DAY 02: PLANNING THE TECHNICAL FOUNDATION

Day 2 My Goal

Today's goal was to transition from business planning to technical preparation for the marketplace. I successfully created a high-level technical plan, including system architecture, workflows, and API requirements. This plan will guide the implementation phase and ensure that the development process aligns with business objectives, leveraging tools like Sanity CMS and third-party APIs to create an efficient, scalable solution.

Recap of Day 01: Business Focus

E-Commerce Project

MarketPlace Name : HomeStyle Market (Furniture/Home Decor)

On Day 01, I focused on laying the foundation for my exciting e-commerce venture. I chose to build a marketplace where customers can explore, customize, and purchase home decor items, furniture, and other related products. The goal was to focus on understanding the business requirements before moving into the technical phase. This step ensured that the marketplace would be versatile, catering to a wide range of customer needs.

1. **Business Goal Defined:**

- Affordable and stylish Products (different style , unique design ,affordable price).
- Sell products at wholesale price for shops (provided the quantity is large enough).
- Timely delivery (with affordable rate , flexible option).
- Dedicated brand pages (high quality brand).
- Support for local Artisans (small businesses, global customer increasing their visibility & sales)
- Reliable platform (secure payment option , detailed product descriptions).
- Enhanced customer support.(help center).

- Target audience for my marketplace

    . Homeowners
    . Renters
    . Shop owners
    . First time buyers
    . Small business
    . Artisans
    . Brand conscious customer
    . Budget conscious shopper & customer

2. **Data Schema drafted**

[Product]
- Product ID
- Name
- Price

- Stock Quantity
- Category
- Description
- Image URLs
- Brand
- Wholesale Price (Optional)
- Wholesale Quantity (Optional)

[Order]
- Order ID
- Customer ID (Reference to [Customer])
- Product ID (Reference to [Product])
- Quantity
- Total Price
- Order Status (Pending, Shipped, Delivered)
- Payment Status (Pending, Completed, Failed)
- Shipping Address
- Order Date

[Customer]
- Customer ID
- Name
- Email
- Contact Info (Phone, Address)
- Business Name (Optional)
- Is Wholesale Customer (Optional)

[Shipment]
- Shipment ID
- Order ID (Reference to [Order])
- Tracking Number
- Shipment Status (In Transit, Delivered, Failed)
- Shipment Date
- Delivery Date (Optional)

[Delivery Zone]
- Zone ID
- Zone Name
- Coverage Area
- Assigned Driver
- Estimated Delivery Time

- Zone Details

Explanation of the Relationships:

1. Product -> Order: A product can be associated with many orders, while each order contains a reference to the product.

2. Order -> Customer: An order is placed by a customer. Each order is associated with one customer.

3. Order -> Shipment: Each order will be shipped through a shipment. A shipment is directly related to one order.

4. Shipment -> Delivery Zone: A shipment is assigned to a specific delivery zone, which defines the area and logistics details for the shipment.

Product —> Order —> Customer

|

—>Shipment—>Delivery Zone

3. **Single Focus:**

   I completely agree that focusing on business requirements before diving into technical aspects helps establish a strong foundation. This approach ensures that the marketplace is developed in an organized and efficient manner, aligning better with the business needs. I also plan to adopt this principle for my marketplace

Day 2 Activities: Transitioning to Technical Planning

1. Define Technical Requirements

To build a user-friendly interface and a fully functional e-commerce website, here's I can describe my approach step by step, highlighting the tools and technologies I'll use:

Frontend Development Approach

1. **Framework: Next.js**

I will use Next.js because it provides server-side rendering (SSR) and static site generation (SSG), ensuring fast page load times and better SEO for my marketplace.

Implementation:

I will create dynamic routes for pages like product details using Next.js' file-based routing system.

I will fetch data from Sanity CMS and display it dynamically using Next.js APIs like getStaticProps and getServerSideProps.

2. **Styling: TailwindCSS**

I'll use TailwindCSS to design a responsive and modern UI quickly. It provides utility-first classes that reduce the need to write custom CSS.

Implementation:

I will use Flexbox and Grid utilities for layouts (e.g., product listing grid).

I will ensure responsiveness with Tailwind's breakpoints like sm, md, and lg.

3. **Interactive Components**:

To make the interface user-friendly, I'll use React components in Next.js for features like dropdowns, carousels, and modals.

