Planning Technical Foundation

Introduction:

This document outlined the technical plan for development an E-commerce Marketplace to empower small business and individuals by providing a platform to sell their products online .

Technical Architecture:

1. Frontend(Next.js) :

- Client-side rendering for speedand responsiveness.
- server-side rendering for product page and SEO preloading.
- Intergration with sanity CMS for dynamic content.

2. Database (DB):

- Collection products,oders,customers,delivery and user authentication.
- NoSQLdatabase to manage flexible and scalable data stuctures.

3. CMS (Sanity):

- Manages dynamic content like banners,featured products, and blog posts.

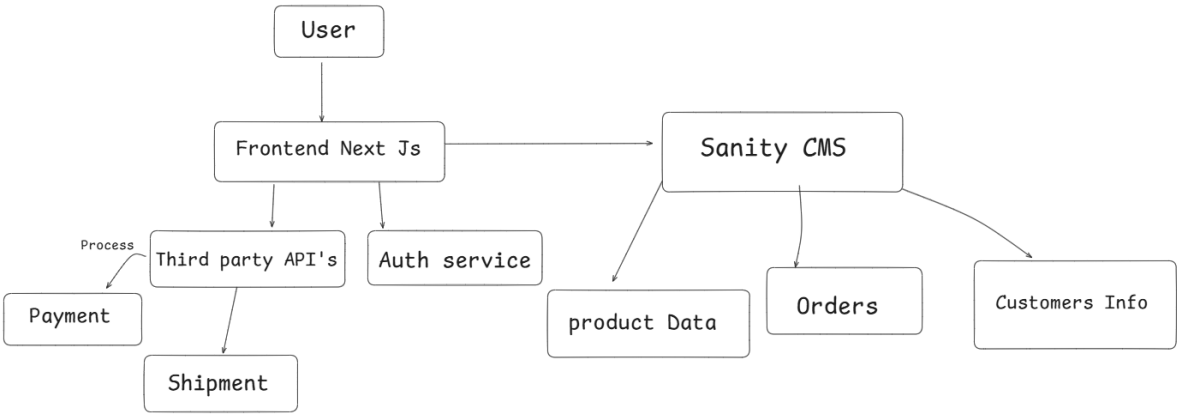
4. Order Tracking (ShipEngine):

- Managed shipment and delivery updates.

5. Authentication (MongoDB):

- Passwords encrypted with hashing algorithem (e.g:bcrypt).
- MongoDB stores user credebtials securely.

System Architecture Overview



Marketplace Technical Foundation

API's End Points

Endpoints	Methods	Description
products	Get	Fetches all Products
products/ID	Get	Fetches a specific Product
Orders	Post	Create a new Order
Orders/ID	Get	Fetches a specific order

Shipment/Tracking/ID	Get	Tracks shipment status

Sanity CMS Schemas

Product Schema Example:

```
export default {
  name: 'product',  // Schema ka naam
  type: 'document',  // Schema ka type
  fields: [
    { name: 'name', type: 'string', title: 'Product Name' },  //
Product ka naam
    { name: 'price', type: 'number', title: 'Price' },  // Product ki
price
    { name: 'stock', type: 'number', title: 'Stock Level' },  // Stock
ki level
  ],
};
```

Order Schema Example:

```
export default {
  name: 'order',  // Schema ka naam
  type: 'document',  // Schema ka type
  fields: [
    { name: 'customerId', type: 'string', title: 'Customer ID' },  //
Customer ka ID
    { name: 'productIds', type: 'array', title: 'Product IDs' },  //
Product IDs ka array
    { name: 'totalPrice', type: 'number', title: 'Total Price' },  //
Total price of the order
    { name: 'status', type: 'string', title: 'Order Status' },  //
Order ka status (e.g., pending, completed)
```

```
  ],
};
```

Customer Schema Example:

```
export const customer = {
  name: 'customer',
  type: 'document',
  fields: [
    { name: 'name', type: 'string', title: 'Customer name' },
    { name: 'email', type: 'string', title: 'email' },
    { name: 'address', type: 'text', title: 'Address' },
    { name: 'phone', type: 'string', title: 'Phone Number' },
  ],
};
```

System Components and Workflow:

1. **User Signup/Login:**

a. **Input**: User credentials (email, password).

b. **Database**: MongoDB for storing user data securely with hashed passwords.

c. **API Endpoint**: POST /register, POST /login, and GET /verify-route for handling user authentication and verification.

d. **Outcome**: JWT token issued for session management.

2. Content Management (Sanity CMS):

a. **Admin Role**: Manages product listings, banners, and blog content.

b.**API Integration:** GROQ Qeries to fetch content dynamically for frontend.

c. **Outcome**: Content stored and updated in Sanity is rendered seamlessly on the Next.js frontend.

3. **Product Browsing and Checkout:**

a. **Frontend**: Next.js provides server-side rendering for product pages.

b. **Database**: MongoDB stores product details (name, price, stock, description, sizes, etc.).

c. **API Endpoint:** GET /products for listing, GET /products/:id for details, and POST /products to add products (admin/seller role only).

d. **Outcome**: Users browse, add products to cart, and proceed to checkout.

4. **Order Management:**

a. **Database**: MongoDB stores order data (customer ID, product ID, quantity, status).
b. **API Endpoint:** POST /orders to create orders (status defaults to "Pending").

c. **Outcome**: Order information processed and stored for tracking. Note: Orders cannot be edited once created.

5. **Shipment Tracking (ShipEngine):**

a. **Integration**: ShipEngine API for real-time shipment tracking.

b. **API Endpoint**: GET /shipments/:orderId to fetch delivery status.

c. **Outcome**: Users receive real-time updates on their order delivery.

6. **Payment Processing (Stripe, Jazz Cash, EasyPaisa, Kuickpay):**

a. **Integration**: Secure payment processing with multiple gateways.

b. **API Endpoint**: Payment-related endpoints for handling transactions, including Cash on Delivery (COD) option.

c. **Outcome**: Orders processed only after successful payment confirmation or COD selection.