"Marketplace Technical Foundation - [MArtNow]"

Transitioning to Technical Planning

Frontend Requirement

1. User-friendly Interface for Browsing Products

- Clear navigation:.
- Visual appeal
- Quick interaction

2. Responsive Design for Mobile and Desktop Users

- Adaptive Layouts
- Flexible grids
- Touch-friendly elements
- Optimized images

3. Essential Pages

- Home:
- Product Listing:
- Product Details:
- Cart:
- Checkout:
- Order Confirmation:

Using Sanity CMS as the Backend for Marketplace:

1. Managing Product Data

- **Schemas**: Define a custom schema for product data that includes essential fields like:
    - **Product Name**
    - **Product Description**
    - **Price**
    - **Product Images** (allow multiple images for each product)
    - **Category** (e.g., clothing, electronics, accessories)
    - **Stock Quantity** (manage inventory levels)
    - **Brand or Manufacturer**
    - **Size/Color Options** (if applicable)
    - **Discounts or Promotions** (if applicable)
    - **Ratings/Reviews** (optional, if your marketplace includes customer feedback)

2. Managing Customer Details

- **First Name** and **Last Name**
- **Email Address**
- **Shipping Address**
- **Order History** (you can reference order records here)
- **Phone Number**

3. Managing Order Records

- **Schemas**: Define a custom schema for order records that captures the key details of each transaction:
  - **Order ID**
  - **Customer Reference** (link to the customer schema)
  - **List of Products** (link to the products schema, including quantity, price, and any customization)
  - **Order Status** (e.g., pending, shipped, delivered)
  - **Payment Information** (transaction ID, payment status)
  - **Shipping Address**
  - **Order Date**

4. Utilizing Sanity's API for Frontend Integration

- **Real-time Data Sync**: Use Sanity’s API to pull product, customer, and order data dynamically into your frontend. This will allow your website to show up-to-date product listings, cart details, and order statuses.
- **Headless Flexibility**: As Sanity is a headless CMS, you can easily integrate it with your frontend built using technologies like React, Vue, or Next.js. This allows you to build a highly responsive and interactive user experience while managing all your backend content through Sanity.

6. Security and Privacy Considerations

- **Data Protection**: For customer and order data, ensure that proper encryption is used when storing sensitive information, especially payment details. While Sanity itself doesn't handle payments, you should implement security measures when integrating with payment gateways.
- **Access Control**: Sanity allows you to manage who can access and modify data. Set up role-based access control (RBAC) for users managing product data, order records, and customer information to ensure data integrity and security.

Integrating Third-Party APIs for Your Marketplace

When building a marketplace, integrating third-party APIs can greatly enhance functionality by connecting to services such as shipment tracking, payment gateways, and other backend services. Here's how to approach integrating these APIs to meet both business and technical needs:

1. Shipment Tracking APIs

- **API Data Needs**:
    - **Tracking Number**: Each order will have a unique tracking number, which must be provided by your shipping provider.
    - **Carrier Information**: You need to capture which carrier is handling the shipment.
    - **Tracking Updates**: Ensure real-time updates on shipment statuses, such as "shipped", "in transit", "out for delivery", or "delivered".

Frontend Functionality:

- **Display Tracking Info**: Provide users with a tracking link or embed a real-time status update on the "Order Confirmation" page or in email notifications.
- **Tracking UI**: Design a simple interface where users can input their tracking number or click to view real-time status.

Integration Example:

- **API Call**: Use a `GET` request to the shipment API with the tracking number.
- **Response**: The API response will contain details about the order's shipping status and location.
- **Display**: Parse and display the tracking information (status, location, estimated delivery date) on the frontend.

---

2. Payment Gateway APIs

Integrating payment gateways is critical for processing customer transactions securely. Depending on your market and customer preferences, you can integrate with various payment providers.

- **API Data Needs**:
    - **Payment Intent/Session**: You'll need to create a payment session or intent through the API to initiate the payment process.
    - **Transaction Confirmation**: After payment, you'll need to confirm whether the transaction was successful.

- **Security**: Payment API integration requires secure communication (SSL) and tokenization to protect user data.

Frontend Functionality:

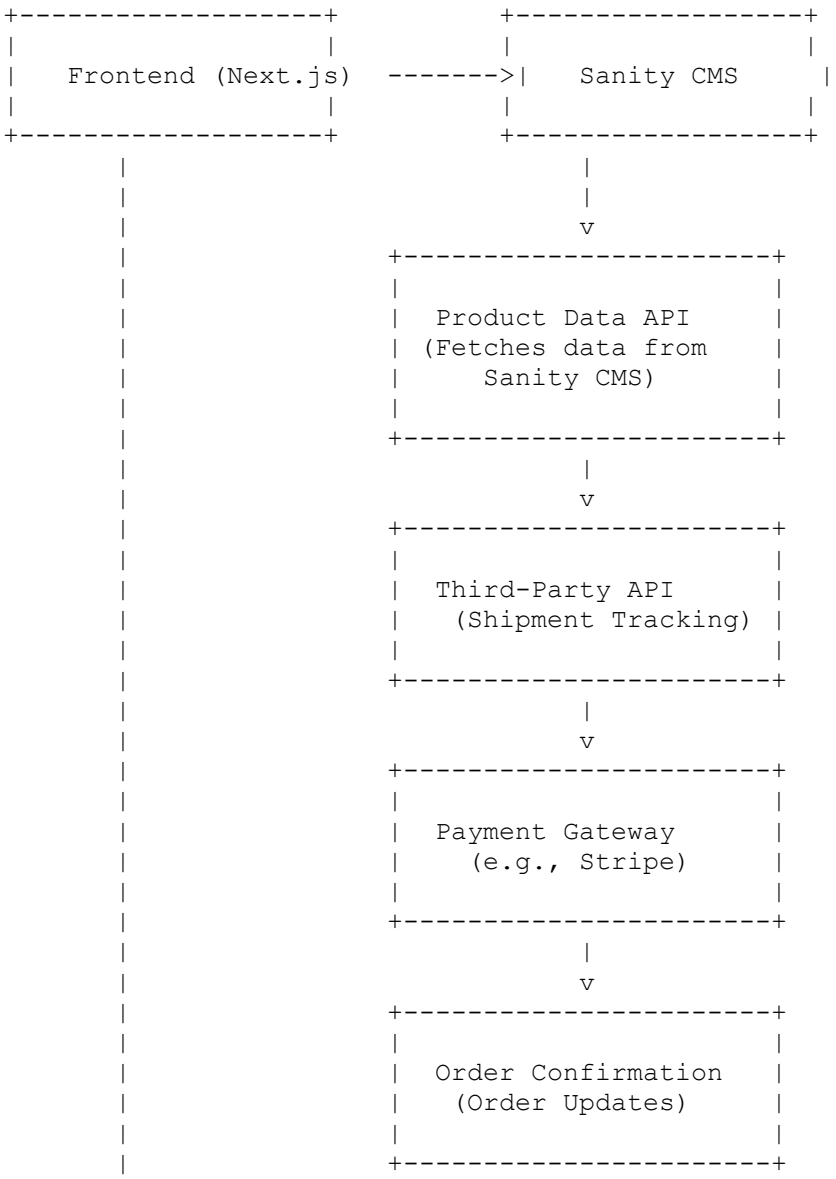
- **Checkout Process**: Integrate the payment gateway into your checkout page, allowing users to enter payment details.
- **Payment Confirmation**: After a successful payment, display a confirmation page with the order details and transaction receipt.
- **Error Handling**: Ensure that any payment failure or error is communicated clearly to the user, with suggestions for next steps.

Integration Example:

- **API Call**: Create a payment session using a `POST` request to the payment API. Include order total, customer info, and payment method details.
- **Response**: The response will contain a confirmation (success/failure) along with a transaction ID.
- **Frontend**: If successful, proceed with redirecting to the order confirmation page. If failed, prompt the user to try another payment method.

High-Level System Architecture Diagram

To create a high-level system architecture for your marketplace, let's break down the components and show how they interact with each other. Here's a simplified diagram to illustrate the flow of data and interactions between the various systems:



Breakdown of the Diagram:

1. **Frontend (Next.js)**
    - **Responsibilities**: Handles the user interface, rendering pages, and making API calls.
    - **Interactions**:
        - Requests product data from **Sanity CMS** through the **Product Data API**.
        - Communicates with **Third-Party APIs** (e.g., Shipment Tracking) to provide real-time tracking updates.
        - Interacts with the **Payment Gateway** for processing customer payments.
2. **Sanity CMS**
    - **Responsibilities**: Stores and manages the product, customer, and order data.
    - **Interactions**:
        - Receives and stores product data (including descriptions, prices, images, etc.).
        - Provides APIs (like **Product Data API**) for fetching the product data to the frontend.
3. **Product Data API**
    - **Responsibilities**: Acts as an intermediary between the frontend and Sanity CMS to fetch product data.
    - **Interactions**:
        - Pulls product information (such as price, images, description) from **Sanity CMS**.
        - Supplies the frontend with product data to render the product listings, details, etc.
4. **Third-Party API (e.g., Shipment Tracking API)**
    - **Responsibilities**: Provides tracking information for shipped orders.
    - **Interactions**:
        - Retrieves shipment status updates via third-party services like FedEx, UPS, etc.
        - The frontend queries the third-party API to provide users with real-time shipment tracking updates.
5. **Payment Gateway (e.g., Stripe)**
    - **Responsibilities**: Handles the payment processing.
    - **Interactions**:
        - Processes payments, including credit cards, digital wallets, and other payment methods.
        - Sends confirmation of successful or failed payments.
        - Updates the order status post-payment, triggering order fulfillment and shipment.
6. **Order Confirmation**
    - **Responsibilities**: Confirms successful transactions and completes the user's purchase flow.
    - **Interactions**:
        - Displays order details and confirmation to the user after payment.

