AVIGNON UNIVERSITÉ

Rapport TP5
Documentation

Étudiant
Michel Marie LAMAH

15 avril 2023

Master Informatique
Intelligence Artificielle

UE Génie logiciel avancé
ECUE Techniques de test

Responsables
Daniel SALAS
Emmanuel FERREIRA

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Sommaire

1 Introduction

Dans cet TP[1], il sera question de mettre en pratique les notions de couverture de tests. Pour réaliser ce travail, j'utiliserai :

- **IDE** : Intellij IDEA [1]
- **Dépôt de repository Git** : Github [2]
- **Checkstyle** : pour la qualité du code
- **JavaDoc** : pour la documentation

Toutes les modifications réalisées ici se trouvent sur la branche **TP05** de mon repository [3].

Mais si vous souhaitez voir toutes les modifications effectuées depuis le TP1 jusqu'à maintenant vous pouvez voir la branche master [2].

2 Checkstyle

2.1 Configuration pom

Pour pouvoir utiliser checkstyle, il faut ajouter le plugin dans notre pom.xml.

```
<reporting>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-checkstyle-plugin</artifactId>
            <version>3.2.1</version>
        </plugin>
    </plugins>
</reporting>
```

Listing 1. Configuration checkstyle

Par la suite pour les bagdes il faudra aussi ajouter le plugin ci-dessous :

```
        <plugin>
            <groupId>com.github.bordertech.buildtools</groupId>
            <artifactId>badger</artifactId>
            <version>1.0.0</version>
            <executions>
                <execution>
                    <phase>site</phase>
                    <goals>
                        <goal>badges</goal>
                    </goals>
                    <configuration>
                        <outputDir>target/site/bagdes</outputDir>
                        <inputFiles>
                            <inputFile>target/checkstyle-result.xml</inputFile>
                        </inputFiles>
                    </configuration>
                </execution>
            </executions>
        </plugin>
```

Listing 2. Configuration badge checkstyle

Dans cette configuration on spécifie de générer les rapport lors de la phase **site**

1. https://www.jetbrains.com/fr-fr/idea/
2. https://github.com/

2.2 Configuration Readme

Dans le code Listing 2, le **ouputDir** indiquait où sera stocké le badge, du coup il nous suffit de spécifier celà dans notre readme.

```
![CheckStyle](target/site/bagdes/checkstyle-result.svg)
```

Listing 3. Intégration badge checkstyle dans le readme

2.3 Résultat

Comme résultat on a :

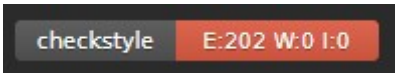


Figure 1. Badge checkstyle

Ici, on peut facilement constater que nous avons :

- **E** (Error) : 202 sérieux problèmes de standards de codage
- **W** (Warning) : 0 problème de maintenabilité
- **I** (Information) : 0 problème de lisibilité

Pour avoir plus d'informations sur ces erreurs, il suffit d'ouvrir le fichier **checkstyle.html** ci-dessous dans un navigateur :

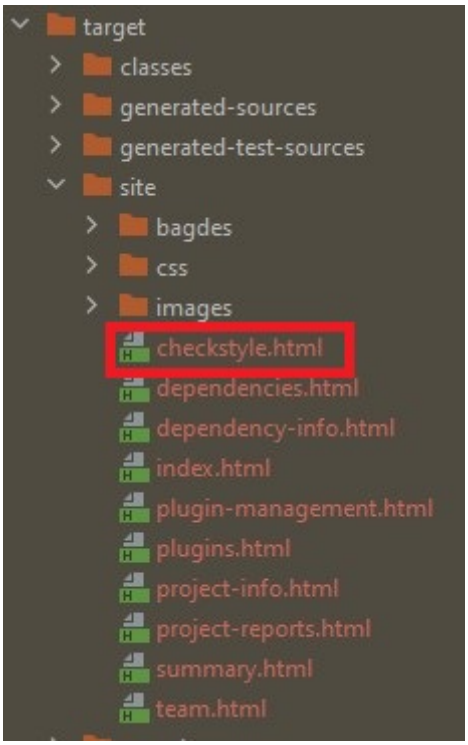
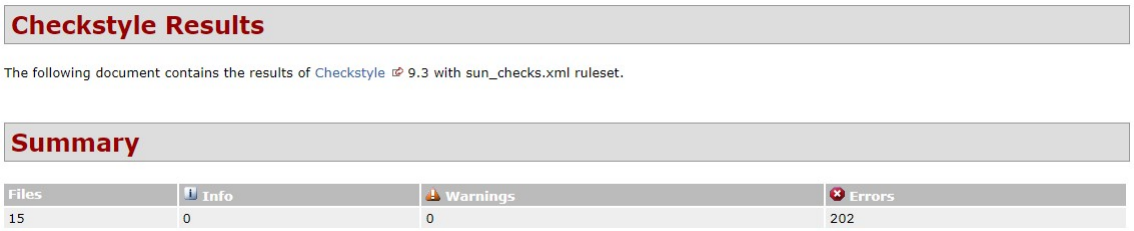


