**Case Study: Coding Screening Using Pharmacy Claims Processing App**

**Background**

To assess engineering candidates’ ability to design and implement scalable, production-grade systems, we introduced a coding screening challenge. The challenge centers around **building a real-world pharmacy claims processing system** using modern Java and Spring technologies, designed to run on Kubernetes.

**The Coding Challenge**

**Problem Statement**

Candidates are asked to create an application that:

* Accepts pharmacy claims containing:
  + **Patient Information**
  + **Claim Information**
  + **Claim Cost**
  + **Insurance Coverage**
* Monitors a queue/topic where live claims are routed lively.
* For each claim:
  + Applies business rules using any open source rule engine (e.g. pharmacy ID, customer ID, type of claim, insurance policy).
  + Stores claims in a relational database.
  + Publishes the processed claim to an internal queue/topic.
  + Marks the claim status (e.g. PROCESSED, ERROR) in the DB.
* Handles **erroneous claims** by:
  + Storing them in a separate table.
  + Returning them to the queue/topic for retry (with configurable retry count).

**Bonus Tasks**

* Generate **daily report** showing:
  + Number of claims received.
  + Number of claims processed.
* Generate **invoice-style report**:
  + Count of pharmacies to be paid and their total claim amounts.

**Technical Requirements**

✅ **Tech stack:**

* Java 21
* Spring Boot 3.x (latest)
* Spring Kafka (or RabbitMQ based on choice)
* Spring Data JPA (PostgreSQL or MySQL)
* Spring Scheduling (for reports)

✅ **DevOps:**

* Build: Maven or Gradle
* Container: Docker
* Deployment: Kubernetes-ready manifests (Deployments, Services, ConfigMaps)
* Configurable retry count (via application.yml or ConfigMap)

✅ **Code Quality:**

* Must demonstrate:
  + Strong **object-oriented design** (e.g. domain models, service layers, repository pattern)
  + Proper exception handling
  + Logging (SLF4J)
  + Unit tests (JUnit + Mockito)

✅ **Code Sharing:**

* Solution should be hosted in a **public GitHub repository**
* README must contain:
  + Build and run instructions (including Docker & Kubernetes deployment steps)
  + Example API calls or Kafka message formats

**Expected Design Elements**

* Claim, Patient, Insurance, Pharmacy domain objects
* ClaimService, ClaimProcessor, ClaimRetryHandler services
* ClaimRepository, ErrorClaimRepository
* Kafka listener to consume claims
* Kafka producer to publish processed claims
* Scheduled job for report generation

**Evaluation Criteria**

| **Aspect** | **What We Look For** |
| --- | --- |
| **Code Structure** | Clean, modular, testable code; appropriate use of Spring features |
| **Object-Oriented Design** | Encapsulation, abstraction, meaningful domain models |
| **Error Handling** | Robust handling of failures, retry logic |
| **Scalability** | Designed with microservice/Kubernetes readiness |
| **Documentation** | Clear, buildable, deployable solution; good README |
| **Bonus** | Implementation of reporting features |

**Outcome**

* The use of AI tools to complete any portion of this case study will result in disqualification
* Candidates submit a link to their **public GitHub repository**.
* Hiring team reviews:
  + **Solution completeness**
  + **Code quality**
  + **Design patterns**
  + **Deployment readiness**
* Successful candidates proceed to a **live code review session** where they explain their design choices.

**Deadline:**

* **07/03/2025 6:00 PM ET.**