Marketplace Technical Foundation – Furniro Shop

Introduction:

our E-commerce is Platform aim to provide **Luxury and Traditional Furniture.** This documents outlines the Technical Foundation required to build a scalable, user-friendly marketplace.

Define Technical Requirements:

1.Front-End Requirements:

- **User-friendly Interface**
- **Responsive Design** - Ensure seamless navigation and browsing on mobile, tablet, and desktop devices.
- **Essential Pages:** Home
- Showcase featured collections, seasonal trends, and a prominent search bar.
- Include sections for Latest Collection, Top Categories, Featured Products, and Trending Products

  - About:
    - Highlight the brand story, mission, values, and what sets your home
  - Contact Us
    - Provide a contact form for customer inquiries (fields for name, email, and message).
    - Include phone numbers, email addresses, and a physical address (if applicable).
    - Embed a Google Map for location visibility.

  - Product Listing/Shop:
    - Offer filtering options (e.g., by price).

  - Product Details:
    - Highlight product features such as dimensions, materials, variations and
    - care instructions.
    - Provide customer reviews and ratings.
  - Cart:
    - Include an estimated shipping cost calculator.
  - Checkout:
    - Enable guest checkout and account creation options.
    - Include secure payment fields and shipping address validation.
  - Order Confirmation:
    - Show order summary with delivery timeframes and tracking details.

2.Sanity CMS as a Backend:

Sanity CMS for managing product data, orders, and customer details.

Here's a detailed schema design for **Product**, **Order**, **Customer**, **Payment**, **Shipment**, and **Delivery Zone using Sanity CMS**:

[Product] export
interface Product (
id: string; name: string; description: string; price: number;
Images: [_type: "Image'; asset: [_ref: string; _type: 'reference"]
[]; dimensions: string; material: string; stock: number; category:
string; tags: string[]:
reviews: number
]

[Order] export
interface Order (
id: string; productID:
string; Quantity:
number totalAmount:
number; orderDate:
string
]

[Customer] export
interface Customer [
id: string;
name: string;
email: string;
phone: string;
address: string;
]

[Payment) export

interface Payment (
id: string; order: Order; paymentMethod:
'Credit Card' 'PayPal" | 'Stripe'; status: 'Pending'
| 'Completed' | 'Failed'; transactionid: string
Amount: number;
paymentDate: string;
}

[Shipment] export
interface Shipment {
_id: string; order: Order; tracking Number: string;
status: 'Pending' | 'In Transit' | 'Delivered' |
'Cancelled'; estimated Delivery: string;
}

```
[Delivery Zone] export
Interface DeliveryZone {
_id: string;
zoneName: string;
coverageareas: string[];
shippingCost: number;
carrier/assignedDrivers: string;
}
```

3.THIRD PARTY API:

Integrate APIs for shipment tracking, payment gateways, and other required backend services.

Design System Architecture:

1. Key Workflows:

- **User Registration**: User fill out the form to signs up in frontend Nextjs
- **Sanity CMS** : Data is stored in Sanity
- **Confirmation**: A Confirmation Email sent to the user.
- **Product Browsing:** User Browse Product Categories.
- **User views product categories:** user view product categories.
- **Sanity API fetches data**: Fetch Products data like (Images,name,Title,description,Price,Quantity)
- **Products displayed on frontend:** Products display dynamically to frontend nextjs
- **Order Placement:**
  - User adds items to the cart and Proceeds to checkout
  - Order details are sent in sanity
  - And store in Sanity
- **Payment Gateway:** securely process the payment and process the transaction
- **Shipment Tracking:** Update the order with shipping details details
- **THIRD-PARTY API:** Order status updates fetched via 3rd-party API
- **Displayed to the user:** Display Shipping status on frontend nextjs

High level Diagram:

$$\begin{tikzcd}
\text{[Front-End (Next js)]} \arrow[d] & \\
\text{[Sanity CMS]} \arrow[r] \arrow[d] & \text{[Product Data API]} \\
\text{[Third Party API]} \arrow[r] \arrow[d] & \text{[Shipment Tarcking api]} \\
\text{[Payment Gateway]} &
\end{tikzcd}$$

