Qualité de Développement

Travaux Pratiques

Juliette ROUSSELET, Anastasiia KONONENKO, Hugo SAINSON

Groupe 33A

Exercice 1

Tâche 1 : Ségrégation des responsabilités

1. Quels sont les principaux domaines métiers de l'application Order flow ?

Les principaux domaines métier visés par l'application sont :

- Shopping en ligne
- Panier d'achat
- Traitement des commandes

D'autres domaines sont présents même s'ils ne sont pas principaux :

- Registre des produits
- Catalogue de produits
- Gestion des stocks
- Gestion des clients
- Notification
- Sourcing d'événements
- Monnaie

2. Comment les microservices sont-ils conçus pour implémenter les domaines métiers ?

Ils sont conçus de manière à ce que chaque domaine métier soit indépendant.Ainsi, les mises à jour et les problèmes techniques n'impactent pas les autres domaines métiers.

L'application order flow est composée de plusieurs microservices implémentant les domaines selon un modèle CQRS et piloté par les événements.

3. Quelles sont les responsabilités des conteneurs de code

`apps/of-api-gateway`, `apps/of-product-registry-microservices/product.registry`, `apps/of-product-registry-microservices/product.registry.read`, `libs/event-sourcing`, `libs/published-language` ?

Chacun des conteneurs de code permet l'implémentation d'un des microservices.

Les conteneurs du dossier apps/ permettent l'intégration des microservices alors que ceux contenus dans le dossier libs/ servent au bon fonctionnement des microservices.

Ces derniers ne sont que des librairies codées ou importées pour permettre à l'application d'être fonctionnelle indépendamment.

Tâche 2 : Questions

1. Quels sont les concepts principaux utilisés dans l'application Order flow ?

Les principaux concepts utilisés sont le DDD, pour Domain-Driven Design et aussi le EDA, pour Event Driven Architecture.

Le DDD est une méthode de conception qui aligne le code sur la logique métier d'une application.

Le EDA, quant à lui, est un modèle de conception de logiciels dans lequel les composants interagissent en émettant et en réagissant à des événements.

2. Comment les concepts principaux sont-ils implémentés dans les microservices ?

L'application Order Flow implémente les concepts principaux en microservices comme suit :

Architecture microservices : Chaque microservice gère un domaine spécifique (panier, commandes, clients), facilitant la maintenance.

Conception pilotée par le métier (DDD) : Les services sont structurés autour de domaines métiers, alignés aux besoins de la plateforme e-commerce.

Sourcing d'événements et communication asynchrone : Les changements sont enregistrés sous forme d'événements (MongoDB) et échangés via Apache Pulsar pour une communication flexible et découplée.

CQRS : Les opérations de lecture et d'écriture sont séparées, optimisant les performances et permettant une cohérence éventuelle entre les microservices.

3. Que fait la bibliothèque libs/event-sourcing ? Comment est-elle utilisée dans les microservices (relation entre métier et structure du code) ?

Elle gère les événements. Elle est étendue pour gérer les évènements particuliers de l'application.

4. Comment l'implémentation actuelle de l'event-sourcing assure-t-elle la fiabilité des états internes de l'application ?

La librairie event-sourcing créée une classe abstraite dont les enfants sont implémentés puis utilisés dans les microservices.

Tâche 3 : Identifier les problèmes de qualité

Fichier Dockerfile

Remove cache after installing packages or store it in a cache mount.
Merge this RUN instruction with the consecutive ones. [+19 locations]
lignes 8/51 : Sort these package names alphanumerically.
ligne 24/36 : Replace this invocation of curl with the ADD instruction.

Fichier ProductRegistryEventEmitter.java

ligne 1 : Missing mandatory Classpath entries. Resolve Project Problems.
ligne 14 : Unused Import
ligne 32 : Remove this field injection and use constructor injection instead.
ligne 93:
Remove the parentheses around the "producer" parameter (sonar.java.source not set. Assuming 8 or greater.)
Remove useless curly braces around statement (sonar.java.source not set. Assuming 8 or greater.)
ligne 101 : Define and throw a dedicated exception instead of using a generic one.
ligne 103 : cannot find symbol
    symbol: variable Log
    location: class ProductRegistryEventEmitter
ligne 106 : cannot find symbol
    symbol: class PulsarClientException
    location: class ProductRegistryEventEmitter
ligne 107 : Define and throw a dedicated exception instead of using a generic one.
ligne 130 : Define and throw a dedicated exception instead of using a generic one.

