Here's a high-level skeleton of the project with Data Model, Controller, and Class definitions for the B-Garage Inventory and Order Management System, following microservices principles and designed in Java with Spring Boot.

**1. Project Structure**

**The project is divided into separate services for Inventory, Order, and Supplier, Scheduler using Spring Boot microservices. Below is the proposed structure**:

**bg-inventory-order-system**

│

├── inventory-service

│ ├── src/main/java/com/garage/inventory

│ │ ├── controller

│ │ ├── model

│ │ ├── repository

│ │ ├── service

│ │ └── InventoryServiceApplication.java

│

├── order-service

│ ├── src/main/java/com/garage/order

│ │ ├── controller

│ │ ├── model

│ │ ├── repository

│ │ ├── service

│ │ └── OrderServiceApplication.java

│

├── supplier-service

│ ├── src/main/java/com/garage/supplier

│ │ ├── controller

│ │ ├── model

│ │ ├── service

│ │ └── SupplierServiceApplication.java

│

└── scheduler-service

├── src/main/java/com/garage/scheduler

│ ├── scheduler

│ └── SchedulerServiceApplication.java

1. **Inventory Service**
2. **Data Model**

package com.garage.inventory.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.EnumType;

import javax.persistence.Enumerated;

import jakarta.persistence.\*;

@Entity

@Table(name = "inventory")

public class Inventory {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String partName;

@ManyToOne

@JoinColumn(name = "supplier\_id", nullable = false)

private Supplier supplier;

@Column(nullable = false)

private int availableQuantity;

@Column(nullable = false)

private int thresholdLimit;

@Column(nullable = false)

private int minimumOrderQuantity;

// Constructor(s)

public Inventory() {}

public Inventory(String partName, Supplier supplier, int availableQuantity, int thresholdLimit, int minimumOrderQuantity) {

this.partName = partName;

this.supplier = supplier;

this.availableQuantity = availableQuantity;

this.thresholdLimit = thresholdLimit;

this.minimumOrderQuantity = minimumOrderQuantity;

}

// Getters and Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getPartName() {

return partName;

}

public void setPartName(String partName) {

this.partName = partName;

}

public Supplier getSupplier() {

return supplier;

}

public void setSupplier(Supplier supplier) {

this.supplier = supplier;

}

public int getAvailableQuantity() {

return availableQuantity;

}

public void setAvailableQuantity(int availableQuantity) {

this.availableQuantity = availableQuantity;

}

public int getThresholdLimit() {

return thresholdLimit;

}

public void setThresholdLimit(int thresholdLimit) {

this.thresholdLimit = thresholdLimit;

}

public int getMinimumOrderQuantity() {

return minimumOrderQuantity;

}

public void setMinimumOrderQuantity(int minimumOrderQuantity) {

this.minimumOrderQuantity = minimumOrderQuantity;

}

}

1. **Repository**

package com.garage.inventory.repository;

import com.garage.inventory.model.Inventory;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface PartRepository extends JpaRepository<Inventory, Long> {

List<Part> findByAvailableQtyLessThan(int threshold);

}

1. **Service**

package com.garage.inventory.service;

import com.garage.inventory.model.Inventory;

import com.garage.inventory.repository.PartRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class InventoryService {

@Autowired

private PartRepository partRepository

private SupplierService supplierService;

public Inventory getPartById(Long partId) {

return partRepository.findById(partId)

.orElseThrow(() -> new ResourceNotFoundException("Part not found with id: " + partId));

}

public List<Inventory> getAllParts() {

return partRepository.findAll();

}

public Inventory addNewPart(Inventory part) {

// Check if part with same name or identifier already exists to avoid duplicates

if (partRepository.existsByName(part.getName())) {

throw new IllegalArgumentException("Part with name " + part.getName() + " already exists.");

}

public Inventory updatePart(Long partId, Part updatedPart) {

// Fetch the part to be updated

Part existingPart = partRepository.findById(partId)

.orElseThrow(() -> new ResourceNotFoundException("Part not found with id: " + partId));

// Update part properties

existingPart.setName(updatedPart.getName());

existingPart.setSupplier(updatedPart.getSupplier());

existingPart.setThresholdLimit(updatedPart.getThresholdLimit());

existingPart.setMinimumOrderQty(updatedPart.getMinimumOrderQty());

// Save the updated part details to the repository

return partRepository.save(existingPart);

}

// Save new part to repository (Inventory)

return partRepository.save(part);

