HACKATHON-3 (DAY-2)

Planning the Technical Foundation

1) System Architecture Design: The diagram below shows the interaction & data flow between Front-End, Sanity CMS & Third-party APIs.

Components & their role:

1. User (End-User):

- The person using the system, interacting with the website or application.
- Sends requests to the frontend (Next.js) and receives data in response.

2. Frontend (Next.js App):

- This is the presentation layer (UI) of the system.
- Takes input from the user and fetches data from Sanity CMS or third-party APIs to display it.
- Requests data from the backend using REST APIs or GraphQL queries.

3. Sanity CMS:

- This is your backend CMS that stores and manages content.
- The Next.js frontend fetches data like products from Sanity CMS via API calls.

4. Database (Connected to Sanity CMS):

- Sanity CMS internally stores content in its database.
- If your client wants to use a custom database (e.g., MySQL, PostgreSQL, MongoDB) instead of Sanity CMS, it can be added here.

5. Third-Party APIs:

- These are external services that provide additional data to your application.
- Examples include Payment Gateways such as Stripe, PayPal or any other services.
- The Next.js frontend fetches data from these APIs and displays it to the user.

Data Flow:

1. User → Frontend (Next.js App): The user's request is received by the frontend.

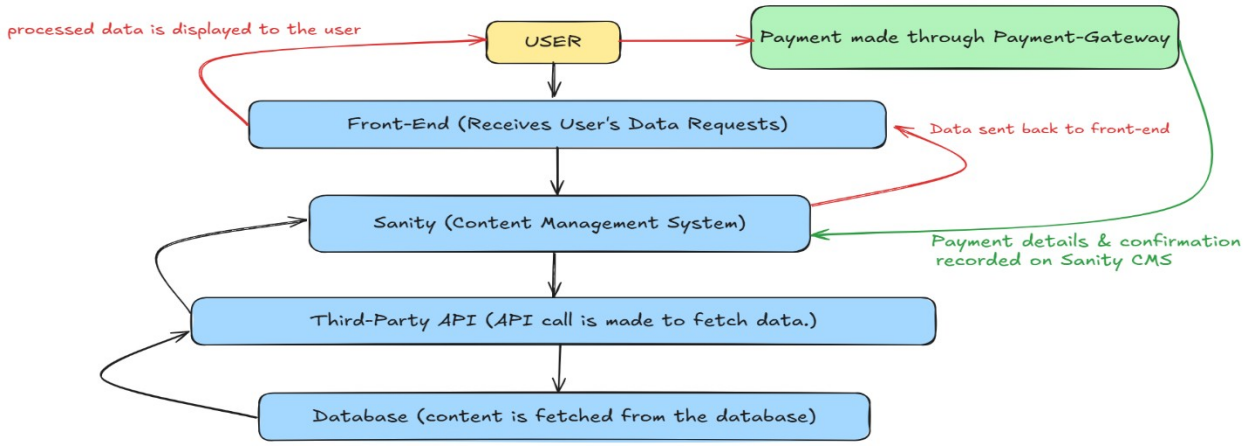
2. Frontend → Sanity CMS / Third-Party API: An API call is made to fetch data.

3. Sanity CMS → Database: The content is fetched from the database.

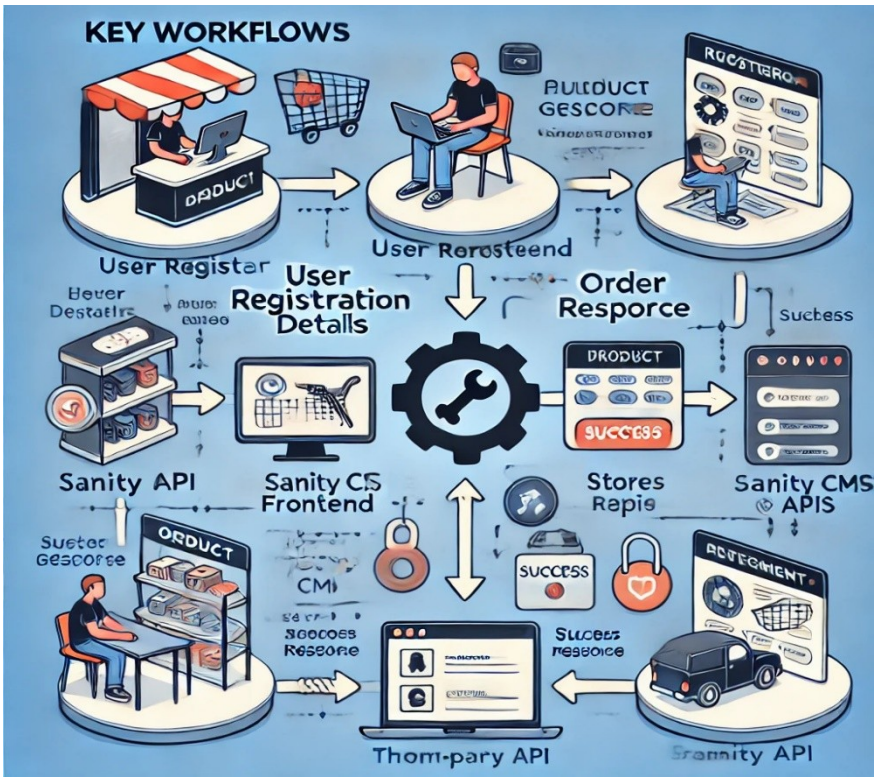
4. Sanity CMS / Third-Party API → Frontend: The data is sent back to the frontend.

