Marketplace Technical Foundation - [Bandage Online Shopping Platform]

Author: NAZIA SHOUKAT

Role: Lead Developer, Bandage Online Shopping

DAY 1 BUSINESS FOCUS:

Primary Purpose: Designed a flexible platform to offer various products with ease, good prices, and reliable delivery to meet modern consumer needs.

Problem Solved: Made shopping easier and quicker with smooth navigation, fast order processing, and user-friendly design, reducing delivery delays.

Target Audience: Aimed at busy shoppers, families looking for convenience, and professionals who want reliable delivery options.

Products and Services:

1. Categories: fashion

2. Extra value: Subscription deliveries, special discounts, and easy returns.

E-Commerce Data Schema:

1. Main Entities: Products, Customers, Orders, Payments, Shipments, and Delivery Areas.
2. Key Connections: Models for real-time tracking, customer order history, and flexible delivery charges.

Marketplace Features:

1. Dynamic product filters.
2. Real-time order tracking and various payment options.

Day 2 Activities: Transitioning to Technical Planning:

This document describes the **technical framework** and **improved processes** for the **Bandage Online Shopping Platform**. It covers system structure,

essential workflows, API interfaces, and a technical plan.

System Architecture

High-Level Diagram

[Frontend (Next.js)]

|

v

[Sanity CMS] <--------> [Product Data (Mock) API]

| ^

| |

[Third-Party APIs] <----> [(ShipEngine) Shipment Tracking API]

| |

| v

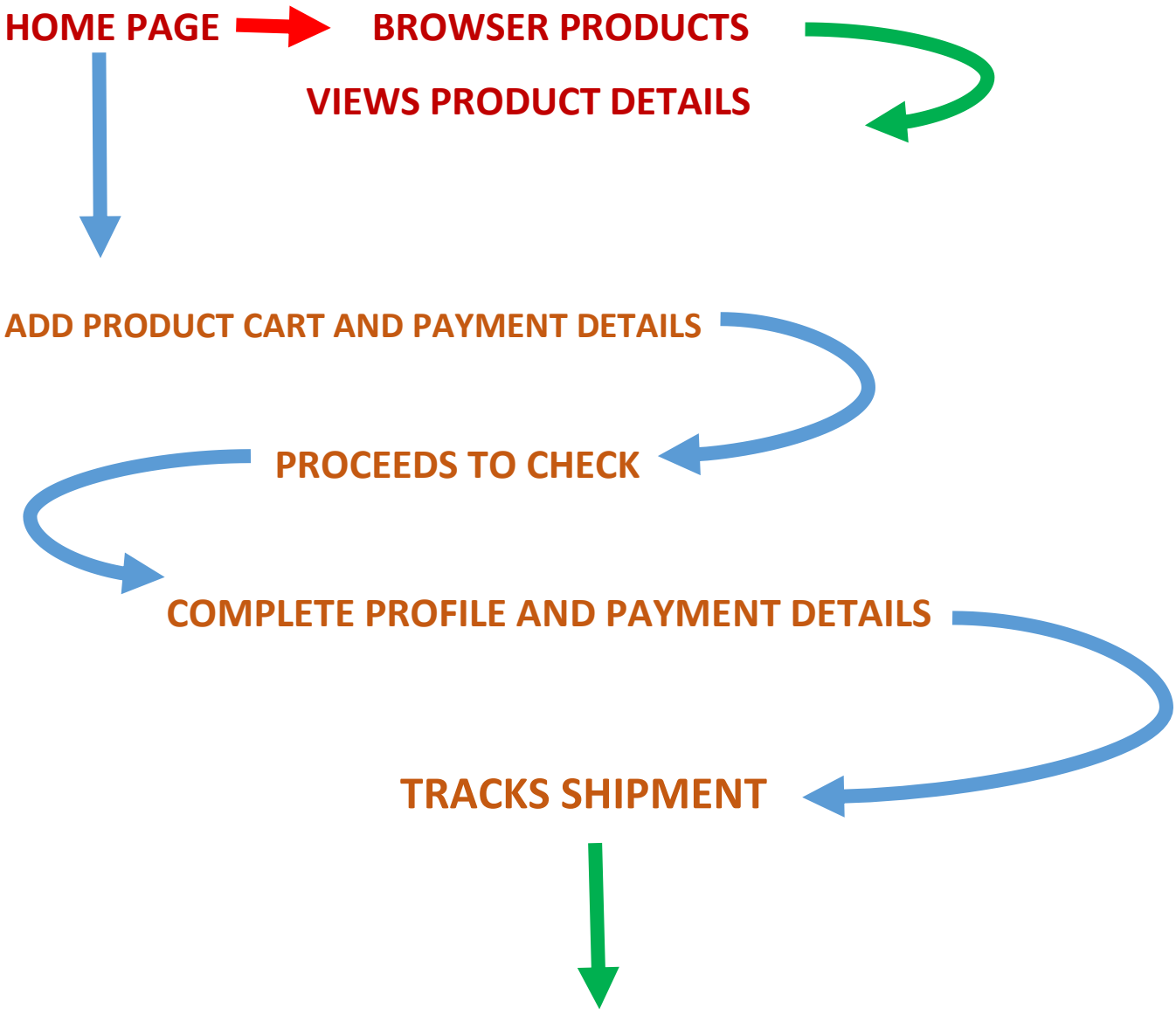
| [Payment Gateway (Stripe)]

