Author

Tabinda Atif

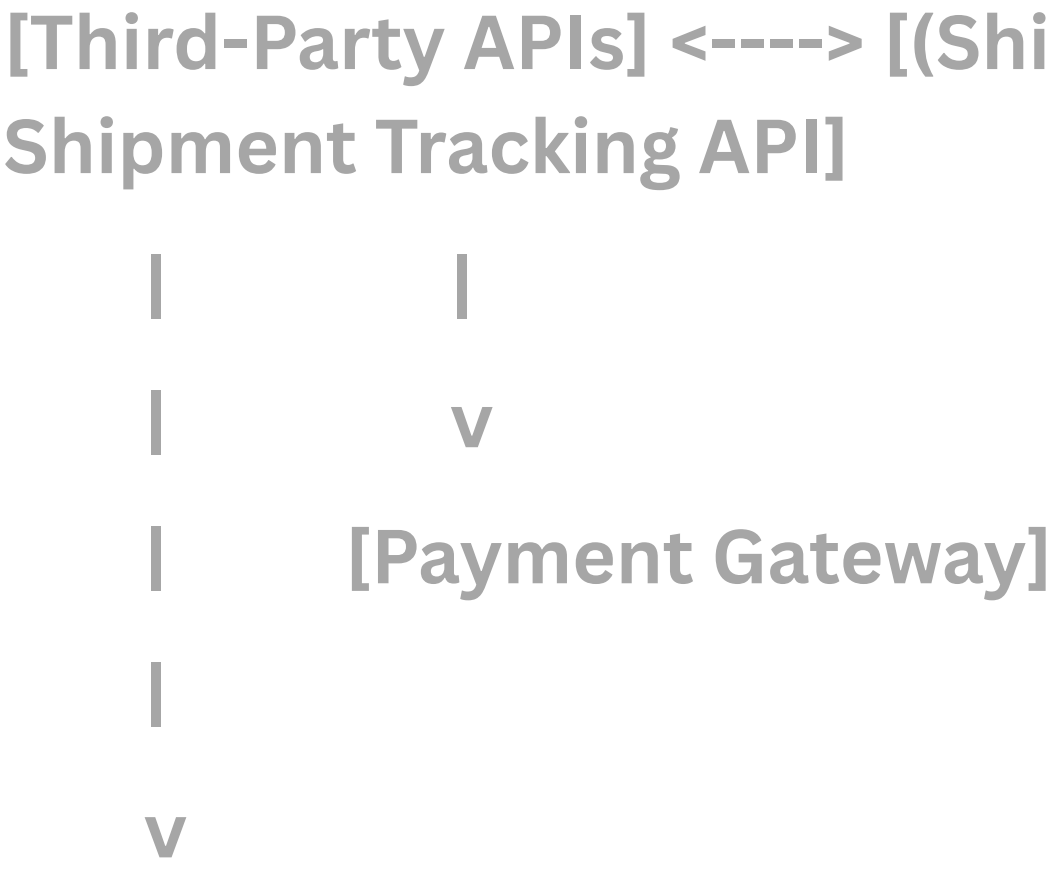
Lead Developer, Bandage Online Shopping

Bandage Online Shopping:

Foundation and Enhanced Workflow

System Architecture Overview

High-Level Diagram



Component Descriptions

Frontend (Next.js):

Provides a responsive and interactive user interface for browsing products, managing orders, and

handling user authentication.

Fetches and displays data from the backend APIs in real-time.

Sanity CMS:

Third-Party APIs:

Authentication (Clerk):

Key Workflows

User Registration

Process:

Product Browsing

Process:

Order Placement

Process:

Shipment Tracking

Process:

Centralized backend for managing product information, user data, order records, and inventory.

Exposes APIs for dynamic data communication with the frontend.

1. Shipment Tracking API (Ship Engine): Fetches real-time shipping updates and generates tracking details.

2. Payment Gateway Processes secure transactions and confirms payment status.

Handles user registration, login, and session management.

Integrates with Sanity CMS to store user data securely.

User signs up via the frontend using Clerk.

Registration details are stored in Sanity CMS.

User navigates through product categories on the frontend.

Sanity CMS API fetches product data (name, price, stock, description, images).

Dynamic product listings are displayed on the frontend.

User adds products to the cart and proceeds

Order details (products, quantities, shipping address) are sent to Sanity CMS.

Payment is processed

A confirmation message is sent to the user’s email, and the order is recorded in Sanity CMS.

After order placement, shipment details are updated using ShipEngine.

