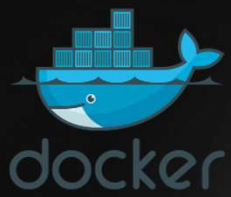
Parcours développeur d'application Java

Etudiant : Grégory Gautier

Mentor : Souleymane Sanogo

Projet n° 09
Testez vos développements Java

Support de soutenance – 22.07.2020



Sommaire

Présentation du projet

Présentation du projet

Contexte

L'équipe de développement est en train de réaliser un système de facturation et de comptabilité pour un client. Le développement a débuté depuis quelques temps et l'équipe doit commencer à vérifier que l'application fonctionne correctement, qu'elle répond bien aux règles de gestion et les respecte.

Objectifs

Réalisations des tests

- Tests unitaires : validation des règles de gestion unitaires de chaque composant de l'application
- Tests d'intégration : validation de la bonne interaction entre les différents composants de l'application

Présentation du projet

Objectifs

Correction des erreurs

L'objectif est de corriger toutes les erreurs qui ont été volontairement disséminées dans le code de l'application.

Amélioration et refactoring du code

L'objectif est d'améliorer le code et de faire du refactoring pour augmenter la maintenabilité et la lisibilité du code et pour faciliter le débogage de l'application. Par exemple :

- tracer les requêtes SQL exécutées
- compléter les messages des exceptions (détail des violations de contraintes...)
- tracer les informations de règles de gestion violées avec leur identifiants associés dans le dossier de spécifications

Présentation du projet

Objectifs

Complément d'implémentation

L'objectif est de compléter l'implémentation de l'application en s'appuyant sur les commentaires TODO insérés dans le code source.

Prérequis

- Développer une application JEE
- Configurer le frawmework Spring avec XML
- Mettre en œuvre et utiliser une base de données
- Testez votre code Java pour réaliser des applications de qualité

Cahier des charges

Cahier des charges

Règles de gestion

RG_Compta_1	Le solde d'un compte comptable est égal à la somme des montants au débit des lignes d'écriture diminuées de la somme des montants au crédit. Si le résultat est positif, le solde est dit "débiteur", si le résultat est négatif le solde est dit "créditeur".
RG_Compta_2	Pour qu'une écriture comptable soit valide, elle doit être équilibrée : la somme des montants au crédit des lignes d'écriture doit être égale à la somme des montants au débit.
RG_Compta_3	Une écriture comptable doit contenir au moins deux lignes d'écriture : une au débit et une au crédit.
RG_Compta_4	Les montants des lignes d'écriture sont signés et peuvent prendre des valeurs négatives (même si cela est peu fréquent).

Cahier des charges

Règles de gestion

RG_Compta_5 La référence d'une écriture comptable est composée du code du journal dans lequel figure l'écriture suivi de l'année et d'un numéro de séquence (propre à chaque journal) sur 5 chiffres incrémenté automatiquement à chaque écriture. Le formatage de la référence est : XX-AAAA/#####.
Ex : Journal de banque (BQ), écriture au 31/12/2016
--> BQ-2016/00001

RG_Compta_6 La référence d'une écriture comptable doit être unique, il n'est pas possible de créer plusieurs écritures ayant la même référence.

RG_Compta_7 Les montants des lignes d'écritures peuvent comporter 2 chiffres maximum après la virgule.

Cahier des charges

Contraintes de réalisation

- Les tests sont implémentés et automatisés à l'aide de JUnit, Mockito, Maven et Travis CI / GitLab CI / Jenkins.
- Les tests sont lancés via le build Maven.
- Les tests d'intégration font l'objet de profils Maven spécifiques (fichier `pom.xml` du dossier parent).
- Cet environnement est construit (à partir des éléments disponibles dans le dépôt Git du projet) et monté à la volée par le système d'intégration continue.
- Tous les commentaires TODO doivent être réalisés.
- L'implémentation des commentaires TODO doit être testée par des tests unitaires et d'intégration.

