Contents

Technical Report: E-commerce Website Development

Introduction

This report documents the steps taken, challenges faced, solutions implemented, and best practices followed during the development of an **e-commerce website**. The website includes features such as product filtering via search-bar, price-range-slider, size-filter, category-filter and a custom design system for a seamless user experience.

---

1. Steps Taken to Build and Integrate Components

1.1. Setup and Initial Development

- **Project Setup**: The project was initiated using **Next.js** with TypeScript, which ensures type safety and scalability. The initial setup involved installing necessary dependencies like **Tailwind CSS** for styling and **ReactIcon** for icon usage and **toast** for notifications.
- **Design Implementation**: A simple UI design was created with a **light theme** incorporating **white, black, and orange** colorsThe homepage featured **product cards** that displayed essential information (e.g., name, image, price). Animations and hover effects were implemented using **Tailwind CSS** for a smooth user experience.

1.2. Search Functionality

- **Search Integration**: A custom search feature was developed that allows filtering of product cards based on search input. The filtering occurs dynamically within the existing product card display, eliminating the need for navigating to a separate page. This provides a seamless and fast user experience.
- **Implementation Details**: The search functionality uses **React state** to dynamically filter product data displayed on the homepage. The search input updates the UI without causing page reloads.

1.3. Product Filtering

- **Category Filtering**: A **category filter** was implemented to allow users to narrow down their search by product categories. This was achieved using React state to update the displayed products based on the selected category.
- **Size Filtering**: A **size filter** was added for size-based product selection (e.g., small, medium, large). This was integrated into the filtering logic, enabling users to refine their search based on the available sizes for each product.
- **Price Range Slider**: A **price range slider** was implemented to allow users to filter products within a specified price range. The slider dynamically adjusts the product list based on the user's price selection, providing a more refined shopping experience.

1.4. Responsive Design

- The components were made responsive using **Tailwind CSS**, ensuring the layout adapts well to different screen sizes (mobile, tablet, desktop).

1.5. Review Section

- **Component Creation**: A **review section** was developed to allow users to leave feedback on products. This includes a star rating system and a comment box where users can share their experiences.
- **Integration**: The review section dynamically updates as new reviews are added.

1.6. Social Media Share Buttons

- **Component Creation**: **Social media share buttons** were integrated to allow users to share products or pages directly to social media platforms like Facebook, Twitter, and WhatsApp.
- **UI/UX Design**: The buttons were designed with clear, recognizable icons and placed in an accessible location on product pages and blog posts.

1.7. Cart Component

- **Component Creation**: A **cart component** was created to allow users to add products they intend to purchase. This includes features such as updating quantities, removing items, and viewing the total cost.
- **State Management**: The cart uses **React Context API** for global state management. This allows the cart data to be accessed and updated across different components in the application without the need to pass props down manually.

1.8. Wishlist Component

- **Component Creation**: A **wishlist component** was developed to allow users to save products for future purchase. Users can add and remove items from the wishlist, and these items are displayed in a dedicated section.
- **State Management**: Like the cart, the wishlist uses **React Context API** for managing state globally. This ensures the wishlist persists across components and users can access their saved items seamlessly.

1.9. Dynamic Product Cards

- **Component Creation**: **Product cards** were dynamically generated using data fetched from **Sanity CMS**. Each card displays essential product details such as the name, price, and image.
- **Data Handling**: The data for each product is fetched dynamically from an external source (Sanity CMS), and **React state** is used to manage and display the products.
- **Rendering**: The product cards are rendered in a grid layout, ensuring they adjust dynamically to different screen sizes using **Tailwind CSS** for responsiveness.

1.10. Dynamic Routing for Product Details

- **Dynamic Routes**: A **dynamic route** was set up using **Next.js app routing** to handle individual product details pages. Each product card is linked to a dedicated product details page, which is accessible by clicking on the product card.
- **Dynamic Path**: The dynamic route uses the **product ID** as a path parameter, and the route is constructed as /products/[_Id]. On visiting this route, the product data is fetched dynamically using the product ID, and the details page displays the specific product information (e.g., full description, additional images, price, reviews).

1.11 FAQ Section with Search Functionality

- **Component Creation**: An FAQ section was developed to address common user queries. This feature includes a search bar that allows users to find answers by entering relevant keywords.

- **Search Implementation**: The search functionality dynamically filters FAQ entries based on the input text. The system highlights matching keywords to improve usability.
- **Responsive Design**: The FAQ section was made fully responsive using Tailwind CSS, ensuring ease of use across all devices.

1.12 Gift Card Purchase and Redemption

- **Purchase**: Users can purchase gift cards by entering an amount. The balance updates automatically, and the transaction is recorded in the usage history.
- **Redeem**: Gift cards can be redeemed using a valid code, with a predefined amount deducted from the balance. Alerts notify users of successful redemptions or errors.
- **History**: A usage history log displays all purchase and redemption transactions for transparency.

2. Challenges Faced and Solutions Implemented

2.1. React Icons Display Issue

- **Problem**: React icons were not rendering correctly in the React project.
- **Solution**: The issue was resolved by ensuring the proper **React Icons** package was installed and imported correctly into the components. No additional setup or style imports were needed.

2.2. Toast Notifications Issue

- **Problem**: Toast notifications were not displaying correctly or were not integrated smoothly into the project.
- **Solution**: The **Toast** notifications library was integrated into the e-commerce website for providing real-time feedback to users (such as for product additions or successful searches). Proper configurations were set up to ensure notifications were shown in the correct position and style, enhancing the user experience.

2.3. Responsive Design and Mobile Compatibility

- **Problem**: Ensuring the design was fully responsive across various screen sizes was initially challenging, particularly when dealing with image sizes and card layouts.
- **Solution**: The issue was addressed by using **Tailwind CSS**'s responsive grid system and **flexbox**. This ensured that components resized and rearranged properly on different devices.

3. Best Practices Followed During Development

3.1. Component-based Architecture

- The website was built with a **component-based architecture**, allowing for reusability and modularity. This approach ensures that components like product cards, search inputs, and filters can be easily reused and maintained.

3.2. Responsive Design

- **Tailwind CSS** was used to implement a **mobile-first, responsive design**. Components were built to adapt seamlessly across devices by using utility-first CSS classes, ensuring a great user experience on desktops, tablets, and mobile phones.

3.3. Code Quality and Readability

- Clean and readable code was prioritized throughout the development process. **TypeScript** was used to enforce type safety, minimizing runtime errors and ensuring a more robust codebase.
- Functions and components were kept small and focused on a single responsibility, following the **Single Responsibility Principle (SRP)**.

3.4. Version Control and Collaboration

- **Git** was used for version control, allowing easy tracking of changes and enabling collaboration. Branching strategies were followed to ensure that features were developed in isolation and merged only after thorough testing.

3.5. Performance Optimization

- Data fetching was optimized to avoid unnecessary re-renders and reduce the amount of data fetched from external sources. Asynchronous operations were handled efficiently to ensure the UI remains responsive.

Conclusion

The development of the e-commerce website has progressed successfully, with core functionalities like product filtering, add to cart or wishlist and dynamic routing integrated. Challenges such as **React icon display issues**, **dynamic routing**, and **responsive design** were effectively addressed using appropriate solutions. By following best practices such as **component-based architecture**, **responsive design**, and **performance optimization**, the website is positioned for further enhancements and scalability.