

// Handle Next Page
const handleNextPage = () => {
  if (currentPage < Math.ceil(products.length / itemsPerPage)) {
    Pieces: Comment | Pieces: Explain
    setCurrentPage((prev) => prev + 1);
  }
};

// Handle Previous Page
const handlePrevPage = () => {
  if (currentPage > 1) {
    Pieces: Comment | Pieces: Explain
    setCurrentPage((prev) => prev - 1);
  }
};
```

5. Add-to-Cart Functionality

- **Objective:** Integrate and implement the "Add to Cart" feature.
- **Challenges:** Encountered errors with image handling during implementation.
- **Solution:** Resolved these errors with the assistance of student leaders.
- **Process:**
  - Created front-end logic using useState and other React functionalities.
  - Designed a user-friendly UI to display cart details.
- **Outcome:** Completed the task, including displaying cart details on the front end.

IMAGE IS NOT SHOWING BCZ OF NETWORK ERROR

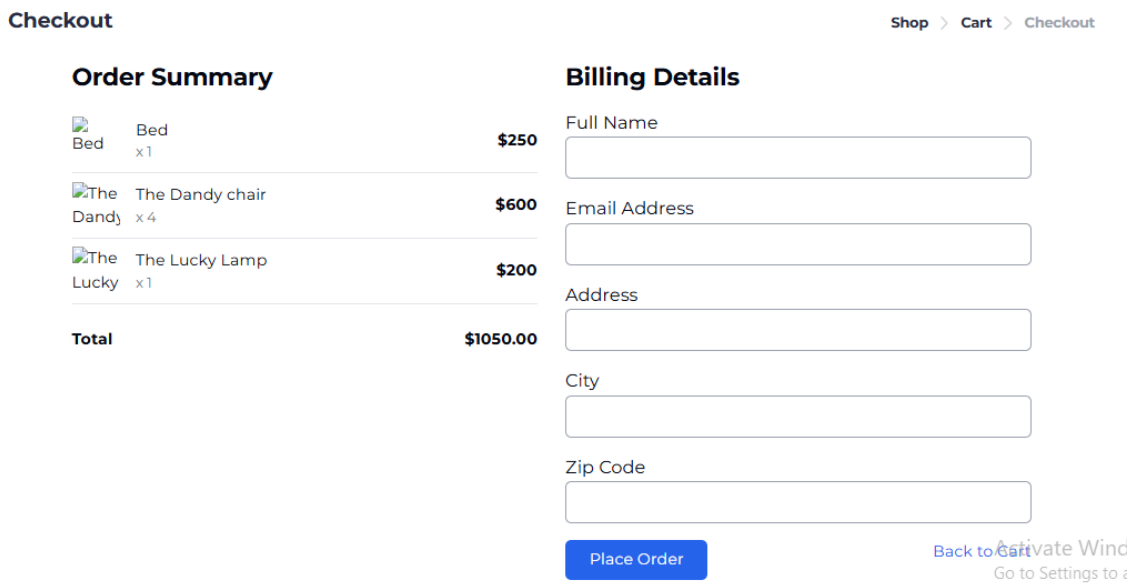


---

6. Checkout and Order Flow

- **Objective:** Finalize the checkout process.
- **Steps:**
  - Designed and implemented the checkout page.
  - Created an order confirmation page.
  - Integrated shipment tracking functionality.
- **Outcome:** Fully functional order flow completed

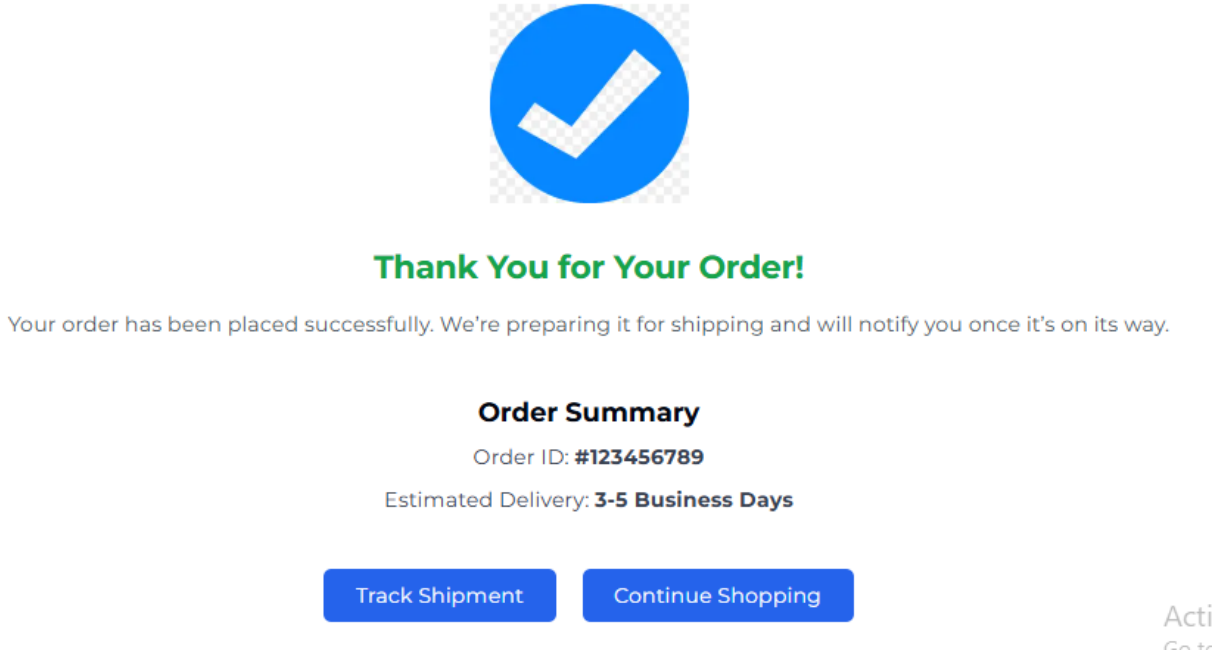
IMAGE IS NOT SHOWING BCZ OF NETWORK ERROR



---

7. Order Confirmation Page

- **Objective:** Build and display an order confirmation page after checkout.
- **Steps:**
  - Fetched the user's order details dynamically using APIs.
  - Displayed the order summary, including product details, prices, and shipping information.
  - Included a success message and a "Back to Home" button for better user experience.
- **Outcome:** The order confirmation page was successfully designed and implemented, enhancing the user's journey post-checkout.



Best Practices Followed During Development

1. Component Reusability

UI components were broken down into smaller, reusable modules, ensuring modularity, maintainability, and ease of updates.

2. Responsive Design

Tailwind CSS was used to create a mobile-first design that adapts seamlessly across all screen sizes, ensuring a great experience on both desktop and mobile.

3. TypeScript for Type Safety

TypeScript was leveraged to enforce type safety, reducing runtime errors and improving overall code reliability, making the development process smoother.

4. Code Readability and Comments

The codebase was thoroughly documented, with comments explaining key logic, decisions, and functionality to improve readability and maintainability.

5. User Experience (UX)

Focused on creating an intuitive interface with clear visual cues and easy navigation, ensuring that users could browse and purchase products with ease.

Summary

Day 4 covered the creation of dynamic product pages, a filtering system, the add-to-cart functionality, the checkout process, and the order confirmation page. Challenges were overcome with collaboration, and the tasks were completed with a focus on clean, reusable code.