FOODBUDS MAREKETPLACE TECHNICAL FOUNDATION

1. Introduction

Our Q-Commerce marketplace is created to make food delivery fast, easy, and affordable. We offer a wide variety of meals, including desi dishes, fast food, exotic cuisines, desserts, and drinks, all made with fresh ingredients.

Our goal is to solve the problem of accessing high-quality, delicious food quickly and conveniently. Whether you're a busy professional, a student, or just craving something tasty late at night, our platform is here for you.

With simple ordering, fast delivery, and options to customize your meals, we aim to provide a great experience for everyone.

2. Project Description

- **User Registration:**
  - Any member can register and browse available food items.
  - Only registered members can place orders and customize meals.
- **User Roles:**
  - **Visitor:**
    - Can view, search, and compare available products.
    - Can customize, and order food.
    - Can also become a member.
    -
  - **Operator:**
    - Can access and manage the admin panel with the ability to add or view records.
  - **Admin:**
    - Holds full privileges, including:
      - Adding, editing, or removing food items.
      - Managing user information (add, edit, or delete users).
      - Tracking orders and confirming deliveries.
- **Order Processing:**
  - Admin can process and ship orders to users and send confirmation notifications.

3. Technical Requirements

a. Frontend (Next.js with TypeScript)

- **Framework:** Next.js (React-based framework) with TypeScript for type safety.

- **Responsibilities:**
  - User interaction (browsing, adding items to cart, placing orders).
  - Sending API requests to the backend.
  - Displaying order statuses and shipment tracking.
  - App Routes: Home, About, Blog, BlogDetails, Shop, ShopDetails, ShoppingCart, Checkout, Sign-in, create-account, f-a-q.
- **Key Features:**
  - Responsive design for desktop and mobile.
  - Integration with the backend API.
  - Dynamic routing for products, orders, and user profiles.
  - Error handling and loading states for API responses.

a. Backend (Sanity CMS)

- **Technology:** Sanity CMS for managing structured content.
- **Responsibilities:**
  - Manage product data (categories, tags, availability).
  - Store and process customer details and orders.
  - Interface with APIs for third-party services like shipment tracking and payments.
- **Key Features:**
  - CRUD operations for products and users.
  - Handle API requests from the frontend.
  - Process orders and update statuses (e.g., Pending, Delivered).

b. Third-Party APIs:

1. **Shipment Tracking API:**
   - **Purpose:** Track and update shipment status.
   - **Integration:** Fetch delivery status and send updates to users.
   - **Key Functions:**
     - Real-time tracking of delivery orders.
     - API authentication for secure communication.
2. **Payment Gateway API:**
   - **Purpose:** Securely process payments.
   - **Integration:** Handle transactions during order placement.
   - **Key Functions:**
     - Payment authorization and confirmation.
     - Refund processing (if required).
3. **Delivery Zone API:**
   - **Purpose:** Fetch available delivery zones and validate addresses.
   - **Integration:** Ensure the customer's location falls within the serviceable zones.
   - **Key Functions:**
     - Return coverage areas.
     - Map orders to delivery zones.

Key Points of System Architecture:

1. User will interact with the frontend.
2. Frontend send request to the backend (Sanity CMS).
3. APIs will handle payment, shipment details and rider's assignment according to the delivery zone.

4. API Requirements:

a. Product API:

Endpoint: **/products**

Method: **GET**

Purpose: Fetch a list of all products from Sanity.

Request Parameter: Not required.

Response: [{
"id": "3",
"name": "Burger",
"price": 21,
"stock": 15,
"image": "url",
"category": "Fast food",
"tags": "spicy"
}]

b. Order API:

Endpoint: **/orders**

Method: **POST**

Purpose: Create a new order in Sanity.

Request Parameter: {

"customer id": "100",

"Products": [

{"productId": "3", "quantity":2,},

{"productId": "1", "quantity":1,},

{"productId": "2", "quantity":1,},

],

"total price": 87,

"delivery Address": "098, Steel Town, Karachi"}

Response: {

“orderId”: “12345”,

“status”: “Order Confirmed”,

“estimatedDelivery”: “2025-01-17”}

c. Shipment Tracking API:

Endpoint: **/ express-delivery-status**

Method: **GET**

Purpose: Retrieve express delivery status of an order.

Request Parameter: The unique Id of the newly placed order.

Response: {

“orderId”: “12345”,

“status”: “In Transit”,

“ETA”:”30mins”}

d. User Authentication API:

- Sign-Up API:

  Endpoint: **/ signup**

  Method: **POST**

  Purpose: Registers a new user.