}

public List<Inventory> checkLowStockParts() {

return partRepository.findByAvailableQtyLessThan(10); // Example threshold check

}

public void checkStockAndPlaceOrder(Long partId) {

Inventory part = partRepository.findById(partId)

.orElseThrow(() -> new ResourceNotFoundException("Part not found with id: " + partId));

// Check if the available quantity is below the threshold

if (part.getAvailableQty() < part.getThresholdLimit()) {

// Determine which supplier to place the order with

Supplier supplier = part.getSupplier();

int orderQuantity = part.getMinimumOrderQty(); // Minimum order quantity

// Place order with Supplier-A or Supplier-B

if (supplier.getName().equals("Supplier-A")) {

supplierService.placeOrderToSupplierA(part.getPartId(), orderQuantity);

} else if (supplier.getName().equals("Supplier-B")) {

supplierService.placeOrderToSupplierB(part.getPartId(), orderQuantity);

}

// You can log or return an acknowledgement that the order has been placed

} else {

System.out.println("Stock level is sufficient for part: " + part.getName());

}

}

}

**Conclusion**

This InventoryService provides the necessary methods to manage the parts inventory, update quantities, and automatically place orders when the stock levels fall below the threshold. The integration with the SupplierService allows for placing orders based on the supplier's specific rules. This design follows the principles of modularity and separation of concerns, ensuring that each component handles a specific aspect of the inventory and order management process.

1. **Controller**

package com.garage.inventory.controller;

import com.garage.inventory.model.Inventory;

import com.garage.inventory.service.InventoryService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/inventory")

public class InventoryController {

@Autowired

private InventoryService inventoryService;

@GetMapping("/partid")

public <Inventory> getPartById (@RequestBody Long partId ) {

return inventoryService. getPartById (partId);

}

@GetMapping("/parts")

public List<Inventory> getAllParts() {

return inventoryService.getAllParts();

}

@PostMapping("/part-add")

public ResponseEntity<Inventory> addNewPart (@RequestBody Inventory part) {

Inventory addPart = inventoryService. addNewPart (part);

return ResponseEntity.ok(addPart);

}

@PostMapping("/part- update")

public ResponseEntity<Inventory> updatePart (@RequestBody Long partId,Inventory part) {

Inventory updatePart = inventoryService. updatePart (partId,part);

return ResponseEntity.ok(updatePart);

}

@GetMapping("/low-stock")

public List<Inventory> checkLowStockParts() {

return inventoryService.checkLowStockParts();

}

@PostMapping("/stock-order")

public ResponseEntity <void> checkStockAndPlaceOrder (@RequestBody Long partId) {

inventoryService. checkStockAndPlaceOrder (partId);

}

}

1. **Order Service**
2. **Data Model**

package com.garage.model;

import jakarta.persistence.\*;

import java.time.LocalDateTime;

@Entity

@Table(name = "orders")

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne

@JoinColumn(name = "inventory\_id", nullable = false)

private Inventory inventory;

@ManyToOne

@JoinColumn(name = "supplier\_id", nullable = false)

private Supplier supplier;

@Column(nullable = false)

private int orderQuantity;

@Column(nullable = false)

private LocalDateTime orderTimestamp;

@Column(nullable = false)

private String orderStatus;

// Constructor(s)

public Order() {}

public Order(Inventory inventory, Supplier supplier, int orderQuantity, LocalDateTime orderTimestamp, String orderStatus) {

this.inventory = inventory;

this.supplier = supplier;

this.orderQuantity = orderQuantity;

this.orderTimestamp = orderTimestamp;

this.orderStatus = orderStatus;

}

// Getters and Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public Inventory getInventory() {

return inventory;

}

public void setInventory(Inventory inventory) {

this.inventory = inventory;

}

public Supplier getSupplier() {

return supplier;

}

public void setSupplier(Supplier supplier) {

this.supplier = supplier;

}

public int getOrderQuantity() {

return orderQuantity;

}

public void setOrderQuantity(int orderQuantity) {

this.orderQuantity = orderQuantity;

}

public LocalDateTime getOrderTimestamp() {

return orderTimestamp;

}

public void setOrderTimestamp(LocalDateTime orderTimestamp) {

this.orderTimestamp = orderTimestamp;

}

public String getOrderStatus() {

return orderStatus;

}

public void setOrderStatus(String orderStatus) {

this.orderStatus = orderStatus;

}

}

1. **Repository**

package com.garage.order.repository;

import com.garage.order.model.Order;

import org.springframework.data.jpa.repository.JpaRepository;

public interface OrderRepository extends JpaRepository<Order, Long> {

}

1. **Service**

package com.garage.order.service;

import com.garage.order.model.Order;

import com.garage.order.repository.OrderRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.time.LocalDateTime;

