Day 2 Activities: Transitioning to Technical Planning

1. **Define Technical Requirements**

   The first step is to translate your business goals into clear technical requirements. For each feature identified on Day 1, outline the following:

   Frontend Requirements:

   - Develop a **user-friendly interface** for browsing products.
   - Ensure a **responsive design** for seamless use on both mobile and desktop devices.
   - Create essential pages:

     Home: A welcoming overview of the website with key highlights.

     Product Listing: Displays a catalog of available products.

     Product Details: Detailed information about a specific product.

     Cart: A summary of selected items for purchase.

     Checkout: Secure and streamlined purchase process.

     Order Confirmation: Acknowledgment of completed orders.

Sanity CMS as Backend

- Utilize **Sanity CMS** to efficiently manage:
  - **Product Data**: Store and organize product details like names, prices, descriptions, images, and categories.

    - **Customer Details**: Keep track of user profiles, shipping addresses, and contact information.
    - **Order Records**: Maintain a database of order histories, including items purchased, payment status, and delivery details.
- Focus Areas:
    - Design **schemas** in Sanity to reflect the business requirements defined on Day 1.
        - Example Schemas:
            - **Product Schema**: Includes fields for title, price, description, images, and tags.
            - **Order Schema**: Tracks order ID, products purchased, customer info, and status.
            - **Customer Schema**: Manages user profiles, contact details, and order history.
- **Why Sanity?**
    - Flexible real-time data management.
    - Easy integration with your frontend (via APIs).
    - Scalable to accommodate future marketplace expansion.

Third-Party APIs

- **Purpose:**
  Integrate third-party APIs to enhance functionality and provide essential backend services for your marketplace.
- **Key Integrations:**

  Shipment Tracking API:

  Enable users to track the status of their orders in real-time.

  Provide estimated delivery times and updates on package locations.

  Payment Gateway API:

  Securely process online payments (e.g., Stripe, PayPal).

Support multiple payment methods (credit cards, debit cards, wallets, etc.).

Other Required APIs:

Currency conversion API (for international marketplaces).

Email or SMS notification APIs for order updates and communication.

Implementation Focus:

Ensure APIs provide **real-time, reliable data** to the frontend for seamless user interaction.

Integrate APIs with proper **error handling** to manage potential failures.

Test API responses to ensure compatibility with the frontend design and logic.

2. Design System Architecture

The goal is to create a high-level diagram to visualize how all components of your system interact. Use tools like **Lucidchart**, **Figma**, or **Excalidraw**, or even a simple pen and paper to sketch the architecture.

Example Architecture Components:

Frontend (Next.js):

User interface where customers interact with the website (browsing, ordering, etc.).

Sanity CMS:

Backend database for managing product data, customer details, and order records.

Third-Party APIs:

APIs for shipment tracking, payment gateways, and additional backend services.

Architecture Diagram Workflow:

Frontend (Next.js):

Sends requests to the backend to fetch data or trigger actions (e.g., display product details, process an order).

Sanity CMS:

Acts as the central hub for product, customer, and order data.

Provides an API that the frontend queries to display updated information.

Product Data API:

Retrieves product information from Sanity CMS and passes it to the frontend.

Third-Party API Integrations:

Shipment Tracking API: Fetches real-time delivery updates for customer orders.

Payment Gateway API: Handles payment processing for checkout.

High-Level Data Flow Example:

User Interaction:

A user browses products or places an order on the Next.js frontend.

Data Request to Sanity CMS:

Frontend sends a request to Sanity's API for product data or customer details.

Product Data Retrieval:

Sanity CMS responds with the requested product data, which is displayed on the frontend.

Payment Process:

During checkout, the frontend communicates with the Payment Gateway API to process payments securely.

Order Confirmation & Shipment Tracking:

The order details are sent to the Sanity CMS and stored as a new record.

The Shipment Tracking API fetches the delivery status and updates the customer via the frontend

Data Flow Example:

User Visits the Marketplace Frontend:

A user opens the marketplace website built with **Next.js**.

The **Next.js frontend** dynamically renders the home page or product listing page using server-side rendering (SSR) or static site generation (SSG), depending on the setup.

Frontend Requests Product Data:

When the user navigates to a specific section (e.g., product listings), the **frontend sends a request** to the **Product Data API**, which is powered by **Sanity CMS**.

