Project Report: E-commerce Shop for Handmade Crochet Items

Subject: Advanced Web Design
Professor: Boban Joksimoski

Ivana Koceva 211045
Ana Janeska 211008

Table of Contents

1. Introduction

This project report details the development of an e-commerce shop specializing in handmade crochet items. The platform is built using React for the frontend, Firebase for the backend, and Redux for state management. The primary goal is to create a seamless and user-friendly online shopping experience where users can browse through different categories, search for specific products, and manage their shopping carts effectively.

2. Project Objectives

The main objectives of this project are:

- To develop a responsive and intuitive e-commerce platform for handmade crochet items.
- To integrate a robust backend using Firebase for data storage and user authentication.
- To implement efficient state management using Redux for a smooth user experience.
- To enable users to filter products by categories and perform searches.
- To provide functionalities for adding and removing items from the shopping cart.

3. System Architecture

The system architecture is divided into three main components: the frontend, the backend, and state management. Each component plays a crucial role in ensuring the platform's functionality and performance.

Frontend

The frontend of the application is developed using React, a popular JavaScript library for building user interfaces. React's component-based architecture allows for reusable and maintainable code. The key components of the frontend include:

- **Product List Component:** Displays a list of products fetched from the backend.
- **Product Detail Component:** Shows detailed information about a selected product.
- **Category Filter Component:** Allows users to filter products by categories.
- **Search Component:** Enables users to search for specific products.
- **Shopping Cart Component:** Manages the items added to the shopping cart.

Backend

The backend is powered by Firebase, a platform that offers a variety of services including real-time databases, authentication, and hosting. The Firebase Realtime Database is used to store product information and user data. Key features of the backend include:

- **Data Storage:** Storing product details, user information, and order history.
- **User Authentication:** Providing secure authentication methods (email/password, Google Sign-In).
- **Real-time Updates:** Ensuring real-time synchronization of data across clients.

State Management

Redux is used for state management to maintain a predictable and consistent application state. Redux helps in managing the state of the shopping cart, user authentication status, and product filtering/search criteria. The key aspects of state management include:

- **Actions:** Define what changes need to be made to the state.
- **Reducers:** Specify how the application's state changes in response to actions.
- **Store:** Holds the entire state tree of the application.

4. Features and Functionality

User Interface

The user interface is designed to be clean, modern, and user-friendly. It provides easy navigation through different sections of the shop, including product listings, product details, and the shopping cart. The responsive design ensures that the platform is accessible on various devices, including desktops, tablets, and smartphones.

Product Filtering and Search

Users can filter products by categories such as hats, scarves, and blankets. The filtering functionality is implemented using dropdown menus that dynamically update the product list based on the selected category. Additionally, the search functionality allows users to find specific products by entering keywords into a search bar. This is achieved through a combination of React state management and Firebase queries.

Shopping Cart Management

The shopping cart functionality enables users to add and remove items from their cart. The cart maintains the state of added items and updates the total price accordingly. Users can view their cart at any time, adjust quantities, and proceed to checkout. The cart's state is managed using Redux, ensuring consistency across the application.

5. Implementation Details

Frontend Development with React

The frontend of the application is implemented using React. The following steps outline the development process:

1. **Setting Up the React Project:** The project was initialized using Create React App, which provides a boilerplate setup for React applications.
2. **Creating Components:** Key components such as ProductList, ProductDetail, CategoryFilter, SearchBar, and ShoppingCart were created. Each component was designed to be reusable and maintainable.
3. **Styling Components:** CSS modules and styled-components were used to style the components, ensuring a consistent and responsive design across the application.
4. **Integrating with Firebase:** The Firebase SDK was integrated into the React project to fetch data from the Firebase Realtime Database and handle user authentication.