@Service

public class OrderService {

@Autowired

private OrderRepository orderRepository;

public Order createOrder(Inventory part, int quantity) {

// Create a new order object

Order order = new Order();

order.setPart(part); // Set the part for which the order is being placed

order.setQuantity(quantity); // Set the quantity to be ordered

order.setSupplier(part.getSupplier()); // Set the supplier for the part

order.setOrderDate(LocalDateTime.now()); // Set the current timestamp as order date

order.setOrderStatus(OrderStatus.PENDING); // Set initial status as 'PENDING'

// Save the order to the repository (database)

return orderRepository.save(order);

}

public List<Order> getAllOrders() {

return orderRepository.findAll(); // Fetch all orders from the database

}

public Order getOrderById(Long orderId) {

return orderRepository.findById(orderId)

.orElseThrow(() -> new ResourceNotFoundException("Order not found with id: " + orderId));

}

public List<Order> getOrdersByPart(Long partId) {

return orderRepository.findByPartId(partId); // Find all orders associated with the partId

}

public List<Order> getOrdersBySupplier(Long supplierId) {

return orderRepository.findBySupplierId(supplierId); // Find all orders associated with the supplierId

}

public void updateOrderStatus(Long orderId, OrderStatus status) {

Order order = orderRepository.findById(orderId)

.orElseThrow(() -> new ResourceNotFoundException("Order not found with id: " + orderId));

// Update the order status

order.setOrderStatus(status);

// Save the updated order to the repository

orderRepository.save(order);

}

**Additional Considerations**

* **Transactionality**: When creating an order, the process may involve interacting with multiple services (e.g., inventory update, supplier service). These operations should be wrapped in a transaction to ensure consistency.
* **Audit Logging**: Since orders need to be persisted for auditing purposes, make sure that all order creation, update, and status change actions are properly logged for traceability.
* **OrderStatus Enum**: You can define an OrderStatus enum to represent different statuses of an order, such as PENDING, PROCESSED, IN\_TRANSIT, and DELIVERED.

public enum OrderStatus {

PENDING,

PROCESSED,

IN\_TRANSIT,

DELIVERED

}

**Conclusion**

The OrderService handles the creation of new orders when inventory is low, tracks the orders based on their part and supplier, and manages the lifecycle of the order status. It integrates with both the inventory system and supplier system, ensuring that orders are placed and tracked effectively. This design follows microservice best practices, focusing on modularity, separation of concerns, and transaction management for consistent operations.

1. **Controller**

package com.garage.order.controller;

import com.garage.order.model.Order;

import com.garage.order.service.OrderService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/order")

public class OrderController {

@Autowired

private OrderService orderService;

@PostMapping("/createOrder ")

public ResponseEntity<Order> createOrder (@RequestParam Inventory part, @RequestParam int quantity) {

Order order = orderService.createOrder (part, quantity);

return ResponseEntity.ok(order);

}

@GetMapping("/orderlist")

public List<Order> getAllOrders () {

return inventoryService.getAllOrders ();

}

@GetMapping("/orderbyId")

public Order getOrderById (Long orderId) {

return inventoryService.getOrderById (orderId);

}

@GetMapping("/orderbyPart")

public List<Order> getOrdersByPart(Long partId) {

return inventoryService. getOrdersByPart(orderId);

}

@GetMapping("/orders-supplier")

public List<Order> getOrdersBySupplier(Long partId) {

return inventoryService. getOrdersBySupplier(orderId);

}

@PostMapping("/updateOrderStatus")

public updateOrderStatus(@RequestParam Long orderId,@RequestParam OrderStatus status) {

orderService.updateOrderStatus(orderId,status);

//return ResponseEntity.ok(order);

}

}

1. **Supplier Service**
2. **Data Model**

package com.garage.model;

import jakarta.persistence.\*;

import java.util.List;

@Entity

@Table(name = "supplier")

public class Supplier {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, unique = true)

private String supplierName;

@Column(nullable = false)

private String supplierType; // e.g., "Local" for Supplier-A, "International" for Supplier-B

@OneToMany(mappedBy = "supplier", cascade = CascadeType.ALL, fetch = FetchType.LAZY)

private List<Inventory> inventoryList;

@OneToMany(mappedBy = "supplier", cascade = CascadeType.ALL, fetch = FetchType.LAZY)

private List<Order> orderList;

// Constructor(s)

public Supplier() {}

public Supplier(String supplierName, String supplierType) {

this.supplierName = supplierName;

this.supplierType = supplierType;