  Request Parameter:
  {
  “name”: “Essa Abbas”,
  “email”: “essa.abbas@outlook.com”,
  “password”: “secured”,
  “phone”: “0987654321”,
  “address”: “098, Steel Town, Karachi”
  }

  Response:
  {
  “message”: “Registration successful”,
  “customerId”: “101”
  }

- Sign-In API:

  Endpoint: **/ signin**

  Method: **POST**

  Purpose: Authentication of an existing user.

  Request Parameter:
  {
  “email”: “essa.abbas@outlook.com”,
  “password”: “secured”,
  }

  Response:
  {
  “token”: “aYsn18BSann2s5Y”,
  “customerId”: “101”
  }

- Token Validation API:

  Endpoint: **/ validate-token**

  Method: **GET**

  Purpose: Validate the JWT token of the user.

  Authorization: Bearer <JWT token>

  Response:
  {
  “isValid”: “true”,
  “customerId”: “101”
  }

5. Data Schema Design:

Product Schema:

```
export default{

  name: 'product',

type: 'document',
```

```
    title: 'Product',
    fields:[
      {name: 'name', type: 'string', title: 'Product Name' },
      {name: 'price', type: 'number', title: 'Price' },
      {name: 'stock', type: 'number', title: 'Stock' },
      {name: 'image', type: 'string', title: 'Image' },
      {name: 'category', type: 'string', title: 'Category' },
      {name: 'tags', type: 'string', title: 'Tags' },
    ]
    };
```

Order Schema:

```
export default {
  name: 'order',
  type: 'document',
  title: 'Order',
  fields: [
    { name: 'orderId', type: 'string', title: 'Order ID' },
    { name: 'customer', type: 'reference', to: [{ type: 'customer' }], title: 'Customer' },
    { name: 'products', type: 'array', of: [{ type: 'reference', to: [{ type: 'product' }] }], title: 'Products' },
    { name: 'quantity', type: 'array', of: [{ type: 'number' }], title: 'Quantity' },
    { name: 'status', type: 'string', title: 'Order Status' },
    { name: 'totalAmount', type: 'number', title: 'Total Amount' },
    { name: 'orderDate', type: 'datetime', title: 'Order Date' }
  ]
};
```

Customer Schema:

```
export default {
  name: 'customer',
  type: 'document',
  title: 'Customer',
  fields: [
    { name: 'customerId', type: 'string', title: 'Customer ID' },
    { name: 'name', type: 'string', title: 'Full Name' },
    { name: 'contactInfo', type: 'object', fields: [
        { name: 'phone', type: 'string', title: 'Phone Number' },
        { name: 'email', type: 'string', title: 'Email Address' }
      ], title: 'Contact Info' },
    { name: 'address', type: 'text', title: 'Delivery Address' },
    { name: 'orderHistory', type: 'array', of: [{ type: 'reference', to: [{ type: 'order' }] }], title: 'Order History' }
  ]
};
```

Delivery Zone Schema:

```
export default {
  name: 'deliveryZone',
  type: 'document',
  title: 'Delivery Zone',
  fields: [
    { name: 'zoneName', type: 'string', title: 'Zone Name' },
    { name: 'coverageArea', type: 'array', of: [{ type: 'string' }], title: 'Coverage Area' },
    { name: 'assignedDrivers', type: 'array', of: [{ type: 'string' }], title: 'Assigned Drivers' }
  ]
};
```

Express Delivery Schema:

```
export default {
  name: 'expressDelivery',
  type: 'document',
  title: 'Express Delivery',
  fields: [
    { name: 'deliveryId', type: 'string', title: 'Delivery ID' },
    { name: 'order', type: 'reference', to: [{ type: 'order' }], title: 'Order' },
    { name: 'status', type: 'string', title: 'Delivery Status' },
    { name: 'deliveryDate', type: 'datetime', title: 'Delivery Date' }
  ]
};
```

6. Workflow Diagram:

[Customer]

↓

[Places Order]

↓

[Order Details]

↓

[Order Confirmation]

↓

[Kitchen/Restaurant]

↓

[Order Preparation]

↓

[Express Delivery] ↔ [Assigned Rider]

↓

[Delivery to Customer]

↓

[Customer Receives Order] ↔ [Customer Feedback/Rating]

7. Conclusion

FoodBuds provide best, fresh and affordable dishes that fulfills the customer’s watery taste buds desire and customer will have great shopping experience with modern technology.