The Product Data API fetches information such as:

Product names, prices, descriptions, images, and categories.

The data is then returned in **JSON format** to the frontend.

The frontend dynamically displays the fetched data using React components.

3. User Places an Order:

Frontend Order Submission:

When the user places an order, the **Next.js frontend** collects the order details, including:

Selected products (name, quantity, price).

Customer information (name, address, contact details).

Payment status (success, pending, or failed).

API Request to Sanity CMS:

The frontend sends an **API POST request** to the **Sanity CMS Order Schema**, where the order details are securely stored in the database.

Example order details stored in Sanity CMS:

- **Order ID**: A unique identifier for the order.
- **Customer Information**: Name, email, and shipping address.
- **Ordered Items**: List of products purchased, including quantity and total price.
- **Order Status**: Pending, confirmed, or shipped.

Response to Frontend:

Sanity CMS sends a response confirming that the order has been successfully recorded.

The frontend displays an **Order Confirmation Page** to the user with a summary of the order.

---

4. Shipment Tracking via Third-Party API:

Trigger for Shipment Tracking:

After the order is placed, the shipment tracking process begins.

A **tracking ID** is generated either by your system or provided by the shipping service integrated with the marketplace.

Fetching Tracking Information:

The **Next.js frontend** makes a request to the **Shipment Tracking API** (e.g., FedEx, DHL, or a local service) using the tracking ID.

The API returns real-time shipment data, including:

- Current status of the shipment (e.g., "In Transit," "Delivered").

Location updates.

Estimated delivery date and time.

Display on Frontend:

The frontend dynamically updates the **Order Status Page** with shipment details fetched from the API.

Example information displayed:

Order Status: In Transit, Delivered, or Out for Delivery.

Tracking Timeline: Date and time for each major update (e.g., dispatched, reached hub, delivered).

Estimated Delivery: A countdown or specific delivery date.

Real-Time Updates:

Using tools like **webhooks** or **polling**, the frontend can refresh shipment status periodically to ensure users have up-to-date information.

5. Payment Processing via Payment Gateway:

Secure Payment Initiation:

When the user proceeds to checkout, the **Next.js frontend** collects payment details (e.g., card information, digital wallet, etc.).

These details are securely sent to the **Payment Gateway API** (e.g., Stripe, PayPal) using an encrypted connection (e.g., HTTPS).

Payment Gateway Processing:

The **Payment Gateway** validates the payment information and processes the transaction.

It returns a response with the following details:

Transaction Status: Success, Pending, or Failed.

Transaction ID: A unique identifier for the payment.

Payment Amount: Total amount charged.

Confirmation to Frontend:

The **Payment Gateway** sends the payment status back to the frontend.

The frontend dynamically updates the **Order Confirmation Page** based on the response:

If **successful**, the page displays an order success message and summary.

If **failed**, the user is notified to retry or check their payment details.

Recording in Sanity CMS:

After a successful payment, the frontend sends an API request to **Sanity CMS** to update the order record with:

Transaction ID: To link the order with the payment.

Payment Status: Mark the order as paid.

Order Completion Timestamp: The date and time when the payment was successful.

Example Data Flow:

Frontend: Sends payment details to the Payment Gateway API.

Payment Gateway: Processes payment and responds with success/failure.

Sanity CMS: Stores the updated order details, including payment confirmation.

1. User Registration

Sign-Up Process:

The user fills out a registration form on the **Next.js frontend** (e.g., name, email, password).

The frontend sends the user’s details securely to the **Sanity CMS** via an API request.

Data Storage in Sanity:

The user's information is stored in the **User Schema** in Sanity CMS.

The schema includes fields like:

User ID (unique identifier).

Name, email, and hashed password (for security).

Timestamps for account creation.

Confirmation to the User:

After successful registration, the system sends a confirmation message (e.g., a welcome email or a success notification on the frontend).

---

2. Product Browsing

User Views Product Categories:

The user navigates to a product category or searches for specific products on the **Next.js frontend.**

Data Fetching via Sanity API:

The frontend sends a request to the **Sanity CMS Product Schema API** to retrieve product data.

Example data fetched:

Product name, price, image URL, description, and category.

