Day 4 - Dynamic Frontend Components - Quick Commerce Coffee Shop

Overview

In today's assignment, we focused on integrating necessary components into the marketplace builder. My project, **Quick Commerce Coffee Shop**, utilizes ten components, five of which were pre-built (from Day 3), while the remaining five were developed today. The new components include:

1. Cart Component
2. Notification Component
3. Checkout Flow Component
4. Reviews and Ratings Component
5. Customer Feedback Component

This document details the implementation of the **Cart Component**, including the design of its pages, context management, and functionality.

Cart Component

The **Cart Component** handles the shopping cart functionality, comprising:

1. **Cart Page**: Displays items in the cart and allows interaction.
2. **Cart Context**: Manages the state and operations of the cart.
3. **Cart Icon in Navbar**: Provides quick access to the cart.
4. **Add to Cart Button**: Integrated into the dynamic product display.

1. Cart Page

The `CartPage` component serves as the main interface for managing cart items. Users can view, update, and remove items from the cart, proceed to checkout, and view success messages.

Code Snippet: `CartPage` Component

```
'use client';

import React, { useState } from 'react';
import Image from 'next/image';
import { useCart } from '../../components/CartContext';
import Navbar from '@/components/Navbar';
import ResponsiveNavbar from '@/components/ResponsiveNavbar';

export default function CartPage() {
  const { items, removeFromCart, updateQuantity } = useCart();
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [formData, setFormData] = useState({
    fullName: '',
    phoneNumber: '',
    address: '',
    paymentMethod: 'cashOnDelivery'
  });
  const [showSuccess, setShowSuccess] = useState(false);

  const total = items.reduce(
    (sum, item) => sum + (item.dicountPercentage
      ? item.price * (1 - item.dicountPercentage / 100)
      : item.price) * item.quantity,
    0
  );

  const handleInputChange = (e: React.ChangeEvent<HTMLInputElement>) =>
{
    const { name, value } = e.target;
    setFormData(prev => ({
      ...prev,
      [name]: value
    }));
  };

  const handleSubmit = (e: React.FormEvent) => {
```

```
    e.preventDefault();
    setIsModalOpen(false);
    setShowSuccess(true);
    setTimeout(() => {
      setShowSuccess(false);
    }, 3000);
  };

  return (
    <div className="container mx-auto px-4 py-12 bg-white relative">
      <section className="navbar-section mb-8">
        <Navbar />
      </section>

      <section className="responsive-navbar-section">
        <ResponsiveNavbar />
      </section>

      <h1 className="text-3xl font-bold mb-8 text-black">Shopping
Cart</h1>

      {items.length === 0 ? (
        <div className="text-center py-12">
          <p className="text-gray-500">Your cart is empty</p>
        </div>
      ) : (
        <div className="grid grid-cols-1 gap-6 text-black">
          {items.map((item) => {
            const itemPrice = item.dicountPercentage
              ? item.price * (1 - item.dicountPercentage / 100)
              : item.price;

            return (
              <div
                key={item._id}
                className="flex gap-4 border rounded-lg p-4"
              >
                <div className="relative w-24 h-24">
                  <Image
                    src={item.productImage}
                    alt={item.title}
                    fill
                    className="object-cover rounded-lg"
```

```
                  />
                </div>

                <div className="flex-1">
                  <h3 className="text-lg
font-semibold">{item.title}</h3>
                  <p className="text-gray-600">
                    ${itemPrice.toFixed(2)} x {item.quantity}
                  </p>
                  <div className="mt-2 flex items-center gap-2">
                    <button
                      onClick={() => updateQuantity(item._id,
item.quantity - 1)}
                      className="px-2 py-1 border rounded"
                    >
                      -
                    </button>
                    <span>{item.quantity}</span>
                    <button
                      onClick={() => updateQuantity(item._id,
item.quantity + 1)}
                      className="px-2 py-1 border rounded"
                    >
                      +
                    </button>
                    <button
                      onClick={() => removeFromCart(item._id)}
                      className="ml-4 text-red-500"
                    >
                      Remove
                    </button>
                  </div>
                </div>

                <div className="text-right">
                  <p className="font-semibold">
                    ${(itemPrice * item.quantity).toFixed(2)}
                  </p>
                </div>
              </div>
            );
          })}
```

```
          <div className="mt-6 text-right">
            <p className="text-xl font-bold mb-4">
              Total: ${total.toFixed(2)}
            </p>
            <button
              onClick={() => setIsModalOpen(true)}
              className="bg-blue-500 text-white px-6 py-3 rounded-lg
hover:bg-blue-600 transition-colors"
            >
              Proceed to Checkout
            </button>
          </div>
        </div>
      )}

      {/* Checkout Modal */}
      {isModalOpen && (
        <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center z-50">
          <div className="bg-white p-8 rounded-lg w-full max-w-md">
            <h2 className="text-2xl font-bold mb-6">Checkout</h2>
            <form onSubmit={handleSubmit} className="space-y-4">
              <div>
                <label className="block text-sm font-medium
text-gray-700 mb-1">
                  Receiver&amp;apos;s Full Name
                </label>
                <input
                  type="text"
                  name="fullName"
                  required
                  value={formData.fullName}
                  onChange={handleInputChange}
                  className="w-full p-2 border rounded-md"
                />
              </div>

              <div>
                <label className="block text-sm font-medium
text-gray-700 mb-1">
                  Phone Number
                </label>
                <input
```

```
                type="tel"
                name="phoneNumber"
                required
                value={formData.phoneNumber}
                onChange={handleInputChange}
                className="w-full p-2 border rounded-md"
              />
            </div>

            <div>
              <label className="block text-sm font-medium
text-gray-700 mb-1">
                Delivery Address
              </label>
              <input
                type="text"
                name="address"
                required
                value={formData.address}
                onChange={handleInputChange}
                className="w-full p-2 border rounded-md"
              />
            </div>

            <div>
              <label className="block text-sm font-medium text-black
mb-1">
                Payment Method
              </label>
              <div className="flex items-center space-x-2
text-black">
                <input
                  type="radio"
                  name="paymentMethod"
                  value="cashOnDelivery"
                  checked={formData.paymentMethod ===
'cashOnDelivery'}
                  onChange={handleInputChange}
                />
                <span>Cash on Delivery</span>
              </div>
            </div>
```

```
            <div className="border-t pt-4 mt-4">
              <p className="text-lg font-semibold">
                Total Amount: ${total.toFixed(2)}
              </p>
            </div>

            <div className="flex gap-4 mt-6">
              <button
                type="button"
                onClick={() => setIsModalOpen(false)}
                className="flex-1 px-4 py-2 border rounded-md
hover:bg-gray-100"
              >
                Cancel
              </button>
              <button
                type="submit"
                className="flex-1 px-4 py-2 bg-blue-500 text-white
rounded-md hover:bg-blue-600"
              >
                Confirm Order
              </button>
            </div>
          </form>
        </div>
      </div>
    )}

    {/* Success Message */}
    {showSuccess && (
      <div className="fixed inset-0 flex items-center justify-center
z-50">
        <div className="bg-white p-6 rounded-lg shadow-lg">
          <p className="text-lg font-semibold text-green-600">
            Order placed successfully! Your order will arrive in 30
mins.
          </p>
        </div>
      </div>
    )}
  </div>
 );
}
```

