Day 4 - Advanced Documentation on Dynamic Frontend Components - Nike

Objective:

To develop dynamic and responsive frontend components for the marketplace, integrating reusable structures, efficient data fetching, and state management while addressing challenges such as API latency, dynamic routing, and pagination.

Procedures Undertaken for Component Development and Integration:

1. Initialization and Data Acquisition:

- Established a connection between the frontend and Sanity CMS through the Sanity client, ensuring secure and efficient communication.
- Validated the structural integrity and accessibility of all data models, including **`` `Products` ``** and **`` `Categories` ``**, via API endpoints.
- Engineered reusable and scalable data-fetching functions for essential components such as `ProductList`, `CategoryFilter`, and `SearchBar`.

2. Development of Core Components:

Product Listing Component:

- Dynamically rendered product data in a grid layout optimized for responsive design.
- Leveraged card-based interfaces to display key attributes such as product name, pricing, and inventory status.

Product Detail Component:

- Utilized dynamic routing within Next.js to generate unique pages for individual product entries.
- Integrated detailed product attributes, including descriptions, pricing, and high-resolution imagery.

Category Filter Component:

- Dynamically fetched category data from APIs to facilitate product categorization ● Enabled real-time filtering of products based on user-selected categories.
  - **Search Bar**:
- Implemented advanced search functionalities to allow filtering of products via names and associated tags.
  - **Pagination Component**:
- Incorporated intuitive navigation mechanisms such as "previous" and "next" buttons to handle extensive product catalogs efficiently.

3. Styling and Adaptive Design:

- Applied Tailwind CSS to achieve a unified, aesthetically pleasing, and mobile-responsive user interface.
- Ensured adaptability of component layouts to various screen sizes through dynamic styling methodologies.

4. Global State Management:

- Adopted React Context to establish a global state management system for the cart and order confirmation functionalities
- This approach facilitated seamless communication between components and enhanced data persistence across the application.

Identified Challenges and Corresponding Solutions:

1. Challenge: API Latency and Response Delays

Issues:

- Prolonged response times during data fetching hindered component rendering efficiency.
- Encountered CORS-related errors while fetching data due to misconfigured origin settings in Sanity CMS.

Solutions:

- Incorporated a loading state and skeleton UI to provide visual feedback during data retrieval.
- Adjusted CORS configurations in Sanity CMS to whitelist the frontend's origin, enabling uninterrupted data flow.

2. Challenge: Errors in Dynamic Routing

Issue:

- Invalid or missing product IDs resulted in failures during page rendering for product details.

Solution:

- : Introduced robust error handling mechanisms and designed fallback pages to gracefully handle missing or invalid product data.

3. Challenge: Complex Filtering and Pagination Integration

Issue:

- Coordinating multiple filters (e.g., category, price range) with pagination presented challenges in maintaining state consistency.

Solution:

- Implemented URL-based query parameters to synchronize filtering and pagination states across browser reloads.

Adopted Best Practices:

Component Reusability:

- Developed modular and reusable components, including `` `ProductCard` `` and `` `CategoryFilter` ``, to promote scalability and maintainability.

Secure Configuration Management:

- Utilized `.env.local` for storing sensitive API keys, enhancing overall security and adherence to industry standards.

Error Mitigation:

- Employed comprehensive error-handling strategies to manage API failures and ensure a seamless user experience.

Responsive Design Principles:

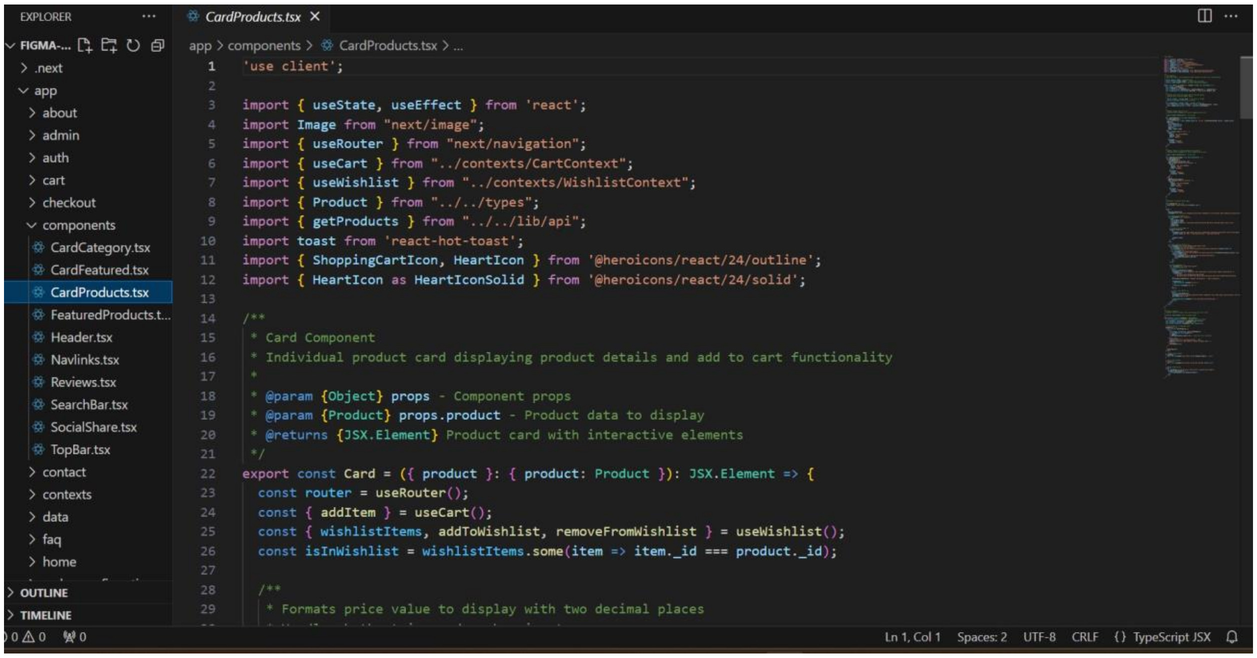
- Thoroughly tested the application across multiple device resolutions to guarantee consistent and accessible user interfaces