Backend Development with Firebase

Firebase was chosen as the backend for its real-time database capabilities and ease of integration with frontend frameworks. The following steps outline the backend development process:

1. **Setting Up Firebase:** A Firebase project was created, and the necessary configurations (database rules, authentication methods) were set up.
2. **Storing Product Data:** Product information was stored in the Firebase Realtime Database, structured in a way that allows efficient querying and retrieval.
3. **Implementing Authentication:** Firebase Authentication was used to handle user sign-up, login, and logout functionalities. Multiple authentication methods were provided to enhance user convenience.
4. **Real-time Data Synchronization:** Firebase's real-time capabilities were leveraged to ensure that any changes in the database (e.g., new products added) were immediately reflected in the frontend.

State Management with Redux

Redux was used to manage the state of the application. The following steps outline the state management implementation:

1. **Setting Up Redux:** Redux was integrated into the React project, and a store was created to hold the application's state.
2. **Creating Actions and Reducers:** Actions and reducers were defined for managing the state of the shopping cart, user authentication, and product filtering/search criteria.
3. **Connecting Components to the Store:** React-Redux was used to connect React components to the Redux store, enabling them to access and update the state as needed.

4. **Implementing Middleware:** Redux Thunk was used as middleware to handle asynchronous actions, such as fetching data from Firebase and updating the state accordingly.

6. Challenges and Solutions

During the development of the e-commerce shop, several challenges were encountered:

Challenge 1: Real-time Data Synchronization

Solution: Leveraging Firebase's real-time capabilities ensured that data changes were immediately reflected across all clients. The integration of Firebase's onSnapshot method allowed for real-time updates in the frontend.

Challenge 2: State Management Complexity

Solution: Redux was implemented to manage the application's state effectively. By defining clear actions and reducers, the complexity of state management was reduced, ensuring a predictable state throughout the application.

Challenge 3: User Authentication

Solution: Firebase Authentication provided a secure and straightforward solution for user authentication

. Implementing multiple authentication methods, such as email/password and Google Sign-In, enhanced the user experience by offering flexible login options.

Challenge 4: Responsive Design

Solution: CSS modules and styled-components were used to ensure a consistent and responsive design. Media queries were implemented to adjust the layout for various screen sizes, ensuring accessibility on desktops, tablets, and smartphones.

Challenge 5: Efficient Product Filtering and Search

Solution: The product filtering and search functionality were optimized by structuring the Firebase database in a way that allows efficient querying. Indexing and categorization of products enabled fast and accurate search results.

7. Future Enhancements

Several future enhancements are planned to improve the e-commerce platform:

Enhancement 1: Enhanced Product Recommendations

Implementing a recommendation system that suggests products based on user behavior and purchase history. Machine learning algorithms could be used to analyze user data and provide personalized recommendations.

Enhancement 2: Advanced Analytics

Integrating advanced analytics to track user interactions, sales performance, and other key metrics. This data can help in making informed decisions to enhance the platform and increase sales.

Enhancement 3: Improved User Profile Management

Expanding the user profile section to include order history, wishlist, and personalized settings. This would improve the overall user experience by providing more control over their shopping activities.

Enhancement 4: Multiple Payment Options

Integrating additional payment gateways to offer users more choices during checkout. This could include options like PayPal, Apple Pay, and various credit/debit card processors.

Enhancement 5: Mobile Application

Developing a mobile application for both iOS and Android platforms to provide a more accessible and convenient shopping experience for mobile users. The mobile app would be built using React Native to leverage the existing React knowledge base.

Enhancement 6: Enhanced Security Features

Implementing additional security measures such as two-factor authentication (2FA), enhanced encryption for data storage, and regular security audits to ensure the safety of user data.

8. Conclusion

The development of the e-commerce shop for handmade crochet items successfully achieved its primary objectives of creating a user-friendly platform with robust backend support and efficient state management. React, Firebase, and Redux proved to be effective technologies in building a responsive and functional e-commerce application.

The project's architecture, including the modular React components, real-time data synchronization with Firebase, and state management with Redux, provided a solid foundation for future enhancements and scalability. The challenges encountered during development were addressed with well-thought-out solutions, ensuring a smooth and seamless user experience.