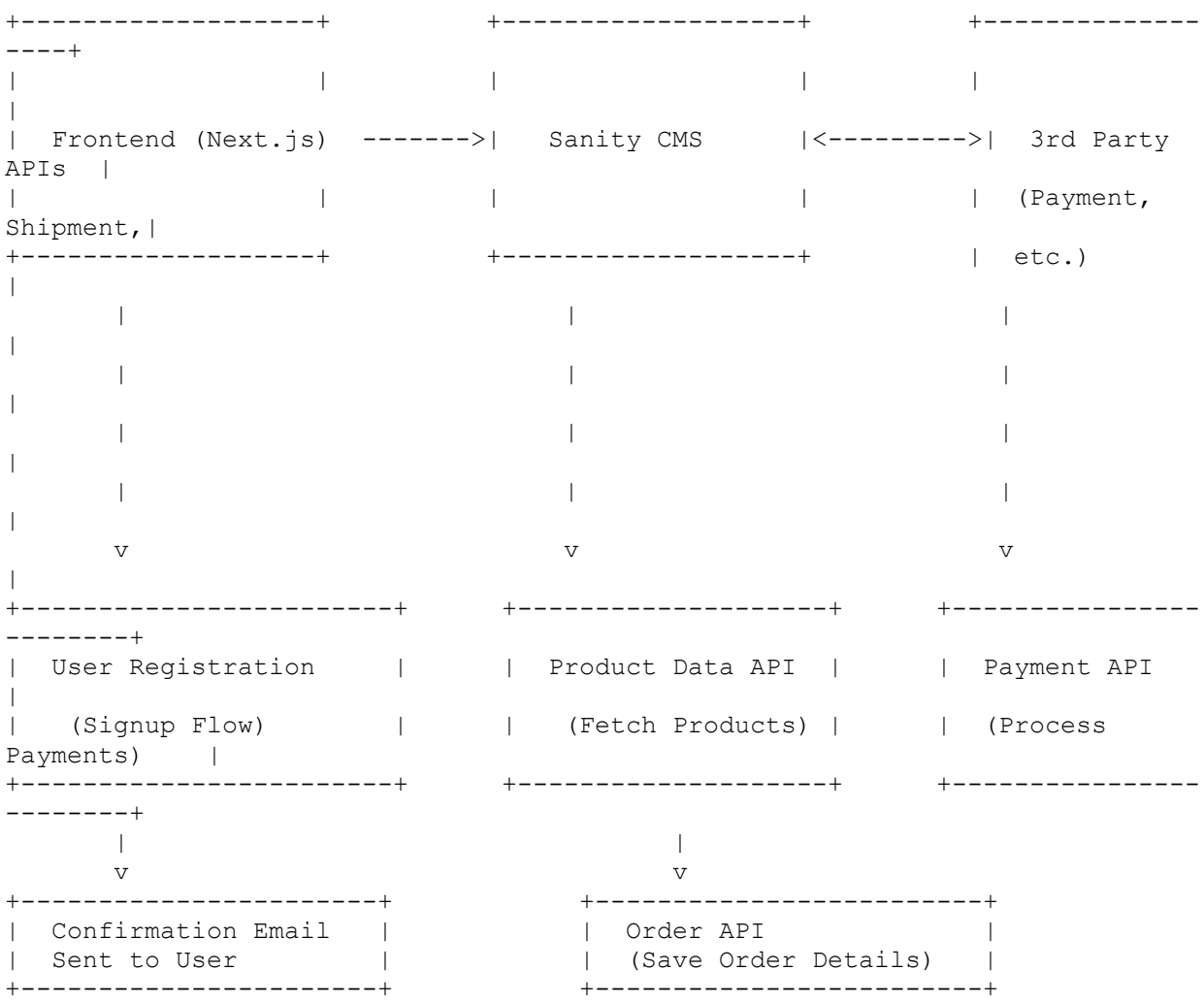
        - Updates order status (e.g., processing, shipped) based on the payment status.

Data Flow in the System:

1. **User Interaction (Frontend)**: The user browses products on the frontend (Next.js), where they can view product listings, details, and add items to their cart.
    - The frontend makes API calls to fetch **product data** from the **Product Data API**, which interacts with **Sanity CMS** to retrieve product information.
2. **Order and Payment**: Once the user proceeds to checkout, they enter payment details.
    - The frontend communicates with the **Payment Gateway** (e.g., Stripe) to process the payment.
    - If payment is successful, the frontend receives a payment confirmation and updates the order status.
3. **Shipment Tracking**: After payment, the order is shipped, and the frontend can request **shipment tracking updates** from the **Third-Party API** (e.g., FedEx, UPS) to show the order's shipping status to the customer.
4. **Order Confirmation**: Finally, once the payment is processed and the order has been shipped, an order confirmation page is displayed, summarizing the order details and tracking information.