2.5 Inventory Management

Process:

Category-Specific Instructions

3 General eCommerce

Focus on:

Workflow Example:

Real-time tracking information is displayed to the user on the frontend.

Product stock levels are managed in Sanity CMS.

Real-time stock updates are fetched from Sanity CMS.

Out-of-stock products are added to wishlist instead of cart.

In-stock products can be added to cart and proceed to checkout.

Standard product browsing, cart management, wishlist, and order placement.

Inventory management and real-time stock updates.

1. Endpoint: /products

2. Method: GET

3. Purpose: Fetch all product listings.

Response Example:

JSON:


```
[
 {
   “name”: “Product Name”,
   “slug”: “product-slug”,
   “image”: “https://productimage.png”,
   “description”: “Product Description”,
   “price”: 58$,
   “discountPrice”: 23$,
```

```
    “inStock”: true,
        “rating”: 4,
    “reviews”: 18
  }
]
```

Sanity Schema Example

Product Schema:

```
import { defineField, defineType } from “sanity”;
export const product = defineType({
  name: “product”,
  title: “Products”,
  type: “document”,
  fields: [
    defineField({
      name: “name”,
      title: “Product Name”,
  type: “string”,
      validation: (Rule) => Rule.required(),
    }),

    defineField({
      name: “image”,
      title: “Product Image”,
      type: “image”,
```

```
    options: {
      hotspot: true,
    },
    validation: (Rule) => Rule.required(),
  }),
  defineField({
    name: “additionalImages”,
    title: “Additional Images”,
    type: “array”,
    of: [{ type: “image”, options: { hotspot: true } }],
  }),
  defineField({
    name: “description”,
itle: “Description”,
    type: “blockContent”,
  }),
  defineField({
    name: “price”,
    title: “Price”,
    type: “number”,
    validation: (Rule) => Rule.required().min(0),
  }),
  defineField({
    name: “discountPrice”,
    title: “Discount Price”,
```

```
    type: “number”,
    description: “Discounted price of the product (optional)”,
    validation: (Rule) =>
     Rule.custom((discountPrice, context) => {
      const doc = context.document;
      if (doc && typeof doc.price === “number”) {
       if (discountPrice && discountPrice >= doc.price) {
        return “Discount price must be less than the original price”;
       }
      }
      return true;
     }),
  }),
  define Field({
    name: “in Stock”,
    title: “In Stock”,
    type: “boolean”,
    description: “Indicates whether the product is currently in stock”,
itialValue: true,
    validation: (Rule) => Rule.required(),
  }),
  define Field({
    name: “stock”,
    title: “Stock Quantity”,
    type: “number”,
```

```
    description: “Number of items available in stock”,
    validation: (Rule) => Rule. required().min(0),
  }),
  define Field({
    name: “rating”,
    title: “Rating”,
    type: “number”,
    description: “Rating from 0 to 5”,
    validation: (Rule) => Rule.min(0).max(5).precision(1),
  }),
  define Field({
    name: “reviews”,
    title: “Reviews Count”,
    type: “number”,
    description: “Number of reviews for the product”,
    validation: (Rule) => Rule. required().min(0),
  }),
],
preview: {
  select: {
    title: “name”,
    media: “image”,
    subtitle: “price”,     inStock: “inStock”,
    stock: “stock”,
  },
```

)};

Technical Roadmap Development Phase:

Authentication:

Product Management:

Payment Integration:

Implement user registration and login using Clerk.

Integrate Clerk with Sanity CMS for user data storage.

Provide Api product data.

Store product data in Sanity CMS.

Fetch and display product data on dynamic frontend pages.

Cart and Wishlist:

Implement add-to-cart functionality with real-time stock checks.

Allow out-of-stock products to be added to wishlist.

Display total bill and proceed to checkout button on cart page.

Getaway payments.

Shipment Tracking:

Inventory Management:

Testing Phase

Security Audits:

Conduct security audits for sensitive data handling, including user authentication and payment processing.

Launch Phase:

Deployment:

Post-Launch:

Conclusion:

Integrate ShipEngine for shipment tracking.

Generate tracking numbers and display on the frontend.

Allow users to track their orders in real-time.

Create API for real-time stock updates in Sanity CMS.

Update stock levels upon order placement.

Prevent out-of-stock products from being added to cart.

Test all workflows, including user registration, product browsing, cart management, checkout, and shipment tracking.

Validate API responses and ensure data accuracy.

Deploy the platform on a cloud hosting service (e.g., Vercel).

Monitor user feedback and optimize for performance.

Collect user feedback for continuous improvement.

Optimize API performance and frontend loading times.

Scale infrastructure based on traffic and demand.