Design document of Automobile Repair Shop

Requirements
- Functional Requirements
- Non-Functional Requirements

Microservices
- ● Inventory Service
- ● Order Service
- ● Notification Service
- ● Supplier Service
- ● User Management Service
- ● Apache Kafka
  - Why Kafka (Event Driven architecture)

Schema Design
- Inventory Service
  - Tables
  - Design Notes
- Order Service
  - Tables
  - Design Notes
- Supplier Service
    - Tables
  - Design Notes:
- Notification Service
    - Tables:
- User Management Service (Optional)
  - Tables:
    - Design Notes:

API Specs
- 1. Inventory Service API
    - Base URL: /api/inventory
- 2. Order Service API
    - Base URL: /api/orders
- 3. Supplier Service API
    - Base URL: /api/suppliers
- 4. Notification Service API
    - Base URL: /api/notifications

Component Diagram

Sequence Diagram
- Order Placement Workflow

Requirements

Functional Requirements

1. **Inventory Management**
    - **Add Parts**: Admin can add new parts to the inventory with details like name, threshold limit, minimum order quantity, supplier, and initial quantity.
    - **Update Parts**: Admin can update existing part details such as available quantity, supplier, and threshold limit.
    - **View Inventory**: Users can view the current inventory status, including available quantity, threshold limit, and supplier information for each part.
    - **Automatic Order Placement**: When a part's quantity falls below the threshold, the system triggers an automatic order to restock the item.
2. **Order Management**
    - **Create Orders**: System automatically creates an order when the inventory falls below the threshold.
    - **Scheduled Orders for Supplier-B**: Orders to Supplier-B must be placed between 12:00 AM and 1:00 AM to benefit from discounts.
    - **Immediate Orders for Supplier-A**: Orders to Supplier-A can be placed anytime without scheduling.
    - **Track Order Status**: System allows users to view the current status of each order (e.g., pending, completed, shipped).
    - **Order Modification**: Admin can manually adjust order quantities or cancel an order if needed.
3. **Supplier Management**
    - **Add and Manage Suppliers**: Admin can add new suppliers, view supplier details, and update supplier information.
    - **Assign Supplier to Parts**: Each part must be linked to a specific supplier (Supplier-A or Supplier-B).
4. **Audit and Reporting**
    - **Inventory Change Logs**: Maintain logs of inventory changes (e.g., part updates, orders placed) for auditing purposes.
    - **Order History Report**: Generate reports of past orders, including details like part name, quantity ordered, supplier, and status.
    - **Inventory Status Report**: Generate reports on current inventory status and threshold levels for parts.
5. **Notification System**
    - **Low Inventory Alerts**: Send alerts to the owner or admin if a part's quantity falls below the threshold.
    - **Order Confirmation and Status Updates**: Notify the admin of order confirmations and any status changes (e.g., shipping, cancellation).

6. **User Management**
    - **Role-Based Access Control**: Only authorized users can perform CRUD operations on parts, suppliers, and orders.

Non-Functional Requirements

1. **Performance**
    - **Response Time**: The system should handle API requests (CRUD operations on parts and orders) with an average response time of less than 2 seconds.
    - **Scalability**: The system should support a growing inventory of parts and an increasing number of users as the business expands to serve four-wheeler vehicles.
2. **Availability**
    - **Uptime**: The system should maintain high availability and be fault tolerant.
3. **Reliability**
    - **Data Consistency**: All inventory and order data should be consistent across the system. Inventory quantities must be accurately updated and orders placed only when necessary.
4. **Security**
    - **Access Control**: Role-based access control to ensure only authorized personnel can manage parts, suppliers, and orders.
5. **Maintainability**
    - **Modularity**: Implement modular services (inventory, orders, suppliers) for ease of maintenance and future updates.
    - **Documentation**: Provide comprehensive documentation for system design, database schema, and APIs for smooth maintenance and onboarding of new developers.
6. **Scalability**
    - **Horizontal Scaling**: Support for horizontal scaling of the backend services and database to accommodate higher load as the business expands.
7. **Observability, Logging, and Monitoring**

Microservices

- Inventory Service
    - Manages the core inventory data, including available quantities, thresholds, and parts information.
    - Checks inventory levels on updates, and if an inventory item falls below its threshold, it publishes a **"Low Inventory Update"** event to Kafka.
    - Updates data in a database.