Example System Architecture with Key Workflows

Here's an expanded version of your **System Architecture** diagram along with the key workflows included in the interactions between the **Frontend (Next.js)**, **Sanity CMS**, and **Third-Party APIs**:



Breakdown of the Key Workflows

1. User Registration (Signup Flow)

2. Product Browsing (View Product Categories)

3. Order Placement (Add Items to Cart, Checkout)

4. Shipment Tracking (Order Status Updates)

Data Flow Example:

1. **User Registration**:
    - Frontend sends user data to **Sanity CMS** (via API).
    - Sanity stores the data and sends a response.
    - Confirmation email is sent via a **3rd Party API** (e.g., SendGrid).
2. **Product Browsing**:
    - User views products, frontend requests product data from **Sanity CMS** through **Product Data API**.
    - Data is displayed to the user (e.g., list of products, details).
3. **Order Placement**:
    - User places an order (adds to cart, checks out).
    - Order details are sent to **Sanity CMS** to be saved.
    - Payment is processed via a **Payment API** (e.g., Stripe).
    - Sanity updates order status to "Paid" upon confirmation.
4. **Shipment Tracking**:
    - After order is shipped, frontend calls **Shipment Tracking API** for tracking info.
    - Real-time shipment updates are displayed to the user (e.g., tracking status).

Plan API Requirements for a Q-Commerce Marketplace

Based on the data schema for a Q-commerce marketplace (focused on quick delivery of products like perishables), we need to define several API endpoints that will interact with both the **frontend** (Next.js), **Sanity CMS**, and **third-party APIs** (for payment processing, shipment tracking, etc.). Below are the API endpoints that your system may require, including their method, description, and response examples.

---

1. User Registration Endpoint

- **Endpoint Name**: `/register`
- **Method**: `POST`
- **Description**: Accept user details for registration and create a new user profile in **Sanity CMS**.
- **Request Example**:
- ```
  {
    "firstName": "John",
    "lastName": "Doe",
    "email": "john.doe@example.com",
    "password": "password123"
  }
  ```
- **Response Example**:
- ```
  {
    "status": "success",
  ```

- `  "message": "User registered successfully.",`
- `  "userId": "abc123"`
- `}`

---

2. User Login Endpoint

- **Endpoint Name**: `/login`
- **Method**: `POST`
- **Description**: Authenticate user credentials and issue a session or JWT token.
- **Request Example**:
- `{`
- `  "email": "john.doe@example.com",`
- `  "password": "password123"`
- `}`
- **Response Example**:
- `{`
- `  "status": "success",`
- `  "message": "Login successful.",`
- `  "token": "JWT_TOKEN_HERE"`
- `}`

---

3. Product Data Endpoint

- **Endpoint Name**: `/products`
- **Method**: `GET`
- **Description**: Fetch a list of products from **Sanity CMS** to display on the frontend. This endpoint can support pagination and filtering by categories.
- **Request Example**:
- `{`
- `  "category": "fruits",`
- `  "limit": 10,`
- `  "page": 1`
- `}`
- **Response Example**:
- `{`
- `  "status": "success",`
- `  "products": [`
- `    {`
- `      "productId": 101,`
- `      "name": "Organic Apples",`
- `      "description": "Fresh and crisp organic apples.",`
- `      "price": 5.99,`
- `      "imageUrl": "https://example.com/apple.jpg"`
- `    },`
- `    {`
- `      "productId": 102,`

```
        "name": "Bananas",
        "description": "Ripe and sweet bananas.",
        "price": 2.99,
        "imageUrl": "https://example.com/banana.jpg"
      }
    ]
  }
```


---

4. Add Item to Cart Endpoint

- **Endpoint Name**: `/cart/add`
- **Method**: `POST`
- **Description**: Add a product to the user's cart (temporary storage in session or database).
- **Request Example**:


```
{
  "userId": "abc123",
  "productId": 101,
  "quantity": 2
}
```


- **Response Example**:


```
{
  "status": "success",
  "message": "Product added to cart successfully.",
  "cartId": "cart123"
}
```


---

5. Checkout and Order Creation Endpoint

- **Endpoint Name**: `/checkout`
- **Method**: `POST`
- **Description**: Complete the checkout process, save order details to **Sanity CMS**, and initiate payment processing via a third-party payment gateway.
- **Request Example**:


```
{
  "userId": "abc123",
  "cartId": "cart123",
  "shippingAddress": {
    "addressLine1": "123 Main St.",
    "city": "New York",
    "zipCode": "10001",
    "country": "USA"
  },
  "paymentMethod": "stripe",
  "totalAmount": 15.99
}
```