Figure 2. Fichier checkstyle.html

On aura alors :



Checkstyle Results

The following document contains the results of Checkstyle 9.3 with sun_checks.xml ruleset.

Summary

Files	Info	Warnings	Errors
15	0	0	202

Figure 3. Résultats dans un navigateur

En vérifiant la source de ces erreurs j'ai remarqué la majeur partie était à des problèmes d'indentation et de manque de documentation, chose que nous allons essayer d'améliorer une fois la documentation mise en place.

3 Documentation

3.1 Configuration pom

Pour pouvoir générer automatiquement la documentation, il faut ajouter le plugin javadoc dans le pom.xml

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-javadoc-plugin</artifactId>
    <version>3.5.0</version>
    <executions>
        <execution>
            <id>attach-javadocs</id>
            <goals>
                <goal>jar</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Listing 4. Configuration pom

Avec ce code pour générer la documentation, on pourra utiliser l'une des deux commandes ci-dessous :

```
mvn site
```

Console 1. Commande 1 Génération documentation

```
mvn javadoc:javadoc
```

Console 2. Commande 2 Génération documentation

3.2 Configuration Readme

Par la suite, il faut suite il faut ajouter le lien vers la documentation dans le fichier Readme, pour permettre aux autres d'y accéder

```
La documentation: https://MichelMarieLamah13.github.io/ceri-m1-techniques-de-test
```

Listing 5. Le lien de la documentation

3.3 Génération clé ssh

Pour la suite il faudra générer une clé ssh pour permettre à CircleCI, de publier dans github.

Pour générer une clé ssh sous windows, il suffit d'utiliser la commande puis de taper entre jusquà la fin.

```
ssh-keygen -t ed25519 -C "lamahmichelmarie@gmail.com"
```

Console 3. Création clé SSH

Cette commande générera deux clés, une clé publique et une autre privée.

3.4 Configuration Github

Dans Github nous allons ajouter la clé publique, pour cela dans le projet sur github, on clique sur **Settings**.

