Marketplace Hackathon Technical Plan

1.Technical Requirements:

• Front-End Requirements:

Front-End is built on Next.js with user friendly interface styled with tailwind css and shadcn ui components library to make pages such as home, about, contact us, teams, pricing, shop and blog.

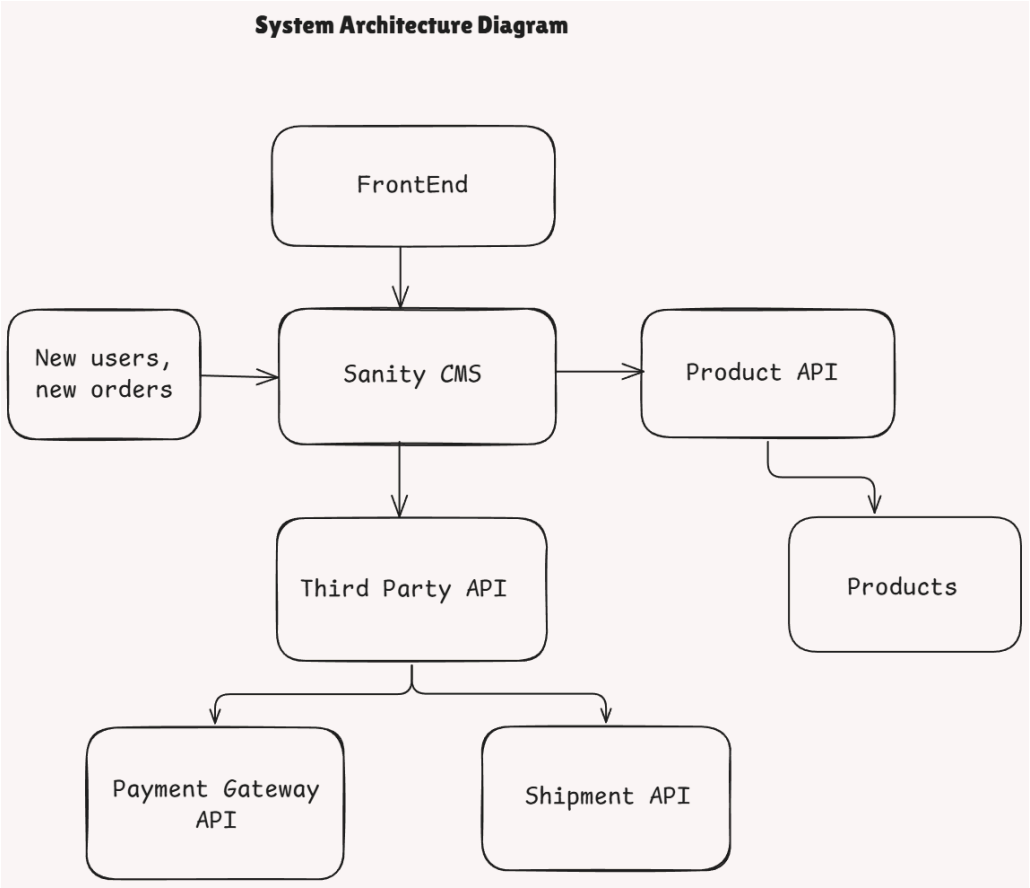
• Sanity CMS as Backend:

Sanity is working as backend and is used to manage products data, customer/user data, payment data, shipping tracking data and order records. We designed schemas in sanity to align with business goals of day 1.

• Third Party APIs:

Used third party APIs for shipment tracking and payment gateways for secure and easy access payments.

2.System Architecture Design:



How it works?

1. When a user visits the marketplace's front-end, they are presented with a range of product listings to browse through. These listings are dynamically loaded by sending requests to the Product Data API, which is powered by Sanity CMS. The API efficiently fetches product details and displays them seamlessly on the website, ensuring a smooth user experience.
2. The marketplace frontend is designed to communicate with the Product Data API through Sanity CMS whenever additional information or updates about products are required. This allows users to explore detailed descriptions, pricing, and availability, all updated in real time to reflect the latest data from Sanity CMS.
3. Once a user finalizes their selection and places an order, the frontend sends all relevant order details to Sanity CMS via an API request. Sanity CMS serves as a centralized system where all order data is securely recorded and stored for further processing and tracking.
4. After the order is logged, the system integrates with a third-party shipping API to retrieve real-time tracking information. This allows the user to monitor the shipment's progress and stay updated on its estimated delivery status through the marketplace interface.
5. To complete the transaction securely, the payment details provided by the user are processed through a trusted Payment Gateway. Upon successful payment, a confirmation message is sent back to the user, and the payment details are recorded in Sanity CMS to ensure a complete and transparent order history.

