DAY 2 PLANNING THE TECHNICAL FOUNDATION

Prepared by Ameen Alam

Completed by Syeda Aliza Masood

First published: PKT 11:08 AM, January 15, 2025

Document Revision Information

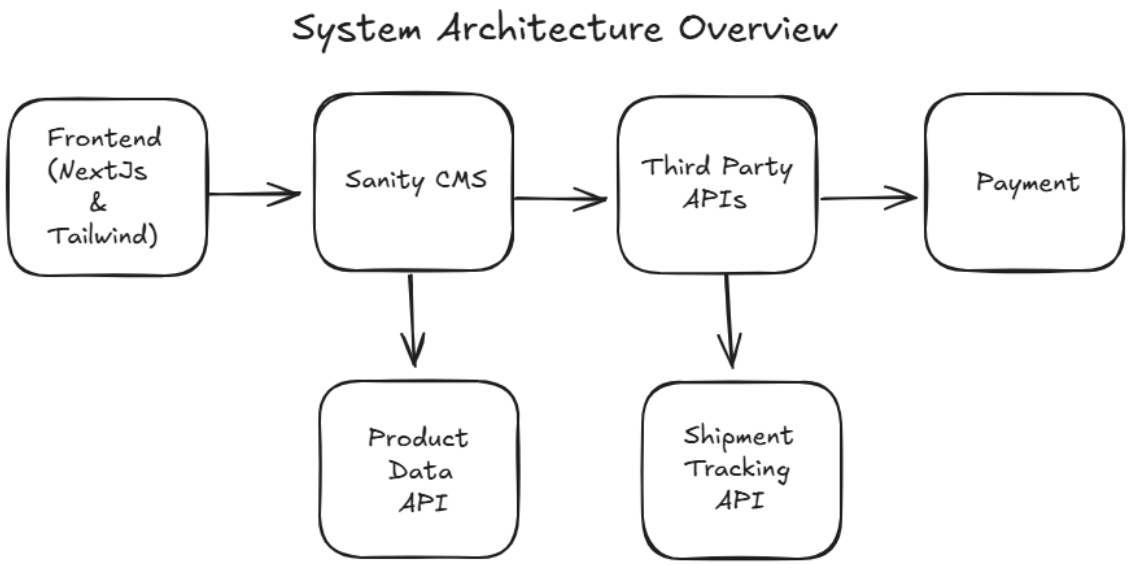
Version	Date	Amendment	Author
1.0	11:08 AM, January 15, 2025	Initial release of Day1	Ameen Alam
1.1	03:00 PM, January 15, 2025	Added Examples in Day1	Ameen Alam
2.0	01:00 PM, January 16, 2025	Day 2 Release	Ameen Alam

Table of Contents

Marketplace Technical Foundation

1. System Architecture Overview

- **Diagram**




```
[Frontend (Next.js)]
        |
[Sanity CMS] ---------> [Product Data API]
        |
[Third-Party APIs] -----> [Shipment Tracking API]
        |
[Payment Gateway]
```

- **Component Descriptions**

Frontend (Next.js): Provides a user-friendly interface for browsing products, managing carts, and completing purchases.

Sanity CMS: Acts as the backend for managing product listings, customer data, and orders.

Third-Party APIs: Integrates external features such as shipment tracking and external logistics services.

Payment Gateway: Handles secure online transactions and processes payment confirmations.

2. Key Workflows

Workflow 1: Browsing Products

1. User visits the homepage.
2. Frontend sends a request to the /products API.
3. Sanity CMS responds with product details, which are dynamically displayed on the frontend.

Workflow 2: Adding to Cart

1. User selects a product and specifies the quantity.
2. The selected product is temporarily stored in the cart (using local storage or session data).

Workflow 3: Order Placement

1. User completes checkout, providing customer information and payment details.
2. Order details (products, user info, address) are sent to Sanity CMS via the /orders API.
3. Payment is processed through the payment gateway, and a confirmation message is displayed to the user.

