# Marketplace Technical Foundation- Layers

## Contents

# Marketplace Technical Foundation- Layers

## 1. Introduction

**1.1 Project Overview** Layers is an online marketplace that offers a diverse range of products for users looking for an easy and reliable shopping experience. The platform is designed to handle transactions smoothly, providing a convenient interface for browsing and purchasing. This document lays out the technical foundation for Layers, ensuring that the system is well-structured and practical for implementation.

**1.2 Purpose of the Document** This document provides an overview of the system architecture, workflows, API setup, and data organization for Layers. It serves as a reference to ensure proper development and maintainability of the marketplace.

---

## 2. System Architecture

### 2.1 Overview

- Frontend**:** Developed using Next.js and Tailwind CSS to create a responsive and accessible interface.
- Authentication**:** User login and registration handled by Clerk for security.
- Backend**:** Sanity CMS is used to manage and store product, customer, and order data.
- APIs**:** External services like Stripe for payments and ShipEngine for shipping calculations.

### 2.2 Workflow Example

1. Users create an account or log in.

2. They browse through product listings on the website.
3. The frontend retrieves product details from Sanity CMS.
4. When a purchase is made, the order is stored in Sanity CMS.
5. Payment and shipping are processed using third-party APIs.

---

## 3. Technical Requirements

### 3.1 Frontend Needs

- Intuitive interface with the following pages:
  - Homepage
  - Product Categories
  - Product Details
  - Shopping Cart
  - Order Tracking
- Optimized for both mobile and desktop users.

### 3.2 Backend Setup

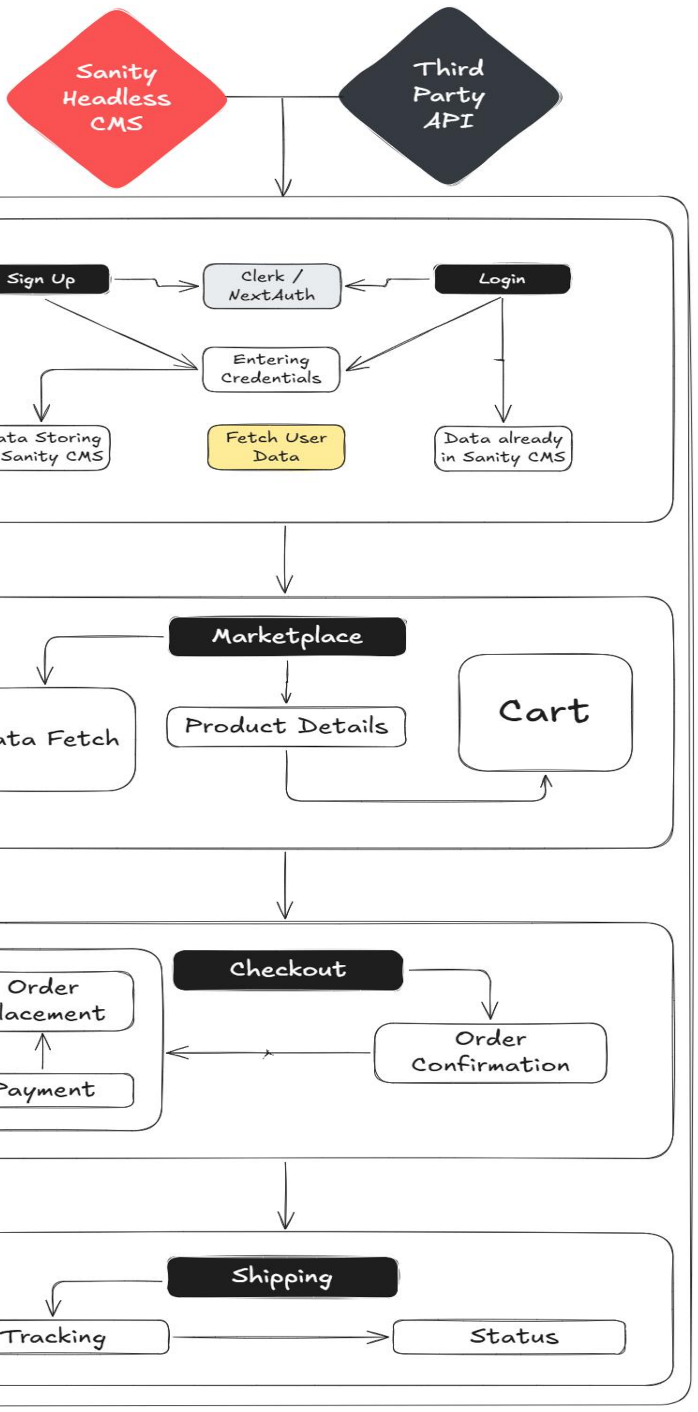
- Sanity CMS used for:
  - Storing product and inventory details
  - Managing customer data
  - Handling orders and transactions

### 3.3 API Integrations

- **Payment Processing:** Stripe for handling secure transactions.

- **Shipping Services:** ShipEngine for calculating rates and tracking shipments.

---



## 4. API Setup

| Endpoint | Method | Purpose | Response Example |
|---|---|---|---|
| /products | GET/POST | Retrieve or update product details | { "id": "123", "name": "Product A", "price": 100 } |
| /orders | POST/GET | Create or view orders | { "orderId": "abc123", "status": "Processing" } |
| /shipment | GET | Get shipping details | { "orderId": "123", "status": "Shipped" } |

---

## 5. Workflows

### 5.1 User Registration

1. A user signs up via Clerk authentication.
2. Their details are stored in Sanity CMS.
3. A confirmation email is sent.

### 5.2 Product Browsing

1. Users navigate through product categories.
2. Product data is pulled from Sanity CMS.
3. The frontend dynamically displays the products.

