Design System Architecture

```mermaid
Frontend -->|Handles User Interaction| LocalStorage/Cookies
Frontend -->|Fetches Data| ShipmentTrackingAPI
Frontend -->|Handles Payments| PaymentGateway

ProductDataAPI -->|Fetches Data| SanityCMS
ProductDataAPI -->|Stores Data| Database

ShipmentTrackingAPI -->|Fetches Data| ThirdPartyShippingAPI

PaymentGateway -->|Handles Payments| PaymentProcessing
PaymentGateway -->|Stores Transactions| Database

SanityCMS -->|Content Management| Content
Database -->|Stores Data| UserData, OrderData, ProductData

ThirdPartyShippingAPI -->|Provides Shipment Data| ShippingData
PaymentProcessing -->|Processes Payments| PaymentData
```

Presentation Layer

- **Frontend (Next.js)**
  - User interface built with Next.js, handling routing, server-side rendering, and client-side interactions.
  - Interacts with the API Layer for data and business logic.

–

- **API Layer**
  - Handles business logic, data validation, and acts as an intermediary between the frontend and various data sources.
  - Communicates with the Security Layer for authentication and authorization.

–

- **Security Layer**
  - Manages authentication and authorization, ensuring only authorized users can access certain parts of the application.

- Implements JWT or OAuth for securing API endpoints.
- Ensures data encryption and input validation.

_

- **Sanity CMS**
  - Manages content like blog posts, about pages, etc.
  - Interacts with the API Layer for content management.

- **Product Data API**
  - Handles retrieval and management of product information.
  - Communicates with the API Layer for product-related requests.

- **Third-Party APIs**
  - Provides external services such as weather data, social media feeds, etc.
  - Integrated with the API Layer for specific functionalities.

- **Shipment Tracking API**
  - Handles order tracking and logistics.
  - Communicates with the API Layer for shipment-related data.

- **Payment Gateway**
  - Processes payments securely.
  - Integrated with the API Layer for financial transactions.

- **Database Layer**
  - Stores user data, orders, preferences, etc.
  - Could be a relational database like PostgreSQL or a NoSQL database like MongoDB.

- **Caching Layer**
  - Improves performance by caching frequently accessed data.
  - Reduces load on the database and APIs.

—

- **Background Processing Queue**
  - Handles asynchronous tasks like sending emails or processing payments.
  - Uses message queues like RabbitMQ or Redis for task management.

—

- **Monitoring & Logging**
  - Tools like Sentry for error tracking and Prometheus/Grafana for performance monitoring.
  - Centralized logging with ELK Stack for log analysis.

—

- **Load Balancer**
  - Distributes incoming traffic across multiple API servers.
  - Ensures high availability and performance under heavy load.

- **Disaster Recovery & Backup**
  - Solutions to ensure data integrity and availability in case of failures.
  - Includes regular backups and redundancy mechanisms.

Data Flow Summary

- **User Interaction:**
  - Users interact with the Frontend (Next.js), which sends requests to the API Layer.

- **Business Logic Execution:**
  - The API Layer processes requests, applying business logic and validating data.
  - Authenticates and authorizes users through the Security Layer.

- **Data Retrieval and Processing:**
  - The API Layer retrieves data from Sanity CMS, Product Data API, Third-Party APIs, Shipment Tracking API, and Payment Gateway.
  - Caches frequently accessed data to improve performance.

- **Background Tasks:**
  - Asynchronous tasks are offloaded to the Background Processing Queue for efficient handling.

- **Monitoring and Logging:**
  - The system monitors performance and logs activities for analysis and troubleshooting.

- **Scalability and Redundancy:**
  - Load balancers ensure even distribution of traffic, and disaster recovery solutions