Tools:

For interactivity, I'll use built-in React hooks like useState and useEffect.

I may use libraries like React Icons for adding icons and Headless UI for accessible components like dropdowns and modals.

Backend Development Approach

1. **Database: Sanity CMS**

I'll use Sanity CMS as a headless CMS to manage product, order, and customer data efficiently.

Implementation:

I will define schemas in Sanity for products, categories, customers, and orders.

I'll use Sanity Studio for adding, editing, and deleting content directly through the admin interface.

2. **Shipment and Order Tracking:**

API Used: EasyPost or Shippo

These APIs provide real-time shipment tracking and order status updates.

Implementation:

I'll integrate the API in Next.js to fetch tracking details and update the order status in Sanity CMS.

I will display live tracking details on the Order Confirmation Page.

3. **Payment Gateway:**

API Used: Stripe

Stripe is secure and user-friendly for handling online payments.

Implementation:

I'll integrate Stripe's API to handle payment processing.

On successful payment, order details will be stored in Sanity CMS.

My Step-by-Step Workflow

1. **Frontend Development:**

I'll build pages like Home, Product Listing, Product Details, Cart, and Checkout using Next.js.

I will use TailwindCSS to make the website responsive and visually appealing.

2. **Backend Setup with Sanity CMS:**

I'll create schemas in Sanity for products, orders, customers, and categories.

I'll use Sanity's GROQ queries to fetch data for dynamic routes.

3. **API Integrations:**

Shipment Tracking: Integrate EasyPost API to update the shipment status in real-time.

Payment Gateway: Integrate Stripe API to process payments.

Product Data Management: Use Sanity APIs to push data dynamically from APIs like AliExpress.

4. **Testing and Deployment:**

I'll test the website's functionality on mobile, tablet, and desktop using Tailwind's responsive utilities.

Finally, I'll deploy the website using Vercel for fast and reliable hosting.

2 .Design System Architecture

Here's a step-by-step guide on how to data flow for the e-commerce system, with the process, and specific API usage for each step.

1.API Data Fetch (Frontend Request)

Step: The user opens the frontend (Next.js) and begins browsing products.

Action: Frontend makes a GET request to the Product Data API, which retrieves the data (e.g., product names, descriptions, prices, images…etc) from Sanity CMS.

API: Sanity CMS (Product Data API)

Use: GET request to fetch product data from the CMS.

Example: GET /products to get all products.

2. Sanity CMS Processes Data

Step: The Product Data API fetches the required data from Sanity CMS (such as product lists).

Action: Sanity CMS queries the database and sends the data back to the frontend.

API: Sanity CMS

Use: Fetch dynamic content from the headless CMS, which stores product data.

Example: Using Sanity client to fetch product details.

3. Display Products on Frontend (User View)

Step: The frontend (Next.js) receives the data from Sanity CMS.

Action: It then displays the products on the browser dynamically (names, images, prices, descriptions).

4. User Places Order (Order Data)

Step: The user adds products to the cart and proceeds to checkout.

Action: Order details (product IDs, quantity, customer information) are sent from the frontend to Sanity CMS to be stored.

API: Sanity CMS

Use: POST request to save order data (order ID, customer info).

Example: POST /orders to save the order details in Sanity.

5. Payment Information (Frontend to Payment Gateway)

Step: The user proceeds to make payment and enters payment details.

Action: The frontend sends payment information (such as credit card details) to the Payment Gateway for processing.

API: Payment Gateway (e.g., Stripe, PayPal)

Use: POST request to process the payment securely.

Example: POST /payments to process the payment.

6. Payment Confirmation

Step: The Payment Gateway processes the payment and confirms whether it was successful or failed.

Action: Payment confirmation is sent back to the frontend and also recorded in Sanity CMS for order tracking.

API: Sanity CMS (for saving payment status)

Use: POST or PUT to update payment status in the order record.

Example: PUT /orders/{orderId} to update payment status to "paid."

7. Shipment Tracking (Order Confirmation)

Step: After successful payment, the order is processed, and shipment tracking information is required.

Action: The frontend requests shipment tracking details from the Shipment Tracking API (via third-party API) to track the status of the delivery.

