Name: NOOR SAHAR

Roll no: 00491703

Hackathon = 3

Day:2

the technical foundation

1.Define Technical Requirements

**Technical Requirements:** refer to the specific technical aspects, tools, systems, and functionalities needed to achieve a project's goals. They outline how the system should function, the technologies required, and the standards to be followed during development. These requirements ensure the solution is feasible, scalable, and aligned with business objectives.

**• Frontend Requirements:**

**1.User-Friendly Interface**

 A simple and intuitive interface for easy browsing of products.

**2.Responsive Design:**

 Ensure the design adapts seamlessly to both mobile and desktop users.

**3.Essential Pages:**

 **Home:** The main page to welcome users and showcase highlights

 **Product Listing:** A page displaying all available products in an organized manner.

 **Product Details:** Detailed information about each individual product.

 **Cart:** A section where users can view the items they have selected.

 **Checkout:** A page to complete the purchase process

 **Order Confirmation:** A page to confirm the successful placement of an order.

* **Sanity CMS as Backend:**

**1.Headless CMS**

• **Sanity** is a headless CMS, which means it only manages content and provides APIs for the frontend. You can use any frontend technology (React, Angular, etc.) to display the content.

**2.Real-time Content Management**:

• **Sanity** allows you to update content in real-time. This features is

especially useful when you need to update content frequently.

**3.API-Driven**:

• **Sanity** has a backend that is API-driven, efficiently transferring data to the

Frontend. It provides both Restful and GraphQL API options.

* **Designing schemas in Sanity to align with the Business Goals:**

we need to focus on creating schemas that will organize this data effectively. Below is a simple design for each of the components that should align with your business goals.

**1.Product Schema**

• This schema will manage all the product-related data.

export default {

name: 'product',

title: 'Product',

type: 'document',

fields: [

{

name: 'title',

title: 'Product Title',

type: 'string',

},

{

name: 'description',

title: 'Description',

type: 'text',

},

{

name: 'price',

title: 'Price',

type: 'number',

},

{

name: 'category',

title: 'Category',

type: 'string', // Can be an array or reference to a category document

},

{

name: 'image',

title: 'Product Image',

type: 'image',

options: {

hotspot: true, // Allows focusing on a part of the image

},

},

{

name: 'stock',

title: 'Stock Quantity',

type: 'number',

},

]

}

 This ensures that each product is listed with all essential details like title, description, price, stock quantity, and images.

### 2.Customer Schema

• This schema will store customer details for future reference.

export default {

name: 'customer',

title: 'Customer',

type: 'document',

fields: [

{

name: 'name',

title: 'Full Name',

type: 'string',

},

{

name: 'email',

title: 'Email Address',

type: 'string',

},

{

name: 'address',

title: 'Shipping Address',

type: 'text',

},

{

name: 'phone',

title: 'Phone Number',

type: 'string',

},

{

name: 'orderHistory',

title: 'Order History',

type: 'array',

of: [{ type: 'reference', to: [{ type: 'order' }] }],

},

]

}

 This stores all necessary customer details, allowing you to track customer orders, contact details, and history.

### 3.****Order Schema****

• This schema will manage the order records placed by customers.

export default {

name: 'order',

title: 'Order',

type: 'document',

fields: [

{

name: 'orderNumber',

title: 'Order Number',

type: 'string',

},

{

name: 'orderDate',

title: 'Order Date',

type: 'datetime',

},

{

name: 'customer',

title: 'Customer',

type: 'reference',

to: [{ type: 'customer' }],

},

{

name: 'products',

title: 'Products in Order',

type: 'array',

of: [{

type: 'reference',

to: [{ type: 'product' }],

}],

},

{

name: 'totalAmount',

title: 'Total Amount',

type: 'number',

},

{

name: 'status',

title: 'Order Status',

type: 'string',

options: {

list: ['Pending', 'Shipped', 'Delivered', 'Cancelled'],

},

},

]

}

• This connects customers with the products they have ordered, tracks the order status, and calculates the total amount for each order.

**• Third-Party APIs:**

To ensure a fully functional and user-friendly marketplace, integrating third-party APIs is essential. Below is the design for integrating APIs for **shipment tracking**, **payment gateways**, and other required backend services.

### ****Shipment Tracking API:****

This integration will allow users to track their order status in real-time.

**Example APIs:**

1. Shipo API
2. EasyPost API

### 2.Payment Gateway API:

Payment gateway integration will enable secure online payments for orders.

**Example APIs:**

1. RazorPay
2. Paypal

• **Additional Backend Services APIs:**

**1.Email Notifications:**

Send order confirmation, shipment updates, and promotional emails.

Use for real-time notifications triggered by user actions.

**2.SMS Notifications:**

Notify users about critical updates like order placement and delivery.

**3.Tax and Currency Conversion:**

Calculate applicable taxes and convert currencies.