Plan API Requirement:

1. **Endpoint Name: /products** :
    - **Method**: GET
    - **Description**: Fetch all product details.
    - **Response Example:** { "id": 1, "name": "Product A", "price": 100 }

```
[
  {
 "_id": "prod_001",
  "name": "Syltherine",
  "description": "A stylish wooden coffee table.",
  "price": 150,
  "images": [
            ("_type": "image", "asset": { "_ref": "image_ref_001", "type": "reference")}
     ],
    "dimensions": "120x60x45 cm",
    "material": "wood",
    "stock": 25,
    "category": "Living Room",
    "tags": ["sale", "new arrival"],
    "review": 4.5
 }
]
```

2. **Endpoint Name: /Order** :
    - **Method**: POST
    - **Description**: Create a new order
    - **Payload:**

```
{
    "Productid": "prod_001",
    "Quantity": 2,
    "Totalamount":300,
     "orderdate": " 2025-01-18"
 }
```

Response Example:

```
{
"status": "Success",
"message": "Order created successfully.",
"order": {
"_id": "order_001",
"productID": "product_001",
"quantity": 2,
"totalAmount": 600,
"orderDate": "2025-01-16"
}
```

3. **Endpoint Name: /Payment** :
    - **Method**: POST
    - **Description**: Create a new order

- **Payload:**

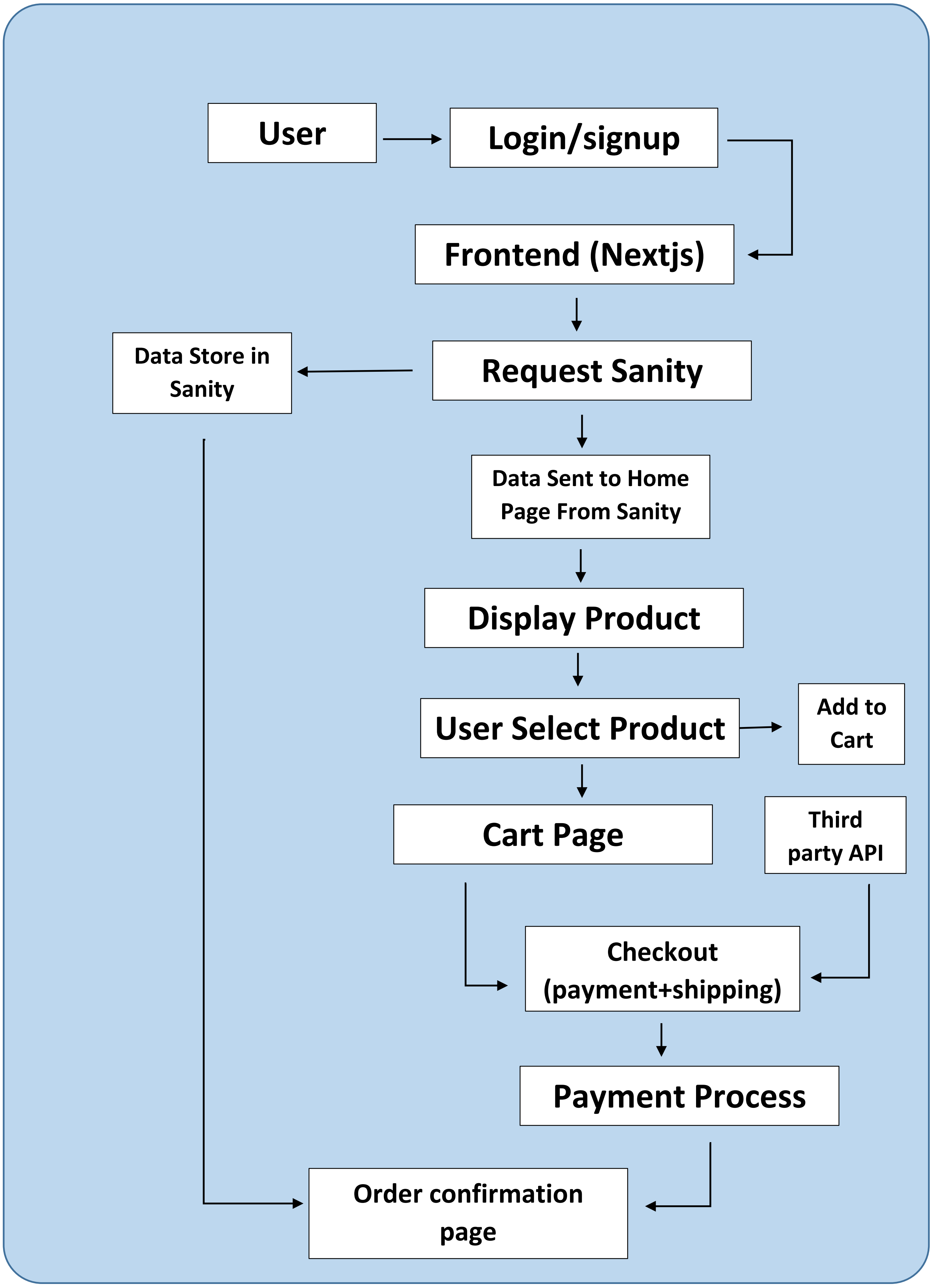
```
{
  "order": {
  "_id": "order_001",
  "totalAmount": 150
},
  "paymentMethod": "Credit Card",
  "status": "Completed",
  "transactionId": "txn_001",
  "Amount": 150,
  "paymentDate": "2025-01-16"
}
```

