"Marketplace Technical Foundation - [F.A E-Commerce]"

Frontend Requirements Documentation

1. Overview

This document outlines the frontend requirements for the e-commerce platform. The aim is to provide a user-friendly and responsive interface for browsing products and completing orders, ensuring a seamless experience across mobile and desktop devices.

2. User Interface (UI) Design

- **User-friendly Interface**:
  - Easy navigation and intuitive layout for a smooth shopping experience.
  - Clear categories and filters for product browsing.
  - Prominent search bar for quick product lookup.
- **Color Scheme & Branding**:
  - Consistent use of the brand colors across the platform.
  - Visually appealing design with proper contrast and readability.
- **Typography**:
  - Use of legible fonts with appropriate sizes for headings, subheadings, and body text.
  - Consistency in font style across all pages.
- **Navigation**:
  - Clear header and footer with essential links such as Home, Product Categories, Contact Us, and Customer Support.
  - Cart icon visible at all times for easy access to the shopping cart.

3. Responsive Design

- **Mobile and Desktop Compatibility**:
  - The design must adjust according to the screen size, from mobile to desktop devices.
  - Essential UI elements (e.g., header, navigation menu, product cards) should be visible and accessible on both small and large screens.
- **Breakpoints**:
  - Mobile (up to 767px): Single-column layout, stacked product images, touch-friendly buttons.
  - Tablet (768px to 1023px): Two-column layout with larger images, accessible filters.
  - Desktop (1024px and above): Grid-based layout with product cards, larger navigation options.
- **Interaction & Touch Support**:

- Ensure that buttons, links, and interactive elements are large enough for easy tapping on touch devices.
- Hover effects should be available for desktops but disabled or optimized for mobile users.

4. Essential Pages

a) Home Page

- **Features**:
  - Hero section with the latest promotions or featured products.
  - Categories list for quick browsing (e.g., Electronics, Fashion, Home & Living).
  - Best sellers, trending products, or recommended items based on the user's browsing history.
  - Call-to-action (CTA) buttons for major actions like "Shop Now" or "View Products".

b) Product Listing Page

- **Features**:
  - Grid display of products with pagination or infinite scroll.
  - Filters (by price, category, rating, etc.) and sorting options (e.g., Price: Low to High, Newest, Popular).
  - Search bar for finding specific products.
  - Clear display of product images, titles, and prices.

c) Product Details Page

- **Features**:
  - High-quality images with zoom functionality.
  - Product title, description, specifications, and price.
  - Add to Cart button and quantity selector.
  - Customer reviews and ratings.
  - Related products or recommendations.

d) Cart Page

- **Features**:
  - List of added products with images, names, prices, and quantities.
  - Option to update quantities or remove items from the cart.
  - Total price calculation (with taxes, shipping, and discounts).
  - Proceed to Checkout button.

e) Checkout Page

- **Features**:
    - User login or guest checkout options.
    - Billing and shipping address form.
    - Payment method selection (credit card, PayPal, etc.).
    - Order summary with the ability to review and modify the cart.
    - Confirmation of shipping and delivery options.
    - Place Order button.

f) Order Confirmation Page

- **Features**:
    - Confirmation message with order number.
    - Summary of the ordered items, delivery address, and payment method.
    - Estimated delivery time and tracking information.
    - Option to view order history or continue shopping.

5. Functionality and Interactions

- **Cart Management**:
    - Ability to add/remove items from the cart and update quantities.
    - Visual cues (e.g., cart icon count) to show the number of items in the cart.
- **Checkout Flow**:
    - Secure payment gateway integration for processing payments.
    - Real-time validation for required fields (address, payment details).

Sanity CMS Backend Integration Documentation

1. Overview

This section explains how **Sanity CMS** will be used as the backend for the e-commerce platform. Sanity will serve as the database to manage essential data, such as products, customers, and orders. By using Sanity, we can leverage its flexibility and ease of use in managing structured content.

2. Sanity CMS as the Backend

Sanity CMS will handle the following responsibilities for the platform:

- **Product Data**: Manage product details like name, description, price, images, categories, and inventory.
- **Customer Data**: Store customer details, including name, email, shipping address, and order history.
- **Order Records**: Track customer orders, including order status, product details, payment information, and shipping updates.

