Körber

Java Microservices Assignment

Purpose

Imagine a growing e-commerce company that manages thousands of products across multiple warehouses. They face challenges in tracking inventory batches with expiry dates and ensuring timely order fulfillment. To streamline operations, they need a robust system where Inventory Service can manage stock efficiently and Order Service can place orders while checking real-time availability. This project simulates that scenario, giving you hands-on experience in building scalable, extensible microservices using modern Java practices.

Objective

Design and implement two Spring Boot microservices — Order Service and Inventory Service — that communicate via REST APIs. The system should be modular and extensible using the Factory Design Pattern, allowing future expansion of services and logic.

Microservice 1: Inventory Service

Requirements:

- Maintain inventory of materials/products.

- Each product can have multiple batches with different expiry dates.

- Implement an endpoint to return inventory batches sorted by expiry date for a given product.

- Use Spring Data JPA with an H2 in-memory database.

- Implement a Factory Pattern to allow future extension of inventory handling logic.

- Include Controller, Service, and Repository layers.

Endpoints:

GET /inventory/{productId} – Returns list of inventory batches sorted by expiry date.

POST /inventory/update – Updates inventory after an order is placed.

Microservice 2: Order Service

Requirements:

- Accept and process product orders.

- Communicate with Inventory Service to check availability and update stock.

- Use RestTemplate or WebClient for inter-service communication.

- Include Controller, Service, and Repository layers.

Körber

- Use Spring Data JPA with H2 database.

Endpoints:

POST /order – Places an order and updates inventory accordingly.

Testing Requirements

- Write unit tests for service logic using JUnit 5 and Mockito.

- Write component/integration tests using @SpringBootTest and H2 database.

- Ensure REST endpoints are covered in tests.

Architecture Requirements

- Follow Factory Design Pattern in Inventory Service to allow future extensibility.

- Ensure all classes are designed to be extendable and loosely coupled.

- Use Lombok to reduce boilerplate code (optional).

- Use Swagger/OpenAPI for API documentation (optional).

Sample CSV Data for Inventory

```
batch_id,product_id,product_name,quantity,expiry_date
1,1001,Laptop,68,2026-06-25
2,1005,Smartwatch,52,2026-05-30
3,1004,Headphones,20,2026-08-12
4,1003,Tablet,35,2026-09-03
5,1005,Smartwatch,39,2026-03-31
6,1004,Headphones,56,2026-06-06
7,1005,Smartwatch,40,2026-04-24
8,1003,Tablet,21,2026-09-09
9,1002,Smartphone,29,2026-05-31
10,1002,Smartphone,83,2026-11-15
```

Sample CSV Data for Orders

```
order_id,product_id,product_name,quantity,status,order_date
1,1005,Smartwatch,10,DELIVERED,2025-12-04
2,1003,Tablet,10,PLACED,2025-12-02
3,1005,Smartwatch,12,SHIPPED,2025-11-14
4,1001,Laptop,12,SHIPPED,2025-12-03
5,1001,Laptop,18,DELIVERED,2025-11-20
6,1005,Smartwatch,9,PLACED,2025-11-13
7,1004,Headphones,7,DELIVERED,2025-11-22
8,1003,Tablet,18,SHIPPED,2025-12-06
9,1001,Laptop,10,DELIVERED,2025-11-10
10,1002,Smartphone,7,SHIPPED,2025-11-18
```

Körber

Liquibase Instructions

- Create Liquibase changelog files to define schema and load CSV data using <loadData> tag.

- Include db.changelog-master.xml and reference individual changelog files for table creation and data loading.

- Ensure Liquibase runs automatically at application startup.

Submission Guidelines

- Create a new GitHub repository

- Push both microservices as separate modules or folders within the same repository.

- Include a README.md file with:

- Project setup instructions
- API documentation
- Testing instructions

- Ensure the project builds and runs using Maven or Gradle.

- Java Version: Minimum Java 8, preferred Java 17.

- Automate data loading into H2 database during application startup using Liquibase changelogs.

- Provide sample API response formats in JSON for clarity.

Example for

GET /inventory/{productId}:

```
{
  "productId": 1001,
  "productName": "Laptop",
  "batches": [
    {
      "batchId": 1,
      "quantity": 50,
      "expiryDate": "2025-12-31"
    },
    {
      "batchId": 2,
      "quantity": 30,
      "expiryDate": "2026-03-15"
    }
  ]
}
```

Körber

POST /order : Request

```
{
 "productId": 1002,
 "quantity": 3
}
```

Response

```
{
 "orderId": 5012,
 "productId": 1002,
 "productName": "Smartphone",
 "quantity": 3,
 "status": "PLACED",
 "reservedFromBatchIds": [3],
 "message": "Order placed. Inventory reserved."
}
```

What we expect to see and what we don't

We expect you to:

- Fulfil the exercise expectations. We want to see if you can assimilate perfectly a task description and stick perfectly to the requirements. It's better for you to clarify any doubt but delivering something that either does not work or is not exactly what it has been asked. Asking is something you will do when working at Koerber, so asking during this code challenge is considered normal.
- Write clear and well-structured code. You won't work alone. It should be easy to understand and modify your code.
- Spend no more than 4 hours. We estimate that this exercise could be done in 4 hours. You already have things to do in your own time and we don't want this exercise to alter too much your life.
- Send working instructions. Do you imagine you buy a product that you need to install by yourself and when you follow the instructions provided, it does not work? It would be frustrating.
- Use technologies you know. This is not time to test something new unless you are really sure about it. Use technologies you already know and have experience with. Choose whatever language you want, services, databases, ... Remember, we just need to be able to follow setup instructions.

Körber

We don't expect you to:

- Be a craftsman. We don't expect to see the most optimized code. Be pragmatic with the time you have and the things you need to do.
- To write all the functionality from scratch. You are free to use any framework or library both in backend and frontend that helps you implement a functionality that is already maintained by someone in the community. Make sure you use libraries that work. We may have the chance to talk about your choices in the next chat we have.
- Implement a production service. Forget about authentication, users, logging, SSL, ... Just focus on the functionality described above.
- Implement the most performant solution. We don't care if data load is not super-fast or the milliseconds to answer. Again, be pragmatic. Just think of things that should be done in a different way with more time, and we will chat about it.