Marketplace Technical Foundation - [Furniture]

Furniture Website System Architecture

Overview:

This document outlines the system architecture of an e-commerce platform focused on affordable and trendy Pinterest-inspired furniture. The platform is developed using Next.js 14 with TypeScript and leverages Sanity as the content management system (CMS). It incorporates a variety of pages, workflows, and technologies to ensure smooth user experience, robust admin functionalities, and efficient data management.

❖ High-Level System Architecture

✧ Frontend Structure:

1. **Framework:** Next.js 14 with TypeScript for server-side rendering (SSR) and fast, dynamic routes.
2. **Pages:**

    - General: Home, About, Products, Product Details (dynamic), Cart, Admin Panel (admin-only access).

    - User Pages: Login, Sign Up, User Portal (order and shipment details).

    - Admin Pages: Analytics, Dashboard, Orders, Stock Management, Users.

✧ Reusable Components:

1. **UI Components:**

    - `CardComponent.tsx`, `Feature.tsx`, `HeroSection.tsx`, `Listing.tsx`, `PopularProduct.tsx` ensure consistent design and reusable patterns across the platform.

2. **Product Components:**

    - `ProductComponent.tsx` for displaying product details.

    - `ProductCardDetails.tsx` for a detailed view of individual products.

3. **Order Components:**

- `CheckoutModal.tsx` collects user information during checkout.

- `PaymentForm.tsx` integrates Stripe for payment processing.

- `DisplayShipmentDetails.tsx` provides shipment tracking details.

4. **Cart Components:**

   - `UserCartComponent.tsx` manages cart data stored locally.

   - `CardItem.tsx` displays individual cart items.

5. **User Authentication:**

   - `UserLogin.tsx` and `UserSignUp.tsx` handle user login and signup processes.

✧ CMS (Sanity):

1. **Sanity Studio:** Manages dynamic content and structured data, such as:

   - Products: Names, prices, images, categories, and inventory levels.

   - Users: Data for authentication and order history.

   - Orders: Details like purchased items, quantities, and shipping information.

   - Shipments: Tracks shipment statuses via the Shippo API.

   - Analytics: Tracks sales performance, revenue, and product popularity.

2. **Data Schemas:**

   - Define structures for Products, Orders, Users, Inventory, and Analytics.

3. **GROQ Queries:**

   - Enable real-time data fetching from Sanity for display on the frontend.

✧ Mock APIs:

1. **Purpose:** Simulate server-side logic and facilitate seamless integration between frontend and backend.
2. **Endpoints:**

   /api/products/productData : Fetch product data dynamically.

   /tracks/${carrier}/${trackingNumber} : Fetch shipment tracking data from Shippo.

/shipments/ : Retrieve shipment data.

3. **Integration:** Mock APIs integrate with NextApiRequest and NextApiResponse using Axios for smooth requests.

✧ Payment Gateway (Stripe):

1. **Purpose:** Securely manage payments and simulate real-world transactions during development.
2. **Key Features:**

- Stripe Elements for securely collecting payment details.

- Dummy transaction processing for testing purposes.

✧ Shipment Tracking (Shippo API):

1. **Purpose:** Provide real-time shipment tracking information to users.
2. **Features:**

- Integrates live tracking on the user's order history page.

- Uses the Shippo API to fetch accurate shipment data post-payment confirmation.

❖ Workflow Overview

✧ User Workflow:

1. **Visit Home Page:**

Browse product categories dynamically fetched from Sanity via mock APIs.

2. **Add to Cart:**

- Add items to the cart, which are temporarily stored locally.

- User can add items without logging in.

3. **Checkout Process:**

- If the user is not logged in during checkout, they are prompted to log in or sign up.

- After successful login or signup, CheckoutModal.tsx collects shipping and payment information.

- Process payment securely via Stripe using PaymentForm.tsx.

4. **Shipment Tracking:**

    - Upon successful payment, generate a shipment request via the Shippo API.

    - Display shipment tracking details using DisplayShipmentDetails.tsx.

5. **User Portal:**

    - Access order and shipment history after logging in.

    - Data is verified against records stored in Sanity.

✧ Admin Workflow:

1. **Login:**

    - Access admin functionalities via AdminLogin.tsx.

2. **Analytics:**

    - View sales data and product performance using Analytics.tsx.

3. **Inventory & Order Management:**

    - Manage stock via Stock.tsx.

