AVIGNON UNIVERSITÉ

Rapport TP4
Code Coverage

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]
- **Codecov** [3] : la plateforme pour calculer le pourcentage de couverture
- **Jacoco** : la bibliothèque pour générer les rapports de tests

Toutes les modifications réalisées ici se trouvent sur la branche **TP04** 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 Configurations

2.1 Connexion à codecov

Pour les configurations, la première étape consiste à créer un compte sur la plateforme codecov en utilisant votre compte github.

Une fois celà effectué, codecov détectera automatiquement les repositories qui se trouve dans votre github, ci-dessous le repository sur lequel je travail durant ces tps.

Figure 1. Répository sur codecov

2.2 Génération token dans codecov

Une fois notre github connecté à codecov, la prochaine étape consiste à configurer CircleCI l'outil d'intégration continu que nous avons intégré dans le TP2.

Pour celà on commence tout d'abord par générer un token pour le projet, du coup dans codecov, dans la liste des projets, il faudra cliquer sur le projet pour lequel vous voulez générer le code, dans notre cas c'est le projet **ceri-m1-techniques-de-test**, en cliquant dessus nous retrouvons sur la page ci-dessous

1. https://www.jetbrains.com/fr-fr/idea/
2. https://github.com/
3. https://about.codecov.io/

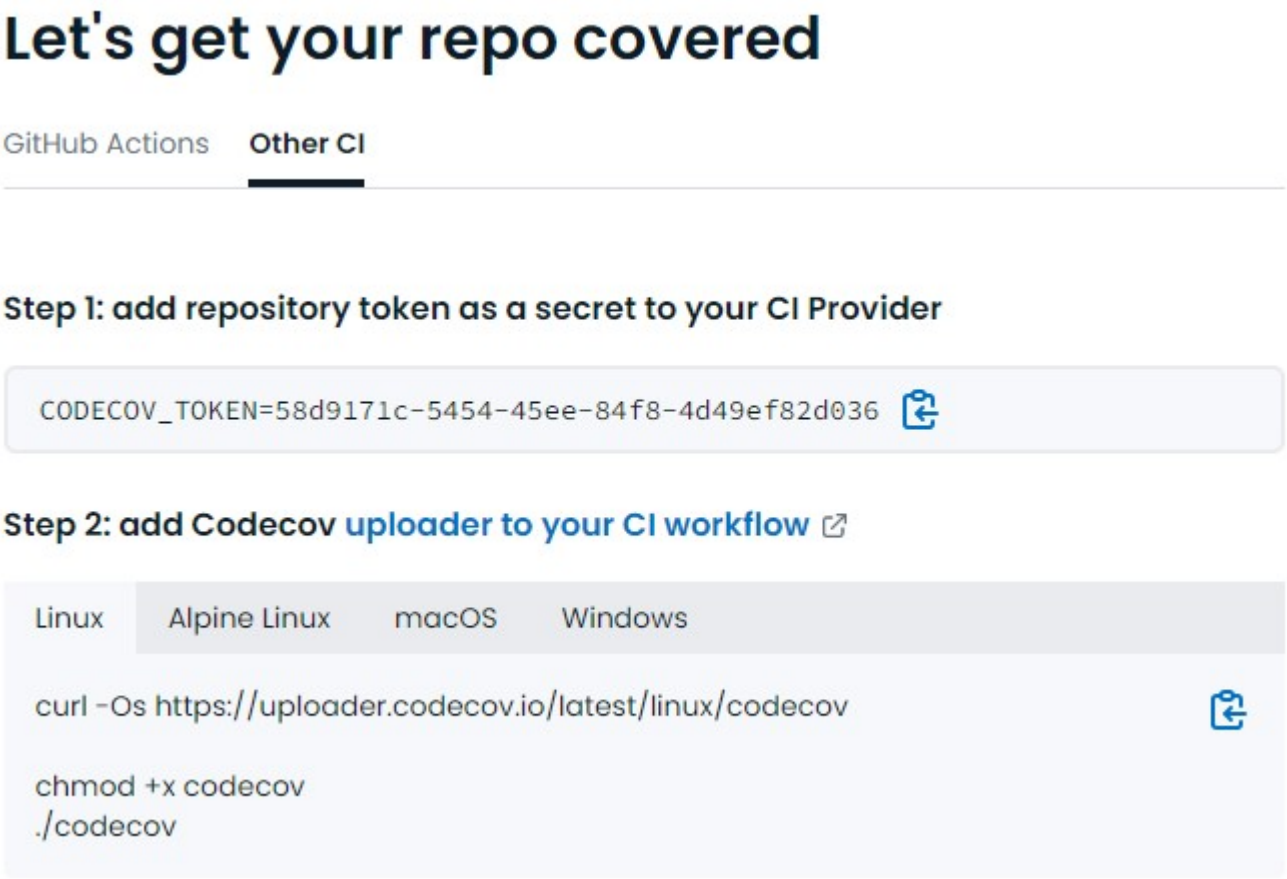


Figure 2. Token du projet

2.3 Création variable d'environnement dans CircleCI

On copie le token générer puis nous nous rendons sur le projet circle ci, pour configurer le projet afin d'ajouter la clé générée.

