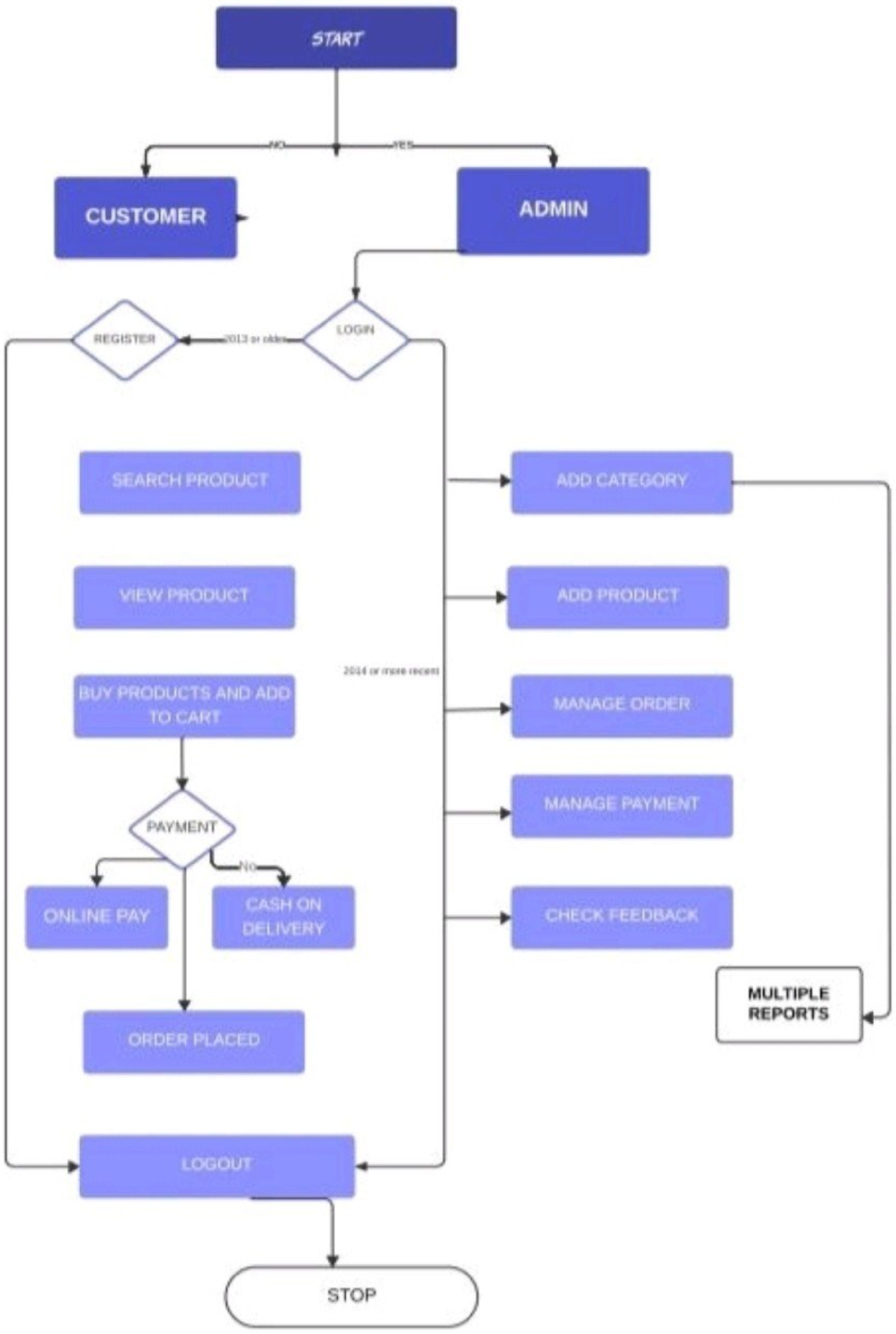
FRONTEND REQUIREMENTS

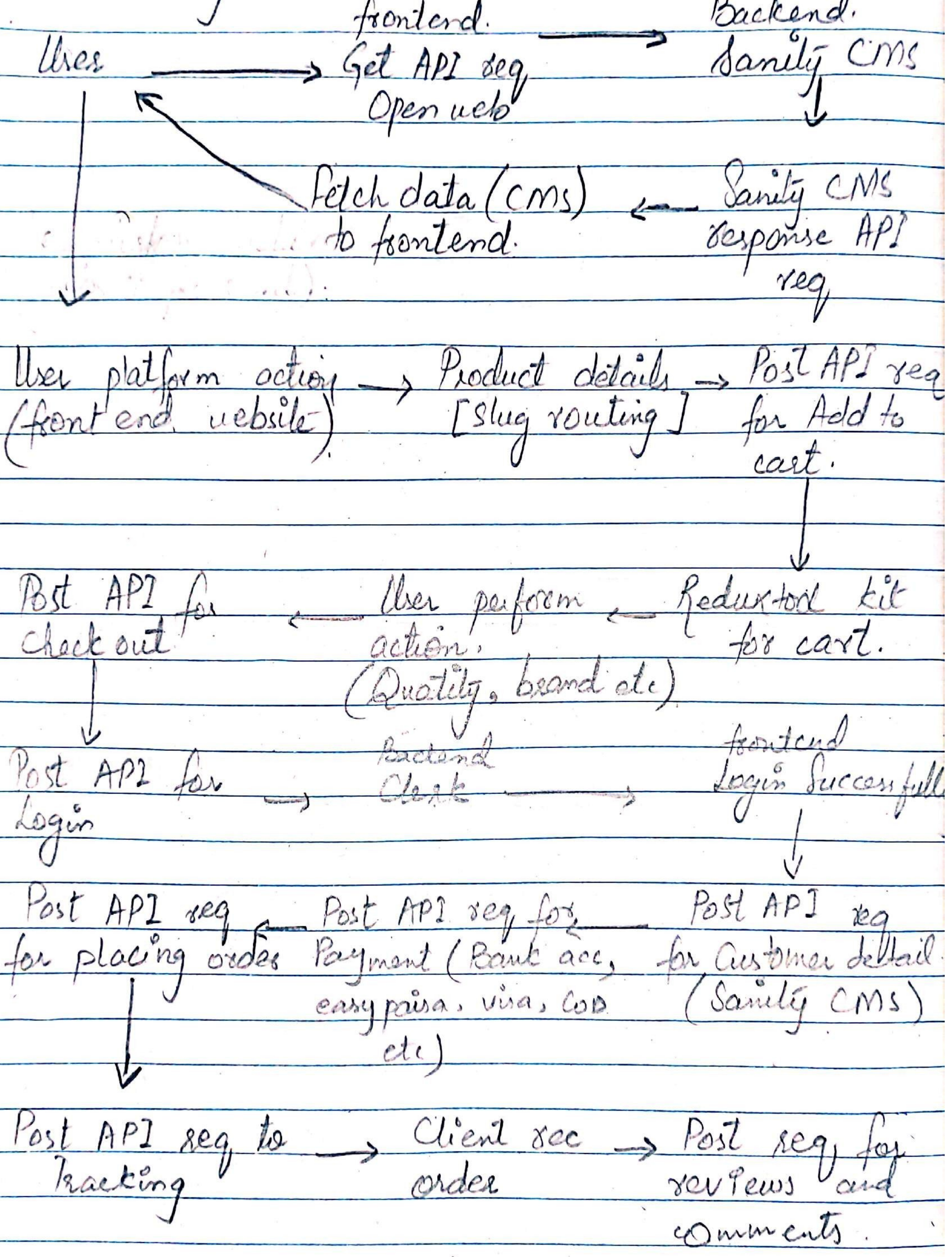
[illegible]



16-Jan-2025

Hackathon #3 Day 2

Planning the Technical Foundation.


Uses
frontend. Get API req Open web
Backend. Sanity CMS
Sanity CMS response API req
Fetch data (CMS) to frontend.
User platform action (front end website)
Product details [Slug routing]
Post API req for Add to cart.
Redux-tool kit for cart.
User perform action. (Quatity, brand etc)
Post API for check out
Post API for Login
Backend Clerk
frontend Login Successfull
Post API req for Customer detail (Sanity CMS)
Post API req for Payment (Bank acc, easypaisa, visa, COD etc)
Post API req for placing order
Post API req to Tracking
Client rec order
Post req for reviews and comments.

1. User Signup:

The user visits the website and fills out a signup form.

The data (e.g., name, email, password) is sent to the backend API to be saved in a database.

2. Data Storage:

The backend processes the data from the form, validates it, and stores it in a database like MongoDB, PostgreSQL, etc.

3. Product Management (Sanity CMS):

Admins use Sanity CMS to manage products (add/edit/delete).

The backend fetches product data from Sanity CMS via APIs and provides it to the frontend for display.

4. Order Process:

The user selects products, adds them to their cart, and places an order.

Payment APIs (e.g., Stripe) are used to handle transactions, and order details are stored in the database.

Step 2: API Designs

APIs define how different parts of the system communicate. I'll provide the following API documentation:

1. Signup API

Endpoint: POST /api/signup