Display on the Frontend:

The frontend dynamically renders the products on the **Product Listing Page**, ensuring a responsive and user-friendly experience.

---

3. Order Placement

Adding Items to the Cart:

The user adds desired products to their shopping cart. The cart state is managed on the frontend (using a state management library like Redux or Context API).

Proceeding to Checkout:

At checkout, the frontend collects user details (e.g., shipping address) and sends the **order details** to the **Sanity CMS Order Schema**.

Order Details Saved in Sanity:

The order record includes:

User ID: To link the order to the registered user.

Ordered Products: List of items (name, quantity, price).

Order Status: Set to "Pending" initially.

---

4. Shipment Tracking

Order Status Updates via Third-Party API:

Once the order is shipped, the system generates or retrieves a **tracking ID** from the shipping service.

The frontend sends a request to the **Shipment Tracking API** (e.g., DHL or FedEx) using the tracking ID.

The API responds with real-time updates, such as:

Shipment status (In Transit, Delivered, etc.).

Current location of the package.

Display to the User:

The shipment details are dynamically displayed on the **Order Status Page** or a dedicated **Tracking Page**.

Users can view updates like:

Delivery timeline and expected arrival date.

Current package location.

3. Plan API Requirements

1. General eCommerce Endpoint

Endpoint Name: `/products`

Method: `GET`

Description: This endpoint is used to fetch all available products from the Sanity CMS. It will return a list of products with relevant details that can be displayed on the frontend.

Response Example:

json

CopyEdit

```
[
  {
    "id": 1,
    "name": "Product A",
    "price": 100,
    "stock": 50,
    "image": "https://example.com/images/productA.jpg"
  },
  {
    "id": 2,
    "name": "Product B",
    "price": 120,
    "stock": 30,
    "image": "https://example.com/images/productB.jpg"
  }
]
```

Fields Description:

- `id`: Unique identifier for the product.
- `name`: Name of the product.
- `price`: Price of the product in the currency unit (e.g., USD).
- `stock`: Quantity available in stock.
- `image`: URL to the image of the product.

---

2. Order Management Endpoint

Endpoint Name: `/orders`

Method: `POST`

Description: This endpoint is used to create a new order in Sanity. When a user places an order, the details (including customer information, product details, and payment status) will be sent here for processing and storage.

Payload Example:
json
CopyEdit
```
{
  "customerInfo": {
    "name": "John Doe",
    "email": "johndoe@example.com",
    "address": "123 Main St, City, Country"
  },
  "productDetails": [
    {
      "id": 1,
      "name": "Product A",
      "quantity": 2,
      "price": 100
    },
    {
      "id": 2,
      "name": "Product B",
      "quantity": 1,
      "price": 120
    }
  ],
  "paymentStatus": "Paid"
}
```

Response Example:

json

CopyEdit

```
{
  "orderId": 123,
  "status": "Order Placed",
  "paymentStatus": "Paid",
  "estimatedDelivery": "2025-01-28"
}
```

Fields Description:

- `customerInfo`: Contains the customer's personal information (e.g., name, email, address).
- `productDetails`: Array of products in the order, each containing:
  - `id`: Product ID.
  - `name`: Name of the product.
  - `quantity`: Quantity of the product in the order.
  - `price`: Price of the product.
- `paymentStatus`: Current payment status (e.g., "Paid", "Pending").

`orderId`: Unique identifier for the order.

`status`: Current status of the order (e.g., "Order Placed").

`estimatedDelivery`: Estimated delivery date of the order.

---

3. Shipment Tracking Endpoint

Endpoint Name: `/shipment`

Method: `GET`

Description: This endpoint tracks the status of an order via a third-party shipment tracking API. It allows the user to monitor their order's progress.

Query Parameters Example:
ruby
CopyEdit
```
?orderId=123
```

Response Example:
json
CopyEdit
```
{
  "shipmentId": "SH12345",
  "orderId": 123,
  "status": "In Transit",
  "expectedDelivery": "2025-01-28",
  "currentLocation": "Distribution Center"
}
```

Fields Description:

`shipmentId`: Unique identifier for the shipment.

`orderId`: Unique identifier for the associated order.

`status`: Current shipment status (e.g., "In Transit", "Delivered").