4. **Endpoint Name: /Shipment** :
   - **Method**: GET
   - **Description**: Track Shipment Status
   - **Example:**

```
{
  "_id": "Ship_001",
  "order": {
          "_id": "Ship_001",
          "TotalAmount": 150,
},
  "TrackingNO": "Track_001",
  "status": "In Transit",
  "EstimatedDelivery": 22-1-25,
}
```

5. **Endpoint Name: /Delivery-zone** :
   - **Method**: GET
   - **Description**: Fetch All delivery Zones and their details
   - **Example:**

```
{
  "_id": "zone_001",
  "ZoneName": "Zone-01",
  "Coveragearea": "In Transit",
  "EstimatedDelivery": 22-1-25,
}
```


User
Login/signup
Frontend (Nextjs)
Request Sanity
Data Store in Sanity
Data Sent to Home Page From Sanity
Display Product
User Select Product
Add to Cart
Cart Page
Third party API
Checkout (payment+shipping)
Payment Process
Order confirmation page

Explanation of the Diagram:

1. Frontend (Next.js)

The user interacts with the frontend, which is built using Next.js. The frontend displays products, manages the cart, and handles the checkout process.

It communicates with Sanity CMS to fetch product data and manage orders.

It also interacts with third-party APts for payment processing and shipment tracking

2. Sanity CMS

Sanity acts as the backend database and CMS. It stores product data, customer information, and order detalls.

The frontend makes Ali requests to Senity to fetch and display product details, add orders, and track inventory.

3. Third-Party APIs:

Payment API (eg, Stripe or PayPal). Handles payment transactions when users make purchases

Shipment API (eg. ShipEngine, AfterShip): Tracks the shipment status of orders, Including real-time updates on delivery.

Email API (eg. SendGrid): Sends emall

notifications to customers for order confirmations, shipping updates, etc.

4. Database (Sanity):

Sanity CMS also functions as a database for storing the products, orders, and other relevant data. It is tightly integrated with the frontend and third-party services.

Conclusion:

This architecture provides a clear and scalable way to build a highly functional e-commerce platform using modern technologies like **Next.js, Sanity CMS,** and **third-party** services such as Stripe and **ShipEngine**. By integrating these components, we ensure a smooth experience for both the frontend user and the backend system, with seamless data flow and real-time updates.