3. Designing Schemas in Sanity

To align with the business goals, we need to design several schemas in Sanity to structure the data appropriately.

a) Product Schema

The Product schema will define the structure of product information in the marketplace. The fields may include:

- **Product Name**: Title of the product (string).
- **Description**: Detailed information about the product (text).
- **Price**: Product price (number).
- **Category**: The product category (reference to a category schema).
- **Images**: An array of images (array of references to image objects).
- **Stock Quantity**: How many items are available (number).
- **SKU (Stock Keeping Unit)**: Unique identifier for the product (string).
- **Brand**: Brand name or manufacturer (string).
- **Rating**: Average customer rating (number).
- **Tags**: Keywords to describe the product (array of strings).
- **Slug**: A URL-friendly version of the product name, often used for SEO purposes.
- **PriceWithoutDiscount**: The original price of the product before any discounts are applied.
- **DiscountPercentage**: The percentage of discount applied to the original price.
- **RatingCount**: The total number of customer reviews or ratings the product has received. This can help in showing a complete rating breakdown.
- **Sizes**: An array of available sizes for the product (e.g., "S", "M", "L", "XL").
- **Colour**: An array of available colour options for the product.

```
export default {

  name: 'product',

  title: 'Product',

  type: 'document',

  fields: [

    { name: 'name', type: 'string' },

    { name: 'slug', type: 'slug', options: { source: 'name', maxLength: 200 } },  // Added slug for SEO-friendly URLs
```

```
  { name: 'description', type: 'text' },

  { name: 'price', type: 'number' },

  { name: 'priceWithoutDiscount', type: 'number' },  // Price without discount

  { name: 'discountPercentage', type: 'number', description: 'Discount percentage applied to the product' },  // Discount percentage

  { name: 'category', type: 'reference', to: [{ type: 'category' }] },

  { name: 'images', type: 'array', of: [{ type: 'image' }] },

  { name: 'stockQuantity', type: 'number' },

  { name: 'sku', type: 'string' },

  { name: 'brand', type: 'string' },

  { name: 'rating', type: 'number' },

  { name: 'ratingCount', type: 'number', description: 'Total number of reviews/ratings given to this product' },  // Rating count

  { name: 'tags', type: 'array', of: [{ type: 'string' }] },

  { name: 'sizes', type: 'array', of: [{ type: 'string' }], description: 'Available sizes for the product (e.g., S, M, L)' },  // Sizes

  { name: 'colour', type: 'array', of: [{ type: 'string' }], description: 'Available colours for the product' },  // Colour options

 ],

};
```

b) Category Schema

This schema organizes products into categories, making it easier for customers to browse.

- **Category Name**: The name of the category (string).
- **Description**: Optional description for the category (text).
- **Parent Category**: For hierarchical categorization (reference to another category).
- **Slug**: Added for SEO-friendly URLs for categories.

- **Products**: An array of references to the product schema, so you can associate products with categories.

```
export default {

  name: 'category',

  title: 'Category',

  type: 'document',

  fields: [

    { name: 'name', type: 'string' },

    { name: 'slug', type: 'slug', options: { source: 'name', maxLength: 200 } },  // Slug for SEO-friendly URL

    { name: 'description', type: 'text' },

    { name: 'parentCategory', type: 'reference', to: [{ type: 'category' }] },

    { name: 'products', type: 'array', of: [{ type: 'reference', to: [{ type: 'product' }] }] },  // List of products in this category

  ],

};
```

c) Customer Schema

The Customer schema will store user details for the e-commerce platform:

- **Name**: Full name of the customer (string).
- **Email**: Customer's email address (string).
- **Shipping Address**: Address where orders will be delivered (object).
- **Order History**: Reference to orders placed by the customer (array of references to order documents).

```
export default {
```

```
  name: 'customer',
  title: 'Customer',
  type: 'document',
  fields: [
   { name: 'name', type: 'string' },
   { name: 'email', type: 'string' },
   { name: 'shippingAddress', type: 'object', fields: [
     { name: 'street', type: 'string' },
     { name: 'city', type: 'string' },
     { name: 'zip', type: 'string' },
     { name: 'country', type: 'string' },
   ] },
   { name: 'orderHistory', type: 'array', of: [{ type: 'reference', to: [{ type: 'order' }] }] },
  ],
};
```

d) Order Schema

The Order schema will store details about customer orders:

- **Customer**: Reference to the customer who placed the order (reference to customer).
- **Product List**: List of products in the order (array of references to product documents).
- **Total Price**: Total price of the order (number).
- **Order Status**: Status of the order (e.g., Pending, Shipped, Delivered).
- **Payment Method**: Payment method used by the customer (string).
- **Shipping Address**: Address where the order will be delivered (object).
- **Order Date**: Date when the order was placed (datetime).
- **ProductList**: Each item in the product list now contains additional information:

1)**Size**: The size of the product ordered (e.g., "M", "L").

2)**Colour**: The color of the product ordered (e.g., "Red", "Black").

3)**Quantity**: The number of items ordered.

4)**Price**: The price of the product at the time of purchase (this allows for price tracking even if the product's price changes later).

- **TotalPrice**: This field stores the total amount of the order.
- **ShippingAddress**: Includes the full shipping details of the customer.

```
export default {

name: 'order',

title: 'Order',

type: 'document',

fields: [

 { name: 'customer', type: 'reference', to: [{ type: 'customer' }] },

 {

  name: 'productList',

  type: 'array',

  of: [{

   type: 'object',

   fields: [

    { name: 'product', type: 'reference', to: [{ type: 'product' }] },

    { name: 'size', type: 'string', description: 'Size of the product ordered' },  // Added size

    { name: 'colour', type: 'string', description: 'Colour of the product ordered' },  // Added color

    { name: 'quantity', type: 'number', description: 'Quantity of the product ordered' },
```

```
        { name: 'price', type: 'number', description: 'Price of the product at the time of purchase'
},

      ],

     }]

   },

   { name: 'totalPrice', type: 'number' },

   { name: 'orderStatus', type: 'string', options: { list: ['Pending', 'Shipped', 'Delivered'] } },

   { name: 'paymentMethod', type: 'string' },

   {

     name: 'shippingAddress',

     type: 'object',

     fields: [

       { name: 'street', type: 'string' },

       { name: 'city', type: 'string' },

       { name: 'zip', type: 'string' },

       { name: 'country', type: 'string' },

     ]

   },

   { name: 'orderDate', type: 'datetime' },

  ],

};
```

4. Integration with Frontend

- **Fetching Data**: Use **Sanity's GROQ** queries to fetch product data, customer details, and order records.
- **Dynamic Content**: On the frontend, dynamically populate product pages, product listings, customer order history, etc., based on the data fetched from Sanity.
- **Real-time Updates**: Sanity's real-time functionality allows you to reflect changes made in the CMS immediately on the frontend.

Third-Party API Integration Documentation

1. Overview

This section describes how we will integrate third-party APIs for shipment tracking, payment gateways, and other necessary backend services into the e-commerce platform. The goal is to ensure that these external services provide the required data and functionality for the frontend.

2. APIs to Integrate

The platform will integrate the following APIs:

- **Shipment Tracking API**: To track shipments and provide real-time updates on order status.
- **Payment Gateway API**: To handle payment processing for customers' orders.
- **Other APIs**: (e.g., for inventory management, customer support, etc.)

---

3. Shipment Tracking API Integration

a) API Selection

Choose a shipment tracking API provider that fits the shipping carriers you plan to use (e.g., FedEx, UPS, DHL, or a more general service like AfterShip). For this example, let's assume we're using **AfterShip**.

b) Key Endpoints:

- **Track Shipments**: Use the endpoint to track the shipment status using a tracking number.
    - GET /v4/trackings
    - Parameters: tracking_number, carrier

c) Frontend Data:

- On the order confirmation page, allow customers to input or view the tracking number for their orders.
- Display real-time shipment status by fetching data from the API.

d) Backend API Call (Example with AfterShip):

Here's an example of how you can make an API call from the backend to AfterShip:

```
import axios from 'axios';



const trackShipment = async (trackingNumber: string) => {

  const apiKey = 'your_aftership_api_key';

  const url = `https://api.aftership.com/v4/trackings/${trackingNumber}`;



  try {

    const response = await axios.get(url, {

      headers: {

        'aftership-api-key': apiKey,

      },

    });

    return response.data;

  } catch (error) {

    console.error('Error fetching shipment tracking info:', error);

    throw error;

  }

};
```