- Order Service
    - Subscribes to **"Low Inventory Update"** events from Kafka to listen for low inventory alerts.
    - Upon receiving an alert for low inventory, it checks if the order is with supplier A, places order by calling supplier service. If the order is with supplier B, send it to the order scheduler which places the order during the discount window.
    - Stores order data in its database for tracking order history and status.
- Notification Service
    - Subscribes to **"Order Status Update"** events from Kafka and sends notifications to the garage owner or admin for order confirmation..
- Supplier Service
    - Manages supplier data, including Supplier-A and Supplier-B details, and handles supplier interactions.
- User Management Service
    - Manage user roles, permissions, and authentication.
    - Enforce access control, ensuring that only authorized users can perform certain actions.
- Apache Kafka
    - Serves as the event bus, facilitating real-time communication between services. Kafka hosts different topics to separate event types

Why Kafka (Event Driven architecture)

This architecture ensures that real-time updates are efficiently handled through Kafka, allowing each service to act independently while maintaining synchronization on inventory levels and orders. Kafka's publish-subscribe model helps achieve a decoupled, scalable system suited for future expansion as B-Garage grows.

Schema Design

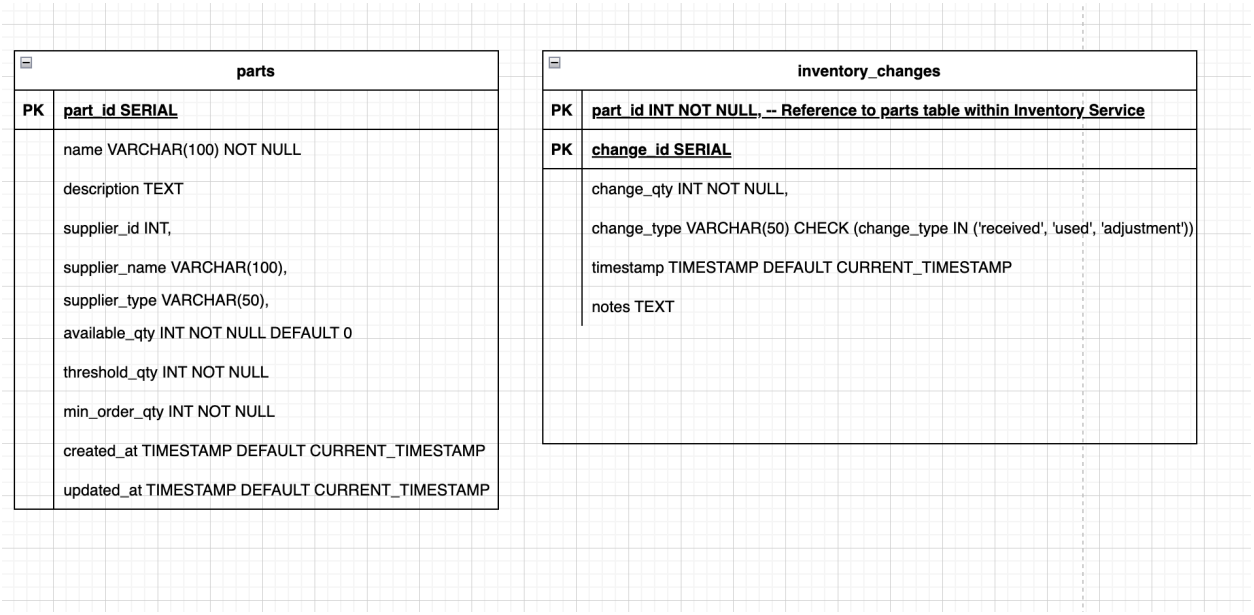
Inventory Service

The Inventory Service will manage data for each part independently. Supplier details will be stored as soft references without enforcing a foreign key constraint. This allows the Inventory Service to function independently, while still keeping relevant supplier data locally.

Tables

1. `parts`: Stores information about each part, including soft references to supplier information.

2. `inventory_changes`: Logs all changes to inventory levels (e.g., stock received or used).