API: Shipment Tracking API (e.g., EasyPost, Shippo)

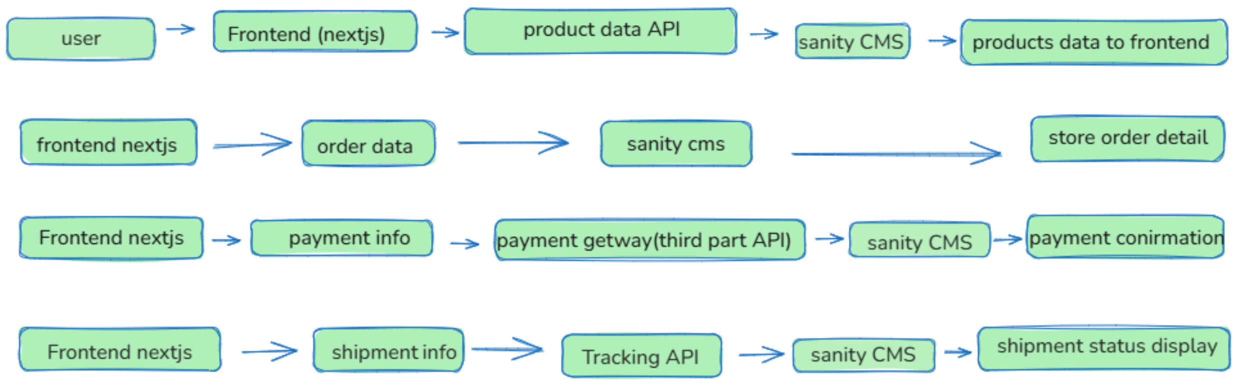
Use: GET request to fetch real-time shipment status.

Example: GET /tracking/{trackingNumber} to get the shipment status.

8. Display Shipment Tracking (Frontend User)

Step: Once the shipment tracking data is fetched, the frontend dynamically updates the user's page with the tracking information (e.g., delivery date, shipping status).

A clear diagram illustrating how the frontend interacts with Sanity CMS and third-party APIs.



<u>System Architecture Diagram Breakdown:</u>

1. Frontend (Next.js):

Receives requests and displays dynamic data (products, order details, payment status, shipment tracking).

Interacts with Sanity CMS and third-party APIs (shipment, payment) to fetch and display data.

2. Sanity CMS:

Stores content data (products, orders, and customer info).

Handles API requests from the frontend for product listings and order processing.

3. Product Data API:

Retrieves product details from Sanity CMS and serves them to the frontend.

4. Payment Gateway:

Processes payment information and returns the payment status (e.g., Stripe or PayPal).

Updates Sanity CMS with the payment confirmation.

5. Shipment Tracking API:

Provides real-time shipment tracking data.

Updates the frontend with the current status of the delivery.

APIs I Can Use:

1. Product Data: Sanity CMS API (for fetching product listings)

2. Order Data: Sanity CMS API (for storing order and customer info)

3. Payment: Stripe API (for payment processing)

4. Shipment Tracking: Shippo API (for tracking shipment status)

3 .Plan API Requirements

Here's a detailed API requirements document based on my schema and requirements:

1. API Endpoints Documentation

Endpoint Name: /products

Method: GET

Description: Fetch all available products from Sanity CMS.

Response Example:

```
  "id": 1,
  "name": "Product A",
  "price": 100,
  "stock": 50,
  "category": "Furniture",
  "description": "A high-quality chair.",
  "image": ["image_url_1", "image_url_2"],
  "brand": "Brand A",
  "wholesale_price": 80,
  "wholesale_quantity": 10
```

Endpoint Name: /orders

Method: POST

Description: Create a new order in Sanity CMS.

Payload:

```
 "customer_id": 123,
 "product_id": 1,
 "quantity": 2,
 "total_price": 200,
 "order_status": "Pending",
 "payment_status": "Pending",
 "shipping_address": "123 Street, City, Country",
 "order_date": "2025-01-17T10:00:00"
```

Response Example:

```
{
  "order_id": 456,
"status": "Success"
```

Endpoint Name: /shipment

Method: GET

Description: Track the status of an order via third-party shipment API.

Response Example:

```
  "shipment_id": 789,
  "order_id": 456,
  "status": "In Transit",
  "expected_delivery_date": "2025-01-20"
```

Endpoint Name: /delivery-zone

Method: GET

Description: Fetch delivery zone details for an order.

