Day 2: Hackathon Task

Marketplace Technical Foundation – [General E-Commerce]

---

1. Business Goals:

- **Increase Revenue:** Maximize sales through expanded product offerings and market reach.
- **Enhance Customer Experience:** Improve platform usability, customer satisfaction, and retention.
- **Improve Operational Efficiency:** Streamline internal processes like inventory and order management.
- **Scalability and Growth:** Build a platform capable of handling increasing traffic and products.
- **Market Expansion:** Reach new geographies and customer segments.

---

2. Translating Business Goals into Clear Technical Requirements:

- **Scalable Architecture:**
    - **Front-End (Next.js):** Use server-side rendering and static site generation for fast performance and scalability.
    - **Back-End (Sanity CMS):** Store dynamic content (e.g., products, categories) and allow easy updates.
    - **Third-Party APIs:** Integrate cloud services (AWS, GCP) for autoscaling and API-driven functionality.
- **User-Friendly Interface:**
    - **Front-End (Next.js):** Create a responsive and dynamic UI with React components for smooth navigation.
    - **Back-End (Sanity CMS):** Manage content dynamically, such as product listings and categories.
    - **Third-Party APIs:** Integrate Algolia for fast and advanced search functionality.
- **Payment Gateway Integration:**
    - **Front-End (Next.js):** Design secure and optimized checkout pages.
    - **Back-End (Sanity CMS):** Handle transaction and order data securely.

    - **Third-Party APIs:** Use Stripe or PayPal for secure and seamless payment processing.

- **Inventory Management:**
    - **Front-End (Next.js):** Display real-time product availability and stock levels.
    - **Back-End (Sanity CMS):** Store and manage inventory data, syncing with back-end systems.
    - **Third-Party APIs:** Integrate with third-party systems like TradeGecko for inventory management.
- **Order Processing & Fulfillment:**
    - **Front-End (Next.js):** Display real-time order status and allow tracking.
    - **Back-End (Sanity CMS):** Manage order data and update customers on their order status.
    - **Third-Party APIs:** Integrate shipping services like ShipEngine or EasyPost for real-time tracking.

3. Designing System Architecture:

Overview: The system architecture for a general e-commerce marketplace involves multiple components working together seamlessly to provide a functional, scalable, and secure platform.

The following components must interact with each other:

- **User Interface (UI) Layer**
- **Business Logic Layer**
- **Data Storage Layer**
- **Third-Party Integrations**
- **Payment and Shipping Systems**
- **Analytics & Reporting**

High-Level System Architecture:

Main Components:

1. **User Interface (UI) Layer:**
    - Front-end web and mobile application

- Customer-facing interface (Product catalog, cart, checkout process)
- Admin interface (Product management, order tracking, user management)