2. Cart Context

The `CartContext` handles state management for the cart, including functions for adding, removing, and updating cart items.

Code Snippet: `CartContext`

```
"use client"

import React, { createContext, useContext, useState, useEffect } from
'react';

// Define the Product interface
interface Product {
  _id: string;
  productImage: string;
  title: string;
  price: number;
  description: string;
  tags: string[];
  dicountPercentage?: number;
  isNew?: boolean;
}

interface CartItem extends Product {
  quantity: number;
}

interface CartContextType {
  items: CartItem[];
  addToCart: (product: Product) => void;
  removeFromCart: (productId: string) => void;
  updateQuantity: (productId: string, quantity: number) => void;
  cartCount: number;
}

const CartContext = createContext<CartContextType |
undefined>(undefined);
```

```
export function CartProvider({ children }: { children: React.ReactNode
}) {
  const [items, setItems] = useState<CartItem[]>([]);
  const [cartCount, setCartCount] = useState(0);

  // Load cart from localStorage on initial render
  useEffect(() => {
    const savedCart = localStorage.getItem('cart');
    if (savedCart) {
      setItems(JSON.parse(savedCart));
    }
  }, []);

  // Save cart to localStorage whenever it changes
  useEffect(() => {
    localStorage.setItem('cart', JSON.stringify(items));
    setCartCount(items.reduce((total, item) => total + item.quantity,
0));
  }, [items]);

  const addToCart = (product: Product) => {
    setItems(prev => {
      const existingItem = prev.find(item => item._id === product._id);
      if (existingItem) {
        return prev.map(item =>
          item._id === product._id
            ? { ...item, quantity: item.quantity + 1 }
            : item
        );
      }
      return [...prev, { ...product, quantity: 1 }];
    });
  };

  const removeFromCart = (productId: string) => {
    setItems(prev => prev.filter(item => item._id !== productId));
  };

  const updateQuantity = (productId: string, quantity: number) => {
    if (quantity < 1) {
      removeFromCart(productId);
      return;
    }
```

```
    setItems(prev =>
      prev.map(item =>
        item._id === productId ? { ...item, quantity } : item
      )
    );
  };

  return (
    <CartContext.Provider
      value={{ items, addToCart, removeFromCart, updateQuantity,
cartCount }}
    >
      {children}
    </CartContext.Provider>
  );
}

export function useCart() {
  const context = useContext(CartContext);
  if (context === undefined) {
    throw new Error('useCart must be used within a CartProvider');
  }
  return context;
}
```