Pour ajouter ce token dans le projet circle ci, une fois dans projet cliquer sur le bouton **Project Settings**

Figure 3. Bouton Project Settings

Par la suite on clique sur **Environment Variables** puis sur **Add Environment Variable**

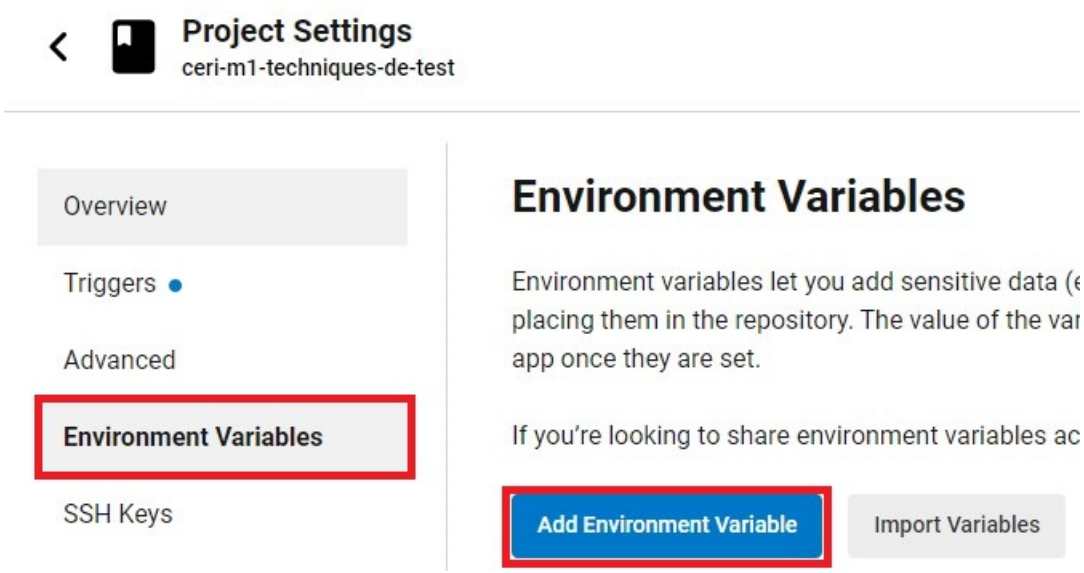


Figure 4. Variable d'environnement

Par la suite on ajoute la variable d'environnement **CODECOV_TOKEN**

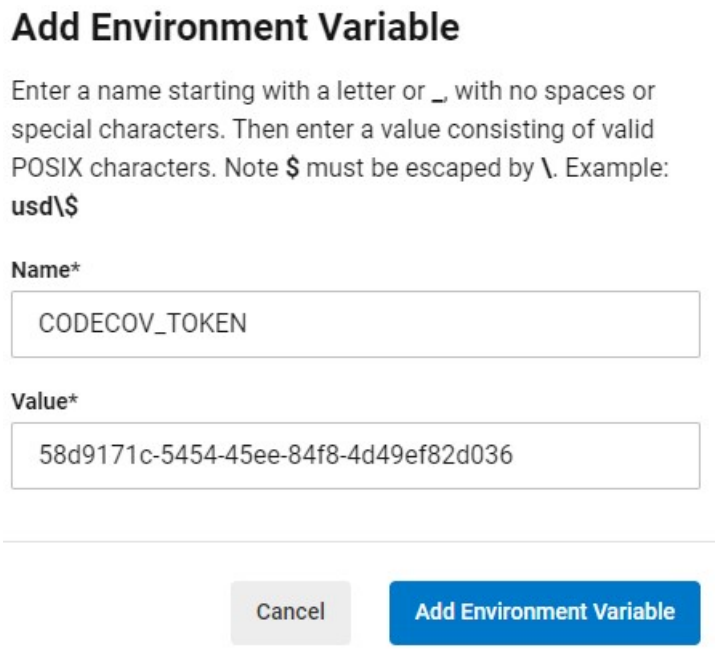
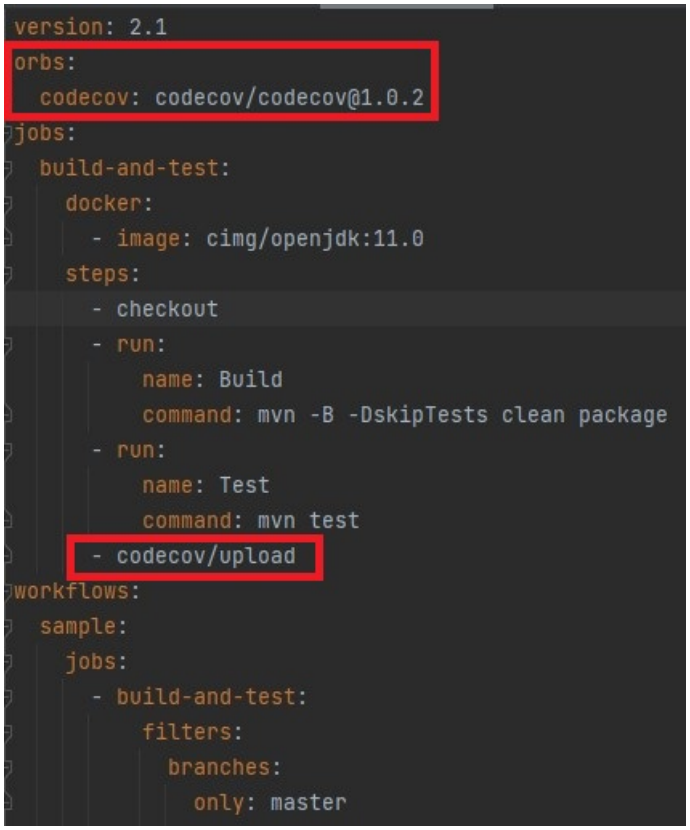


