Project Report: E-Commerce Website Development

Table of Contents

---

1. Introduction

This report outlines the development and testing process for the e-commerce website. The goal was to build a fully functional e-commerce platform with dynamic product listings, cart functionality, and secure customer login. The project was developed using **Next.js**, **TypeScript**, **Sanity**, and other technologies tailored for the web development ecosystem.

---

2. Project Overview

The e-commerce platform was designed to cater to users looking for specific services, such as catering, decoration, reservation, and consultancy. The website includes the following features:

- **Product Listing**: A dynamic list of products, categorized into different services, allowing users to explore and purchase.
- **Cart Functionality**: A cart system for users to add, view, and remove items without navigating to a separate page.
- **Customer Login**: Secure login functionality to allow customers to access and manage their profiles.

---

3. Development Process

3.1 Technologies Used

- **Next.js**: Framework for building server-side rendered React applications.
- **TypeScript**: Strongly typed JavaScript used to enhance code reliability and maintainability.
- **Sanity**: Headless CMS used for content management.
- **React**: Frontend JavaScript library for building user interfaces.
- **Axios/Fetch**: For handling API requests.

3.2 Key Features Implemented

- **Dynamic Routing**: The site includes dynamic pages for each product/service, allowing users to explore details based on selected categories like catering, decoration, and consultancy.
- **Cart Functionality**: The shopping cart opens as a sheet directly from the navbar, improving user experience.
- **Authentication**: A login page integrated with mock API functionality and a migration script for customer data.
- **Admin Panel**: Integration with Sanity to manage product and customer data efficiently.

---

4. Testing

4.1 Testing Methodology

Testing was conducted using both manual and automated approaches. The following strategies were used:

- **Unit Testing**: Tests for individual components like the `Cart`, `ProductList`, and `ProductDetails` were created.
- **Integration Testing**: Ensured that components like the cart, checkout, and login functionalities integrated smoothly with the API and the backend (Sanity).
- **End-to-End Testing**: Simulated user interactions to ensure the full flow of the website worked as expected from browsing to checkout.

4.2 Test Reports

Test 1: Cart Functionality

- **Test Case**: Add items to cart and validate cart contents.

- **Test Result**: Passed. Items were successfully added, removed, and displayed correctly in the cart.

Test 2: Dynamic Routing

- **Test Case**: Verify navigation from product list to product detail page.
- **Test Result**: Passed. Routing was dynamic and correctly rendered based on selected product ID.

Test 3: Login Functionality

- **Test Case**: Test login with valid and invalid credentials.
- **Test Result**: Passed with adjustments. Mock API login handled correctly. Invalid credentials led to appropriate error messages.

Test 4: Sanity Integration

- **Test Case**: Check content from Sanity API.
- **Test Result**: Passed. Sanity CMS integration provided the correct data.

4.3 Test Case Details

Test Case ID	Test Description	Expected Outcome	Actual Outcome	Status
TC001	Cart items add/remove	Items should be added to cart and removed properly.	Pass	Passed
TC002	Dynamic routing for product detail	On clicking a product, user should be redirected to the respective detail page.	Pass	Passed
TC003	User login with valid/invalid credentials	Valid credentials should authenticate the user, invalid credentials should show error.	Pass	Passed
TC004	Content from Sanity	Sanity API should provide correct content for product listings.	Pass	Passed

---

4.4 Testing Tools Used

- **Jest**: Used for unit testing and testing React components.
- **React Testing Library**: For testing React components by simulating user interaction.
- **Cypress**: Used for end-to-end testing and simulating real user interactions with the entire website.
- **Postman**: For testing the mock API and validating API responses.
- **Sanity Studio**: For content management and API validation.

4.5 Bug Fixes and Resolutions

Bug 1: Client-Side `useCart` Hook Error

- **Issue**: `useCart` was called on the server side, leading to an error.
- **Resolution**: The cart functionality was moved to a client-side component, and the state was correctly managed.

Bug 2: Cart Items Lacking Unique Keys

- **Issue**: React warning for missing unique keys in the cart list.
- **Resolution**: The key for each cart item was updated to use both `item.id` and the `index` for uniqueness.

Bug 3: Incorrect Param Handling in Dynamic Routes

- **Issue**: Access to `params` was directly attempted, leading to async issues.
- **Resolution**: Used `React.use()` to unwrap `params` and handle async operations properly.

---

5. Optimization

5.1 Performance Optimization

- **Lazy Loading**: Implemented lazy loading for images and non-critical components to improve initial page load time.
- **Code Splitting**: Used Next.js's automatic code splitting to ensure only the necessary JavaScript is loaded per page.
- **Caching**: Implemented caching strategies for API calls and static assets to reduce unnecessary network requests.

5.2 Code Optimization

- **Refactored Code**: Removed redundant code and made use of reusable components to keep the codebase clean.
- **Component Optimization**: Used `React.memo()` and `useMemo()` to prevent unnecessary re-renders of components.
- **Asynchronous Loading**: Implemented async components loading to improve performance and reduce initial load time.

5.3 Image and Asset Optimization

- **Next.js Image Optimization**: Leveraged Next.js's `Image` component for optimized image loading (resizing, lazy loading).
- **SVG Icons**: Used SVG for icons to reduce file sizes and improve rendering performance.

- **Compressed Assets**: Compressed and minified CSS, JavaScript, and image files to reduce their size and improve load time.

---

6. Best Practices Followed

- **Modular Code Design**: Components were kept small and reusable to maintain scalability.
- **Separation of Concerns**: Backend logic (Sanity API) was separated from frontend components for better maintainability.
- **Responsive Design**: The website was designed to be mobile-friendly, using CSS utilities like Flexbox and Grid.
- **Error Handling**: Proper error messages were provided for common issues such as incorrect login credentials or missing products.

---

7. Conclusion

This project successfully met all its core requirements, delivering a user-friendly and functional e-commerce website. Extensive testing, combined with continuous integration of bug fixes and improvements, resulted in a stable and reliable platform. By adhering to best practices and maintaining a modular architecture, the project is well-positioned for future scalability and enhancements.