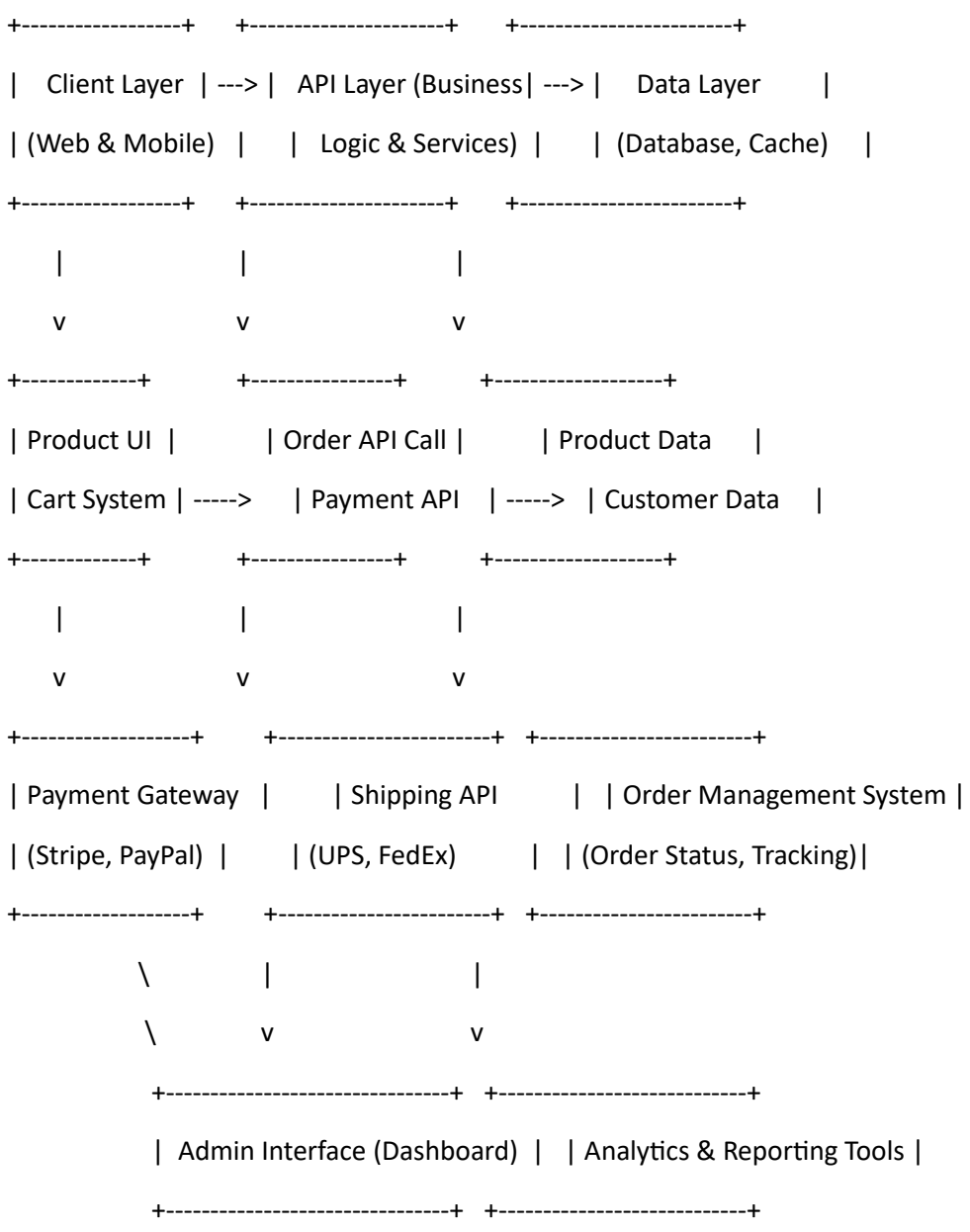
2. **Business Logic Layer:**
   - Cart management, promotions, order processing, inventory management
   - Customer authentication and authorization
3. **Data Storage Layer:**
   - **Inventory Management:** Real-time updates to stock levels and availability
   - **Order Management:** Storage of order status, history, and fulfillment data
   - **Customer Data:** User profiles, preferences, and purchase history
4. **Third-Party Integrations:**
   - **Payment Gateways (e.g., Stripe, PayPal):** Secure payment processing
   - **Shipping Providers (e.g., UPS, FedEx):** Integration for order fulfillment and tracking
5. **Payment and Shipping Systems:**
   - Handling secure transactions and payment processing
   - Integration with shipping services for order fulfillment and tracking