- **Data Returned**: The response will include shipment status, estimated delivery date, and any updates from the carrier

4. Payment Gateway API Integration

a) API Selection

You'll need to choose a payment gateway that supports various payment methods (credit/debit cards, PayPal, etc.). Popular choices include **Stripe**, **PayPal**, and **Razorpay**.

b) Key Endpoints:

- **Create Payment Intent**: To initialize a payment request.
  - POST /v1/payment_intents
- **Confirm Payment**: To confirm the payment after processing.
  - POST /v1/payment_intents/{payment_intent_id}/confirm

c) Frontend Data:

- Display payment options to the user (credit card, PayPal, etc.).
- Collect payment information securely (do not store sensitive data directly on your servers).

d) Backend API Call (Example with Stripe):

Here's how to handle the payment processing with **Stripe**:

```
import stripePackage from 'stripe';
const stripe = stripePackage('your_stripe_api_key');

// Create a Payment Intent
const createPaymentIntent = async (amount: number) => {
  try {
    const paymentIntent = await stripe.paymentIntents.create({
      amount: amount * 100, // Amount is in cents
      currency: 'usd',
    });
    return paymentIntent.client_secret;
  } catch (error) {
    console.error('Error creating payment intent:', error);
    throw error;
  }
};
```

- **Frontend**: On the frontend, you would use Stripe's JS library to securely handle card information and complete the payment.

---

5. Other Backend Services

a) Inventory Management API:

- If you are using a third-party service to manage inventory, you will need to integrate APIs to fetch product availability, update stock levels, etc.

b) Customer Support API:

- Some platforms provide customer support ticket management APIs (like Zendesk, Freshdesk).
    - For example, create a ticket for customers by sending their order details to the API.

---

6. API Data Flow and Frontend Integration

- **Order Process**:
    - Once a customer places an order, the order data (including the shipping details and payment status) will be sent to your backend.
    - The backend will interact with the **payment gateway API** to process payments and then with the **shipment tracking API** to initiate tracking.
    - You'll use the returned data from both APIs to update the frontend, displaying the order status, payment confirmation, and shipment tracking details.
- **Order Confirmation Page**:
    - Display payment status (successful/failed).
    - Provide a tracking number and link to real-time tracking.
- **Real-Time Updates**:
    - For shipment updates, periodically poll the **shipment tracking API** or use webhooks (if supported) to get updates automatically.

---

7. Security and Compliance

- **Payment Information**: Never store sensitive payment information (such as card numbers) on your servers. Use tokenization services (like Stripe's JS library) to process payments securely.

- **Shipping Information**: Make sure you're complying with local data protection laws (such as GDPR) when storing and handling customer information.

8. Testing and Error Handling

- **Testing**: Test the integration in a sandbox environment provided by payment gateways (e.g., Stripe's test mode, PayPal's sandbox).
- **Error Handling**: Implement proper error handling in case of API failures (e.g., payment failure, shipment not found). Show user-friendly messages on the frontend.

API Requirements for the E-Commerce Platform

1. Product Management APIs

a) Get All Products

- **Endpoint Name**: /products
- **Method**: GET
- **Description**: Fetch all available products from Sanity.
- **Request Example**:
  - No request body required.
- **Response Example**:

```
[
 {
  "id": "123",
  "name": "Product A",
  "price": 100,
  "stock": 50,
  "image": "url_to_image",
  "description": "Description of Product A",
  "category": "electronics",
  "sizes": ["S", "M", "L"],
  "colours": ["Red", "Blue"]
```

```
  },
  {
    "id": "124",
    "name": "Product B",
    "price": 150,
    "stock": 30,
    "image": "url_to_image",
    "description": "Description of Product B",
    "category": "fashion",
    "sizes": ["M", "L"],
    "colours": ["Black", "White"]
  }
]
```

b) Get Product Details

- **Endpoint Name**: /products/{id}
- **Method**: GET
- **Description**: Fetch detailed information of a specific product by its ID.
- **Request Example**:
  - No request body required, just the product ID in the URL.
- **Response Example**:

```
{
  "id": "123",
  "name": "Product A",
  "price": 100,
```

```
  "stock": 50,
  "image": "url_to_image",
  "description": "Description of Product A",
  "category": "electronics",
  "sizes": ["S", "M", "L"],
  "colours": ["Red", "Blue"],
  "rating": 4.5,
  "discountPercentage": 10,
  "priceWithoutDiscount": 110
}
```