- **Response Example**:

- {
- "status": "success",
- "message": "Order created successfully.",
- "orderId": 12345,
- "paymentLink": "https://paymentgateway.com/checkout/xyz"
- }

---

6. Order Payment Status Endpoint

- **Endpoint Name**: `/order/payment-status`
- **Method**: `GET`
- **Description**: Fetch the payment status of a specific order (e.g., whether it is successful or pending).
- **Request Example**:
- {
- "orderId": 12345
- }
- **Response Example**:
- {
- "status": "success",
- "paymentStatus": "completed",
- "transactionId": "txn_abc123"
- }

---

7. Shipment Tracking Endpoint

- **Endpoint Name**: `/shipment-tracking`
- **Method**: `GET`
- **Description**: Fetch the real-time shipment status for an order, including ETA, current location, and status.
- **Request Example**:
- {
- "orderId": 12345,
- "carrier": "UPS"
- }
- **Response Example**:
- {
- "status": "success",
- "shipmentStatus": "In Transit",
- "currentLocation": "Warehouse, New Jersey",
- "ETA": "2 hours"
- }

---

8. Express Delivery Status (Q-Commerce)

- **Endpoint Name**: `/express-delivery-status`
- **Method**: `GET`
- **Description**: Fetch real-time delivery updates specifically for perishable items (e.g., food), including the delivery time estimate.
- **Request Example**:
- ```
  {
    "orderId": 12345
  }
  ```
- **Response Example**:
- ```
  {
    "orderId": 12345,
    "status": "In Transit",
    "ETA": "15 mins",
    "temperature": "5°C",  // Example: Useful for perishables
    "deliveryNote": "Keep refrigerated"
  }
  ```

---

9. Order Status Update Endpoint

- **Endpoint Name**: `/order/status`
- **Method**: `GET`
- **Description**: Fetch the status of a particular order (e.g., Pending, Shipped, Delivered).
- **Request Example**:
- ```
  {
    "orderId": 12345
  }
  ```
- **Response Example**:
- ```
  {
    "status": "success",
    "orderStatus": "Shipped",
    "orderId": 12345,
    "trackingNumber": "UPS12345"
  }
  ```

---

10. Admin Order Management Endpoint

- **Endpoint Name**: `/admin/orders`
- **Method**: `GET`
- **Description**: Fetch all orders for administrative management, showing their current status (pending, shipped, delivered).
- **Request Example**:
- ```
  {
    "statusFilter": "pending"
  }
  ```
- **Response Example**:
  ```

- ```
  {
    "status": "success",
    "orders": [
      {
        "orderId": 12345,
        "userId": "abc123",
        "totalAmount": 15.99,
        "orderStatus": "pending",
        "paymentStatus": "pending"
      },
      {
        "orderId": 12346,
        "userId": "abc124",
        "totalAmount": 22.50,
        "orderStatus": "shipped",
        "paymentStatus": "completed"
      }
    ]
  }
  ```

API Endpoints Based on Data Schema

Here's a structured list of API endpoints based on your data schema. These endpoints will allow your system to fetch product details, create orders, and track shipments.

---

1. Fetch All Available Products

- **Endpoint Name**: `/products`
- **Method**: `GET`
- **Description**: Fetch all available products from **Sanity CMS**. This endpoint retrieves product details, including name, price, stock level, and an image URL, from the database.
- **Request Example**:
- ```
  {
    "limit": 20,
    "page": 1
  }
  ```
- **Response Example**:
- ```
  {
    "status": "success",
    "products": [
      {
        "id": "product123",
        "name": "Organic Apple",
        "price": 5.99,
        "stock": 100,
  ```
  ```

```
        "image": "https://example.com/images/apple.jpg"
      },
      {
        "id": "product124",
        "name": "Fresh Banana",
        "price": 2.49,
        "stock": 200,
        "image": "https://example.com/images/banana.jpg"
      }
    ]
  }
```

---

2. Create a New Order

- **Endpoint Name**: `/orders`
- **Method**: `POST`
- **Description**: Create a new order in **Sanity CMS**. This endpoint saves order details, including customer information, product details, and payment status. It triggers the order creation process in the backend and updates the status.
- **Payload Example**:

```
{
  "customerInfo": {
    "name": "John Doe",
    "email": "johndoe@example.com",
    "phone": "123-456-7890",
    "shippingAddress": "123 Main St, City, Country"
  },
  "orderItems": [
    {
      "productId": "product123",
      "quantity": 2,
      "price": 5.99
    },
    {
      "productId": "product124",
      "quantity": 1,
      "price": 2.49
    }
  ],
  "totalAmount": 14.47,
  "paymentStatus": "pending",
  "shippingMethod": "express"
}
```

