DAY 2

GENERAL E COMMERCE

Prepared by:
Muhammad Mubashir Saeedi

Day2_Planning_the_Technical_Foundation

DAY 2 PLANNING THE TECHNICAL FOUNDATION

Hackathon Day 2: Planning the Technical Foundation

Recap of Day 1: Business Focus

Day 1 Recap: Laying the Marketplace Foundation

Key Takeaways

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

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.

3. **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.
2. 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.

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

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)**.

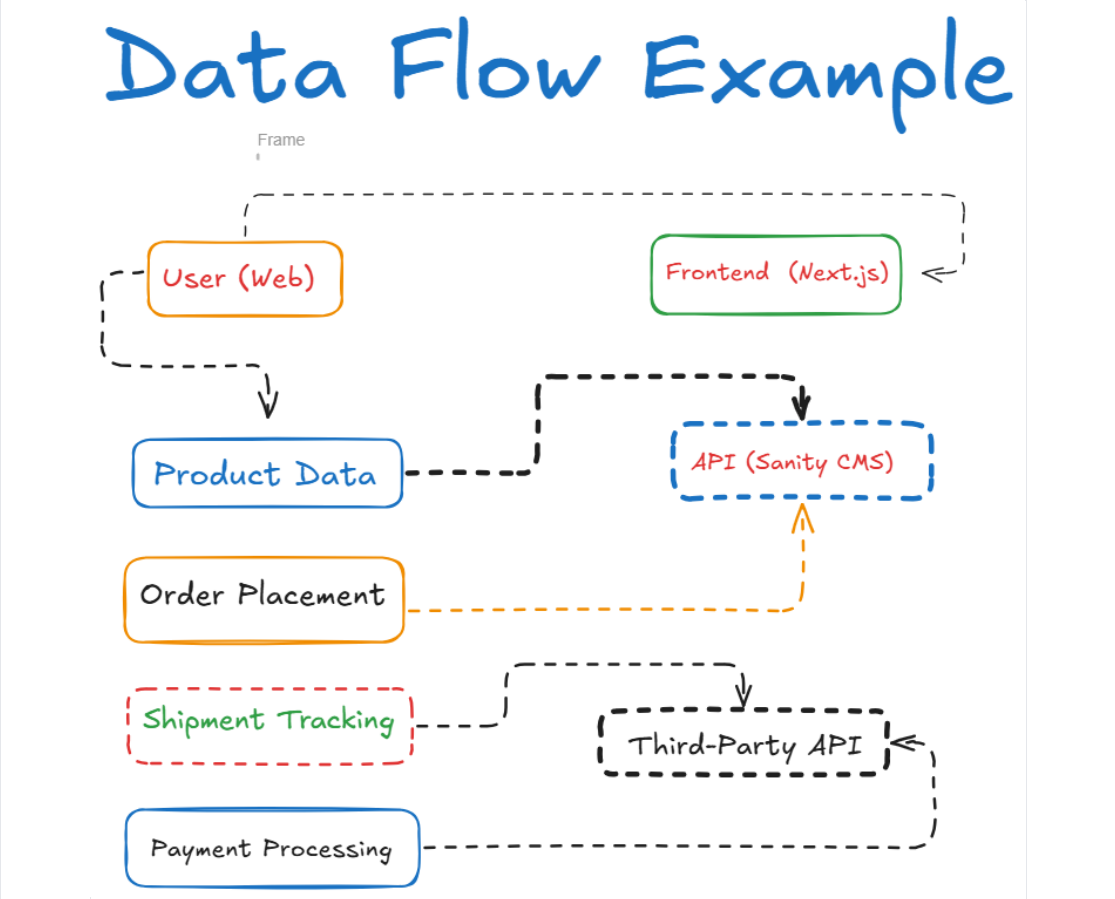
Development Pipeline

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:



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.

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/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}/prod	GET	Fetches	None	{ "status":

Endpoint Name	Method	Description	Request Body	Response Example
ucts		products within a specific category.		"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": [...] } }

This table provides an at-a-glance overview of the API endpoints, their purposes, and example data formats for requests and responses.