2. Order Management APIs

a) Create Order

- **Endpoint Name**: /orders
- **Method**: POST
- **Description**: Create a new order in Sanity and save the order details.
- **Request Example**:

```
{
  "customerId": "456",
  "productList": [
    {
      "productId": "123",
      "size": "M",
      "colour": "Red",
      "quantity": 2,
```

```
    "price": 100
   },
   {
    "productId": "124",
    "size": "L",
    "colour": "Black",
    "quantity": 1,
    "price": 150
   }
  ],
  "totalPrice": 350,
  "orderStatus": "Pending",
  "paymentMethod": "Credit Card",
  "shippingAddress": {
   "street": "123 Main St",
   "city": "City Name",
   "zip": "12345",
   "country": "Country Name"
  },
  "orderDate": "2025-01-16T10:00:00Z"
}
```


- **Response Example**:

```
{
  "message": "Order created successfully",
  "orderId": "789"
}
```


3. Shipment Tracking API

a) Track Shipment Status

- **Endpoint Name**: /shipment/{orderId}
- **Method**: GET
- **Description**: Track order status through a third-party shipment tracking API.
- **Request Example**:
    - The order ID is passed in the URL to fetch shipment status.
- **Response Example**:


```
{

  "shipmentId": "ABC123456",

  "orderId": "789",

  "status": "Shipped",

  "expectedDeliveryDate": "2025-01-20",

  "currentLocation": "New York, USA",

  "trackingUrl": "https://tracking.com/ABC123456"

}
```


4. Cart Management APIs

a) Get Cart

- **Endpoint Name**: /cart
- **Method**: GET
- **Description**: Retrieve the items in the user's cart.
- **Request Example**:
    - No request body required.
- **Response Example**:


```
{
 "userId": "456",
 "cartItems": [
  {
   "productId": "123",
   "size": "M",
   "colour": "Red",
   "quantity": 2,
   "price": 100
  },
  {
   "productId": "124",
   "size": "L",
   "colour": "Black",
   "quantity": 1,
   "price": 150
  }
 ],
 "totalPrice": 350
```

```
}
```

b) Add Item to Cart

- **Endpoint Name**: /cart/add
- **Method**: POST
- **Description**: Add a product to the user's cart.
- **Request Example**:

```
{
  "userId": "456",
  "productId": "123",
  "size": "M",
  "colour": "Red",
  "quantity": 2
}
```

- **Response Example**:

```
{
  "message": "Product added to cart successfully"
}
```

c) Remove Item from Cart

- **Endpoint Name**: /cart/remove
- **Method**: POST
- **Description**: Remove a product from the user's cart.
- **Request Example**:

```
{
  "userId": "456",
  "productId": "123"
```

```
}
```

- **Response Example**:

```
{
  "message": "Product removed from cart successfully"
}
```

5. Payment API

a) Create Payment Intent

- **Endpoint Name**: /payment
- **Method**: POST
- **Description**: Create a payment intent for order processing.
- **Request Example**:

```
{
  "orderId": "789",
  "amount": 350,
  "currency": "USD",
  "paymentMethod": "Credit Card"
}
```

- **Response Example**:

```
{
  "paymentIntentId": "pi_123456789",
  "clientSecret": "secret_here"
}
```

b) Confirm Payment

- **Endpoint Name**: /payment/confirm
- **Method**: POST
- **Description**: Confirm payment for an order.
- **Request Example**:

```
{
  "paymentIntentId": "pi_123456789",
  "paymentMethod": "Credit Card",
  "paymentDetails": {
    "cardNumber": "1234567890123456",
    "expiry": "12/25",
    "cvv": "123"
  }
}
```

- **Response Example**:

```
{
  "message": "Payment confirmed successfully",
  "paymentStatus": "Completed"
}
```