Figure 5. Variable d'environnement CODECOV_TOKEN

2.4 Modification fichier config.yml

Maintenant dans notre projet, il nous faudra modifier le fichier **config.yml**



```yaml
version: 2.1
orbs:
  codecov: codecov/codecov@1.0.2
jobs:
  build-and-test:
    docker:
      - image: cimg/openjdk:11.0
    steps:
      - checkout
      - run:
          name: Build
          command: mvn -B -DskipTests clean package
      - run:
          name: Test
          command: mvn test
      - codecov/upload
workflows:
  sample:
    jobs:
      - build-and-test:
          filters:
            branches:
              only: master
```

Figure 6. Modification config.yml

2.5 Modification fichier pom.xml

Par la suite on ajoute la bibliothèque JaCoCo à notre pom.xml dans la section build.

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

    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-site-plugin</artifactId>
        <version>3.7.1</version>
    </plugin>

    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>3.0.0</version>
    </plugin>

    <plugin>
        <groupId>org.jacoco</groupId>
        <artifactId>jacoco-maven-plugin</artifactId>
        <version>${jacoco.version}</version>
        <executions>
            <execution>
                <id>pre-unit-test</id>
```

```
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>
                </execution>
                <execution>
                    <id>post-unit-test</id>
                    <phase>test</phase>
                    <goals>
                        <goal>report</goal>
                    </goals>

                </execution>
                <execution>
                    <id>pre-integration-test</id>
                    <phase>pre-integration-test</phase>
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>

                </execution>
                <execution>
                    <id>post-integration-test</id>
                    <phase>post-integration-test</phase>
                    <goals>
                        <goal>report</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Listing 1. Bibliothèques pour la couverture et les rapports

Avec cette configuration les rapports seront générer avant et après chaque test et intégration.

Pour générer les rapports il nous suffira de faire par exemple :

```
mvn clean test
```

