***Hackathon Day 2: the Technical Foundation***

1. Define Technical Requirements

This document outlines the technical planning phase for the e-commerce marketplace, focusing on three key areas: frontend requirements, backend integration using Sanity CMS, and third-party API integrations.

1. Frontend Requirements The frontend will deliver a seamless, user-friendly experience with the following pages and features:

Essential Pages:

Homepage:

1. Highlights: Featured products, promotional banners, category shortcuts.

2. Call-to-Actions (CTAs): "Shop Now," "Browse Categories," "View Deals."

Shop Section:

1. Category Pages: Allow users to browse products by categories (e.g., Groceries, Electronics, Fashion).

2. Product Listing Page:

Displays products with:

Filters: Price, category, ratings, availability, Sorting Options: Best Sellers, Price (Low to High), New Arrivals.

1. Product Details Page: Key Features:

Product Title, Images, Description. Price, Stock Availability, Discounts. Color Family Selector and Size Options. Ratings and Reviews. Questions about the Product (FAQ Section). Add-to-Cart and Add-to-Wishlist functionality. Recommendations for similar products. Cart Page: 1. Displays selected products with quantity and price breakdown.

1. Options to update quantities or remove items.

Checkout Page:

1. Captures shipping details, payment method, and order summary.

2. Features for applying discount codes and selecting delivery preferences.

Order Confirmation Page:

1. Displays order details, estimated delivery time, and shipment tracking. About and Contact Pages:

1. Business details and a contact form for customer inquiries.

Technical Stack:

Frameworks: React.js and Next.js for building dynamic and SEO-friendly pages.

Component Library: shadcn/ui for customizable, reusable components.

Styling: Tailwind CSS for responsive and visually appealing design.

1. Backend with Sanity CMS Sanity CMS will serve as the backend to manage dynamic data like products, customers, and orders.

Sanity Schema Design:

Products Schema:

Fields:

ProductID: Primary Key.

Name, Description, Category, Price, Stock Quantity.

Color Options, Size Options.

Ratings, Reviews, and FAQs.

Discount (if applicable).

Customer Schema:

Fields:

CustomerID: Primary Key.

Full Name, Email, Phone Number, Address.

Order History, Loyalty Points (optional).

Orders Schema: o Fields:

OrderID: Primary Key.

CustomerID: Foreign Key.

ProductID(s): Many-to-Many relationship.

Order Date, Status (e.g., Pending, Shipped, Delivered).

Total Amount. Payments Schema:

Fields:

PaymentID: Primary Key.

OrderID: Foreign Key.

Amount Paid, Payment Method (e.g., Credit Card, UPI, Wallet).

Payment Status (e.g., Successful, Pending).

Shipment Schema: o Fields:

ShipmentID: Primary Key.

OrderID: Foreign Key.

Courier Service, Tracking Number.

Estimated Delivery Date, Shipment Status.

Implementation Steps:

1. Use Sanity Studio to design and test schemas.

2. Fetch and manipulate data on the frontend using GROQ queries.

3. Optimize schemas for scalability and future expansion.

3. Third-Party API Integrations To provide critical marketplace functionality, integrate the following APIs:

Payment Gateways: Stripe:

Features: Secure payments, support for multiple payment methods, and real-time transaction updates.

Integration: Use Stripe SDKs and APIs for seamless integration.

PayPal:

Features: Widely accepted payment solution with options for credit/debit card payments and wallets.

Integration: Use PayPal’s REST API for transactions.

Shipment Tracking APIs:

ShipEngine:

Features: Multi-carrier support, real-time tracking, and shipping rate comparison.

Use Case: Efficient shipment label generation and tracking.

AfterShip:

Features: Real-time shipment tracking and customer notifications.

Use Case: Provide live tracking updates for customers.

EasyPost:

Features: API for shipping label creation, rate calculation, and tracking.

Use Case: Streamline the backend for logistics and delivery management.

Additional APIs:

1. Google Maps API:

Use Case: Address validation and delivery zone mapping.

2. Notification APIs (Email/SMS):

Use Case: Send order confirmations and delivery status updates.

Middleware Implementation:

Use Node.js and Express.js to handle API requests and process server-side logic.

Secure API endpoints using JWT (JSON Web Tokens).

Development Pipeline

1. Frontend Development: Build responsive pages using React.js, Next.js, and Tailwind CSS. 2. Backend Development: Implement schemas in Sanity CMS and connect the frontend via APIs.

3. API Integration: Integrate payment and shipment APIs to ensure seamless functionality.

4. Testing: Conduct thorough testing for functionality, responsiveness, and security.

5. Deployment: Host the platform on Vercel (frontend) and Heroku or AWS Lambda (backend). This detailed plan aligns technical implementation with business goals, ensuring a scalable, user-centric, and efficient e-commerce platform.

2. Design System Architecture

To visualize how these components interact, consider the following high-level architecture:

Data flow

User (web) Frontend(next.js)

Product data API(sanity cms)

Shipment tracking

Payment processing third party API

1.User Browsing: A user visits the marketplace frontend to browse products. The frontend requests product listings from the Product Data API.

2.Product Display: The Product Data API fetches data from Sanity CMS. Product details are displayed dynamically on the site.

3.Order Placement: When the user places an order, the order details are sent to Sanity CMS via an API request. The order is recorded in Sanity CMS.

4.Shipment Tracking: Shipment tracking information is fetched through a Third-Party API. Real-time tracking updates are displayed to the user.

5.Payment Processing: Payment details are securely processed through the Payment Gateway. A confirmation is sent back to the user and recorded in Sanity CMS. By following this detailed plan, you can create a robust and scalable marketplace that meets both business and technical requirements.

Collaborate and Refine

1. Feedback Integration: Continuously collect feedback from stakeholders and end-users to enhance features.

2. Code Reviews: Conduct thorough peer reviews to maintain code quality and identify potential issues early.

3. Iterative Testing: Implement unit, integration, and UI testing to ensure robustness.

4. Documentation Updates: Regularly update this documentation to reflect changes in architecture, workflows, or API functionality.