Looking forward, the planned future enhancements will further enrich the platform, making it more user-centric and competitive in the e-commerce market. The project demonstrates the potential of combining modern web development technologies to create a feature-rich and scalable e-commerce solution.

Appendices

Appendix A: Code Snippets

Example of a React Component for Product List:

```
import styled from 'styled-components';
import { signOut } from "firebase/auth";
import { auth } from "../../firebase";
import { useNavigate } from "react-router";
import { useState } from "react";
import ProductCardComponent from "../../components/ProductCardComponent";
import { useSelector } from "react-redux";
import SubheadingComponent1 from "../../components/SubheadingComponent1";
const Products = () => {
    const {items}= useSelector(state=>state.products)
    const navigate = useNavigate();
    const [search, setSearch] =useState('');
    const [selectedCategory, setSelectedCategory]=useState('All')
    function handleAdd(e){
        setSelectedCategory(e.target.value)
     }
```

```
    return(
    <div className="container px-5">
        <div className="container my-5">
            <SubheadingComponent1 text="All Products"
className="my-3"></SubheadingComponent1>
            <ProductFilterWrapper className='d-flex justify-content-end'>
                <form>
                    <SearchFilterContainer placeholder='Search products'
                    onChange={(e) => setSearch(e.target.value)}>
                    </SearchFilterContainer>
                </form>
                <CategoryFilterContainer name="filter"
onChange={handleAdd}>
                    <option value="All">All</option>
                    <option value="Home">Home</option>
                    <option value="Accessory">Accessory</option>
                    <option value="Clothing">Clothing</option>
                </CategoryFilterContainer>
            </ProductFilterWrapper>
            <ProductsWrapper className='row'>
                {items
                .filter((item) => {
                    return search.toLowerCase() === ''
                    ? item
```

```
                    : item.name.toLowerCase().includes(search)

                })

                .filter((item)=>{

                    if(selectedCategory==='All')

                    return true

                    else

                    return item.category===selectedCategory

                })

                .map((product) => (

                    <ProductCardComponent key={product.id}
data={product}/>

                ))}

            </ProductsWrapper>

        </div>

    </div>

    );

};

export default Products;
```

Custom Hook for Fetching Products:

```
import {db} from "./firebase"

import { useEffect, useState, useMemo } from "react";

import {getDocs,collection} from 'firebase/firestore'
```

```
const useGetProducts=()=>{

    const [productList,setProductList]=useState([])

    const productCollectionRef=useMemo(() => collection(db, "Product"),
[]);

    useEffect(()=>{

        const getProductList= async()=>{

            try{

                const data= await getDocs(productCollectionRef)

                const filteredData=data.docs.map((doc)=>({

                    ...doc.data(),

                    id:doc.id

                }))

                setProductList(filteredData)

            }catch (err){

                console.log(err)

            }

        }

        getProductList();

    },[])



    return {productList}

}

export default useGetProducts
```

Redux Reducer for Products:

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";

import { db } from "../firebase";

import { collection, getDocs } from 'firebase/firestore';

export const fetchProducts = createAsyncThunk('products/fetchProducts',
async () => {

    const productCollectionRef = collection(db, "Product");

    const data = await getDocs(productCollectionRef);

    return data.docs.map(doc => ({ id: doc.id, ...doc.data() }));

});

const initialState = {

    items: [],

    status: null

}

const productsSlice = createSlice({

    name: "products",

    initialState,

    reducers:{},

    extraReducers:(builder) => {

        builder

            .addCase(fetchProducts.pending, (state) => {

                state.status = 'loading';

            })

            .addCase(fetchProducts.fulfilled, (state, action) => {
```

```
                state.status = 'succeeded';

                state.items = action.payload;

            })

            .addCase(fetchProducts.rejected, (state, action) => {

                state.status = 'failed';

                state.error = action.error.message;

            });

    }

})

export default productsSlice.reducer
```

Firebase Configuration:

```
import { initializeApp } from "firebase/app";

import { getAuth } from "firebase/auth";

import { getFirestore } from "firebase/firestore";

const firebaseConfig = {

  apiKey: "AIzaSyBZInTvInl2Q_4AYB4bTxoQhiJwrWOAULw",

  authDomain: "online-shop-app-78041.firebaseapp.com",

  projectId: "online-shop-app-78041",

  storageBucket: "online-shop-app-78041.appspot.com",

  messagingSenderId: "1062424892801",

  appId: "1:1062424892801:web:bed582d6268f1c8f268480"

};
```

```
const app = initializeApp(firebaseConfig);

export const auth = getAuth();

export const db = getFirestore(app);
```

Appendix B: Firebase Database Structure, Set-up and Authentication

```
- products

  - productId

    - name: "Pillow"

    - description: "A warm and cozy handmade crochet pillow."

    - price: 500

    - category: "Home"

    - imageUrl: "url_to_image"

- users

  - userId

    - email: "user@example.com"

    - displayName: "John Doe"

    - shoppingCart uid
```

This report provides a comprehensive overview of the e-commerce project for handmade crochet items, detailing the architecture, implementation, challenges, solutions, and future enhancements. It demonstrates the effective use of modern web development technologies to create a robust and user-friendly e-commerce platform.

Connecting to the database

In the development of our online shop application, establishing a reliable and scalable backend was crucial for storing and managing data. We selected Firebase Firestore due to its real-time capabilities, ease of integration with React, and comprehensive documentation. This section provides a detailed explanation of the process of setting up and connecting the Firestore database, focusing on the essential tasks and considerations involved.

Setting Up Firebase

The initial step involved configuring Firebase for our project. This required creating a Firebase configuration object containing essential details such as the API key, authentication domain, project ID, storage bucket, messaging sender ID, and application ID. These details are necessary to initialize Firebase and connect our application to the Firebase backend. With the configuration object ready, we proceeded to initialize Firebase. This was accomplished using the `initializeApp` function from the Firebase SDK. Additionally, we initialized Firestore and Firebase Authentication services. These initialized services were then exported for use in other parts of the application, enabling seamless integration of Firebase functionalities.

Designing the Database Schema

Designing an efficient database schema was crucial for the application's performance and scalability. We needed to handle various types of data, including product information as well as user data. Given that Firestore is a NoSQL database, it allows for flexible and hierarchical data structures. We structured our database to efficiently store and retrieve data, ensuring quick access and manipulation.

Products Collection:

This collection stores detailed information about each product, such as the product name, description, price, image and category which it belongs to. The structure was designed to facilitate easy updates and retrieval of product data, essential for displaying product listings and details on the frontend.

Users Collection:

This collection stores user-related information, including authentication details and user-specific data like the users display name, email and uid. It was crucial to securely store and manage user data to provide personalized user experiences and maintain data integrity.

Integrating Firestore with React

To integrate Firestore with our React application, we installed the necessary Firebase SDK packages. These packages enabled us to utilize Firestore's services within our React components. We then wrote functions to perform operations in Firestore and then integrated these functions into various React components, allowing them to interact with the Firestore database seamlessly. For instance, fetching products from the database involved querying the Firestore collection, retrieving the documents, and mapping them to a format suitable for the frontend.

User Authentication

User authentication is a crucial aspect of any web application, especially for an online shop where sensitive user data, such as personal details need to be securely managed. For our online

shop application, we implemented user authentication using Firebase Authentication, which offers various methods to authenticate users, including email/password authentication and third-party providers like Google. This section details the implementation of user authentication, focusing on email/password and Google authentication.

Email and Password Authentication

To get started with Firebase Authentication, we first initialized Firebase and Firestore as described in the previous section. We imported the necessary functions from the Firebase Authentication SDK, such as creating a user with email and password for registration and signing in with email and password for login.

For user registration, we created a registration component. This component included a form that allowed users to enter their email, password, and display name. The form submission triggered a function that handled the registration process. The process involved creating a new user with the provided email and password using Firebase's authentication service. Upon successful creation, the user's details were stored in Firestore under the users' collection. This allowed us to maintain additional user information, such as the display name and shopping cart, which could be expanded with more fields as needed.

