Marketplace Technical Foundation - [FOOD RESTAURANT]

---

1. Technical Plan

Overview: This proposal outlines the technical foundation for [Your Marketplace Name], a marketplace platform designed to enable users to browse products, place orders, and track shipments in real-time. The marketplace will integrate essential features such as user registration, product browsing, payment processing, and shipment tracking.

Tech Stack:

- **Frontend:** Next.js (React-based framework for SSR and SEO optimization)
- **CMS:** Sanity CMS (for managing product data, content flexibility)
- **Payments:** Stripe (secure payment processing)
- **Shipping:** ShipEngine API (real-time shipment tracking)
- **Database:** Sanity (for structured content management and products)

Development Milestones:

- **Week 1:** Set up the environment, implement the frontend UI (Next.js), and integrate Sanity CMS.
- **Week 2:** Implement API endpoints for product management and orders.
- **Week 3:** Integrate Stripe for payments and ShipEngine API for shipment tracking.
- **Week 4:** Testing, bug fixes, and finalizing the admin dashboard.

---

2. Workflows

Key Workflows:

1. **User Registration Workflow:**
   - User accesses the marketplace and clicks "Sign Up."
   - User provides their email, password, and other necessary information.
   - The system sends an email verification link.
   - User verifies the email and logs in.
2. **Product Browsing Workflow:**
   - User navigates to the "Products" section.
   - User searches, filters by category, and views product details.
   - User adds products to their cart or wishlist.
3. **Order Placement Workflow:**
   - User proceeds to checkout with the cart items.
   - User enters shipping details and selects a payment method.
   - User processes the payment via Stripe API.
   - Order details are saved in the system and shipment tracking is initialized via ShipEngine.
4. **Admin Dashboard Workflow:**
   - Admin accesses the dashboard to manage products, orders, and users.
   - Admin can update product details, view order status, and manage user queries.

Component Roles:

- **Frontend (Next.js)**: Handles UI/UX, interaction with APIs.
- **Sanity CMS**: Stores and manages product data, user profiles.

- **Stripe**: Processes user payments securely.
- **ShipEngine**: Provides shipment tracking and updates.

---

3. API Requirements

Endpoint	**Method**	**Payload**	**Response**
/api/products	GET	None	List of products with IDs, names, prices, etc.
/api/product	POST	{name, description, price, stock}	Product created/updated with success message
/api/orders	POST	{user_id, product_ids, shipping}	Order confirmation, order ID, total cost
/api/orders/{id}	GET	None	Order details (status, products, shipping)
/api/payment	POST	{order_id, payment_details}	Payment confirmation with success or failure
/api/shipment	GET	{order_id}	Shipment status (tracked via ShipEngine API)

```
{  "user_id": "1234",
  "product_ids": [101, 102],
  "shipping": "123 Main St"
}
{
  "status": "success",
  "order_id": "5678",
  "total_cost": 199.99
}
```

---

4. Sanity Schema

Products Schema:

```
{
  "name": "string",
  "description": "text",
  "price": "number",
  "stock": "number",
  "category": "string",
  "images": ["url"]
}
```

Orders Schema:

```
{
  "user_id": "reference to User",
  "products": ["reference to Product"],
  "shipping_address": "text",
  "payment_status": "string",
  "order_date": "date"
}
```

User Schema:

```
{
  "name": "string",
  "email": "string",
  "password": "string",
  "address": "text",
  "order_history": ["reference to Order"]
}
```

Relationships:

- A **User** has many **Orders**.
- An **Order** belongs to a **User** and contains many **Products**.
- **Products** are categorized and have multiple **images**.

---

5. Collaboration Notes

- **Peer Collaboration:** Worked closely with the frontend and design teams to ensure the UI/UX is aligned with the backend functionalities. Regular discussions on product listing structures and payment flow.
- **Challenges Faced:** Encountered challenges integrating the ShipEngine API to provide real-time shipment updates. Managed to resolve by thoroughly reviewing API documentation and testing with mock data.
- **Feedback:** Peer feedback emphasized improving the product search experience by implementing advanced filtering. Also, the feedback on payment flow led to optimizing Stripe integration for a smoother user experience.