Cahier des charges

Référentiel d'évaluation

- Les 4 erreurs de développements dans le projet fourni sont identifiées et résolues.
- Le développement a été complété en suivant les TODO.
- Les tests unitaires ont été réalisés à l'aide de JUnit.
- Les tests d'intégration ont été réalisés à l'aide des "profiles" Maven.
- L'ensemble des modules a un code coverage de 75% minimum.
- Un serveur d'intégration est installé et configuré (Jenkins / Travis CI / GitLab CI au choix).
- Un rapport d'exécution des tests est automatiquement généré à chaque commit.
- Un logiciel de versionning a été correctement utilisé.

Démonstration

Implémentation des tests

Stratégie de test d'intégration

Stratégie de test d'intégration

- Les tests d'intégration sont implémentés dans un répertoire séparé, en dehors de l'arborescence Maven.
- Les répertoires des sources et des ressources des tests d'intégration sont ajoutés lors du build Maven.
- Les tests d'intégration sont lancés via des profiles Maven.
- Les tests unitaires et les tests d'intégration sont lancés par défaut.
- Les tests d'intégration sont lancés via la commande Maven "verify" (plugin failsafe).

Système d'intégration continue

Système d'intégration continue

- Le système d'intégration continue est Travis CI.
- Travis CI est automatiquement exécuté à chaque commit à partir des éléments du dépôt Git.
- Un rapport d'exécution des tests est automatiquement généré par JaCoCo (via le build Maven) à chaque commit à partir des éléments du dépôt Git.

Système de qualité continue

Système de qualité continue

- Le système de qualité continue est SonarCloud.
- Les rapports de SonarCloud sont automatiquement mis à jour après l'exécution du système d'intégration continue Travis CI.
- Le rapport code coverage de SonarCloud s'appuie sur les rapports XML de JaCoCo (agrégation des rapports des test unitaires et d'intégration de chaque module Maven).

Erreurs de développement

Erreurs de développement

4 erreurs de développement ont été identifiées grâce à la réalisation des tests unitaires et d'intégration :

1. EcritureComptable.getTotalCredit()

```
public BigDecimal getTotalCredit() {
        BigDecimal vRetour = BigDecimal.ZERO;
        for (LigneEcritureComptable vLigneEcritureComptable : listLigneEcriture) {
                if (vLigneEcritureComptable.getDebitgetCredit() != null) {
                        vRetour = vRetour.add(vLigneEcritureComptable.getDebitgetCredit());
                }
        }
        return vRetour;
}
```

Erreurs de développement

2. EcritureComptable.reference

```
@Pattern(regexp = "\\dD{1,5}-\\d{4}/\\d{5}")
private String reference;
```

3. sqlContext.xml

```
<property name="SQLinsertListLigneEcritureComptable">
    <value>
        INSERT INTO myerp.ligne_ecriture_comptable (
        ecriture_id,
        ligne_id, compte_comptable_numero, libelle, debit,
        credit
        ) VALUES (
        :ecriture_id, :ligne_id, :compte_comptable_numero, :libelle, :debit,
        :credit
        )
    </value>
</property>
```

Virgule manquante

Erreurs de développement

4. ComptabiliteManagerImpl.updateEcritureComptable(EcritureComptable)

```
public void updateEcritureComptable(EcritureComptable pEcritureComptable) throws FunctionalException {
        LOGGER.trace(new EntreeMessage());
        checkEcritureComptable(pEcritureComptable);
        TransactionStatus vTS = getTransactionManager().beginTransactionMyERP();
        try {
                getDaoProxy().getComptabiliteDao().updateEcritureComptable(pEcritureComptable);
                getTransactionManager().commitMyERP(vTS);
                vTS = null;
        } finally {
                getTransactionManager().rollbackMyERP(vTS);
        }
        LOGGER.trace(new SortieMessage());
}
```

Implémentation des TODO

Implémentation des TODO

Les commentaires TODO suivants insérés dans le code source ont tous été réalisés :

- ✓ **TODO à implémenter**
  ComptabiliteManagerImpl.addReference(EcritureComptable)
- ✓ **TODO à tester**
  ComptabiliteManagerImpl.addReference(EcritureComptable)
- ✓ **TODO à tester**
  ComptabiliteManagerImpl.checkEcritureComptable(EcritureComptable)
- ✓ **TODO tests à compléter**
  ComptabiliteManagerImpl.checkEcritureComptableUnit(EcritureComptable)
- ✓ **TODO RG_Compta_5**
  ComptabiliteManagerImpl.checkEcritureComptableUnit(EcritureComptable)
- ✓ **TODO à tester**
  EcritureComptable.getTotalDebit()