5. Frontend → User: The processed data is displayed to the user.

FLOWCHART DIAGRAM:



2) Key Workflows: The diagram below explains the essential workflows of the system, detailing the step-by-step interactions between the Next.js frontend, Sanity CMS, and third-party APIs.



User Registration Workflow

1. User Fills Registration Form (Frontend - Next.js):

- The user enters their credentials (name, email, password).
- The Next.js frontend validates the form.

2. Frontend Sends Data to Authentication API (Backend Service):

- The registration request is sent to an authentication service or backend API.

3. Authentication API Validates and Stores Data (Database):

- The system checks if the email already exists.
- If everything is valid, the user's data is securely stored in the database.

4. Success Response Sent Back to Frontend:

- A success message or confirmation email is sent to the user.
- The user can now log in and use the system.

Product Browsing Workflow

1. User Opens Product Page (Frontend - Next.js):

- The user opens the product page on the website or mobile app.

2. Frontend Fetches Data from Sanity CMS:

- The frontend sends a request to the Sanity CMS API to retrieve the latest product data.
- For an E-Commerce store, product details such as name, description, price, and images are fetched from Sanity CMS.

3. Sanity CMS Retrieves Data from Database:

- Sanity CMS fetches the relevant product data from its database.
- The CMS ensures the data sent to the frontend is accurate and up-to-date.

4. Frontend Displays Product Data to User:

- Once the response is received from Sanity CMS, the Next.js frontend dynamically displays the product data.
- Products are shown to the user with options for filtering or categorizing them.

Order Placement Workflow

1. User Adds Product to Cart (Frontend - Next.js):

- The user adds a product to the cart and clicks the checkout button.

2. Frontend Sends Order Details to Payment Gateway (Third-Party API, e.g., Stripe/PayPal):

- During checkout, the frontend sends a request to the payment gateway API (e.g., Stripe, PayPal, Razorpay).
- The payment gateway collects the user's card details & other credentials.

