**Low-Level Technical Design (LLD) for Garage Inventory and Order Management System**

**1. Overview**

The Garage Inventory and Order Management System will be built using a microservices architecture in Java Spring Boot. The system will automate inventory tracking, order placements to local and international suppliers, and ensure that parts are always available. The key components include services for Inventory Management, Order Management, Supplier Management, and Scheduler. The architecture will follow microservices patterns like event-driven communication and API gateway for external access.

**2. Key Design Considerations**

* **Microservices Principles**: Each service is independently deployable, scalable, and loosely coupled.
* **Data Consistency**: Eventual consistency through asynchronous communication for order placement.
* **Scalability**: Horizontal scalability to handle increasing load.
* **Persistence**: Relational database for auditability and data integrity.
* **Concurrency**: Optimized for handling multiple orders and inventory updates simultaneously.

**3. Microservice Components**

**3.1 Inventory Service**

* **Responsibilities**:
  + Manage available quantities of parts.
  + Update stock levels based on manual input.
  + Check threshold limits and trigger automated order placement.
* **Entities**:
  + **Part**: Contains details like partId, name, supplier, threshold, availableQty, minOrderQty.
  + **Supplier**: Enum to indicate either Supplier-A or Supplier-B.
* **APIs**:
  + GET /inventory/parts : Fetch all part details.
  + GET /inventory/part : Fetch part details for specific partId.
  + POST /inventory/part-add: Add new parts.
  + POST /inventory/part-update: Update/modify existing parts.
  + GET /inventory/low-stock: Check Low Stock Parts.
  + POST /inventory/stock-order: checkStockAndPlaceOrder of Parts.

**3.2 Order Service**

* **Responsibilities**:
  + Manage and place orders to suppliers.
  + Ensure orders are placed automatically when part quantity falls below the threshold.
  + Enforce discount rules for Supplier-B orders (only place between 12:00 AM to 1:00 AM).
* **Entities**:
  + **Order**: Contains orderId, partId, supplier, quantity, status (PENDING, PLACED, COMPLETED).
* **APIs**:
  + GET /order/orderbyId : Fetch order details by orderId.
  + GET /order/orderlist : Fetch all orders.
  + GET /order/orderbyPart :Fetch all orders for a specific Inventory part
  + GET /order/orders-supplier : Fetch all orders for a specific supplier
  + POST /order/createOrder : Place an order/Create a new order for an Inventory part.
  + POST/order/ updateOrderStatus : Update order status (e.g., processed, delivered)

**3.3 Supplier Service**

* **Responsibilities**:
  + Handle interactions with suppliers (Supplier-A and Supplier-B).
  + Communicate discount windows and ordering rules.
* **APIs**:
  + Internal service for placing orders with Supplier-A and Supplier-B.
  + POST/order/supplier-a – Place the order for Supplier-A
  + POST/order/supplier-b – Place the order for Supplier-B with Time Validation

**3.4 Scheduler Service**

* **Responsibilities**:
  + A scheduled task to place orders to Supplier-B only during the discount window (12:00 AM to 1:00 AM).
  + Run automated order placement for low-stock parts outside business hours.

**4. Database Design**

* **Inventory Table**:
  + part\_id: Unique identifier for the part.
  + name: Name of the part.
  + supplier\_id: Enum (Supplier-A, Supplier-B).
  + threshold\_limit: Minimum stock level before triggering an order.
  + available\_qty: Current stock level.
  + min\_order\_qty: Minimum quantity to order.
* **Order Table**:
  + order\_id: Unique identifier for the order.
  + part\_id: Foreign key to the Inventory table.
  + quantity: Ordered quantity.
  + supplier\_id: Supplier for the order.
  + status: Status of the order (PENDING, PLACED, COMPLETED).
  + order\_time: Timestamp of the order.
* **Supplier Table**:
  + supplier\_id: Unique ID for the supplier.
  + name: Name of the Supplier(A/B).
  + type: Supplier type (Local/International).

**5. Microservice Architecture Overview**

The system will be designed using microservices following key design principles:

* **Single Responsibility Principle (SRP):** Each service will have a well-defined scope and focus on a single business function.
* **Separation of Concerns:** Core functionalities will be distributed across different services such as InventoryService, OrderService, and SupplierService.
* **Scalability:** Each service will be independently deployable and scalable.
* **Resilience:** Circuit Breaker and Retry patterns will be applied to handle failures, especially in external API calls to suppliers.