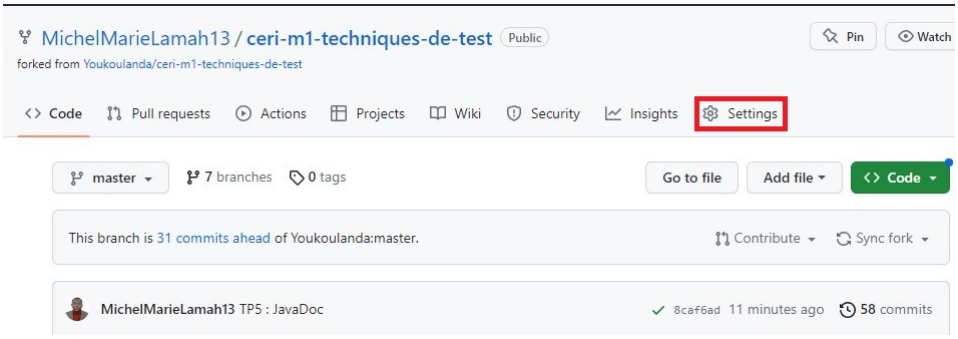


Figure 4. Settings

Dans la section **Sécurité** on clique sur **Deploy Keys**

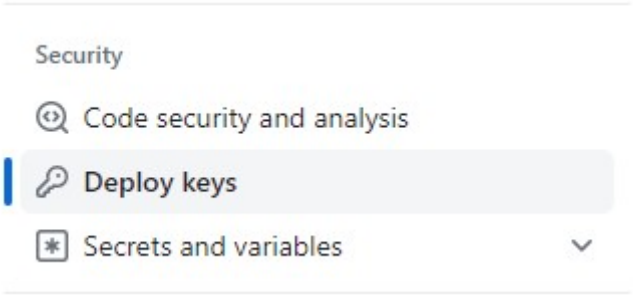


Figure 5. Deploy Keys

Puis on clique **Add Deploy Key**, pour ajouter la clé publique

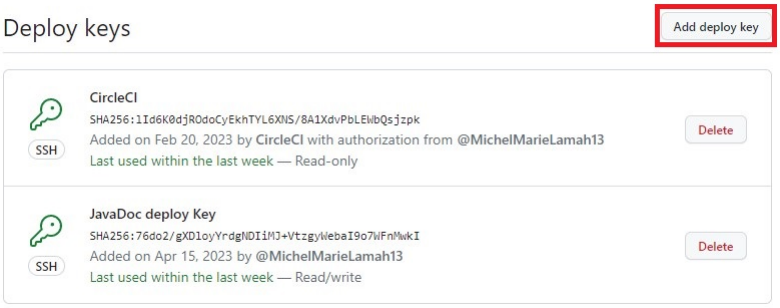


Figure 6. Add Deploy Key

3.5 Configuration CircleCI

Dans CircleCI, nous devons ajouter la clé privée et pour cela, on procède comme suit : Dans le projet sur CircleCI, on clique sur **Project Settings**

Figure 7. Project Settings

Par la suite, on clique sur **SSH Keys**

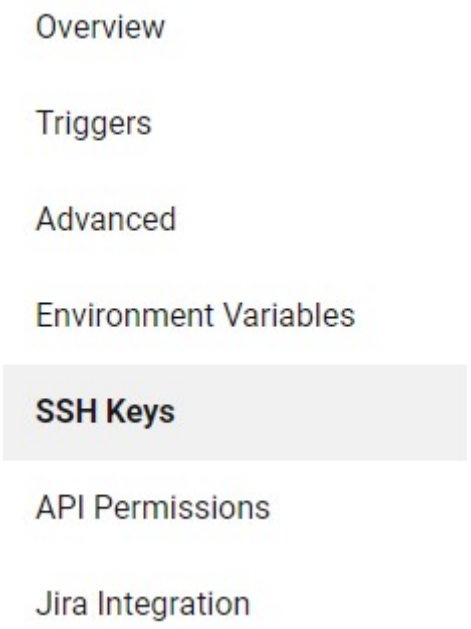


Figure 8. SSH Keys

Puis dans **Additional SSH Keys**, on ajoute la clé privée avec pour hostname **github.com**

