Market Place Hackathon Day 2, Planning and Technical Foundation

1. Technical Requirements

- **Frontend**:
  - **Framework**: Using **Next.js** for building frontend. Next.js is ideal for building scalable, fast, and SEO-friendly web applications.
  - **Design System**: Create a reusable set of components like buttons, forms, and card layouts that match your branding and UX requirements.
- **Backend**:
  - **CMS**: Using **Sanity CMS** as backend. Sanity will manage the content (products, orders, shipment data) and will provide an API for fetching that data.
  - **APIs**: Integrate third-party APIs (like payment gateways, shipping APIs) to extend the marketplace's functionality.
- **Third-party APIs**:
  - To integrate APIs for external services such as:
    - **Payment processing** (e.g., Stripe, PayPal).
    - **Shipping** (e.g., Shippo, EasyPost, Ship engine).
    - **Authentication** (e.g., OAuth, Firebase Authentication).

2. Design System Foundation Diagram

Think of the design system as a foundation of reusable components that will help maintain consistency in UI across my app.

Diagram structure:

- **UI Components**
  - Buttons
  - Form elements (input fields, checkboxes, dropdowns)
  - Modals & Alerts
  - Cards (Product Card, Order Summary Card, etc.)
  - Navigation (Header, Footer, Sidebar)
  - Typography (Fonts, Text styles)
- **Pages**
  - Homepage
  - Product Listing Page
  - Product Detail Page
  - Cart & Checkout Pages
  - Order Confirmation Page
- **State Management**
  - Global state (via Context API, Redux, or Zustand) for things like cart items, user authentication, etc.

3. Plan API Requirements

We need APIs to interact with both the frontend and backend.

API Endpoints:

- **Products**:
  - `GET /api/products`: Fetch all products
  - `GET /api/products/:id`: Fetch product by ID
  - `POST /api/products`: Create a new product (for admin)
  - `PUT /api/products/:id`: Update product details (for admin)
  - `DELETE /api/products/:id`: Delete a product (for admin)
- **Orders**:
  - `GET /api/orders`: Fetch all orders (for admin)
  - `POST /api/orders`: Create a new order
  - `GET /api/orders/:id`: Fetch order details by ID
  - `PUT /api/orders/:id`: Update order status (e.g., shipping, completed)
  - `DELETE /api/orders/:id`: Delete an order (for admin)
- **Shipments**:
  - `GET /api/shipments`: Fetch all shipments
  - `POST /api/shipments`: Create a shipment
  - `GET /api/shipments/:id`: Fetch shipment details
  - `PUT /api/shipments/:id`: Update shipment status
  - `DELETE /api/shipments/:id`: Delete a shipment (for admin)

Third-Party API Integrations:

- **Payment Gateway API** (e.g., Stripe)
  - `POST /api/payment`: Process payment for orders
- **Shipping API** (e.g., Shippo)
  - `GET /api/shipping-rate`: Fetch shipping rates
  - `POST /api/shipment-create`: Create a shipment
- **Authentication API**
  - `POST /api/login`: Login a user
  - `POST /api/register`: Register a user

4. Q-Commerce (Quick Commerce)

- **Instant Product Listings**: Ensure that products can be added, updated, or removed quickly through the CMS (Sanity).
- **Real-time Cart Updates**: Keep cart items and order status updated in real-time using websockets or APIs.
- **Fast Checkout Process**: Use APIs to simplify and speed up the checkout (payment, shipping options).
- **Efficient Shipment Tracking**: Integrate with shipping APIs to track orders and notify users.

5. API Methods & Endpoints Naming

- **Methods**:
  - `GET`: Fetch data from the server (e.g., products, orders, shipments).
  - `POST`: Send data to the server (e.g., create an order, add a product).
  - `PUT`: Update existing data (e.g., update order status, modify product details).
  - `DELETE`: Remove data (e.g., delete a product or order).

6. Write technical documentation

- **Schema:**

```
export default{
name: “product”,
type: “document”,
title: “Product”,
        fields: [
           {
           name: “name”,
           type: “strings”,
           title: “Product Name”
    },
          {
           name: “price”,
           type: “number”,
           title: “Price”
    },
          {
           name: “description”,
           type: “strings”
           title: “Description
    }
  ]
}
```