**6. Microservice Design Patterns**

1. **Domain-Driven Design (DDD):** The domain is divided into bounded contexts:
   * **Inventory Context:** Handles all operations related to managing parts in the inventory.
   * **Order Context:** Handles automatic and manual order placements.
   * **Supplier Context:** Manages supplier data.
2. **Event-Driven Architecture (EDA):**
   * Events are triggered when the stock falls below a threshold. These events will be handled by the OrderService to automatically place orders.
3. **API Gateway Pattern:**
   * A centralized API Gateway will expose APIs to external clients for accessing inventory, orders, and suppliers.
4. **Scheduler Pattern:**
   * A scheduler will be implemented to manage the timing of orders placed with Supplier-B between 12:00 AM and 01:00 AM.
5. **Circuit Breaker Pattern:**
   * Circuit breakers will be applied in external service calls to suppliers to handle timeouts or failures.

**7. Service Breakdown**

* **Inventory Service:**
  + Manages the CRUD operations for inventory.
  + Exposes endpoints for adding, updating, retrieving parts.
  + Generates events when stock falls below threshold.
* **Order Service:**
  + Listens for events triggered by the Inventory Service.
  + Places automatic orders when thresholds are crossed.
  + Schedules orders for Supplier-B at specific times.
* **Supplier Service:**
  + Manages suppliers (Supplier-A, Supplier-B).
  + Exposes endpoints to manage supplier information.
* **Audit Service:**
  + Stores and retrieves order and inventory transactions for auditing purposes.

**8. Component Interactions**

1. **Inventory Service:**
   * Upon stock update, if a part’s quantity is below the threshold, an event is sent to Order Service.
2. **Order Service:**
   * Upon receiving an event from the Inventory Service, the order is placed.
   * For Supplier-B, the order is scheduled during the discount window.
3. **Supplier Service:**
   * Manages the details and schedule constraints for Supplier-A and Supplier-B.
4. **API Gateway:**
   * Routes API requests to respective services, hiding internal service details.

**9. Communication and Integration**

* **Event-Driven Communication**: The Inventory Service will emit events when stock levels fall below the threshold, triggering the Order Service to place an order asynchronously.
* **RESTful APIs**: Each microservice exposes REST endpoints for external communication.
* **API Gateway**: Acts as a single entry point for accessing all microservices, managing authentication and routing.

**10. Non-Functional Requirements**

* **Scalability**: Services should be capable of handling increased load as the business grows.
* **Reliability**: The system must handle failures gracefully and ensure data integrity during order placement.
* **Auditability**: Orders and inventory updates must be persisted for historical analysis.
* **Availability**: System uptime of 99.9% to ensure minimal disruption to business operations.
* **Performance**: Inventory updates and order placements must be processed within milliseconds to avoid stockouts.

**11. Sequence Flow**

1. **Inventory Update**: Garage owner updates the available stock manually.
2. **Threshold Check**: Inventory Service checks if any part falls below its threshold limit.
3. **Event Emission**: If stock is low, an event is emitted to trigger the Order Service.
4. **Order Placement**:
   * For Supplier-A: Order is placed immediately.
   * For Supplier-B: Scheduler ensures that the order is placed between 12:00 AM and 1:00 AM.
5. **Audit**: Orders are stored for auditing purposes and can be queried later.

**12. Error Handling**

* **Order Placement Failures**: Retry logic for failed orders to ensure eventual success.
* **Invalid Data**: Validation at API level to prevent invalid part or order entries.
* **Concurrency**: Optimistic locking to handle concurrent updates to inventory and orders.

**13. Deployment and Scalability**

* **Containerization**: Each microservice will be deployed in its own container using Docker.
* **Kubernetes**: Orchestration will be managed using Kubernetes, enabling auto-scaling and fault-tolerance.
* **Cloud Infrastructure**: Use AWS RDS for database persistence and S3 for backup and auditing logs.

**14. Technology Stack**

* **Programming Language**: Java 17
* **Framework**: Spring Boot 3.x for microservices
* **Database**: H2 (in-memory RDBMS for development), MySQL for production
* **Message Broker**: RabbitMQ for event-driven architecture
* **Scheduler**: Quartz Scheduler for scheduling Supplier-B orders

**15. Security Considerations**

* **API Security**: OAuth2 for securing RESTful endpoints.
* **Data Protection**: TLS for data in transit, and database encryption for sensitive information.

**This LLD outlines the system design for automating the inventory and order management system for Garage, ensuring that the system is scalable, reliable, and maintainable.**