Hackathon Day 2: Planning the Technical Foundation

Recap of Day 1: Business Focus

Primary Purpose:

Designed a versatile platform to offer diverse products with convenience, competitive pricing, and reliable delivery tailored to modern consumer needs.

Problem Solved:

Simplified fragmented shopping experiences and reduced delivery delays with seamless navigation, swift fulfillment, and a user-centric design.

Target Audience:

Focused on time-conscious shoppers, families seeking convenience, and professionals looking for reliable delivery options.

Products and Services:

1. **Categories:** Groceries, fashion, home essentials, health & wellness items, and electronics.
2. **Added Value:** Subscription deliveries, exclusive discounts, and hassle-free returns.

E-Commerce Data Schema:

1. **Core Entities:** Products, Customers, Orders, Payments, Shipment, and Delivery Zones.
2. **Key Relationships:** Integrated models for real-time tracking, customer order history, and dynamic delivery charges.

Marketplace Features:

1. Dynamic filters for products.
2. Real-time order tracking and flexible payment options.
3. AI-powered personalized recommendations and loyalty programs.

Day 2 Activities: Transitioning to Technical Planning

1. Define Technical Requirements

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

Essential Pages:

1. **Homepage:**
    a. Highlights: Featured products, promotional banners, category shortcuts.
    b. Call-to-Actions (CTAs): "Shop Now," "Browse Categories," "View Deals."
2. **Shop Section:**
    a. **Category Pages:** Allow users to browse products by categories (e.g., Groceries, Electronics, Fashion).
    b. **Product Listing Page:**
        i. Filters: Price, category, ratings, availability.
        ii. Sorting Options: Best Sellers, Price (Low to High), New Arrivals.
    c. **Product Details Page:**
        i. Key Features: Product Title, Images, Description, Price, Stock Availability, Discounts.
        ii. Options: Color Family Selector, Size Options.
        iii. Ratings, Reviews, FAQs, Add-to-Cart and Add-to-Wishlist functionality.
        iv. Recommendations for similar products.
3. **Cart Page:**
    a. Displays selected products with quantity and price breakdown.
    b. Options to update quantities or remove items.
4. **Checkout Page:**
    a. Captures shipping details, payment method, and order summary.
    b. Features for applying discount codes and selecting delivery preferences.
5. **Order Confirmation Page:**
    a. Displays order details, estimated delivery time, and shipment tracking.
6. **About and Contact Pages:**
    a. 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.

2. 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:

- 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:

- 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

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:** Address validation and delivery zone mapping.
2. **Notification APIs (Email/SMS):** 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:**
    a. Build responsive pages using React.js, Next.js, and Tailwind CSS.
2. **Backend Development:**
    a. Implement schemas in Sanity CMS and connect the frontend via APIs.
3. **API Integration:**
    a. Integrate payment and shipment APIs to ensure seamless functionality.
4. **Testing:**
    a. Conduct thorough testing for functionality, responsiveness, and security.
5. **Deployment:**
    a. Host the platform on Vercel (frontend) and Heroku or AWS Lambda (backend).

4. Design System Architecture

System Architecture Workflow:

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

2. **Product Display:**
    a. The Product Data API fetches data from Sanity CMS.
    b. Product details are displayed dynamically on the site.
3. **Order Placement:**
    a. When the user places an order, the order details are sent to Sanity CMS via an API request.
    b. The order is recorded in Sanity CMS.
4. **Shipment Tracking:**
    a. Shipment tracking information is fetched through a Third-Party API.
    b. Real-time tracking updates are displayed to the user.
5. **Payment Processing:**
    a. Payment details are securely processed through the Payment Gateway.
    b. A confirmation is sent back to the user and recorded in Sanity CMS.

5. Plan API Requirements

Endpoint Name	Method	Description	Request Body	Response Example
/api/users/register	POST	Registers a new user.	{ "username": "john", "email": "john@example.com", "password": "pass123" }	{ "status": "success", "message": "User registered successfully." }
/api/users/login	POST	Authenticates a user and generates a JWT.	{ "email": "john@example.com", "password": "pass123" }	{ "status": "success", "token": "jwt.token.here" }
/api/users/{id}	PUT	Updates user details.	{ "username": "john_updated", "email": "updated@example.com" }	{ "status": "success", "message": "Profile updated successfully." }
/api/categories	GET	Retrieves a list of all product categories.	None	{ "status": "success", "data": [{ "id": 1, "name": "Groceries" }] }

/api/categories/{id}/products	GET	Fetches products within a specific category.	None	{ "status": "success", "data": [{ "id": 101, "name": "Apples" }] }
/api/products/{id}	GET	Retrieves details of a specific product.	None	{ "status": "success", "data": { "id": 101, "name": "Apples" } }
/api/cart/add	POST	Adds a product to the user's cart.	{ "product_id": 101, "quantity": 2 }	{ "status": "success", "message": "Item added to cart." }
/api/cart	GET	Retrieves the current state of the user's cart.	None	{ "status": "success", "data": { "items": [{ "id": 101 }] } }
/api/checkout	POST	Processes payment and places an order.	{ "payment_method": "card", "shipping_address": { ... } }	{ "status": "success", "message": "Order placed successfully." }
/api/orders/{id}	GET	Retrieves the details of a specific order.	None	{ "status": "success", "data": { "order_id": 12345 } }
/api/homepage	GET	Retrieves homepage content, including featured items.	None	{ "status": "success", "data": { "featured_products": [...] } }

6. Write Technical Documentation

Technical Documentation for eCommerce System

System Architecture Overview:

- **Frontend:**
  - Framework: React.js / Next.js for fast and responsive UI.
  - Styling: TailwindCSS or Material-UI for component-based styling.
  - State Management: Redux or Context API for seamless state handling.
- **Backend:**
  - Framework: Node.js with Express.js for API creation.
  - Database: MongoDB for a scalable, NoSQL solution to store products, users, and orders.

    - Authentication: JSON Web Tokens (JWT) for secure user authentication and authorization.
- **CMS:**
    - Sanity.io: For managing dynamic content.