}

// Getters and Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getSupplierName() {

return supplierName;

}

public void setSupplierName(String supplierName) {

this.supplierName = supplierName;

}

public String getSupplierType() {

return supplierType;

}

public void setSupplierType(String supplierType) {

this.supplierType = supplierType;

}

public List<Inventory> getInventoryList() {

return inventoryList;

}

public void setInventoryList(List<Inventory> inventoryList) {

this.inventoryList = inventoryList;

}

public List<Order> getOrderList() {

return orderList;

}

public void setOrderList(List<Order> orderList) {

this.orderList = orderList;

}

}

1. **Repository**

package com.garage.supplier.repository;

import com.garage.order.model.Supplier;

import org.springframework.data.jpa.repository.JpaRepository;

public interface SupplierRepository extends JpaRepository<Supplier, Long> {

}

1. **Service**

package com.garage.supplier.service;

import org.springframework.stereotype.Service;

@Service

public class SupplierService {

public void placeOrderToSupplierA(Long orderId) {

// Logic to place order to Supplier-A

@Override

public Order placeOrderToSupplierA(Long partId, int orderQuantity) {

Part part = partRepository.findById(partId)

.orElseThrow(() -> new ResourceNotFoundException("Part not found with id: " + partId));

// Check if part's available quantity is below the threshold

if (part.getAvailableQty() < part.getThresholdLimit()) {

int finalOrderQuantity = Math.max(orderQuantity, part.getMinimumOrderQty());

// Create new order

Order order = new Order();

order.setPart(part);

order.setSupplier(part.getSupplier()); // Supplier-A

order.setOrderDate(LocalDate.now());

order.setQuantity(finalOrderQuantity);

order.setStatus("ORDERED");

// Save order and update inventory

orderRepository.save(order);

// Update inventory after placing order

part.setAvailableQty(part.getAvailableQty() + finalOrderQuantity);

partRepository.save(part);

return order;

} else {

throw new IllegalArgumentException("Part stock is sufficient. No need to place an order.");

}

}

}

public void placeOrderToSupplierB(Long orderId) {

// Logic to place order to Supplier-B

@Override

public Order placeOrderToSupplierB(Long partId, int orderQuantity) {

Part part = partRepository.findById(partId)

.orElseThrow(() -> new ResourceNotFoundException("Part not found with id: " + partId));

// Get the current time and check if it's between 12:00 AM and 01:00 AM

LocalTime currentTime = LocalTime.now();

LocalTime discountStartTime = LocalTime.of(0, 0); // 12:00 AM

LocalTime discountEndTime = LocalTime.of(1, 0); // 01:00 AM

if (currentTime.isAfter(discountStartTime) && currentTime.isBefore(discountEndTime)) {

// Check if part's available quantity is below the threshold

if (part.getAvailableQty() < part.getThresholdLimit()) {

int finalOrderQuantity = Math.max(orderQuantity, part.getMinimumOrderQty());

// Create new order

Order order = new Order();

order.setPart(part);

order.setSupplier(part.getSupplier()); // Supplier-B

order.setOrderDate(LocalDate.now());

order.setQuantity(finalOrderQuantity);

order.setStatus("ORDERED");

// Save order and update inventory

orderRepository.save(order);

// Update inventory after placing order

part.setAvailableQty(part.getAvailableQty() + finalOrderQuantity);

partRepository.save(part);

return order;

} else {

throw new IllegalArgumentException("Part stock is sufficient. No need to place an order.");

}

} else {

throw new IllegalStateException("Order to Supplier-B can only be placed between 12:00 AM and 01:00 AM.");

}

}

}

}

1. **Controller**

package com.garage.supplier.controller;

import com.garage.supplier.service.SupplierService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/supplier")

public class SupplierController {

@Autowired

private SupplierService supplierService;

@PostMapping("/order/supplier-a")

public void orderToSupplierA(@RequestParam Long orderId) {

supplierService.placeOrderToSupplierA(orderId);

}

@PostMapping("/order/supplier-b")

public void orderToSupplierB(@RequestParam Long orderId) {

supplierService.placeOrderToSupplierB(orderId);

}

}

1. **Scheduler Service**
2. **Scheduler**

package com.garage.scheduler.scheduler;

import com.garage.supplier.service.SupplierService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class SupplierBScheduler {

@Autowired

private SupplierService supplierService;