Workflow 4: Shipment Tracking

1. Once the order is shipped, the frontend queries the /shipment API to track the status of the shipment.
2. The shipment status is updated in real-time by the third-party API and displayed on the user's dashboard.

3. API Endpoints

Endpoint	Method	Purpose	Reason Example
/products	GET	Fetch all available product details	{ "id": 1, "name": "Product A", "price": 500, "stock": 20, "image": "box.jpg" }
/orders	POST	Place a new order	{ "orderId": 12345, "status": "Order Placed", "estimatedDelivery": "2025-01-18" }
/shipment	GET	Track the status of a shipment	{ "shipmentId": "SHIP123", "status": "In Transit", "expectedDelivery": "2025-01-20" }
/express-delivery-status	GET	Fetch real-time updates for express orders	{ "orderId": 56789, "status": "Out for Delivery", "ETA": "30 minutes" }
/customer	POST	Register or update customer details	{ "customerId": 456, "status": "Customer Registered" }

4. Sanity Schema

1. Product Schema

```
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' },
      { name: 'description', type: 'string', title: 'Description' },
      { name: 'image', type: 'image', title: 'Product Image' }
    ]
  };
```

2. Customer Schema

```
export default {
    name: 'customer',
    type: 'document',
    fields: [
      { name: 'name', type: 'string', title: 'Customer Name' },
      { name: 'email', type: 'string', title: 'Email Address' },
      { name: 'address', type: 'string', title: 'Shipping Address' },
      { name: 'phone', type: 'string', title: 'Phone Number' },
      { name: 'wishlist', type: 'array', title: 'Wishlist', of: [{ type: 'reference', to: [{ type: 'product' }] }] },
      { name: 'orderHistory', type: 'array', title: 'Order History', of: [{ type: 'reference', to: [{ type: 'order' }] }
    ]
  };
```

3. Order Schema

```
export default {
    name: 'order',
    type: 'document',
    fields: [
      { name: 'customerId', type: 'reference', to: [{ type: 'customer' }], title: 'Customer' },
      { name: 'products', type: 'array', title: 'Ordered Products', of: [{ type: 'reference', to: [{ type: 'product' }] }]
      { name: 'paymentStatus', type: 'string', title: 'Payment Status' },
      { name: 'totalAmount', type: 'number', title: 'Total Amount' },
      { name: 'deliveryMethod', type: 'string', title: 'Delivery Method' }
    ]
  };
```

5. Design System Architecture

System Components Overview

- **Frontend (Next.js)**: A dynamic, interactive interface that allows users to browse products, manage the shopping cart, and complete orders.
- **Sanity CMS**: Serves as the content management system to manage product, customer, and order data.
- **Third-Party APIs**: Provide external services like shipment tracking, real-time currency conversion, and notifications.
- **Payment Gateway**: Securely handles transactions and manages payment confirmation.

6. Data Flow and Workflows

User Registration Workflow

1. User registers or logs in on the frontend.
2. User data is sent to Sanity CMS, stored securely.
3. A confirmation email or message is sent to the user via a third-party API.

Product Browsing Workflow

1. User browses categories and products on the homepage.
2. The frontend sends a request to the /products API.
3. The Sanity CMS API responds with product details, which are displayed dynamically on the website.

Order Placement Workflow

1. User adds products to their cart and proceeds to checkout.
2. Order details (product list, shipping information, customer details) are sent to Sanity CMS via the /orders API.
3. Payment is processed through a payment gateway, and the order confirmation is displayed to the user.

Shipment Tracking Workflow

1. After the order is shipped, the frontend queries the /shipment API.
2. The shipment status is updated in real-time by the third-party API and displayed to the user on the "My Orders" page.

Payment Processing Workflow

1. User provides payment details during checkout.
2. Payment is securely processed through the payment gateway (e.g., Stripe, PayPal).
3. Confirmation of payment is sent to Sanity CMS and a confirmation message is displayed to the user.