3. Payment Gateway Processes Payment and Returns Confirmation:

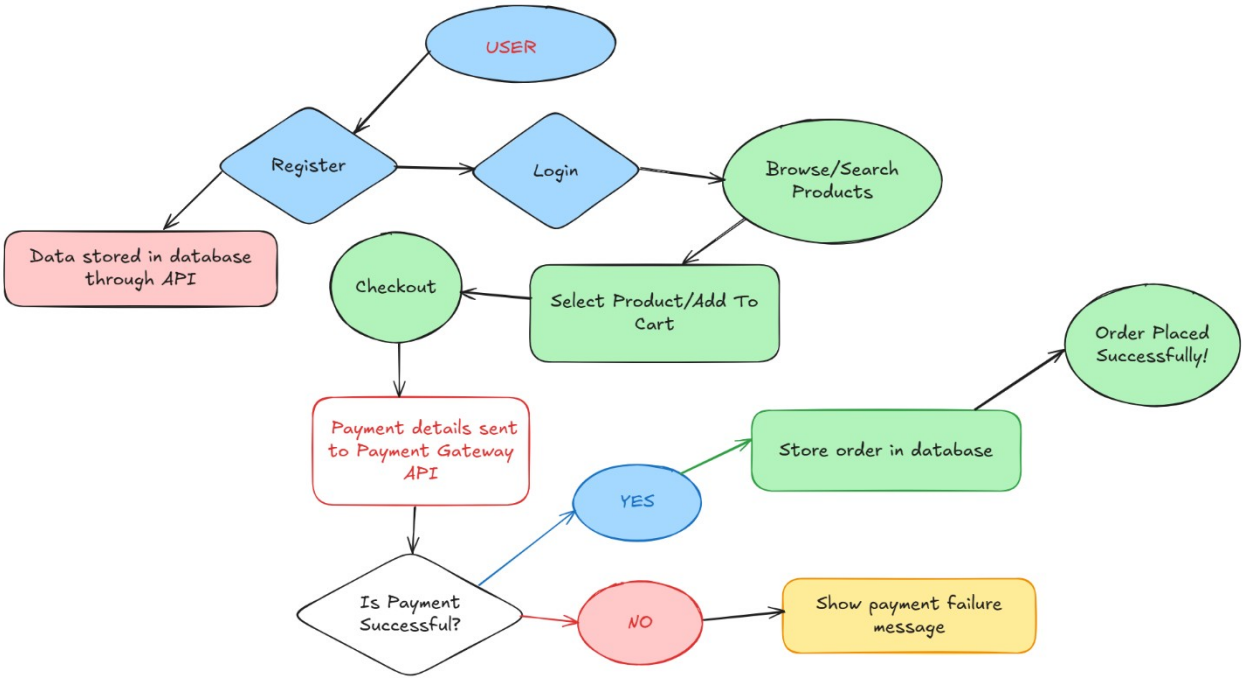
- The payment gateway verifies the transaction and sends a success or failure response to the backend.
- If the payment fails, the user is shown an error message.

4. Order is Stored in Database & Confirmation Sent to User:

- If the payment is successful, the order details are securely stored in the database.
- The user receives a confirmation message or email.

- The order status is updated in the admin panel as well.

FLOWCHART DIAGRAM:



3) API Requirement:

User Registration API

Endpoint	Method	Payload	Response
/api/register	POST	{ "name": "string", "email": "string", "password": "string" }	{ "status": "success", "userId": "123" }
/api/login	POST	{ "email": "string", "password": "string" }	{ "status": "success", "token": "JWT-Token" }

Product Browsing API

Endpoint	Method	Payload	Response
/api/products	GET	None	[{ "id": "1", "name": "Product 1", "price": "100" }, { ... }]
/api/products/:id	GET	None	{ "id": "1", "name": "Product 1", "details": "Description" }

Order Placement API

Endpoint	Method	Payload	Response
/api/orders	POST	{ "userId": "123", "products": [{ "id": "1", "quantity": 2 }], "paymentInfo": { "method": "card", "details": "..." } }	{ "status": "success", "orderId": "ABC123" }

Payment Gateway API (Third-party)

Endpoint	Method	Payload	Response
/api/payments	POST	{ "amount": "100", "method": "card", "details": "Card Info" }	{ "status": "success", "transactionId": "XYZ789" }

Shipment API

Endpoint	Method	Payload	Response
/api/shipment	POST	{ "orderId": "ABC123", "address": { "line1": "string", "city": "string", "zip": "string" }, "deliveryMethod": "standard" }	{ "status": "success", "trackingId": "SHIP123" }

4) Sanity Schema Design:

Product Schema

```
export default {
  name: "product",
  title: "Product",
  type: "document",
  fields: [
    { name: "name", title: "Name", type: "string" },
    { name: "price", title: "Price", type: "number" },
    { name: "description", title: "Description", type: "text" },
    { name: "image", title: "Image", type: "image" },
    { name: "category", title: "Category", type: "string" }
  ]
};
```

Order Schema

```
export default {
  name: "order",
  title: "Order",
  type: "document",
  fields: [
    { name: "userId", title: "User ID", type: "string" },
    { name: "products", title: "Products", type: "array", of: [{ type: "reference", to:
[{ type: "product" }] }] },
```

```
    { name: "totalAmount", title: "Total Amount", type: "number" },
    { name: "status", title: "Status", type: "string" }
  ]
};
```

User Schema

```
export default {
  name: "user",
  title: "User",
  type: "document",
  fields: [
    { name: "name", title: "Name", type: "string" },
    { name: "email", title: "Email", type: "string" },
    { name: "password", title: "Password", type: "string" },
    { name: "orders", title: "Orders", type: "array", of: [{ type: "reference", to: [{ type: "order" }] }] }
  ]
};
```