Key Features

1. **Dynamic Item Management**: Add, remove, and update cart items with live updates.
2. **Persistent State**: Cart data is stored in `localStorage` for persistence.
3. **Checkout Modal**: Captures user information and simulates order placement.
4. **Responsive Design**: Optimized for various screen sizes with Tailwind CSS.

3. Integration of the Review Component

In this section, we describe the integration of the **Review Component** into the project. The Review Component enhances the interactivity of the website by enabling users to submit and view reviews for products dynamically. Below is an overview of the process:

Purpose of the Review Component

The Review Component is designed to provide a seamless experience for users to share their opinions, rate products, and engage with other users' reviews. Key features include:

1. Displaying a list of reviews with user ratings, timestamps, and like counts.
2. Allowing users to submit new reviews with a star rating system.
3. Calculating and displaying the average product rating dynamically.
4. Providing interactive features, such as liking reviews and replying.

Key Functionalities

Dynamic Review Submission

- Users can submit a review by entering text and selecting a star rating (1-5).
- A real-time rating system updates the displayed average rating as new reviews are added.

Real-Time Interactivity

- Users can like reviews, encouraging community engagement.
- Reviews are displayed with timestamps (e.g., "2 hours ago") for added relevance.

User-Friendly Design

- Reviews are formatted with clean, readable layouts.

- A responsive design ensures usability across all screen sizes.