High-Quality Code Standards:

- Adopted descriptive naming conventions and implemented detailed code comments to facilitate readability and future development efforts.

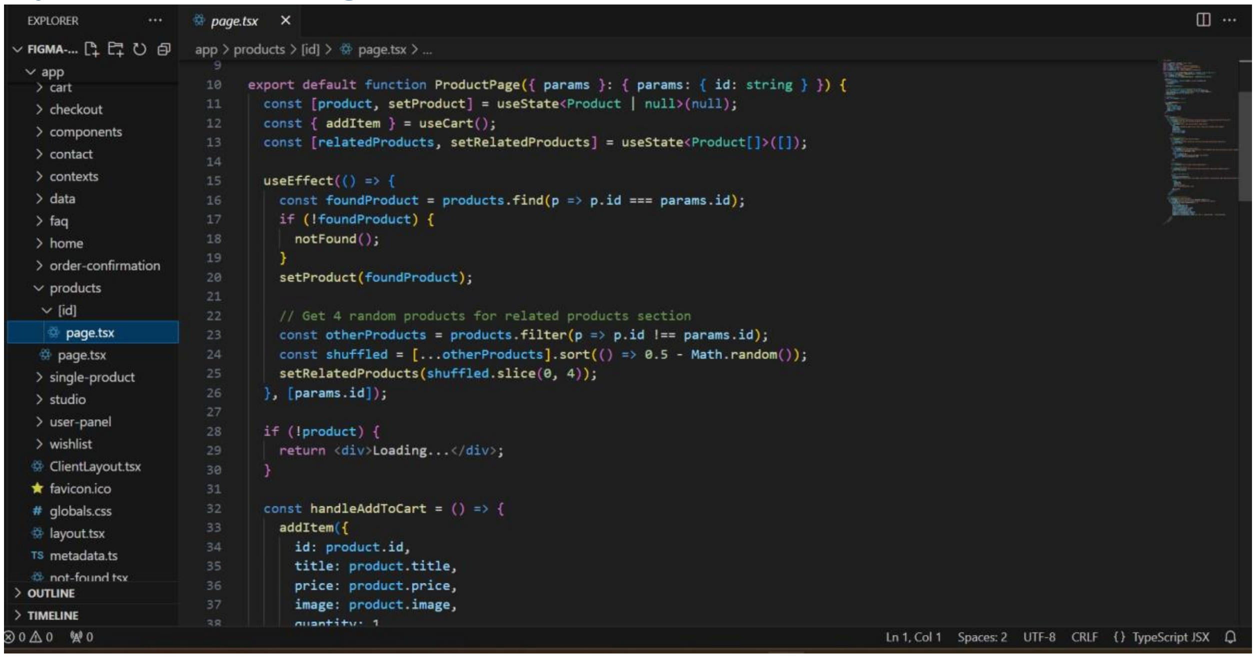
Screenshots:

Card Featured:



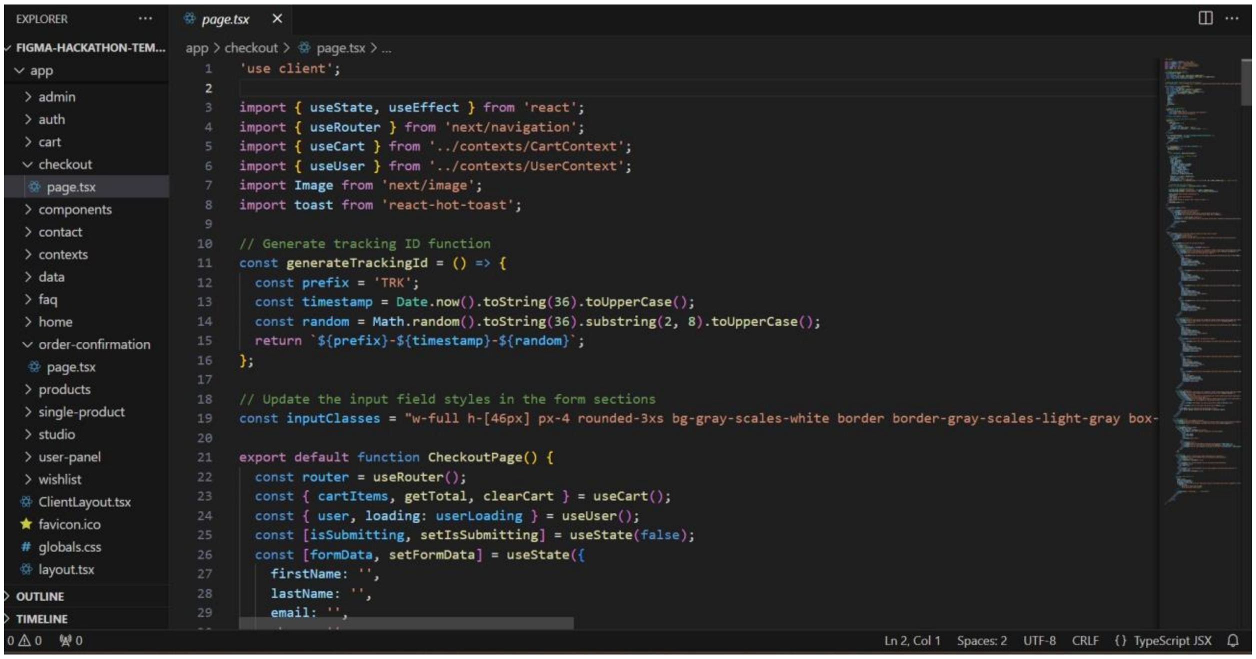
```tsx
'use client';

import { useState, useEffect } from 'react';
import Image from "next/image";
import { useRouter } from "next/navigation";
import { useCart } from "../contexts/CartContext";
import { useWishlist } from "../contexts/WishlistContext";
import { Product } from "../../types";
import { getProducts } from "../../lib/api";
import toast from 'react-hot-toast';
import { ShoppingCartIcon, HeartIcon } from '@heroicons/react/24/outline';
import { HeartIcon as HeartIconSolid } from '@heroicons/react/24/solid';

/**
 * Card Component
 * Individual product card displaying product details and add to cart functionality
 *
 * @param {Object} props - Component props
 * @param {Product} props.product - Product data to display
 * @returns {JSX.Element} Product card with interactive elements
 */
export const Card = ({ product }: { product: Product }): JSX.Element => {
  const router = useRouter();
  const { addItem } = useCart();
  const { wishlistItems, addToWishlist, removeFromWishlist } = useWishlist();
  const isInWishlist = wishlistItems.some(item => item._id === product._id);

  /**
   * Formats price value to display with two decimal places
```

Dynamic Routing:



```tsx
export default function ProductPage({ params }: { params: { id: string } }) {
  const [product, setProduct] = useState<Product | null>(null);
  const { addItem } = useCart();
  const [relatedProducts, setRelatedProducts] = useState<Product[]>([]);

  useEffect(() => {
    const foundProduct = products.find(p => p.id === params.id);
    if (!foundProduct) {
      notFound();
    }
    setProduct(foundProduct);

    // Get 4 random products for related products section
    const otherProducts = products.filter(p => p.id !== params.id);
    const shuffled = [...otherProducts].sort(() => 0.5 - Math.random());
    setRelatedProducts(shuffled.slice(0, 4));
  }, [params.id]);

  if (!product) {
    return <div>Loading...</div>;
  }

  const handleAddToCart = () => {
    addItem({
      id: product.id,
      title: product.title,
      price: product.price,
      image: product.image,
      quantity: 1
```

Checkout:



```tsx
'use client';

import { useState, useEffect } from 'react';
import { useRouter } from 'next/navigation';
import { useCart } from '../contexts/CartContext';
import { useUser } from '../contexts/UserContext';
import Image from 'next/image';
import toast from 'react-hot-toast';

// Generate tracking ID function
const generateTrackingId = () => {
  const prefix = 'TRK';
  const timestamp = Date.now().toString(36).toUpperCase();
  const random = Math.random().toString(36).substring(2, 8).toUpperCase();
  return `${prefix}-${timestamp}-${random}`;
};

// Update the input field styles in the form sections
const inputClasses = "w-full h-[46px] px-4 rounded-3xs bg-gray-scales-white border border-gray-scales-light-gray box-

export default function CheckoutPage() {
  const router = useRouter();
  const { cartItems, getTotal, clearCart } = useCart();
  const { user, loading: userLoading } = useUser();
  const [isSubmitting, setIsSubmitting] = useState(false);
  const [formData, setFormData] = useState({
    firstName: '',
    lastName: '',
    email: '',
```