- **Response Example**:

```
{
  "status": "success",
  "message": "Order created successfully.",
```

```
  "orderId": "order12345",
  "paymentStatus": "pending",
  "shippingStatus": "awaiting fulfillment"
}
```


---

3. Track Order Shipment Status

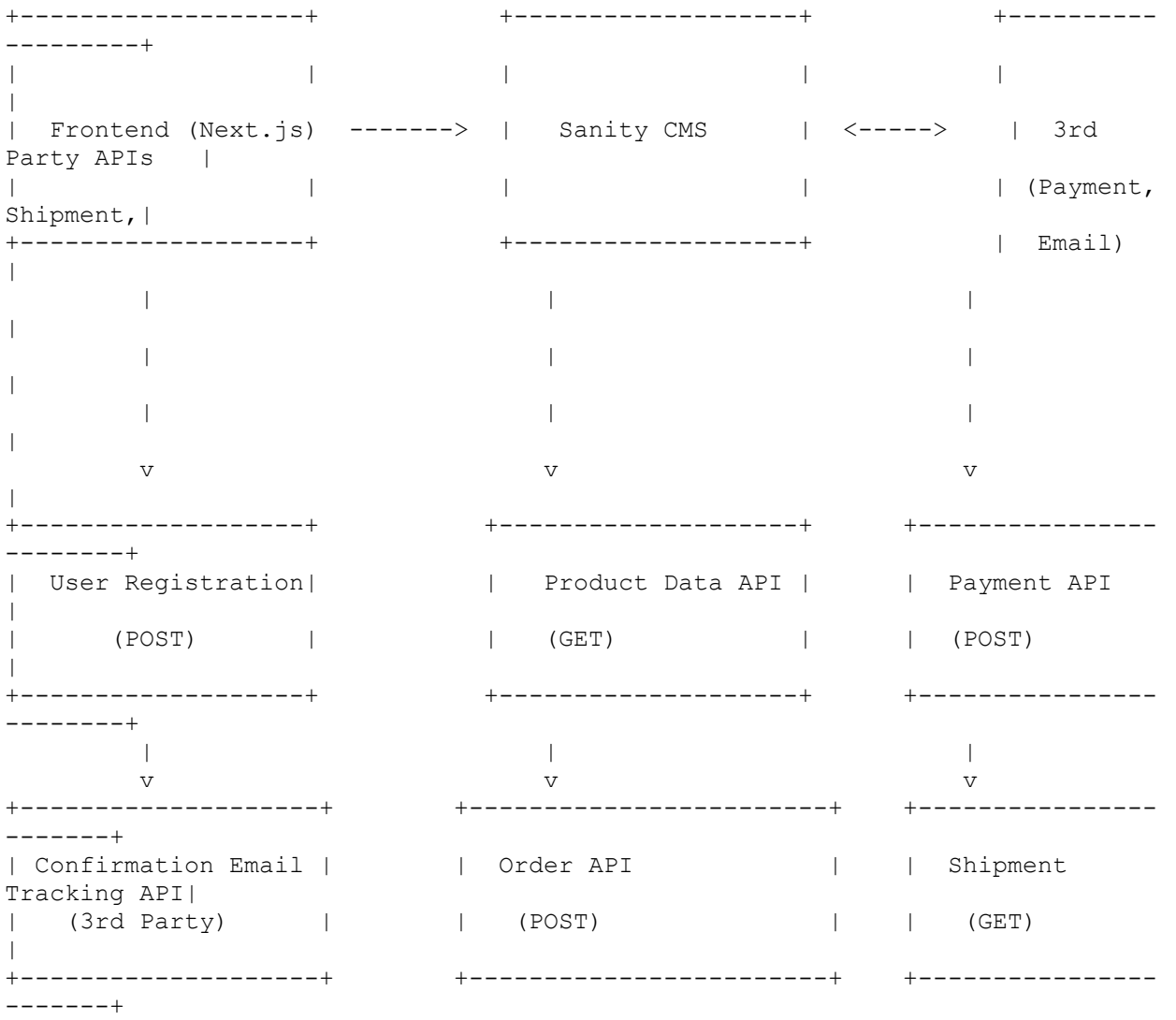
- **Endpoint Name**: `/shipment`
- **Method**: `GET`
- **Description**: Track the status of an order's shipment through a third-party API (e.g., FedEx, UPS). This endpoint retrieves tracking details like shipment ID, order ID, current shipment status, and expected delivery date.
- **Request Example**:


```
{
  "orderId": "order12345"
}
```


- **Response Example**:


```
{
  "status": "success",
  "shipmentDetails": {
    "shipmentId": "shp67890",
    "orderId": "order12345",
    "status": "In Transit",
    "expectedDeliveryDate": "2025-01-18T14:30:00Z",
    "carrier": "UPS"
  }
}
```

System Architecture Diagram:



2. API Specification Document

This document defines the API endpoints needed to interact with the system, including methods, request payloads, and responses.