The login component facilitated user login with email and password. Similar to the registration component, it included a form where users could input their email and password. The form submission triggered a function that handled the login process. This function used Firebase's authentication service to sign in the user with the provided email and password. Upon successful authentication, the user was redirected to the homepage, and the application state was refreshed to reflect the logged-in status.

Google Authentication

In addition to email/password authentication, we implemented Google authentication to provide users with a convenient and quick login option. This involved using the Google Authentication Provider from Firebase Authentication.

The login component also supported Google login through a dedicated function. This function initiated the Google login process using a popup window. If successful, the user was authenticated and redirected to the homepage. This method provided a seamless and familiar login experience for users with Google accounts.

Form Validation and Error Handling

We incorporated basic form validation and error handling to enhance the user experience. For instance, if a user attempted to register with an invalid email or weak password, appropriate error messages were displayed. Similarly, during login, incorrect email/password combinations triggered error messages to inform the user of the issue.

Managing user states was crucial for ensuring a seamless user experience. Upon successful registration or login, user details were stored in the application's state, enabling personalized features like displaying the user's name or accessing their shopping cart. This state management was integrated with the application's routing, ensuring that users were appropriately redirected to the homepage or relevant sections after authentication.

Additionally, security was a top priority in implementing authentication. Firebase Authentication provides robust security features, including email verification, password strength checks, and multi-factor authentication. These features were leveraged to ensure that user accounts were secure. Additionally, Firestore security rules were configured to restrict access to user data, allowing only authenticated users to read or write their own data.

In conclusion, setting up and connecting the Firestore database was a foundational component of our online shop application. By leveraging Firestore's robust features, we were able to create a scalable and real-time backend solution that seamlessly integrated with our React frontend. This provided a solid foundation for the application's functionality, ensuring data integrity, security, and performance.

Appendix C: Responsive Components and Routing

Routing

In our e-commerce application, we implemented a comprehensive routing system using React Router to facilitate seamless navigation across different sections of the application. This routing setup ensures that users can easily access various pages such as the home page, product listings, individual product details, user registration, login, shopping cart, and more. By encapsulating the routes within a `BrowserRouter` component, we established a clear and maintainable structure for our application's navigation.

The primary routes are defined within the `Routes` component, which includes a variety of paths tailored to different user needs. For instance, the root path (`'/'`) directs users to the home page, while `/products` displays the main product listing page. Individual product details are accessible via dynamic routes like `/product/:id`, which leverage URL parameters to fetch and display information about specific products.

Additionally, we included routes for user authentication, such as `/register` and `/login`, enabling users to create accounts and log in securely. The shopping cart functionality is accessible through the `/shopping-cart` route, providing users with an intuitive way to manage their selected items. We also implemented specialized routes for different product categories, such as `/products/home`, `/products/clothes`, and `/products/accessories`, enhancing the user experience by allowing them to browse specific types of products easily.

To handle invalid routes gracefully, we included a fallback route that displays a `NotFound` component, ensuring users receive appropriate feedback when they attempt to access non-existent pages. The integration of `Header` and `Footer` components within the routing structure maintains a consistent layout across all pages, while the use of context (`ShopContextProvider`) ensures state management is efficiently handled throughout the application. This robust routing system not only enhances navigability but also contributes significantly to the overall user experience.

Components and Responsiveness

The components within the given React project are designed with a high level of responsiveness and interactivity, achieved through the use of styled-components, React hooks, Redux (for state management) and Bootstrap. These components form the core structure of a responsive e-commerce platform, ensuring a seamless user experience across different devices and screen sizes.