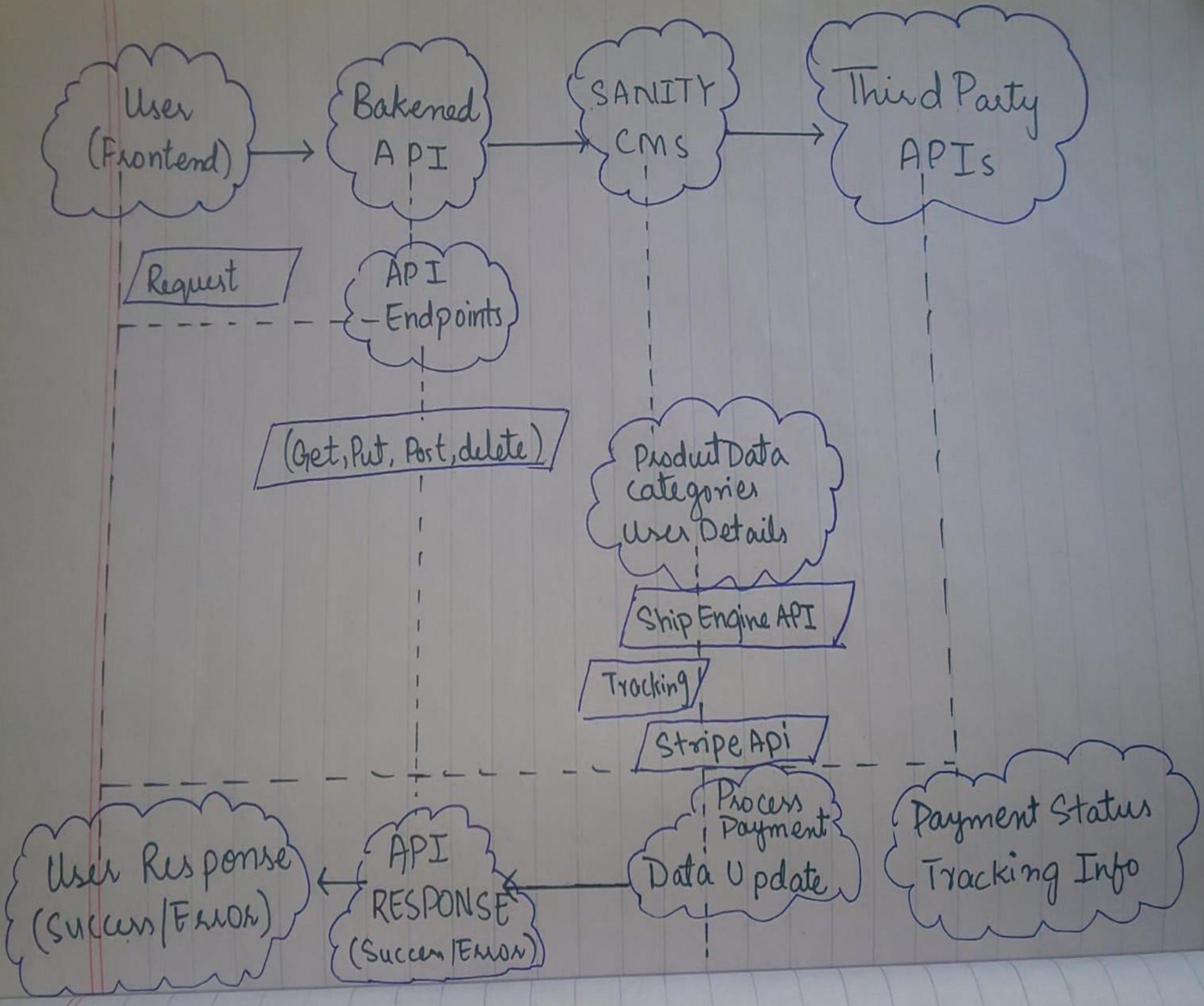
System Architecture Overview

The system will include the following components:

- **Frontend (Next.js):** The user interface for browsing products, managing the cart, placing orders, and managing accounts.
- **Sanity CMS:** A headless CMS for storing product data, categories, user details, and orders.
- **Backend API:** Manages the interaction between the frontend and CMS, handles user registration, order placement, and payment processing.
- **Third-Party APIs:** Stripe for payment handling and ShipEngine for shipment tracking.

The architecture diagram would look like this:

1. **Frontend (Next.js)** interacts with the **Backend API**.
2. **Backend API** communicates with **Sanity CMS** for fetching and storing product and order data.
3. **Stripe API** is used for payment processing, and **ShipEngine API** for real-time shipment tracking.


User (Frontend)
Bakened API
SANITY CMS
Third Party APIs
Request
API Endpoints
(Get, Put, Post, delete)
Product Data categories User Details
Ship Engine API
Tracking
Stripe Api
Process Payment
Data Update
Payment Status Tracking Info
API RESPONSE (Succes/Error)
User Response (Succes/Error)