Endpoint 1: Fetch All Available Products

- **Endpoint**: `/products`
- **Method**: `GET`
- **Description**: Fetch all available products from **Sanity CMS**.

Request:

```
{
```

```
  "limit": 20,
  "page": 1
}
```

Response:

```
{
  "status": "success",
  "products": [
    {
      "id": "product123",
      "name": "Organic Apple",
      "price": 5.99,
      "stock": 100,
      "image": "https://example.com/images/apple.jpg"
    },
    {
      "id": "product124",
      "name": "Fresh Banana",
      "price": 2.49,
      "stock": 200,
      "image": "https://example.com/images/banana.jpg"
    }
  ]
}
```

Endpoint 2: Create a New Order

- **Endpoint**: `/orders`
- **Method**: `POST`
- **Description**: Create a new order in **Sanity CMS**.

Request:

```
{
  "customerInfo": {
    "name": "John Doe",
    "email": "johndoe@example.com",
    "phone": "123-456-7890",
    "shippingAddress": "123 Main St, City, Country"
  },
  "orderItems": [
    {
      "productId": "product123",
      "quantity": 2,
      "price": 5.99
    },
    {
      "productId": "product124",
      "quantity": 1,
      "price": 2.49
    }
  ],
  "totalAmount": 14.47,
  "paymentStatus": "pending",
  "shippingMethod": "express"
}
```

Response:

```
{
  "status": "success",
  "message": "Order created successfully.",
  "orderId": "order12345",
  "paymentStatus": "pending",
  "shippingStatus": "awaiting fulfillment"
}
```

Endpoint 3: Track Order Shipment

- **Endpoint**: `/shipment`
- **Method**: `GET`
- **Description**: Track the order shipment status via a third-party API.

Request:

```
{
  "orderId": "order12345"
}
```

Response:

```
{
  "status": "success",
  "shipmentDetails": {
    "shipmentId": "shp67890",
    "orderId": "order12345",
    "status": "In Transit",
    "expectedDeliveryDate": "2025-01-18T14:30:00Z",
    "carrier": "UPS"
  }
}
```

---

3. Workflow Diagram

The **workflow diagram** visualizes user interactions and data flows through the system:

- **User Registration**: Users register through the frontend → Data is sent to **Sanity CMS** → Confirmation email is sent through **3rd Party Email API**.
- **Product Browsing**: Users browse products → Frontend fetches product data from **Sanity CMS** via the **Product Data API**.
- **Order Placement**: Users add products to the cart → User proceeds to checkout → Order data is sent to **Sanity CMS** → Payment details are processed through **Payment API** → Order status is updated.
- **Shipment Tracking**: After the order is placed, shipment status is fetched via **Shipment Tracking API** → Data displayed on the frontend.

---

4. Data Schema Design

This document outlines the entities and relationships for the database or CMS.

Entities and Relationships:

1. **User**:
    - Attributes: `userId`, `name`, `email`, `password`, `address`
    - Relationship: One-to-many with **Order** (A user can have many orders)
2. **Product**:
    - Attributes: `productId`, `name`, `description`, `price`, `stock`, `imageUrl`
    - Relationship: Many-to-many with **OrderItem** (A product can appear in many orders)
3. **Order**:
    - Attributes: `orderId`, `userId`, `totalAmount`, `paymentStatus`, `orderDate`
    - Relationship: Many-to-one with **User**, Many-to-many with **Product** through **OrderItem**
4. **OrderItem**:
    - Attributes: `orderItemId`, `orderId`, `productId`, `quantity`, `price`
    - Relationship: Many-to-one with both **Order** and **Product**

---

5. Technical Roadmap

A **technical roadmap** outlines the steps to complete the project, including milestones and deliverables:

Phase 1: Planning and Design

- **Duration**: 2 weeks
- **Milestones**:
    - Define system architecture and components.
    - Create initial database schema and CMS design.
    - Develop API specifications and document requirements.

Phase 2: Backend Development

- **Duration**: 4 weeks
- **Milestones**:
    - Set up **Sanity CMS** with required schemas (Product, Order, User).
    - Develop backend API endpoints for product management, order processing, and shipment tracking.
    - Integrate third-party APIs for payment and shipment tracking.

Phase 3: Frontend Development

- **Duration**: 4 weeks
- **Milestones**:
    - Develop user interfaces for product browsing, shopping cart, and order management.
    - Implement communication with backend APIs.

- Implement responsive design for mobile and desktop.

Phase 4: Testing and Quality Assurance

- **Duration**: 2 weeks
- **Milestones**:
  - Conduct unit tests, integration tests, and UI tests.
  - Perform load testing for high-traffic scenarios.
  - Address bugs and optimize performance.

