Date

* Day 2: Planning The Technical Foundation

• Introduction :-

The technical foundation of any marketplace is critical to its success. Building on the business objectives and data schema define on Day 1, Day 2 focuses on transitioning into the technical domain by creating a robust system architecture, defining workflows, and planning API requirements. This ensures seamless integration between frontend, backend, and third-party services, preparing the marketplace for implementation and scalability.

• Defining Technical Requirements :-

I. Frontend Requirements :-

- User Interface (UI): A visually appealing and user-friendly design for desktop and mobile users.
- Core Pages:
  - → Home
  - → Product Listing
  - → Product Details
  - → Cart
  - → Checkout
  - → Order Confirmation

2. Backend with Sanity CMS :-

- Purpose: Sanity CMS will act as the database, managing products, orders, and customer information.

Signature No

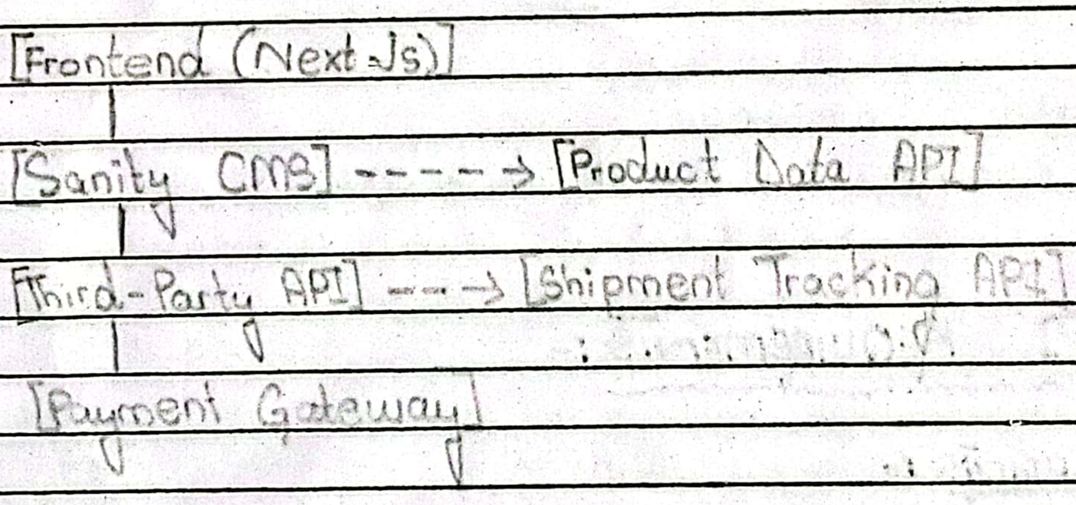
- Schema Design: Define fields for products, Customers, and orders to align with the data schema from Day 1.
- Benefits: Simplifies backend management and ensures seamless integration with the forntend.

3. THIRD-PARTY API INTEGRATION :-

- Shipment Tracking API : Provides real-time delivery updates.
- Payment Gateway API : Ensures secure transactions.
- Additional APIs : Handles marketplace-specific requiremen like inventory updates.

- **SYSTEM ARCHITECTURE :-**

→ HIGH-LEVEL DIAGRAM :-



Date

• WORKFLOWS :-

1. User Registration

    • User Signs up → Data is stored in Sanity → Confirmation sent to the user.

2. Product Browsing

    • User views products → Sanity API fetches data → products displayed dynamically.

3. Order Placement
    • Items added to the cart → Proceed to checkout → order saved in Sanity

4. Shipment Tracking
    • Order updates fetched from a third-party API → Displayed to the user.

5. Payment Processing
    • Payment processed via gateway → Confirmation sent → Recorded in Sanity.

• API REQUIREMENTS :-

ENDPOINTS :-

1. Product Data
    → Endpoint Name : /products
    → Method : GET

→ Description : Fetch all available products
→ Response Example:

```
{"id": 1, "name": "Product A", "price" : 100, "Stock": 50}
```

2. Order Management

→ Endpoint Name : /orders
→ Method : POST
→ Description : Create a new order.
→ Payload :

```
{"CustomerId" : 123, "items": [1, 2], "totalPrice": 200}
```

3. Shipment Tracking

→ Endpoint Name : /shipment
→ Method : GET
→ Description : Track the delivey status
→ Response Example :

```
{"orderId" : 456, "Status" : "In Transit", "ETA" : "20 minutes"}
```

- **TECHNICAL DOCUMENTATION :-**

1. **SYSTEM ARCHITECTURE OVERVIEW :-**

- Diagram : A detailed representation of how the ~~format~~ frontend, backend, and APIs Interact.
- Component Roles :
  → Sanity CMS : Stores and managers data
  → APIs : Facilitates real-time updates and

Date

Secure transactions.

2. WORKFLOW DESCRIPTIONS :-

- Each workflow is broken into steps, detailing how users interact with the system.

3. API SPECIFICATIONS :-

For each endpoint, document :
→ Method
→ URL
→ Payload
→ Example response

4. SANITY SCHEMA EXAMPLE :-

```
export default {
    name: "product",
    type : "document",
    fields : [
      { name: "name", type: "string", title: "Product Name"},
      { name: "price", type: "number", title: "Price"},
      {name: "stock", type: "number", title: "Stock Level"}
    ].
  };
```

5. TECHNICAL ROADMAP :-

- Week 1 : Finalize System architecture and API endpoints.
- Week 2 : Implement Sanity schemas and Connect APIs.
- Week 3 : Test integration and ensure frontend-backend

No

Synchronization.

• **CHALLENGES AND CONSIDERATION :-**

1) Scalability :

•Ensure the system can handle high traffic and data volumn.

2) Real-Time Features :

Implement seamless updates for investory, order tracking, and payments.

3) Security :

Securite payment transactions and user data with encryption.

4) API Rate limits :

Monitor and manage API usage to avoid disruptions.

• **CONCLUSION :-**

Day2 serves as the technical blueprint for implementing the marketplace. By Creating a scalable system architecture, planning workflows, and documenting APIs, this foundation ensures the marketplace is ready for seamless development and deployment. Addressing key challenges further strengthens the system's reliabilit and performance.