|

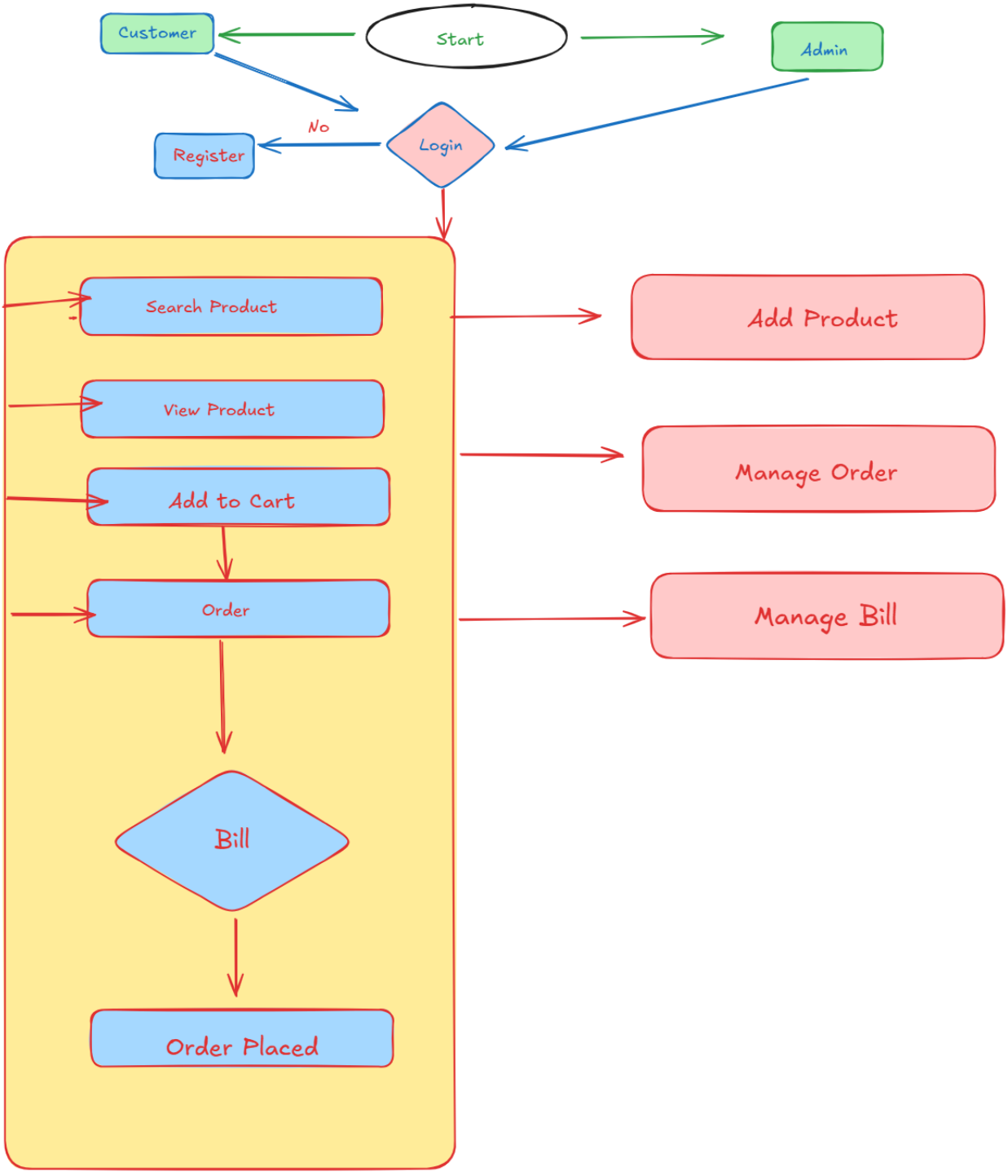
v

[Authentication (Clerk)]

WORKFLOW DIAGRAM VISUAL REPRESENTATION



RECEIVE DELIVERY AT HOME



Component Descriptions

Frontend (Next.js):

- Offer a seamless and engaging user interface for exploring products, tracking orders, and managing user accounts.
- Retrieves and updates data from backend APIs instantly, ensuring real-time display and interactivity.
- By enhancing the user experience with real-time data fetching and interactive elements, the platform guarantees smooth navigation and efficient order management. Whether browsing through product listings or handling authentication processes, users enjoy a fluid and responsive interface tailored to meet their needs.

Sanity CMS:

• Serves as a unified backend system, effectively handling product details, customer information, order histories, and stock levels.

- Provides powerful APIs that enable easy and quick data exchange with the frontend.

- By storing all important data in one place, Sanity CMS makes it simple to manage product information, user details, order records, and inventory. It helps to keep everything updated and reduces errors. The APIs allow for real-time data updates, so the frontend always shows the most current information. This improves user experience by providing accurate and up-to-date data on products, user profiles, and orders, ensuring smooth and responsive interactions on the platform.

Third-Party APIs:

1. ShipEngine (Shipment Tracking API):

- Provides real-time updates on shipping status
- Creates tracking information for packages

Enables users to track packages from dispatch to delivery

- Improves transparency and customer satisfaction

2. Stripe (Payment Gateway):

- Handles secure and reliable financial transactions
- Ensures safety through encryption and fraud detection
- Facilitates seamless payment processing
- Provides real-time verification of payment status

- Supports multiple payment methods and currencies

Authentication (Clerk):

- Manages users create accounts and log in
- Keeps user information safe with Sanity CMS
- Ensures the login process is secure and works well
- Protects user data and keeps it private

Key Workflows

1. User Registration

Process:

- User signs up on the frontend using Clerk.
- Registration details are saved in Sanity CMS.

2.Browsing Products:

Process:

- User explores product categories on the frontend.

- Sanity CMS API retrieves product information (name, price, availability, description, images).
- The frontend displays updated product listings dynamically.

3. Order Placement:

Process:

- User adds products to the cart and goes to checkout.
- Order details (products, quantities, shipping address) are sent to Sanity CMS.
- Payment is handled through Stripe.
- A confirmation message is sent to the user's email, and the order is recorded in Sanity CMS.

4. Shipment Tracking:

Process:

- After order placement, shipment details are updated using ShipEngine.
- Real-time tracking information is displayed to the user on the frontend.