3. Key WorkFlow:

1. Account Creation Process:

- Users create an account by providing their details.
- The data is securely stored in Sanity CMS.
- A confirmation message is sent to the user upon successful registration.

2. Exploring Product Categories:

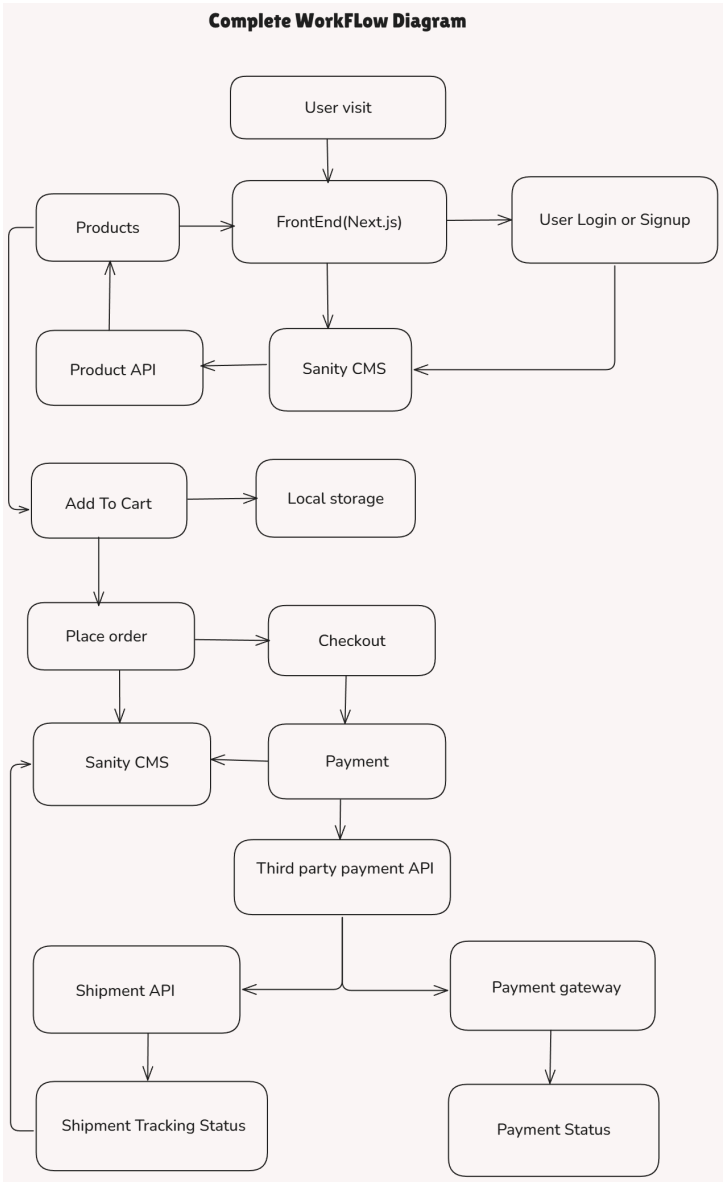
- Users can browse through various product categories.
- Product information is dynamically retrieved using the Sanity API.
- The fetched data is displayed seamlessly on the frontend for users to explore.

3. Placing an Order:

- Users select items and add them to their shopping cart.
- They proceed to the checkout process to finalize their purchase.
- Order details are recorded in Sanity CMS for processing.

4. Tracking Shipments

- Shipment updates are obtained through integration with a third-party API.
- Real-time tracking information is displayed to the user, ensuring transparency.