@Scheduled(cron = "0 0 0-1 \* \* ?") // Runs between 12:00 AM and 01:00 AM

public void scheduleOrderToSupplierB() {

// Logic to check low stock parts and place orders to Supplier-B

}

}

1. **Event Handling**
2. **LowStockEvent**

package com.garage.event;

import com.garage.model.Part;

public class LowStockEvent {

private Part part;

public LowStockEvent(Part part) {

this.part = part;

}

public Part getPart() {

return part;

}

}

1. **OrderPlacedEvent**

package com.garage.event;

import com.garage.model.Order;

public class OrderPlacedEvent {

private Order order;

public OrderPlacedEvent(Order order) {

this.order = order;

}

public Order getOrder() {

return order;

}

}

**Additional Considerations**

* **Database**: H2 for in-memory storage (during development), switch to MySQL/PostgreSQL in production.
* **Event-Driven Approach**: Use RabbitMQ or Kafka for communication between services.
* **Security**: Implement OAuth2 for securing the endpoints and ensure data safety.
* **API Gateway**: Configure API Gateway (e.g., Spring Cloud Gateway) to route requests to the respective services.

This skeleton provides a scalable, modular approach for building the Inventory and Order Management System using Spring Boot microservices and follows best practices for clean code and architecture.

Below are the details for the application.yaml, pom.xml, and the main application start file (GarageApplication.java) for **Garage Inventory and Order Management System** using Spring Boot.

1. **application.yaml Configuration**

This file contains configuration settings for the application, such as database settings, server port, and more.

# application.yaml

spring:

datasource:

url: jdbc:h2:mem:bgaragedb

driver-class-name: org.h2.Driver

username: sa

password: password

h2:

console:

enabled: true # Enable H2 database console for development/testing purposes

jpa:

hibernate:

ddl-auto: update # Automatically update schema based on JPA entities

show-sql: true # Show SQL queries in the console

# Server properties

server:

port: 8080 # Port on which the application will run

# Scheduling configuration for Supplier-B automated orders

scheduler:

supplierB:

enabled: true # Enable scheduling for Supplier-B

discount-time: "12:00-01:00" # Discount window for Supplier-B orders

# Application-specific properties (could be expanded)

inventory:

threshold-check-interval: "PT24H" # Threshold check interval (can use ISO 8601 duration format)

logging:

level:

org.springframework: INFO # Set logging level to INFO

com.bgarage: DEBUG # Debugging for our application

# Profiles (optional for future expansions)

spring:

profiles:

active: development

1. **pom.xml Configuration**

This is the Maven build file for your Spring Boot microservice. It includes dependencies for Spring Boot, H2 Database (in-memory), JPA, scheduling, and other necessary libraries.

<!-- pom.xml -->

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.garage</groupId>

<artifactId>bg-inventory-order-management</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<name>bg-inventory-order-management</name>

<description>Spring Boot microservice for B-Garage Inventory and Order Management System</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.0.0</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Spring Boot Dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- In-Memory Database (H2) -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Actuator (Optional for monitoring endpoints) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<!-- Scheduler for Supplier-B Orders -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-quartz</artifactId>

</dependency>

<!-- Lombok (Optional for generating boilerplate code like getters, setters, constructors) -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<scope>provided</scope>

</dependency>

<!-- Testing Dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Spring Boot Maven Plugin -->

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<version>3.0.0</version>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

1. **Main Application Class (GarageApplication.java)**

This is the main entry point of your Spring Boot application, which will bootstrap the microservices.

package com.garage;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication

@EnableScheduling // Enables scheduling for the Supplier-B discount time scheduling

public class GarageApplication {

public static void main(String[] args) {

SpringApplication.run(GarageApplication.class, args);

}

}

**Explanation:**

1. **application.yaml**:
   * Configures the H2 database for development purposes. The application can later be scaled to use MySQL, PostgreSQL, or any other persistent database in production.
   * Enables the H2 console for development/debugging.
   * Sets up server port and basic logging configurations.
   * Contains settings for scheduling Supplier-B orders between 12:00 AM and 01:00 AM.
2. **pom.xml**:
   * Includes Spring Boot starter dependencies for web services (spring-boot-starter-web), JPA for data access (spring-boot-starter-data-jpa), in-memory H2 database, scheduling (spring-boot-starter-quartz), and actuator (for monitoring endpoints).
   * Lombok is included (optional) for reducing boilerplate code like getters, setters, and constructors.
   * Java 17 is specified for the project as the Java version.
3. **GarageApplication.java**:
   * This is the entry point for the Spring Boot application. The @EnableScheduling annotation allows scheduled tasks to run, which will be useful for automating Supplier-B orders during the discount time.