`expectedDelivery`: Estimated delivery date.

`currentLocation`: Current location of the shipment (e.g., "Distribution Center", "On the Way").

---

4. Quick Commerce (Q-Commerce) Endpoint

Endpoint Name: **`/express-delivery-status`**

Method: `GET`

Description: This endpoint fetches real-time delivery updates for perishable items or high-priority orders requiring express delivery. It allows customers to track urgent deliveries.

Query Parameters Example:
ruby
CopyEdit
```
?orderId=456
```

Response Example:
json
CopyEdit
```
{
  "orderId": 456,
  "status": "In Transit",
  "ETA": "15 mins"
}
```

Fields Description:

- `orderId`: Unique identifier for the order.
- `status`: Current status of the express delivery (e.g., "In Transit", "Out for Delivery").
- `ETA`: Estimated time of arrival (e.g., "15 mins").

---

5. Rental eCommerce Example

Endpoint Name: **`/rental-duration`**

Method: `POST`

Description: This endpoint adds rental details for a specific product. It will allow users to rent a product for a specific duration with a deposit.

Payload Example:
json
CopyEdit
```
{
  "productId": 789,
  "duration": "7 days",
  "deposit": 500
}
```

Response Example:
json

CopyEdit

```
{
  "confirmationId": 1011,
  "status": "Success"
}
```

Fields Description:

- `productId`: Unique identifier of the rented product.
- `duration`: Duration of the rental (e.g., "7 days").
- `deposit`: The deposit amount for the rental.

General API Guidelines:

Authentication: Ensure secure authentication is applied to sensitive endpoints (e.g., `/orders`, `/rental-duration`) using methods like API keys or OAuth tokens.

Error Handling: The API should return appropriate HTTP status codes (e.g., `200 OK`, `400 Bad Request`, `404 Not Found`, `500 Internal Server Error`) along with descriptive error messages when needed.

Versioning: It's recommended to version the API (e.g., `/v1/products`) to manage future changes and maintain backward compatibility.

CORS Support: Ensure the API supports CORS (Cross-Origin Resource Sharing) if the frontend will be hosted on a different domain.

Technical Documentation :

System Architecture Overview

The system architecture outlines the components involved in delivering the **Comforty** marketplace, illustrating the flow of data and interactions between the frontend, backend, content management system (CMS), and third-party services.

System Components:

Frontend (React/Next.js):
The frontend is built using Next.js to provide server-side rendering (SSR) for faster load times and better SEO. It handles user interactions, product displays, cart management, and order placement.

Backend (Node.js / Express.js):
The backend, powered by Node.js and Express.js, processes requests from the frontend. It interacts with the CMS and third-party APIs to handle tasks like product fetching, order creation, payment processing, and shipment tracking.

Sanity CMS:
Sanity CMS serves as the headless content management system where product data (e.g., sofa and chair details, prices, images) and user orders are stored. It is flexible and allows for real-time data fetching via its APIs.

Third-party APIs:

Payment Gateway (e.g., Stripe, PayPal): Handles secure payments and transaction processing.

Shipment Tracking API: Tracks the delivery status of orders.

System Flow Diagram:

plaintext
CopyEdit

```
Frontend (React/Next.js)
    |
    |--> API Calls (REST API)
    |
Backend (Node.js/Express)
    |
    |--> Fetch Data (Sanity CMS)
    |
Sanity CMS (Product & Order Management)
    |
    |--> External Integrations
    |      |--> Payment Gateway (e.g., Stripe)
    |      |--> Shipment Tracking API
```

Key Workflows

User Workflow 1: Product Browsing and Display

User Action: A customer browses the product categories and selects a sofa or chair.

Frontend: Sends a GET request to `/products` endpoint to fetch all available products.

Backend: Retrieves product data (ID, name, price, image) from Sanity CMS.

Frontend: Displays products in the user interface.

User Workflow 2: Adding Items to Cart

User Action: Customer clicks on "Add to Cart" for a selected chair or sofa.

Frontend: Adds the chair/sofa's details (ID, name, price) to the cart in the local session.

Frontend: The cart updates in real-time, showing the added products.

User Workflow 3: Order Creation

User Action: Customer proceeds to checkout and enters shipping information.