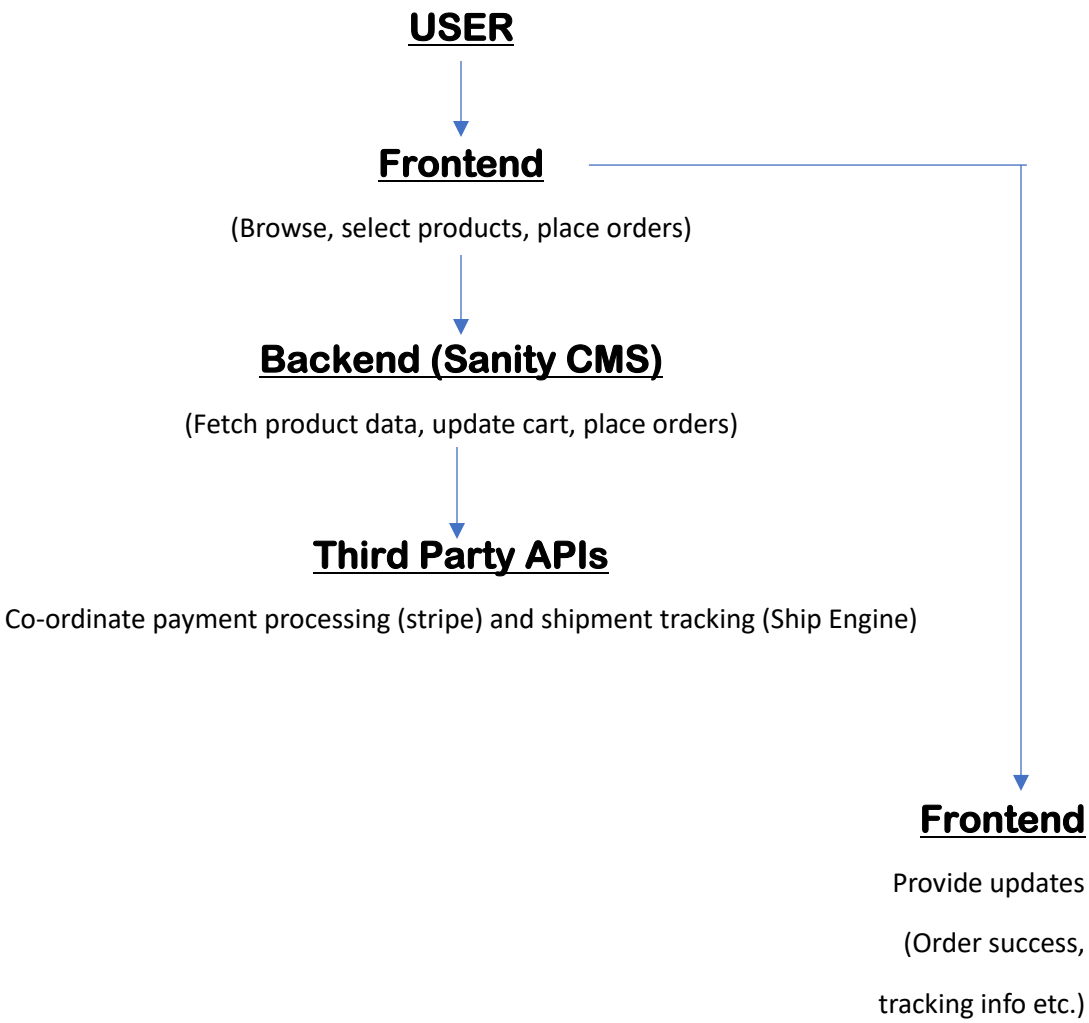
System Architecture Flowchart :

Here's a flow or mind map outline:

1. **User Layer (Front-End & Mobile App)**
   - Interaction through UI (Product Search, Cart, Checkout)
   - Communicates with Business Logic Layer for user operations
2. **Business Logic Layer**
   - Handles operations like cart management, payment processing, etc.
   - Interfaces with Data Storage for product info and customer data
   - Coordinates third-party integrations (Payment, Shipping, Marketing tools)
3. **Data Storage Layer**
   - Product Data (Database for catalog)
   - User Data (Profiles, preferences)
   - Order Data (Order history, statuses)

4. **Payment & Shipping Systems**
    - Payment Gateway: Secure processing (Stripe, PayPal, etc.)
    - Shipping Provider API: Integrates with services like FedEx, UPS
5. **Analytics & Reporting Tools**
    - Real-time data gathering from user interactions and transactions
    - Sends data to Business Intelligence Tools for insights





Key Workflows in E-commerce System:

1. User Registration

- **Step 1:** User clicks on "Register" → **User Inputs Details** (name, email, password).
- **Step 2:** System validates inputs → **Store in Database**.
- **Step 3:** Confirmation message sent to user.

