IUT de Bayonne et du Pays Basque – BUT Informatique

Semestre 4

Module R4.02 – Qualité de développement

Année universitaire 2022-23

Pantxika DAGORRET, Damien URRUTY

TP n°3 : Couverture de tests et automatisation – Sujet

Durée prévue en nombre de séances : 1

Préambule : Avez-vous des questions sur les enseignements dispensés précédemment ?

Objectif :

- Calculer la couverture du code par les tests
- Automatiser l'exécution des tests

Maintenant que le coeur de métier de l'application a été couvert par des tests unitaires, et avant de continuer sur le reste de l'application, faisons un état des lieux. Comment savoir ce qui a déjà été couvert et ce qu'il reste à couvrir ? Nous allons calculer la métrique de la couverture de code. Nous allons aussi automatiser l'exécution des tests, pour nous assurer que les tests déjà écrits soit exécutés régulièrement, au moins à chaque modification du code sur le dépôt. Ceci peut se faire grâce au mécanisme des GitHub Actions.

Travail à faire :

1. Dans IntelliJ, calculer la couverture de code en faisant clic droit sur src/test/java > More Run/Debug > Run 'Tests in PokeBagarre' with Coverage. Quels sont les différents indicateurs retournés ?
2. Lancer les tests en ligne de commande en faisant .\gradlew.bat test et s'assurer qu'ils passent
3. Construire le jar de l'application en faisant .\gradlew.bat build. Il se trouvera dans build/libs/
4. Dans une branche sur le dépôt local, introduire une GitHub action pour construire l'application sur chaque push sur le dépôt. Vous pouvez utiliser la commande précédente qui lancera aussi les tests. Instructions sur https://docs.github.com/fr/actions/automating-builds-and-tests/building-and-testing-java-with-gradle.
5. Pousser sur GitHub, ouvrir une Pull Request et vérifier le résultat de l'action dans les « checks » de la PR ou dans l'onglet Actions du projet. Faire des modifications si nécessaire. Merger la Pull Request lorsque le build passe

Pour aller plus loin :

1. Ajouter une étape dans le fichier yml pour utiliser l'action *test-reporter*. Elle permet de voir le résultat de l'exécution des tests sur GitHub. Pour la configurer, voir https://github.com/marketplace/actions/test-reporter

2. Ajouter le plugin JaCoCo dans le fichier build.gradle et faire échouer la construction du projet si un certain niveau de coverage n'est pas atteint. Ce niveau de coverage sera déterminé en fonction de ce que vous aurez observé à la question 1. Plus d'informations : https://docs.gradle.org/current/userguide/jacoco_plugin.html
3. Ajouter une étape dans le fichier yml pour utiliser l'action *JaCoCo Reporter*. Elle permet d'afficher la métrique de couverture du code par fichier et sur le projet au global. Pour la configurer, voir https://github.com/marketplace/actions/jacoco-reporter