Code Snippet:

```
'use client'

import { useState, FormEvent } from 'react';
import { UserCircle, Send, Star, ThumbsUp, Reply, MoreHorizontal } from
'lucide-react';

interface Review {
  id: number;
  author: string;
  content: string;
  rating: number;
  likes: number;
  timestamp: string;
}

const ReviewSection = () => {
  const [reviews, setReviews] = useState<Review[]>([
    {
      id: 1,
      author: "John Doe",
      content: "Amazing product! The quality is outstanding and
delivery was quick.",
      rating: 5,
      likes: 5,
      timestamp: "2 hours ago"
    },
    {
      id: 2,
      author: "Jane Smith",
      content: "Good product but could be better. The size runs a bit
small.",
      rating: 4,
      likes: 3,
      timestamp: "1 hour ago"
    },
    {
      id: 3,
      author: "Mike Johnson",
```

```
      content: "Decent quality for the price. Shipping was fast.",
      rating: 3,
      likes: 2,
      timestamp: "30 minutes ago"
    }
  ]);

  const [newReview, setNewReview] = useState("");
  const [rating, setRating] = useState(0);
  const [hoveredRating, setHoveredRating] = useState(0);

  const handleReviewSubmit = (e: FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    if (!newReview.trim() || rating === 0) return;

    const review: Review = {
      id: reviews.length + 1,
      author: "Current User",
      content: newReview,
      rating: rating,
      likes: 0,
      timestamp: "Just now"
    };

    setReviews([...reviews, review]);
    setNewReview("");
    setRating(0);
  };

  const StarRating = ({ filled, hovered }: { filled: boolean; hovered:
boolean }) => (
    <Star
      className={`w-6 h-6 ${
        filled
          ? 'text-yellow-400 fill-yellow-400'
          : hovered
            ? 'text-yellow-200 fill-yellow-200'
            : 'text-gray-300'
      }`}
    />
  );

  // Calculate average rating
```

```
  const averageRating = reviews.reduce((acc, review) => acc +
review.rating, 0) / reviews.length;

  return (
    <div className="w-full max-w-4xl mx-auto py-8">
      <div className="mb-8">
        <h2 className="text-2xl font-bold">Product Reviews</h2>
        <div className="flex items-center gap-2 mt-2">
          <div className="flex">
            {[1, 2, 3, 4, 5].map((star) => (
              <StarRating
                key={star}
                filled={star <= averageRating}
                hovered={false}
              />
            ))}
          </div>
          <span className="text-lg font-semibold">
            {averageRating.toFixed(1)} out of 5
          </span>
          <span className="text-gray-500">
            ({reviews.length} reviews)
          </span>
        </div>
      </div>

      {/* Review Form */}
      <form onSubmit={handleReviewSubmit} className="mb-8">
        <div className="flex gap-4">
          <UserCircle className="w-10 h-10 text-gray-400" />
          <div className="flex-1">
            <div className="mb-4">
              <label className="block text-sm font-medium mb-2">Your
Rating</label>
              <div className="flex gap-1">
                {[1, 2, 3, 4, 5].map((star) => (
                  <button
                    key={star}
                    type="button"
                    onClick={() => setRating(star)}
                    onMouseEnter={() => setHoveredRating(star)}
                    onMouseLeave={() => setHoveredRating(0)}
                  >
```

```
                  <StarRating
                    filled={star <= (hoveredRating || rating)}
                    hovered={star <= hoveredRating}
                  />
                </button>
              ))}
            </div>
          </div>
          <textarea
            value={newReview}
            onChange={(e) => setNewReview(e.target.value)}
            className="w-full p-4 border rounded-lg resize-none h-24"
            placeholder="Write your review..."
          />
          <div className="flex justify-end mt-2">
            <button
              type="submit"
              disabled={rating === 0}
              className={`flex items-center gap-2 px-4 py-2 bg-black
text-white rounded-lg ${
                rating === 0 ? 'opacity-50 cursor-not-allowed' :
'hover:bg-gray-800'
              }`}
            >
              <Send className="w-4 h-4" />
              Post Review
            </button>
          </div>
        </div>
      </div>
    </form>

    {/* Reviews List */}
    <div className="space-y-6">
      {reviews.map((review) => (
        <div key={review.id} className="flex gap-4">
          <UserCircle className="w-10 h-10 text-gray-400" />
          <div className="flex-1">
            <div className="bg-gray-50 p-4 rounded-lg">
              <div className="flex justify-between items-start mb-2">
                <div>
                  <h3 className="font-semibold">{review.author}</h3>
                  <div className="flex items-center gap-2">
```

```
                    <div className="flex">
                      {[1, 2, 3, 4, 5].map((star) => (
                        <StarRating
                          key={star}
                          filled={star <= review.rating}
                          hovered={false}
                        />
                      ))}
                    </div>
                    <span className="text-sm text-gray-500">
                      {review.timestamp}
                    </span>
                  </div>
                </div>
                <button className="text-gray-400
hover:text-gray-600">
                  <MoreHorizontal className="w-5 h-5" />
                </button>
              </div>
              <p className="text-gray-700">{review.content}</p>
            </div>
            <div className="flex gap-4 mt-2 text-sm text-gray-500">
              <button className="flex items-center gap-1
hover:text-gray-700">
                <ThumbsUp className="w-4 h-4" />
                {review.likes}
              </button>
              <button className="flex items-center gap-1
hover:text-gray-700">
                <Reply className="w-4 h-4" />
                Reply
              </button>
            </div>
          </div>
        </div>
      ))}
    </div>
  </div>
);
};

export default ReviewSection;
```

4. Checkout Component Integration

The Checkout Component plays a pivotal role in the e-commerce experience, serving as the final step in the customer's purchasing journey. Its design and functionality ensure a seamless and user-friendly process, enhancing the overall customer satisfaction. Here's an overview of the Checkout Component:

Features and Purpose

The Checkout Component is responsible for:

1. **Order Summary Display:**
   Displays a detailed breakdown of the customer's order, including item names, quantities, prices, and the total amount to be paid.
2. **Payment Processing:**
   Supports secure payment gateways for credit/debit card transactions, digital wallets, and other payment options as needed.
3. **Shipping Information:**
   Collects and validates the customer's shipping address and preferences, ensuring accurate and timely delivery.
4. **Promotions and Discounts:**
   Includes the ability to apply promo codes, discounts, or loyalty rewards directly at checkout.
5. **Responsive Design:**
   Optimized for different screen sizes, providing a smooth experience across devices.