Phase 5: Deployment and Maintenance

- **Duration**: 1 week
- **Milestones**:
  - Deploy the system to production.
  - Set up monitoring and logging for ongoing maintenance.
  - Address any post-launch issues.

Q-Commerce Focus

For **Q-Commerce**, the platform needs to emphasize **real-time inventory updates**, **delivery SLA tracking**, and **express delivery workflows** to ensure that perishable items or time-sensitive products are processed and delivered quickly. The following endpoints and functionalities are crucial:

Express Delivery Workflow

- **Endpoint Name**: `/express-delivery-status`
- **Method**: `GET`
- **Description**: Fetch real-time delivery updates for perishable or time-sensitive products. This includes tracking delivery SLAs (Service Level Agreements) and expected delivery times.

Request Example:

```
{
  "orderId": "order12345"
}
```

Response Example:

```
{
  "status": "success",
  "orderId": "order12345",
  "shipmentStatus": "In Transit",
  "expectedDeliveryTime": "2025-01-18T14:30:00Z",
  "temperature": "5°C",  // Important for perishable goods
  "specialInstructions": "Keep refrigerated until delivery"
}
```

Real-Time Inventory Updates

For **Q-Commerce**, it's essential that product availability is continuously updated in real time. Whenever an item is added to a cart, the system should check whether it is still available and update the stock count in **Sanity CMS** accordingly.

- **Product API**: Every time a user adds a product to the cart, the frontend checks the stock level in **Sanity CMS**.
- **Inventory Updates**: When an order is placed, **Sanity CMS** updates the stock level and triggers an API call to update inventory.

Express Delivery SLA Tracking

The system must support **SLA tracking** to ensure that customers are informed of expected delivery times. Express delivery features should allow:

1. **Real-time ETA (Estimated Time of Arrival)** updates.
2. **Delivery Status** tracking that indicates if the delivery is on time, delayed, or completed.

API Endpoints Documentation

Below is a structured table that documents the key API endpoints for the **QuickMart** marketplace. These endpoints are used to fetch product details, create orders, and track shipments. This format ensures clarity and provides a quick reference for developers and stakeholders.

Endpoint	Method	Purpose	Request Example	Response Example
/products	GET	Fetches all product details	{ "limit": 20, "page": 1 }	{ "status": "success", "products": [{ "id": 1, "name": "Product A", "price": 100 }] }
/orders	POST	Creates a new order	{ "customerInfo": {...}, "orderItems": [{...}], "totalAmount": 200 }	{ "status": "success", "orderId": "order12345", "message": "Order created successfully." }
/shipment	GET	Fetches real-time shipment status	{ "orderId": "order12345" }	{ "status": "success", "shipmentDetails": { "shipmentId": "shp67890", "status": "In Transit", "expectedDeliveryDate": "2025-01-18T14:30:00Z" } }

Endpoint	Method	Purpose	Request Example	Response Example
/express-delivery-status	GET	Fetches real-time delivery status for express deliveries	`{ "orderId": "order12345" }`	`{ "status": "success", "orderId": "order12345", "shipmentStatus": "In Transit", "expectedDeliveryTime": "2025-01-18T14:30:00Z", "temperature": "5°C" }`

Sanity Schema Example

This section provides an example schema for the **Product** document within **Sanity CMS**. The schema defines the fields that a product will have and how it will be stored in the CMS. This structure is designed to align with the backend and API interactions.

Product Schema Example

```
export default {
  name: 'product',
  type: 'document',
  fields: [
    {
      name: 'name',
      type: 'string',
      title: 'Product Name',
      description: 'The name of the product that will be displayed on the
marketplace.'
    },
    {
      name: 'price',
      type: 'number',
      title: 'Price',
      description: 'The price of the product, in the relevant currency.'
    },
    {
      name: 'stock',
      type: 'number',
      title: 'Stock Level',
      description: 'The current stock level of the product. This will be used
to determine availability.'
    },
    {
      name: 'image',
      type: 'image',
      title: 'Product Image',
      description: 'An image of the product to display on the product page.'
    },
    {
      name: 'description',
      type: 'text',
      title: 'Product Description',
```

```
      description: 'A detailed description of the product that will help
users understand the features and benefits.'
    },
    {
      name: 'category',
      type: 'string',
      title: 'Category',
      description: 'The category to which this product belongs (e.g.,
electronics, groceries, etc.).'
    }
  ]
};
```

Explanation of Schema Fields:

- **`name`**: The name of the product, a required field.
- **`price`**: A numeric value representing the product price.
- **`stock`**: The quantity available for the product.
- **`image`**: Stores an image of the product, essential for product display on the frontend.
- **`description`**: A detailed text description of the product, which can be used on the product page to provide additional context.
- **`category`**: The category that the product belongs to, which helps in filtering and browsing on the frontend.

This schema will be used in **Sanity CMS** to manage product data and will interact with the API to fetch the necessary details for the frontend.