5. Inventory Management:

Process:

- **Product stock levels are managed in Sanity CMS.**
- **Real-time stock updates are fetched from Sanity CMS.**
- **Out-of-stock products are added to the wishlist instead of the cart.**
- **In-stock products can be added to the cart and proceed to checkout.**

API Endpoints

Endpoint	Methods	Purpose	Response Example
/products	GET	Fetch all product details	[{ "name": "Product Name", "slug": "product-slug", "price": 100 }]
/order	POST	Submit new order details	{ "orderId": 123, "status": "success" }
/shipment-tracking	GET	Fetch real-time tracking updates	{ "trackingId": "AB123", "status": "In Transit" }
/delivery-status	GET	Fetch express delivery information	{ "orderId": 456, "deliveryTime": "30 mins" }
/inventory	GET	Fetch real-time stock levels	{ "productId": 789, "stock": 50 }
/cart	POST	Add product to cart	{ "cartId": 101, "items": [...] }
/wishlist	POST	Add product to wishlist	{ "wishlistId": 202, "items": [...] }

Sanity Schema Example:

```
import { TrolleyIcon } from
"@sanity/icons";
import { defineField, defineType }
from "sanity";

export const productTypes =
defineType({
  name: "product",
  title: "Products",
  type: "document",
  icon: TrolleyIcon,
  fields: [
    defineField({
      name: "name",
      title: "Product Name",
      type: "string",
      validation: (Rule) =>
Rule.required(),
    }),
```

```
    defineField({
      name: "slug",
      title: "Slug",
      type: "slug",
      options: {
        source: "name",
        maxLength: 96,
      },
      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",
      title: "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;
        }),
    }),
```

```
    defineField({
      name: "inStock",
      title: "In Stock",
      type: "boolean",
      description: "Indicates whether
the product is currently in stock",
      initialValue: true,
      validation: (Rule) =>
Rule.required(),
    }),
    defineField({
      name: "stock",
      title: "Stock Quantity",
      type: "number",
      description: "Number of items
available in stock",
      validation: (Rule) =>
Rule.required().min(0),
    }),
    defineField({
      name: "rating",
```

```
      title: "Rating",
      type: "number",
      description: "Rating from 0 to
5",
      validation: (Rule) =>
Rule.min(0).max(5).precision(1),
    }),
    defineField({
      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",
    },
    prepare({ title, subtitle, media,
inStock, stock }) {
      return {
        title,
        subtitle: `$${subtitle} |
${inStock ? `In Stock (${stock})` :
"Out of Stock"}`,
        media,
      };
    },
  },
});
```

Technical Roadmap

This document describes the technical plan for the **Bandage Online Shopping Platform**. It includes the stages of development, testing, and launching, as well as important features and workflows.

Development Phase

Authentication

- Set up user sign-up and login with **Clerk**.
- Connect Clerk to **Sanity CMS** to store user information.

Product Management

- Build a sample API for product information.
- Save product data in **Sanity CMS**.
- Fetch and display product data on dynamic frontend pages.

Cart and Wishlist:

- Add the feature to put items in the cart and check stock in real-time.
- Let users add out-of-stock items to their wishlist.
- Show the total bill and a "Proceed to Checkout" button on the cart page.

Payment Integration

- Integrate **Stripe** for secure payments.

- Use a Stripe test account while developing.
- Manage both successful and failed payment cases.

Shipment Tracking

- Integrate **ShipEngine** for tracking shipments.
- Create tracking numbers and show them on the frontend.
- Allow users track their orders live

Inventory Management

- Develop an API for updating stock levels in **Sanity CMS** in real-time.
- Adjust stock quantities when an order is placed.
- Prevent out-of-stock products from being added to the cart.

Testing Phase

End-to-End Testing

- **Test all workflows, including:**
- **User registration.**
- **Product browsing.**
- **Cart management.**
- **Checkout process.**
- **Shipment tracking.**
- **Validate API responses and ensure data accuracy.**

Security Checks

Perform security checks for handling important data, such as:

- User logins.
- Payment transactions.

Launch Phase

Deployment

- Deploy the platform on a cloud hosting service (e.g., **Vercel**, **Netlify**).
- Monitor user feedback and optimize for performance.

Post-Launch

- Collect user feedback to keep improving.

- Enhance API performance and speed up the frontend loading times.

Conclusion:

This technical plan describes the structure, workflows, and API endpoints for the **Bandage Online Shopping Platform**. The platform aims to offer a smooth eCommerce experience, including:

- Strong user authentication.
- Effective inventory management.
- Live shipment tracking.