HACKATHON-03
DAY-02
Planning the Technical Foundation

1. Define Technical Requirements

01: Frontend Requirements:

Framework & Styling: Building a Fast, Modern Experience 🚀

We've built our platform using **Next.js**, ensuring a fast, dynamic, and SEO-friendly shopping experience. For styling, **Tailwind CSS** gives us the flexibility to create sleek, responsive designs effortlessly, keeping the interface clean and user-friendly.

Seamless Navigation & Routing 🔗

Navigating our platform is smooth and intuitive, thanks to **Next.js's file-based routing**. Whether it's static pages, dynamic product listings, or nested routes, everything is designed to be user-friendly—so you can find what you need without any hassle.

Optimized for Speed & Performance ⚡

To make browsing lightning-fast, we use:

- **Server-Side Rendering (SSR)** for critical pages like the homepage and product listings, ensuring they load quickly and rank well on search engines.
- **Static Site Generation (SSG)** for pages like FAQs, providing instant load times.
- **Incremental Static Regeneration (ISR)** to update content without slowing things down—keeping product details fresh without constant site rebuilds.

02: Sanity CMS as Backend:

Sanity CMS: The Heart of Our Content Management 🛠️

Sanity CMS is the central hub for managing all content on our platform—whether it's **products, categories, blogs, or promotional banners**.

Why Sanity?

✅ **Flexible & Structured** – Its dynamic content model allows us to define and organize data seamlessly.
✅ **Real-Time Collaboration** – Multiple team members can edit content at the same time, ensuring efficiency.

✅ **Instant Updates** – Any changes are instantly reflected on the frontend, keeping information fresh and accurate for users.

03: Third-Party APIs:

Payment Gateways: Handle secure transactions and multiple payment options. Stripe

Shipping and Logistics: Provide real-time shipping rates, tracking, and label generation. ShipEngine

Cart and Checkout: Snipcart is a powerful and easy-to-integrate solution for adding cart and checkout functionality to any website or e-commerce platform.

Inventory Management: Tracks product stock levels directly within Snipcart. Updates inventory in real-time as customers complete purchases.

Design System Architecture

[frontend (next.js)]

| | |

|

[Sanity CMS] **[Third-Party APIs]** **[Payment Gateway]**

[Product Data (API)] **[Shipment Tracking(API)]** **[stripe(payment)]**

Detailed Key Workflows

User Registration:

A new user signs up on the platform.

The user's data (e.g., name, email, password) is securely stored in Sanity CMS.

A confirmation email is sent to the user to verify their account.

Product Browsing:

Users explore product categories and use filters to refine their search.

The frontend fetches product data dynamically via the Sanity API.

Products, along with images, descriptions, and prices, are displayed seamlessly to the user.

Order Placement:

The user selects items to add to their shopping cart.

At checkout, payment details are entered, and the order is processed securely.

Order details, including products, customer info, and payment status, are saved in Sanity CMS for tracking.

Shipment Tracking:

The system fetches order status updates (e.g., "-ShipEngine," "In Transit") using a third-party shipping API.

Real-time shipment tracking details are displayed to the user on their order history or tracking page.

Payment Processing:

The selected payment method is securely processed through a **third-party payment gateway** (e.g., Stripe).

Fraud prevention and validation checks are performed during the transaction.

Plan API Requirements:

1: All products fetch:

Endpoint Name:/Product

Method: Get

Description: This endpoint provides real-time updates on the delivery status of perishable items, ensuring that customers can track the delivery progress for time-sensitive orders.

Response Example

[{

"orderId": 123,

"name": "Iphone-15",

"image": "image(url)",

"Price": 20000,

}]

02: Create a new order

Endpoint Name:/order

Method: post

Description: This endpoint is used to create a new order in the system and store the relevant details in Sanity

Payload Example:

{

"Customer Info": {
"name": "John Doe",
"email": "john@example.com",
"address": "123 Main St, City, Country",
"phone": "123-456-7890"
},
"products": [{
"productid": 1,
"ProductName": " Headphones",
"quantity": 2,
"price": 200 },
{
"productid": 2,
"ProductName": "Bluetooth Speaker",
"quantity": 1,
"price": 5000
}],
"Payment Status": "Paid"
}
Response Example:
{“order Id”:” ORD111”,” status”: “Order Created”}

03: Tracking and shipment
Endpoint Name:/Shipment
Method: GET
Description: This endpoint allows users to track the status of their orders by fetching real-time shipment updates via a third-party API. It provides details such as the shipment ID, order ID, status, and the expected delivery date.
Response Example:
{
"shipmentId": "ABC123456",
"orderId": 123,
"status": "Out for Delivery",
"expectedDeliveryDate": "2025-01-18"

}

Sanity Schema Example:

Product schema:

```
import { defineType, defineField } from "sanity"

export const product = defineType({
    name: "product",
    title: "Product",
    type: "document",
    fields: [
        defineField({
            name: "title",
            title: "Title",
            validation: (rule) => rule.required(),
            type: "string"
        }),
        defineField({
            name: "slug",
            type: "slug",
            title: "Slug",
            options: {
              source: "title",
              maxLength: 96,
            },
          },),
        defineField({
            name: "productImage",
            type: "image",
            validation: (rule) => rule.required(),
            title: "Product Image"
        }),
        defineField({
            name: "price",
            type: "number",
            validation: (rule) => rule.required(),
            title: "Price",
        }),
        defineField({
            name: "quantity",
```

```
            title: "Quantity",
            type: "number",
            validation: (rule) => rule.min(0),
          }),
        defineField({
            name: "tags",
            type: "array",
            title: "Tags",
            of:[{
                type: "string"
            }]
        }),
        defineField({
            name: 'description',
            title: 'Description',
            type: 'text',
            description: 'Detailed description of the product',
          }),
          defineField({
            name: 'features',
            title: 'Features',
            type: 'array',
            of: [{ type: 'string' }],
            description: 'List of key features of the product',
          }),
          defineField({
            name: 'dimensions',
            title: 'Dimensions',
            type: 'object',
            fields: [
              { name: 'height', title: 'Height', type: 'string' },
              { name: 'width', title: 'Width', type: 'string' },
              { name: 'depth', title: 'Depth', type: 'string' },
            ],
            description: 'Dimensions of the product',
          }),
    ]
})
```