Design Notes

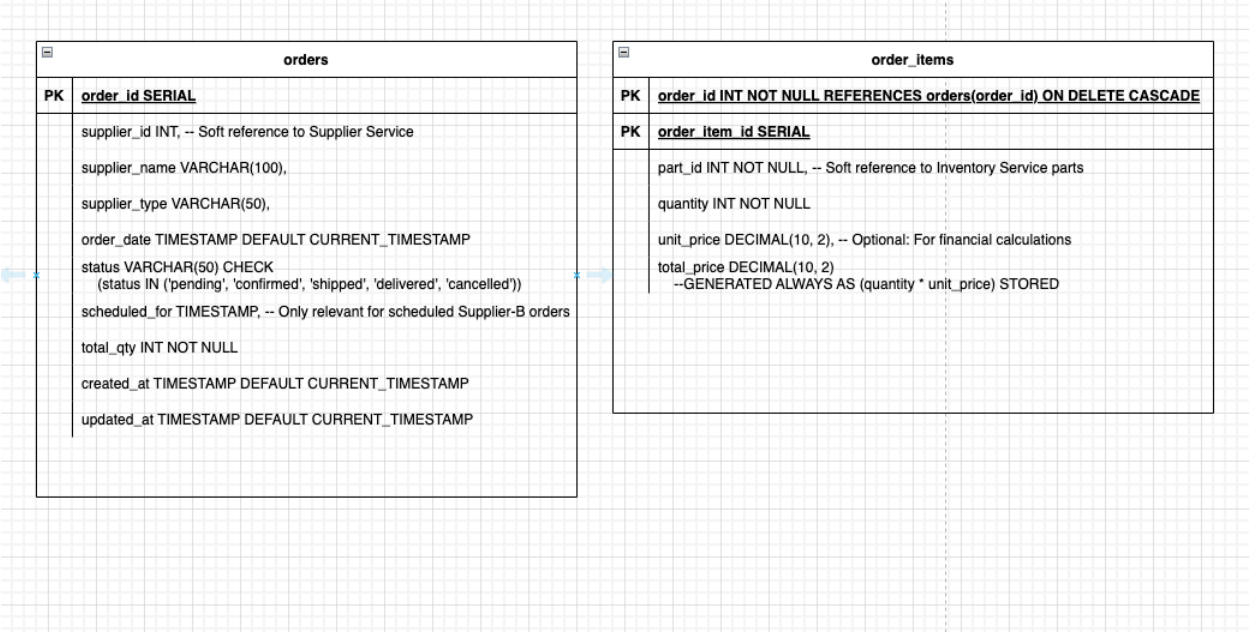
- **Soft References**: `supplier_id`, `supplier_name`, and `supplier_type` are stored in the `parts` table without enforcing foreign key constraints, allowing for local supplier information without dependency on the Supplier Service.
- **Data Syncing**: Supplier data can be periodically synchronized via events from the Supplier Service to keep the cached information up-to-date.

Order Service

The Order Service is responsible for managing orders and order items independently. This service handles its own data for each order, with soft references to supplier details to track which supplier will fulfill each order.

Tables

1. `orders`: Stores each order, including soft references to the supplier information.
2. `order_items`: Stores individual items within an order, enabling the service to manage multi-item orders if needed.



Design Notes

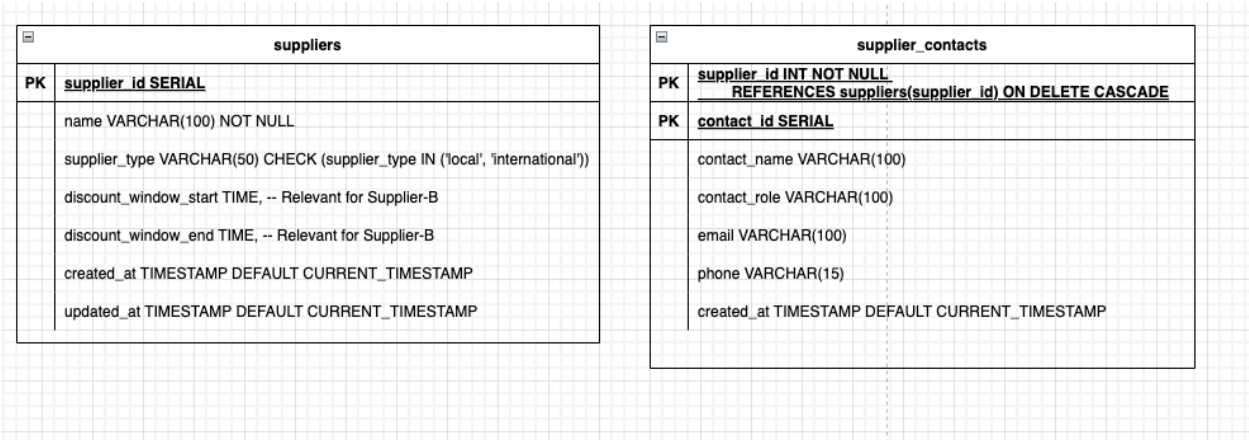
- Soft References to Suppliers: supplier_id, supplier_name, and supplier_type are cached for each order, which keeps the order record independent of the Supplier Service.
- Multi-Item Orders: The order_items table supports orders with multiple parts, allowing flexibility for more complex orders.
- Scheduled Orders: The scheduled_for field allows the system to handle time-based ordering for Supplier-B.

