Case Study 8: Developing a Dynamic Cart Page for a Cross-Platform E-commerce Skincare App

Objective

The goal of this project was to create a fully functional cart page for a cross-platform e-commerce skincare app. The cart page enables users to review selected items, adjust quantities, apply promotional codes, view the subtotal and delivery costs, and proceed to checkout. This feature was implemented to enhance the overall user experience, streamline the purchase process, and increase conversion rates by providing an intuitive and interactive shopping experience.

Project Breakdown

Frontend Development

1. **Cart Page Layout and Styling**
    - **Responsive Design**: Using **React Native's Flexbox** layout, we created a responsive cart page that adapts seamlessly across different screen sizes. Each cart item is displayed with product information, quantity adjustment buttons, and a remove button.
    - **UI Consistency**: The styling follows a minimalist and clean approach, matching the brand's aesthetic with the Inter font, high-contrast text, and consistent button styles. Key elements, such as the checkout button and subtotal summary, are visually prominent to encourage user interaction.

javascript

```
// CartScreen.js

import React from 'react';

import { View, Text, FlatList, TouchableOpacity, Image } from 'react-native';


const CartScreen = ({ cartItems, onQuantityChange, onRemoveItem }) => {

  const renderCartItem = ({ item }) => (

    <View style={styles.cartItem}>

      <Image source={{ uri: item.image }} style={styles.productImage} />
```

```
      <View style={styles.productDetails}>
        <Text>{item.name}</Text>
        <Text>${item.price}</Text>
      </View>
      <View style={styles.quantityControl}>
        <TouchableOpacity onPress={() => onQuantityChange(item.id, -1)}>
          <Text>-</Text>
        </TouchableOpacity>
        <Text>{item.quantity}</Text>
        <TouchableOpacity onPress={() => onQuantityChange(item.id, 1)}>
          <Text>+</Text>
        </TouchableOpacity>
      </View>
      <TouchableOpacity onPress={() => onRemoveItem(item.id)}>
        <Text>🗑</Text>
      </TouchableOpacity>
    </View>
  );

  return (
    <FlatList
      data={cartItems}
      renderItem={renderCartItem}
      keyExtractor={(item) => item.id.toString()}
      contentContainerStyle={styles.container}
    />
```

```
  );
};
```

2. **Interactive Quantity Adjustment**
    - **Quantity Controls**: Implemented + and - buttons to adjust product quantities. The quantity is updated in real time and recalculates the subtotal automatically.
    - **Remove Item Button**: Each item in the cart has a delete button (trash icon) for removing it from the cart. This feature allows users to customize their cart contents easily.
3. **Promotion Code Application**
    - **Promotion Input**: Designed a promotion code input field where users can enter and apply discount codes. This field validates the entered code and updates the subtotal accordingly.
    - **Apply Button**: An "Apply" button triggers validation and checks for code validity. If valid, the discount is applied to the total cost; otherwise, an error message is displayed.
4. **Subtotal and Total Calculation**
    - **Dynamic Calculations**: The cart dynamically calculates the subtotal based on item prices and quantities. Additional charges, like delivery fees, are displayed separately, with the grand total prominently shown.
    - **Currency Formatting**: Prices and totals are formatted to include currency symbols and two decimal places for consistency.
5. **Proceed to Checkout Button**
    - **Visual Hierarchy**: The "Proceed to Checkout" button is designed with high contrast (black background and white text) to stand out, making it the primary action on the cart page.
    - **User Feedback**: This button provides immediate feedback when tapped, guiding the user to the next step in the purchasing process.