Request Body:

```
{
 "name": "John Doe",
 "email": "john@example.com",
 "password": "password123"
}
```

Response:

```
{
 "message": "User created successfully",
 "userId": "12345"
}
```

2. Product Fetch API

Endpoint: GET /api/products

Response:

```
[
 {
  "id": "prod1",
  "name": "Product 1",
  "price": 100,
  "image": "url-to-image"
 }
]
```

3. Order Placement API

Endpoint: POST /api/order

Request Body:

```
{
 "userId": "123",
 "products": [
  { "id": "prod1", "quantity": 2 }
 ],
 "paymentId": "payment123"
}
```

Response:

```
{
 "message": "Order placed successfully",
 "orderId": "order123"
}
```

Step 3: Database Schemas

Schemas define how data is structured in the database. Here's how I'll define the schemas:

1. User Schema:

Fields:

name: String

email: String (unique)

password: String (hashed)

2. Product Schema:

Fields:

name: String

price: Number

description: String

image: String

3. Order Schema:

Fields:

userId: String (reference to user)

products: Array of objects with productId and quantity.

paymentId: String

Step 4: Sanity CMS Integration

To integrate Sanity CMS, I'll follow these steps:

1. Install Sanity Client:

Install the @sanity/client package in the backend:

```
npm install @sanity/client
```

2. Configure the Client:

Set up the Sanity client in your backend:

```
import sanityClient from '@sanity/client';

const client = sanityClient({
  projectId: 'yourProjectId',
  dataset: 'production',
  apiVersion: '2023-01-01',
  useCdn: true
});
```

3. Fetch Data:

Use a query to fetch data from Sanity:

```
const query = '*[_type == "product"]';
const products = await client.fetch(query);
```