**• General API Guidelines:**

1. **Data Availability**:

Shipment status and estimated delivery.

Payment status and transaction history.

Notifications and user alerts.

1. **Security Considerations**:

Use **HTTPS** for secure communication.

Implement **API keys**

1. **Error Handling:**

Implement fallback mechanisms for API failures.

Provide user-friendly error messages on the frontend.

2.Design System Architecture:

Sanity

Frontend

Nextjs

Sanity

CPS

Shipment

Tracking API

Sanity

Product

Data API

Payment

Gateway API

Here is the requested high-level system architecture diagram, showcasing the interaction between the components like Frontend (Next.js), Sanity CMS, Product Data API, Third-Party APIs (Shipment Tracking and Payment Gateway).

 User visits the **Frontend (Next.js)** to browse products.

 Frontend sends a **request to the Product Data API** (powered by Sanity CMS).

 Sanity CMS responds with product listings and details, which are dynamically displayed on the frontend.

 User selects a product and places an order.

 Order details (product info, user details) are sent to **Sanity CMS** via an **API request**, where the order is recorded.

 Payment details are securely sent to a **Payment Gateway** (e.g., Stripe, PayPal).

 The Payment Gateway processes the payment and sends a **confirmation response** back to the frontend.

 Payment confirmation is recorded in **Sanity CMS**.

 Once the order is shipped, tracking details are fetched from a **Third-Party Shipment API** (e.g., FedEx, Shippo).

 Real-time tracking updates are displayed on the frontend for the user.

**• Example System Architecture:**

 **Frontend** (Next.js) sends requests to **Sanity CMS** via API to fetch product and content data.

 **Frontend** communicates with **Payment Gateway API** to handle transactions securely.

 **Frontend** interacts with **Shipment Tracking API** to provide real-time updates on order delivery.

 **Product Data API** links **Sanity CMS** with the **Frontend** to keep everything in sync.

**• Key Workflows to Include:**

**1. User Browses the Platform:**

 **Frontend (Next.js)** displays the homepage, product listings, and categories using data fetched from **Sanity CMS** via **Product Data API.**

**2. User Selects a Product:**

 When a user selects a product, the **Frontend** fetches detailed product information (like price, description, availability) from **Sanity CMS** using the **Product Data API**.

**3. User Adds Product to Cart:**

 User adds the product to the cart, and the **Frontend** stores the selected items in the session or cart state.

**4. Checkout Process:**

* User selects the **Payment Gateway API** to make payment securely.

**5. Payment Processing:**

 **Frontend** interacts with the **Payment Gateway API** to securely process the payment.

**6. Order Confirmation and Shipment:**

 The order details, including customer and product info, are sent to **Sanity CMS** or a backend system to store the order.

**7. Shipment Tracking:**

 After the order is confirmed, **Frontend** calls the **Shipment Tracking API** to provide real-time tracking information about the order's shipment.

**8. User Reviews and Feedback:**

* Reviews are managed via **Sanity CMS** and displayed dynamically on the product page via **Frontend**.

**9. Order History:**

* **Frontend** queries **Sanity CMS** for past order records and displays them for the user.

**10. Notifications:**

 Email/SMS notifications are sent to the customer for key actions, such

 Payment status  Order confirmation

 Shipment update  Delivery confirmation

This **key workflow** ensures a smooth interaction between the user, the **Frontend (Next.js)**, **Sanity CMS**, and third-party APIs like **payment gateways** and **shipment tracking**. It covers all the major steps from browsing products to managing orders and customer feedback.

3.Plan API Requirements:

**• General E-Commerce Example:**

 **Purpose**: Fetch a list of all products in the store.

 **Method**: GET

 **Endpoint Name**: /products

 **Description**: This endpoint fetches a list of all products available on the platform. You can filter the results by various parameters such as category, price range, or availability status.

[

{

"id": "123",

"name": "Product Name",

"description": "Product description",

"price": 29.99,

"category": "Electronics",

"availability": true

}

]

To ensure that the API documentation aligns with **marketplace-specific workflows** and provides clarity for **implementation**, you need to carefully define the API endpoints based on your data schema.

 **Product:**

 **Purpose**: Fetch all available products in the marketplace.

[

{

"id": "123",

"name": "Product Name",

"description": "Product description",

"price": 29.99,

"category": "Electronics",

"availability": true

}

]

 **Order:**

 **Purpose**: Place a new order.

{

"userId": "u123",

"productId": "123",

"quantity": 2,

"shippingAddress": "123 Main Street, City, Country",

"paymentMethod": "Credit Card",

"totalPrice": 59.98

}

 **Shipment:**

 **Purpose**: Get shipment status using the tracking number.

{

"trackingNumber": "TRACK123",

"status": "In Transit",

"estimatedDeliveryDate": "2025-01-25"

}

### 