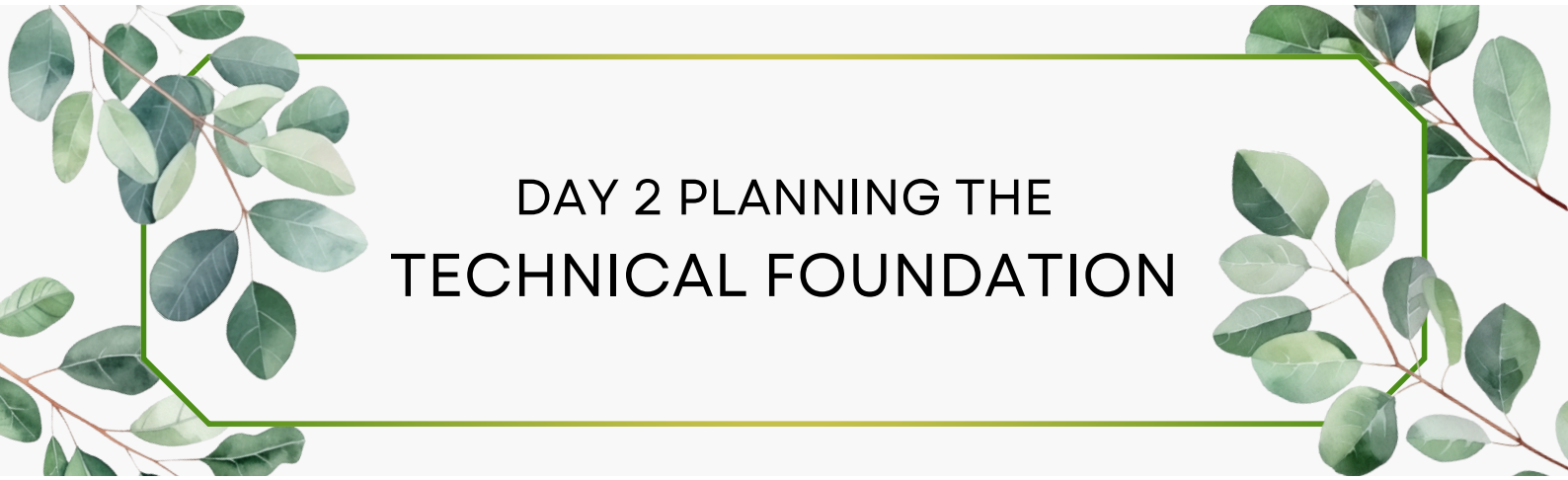

Hackathon Day 2: Planning the

Technical Foundation

Recap of Day 1: Business Focus

Day 1 Recap: Laying the Marketplace Foundation

Key Takeaways

1. **Primary Purpose**: Designed a versatile platform to offer **sofas, chairs, and electronic items** with convenience, competitive pricing, and reliable delivery tailored to modern consumer needs.
2. **Problem Solved**: Simplified fragmented shopping experiences and reduced delivery delays with seamless navigation, swift fulfillment, and a user-centric design.
3. **Target Audience**: Focused on:
   - 🏡 **Homeowners** looking for stylish and comfortable furniture.
   - 💻 **Tech enthusiasts** seeking the latest electronic gadgets.
   - 🏢 **Businesses** in need of ergonomic chairs and office electronics.
4. **Products and Services**:
   - **Categories**: Sofas, chairs, and electronic items.
   - **Added Value**: Subscription-based deliveries, exclusive discounts, and hassle-free returns.
5. **E-Commerce Data Schema**:
   - **Core Entities**: Products, Customers, Orders, Payments, Shipment.
   - **Key Relationships**: Integrated models for real-time tracking, customer order history, and dynamic delivery charges.
6. **Marketplace Features**:
   - Dynamic filters for products.
   - Real-time order tracking and flexible payment options.
   - AI-powered personalized recommendations and loyalty programs.

Day 2 Activities: Transitioning to Technical Planning

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**.

Frontend Requirements

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

Essential Pages

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

Backend with Sanity CMS

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

Sanity Schema Design

1. **Products Schema**:
   - Fields:
     - ProductID: Primary Key.
     - Name, Description, Category, Price, Stock Quantity.
     - Ratings, Reviews, and FAQs.
     - Discount (if applicable).
2. **Customer Schema**:
   - Fields:
     - CustomerID: Primary Key.
     - Full Name, Email, Phone Number, Address.
     - Order History, Loyalty Points (optional).
3. **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.
4. **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).
5. **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.

Third-Party API Integrations

To provide critical marketplace functionality, integrate the following APIs:

1. **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.
2. **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.
3. **Additional APIs**:
    - **Google Maps API**:
        - Use Case: Address validation and delivery zone mapping.
    - **Notification APIs (Email/SMS)**:
        - Use Case: Send order confirmations and delivery status updates.

2. Design System Architecture

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

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.

3. Plan API Requirements

Here is a table summarizing the API endpoints for the eCommerce platform:

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/products	GET	Retrieves a list of all products.	None	{ "status": "success", "data": [{ "id": 1, "name": "Sofa" }] }

/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/checkout	POST	Processes payment and places an order.	{ "payment_method": "card", "shipping_address": { ... } }	{ "status": "success", "message": "Order placed successfully." }

4. Write Technical Documentation

Document your system architecture, workflows, and API requirements in a structured format. Use headings, diagrams, and bullet points for clarity.

5. 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.