### 5.3 Order Processing

1. Users add products to their cart and proceed to checkout.
2. Order details are saved in Sanity CMS.
3. Payment is processed via Stripe, and the user receives a confirmation.

### 5.4 Shipping Updates

1. The system fetches shipment status from ShipEngine.
2. The latest delivery status is displayed to the customer.

---

## 6. Data Model

### 6.1 Sanity CMS Structure

#### Product Schema

```
export const product = defineType({
  name: "product",
  type: "document",
  title: "Product",
  fields: [
    defineField({
      name: "name",
      type: "string",
      title: "Product Name",
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "description",
      type: "text",
      title: "Product Description",
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "images",
      type: "array",
      title: "Product Images",
      of: [
        {
          type: "image",
          options: {
            hotspot: true,
          },
        },
      ],
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "price",
      title: "Price",
      type: "number",
      validation: (Rule) => Rule.required(),
    }),
    },

    defineField({
      name: "discountPercent",
      title: "Discount Percent",
      type: "number",
      validation: (Rule) => Rule.required(),
    }),

    defineField({
      name: "category",
      title: "Category",
      type: "string",
      options: {
        list: [
          { title: "Casual Wear", value: "casual-wear" },
          { title: "Western Wear", value: "western-wear" },
          { title: "Sports Wear", value: "sports-wear" },
          { title: "Festive Wear", value: "festive-wear" },
          { title: "Kids Wear", value: "kids-wear" },
          { title: "Formal Wear", value: "formal-wear" },
        ],
      },
    }),
    defineField({
      name: "subcategory",
      type: "string",
      title: "Subcategory",
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "sizes",
      title: "Available Sizes",
      type: "array",
      of: [{ type: "string" }],
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "colors",
      title: "Available Colors",
      type: "array",
      of: [{ type: "string" }],
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "stock",
      title: "Product Stock",
      type: "number",
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "slug",
      type: "slug",
      title: "Slug",
      options: {
        source: "name",
        maxLength: 96,
        slugify: (input) => {
          const slugSuffix = uuidv4();
          const baseSlug = input
            .toLowerCase()
            .replace(/\s+/g, "-")
            .replace(/[^\w\-]+/g, "");
          return `${baseSlug}-${slugSuffix}`.slice(0, 96);
        },
      },
      validation: (Rule) =>
        Rule.required().custom(async (slug, context) => {
        if (!slug?.current) {
          return "Slug is required and must be defined.";
        }

        // Get current document ID (if exists)
        const { document } = context;
        const id = document?._id?.replace(/^drafts\./, ""); // Remove
draft prefix

        // Query for existing slugs excluding current document
        const existing = await client.fetch(
          `*[_type == "product" && slug.current == $slug && _id != $id]`,
          { slug: slug.current, id: id || "" }
        );

        return existing.length === 0 || "Slug must be unique.";
      }),
    }),

    defineField({
      name: "reviews",
      title: "Reviews",
type: "array",
of: [
  defineField({
    name: "review",
    title: "Review",
    type: "object",
    fields: [
      defineField({
        name: "userName",
        title: "User Name",
        type: "string",
        validation: (Rule) => Rule.required(),
      }),
      defineField({
        name: "comment",
        title: "Comment",
        type: "text",
        validation: (Rule) => Rule.required(),
      }),
      defineField({
        name: "rating",
        title: "Rating",
              type: "number",
              validation: (Rule) =>
                Rule.required()
                  .min(1)
                  .max(5)
                  .error("Rating must be between 1 and 5."),
            }),
            defineField({
              name: "date",
              title: "Review Date",
              type: "datetime",
              validation: (Rule) => Rule.required(),
            }),
          ],
        }),
      ],
    }),
  ],
});
```

## Order Schema

```
export default {
  name: 'order',
  type: 'document',
  fields: [
    { name: 'orderId', type: 'string', title: 'Order ID' },
    { name: 'customerId', type: 'string', title: 'Customer ID' },
    { name: 'products', type: 'array', of: [{ type: 'object', to: [{ type:
'product' }] }], title: 'Products' },
    { name: 'status', type: 'string', title: 'Order Status' },
    { name: 'createdAt', type: 'datetime', title: 'Order Created At' }
  ]
};
```

### Customer Schema

```
export default {
  name: 'customer',
  type: 'document',
  fields: [
    { name: 'customerId', type: 'string', title: 'Customer ID' },
    { name: 'name', type: 'string', title: 'Customer Name' },
    { name: 'email', type: 'string', title: 'Email' },
    { name: 'contact', type: 'string', title: 'Contact' },
    { name: 'address', type: 'string', title: 'Address' }
  ]
};
```

---

## 7. Development Roadmap

### Phase 1: UI & Frontend Development

- Build user interface with Next.js and Tailwind CSS.
- Ensure mobile-friendly design.

### Phase 2: Backend Configuration

- Set up Sanity CMS for handling product and order data.
- Develop API endpoints.

### Phase 3: API Integration

- Implement Stripe for transactions.
- Set up ShipEngine for delivery tracking.

### Phase 4: Testing & Deployment

- Conduct comprehensive testing.
- Deploy the application with scalable hosting.

---