UQAM

RAPPORT 1ER LIVRABLE

ERIC GAGNON - GAGE22037705

JIMMY PELLETIER - PELJ29088708

MARC-ANDRÉ BOUDREAULT - BOUM04086809

TRAVAIL PRÉSENTÉ

À

SEBASTIEN MOSSER

MGL7460

RÉALISATION ET MAINTENANCE DE LOGICIELS

19 MARS 2020

Ce rapport présente l'état de situation du projet technique en version V1, ainsi que les améliorations que nous envisageons pour livrer un logiciel de gestion de produits bancaires viable.

Description du projet :

Comme l'exige la spécification nous avons trois modules : bank, client et employee.

Tous les modules ont été découpés en architecture 3 couches : présentation, domaine et infrastructure. La couche présentation est ce qu'on expose pour communiquer avec notre domaine. La couche domaine contient notre logique d'affaires et la couche infrastructure contient tout ce qui est services techniques permettant d'accéder aux données et à la configuration technique du module.

Chacun des modules utilise le framework de test JUnit et Mockito permettant d'effectuer les essais unitaires automatisées. Chaque module a son propre fichier « pom » avec sa configuration nécessaire pour générer le rapport JaCoCo utile pour le serveur d'analyse statique SonarQube.

Le module bank a été développé avec l'aide du framework Spring Boot et Spring Data JPA. Nous avons décidé d'exposer des services REST permettant d'effectuer les diverses opérations de gestion de produits bancaires permettant d'interagir avec les modules de client et employee.

Les modules client et employee ont été développés à l'aide de Apache Commons CLI qui nous permet d'effectuer la gestion d'arguments reçus par ligne de commande. De plus, ces modules sont responsables de faire la gestion des appels http au serveur de la banque.

Finalement, des scripts Bash ont été utilisés pour encapsuler les appels aux .jar.

Forces du projet :

1. Mise en place de pipelines et processus

Nous avons mis en place un pipeline de CI assez complet pour une version MVP. Ce pipeline est divisé en deux parties:

1. Pipeline de master
2. Pipeline de merge request

La branche master est une branche protégée et en ce sens, il est impossible de pousser directement dans cette branche. Les développeurs doivent faire une demande de merge request sur Gitlab pour pouvoir débuter l'intégration de la nouvelle fonctionnalité dans master. C'est ici que le pipeline de merge request débute. Le merge request a été configuré pour qu'il soit impossible d'accepter un merge si toutes ces conditions sont vraies:

1. La branches du merge request doit être en fast-forward avec master. De cette façon nous sommes certains que les validations de qualités inclues les derniers développements de master (évite les régressions).
2. Le pipeline de merge request doit être en succès. Ce pipeline exécute le build, les tests et l'analyse statique du code avec SonarQube. Le gating de SonarQube gratuit nécessite une cote

de A pour la Reliability, Security et Maintainability, de 80% de couverture de code ainsi que 3% de duplication maximum.
3. Avoir résolu tous les commentaires du code review effectués par les pairs.

Lorsque le merge request est accepté, il sera fusionné dans la branche master. À ce moment, il y aura encore une fois un build, test et analyse statique de fait. Il va nous rester à créer les artefacts de release et déploiement continue.

2. Utilisation de Docker

Nous avons utilisé le container docker MySql pour notre serveur de base de données. Cette décision a facilité la mise en place de la base de données et nous avons pu ajouter la persistance rapidement dans notre application avec ce choix. Aussi l'utilisation de docker-compose pour lancer ce container a beaucoup simplifié le setup initial pour tous les utilisateurs du système.

3. Architecture DDD

Nous avons misé sur une architecture qui s'inspire du DDD (Domain Driven Design). Le domaine n'ayant aucune dépendance sur les autres couches du système est facilement testable et réutilisable. Les autres couches tel que l'infrastructure dépendent seulement du domaine et l'API dépend seulement du domaine et utilise l'infrastructure par injection de dépendances.

4. Traçabilité du code avec les user stories

Chaque commit effectué dans le repository est lié à un user story (issue dans GitLab). Ceci est fait en ajoutant par exemple "Ref #1" dans le message de commit pour que GitLab soit en mesure de lier l'issue #1 avec le commit.

Faiblesses du projet :

1. Image Docker non publié sur DockerHub

L'image Docker utilisée pour la persistance des données demande une préparation manuelle afin de créer les données. L'installation nécessite des manipulations qui peuvent engendrer des erreurs. Pour la version finale, nous envisageons la création d'une image et la publication de celle-ci sur le DockerHub. Les données de pilotage de l'application, tels les produits seront aussi inclus dans cette image puisqu'il n'existe pas de fonctionnalités d'ajout de produit.

2. Gestion de dépendances non centralisées

Notre gestion de dépendances Maven est aussi déficiente et peut porter à confusion, car les versions sont répétées inutilement dans le fichier de configuration (pom) de chaque module. La centralisation vers le pom parent sera la solution. Nous allons ajouter une section dependencies management dans le pom parent permettant de gérer les versions utilisées dans notre logiciel.

3. Gestion d'arguments non robuste

Dans le module client, la gestion des arguments reçu en invite de commandes n'est pas robuste. Une amélioration est à faire à ce niveau pour retourner des messages d'erreurs claires et non une erreur technique non gérée.

4. Manque d'utilisation de design pattern

Dans les modules client et employee, nous avons une classe « CommandLineProcessor » permettant d'effectuer l'opération reçu. Un design pattern est à prévoir pour être en mesure de gérer toutes les opérations possibles au lieu d'un bloc de if/else statement. Un exemple de design pattern à utiliser serait « Strategy ».

5. Compatibilité d'Apache CLI & Windows

Nous avons découvert une incapacité pour l'api Apache Commons Cli sous Windows qui est incapable de discerner les arguments sans signe d'égalité pour les paramètres de format long (ceux qui sont représentés par un "--"). Par exemple, la librairie n'arrive pas à distinguer le paramètre suivant: "--list CLIENT_NAME" tout simplement à cause de l'espace entre les mots. L'idée est donc de boucler dans les paramètres lors de la lecture des arguments dans le script Bash et d'ajouter le signe "égal" (=). Actuellement, si d'autres paramètres sont ajoutés à la ligne de commande, on ne les passe tout simplement pas au Java, ce qui enfreint l'intégrité puisque ces règles doivent être au niveau du domaine dans les applications Java concernée.