Header Component

Starting with the `HeaderComponent`, it leverages styled-components to create a visually appealing and functional navigation header. The `Header` styled component defines a consistent background and subtle shadow to enhance the UI. The `HeaderLinkContainer` and `HeaderLink` components ensure the links are properly spaced and hidden on smaller screens using media queries. The logo, wrapped in the `HeaderImage` component, is prominently displayed and adapts well to different screen sizes due to its padding and border-radius properties. The `ProductButtonContainer` is used for buttons like login/logout, which are styled for both appearance and usability, making them easy to identify and interact with. The header also includes a shopping cart icon, which dynamically displays the total quantity of items in the cart by integrating with Redux through the `useSelector` hook. This dynamic display enhances user awareness of their cart status in real-time.

Footer Component

The `FooterComponent` similarly employs styled-components to create a footer that is both informative and visually cohesive. The `Footer` styled component sets the background color and uses flexbox for layout, ensuring the content is well-spaced and wraps correctly on smaller screens. Within the footer, the `FooterLinkContainer`, `FooterLinksWrapper`, `FooterTitle`, and `FooterLink` components structure the navigation and informational links. These links adjust their layout based on screen size, ensuring usability on both desktops and mobile devices. Additionally, social media icons are included using components from React Icons, linking to their respective platforms and styled to match the overall design.

Product Components

Product-related components, such as `ProductCardComponent`, `ProductDetailsComponent`, and `ProductInCartComponent`, form the core of the e-commerce functionality. The `ProductCardComponent` displays individual products with their images, names, prices, and categories. The `ProductImageContainer`, `ProductHeadingContainer`, `ProductDescriptionContainer`, and `ProductPriceContainer` styled-components ensure that each piece of product information is displayed clearly and consistently. The `ProductButtonContainer` and `DetailsButtonContainer` components provide interactive buttons for adding products to the cart and viewing product details, respectively. The use of `Link` from `react-router-dom` facilitates navigation between different views.

The `ProductDetailsComponent` further enhances the user experience by providing detailed information about each product. It employs similar styled-components to maintain consistency in design and ensures that the product details, including images, descriptions, and prices, are well-presented. The `ProductButtonContainer` is used here to add products to the cart directly from the details view, leveraging the `useDispatch` hook from Redux to handle state changes.

In the shopping cart functionality, `ProductInCartComponent` is designed to display items in the cart along with their quantities, prices, and options to remove or adjust quantities. The `ProductInCartWrapper`, `ProductInCartImage`, `ProductInCartContainer`, `ProductInCartDetailsContainer`, and `ProductInCartPrice` components structure the cart items neatly. Buttons like `ProductInCartQuantity` and `ProductButtonContainer` provide interactive elements for modifying the cart contents. Redux hooks `useDispatch` and `useSelector` manage the cart state, ensuring that any changes are reflected immediately.

Button Components

The PrimaryButtonComponent and SecondaryButtonComponent are integral to the user interface, providing visually distinct, responsive buttons for various interactions within the platform.

The `PrimaryButtonComponent` features a solid background color, rounded edges, and text styled in white. This button is typically used for primary actions such as adding products to the cart or proceeding to checkout. Its design ensures it stands out prominently, drawing users' attention to crucial actions that drive the user journey forward.

The `SecondaryButtonComponent`, on the other hand, has a transparent background with a bordered edge and a more subtle text color. It serves secondary actions like navigation links, additional options, or less urgent calls-to-action. The design of this button ensures it complements the primary button while maintaining clear visual hierarchy and usability.

Both button components adjust their padding and font size based on screen width, ensuring that they remain accessible and readable on various devices. This responsiveness, coupled with real-time state management, makes the platform both functional and visually appealing, enhancing overall user engagement and satisfaction.

In conclusion, these components, through the strategic use of styled-components, React hooks, and Redux, are the building blocks for a responsive, interactive, and cohesive e-commerce platform. Each component is designed to adapt to different screen sizes, ensuring a consistent and user-friendly experience across all devices. This responsiveness, coupled with real-time state management, makes the platform both functional and visually appealing, enhancing overall user engagement and satisfaction.