2. Product Browsing

- **Step 1:** User selects product category → **Fetch Products from Database**.
- **Step 2:** User applies filters/search → **Display Products**.
- **Step 3:** User views product details → **Fetch Product Info**.

3. Order Placement

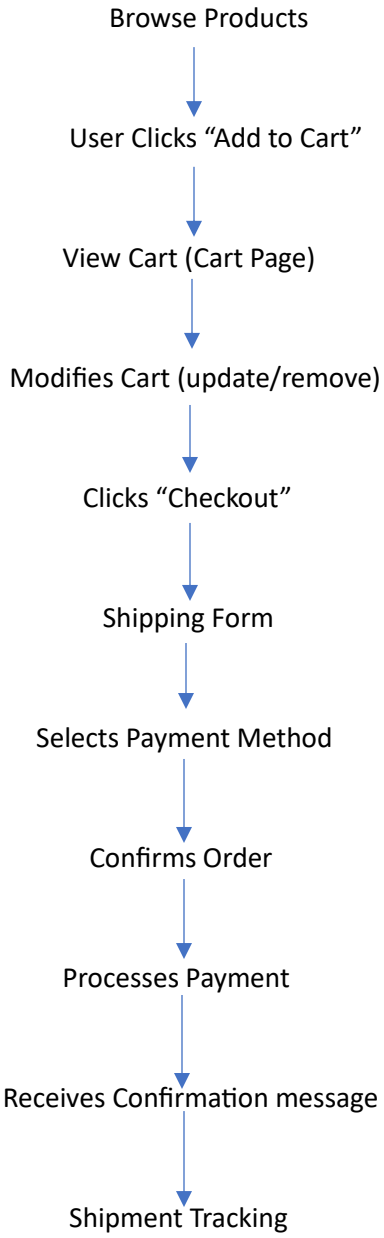
- **Step 1:** User adds products to cart → **Proceed to Checkout**.

- **Step 2:** User enters shipping/payment details → **Process Payment**.
- **Step 3:** Payment successful → **Store Order in Database** → **Send to Shipment API**.

4. Shipment Tracking

- **Step 1:** User receives tracking ID → **Track Order Section**.
- **Step 2:** System fetches tracking info from shipping provider.
- **Step 3:** Display tracking status to user.

User Work Flow



4. API Requirements Plan:

Endpoints:

1. /api/products

- **HTTP Method: GET**
- **Description: Get a list of all products.**
- **Response Format: JSON (Array of Objects)**
- **Response Example:**

[{ "id": 1, "name": "Product A", "price": 100 }, { "id": 2, "name": "Product B", "price": 200 }]

2. /api/products/{id}

- **HTTP Method: GET**
- **Description: Get product details by ID.**
- **Response Example:**

{ "id": 1, "name": "Product A", "price": 100, "description": "Product A description" }

3. /api/cart

- **HTTP Method: GET**
- **Description: Get user's shopping cart items.**
- **Response Format: JSON (Array of Objects)**
- **Response Example:**

[{ "productId": 1, "quantity": 2 }, { "productId": 3, "quantity": 1 }]

4. /api/checkout

- **HTTP Method: POST**
- **Description: Place an order by submitting shipping and payment info.**
- **Response Format: JSON**
- **Response Example:**

{ "message": "Order placed successfully", "orderId": 789 }

5. /api/orders/{id}

- **HTTP Method: GET**
- **Description: Get details of an order by order ID.**
- **Response Format: JSON**
- **Response Example:**