Livrable 01

Code source de l'application

```
...tendWith(MockitoExtension.class)
public class AbstractDbConsumerTest {

    @Spy
    private AbstractDbConsumer abstractDbConsumer;

    @Mock
    private DaoProxy daoProxy;
    @Mock
    private Map<DataSourcesEnum, DataSource> mapDataSource;
    @Mock
    private DataSource dataSource;
    @Mock
    private JdbcTemplate jdbcTemplate;

    // === getDaoProxy() ===

    @Test
    public void getDaoProxy_returnsDaoProxy() {
        // GIVEN
        ReflectionTestUtils.setField(ConsumerHelper.class, "daoProxy", daoProxy);

        // WHEN
        DaoProxy actualDaoProxy = AbstractDbConsumer.getDaoProxy();

        // THEN
        assertThat(actualDaoProxy).isEqualTo(daoProxy);
    }

    // === getDataSource(DataSourcesEnum) ===

    @ParameterizedTest
    @EnumSource(DataSourcesEnum.class)
    @NullSource
```

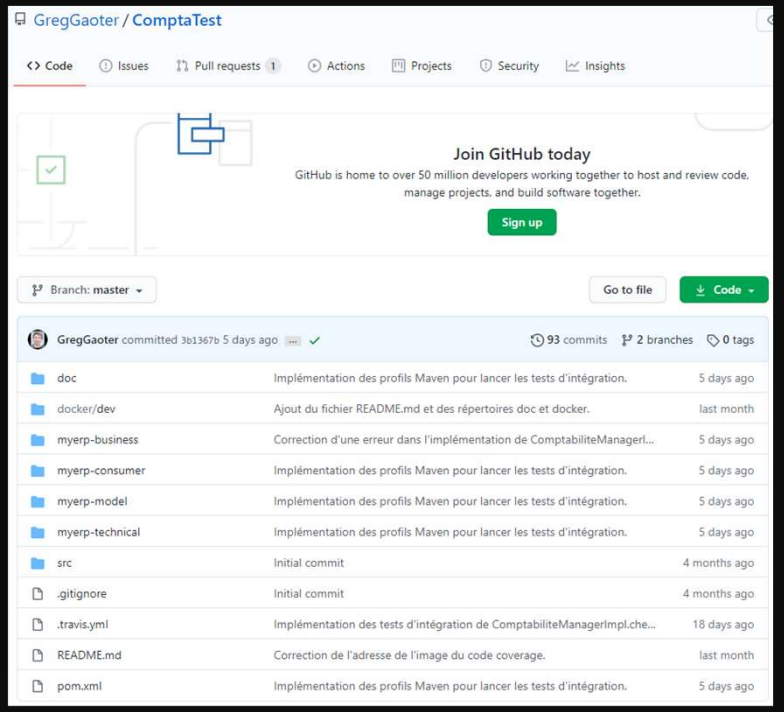
Livrable code source

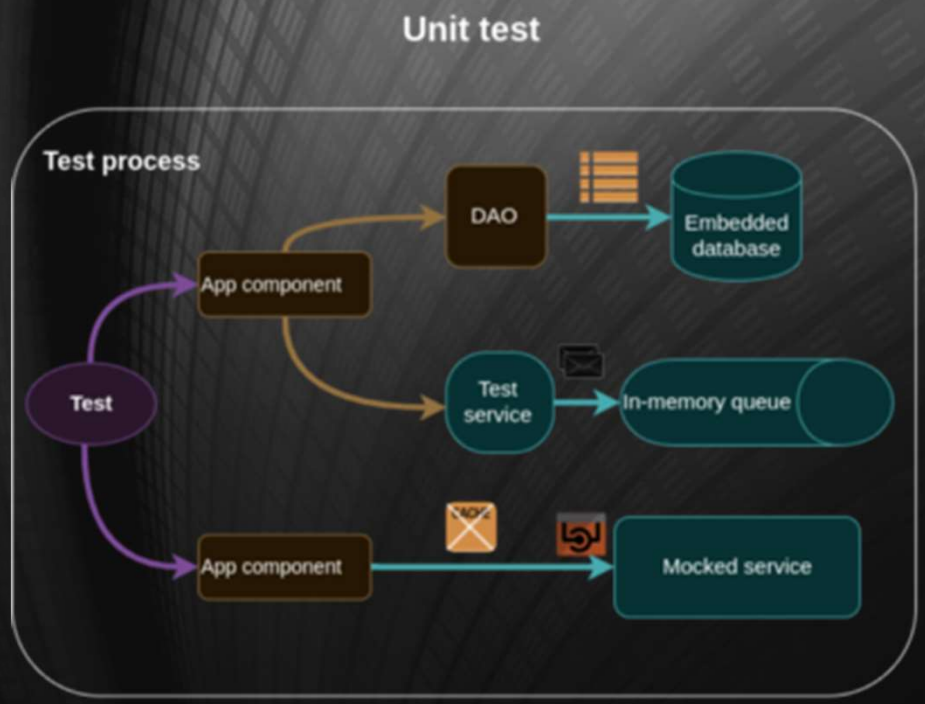
Objectif

Fournir dans un seul dépôt Git sur GitHub le code source de l'application.

Résultat

https://github.com/GregGaoter/ComptaTest/tree/master





Livrable 02

Eléments d'automatisation des tests unitaires

Eléments d'automatisation des tests unitaires

Objectif

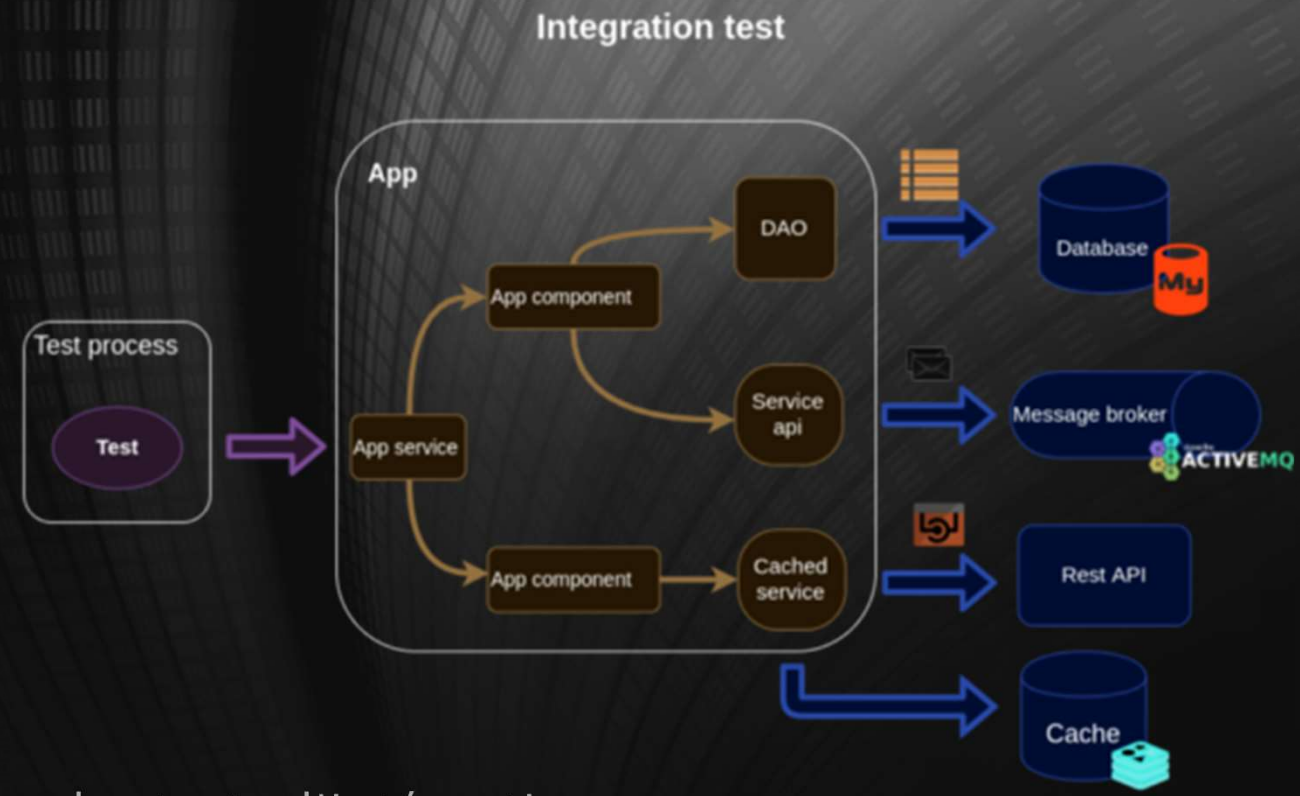
Fournir dans un seul dépôt Git sur GitHub les éléments d'automatisation des tests unitaires.