Complete WorkFLow Diagram
User visit
Products
FrontEnd(Next.js)
User Login or Signup
Product API
Sanity CMS
Add To Cart
Local storage
Place order
Checkout
Sanity CMS
Payment
Third party payment API
Shipment API
Payment gateway
Shipment Tracking Status
Payment Status

4. API End-Points:

Resource	Method & Endpoint	What It Does	Response
Products	GET /api/products	Retrieves a list of all products.	[{id, name, price, stock, category, image}])
	GET /api/products/:id	Retrieves details of a specific product by its ID.	{id, name, description, price, stock, category, image}
	POST /api/products	Creates a new product in the database.	{message: 'Product created', product}
	PUT /api/products/:id	Updates details of an existing product (e.g., stock or price).	{message: 'Product updated', product}
	DELETE /api/products/:id	Deletes a product by its ID.	(e.g., {message: 'Product deleted', id})
Users	GET /api/users	Retrieves a list of all registered users.	[{id, name, email, address, phone}]
	GET /api/users/:id	Retrieves details of a specific user by their ID.	{id, name, email, address, phone}
	POST /api/users	Registers a new user in the system.	,{message: 'User registered', user}
	PUT /api/users/:id	Updates a user's details (e.g., address or phone number).	{message: 'User updated', user}
	DELETE /api/users/:id	Deletes a user by their ID.	{message: 'User deleted', id}
Orders	GET /api/orders	Retrieves a list of all orders.	[{id, userId, products, totalAmount, status}]
	GET /api/orders/:id	Retrieves details of a specific order by its ID.	{id, userId, products, totalAmount, status, paymentId, shipmentId}
	POST /api/orders	Creates a new order in the system.	{message: 'Order placed', order}
	PUT /api/orders/:id	Updates the status or details of an existing order.	{message: 'Order updated', order}
	DELETE /api/orders/:id	Deletes an order by its ID.	{message: 'Order deleted', id}
Payments	GET /api/payments	Retrieves a list of all payments.	[{id, orderId, userId, amount, method, status, transactionId}]
	GET /api/payments/:id	Retrieves details of a specific payment by its ID.	{id, orderId, userId, amount, method, status, transactionId}.
	POST /api/payments	Records a new payment in the system.	{message: 'Payment recorded', payment}
	PUT /api/payments/:id	Updates the status of an existing payment (e.g., pending to success).	{message: 'Payment updated', payment}
	DELETE /api/payments/:id	Deletes a payment by its ID.	{message: 'Payment deleted', id}
Shipments	GET /api/shipments	Retrieves a list of all shipments.	[{id, orderId, userId, status, trackingNumber, carrier}]
	GET /api/shipments/:id	Retrieves details of a specific shipment by its ID.	{id, orderId, userId, status, trackingNumber, carrier}
	POST /api/shipments	Creates a new shipment record.	{message: 'Shipment created', shipment}
	PUT /api/shipments/:id	Updates the status of an existing shipment (e.g., in_transit to delivered).	{message: 'Shipment updated', shipment}
	DELETE /api/shipments/:id	Deletes a shipment record by its ID.	{message: 'Shipment deleted', id}

5. Sanity Schema Example:

- **Product Schema**

```
export const productSchema = {
    name: "product",
    title: "Product",
    type: "document",
    fields: [
      { name: "id", title: "Product ID", type: "string" }, // Unique identifier for product
      { name: "name", title: "Name", type: "string" },
      { name: "description", title: "Description", type: "text" },
      { name: "price", title: "Price", type: "number" },
      { name: "stock", title: "Stock", type: "number" },
      { name: "category", title: "Category", type: "string" },
      { name: "image", title: "Image", type: "image", options: { hotspot: true } },
    ],
  };
```

- **Order Schema**

```
export const orderSchema = {
    name: "order",
    title: "Order",
    type: "document",
    fields: [
      { name: "orderNumber", title: "Order Number", type: "string" },
      { name: "user", title: "User/Customer", type: "reference", to: [{ type: "user" }] },
      { 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", options: { list: ["Pending", "Processing", "Shipped",
      { name: "orderDate", title: "Order Date", type: "datetime" },
    ],
  };
```