Implementation Details

The Checkout Component is designed with React and TypeScript, ensuring maintainability and scalability. Key functionalities include:

- **Validation:**
  Leveraging form validation techniques to ensure accurate user input for payment and shipping details.

- **Integration with Backend APIs:**
  Communicates with the backend to process payments, verify discounts, and finalize orders.
- **User Experience Enhancements:**
  Provides visual feedback during the process (e.g., loading spinners for payment processing) and error messages if issues arise.

Example Code Snippet

Below is a simplified example of how the Checkout Component might be structured:

```
{/* Checkout Modal */}

      {isModalOpen && (

        <div className="fixed inset-0 bg-black bg-opacity-50 flex
items-center justify-center z-50">

          <div className="bg-white p-8 rounded-lg w-full max-w-md">

            <h2 className="text-2xl font-bold mb-6">Checkout</h2>

            <form onSubmit={handleSubmit} className="space-y-4">

              <div>

                <label className="block text-sm font-medium
text-gray-700 mb-1">

                  Receiver&apos;s Full Name

                </label>

                <input

                  type="text"

                  name="fullName"

                  required

                  value={formData.fullName}
```

```
                onChange={handleInputChange}
                className="w-full p-2 border rounded-md"
              />
            </div>

            <div>
              <label className="block text-sm font-medium
text-gray-700 mb-1">
                Phone Number
              </label>
              <input
                type="tel"
                name="phoneNumber"
                required
                value={formData.phoneNumber}
                onChange={handleInputChange}
                className="w-full p-2 border rounded-md"
              />
            </div>

            <div>
              <label className="block text-sm font-medium
text-gray-700 mb-1">
                Delivery Address
              </label>
              <input
```

```
                  type="text"
                  name="address"
                  required
                  value={formData.address}
                  onChange={handleInputChange}
                  className="w-full p-2 border rounded-md"
                />
              </div>

              <div>
                <label className="block text-sm font-medium text-black
mb-1">
                  Payment Method
                </label>
                <div className="flex items-center space-x-2
text-black">
                  <input
                    type="radio"
                    name="paymentMethod"
                    value="cashOnDelivery"
                    checked={formData.paymentMethod ===
'cashOnDelivery'}
                    onChange={handleInputChange}
                  />
                  <span>Cash on Delivery</span>
                </div>
```

```
            </div>


            <div className="border-t pt-4 mt-4">
              <p className="text-lg font-semibold">
                Total Amount: ${total.toFixed(2)}
              </p>
            </div>


            <div className="flex gap-4 mt-6">
              <button
                type="button"
                onClick={() => setIsModalOpen(false)}
                className="flex-1 px-4 py-2 border rounded-md
hover:bg-gray-100"
              >
                Cancel
              </button>
              <button
                type="submit"
                className="flex-1 px-4 py-2 bg-blue-500 text-white
rounded-md hover:bg-blue-600"
              >
                Confirm Order
              </button>
            </div>
          </form>
```

```
          </div>
        </div>
      )}

      {/* Success Message */}
      {showSuccess && (
        <div className="fixed inset-0 flex items-center justify-center
z-50">
          <div className="bg-white p-6 rounded-lg shadow-lg">
            <p className="text-lg font-semibold text-green-600">
              Order placed successfully! Your order will arrive in 30
mins.
            </p>
          </div>
        </div>
      )}
```

Conclusion

The Quick Commerce Coffee Shop project highlights the seamless integration of dynamic frontend components to create a user-friendly and engaging e-commerce platform. Each component has been meticulously designed to enhance the shopping experience, ensuring efficiency, accessibility, and interactivity across all user interactions.

From the **Cart Component** for managing purchases to the **Checkout Flow Component** for secure and intuitive transactions, every step of the shopping journey is streamlined. The **Notification Component** ensures users are always informed, while the **Reviews and Ratings**

Component fosters trust and community engagement by encouraging valuable customer feedback.

Together, these components demonstrate the potential of modern frontend development in delivering a high-quality e-commerce solution. By combining responsive design, dynamic functionality, and user-focused features, this project lays a strong foundation for a scalable and customer-centric application.