    - Oversee user orders using Orders.tsx and user details via Users.tsx.

    - Admin changes are instantly reflected in Sanity.

4. **Navigation:**

    - Use SideBar.tsx for seamless navigation between admin features.

❖ Collaboration Notes

✧ Challenges Faced:

1. **Schema Design:**

    Developing efficient schemas in Sanity for Products, Users, and Orders was challenging. Iterative refinements helped create scalable data models.

2. **API Integration:**

    Integrating mock APIs and understanding the interplay between Sanity and the frontend required extensive debugging.

3. **Workflow Implementation:**

Mapping workflows like cart management, payment, and shipment tracking to reusable components demanded significant planning.

✧ Learning Outcomes:

- Enhanced understanding of system architecture and e-commerce workflows.

- Improved skills in TypeScript, Next.js, and CMS integrations.

- Practical experience in designing scalable APIs and seamless user experiences.

✧ Feedback and Adaptations

- User testing revealed the need for a more intuitive cart interface, leading to updates in `UserCartComponent.tsx`.

- Admin feedback suggested improving analytics visuals, prompting changes to `Analytics.tsx` for better insights.

❖ Technologies and Tools:

1. **Frontend:** Next.js 14 with TypeScript.
2. **CMS:** Sanity (content and data management).
3. **Payment Gateway:** Stripe (payment processing).
4. **Shipment Tracking:** Shippo API (live shipment updates).
5. **Deployment:** Hosted on Vercel for fast and reliable delivery.

❖ Folder and Component Structure

✧ Components Folder Structure:

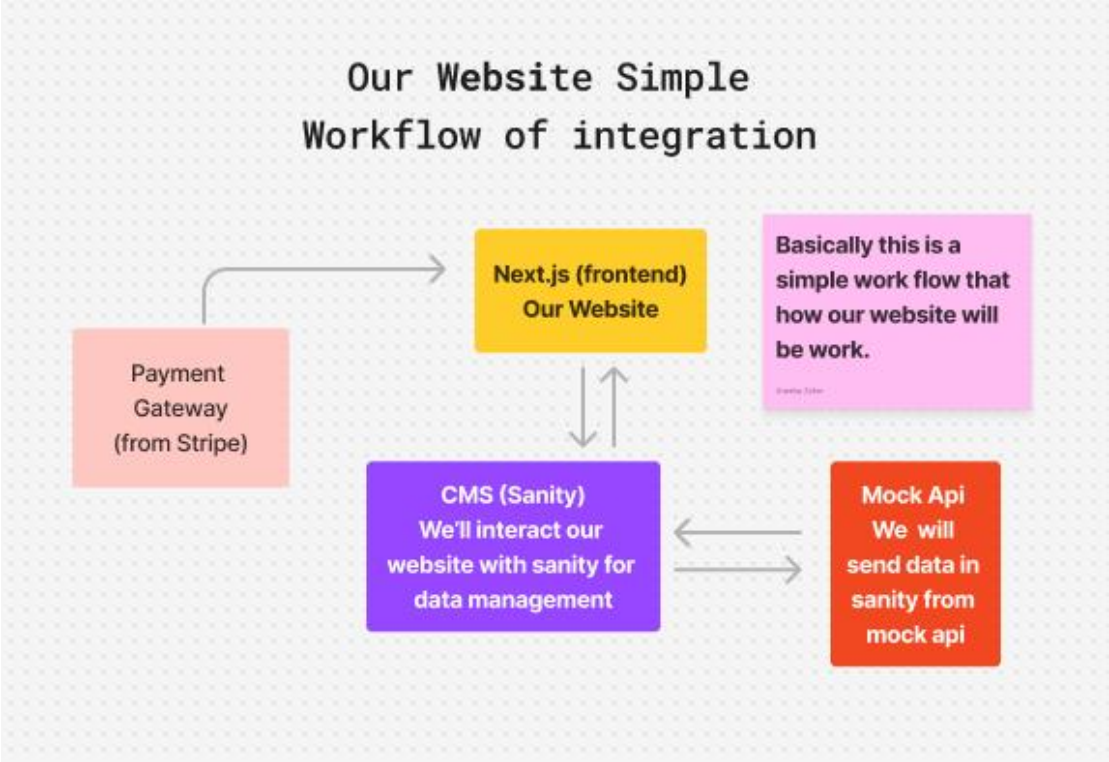
```
/components
 /about
   AboutBrand.tsx
   AboutFeature.tsx
   AboutGetInTouch.tsx
   AboutSection.tsx
   AboutSignUp.tsx
 /adminPanel
   Analytics.tsx
   Dashboard.tsx
   Order.tsx
   Stock.tsx
   Users.tsx
   SideBar.tsx
 /orderSystem
   CheckoutModal.tsx
   PaymentForm.tsx
```