Response Example:

```
  "zone_id": 1,
  "zone_name": "Urban Area",
  "coverage_area": "City Center",
  "assigned_driver": "Driver A",
  "estimated_delivery_time": "3 hours",
  "zone_details": "Available for all products"
```

2. Schema Overview

[Product]

Fields: Product ID, Name, Price, Stock Quantity, Category, Description, Image URLs, Brand, Wholesale Price (Optional), Wholesale Quantity (Optional)

Relationships:

Product ID is referenced by the Order entity.

[Order]

Fields: Order ID, Customer ID, Product ID, Quantity, Total Price, Order Status, Payment Status, Shipping Address, Order Date

Relationships:

Customer ID is referenced from the Customer entity.

Product ID is referenced from the Product entity.

[Customer]

Fields: Customer ID, Name, Email, Contact Info, Business Name (Optional), Is Wholesale Customer (Optional)

Relationships:

Customer ID is referenced by the Order entity.

[Shipment]

Fields: Shipment ID, Order ID, Tracking Number, Shipment Status, Shipment Date, Delivery Date

Relationships:

Order ID is referenced from the Order entity.

[Delivery Zone]

Fields: Zone ID, Zone Name, Coverage Area, Assigned Driver, Estimated Delivery Time, Zone Details

Relationships: No relationships needed.

3. Implementation Notes:

I Ensure that the /products endpoint fetches data from Sanity CMS.

/orders and /shipment endpoints should interact with Sanity CMS and third-party APIs respectively for order and delivery updates.
Consider adding error handling (e.g., for invalid product IDs, failed shipments) in all API responses.

4. Create Schema

Here's the schema for my e-commerce marketplace, designed to be used with Sanity CMS. The structure is based on your original requirements and adapted to work with Sanity's flexible schema system.

1. Product Schema

```
export default {
  name: 'product',
  title: 'Product',
  type: 'document',
  fields: [
    {
      name: 'productID',
      title: 'Product ID',
      type: 'string',
      validation: Rule => Rule.required().min(1)
    },
    {
      name: 'name',
      title: 'Product Name',
      type: 'string',
      validation: Rule => Rule.required().min(1)
    },
    {
      name: 'price',
      title: 'Price',
      type: 'number',
      validation: Rule => Rule.required().positive()
    },
    {
      name: 'stockQuantity',
      title: 'Stock Quantity',
      type: 'number',
      validation: Rule => Rule.required().integer().positive()
    },
    {
      name: 'category',
      title: 'Category',
      type: 'string',
      validation: Rule => Rule.required()
    },
    {
      name: 'description',
      title: 'Description',
      type: 'text'
    },
    {
      name: 'imageURLs',
      title: 'Image URLs',
      type: 'array',
      of: [{ type: 'url' }]
    },
    {
      name: 'brand',
      title: 'Brand',
      type: 'string'
    },
    {
      name: 'wholesalePrice',
      title: 'Wholesale Price',
      type: 'number'
    },
    {
      name: 'wholesaleQuantity',
      title: 'Wholesale Quantity',
      type: 'number'
    }
  ]
}
```

2. Order Schema

```
export default {
  name: 'order',
  title: 'Order',
  type: 'document',
  fields: [
    {
      name: 'customerID',
      title: 'Customer ID',
      type: 'reference',
      to: [{ type: 'customer' }],
      validation: Rule => Rule.required()
    },
    {
      name: 'productID',
      title: 'Product ID',
      type: 'reference',
      to: [{ type: 'product' }],
      validation: Rule => Rule.required()
    },
    {
      name: 'quantity',
      title: 'Quantity',
      type: 'number',
      validation: Rule => Rule.required().positive()
    },
    {
      name: 'totalPrice',
      title: 'Total Price',
      type: 'number',
      validation: Rule => Rule.required().positive()
    },
    {
      name: 'orderStatus',
      title: 'Order Status',
      type: 'string',
      options: {
        list: [
          { title: 'Pending', value: 'pending' },
          { title: 'Shipped', value: 'shipped' },
          { title: 'Delivered', value: 'delivered' }
        ]
      },
      validation: Rule => Rule.required()
    },
    {
      name: 'paymentStatus',
      title: 'Payment Status',
      type: 'string',
      options: {
        list: [
          { title: 'Pending', value: 'pending' },
          { title: 'Completed', value: 'completed' },
          { title: 'Failed', value: 'failed' }
        ]
      },
      validation: Rule => Rule.required()
    },
    {
      name: 'shippingAddress',
      title: 'Shipping Address',
      type: 'text',
      validation: Rule => Rule.required()
    },
    {
      name: 'orderDate',
      title: 'Order Date',
      type: 'datetime',
      validation: Rule => Rule.required()
    }
  ]}
```

