Marketplace Technical Foundation - Q-Commerce (Food Truck)

Introduction

Our Q-Commerce platform aims to simplify and enhance the customer experience while bringing unparalleled convenience to the digital space. By launching this platform online, users will gain easy access to a wide range of products with swift delivery, eliminating traditional barriers to quick and efficient service.

In the future, we plan to introduce innovative features that are not yet available in the market but address the essential needs of today's fast-paced lifestyle. These features will include AI-driven recommendations, real-time delivery tracking, personalized user experiences, and eco-friendly packaging solutions, ensuring our platform remains innovative and continues to meet evolving customer needs.

Technical Requirements

Frontend Requirements

1. **User-Friendly Interface**
    - Clear navigation with easy-to-use buttons and menus.
    - Intuitive design to guide users seamlessly.
    - Consistent layout across all pages for a cohesive experience.
2. **Essential Pages**
    - **Home Page:** Main page with featured products, highlighting the latest designs and promotions.
    - **Product Listing Page:** Display all products with filters and sorting options for a personalized shopping experience.

- **Product Details Page**
    - Show full details of a product with "Add to Cart" functionality for quick purchase.
    - Include customer support contact details for any queries or issues.

Sanity CMS as Backend

- Store all product details in Sanity CMS (e.g., name, ID, categories, stock, price, image, description).
- Easily add, update, or delete products through the Sanity dashboard.

Customer Details Management

- Store customer information such as email, contact name, phone number, and address.
- Track customer orders and preferences for personalized experiences.

Order Record Management

- Save order details (e.g., order ID, products, quantity, total order value).
- Maintain order history for each customer for tracking and future reference.

Product Schema

Manages all product-related information including name, category, price, stock, and description.

```
{
  "name": "product",
  "title": "Product",
  "type": "document",
  "fields": [
    {
      "name": "name",
      "title": "Product Name",
      "type": "string"
    },
    {
      "name": "slug",
      "title": "Slug",
      "type": "slug",
      "description": "A unique URL-friendly identifier"
    },
    {
      "name": "category",
      "title": "Category",
      "type": "string"
    },
    {
      "name": "price",
      "title": "Price",
      "type": "number"
    },
    {
      "name": "stock",
      "title": "Stock",
      "type": "number"
    },
    {
      "name": "description",
      "title": "Description",
      "type": "text"
    },
    {
      "name": "image",
```

```
      "title": "Product Image",
      "type": "image",
      "options": {
        "hotspot": true
      }
    },
    {
      "name": "createdAt",
      "title": "Creation Date",
      "type": "datetime"
    }
  ]
}
```

Order Schema

Contains information about each order, such as the unique order ID, products purchased, quantities, and prices.

```
json
CopyEdit
{
  "name": "order",
  "title": "Order",
  "type": "document",
  "fields": [
    {
      "name": "orderId",
      "title": "Order ID",
      "type": "string"
    },
    {
      "name": "customer",
      "title": "Customer",
      "type": "reference",
      "to": [{ "type": "customer" }]
    },
    {
      "name": "products",
      "title": "Products",
      "type": "array",
      "of": [
        {
          "type": "object",
          "fields": [
            {
              "name": "product",
              "title": "Product",
              "type": "reference",
              "to": [{ "type": "product" }]
            },
            {
              "name": "quantity",
              "title": "Quantity",
              "type": "number"
```

```
            },
            {
              "name": "price",
              "title": "Price",
              "type": "number"
            }
          ]
        }
      ]
    },
    {
      "name": "totalAmount",
      "title": "Total Amount",
      "type": "number"
    },
    {
      "name": "status",
      "title": "Order Status",
      "type": "string",
      "options": {
        "list": [
          { "title": "Pending", "value": "pending" },
          { "title": "Shipped", "value": "shipped" },
          { "title": "Delivered", "value": "delivered" },
          { "title": "Cancelled", "value": "cancelled" }
        ]
      }
    },
    {
      "name": "createdAt",
      "title": "Order Date",
      "type": "datetime"
    }
  ]
}
```