Fichier EventStore.java

ligne 1 : Missing mandatory Classpath entries. Resolve Project Problems.

Fichier ProductRegistryProjector.java

ligne 31 : Remove this field injection and use constructor injection instead.
ligne 65 :
Complete the task associated to this TODO comment.
TODO: Log an error

Fichier ProductRegistryEventConsumer.java

ligne 14 : Remove this field injection and use constructor injection instead.
ligne 22 : Complete the task associated to this TODO comment.

Fichier ProductRegistryService.java

ligne 38 : Remove this field injection and use constructor injection instead.
ligne 44 : Remove this field injection and use constructor injection instead.
ligne 85 : Complete the task associated to this TODO comment.

Fichier ProductService.java

ligne 24 : Remove this field injection and use constructor injection instead.

Fichier RegisterProductCommandDtoMapperTest.java

ligne 9/12 : Remove this 'public' modifier.

Problèmes:

Il y a des exceptions trop générales, des fonctions/classes publiques alors qu'elles ne devraient pas, des injections, des fonctionnalités qui ne sont pas finies (TODOs), des imports inutilisés,...

Exercice 2

Tâche 1 & 2 : Compléter les commentaires et la Javadoc et Corriger les RuntimeException paresseuses

Lien du dépot github du groupe :
https://github.com/Norikokonut/quali-dev

Tâche 3 : Convertir les payloads des Entités (couche de persistance) en records

ProductRegisteredEventEntity.java

```
public static record Payload{
        String productId,
        String name,
        String productDescription
}
```

ProductRemovedEventEntity.java

```
public static record Payload{
        String productId
}
```

ProductUpdatedEventEntity.java

```
public static record Payload{
        String productId,
        String name,
        String productDescription
}
```

Tâche 4 : Modifier les Entités (couche de persistance) pour utiliser des champs privés avec des accesseurs

```
/**
     * The payload for the event.
     */
    private Payload payload;

    /**
     * Getter for the payload.
     *
     * @return the payload
     */
    public Payload getPayload() {
        return payload;
    }

    /**
     * Setter for the payload.
     *
     * @param payload the payload to set
     */
    public void setPayload(Payload payload) {
        this.payload = payload;
    }

    /**
     * Payload for the event.
     * This is now a record.
     */
    public static record Payload(
        String productId,
        String name,
        String productDescription
    ) {}
}
```

Exercice 3

Tâche 1 : Modèle de contrôle de version

Pour ce qui est du contrôle de version et la collaboration entre deux équipes, la meilleure solution est l'utilisation de Git dans lequel chacune des équipes fait un fork de l'application. Pour cela, chaque équipe code ses fonctionnalités de son côté puis les merge au projet principal une fois celles-ci implémentées.

Pour ce faire, chaque fonctionnalité doit être indépendante et fonctionner même si l'une des API est dysfonctionnelle.

Tâche 2 : Responsabilités des équipes

Pour définir la responsabilité du code de chaque équipe, on peut faire en sorte que les différents modules de l'application aux différentes équipes pour ne pas qu'elles empiète sur les codes les unes les autres.

Tâche 3 : README et règles de collaboration

fichier CONTRIBUTING.md:
communication entre les équipes:
comment travailler ensemble:

Tâche 4 : Divisez votre groupe en 2 équipes

Pour la répartition des rôles au sein de l'équipe, il faut qu'une personne d'occupe de la gestion du projet en tant que manager ou au minimum responsable. Ce responsable de projet doit s'occuper de donner au développeurs des tâches à faire ainsi que de vérifier le bon fonctionnement des fonctionnalités implémentés à travers les pipelines git.

Les autres membres du groupe s'occupent du développement de l'application, de l'implémentation des tests. Ils répondent donc aux exigences du manager. L'un des deux développeurs va se concentrer sur la mise en place de multiples tests (End 2 End, Unitaire, Régression, Sécurité, etc.) tandis que l'autre va développer les fonctions et les programmes permettant de réaliser la tâche exigée.