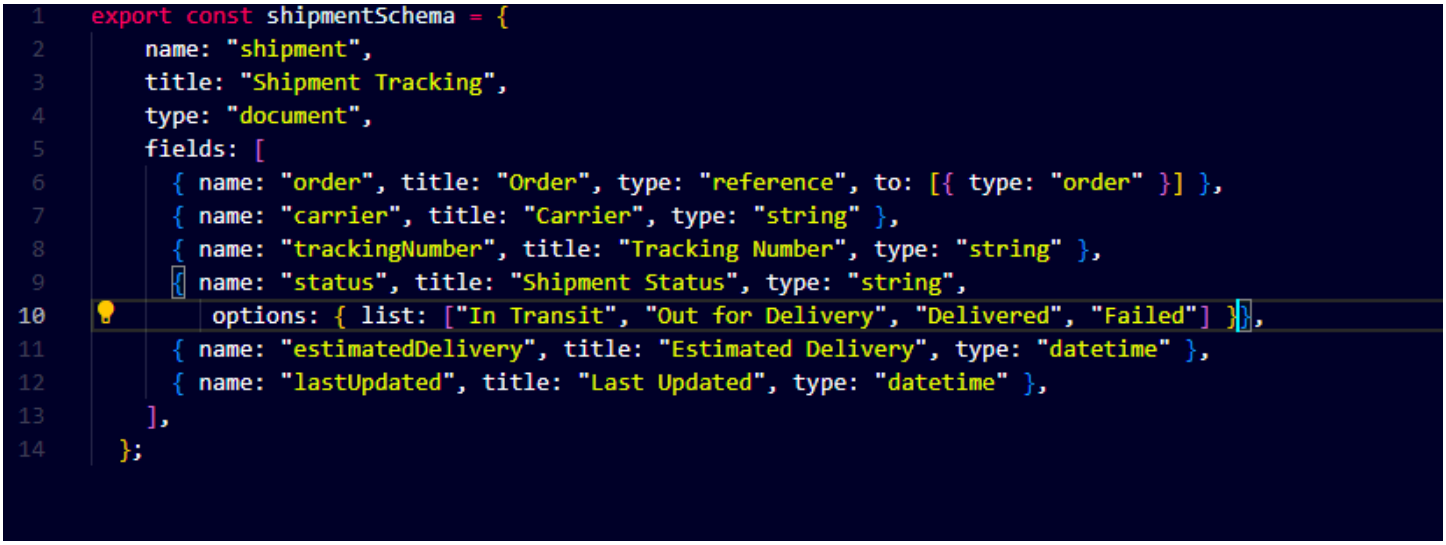
- **User Schema**

```
export const userSchema = {
    name: "user",
    title: "User/Customer",
    type: "document",
    fields: [
      { name: "name", title: "Name", type: "string" },
      { name: "email", title: "Email", type: "string" },
      { name: "password", title: "Password", type: "string" }, // Store hashed passwords only.
      { name: "phone", title: "Phone", type: "string" },
      { name: "address", title: "Address", type: "text" },
      { name: "orders", title: "Orders", type: "array", of: [{ type: "reference", to: [{ type: "order" }] }] },
    ],
  };
```

- **Payment Schema**

```
export const paymentSchema = {
    name: "payment",
    title: "Payment",
    type: "document",
    fields: [
      { name: "order", title: "Order", type: "reference", to: [{ type: "order" }] },
      { name: "paymentMethod", title: "Payment Method", type: "string", options: { list: ["Credit Card", "PayPa
      { name: "paymentStatus", title: "Payment Status", type: "string", options: { list: ["Pending", "Completed"
      { name: "transactionId", title: "Transaction ID", type: "string" },
      { name: "amount", title: "Amount", type: "number" },
      { name: "paymentDate", title: "Payment Date", type: "datetime" },
    ],
  };
```

- **Shipment Schema**



```
export const shipmentSchema = {
    name: "shipment",
    title: "Shipment Tracking",
    type: "document",
    fields: [
      { name: "order", title: "Order", type: "reference", to: [{ type: "order" }] },
      { name: "carrier", title: "Carrier", type: "string" },
      { name: "trackingNumber", title: "Tracking Number", type: "string" },
      { name: "status", title: "Shipment Status", type: "string",
        options: { list: ["In Transit", "Out for Delivery", "Delivered", "Failed"] } },
      { name: "estimatedDelivery", title: "Estimated Delivery", type: "datetime" },
      { name: "lastUpdated", title: "Last Updated", type: "datetime" },
    ],
  };
```