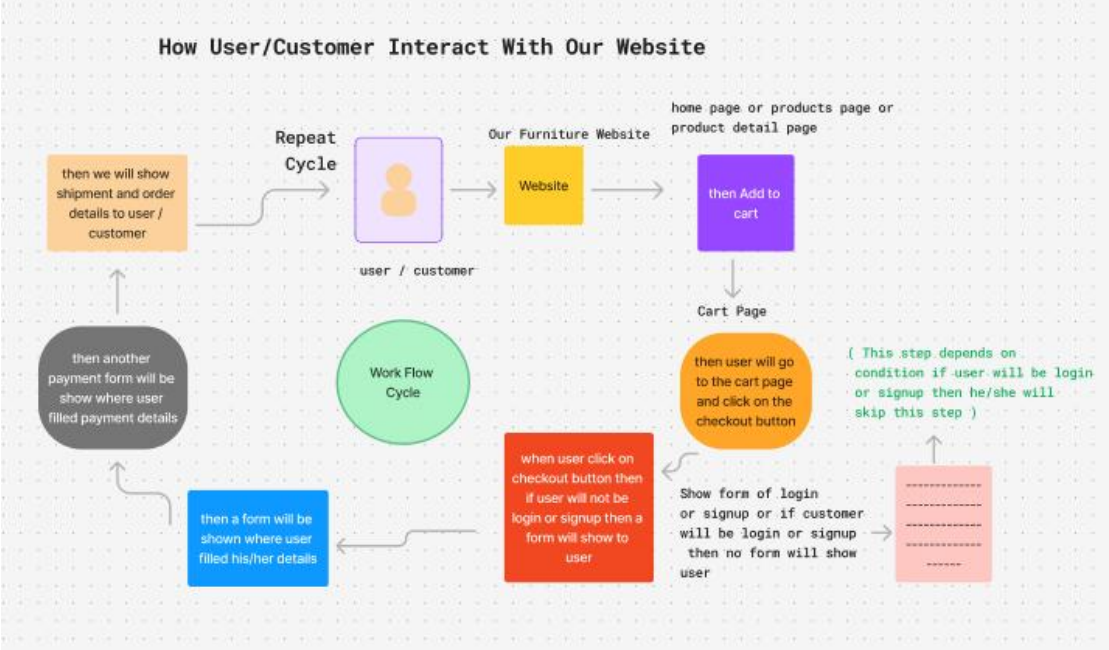
```
  DisplayShipmentDetails.tsx
  UserLoginAndSignUp.tsx
 /userPortal
  LoginPortal.tsx
  UserDetail.tsx
 /userCart
  CardItem.tsx
  UserCartComponent.tsx
 /reusableComponents
  ProductCard.tsx
  AboutFeatureCard.tsx
  FeatureCard.tsx
 /heroSection
  Hero.tsx
  Listing.tsx
  PopularProducts.tsx
  Feature.tsx
  GetInTouch.tsx
  SignUp.tsx
 /product
  ProductComponent.tsx
  ProductCardDetails.tsx
 /header
  Header.tsx
 /footer
  Footer.tsx
  FooterHeadings.tsx
  FooterLinks.tsx
```

✧ Folder Structure:

```
/project-root
/src
 /app
  /about
   page.tsx
  /cart
   page.tsx
  /product
   page.tsx
   /[id]
    page.tsx
  /contact
   page.tsx
  /user
   page.tsx
  /admin
   page.tsx
 /sanity
  /schemaTypes
   /modelTypes
    analytics.ts
    // Other schemas for user, order, shipment, stock, products
 /types
  componentTypes.ts
 /components
```

// Folder for app components





❖ Conclusion:

This system architecture ensures scalability, efficiency, and a user-friendly experience. By combining reusable components, dynamic rendering, and robust CMS support, the platform is prepared to handle both user and admin requirements effectively. Continuous iteration and integration of feedback will further enhance the system's capabilities and usability.

==========

Prepared by: Areeba Zafar (Student Leader)