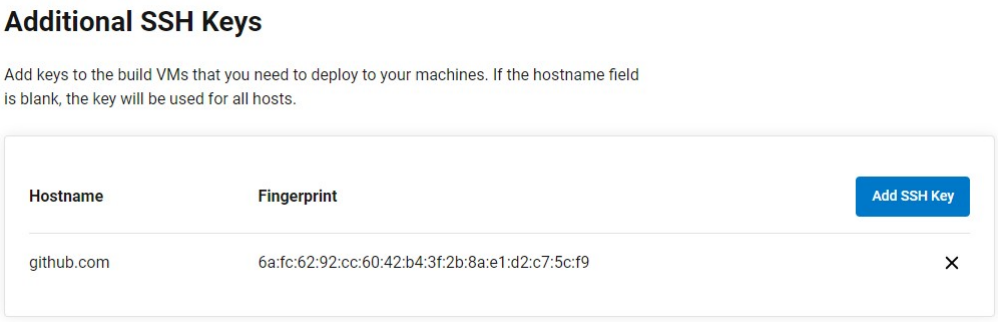


Figure 9. Additional SSH Keys

Par la suite il faudra aussi configurer le fichier **config.yml** du projet, en y ajoutant dans les **jobs** le code ci-dessous :

```
docs-deploy:
  docker:
    - image: node:8.10.0
  steps:
    - checkout
    - attach_workspace:
        at: target/site
    - run:
        name: Install and configure dependencies
        command: |
          npm install -g --silent gh-pages@2.0.1
          git config user.email "lamahmichelmarie@gmail.com"
          git config user.name "MichelMarieLamah13"
    - add_ssh_keys:
        fingerprints:
          - "6a:fc:62:92:cc:60:42:b4:3f:2b:8a:e1:d2:c7:5c:f9"
    - run:
        name: Deploy docs to gh-pages branch
        command: gh-pages --dist target/site/apidocs
```

Listing 6. Job Dépoloiement du Javadoc

Et dans les workflows :

```
    - docs-deploy:
        requires:
          - build-and-test
        filters:
            branches:
              only: master
```

Listing 7. Workflow Dépoloiement du Javadoc

3.6 Résultat

Il suffira de faire un push sur votre branch master et CircleCI, fera le reste du travail, comme on peut le voire ci-dessous.

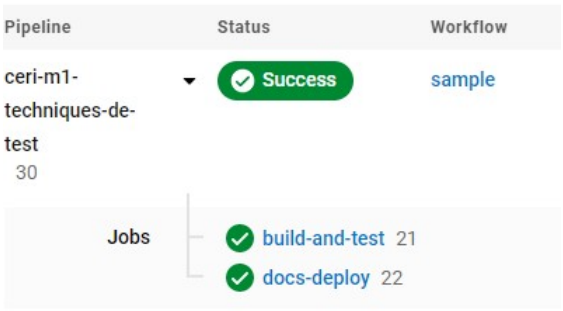


Figure 10. Résultat dans CircleCI

En cliquant sur le lien de la documentation, on obtient alors :

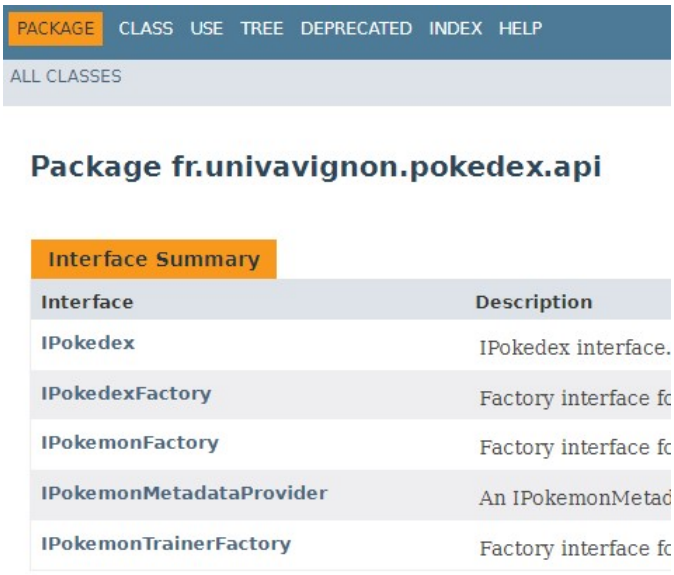


Figure 11. Résultat dans CircleCI

4 Conclusion

Dans cet tp, il était question de mesurer la qualité de notre code en utilisant l'outil **Checkstyle** mais aussi de générer automatiquement la documentation de notre code en utilisant **JavaDoc**.

Références

[1] Daniel SALAS. *TP5 : Documentation*. Université Avignon. 2023. url : https://github.com/Youkoulanda/ceri-m1-techniques-de-test/blob/master/TPs/TP5.md.

[2] LAMAH Michel Marie. *TP5 : Documentation [master]*. Université Avignon. 2023. url : https://github.com/MichelMarieLamah13/ceri-m1-techniques-de-test/tree/master.

[3] LAMAH Michel Marie. *TP5 : Documentation [TP05]*. Université Avignon. 2023. url : https://github.com/MichelMarieLamah13/ceri-m1-techniques-de-test/tree/TP05.