{ "id": 789, "status": "Pending", "total": 500, "products": [{ "id": 1, "name": "Product A", "quantity": 2 }] }

6. /api/orders/{id}/track

- **HTTP Method: GET**
- **Description: Get shipment tracking status.**
- **Response Format: JSON**
- **Response Example:**

{ "status": "Shipped", "estimatedDelivery": "2025-01-25", "trackingId": "TRACK123" }

5. Category-Specific Instructions: General E-Commerce

1. Product Browsing

- **User Actions:**
    - Users can browse the product catalog through categories or search.
    - Filter options such as price, brand, and ratings help narrow choices.
    - Detailed product pages show descriptions, images, prices, and stock availability.
- **Workflow:**
    - **Search Bar**: User inputs keywords.
    - **Categories/Filters**: User selects category or filters.
    - **Product List**: User scrolls through the available products.
    - **Product Detail Page**: User clicks on a product for more information.

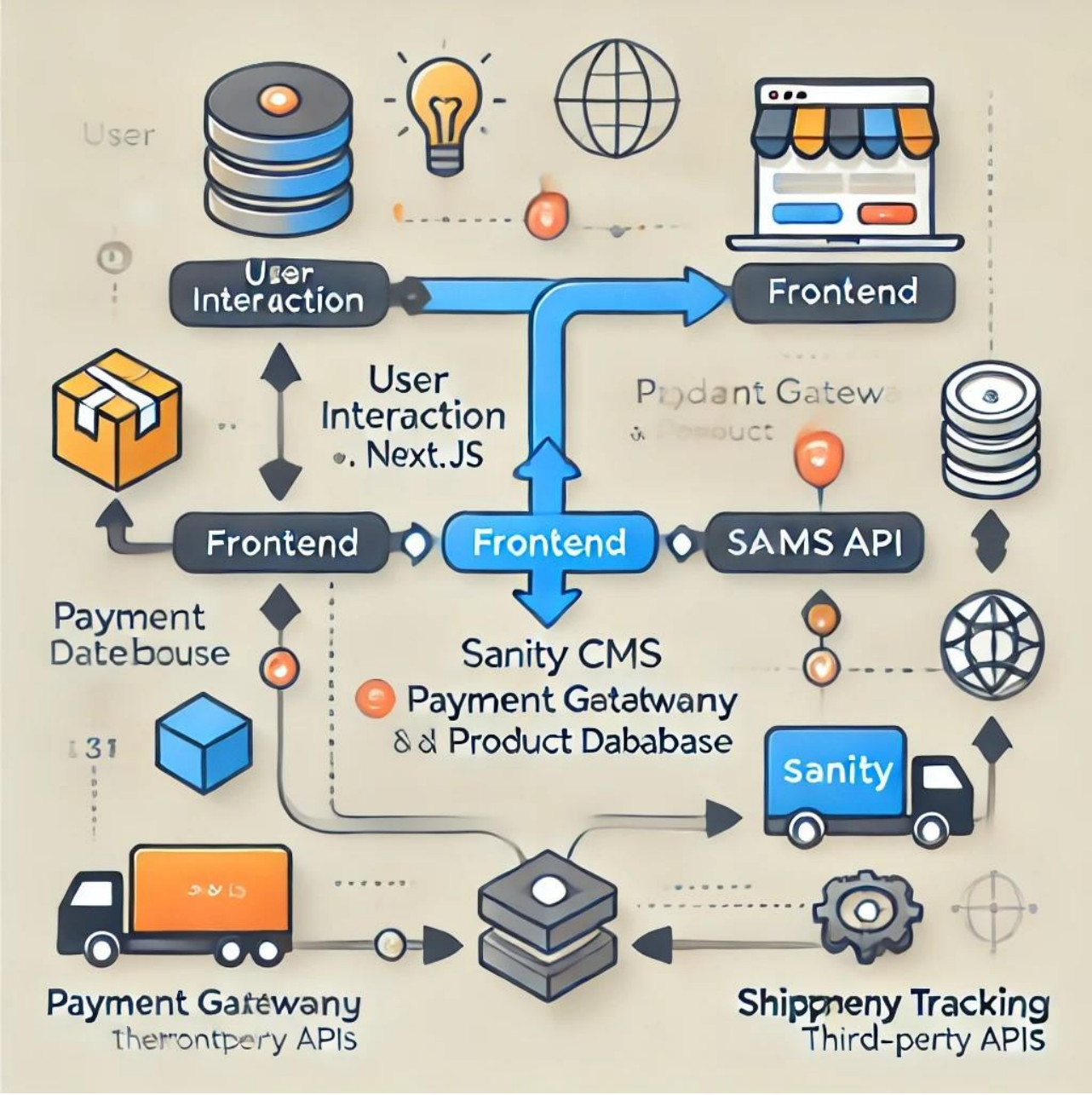
2. Cart Management

- **User Actions:**
    - Users can add products to their shopping cart.
    - The cart is dynamically updated when items are added or removed.
    - Users can adjust quantities or delete items from the cart.
- **Workflow:**
    - **Add to Cart**: User clicks "Add to Cart" on product page.
    - **View Cart**: User accesses the cart to review items.
    - **Modify Cart**: User can update quantities or remove items.
    - **Proceed to Checkout**: User proceeds to payment page.

3. Order Placement

- **User Actions:**
    - After reviewing the cart, users enter shipping and payment details.
    - Users confirm the order before completing the transaction.
- **Workflow:**
    - **Shipping Info**: User provides delivery address.
    - **Payment Info**: User selects payment method (e.g., credit card, PayPal).
    - **Order Confirmation**: User reviews and confirms the order.
    - **Order Success**: User receives a confirmation message and order details.

Here's how the Architecture Works



6.Data Schema Design: General E-Commerce:

1. User Entity	2. Product Entity	3. Order Entity	4. Cart Entity	5. Payment Entity

export default interface User {
userId: number;
firstName: string;
lastName: string;
email: string;
passwordHash: string;
address: string;
phoneNumber: string;
}
	export default interface Product {
	productId: number;
	name: string;
	description: string;
	price: number;
	category: string;
	stock: number;
	imageUrl: string;
	ratings: number;
	}
		export default interface Order {
		orderId: number;
		userId: number;
		orderDate: string;
		status: 'Pending' | 'Shipped' | 'Delivered';
		totalAmount: number;
		shippingAddress: string;
		paymentStatus: 'Paid' | 'Unpaid';
		}
			export default interface Cart {
			cartId: number;
			userId: number;
			products: Array<{
			productId: number;
			quantity: number;
			}>;
			totalAmount: number;
			}
				export default interface Payment {
				paymentId: number;
				orderId: number;
				paymentMethod: 'Credit Card' | 'PayPal' | 'Bank Transfer' | 'Other';
				paymentDate: string;
				paymentStatus: 'Successful' | 'Failed' | 'Pending';}

Conclusion:

In summary, this presentation has covered the essential aspects of designing a **General E-Commerce Marketplace**, including:

- **Business Goals**: Defined clear objectives and translated them into technical requirements.
- **System Architecture**: Presented a high-level system design to efficiently manage the interactions of key components.
- **Key Workflows**: Outlined user registration, product browsing, order placement, and shipment tracking processes.
- **API Requirements**: Defined API endpoints, methods, and responses to ensure seamless communication between the front end and back end.
- **Category-Specific Instructions**: Focused on workflows like product browsing, cart management, and order placement specific to the e-commerce domain.
- **Data Schema Design**: Detailed entity structures for **User**, **Product**, **Order**, **Cart**, **Payment**, and **Shipment** to ensure organized and scalable data management.