Supplier Service

The Supplier Service manages data on suppliers independently, including contact information and ordering preferences. It owns supplier data and provides other services with relevant supplier information via API calls or events.

Tables

1. **`suppliers`**: Stores each supplier's details.
2. **`supplier_contacts`**: Stores multiple contact methods for each supplier.



Design Notes:

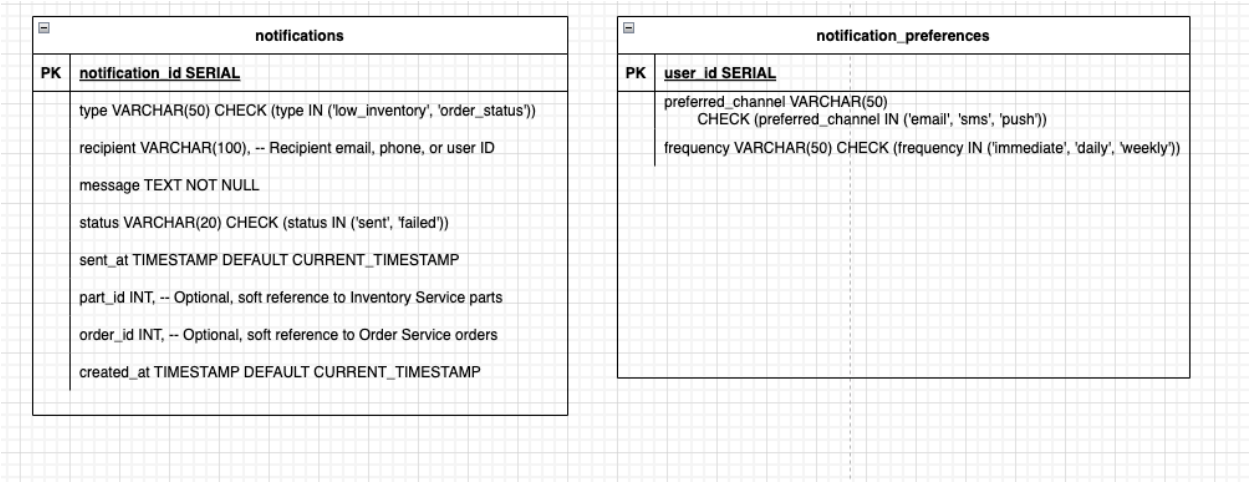
- Supplier Contacts: The `supplier_contacts` table allows multiple contacts per supplier, providing flexibility for managing supplier relationships.
- Discount Window: The `discount_window_start` and `discount_window_end` fields store the specific time window for placing orders with Supplier-B to get a discount.

Notification Service

The Notification Service tracks notifications independently, allowing the system to store, audit, and retrieve notification records without dependencies on other services.

Tables:

1. **`notifications`**: Stores each notification sent to users.
2. **`notification_preferences`** (optional): Stores users' preferences for receiving notifications.