Frontend: Sends a POST request to `/orders` endpoint, including order details and customer information.

Backend: The backend processes the order, creates an entry in the Sanity CMS, and integrates with the payment gateway.

Payment Gateway: Processes the payment and returns the transaction result.

Frontend: Displays the order confirmation with payment status.

User Workflow 4: Shipment Tracking

User Action: Customer wants to track the status of their order.

Frontend: Sends a GET request to `/shipment?orderId={orderId}` endpoint to track the shipment.

Backend: Fetches shipment status from the Shipment Tracking API.

Frontend: Displays real-time tracking information (e.g., current location, ETA).

Category-Specific Instructions

Q-Commerce: Real-Time Delivery and Inventory Updates

Real-Time Inventory Updates: Use WebSockets or polling to fetch real-time updates on product stock availability.

Express Delivery: Include workflows for quick delivery tracking via the `/express-delivery-status` endpoint to fetch real-time updates.

Example Endpoint for Q-Commerce:

Endpoint: `/express-delivery-status`

Method: GET

Purpose: Fetch real-time delivery status for express deliveries.

Response Example:

json
CopyEdit
```
{
  "orderId": 123,
  "status": "In Transit",
  "ETA": "15 mins"
}
```

General E-Commerce: Product Browsing and Order Management

Product Browsing: Users can view all sofas and chairs and apply filters (e.g., by price, comfort level, or style).

Cart Management: Users can manage items in their cart before proceeding to checkout.

Order Management: Facilitates the order placement process, including shipping details and payment integration.

API Endpoints

Endpoint	Method	Purpose	Response Example
/products	GET	Fetch all available products	[{ "id": 1, "name": "Comfortable Recliner", "price": 150, "stock": 20, "image": "https://example.com/recliner.jpg" }]
/orders	POST	Create a new order	{ "orderId": 123, "status": "Success" }
/shipment	GET	Track order shipment status	{ "shipmentId": "SH12345", "status": "In Transit", "expectedDeliveryDate": "2025-01-30" }

/express-delivery-status	GET	Fetch express delivery updates	{ "orderId": 456, "status": "In Transit", "ETA": "15 mins" }

Detailed API Endpoint Descriptions

/products (GET):

Description: Fetches all product details such as name, price, stock, and image.

Request: `GET /products`

Response Example:

json
CopyEdit

```
[
  {
    "id": 1,
    "name": "Comfortable Recliner",
    "price": 150,
    "stock": 20,
    "image": "https://example.com/recliner.jpg"
  }
]
```

/orders (POST):

Description: Creates a new order for the user.

Request:

json
CopyEdit

```
{
  "customerInfo": {
    "name": "John Doe",
    "address": "123 Street, City, Country"
  },
  "productDetails": [
    {
      "productId": 1,
      "quantity": 2
    }
  ],
  "paymentStatus": "Paid"
}
```

Response Example:

json
CopyEdit

```
{
  "orderId": 123,
  "status": "Success"
}
```

/shipment (GET):

Description: Fetches the current shipment status for a specific order.

Request: `GET /shipment?orderId=123`

Response Example:

json

CopyEdit

```
{
  "shipmentId": "SH12345",
  "status": "In Transit",
  "expectedDeliveryDate": "2025-01-30"
}
```

Data Schema Design

Entities and Relationships

Product Entity:

`id`: Unique identifier for the product (Integer)

`name`: Product name (String)

`price`: Price of the product (Float)

`stock`: Number of units available (Integer)

`image`: Image URL for the product (String)

Order Entity:

`orderId`: Unique order identifier (Integer)

`customerInfo`: Contains customer details (Name, Address)

`productDetails`: List of products ordered, with quantity (Array of objects)

`paymentStatus`: Status of the payment (String)

Technical Roadmap

Phase 1: Setup and Configuration

Implement Sanity CMS for product management.

Setup the backend with Node.js and Express.js.

Phase 2: Frontend Development

Develop product browsing and cart management features.

Integrate with backend APIs to fetch products and manage orders.

Phase 3: Payment Gateway and Order Management

Integrate a payment gateway for secure transactions.

Implement order creation, payment processing, and confirmation.

Phase 4: Shipment Tracking Integration

Integrate shipment tracking API to fetch delivery status.

Provide real-time updates for orders.