4. Write Technical Documentation

Technical Documentation for eCommerce System

This documentation provides a comprehensive guide to the eCommerce system's architecture, workflows, API endpoints, and Sanity CMS schema examples. It aims to assist developers, project managers, and stakeholders in understanding and implementing the platform effectively.

1. System Architecture Overview

The eCommerce platform is built with a modular architecture to ensure scalability, maintainability, and performance.

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 such as categories, product descriptions, and blogs.

Deployment:

- **Hosting**: AWS, Vercel, or Netlify for deployment.
- **CI/CD**: GitHub Actions or Jenkins for automated deployment pipelines.

2. Key Workflows

2.1 User Registration & Authentication

1. **User Signup**: Users register by providing their email, password, and profile details. Data is stored in the database after validation.
2. **Login**: Users enter their credentials to obtain a JWT token, enabling secure session handling.
3. **Password Recovery**: Users reset passwords via a token-based recovery system.

2.2 Product Browsing & Filtering

1. Users view categories fetched from the CMS.
2. Clicking a category triggers the /categories/{id}/products API to display relevant products.

3. Users can filter products by attributes such as price, ratings, or availability.

2.3 Cart Management

1. Users add products to their cart via the /cart/add endpoint.
2. The cart updates dynamically, storing items in the database or local storage for guest users.
3. The cart is displayed using the /cart endpoint.

2.4 Checkout & Payment

1. The user proceeds to checkout, providing payment and shipping details.
2. The /checkout endpoint processes the payment and creates an order.
3. Users receive order confirmation via email.

3. Category-Specific Instructions

Groceries

- Ensure real-time stock updates using WebSockets or API calls.
- Implement expiry tracking for perishable items.

Fashion

- Enable size and color selection for products.
- Support a "virtual try-on" feature using AR/VR for an enhanced user experience.

Home Essentials

- Use bundle offers to encourage bulk purchases.
- Provide detailed specifications and care instructions.

Health & Wellness

- Include certifications and lab test reports for products.
- Highlight subscriptions for recurring orders (e.g., vitamins).

Electronics

- Showcase detailed product specs with comparison tools.
- Offer extended warranty and service plans.

4. API Endpoints

Authentication

1. /api/users/register – Register a new user.
2. /api/users/login – Authenticate user and return a JWT token.

Categories

1. /api/categories – Retrieve all categories.
2. /api/categories/{id}/products – Retrieve products under a specific category.

Products

1. /api/products/{id} – Retrieve detailed information about a product.

Cart

1. /api/cart/add – Add a product to the cart.
2. /api/cart – Retrieve cart details.

Orders

1. /api/checkout – Process payment and create an order.

5. Sanity Schema Example

Here's an example schema for managing product categories and products in Sanity.io:

```
// schemas/category.js
export default {
  name: 'category',
  title: 'Category',
  type: 'document',
  fields: [
    {
      name: 'name',
      title: 'Name',
      type: 'string',
      validation: (Rule) => Rule.required(),
    },
    {
      name: 'description',
      title: 'Description',
      type: 'text',
    },
    {
      name: 'image',


 title: 'Image',
      type: 'image',
```

```
      options: {
        hotspot: true,
      },
    },
  ],
};

// schemas/product.js
export default {
  name: 'product',
  title: 'Product',
  type: 'document',
  fields: [
    {
      name: 'name',
      title: 'Name',
      type: 'string',
      validation: (Rule) => Rule.required(),
    },
    {
      name: 'description',
      title: 'Description',
      type: 'text',
    },
    {
      name: 'price',
      title: 'Price',
      type: 'number',
      validation: (Rule) => Rule.min(0).required(),
    },
    {
      name: 'category',
      title: 'Category',
      type: 'reference',
      to: [{ type: 'category' }],
    },
    {
      name: 'stock',
      title: 'Stock',
      type: 'number',
      validation: (Rule) => Rule.min(0).required(),
    },
    {
      name: 'image',
      title: 'Image',
      type: 'image',
      options: {
        hotspot: true,
      },
    },
  ],
};
```

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.