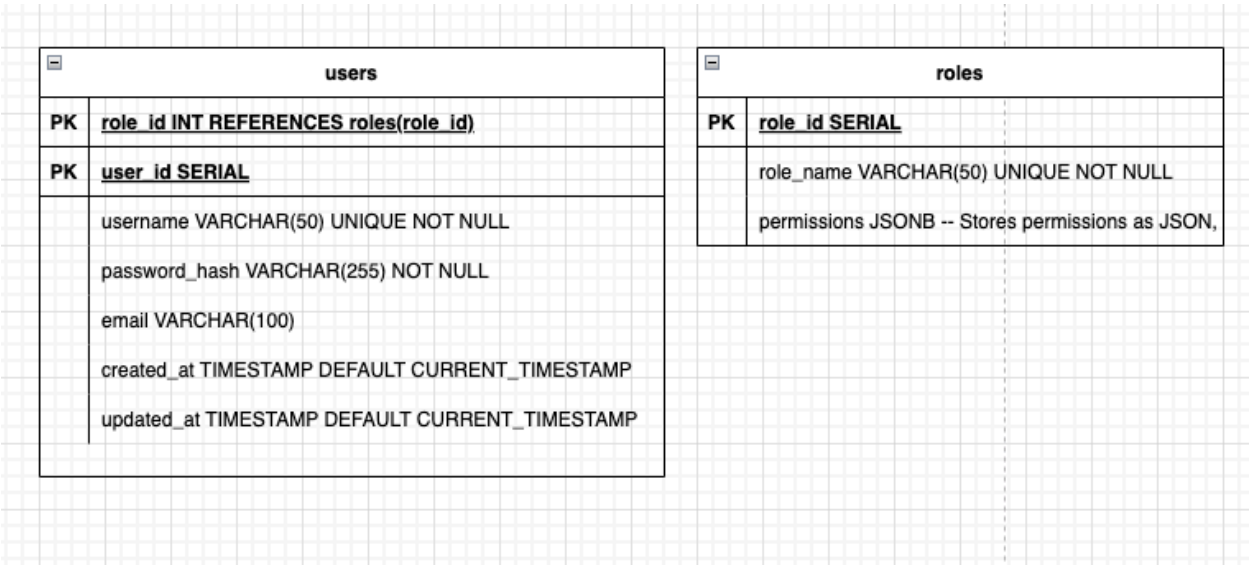


User Management Service (Optional)

If the system requires user and role management, this service manages user data independently, including access roles and permissions.

Tables:

1. `users`: Stores user information and login details.
2. `roles`: Stores role definitions and associated permissions.



Design Notes:

- **Role-Based Access Control**: The `roles` table allows defining access permissions for different types of users (e.g., admin, manager).
- **JSON Permissions**: The `permissions` column stores permissions in a flexible JSON format, enabling quick updates to user roles and permissions without altering the table schema.

API Specs

1. Inventory Service API

Base URL: `/api/inventory`

HTTP Method	Endpoint	Description
POST	`/parts`	Add a new part to the inventory, including details like name, supplier, threshold, minimum order quantity, etc.
GET	`/parts`	Retrieve a list of all parts in inventory along with current quantities, threshold limits, and suppliers.
GET	`/parts/{id}`	Retrieve details of a specific part by its ID.
PUT	`/parts/{id}`	Update details of a part (e.g., threshold, quantity, supplier) by its ID.
DELETE	`/parts/{id}`	Remove a part from inventory by its ID.
GET	`/parts/{id}/status`	Check the inventory status of a specific part, including current quantity and threshold.
GET	`/low-inventory`	Retrieve a list of parts that are below their threshold limits.
POST	`/parts/{id}/adjust-quantity`	Adjust the quantity of a specific part by a given amount (positive or negative), such as for stock received or used.

2. Order Service API

Base URL: `/api/orders`

HTTP Method	Endpoint	Description
POST	`/`	Place a new order for a specific part. The system can call this endpoint directly, or it can be triggered automatically.
GET	`/`	Retrieve a list of all orders, including details like part name, quantity, supplier, and status.
GET	`/orders/{id}`	Retrieve details of a specific order by its ID.

PUT	`/orders/{id}/update-status`	Update the status of a specific order (e.g., "pending," "shipped," "delivered") by its ID.
DELETE	`/orders/{id}`	Cancel an order by its ID.
POST	`/place-auto-order`	Endpoint to trigger automated ordering based on low inventory alerts from the Inventory Service.

3. Supplier Service API

Base URL: `/api/suppliers`

HTTP Method	Endpoint	Description
POST	`/`	Add a new supplier, including name, contact information, and supplier type (Supplier-A or Supplier-B).
GET	`/`	Retrieve a list of all suppliers, including contact information and types.
GET	`/suppliers/{id}`	Retrieve details of a specific supplier by its ID.
PUT	`/suppliers/{id}`	Update details of a supplier (e.g., contact information, supplier type) by its ID.
DELETE	`/suppliers/{id}`	Remove a supplier from the system by its ID.
GET	`/suppliers/{id}/orders`	Retrieve a list of all orders associated with a specific supplier by its ID.