Backend Development

1. **Cart API Endpoint**
    - **GET Request**: Created an endpoint to fetch the current contents of the user's cart from the database. This includes product details, quantities, and any applied discounts.
    - **POST and PUT Requests**: Set up endpoints to handle adding and removing items from the cart, adjusting quantities, and applying promotional codes.

```
javascript

// cartRoutes.js

router.get('/cart', async (req, res) => {
  try {
    const cart = await Cart.findOne({ userId: req.user.id }).populate('items.product');
    res.json(cart);
  } catch (error) {
    res.status(500).json({ message: 'Failed to fetch cart' });
  }
});
```

2. **Promotion Code Validation**
    - **Promotion Code Database**: Created a collection in MongoDB to store active promotion codes, their discount values, and expiration dates.
    - **Validation Logic**: Implemented validation checks on the server side to ensure that only active codes can be applied. Invalid codes trigger an error message, and valid codes apply the appropriate discount to the total cost.
3. **Subtotal and Total Calculations**
    - **Backend Logic**: Calculated subtotal and total on the server to ensure accurate data when sending the cart's information to the frontend. This setup also helps prevent any manipulation from the client side.
    - **Delivery Charges**: Added logic to include delivery fees for final calculations, which is then sent to the frontend as part of the cart details.
4. **Checkout Integration**

- **Checkout API**: Created an endpoint to initiate the checkout process. This endpoint verifies the cart contents, calculates the final total, and generates a checkout session.
- **Stripe Integration**: For payment, the backend is integrated with **Stripe API** to handle secure transactions. The cart data is passed to Stripe, where a payment link or checkout session is generated.

Key Features Implemented

1. **Dynamic Cart Item Display**: Each item in the cart is displayed with an image, name, price, and quantity adjustment controls. Users can easily view and interact with each item.
2. **Promotion Code Application**: A promotion code feature allows users to apply discounts to their cart. The app verifies the validity of codes and updates the cart total accordingly.
3. **Automatic Subtotal and Delivery Calculation**: The cart updates the subtotal in real time as users adjust quantities or add/remove items. Delivery fees are displayed separately, with the grand total shown prominently.
4. **Secure Checkout with Stripe**: Integrated secure payment handling through Stripe, allowing users to proceed to checkout with confidence. This feature ensures that the entire cart and payment process is streamlined and secure.

Challenges and Solutions

1. **Maintaining Real-Time Updates Across the Cart**
    - **Solution**: Used **React Native's useState** and **useEffect** hooks to manage and update the cart's state dynamically. The cart recalculates totals whenever items are added, removed, or adjusted.
2. **Validating and Applying Promotion Codes**
    - **Solution**: Implemented backend validation logic to check promotion codes against a database of active codes. This approach ensures that only valid codes can be applied, and any discounts are reflected in the total.
3. **Cross-Platform Styling Consistency**

- **Solution**: Ensured a consistent design on both iOS and Android by leveraging React Native's StyleSheet API. Careful testing on both platforms helped identify and resolve styling inconsistencies.

Tools and Technologies Used

- **Frontend**:
  - **React Native**: Built the cart interface and interactive components, making the app compatible with both iOS and Android.
  - **Axios**: Used for API requests to fetch and update cart data, apply promotion codes, and initiate checkout.
  - **React Navigation**: Implemented smooth transitions between the cart and other sections of the app.
- **Backend**:
  - **Node.js and Express**: Set up API endpoints to manage cart items, apply promotions, and handle checkout.
  - **MongoDB and Mongoose**: Stored user cart data, product details, and promotion codes, ensuring quick retrieval and secure storage.
  - **Stripe API**: Integrated Stripe for secure payments, enabling users to complete purchases directly from the app.

Outcomes and Results

1. **Enhanced User Engagement**: The cart's interactive design, with features like quantity adjustment and promotion code application, provided users with a streamlined shopping experience. This feature was well-received by users during testing, who appreciated the ease of use.
2. **Increased Conversion Rates**: By including a prominent "Proceed to Checkout" button and secure payment processing with Stripe, the app made it easier for users to complete their purchases, leading to higher conversion rates.
3. **Positive Feedback on UX**: The clean, intuitive layout of the cart page received positive feedback. Users found the design simple to navigate and appreciated the quick, real-time updates.

Lessons Learned and Future Enhancements

1. **Enhanced Personalization**: Future improvements could involve personalized recommendations within the cart page, such as cross-sell or up-sell suggestions based on cart items.
2. **A/B Testing for Promotion Code Placement**: Conducting A/B testing could help determine the best placement for the promotion code input to maximize user engagement with discount offers.
3. **User-Specific Discounts**: Adding features like user-specific discounts and loyalty rewards would enhance engagement and incentivize repeat purchases.