Customer Schema

Stores customer details such as name, email, phone, shipping address, and preferences.

```
json
CopyEdit
{
  "name": "customer",
  "title": "Customer",
  "type": "document",
  "fields": [
    {
      "name": "firstName",
      "title": "First Name",
      "type": "string"
    },
    {
      "name": "lastName",
```

```
      "title": "Last Name",
      "type": "string"
    },
    {
      "name": "email",
      "title": "Email",
      "type": "string"
    },
    {
      "name": "phone",
      "title": "Phone Number",
      "type": "string"
    },
    {
      "name": "address",
      "title": "Shipping Address",
      "type": "string"
    },
    {
      "name": "preferences",
      "title": "Customer Preferences",
      "type": "text"
    },
    {
      "name": "createdAt",
      "title": "Registration Date",
      "type": "datetime"
    }
  ]
}
```

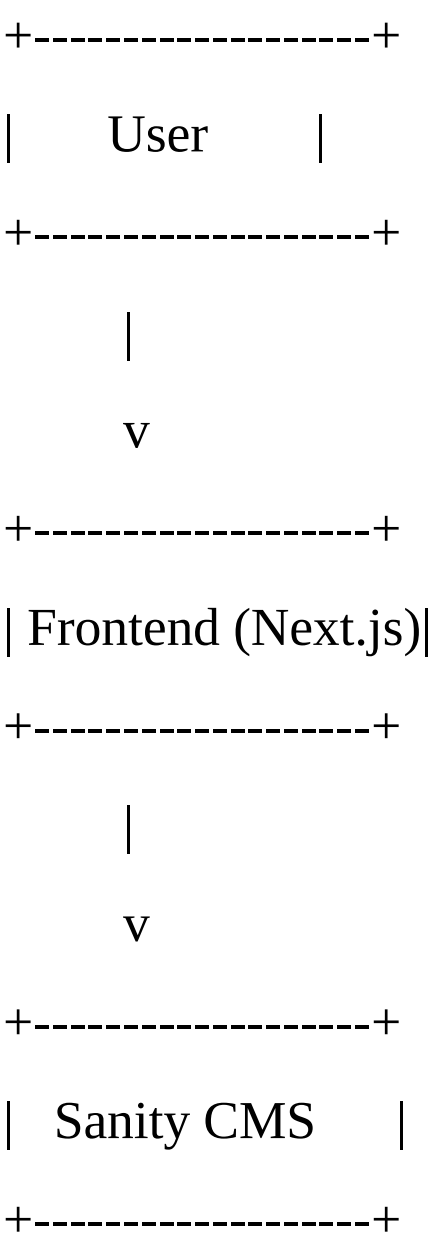
Third-Party API Integration

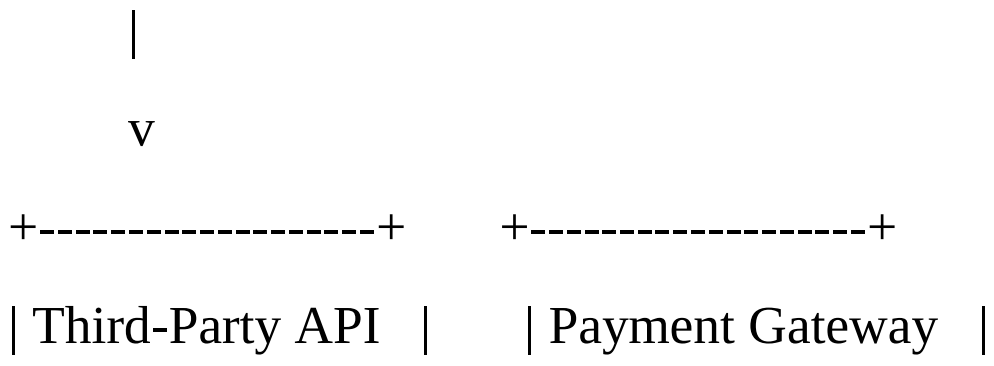
1. **Shipment Tracking API Integration:**
    - Integrate third-party APIs to provide real-time shipment tracking information.
    - Ensure that the shipment status (e.g., "Shipped", "In Transit", "Delivered") is updated dynamically on the user interface.
    - Allow users to track their orders directly from the platform with seamless updates.
2. **Payment Gateway API Integration:**
    - Integrate reliable payment gateway APIs (e.g., Stripe, PayPal) to process customer payments securely.
    - Enable multiple payment methods such as credit/debit cards, wallets, and net banking.
    - Implement payment confirmation and failure handling to update customers about the payment status.
3. **Other Backend Service APIs:**
    - Integrate third-party APIs to manage inventory, product data, and customer information.
    - Use APIs for fraud detection, email/SMS notifications, and customer support services.

