**System Architecture Design**

**1. Key Components and Interaction Flow:**

**Frontend (User Interface):**

* **Framework**: Next.js (or React) for building the frontend interface.
* **Pages**: Home, Product Listing, Product Details, Cart, Checkout, Order Confirmation.
* **User Actions**: Browsing products, adding items to the cart, proceeding to checkout, and completing the order.

**Backend (Sanity CMS):**

* **Sanity CMS**: Acts as the Content Management System for managing all product data (e.g., furniture descriptions, prices, stock levels, images).
* **Product Data API**: Retrieves product details from Sanity CMS and displays them dynamically on the frontend.
* **Order Management**: When a user places an order, the order details (user info, products, payment status) are sent to Sanity CMS for storage and tracking.

**Third-Party APIs:**

* **Shipment Tracking API**: Provides real-time updates on the status of an order's shipment, such as "Shipped", "In Transit", and "Delivered". Data is fetched from third-party logistics services like **ShipEngine**, **UPS**, or **AfterShip**.
* **Payment Gateway**: Handles secure payments and processes transactions using services like **Stripe**, **PayPal**. After successful payment, the transaction confirmation is recorded in Sanity CMS.

**System Workflow:**

**Step 1: Product Browsing**

* The user visits the furniture website.
* The frontend (Next.js) sends a request to the **Product Data API** powered by Sanity CMS.
* **Sanity CMS** retrieves the relevant product details (name, price, description, images) and sends it back to the frontend.
* The frontend dynamically displays products to the user.

**Step 2: Add Products to Cart**

* The user adds products to the cart.
* The cart details are temporarily stored on the frontend and include the products the user has selected.

**Step 3: Checkout Process**

* When the user proceeds to checkout, the frontend sends the order details (user’s information, products, and shipping address) to **Sanity CMS** via an API.
* The backend records the order in Sanity CMS and updates the order status.

**Step 4: Payment Processing**

* The frontend redirects the user to the **Payment Gateway** (e.g., **Stripe** or **PayPal**) to securely process payment.
* Upon successful payment, the payment gateway sends a confirmation back to the frontend.
* The frontend updates the user with the payment success and confirms the order.

**Step 5: Shipment Tracking**

* Once the order is placed and payment is successful, the **Shipment Tracking API** is used to fetch real-time updates about the delivery status.
* The user can view the current status of their order, including shipment tracking number, expected delivery time, and current location updates.

**Step 6: Order Confirmation**

* After successful payment and shipment tracking setup, an **Order Confirmation Page** is displayed to the user with all order details: product list, total price, shipment status, and expected delivery date.
* The order details are stored in **Sanity CMS** for future reference and order management.

**High-Level System Architecture Diagram:**

[Frontend (Next.js)]

|

v

[Product Data API] --> [Sanity CMS] --> [Order Data, Product Data]

|

v

[Payment Gateway (Stripe/PayPal)]

|

v

[Shipment Tracking API]

|

v

[User's Browser]

* **Frontend (Next.js)** communicates with the **Product Data API** and displays the products.
* **Sanity CMS** serves as the centralized database, managing all product information and order records.
* **Payment Gateway** securely processes payments.
* **Shipment Tracking API** provides real-time updates on the order’s delivery status.
* **User's Browser** receives the final output, showing the product details, order confirmation, and delivery status.

**Conclusion:**

This system architecture ensures seamless interaction between various components of the furniture ecommerce platform, facilitating smooth product browsing, secure payment processing, and real-time shipment tracking for customers. By integrating **Sanity CMS** for content management and leveraging third-party APIs for payments and logistics, this architecture supports both frontend and backend functionality in a scalable, efficient manner.