Résultat

Les éléments d'automatisation des tests unitaires se trouvent dans le fichier "`pom.xml`" à la racine du projet et dans les fichiers "`pom.xml`" à la racine des modules Maven.

Ces éléments d'automatisation sont synthétisés dans des fichiers XML dans le répertoire "`livrables/elements-automatisation-tests-unitaires`".

- myerp-pom.xml : synthèse du projet parent.
- myerp-business-pom.xml : synthèse du module business.
- myerp-consumer-pom.xml : synthèse du module consumer.
- myerp-model-pom.xml : synthèse du module model.
- myerp-technical-pom.xml : synthèse du module technical.



Livrable 03

Eléments d'automatisation des tests d'intégration

Eléments d'automatisation des tests d'intégration

Objectif

Fournir dans un seul dépôt Git sur GitHub les éléments d'automatisation des tests d'intégration.

Résultat

Les éléments d'automatisation des tests d'intégration se trouvent dans le fichier "`pom.xml`" à la racine du projet et dans les fichiers "`pom.xml`" à la racine des modules Maven.

Ces éléments d'automatisation sont synthétisés dans des fichiers XML dans le répertoire "`livrables/elements-automatisation-tests-integration`".

- myerp-pom.xml : synthèse du projet parent.
- myerp-business-pom.xml : synthèse du module business.
- myerp-consumer-pom.xml : synthèse du module consumer.
- myerp-model-pom.xml : synthèse du module model.
- myerp-technical-pom.xml : synthèse du module technical.

Livrable 04

Configuration du serveur d'intégration

Configuration du serveur d'intégration

Objectif

Fournir dans un seul dépôt Git sur GitHub la configuration du serveur d'intégration continue.

Résultat

La configuration du serveur d'intégration continue Travis CI se trouve dans le fichier ".travis.yml" à la racine du projet. Son contenu est :

```
addons:
  sonarcloud:
    organization: "greggaoter"

script:
  - mvn clean verify sonar:sonar -Dsonar.login=4b000c15eb49e902834b2f506a789f29dc91a32f -Dsonar.projectKey=GregGaoter_ComptaTest
```

Synthèse

Synthèse

Point demandé	Point réalisé
Réalisation des tests unitaires.	✓
Réalisation des tests d'intégration.	✓
Correction des 4 erreurs disséminées dans le code.	✓
Amélioration et refactoring du code source.	✓
Réalisation de tous les commentaires TODO.	✓
Automatisation des tests à l'aide de JUnit 5 et Mockito.	✓
Lancement des tests via le buid Maven.	✓
Création de profils Maven pour lancer les tests d'intégration.	✓
Construction de l'environnement de test à la volée par le système d'intégration continue Travis CI.	✓
Couverture de code par les tests de 75% minimum (85% de couverture réalisé).	✓
Génération automatique d'un rapport de test.	✓
Utilisation du logiciel de versioning Git.	✓

Conclusion

Conclusion

- Satisfait du travail accompli
- Confiant et motivé pour la suite du parcours
- Perspectives :
  - Tests des développements Java
  - Utilisation du framework JUnit
  - Intégration continue des développements
  - Qualité continue des développements
  - Améliorer l'application pour une futur version :
    - Refactoring du code
    - Amélioration de la qualité du code basée sur les rapports de SonarCloud


MERCI
Place aux questions et aux discussions