Marketplace Builder Hackathon Day-2 [Heckto]

General E-Commerce Marketplace Plan

Objective

The aim here is to devise an effective e-commerce strategy that allows for hassle-free scaling with singular built-in utilities like:

- Product browsing and management via Sanity CMS.
- Authentication using Clerk.
- Secure payments via Stripe.
- Modern tools like `useContext` for cart functionality.

System Architecture Diagram

graph TD

User[User] -->|Logs In| Clerk[Clerk Authentication]
User -->|Browses| Frontend[Frontend (Next.js)]
Frontend -->|Fetches Data| Sanity[Sanity CMS]
Frontend -->|Manages Cart| Cart[useContext API]
Frontend -->|Places Order| Stripe[Stripe Checkout]
Sanity -->|Stores| Product[Product Data]
Sanity -->|Stores| Order[Order Data]

Features & Workflow

Frontend

User Authentication (Clerk):

- Use Clerk's pre-built authentication components.
- Manage user sessions without storing data in Sanity CMS.

Product Browsing:

- Fetch and display products from Sanity CMS using GROQ queries.

Cart Management:

- Use `useContext` to manage cart state globally.
- Add/remove items and calculate totals dynamically.

Checkout Process:

- Collect user details and payment via Stripe-hosted checkout.
- Display order confirmation after successful payment.

Backend

- **Sanity CMS:**
  - Manage products and orders using Sanity Studio.
- **Custom APIs:**
-
  - `/api/checkout`: Integrate with Stripe for payments.
- **Admin Panel:**
  - Use Sanity Studio for inserting and managing data.

Data Models

Sanity Schemas

Product Schema

```
export default {
  name: 'product',
  title: 'Product',
  type: 'document',
  fields: [
    { name: 'name', title: 'Name', type: 'string' },
```

```
    { name: 'description', title: 'Description', type: 'text' },
    { name: 'price', title: 'Price', type: 'number' },
    { name: 'image', title: 'Image', type: 'image' },
    { name: 'stock', title: 'Stock', type: 'number' },
  ],
};
```

Order Schema

```
export default {
  name: 'order',
  title: 'Order',
  type: 'document',
  fields: [
    { name: 'userEmail', title: 'User Email', type: 'string' },
    { name: 'items', title: 'Items', type: 'array', of: [{ type: 'reference', to: [{ type: 'product' }] }] },
    { name: 'totalAmount', title: 'Total Amount', type: 'number' },
    { name: 'status', title: 'Status', type: 'string', options: { list: ['pending', 'confirmed', 'failed'] } },
    { name: 'shippingLabelId', title: 'Shipping Label ID', type: 'string' },
  ],
};
```

API Requirements

Endpoint	Method	Description
`/api/checkout`	POST	Integrate Stripe for payment processing.

Tools & Libraries

- **Clerk:** Authentication.
- **Sanity CMS:** Content management.
- **ShipEngine API:** Shipping and tracking.
- **Stripe:** Payment gateway.
- **React Context API:** Cart functionality.

Development Steps

1. **Set Up Next.js Project:**
    - Create a new project: `npx create-next-app@latest my-app --typescript`.
2. **Install Dependencies:**
    - `npm install @clerk/nextjs @sanity/client shipengine stripe`.
3. **Configure Clerk:**
    - Set up Clerk in `_app.tsx` and integrate authentication components.
4. **Set Up Sanity CMS:**
    - Create schemas for products and orders.
    - Use Sanity Studio to manage data.
5. **Integrate APIs:**
    - Create custom API routes for ShipEngine and Stripe.
6. **Develop Frontend Pages:**
    - Home: Product listing.
    - Cart: Display selected items.
    - Checkout: Integrate with Stripe.
7. **Test Functionality:**
    - Test cart management, order placement, and checkout sessions

---

Deliverables

1. **System Architecture Diagram:** Shows component interaction.
2. **Sanity Schemas:** For products and orders.
3. **API Endpoints:** For shipping, tracking, and payments.
4. **Frontend Pages:** Authentication, product browsing, cart management, and order confirmation.
5. **Portfolio-Ready Submission:** Polished project showcasing full-stack e-commerce skills.