- Ensure proper error handling and smooth communication between APIs and the frontend.

4. **Frontend Data Handling:**
   - Ensure that the APIs provide the necessary data to the frontend for a smooth user experience.
   - Fetch relevant data (e.g., product details, order status, payment status) and display it in real-time.
   - Handle API responses efficiently to minimize delays and provide seamless navigation for the user.

System Architecture





User Registration:

User -> Frontend: Signs up

Frontend -> Sanity CMS: Saves user data

Sanity CMS -> User: Sends confirmation

Product Browsing:

User -> Frontend: Views product categories

Frontend -> Sanity CMS: Fetches product data

Sanity CMS -> Frontend: Sends product details

Frontend -> User: Displays products

Order Placement:

User -> Frontend: Adds items to cart and checks out

Frontend -> Sanity CMS: Saves order details

Frontend -> Payment Gateway: Processes payment

Payment Gateway -> Frontend: Sends payment confirmation

Frontend -> User: Displays order confirmation

Shipment Tracking:

Frontend -> Third-Party API: Fetches tracking info

Third-Party API -> Frontend: Sends tracking status

Frontend -> User: Displays tracking status

API Requirements for Q-Commerce (Restaurant Website)

Fetch Menu Items

- **Endpoint Name**: /menu
- **Method**: GET
- **Description**: Fetches all available menu items from Sanity CMS.
- **Response Example**:


```
[
  {
    "id": 1,
    "name": "Margherita Pizza",
    "price": 500,
    "description": "Classic pizza with fresh mozzarella and basil.",
    "image": "https://example.com/margherita-pizza.jpg"
  },
  {
    "id": 2,
    "name": "Chicken Burger",
    "price": 300,
    "description": "Juicy chicken patty with lettuce and mayo.",
    "image": "https://example.com/chicken-burger.jpg"
  }
]
```

Place an Order

- **Endpoint Name**: /place-order
- **Method**: POST
- **Description**: Creates a new order and saves it in Sanity CMS.
- **Payload**:


```
{
  "customerId": 123,
  "items": [
    {
      "itemId": 1,
      "quantity": 2
    },
    {
      "itemId": 2,
      "quantity": 1
    }
  ],
  "deliveryAddress": "123 Main Street, City, Country",
  "paymentStatus": "Pending"
}
```


Response Example:


```
{
  "orderId": 456,
  "status": "Success",
  "message": "Order placed successfully.",
```

"eta": "30 mins"

}Track Order Status

- **Endpoint Name**: /track-order
- **Method**: GET
- **Description**: Fetches real-time order status and delivery updates.
- **Query Parameters**:
  - orderId: The ID of the order to track.
- **Response Example**:

{

"orderId": 456,

"status": "Out for Delivery",

"eta": "15 mins",

"deliveryAgent": "John Doe",

"contact": "+1234567890"

}

Process Payment

- **Endpoint Name**: /process-payment
- **Method**: POST
- **Description**: Processes payment for an order.
- **Payload**:

{

"orderId": 456,

"paymentMethod": "Credit Card",

"cardDetails": {

"cardNumber": "4111111111111111",

"expiryDate": "12/25",

"cvv": "123"

```
  }
}
```

Response Example:

```
{
  "paymentId": 789,
  "status": "Success",
  "message": "Payment processed successfully."
}
```

Fetch Express Delivery Status

- **Endpoint Name**: /express-delivery-status
- **Method**: GET
- **Description**: Fetches real-time delivery updates for express orders.
- **Query Parameters**:
  - orderId: The ID of the order to track.
- **Response Example**:

```
{
  "orderId": 456,
  "status": "In Transit",
  "eta": "10 mins"
}
```

Submit Rating

- **Endpoint Name**: /submit-rating
- **Method**: POST
- **Description**: Saves the customer's rating and feedback.
- **Payload**:

```
{
  "orderId": 456,
```

```
  "rating": 5,
  "feedback": "Great service and delicious food!"
}
```


Response Example:


```
{
  "status": "Success",
  "message": "Thank you for your feedback!"
}
```


Conclusion

This technical documentation provides a comprehensive blueprint for the development and implementation of **[Q_COMMERCE]**. By outlining the system architecture, key workflows, category-specific instructions, API endpoints, and Sanity schema, we have established a solid foundation for building a scalable and efficient marketplace.