Technical Report: E-Commerce Platform Development

Project Overview

This project involves the development of an e-commerce platform using **Next.js** with the **App Router** and **Sanity CMS** for content management. The primary features include the ability to display and manage product listings, handle customer cart functionality, and provide dynamic detail pages for categories like catering, decoration, consultancy, and reservation.

---

1. Steps Taken to Build and Integrate Components

1.1 Setting Up Next.js with Sanity CMS

The first step was to set up a **Next.js project** using the **App Router** for routing. Sanity CMS was integrated to handle content like products, services, and categories dynamically.

- **Sanity Client Configuration:** The project connects to Sanity using the `@sanity/client` to fetch and manage content such as product details and categories.
- **API Integration:** Custom API endpoints were created to fetch data from Sanity, specifically for dynamic pages like catering, decoration, consultancy, and reservation services.
- **Dynamic Routes:** Implemented dynamic routing to handle product and service details based on their unique identifiers (ID) by utilizing the Next.js `[id]` route syntax.

1.2 Cart Functionality and Integration

The cart was integrated into the application, where the data for the cart is stored and updated dynamically.

- **Cart Context Provider:** Created a `CartContext` using React's Context API to manage the cart state (add, remove, update items).
- **Cart Modal:** Implemented a cart modal that opens when the user clicks on the cart icon, displaying the items in the cart and their total cost. This modal dynamically updates the cart contents when items are added or removed.
- **Add to Cart Functionality:** Each product page includes an "Add to Cart" button, which, when clicked, adds the product to the cart and updates the modal.

1.3 Layout and Design

The layout of the application was developed with responsiveness and user experience in mind.

- **Font Integration:** Custom fonts were integrated using Next.js's `localFont` API to ensure the design met the desired aesthetic.
- **Components Used:** Various reusable components like `Header`, `Footer`, `CartContext`, and product detail components were built. The design includes a

responsive header, a footer with contact details, and a dedicated product section for each category (catering, decoration, consultancy, etc.).
- **Dynamic Category Pages:** Categories like catering, decoration, consultancy, and reservations were made dynamic, with each having its own detail page fetched from Sanity and displayed appropriately.

---

2. Challenges Faced and Solutions Implemented

2.1 Data Fetching and Integration Issues

- **Challenge:** Initially, there were issues with properly fetching dynamic data from Sanity, especially for categories with nested or complex data structures (e.g., menus in catering).
- **Solution:** The data was fetched using Sanity's query syntax, ensuring all required fields (like `name`, `details`, `price`, etc.) were correctly queried for each product or service. The queries were optimized to only fetch the necessary fields to minimize data transfer and improve page load times.

2.2 Cart Modal Implementation

- **Challenge:** A major challenge was integrating the cart modal to open and close dynamically without navigating to a separate page.
- **Solution:** This was solved by implementing the cart functionality using a **Cart Context** provider, which allowed the cart state to be shared across the application. The modal was dynamically displayed on top of the content when the user clicked the cart icon, with updates reflected in real-time.

2.3 Font and Static Assets Issue

- **Challenge:** There was an issue with loading custom font files (`.woff` and `.woff2`) due to incorrect paths in the project configuration.
- **Solution:** The font files were correctly placed in the `public/fonts/` folder, and paths in the `localFont` configuration were updated to reflect the correct static path.

2.4 Handling Product Variants and Price Updates

- **Challenge:** Managing variants for products, like different catering menus or decoration services, and reflecting the price changes in the cart dynamically was complex.
- **Solution:** A more robust data structure was created to handle product variants and their prices. This allowed for easy updates to product details and ensured that any price changes were immediately reflected in the cart modal.

---

3. Best Practices Followed During Development

3.1 Code Modularity and Reusability

- The project was designed with modularity in mind. Reusable components such as `Header`, `Footer`, `CartModal`, and product detail components were created to ensure that any future features could be added without causing a significant impact on existing functionality.

3.2 Responsiveness and Mobile-First Design

- The website layout was developed to be fully responsive, using CSS Grid and Flexbox to ensure a smooth experience on both desktop and mobile devices. This ensures the platform is accessible and user-friendly across a variety of screen sizes.

3.3 Context API for State Management

- The **React Context API** was used to manage the cart state globally. This practice ensures that the cart's data is consistently accessible throughout the app, without the need for prop drilling or unnecessary state lifting.

3.4 Error Handling

- The application includes proper error handling for situations like missing product data or unavailable services. For example, if a product or service is not found, the user is redirected to a `notFound()` page.

3.5 Performance Optimization

- The queries to Sanity were optimized to only request necessary fields, and the `useCdn: false` flag was set to ensure the latest data was always fetched. Additionally, all images and assets are optimized for performance, ensuring fast page load times.

Conclusion

This project resulted in a fully functional and dynamic e-commerce platform with categories for catering, decoration, consultancy, and reservation services. It leverages the flexibility of **Next.js** and **Sanity CMS** for content management and dynamic routing. By following best practices for modularity, state management, and responsive design, the platform ensures both a high-quality user experience and scalable, maintainable code.