3. Customer Schema

```
export default {
  name: 'customer',
  title: 'Customer',
  type: 'document',
  fields: [
    {
      name: 'name',
      title: 'Name',
      type: 'string',
      validation: Rule => Rule.required().min(1)
    },
    {
      name: 'email',
      title: 'Email',
      type: 'string',
      validation: Rule => Rule.required().email()
    },
    {
      name: 'contactInfo',
      title: 'Contact Info',
      type: 'object',
      fields: [
        { name: 'phone', title: 'Phone', type: 'string' },
        { name: 'address', title: 'Address', type: 'string', validation: Rule => Rule.required() }
      ]
    },
    {
      name: 'businessName',
      title: 'Business Name',
      type: 'string'
    },
    {
      name: 'isWholesaleCustomer',
      title: 'Wholesale Customer',
      type: 'boolean',
      default: false
    }
  ]
}
```

4. Shipment Schema

```
export default {
  name: 'shipment',
  title: 'Shipment',
  type: 'document',
  fields: [
    {
      name: 'orderID',
      title: 'Order ID',
      type: 'reference',
      to: [{ type: 'order' }],
      validation: Rule => Rule.required()
    },
    {
      name: 'trackingNumber',
      title: 'Tracking Number',
      type: 'string',
      validation: Rule => Rule.required()
    },
    {
      name: 'shipmentStatus',
      title: 'Shipment Status',
      type: 'string',
      options: {
        list: [
          { title: 'In Transit', value: 'in_transit' },
          { title: 'Delivered', value: 'delivered' },
          { title: 'Failed', value: 'failed' }
        ]
      },
      validation: Rule => Rule.required()
    },
    {
      name: 'shipmentDate',
      title: 'Shipment Date',
      type: 'datetime',
      validation: Rule => Rule.required()
    },
    {
      name: 'deliveryDate',
      title: 'Delivery Date',
      type: 'datetime'
    }
  ]
}
```

5. Delivery Zone Schema

```
export default {
  name: 'deliveryZone',
  title: 'Delivery Zone',
  type: 'document',
  fields: [
    {
      name: 'zoneName',
      title: 'Zone Name',
      type: 'string',
      validation: Rule => Rule.required()
    },
    {
      name: 'coverageArea',
      title: 'Coverage Area',
      type: 'string',
      validation: Rule => Rule.required()
    },
    {
      name: 'assignedDriver',
      title: 'Assigned Driver',
      type: 'string'
    },
    {
      name: 'estimatedDeliveryTime',
      title: 'Estimated Delivery Time',
      type: 'string'
    },
    {
      name: 'zoneDetails',
      title: 'Zone Details',
      type: 'text'
    }
  ]
}
```

Importing Schemas in Sanity

To use these schemas in Sanity, make sure you import and export them properly in your schema.ts file:

```
// schemas/schema.ts
import product from './product';
import order from './order';
import customer from './customer';
import shipment from './shipment';
import deliveryZone from './deliveryZone';
```

Explanation

References: Many fields, such as customerID and productID, reference other documents using the type: 'reference' property. This links the product to an order, and an order to a customer.
Validation: I've included basic validation rules, such as required fields and positive numbers for prices and quantities.

Datetime: The orderDate, shipmentDate, and deliveryDate fields are set as datetime, allowing me to capture exact times.

This schema structure will allow me to manage the key components of myr marketplace effectively using Sanity CMS.

………… XXX…………

I have thoughtfully planned the technical foundation for how I will transform my e-commerce website into a thriving marketplace. My strategy includes implementing all these features and ensuring their execution during the upcoming hackathon.

About Me

My name is Saira, and I am a student of the Tuesday 7–10 PM time slot, learning under the guidance of Sir Ali Jawwad.

These documents have been prepared as per the task assigned on Day 2 of Hackathon 3 by Sir Ameen. I have carefully followed the given instructions to ensure everything is accurately completed.

I have completed all the work on my own as I prefer to work late at night. During that time, my peers are usually busy, so I was unable to receive feedback from them. Therefore, I carried out the tasks independently, ensuring they were done to the best of my ability.

And I am ready for the next day's task.