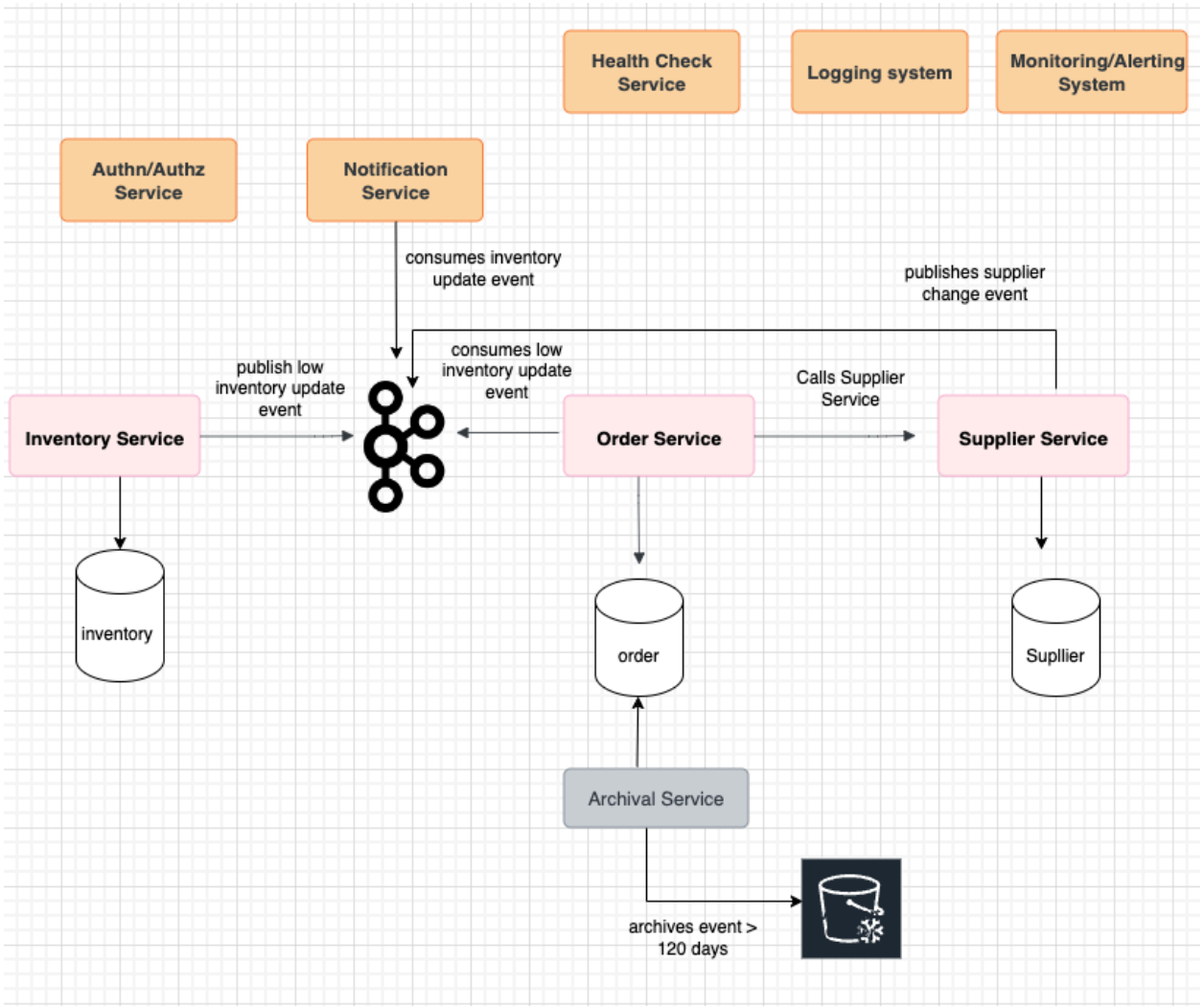
4. Notification Service API

Base URL: `/api/notifications`

HTTP Method	Endpoint	Description
POST	`/send-alert`	Send an alert notification for low inventory. This can be triggered automatically based on Inventory Service events.

POST	`/send-order-status`	Send a notification about an order status update (e.g., "shipped," "delivered").
GET	`/alerts`	Retrieve a history of alerts sent for low inventory events.
GET	`/alerts/orders`	Retrieve a history of notifications sent for order status updates.

Component Diagram



Sequence Diagram

Order Placement Workflow