Console 1. Générer les rapports

Les résultats des rapports sont stocké dans le répertoire `./target/site/jacoco`

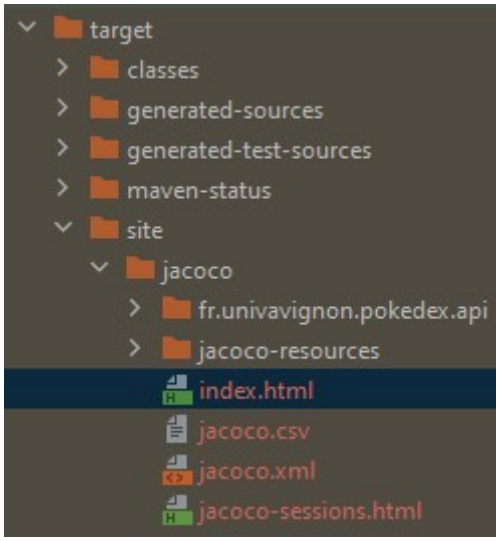


Figure 7. Les rapports

Pour voire les résultats il suffit d'ouvrir le fichier **index.html**

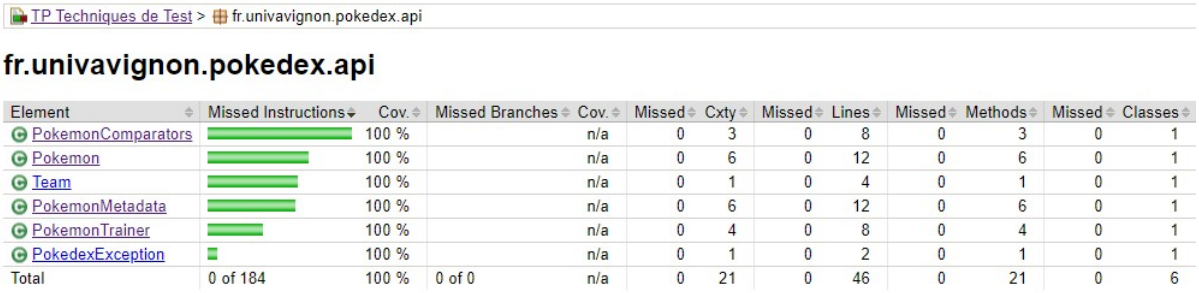


Figure 8. Les rapports dans le navigateur

Comme on peut le voire, j'ai testé toutes les méthodes du projet, raison pour laquelle j'ai pourcentage de **100%**

Sur CircleCI, on peut facilement voire qu'il a générer les rapport et transférer sur CodeCOV.

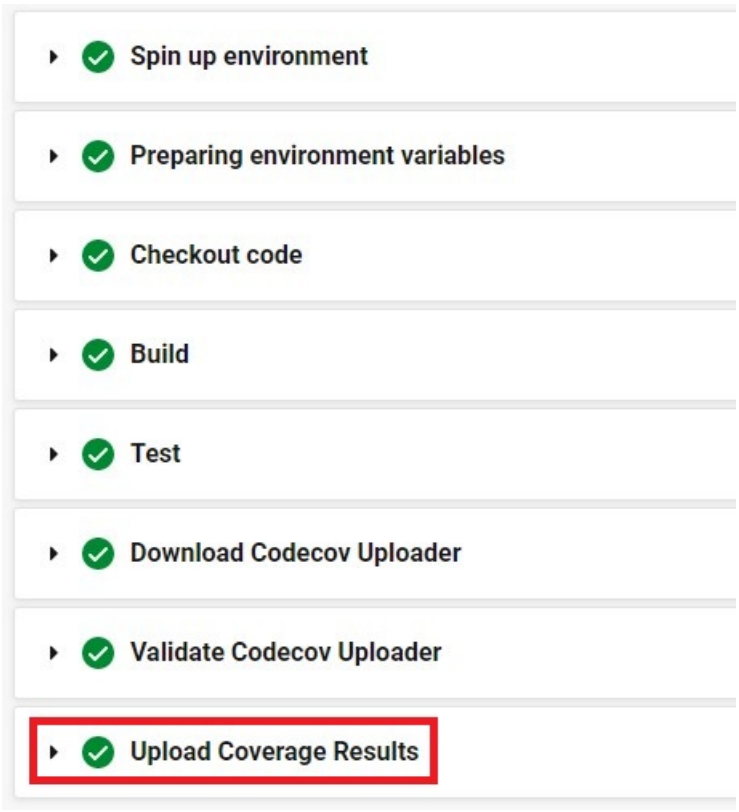


Figure 9. Les résultats sur CircleCI

Sur CodeCOV aussi on peut facilement voire les résultats :

Figure 10. Les résultats sur CodeCOV

3 Les implémentations

Pour les implémentations, il suffisait de créer des classes qui implémenteraient les interfaces que nous avons au préalable.

Ci-dessous la liste des classes céées.

- **PokemonMetadataProvider**
- **Pokedex**
- **PokemonFactory**
- **PokemonTrainerFactory**
- **PokedexFactory**

Après la création de ces classes, notre code coverage diminue drastiquement.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PokemonMetadataProvider		0 %		0 %	9	9	19	19	3	3	1	1
Pokedex		0 %		0 %	12	12	21	21	8	8	1	1
PokemonFactory		0 %		n/a	2	2	12	12	2	2	1	1
PokemonTrainerFactory		0 %		n/a	6	6	12	12	6	6	1	1
PokedexFactory		0 %		n/a	2	2	2	2	2	2	1	1
PokemonComparators		100 %		n/a	0	3	0	8	0	3	0	1
Pokemon		100 %		n/a	0	6	0	12	0	6	0	1
Team		100 %		n/a	0	1	0	4	0	1	0	1
PokemonMetadata		100 %		n/a	0	6	0	12	0	6	0	1
PokemonTrainer		100 %		n/a	0	4	0	8	0	4	0	1
PokedexException		100 %		n/a	0	1	0	2	0	1	0	1
Total	268 of 452	40 %	20 of 20	0 %	31	52	66	112	21	42	5	11

Figure 11. Les rapports dans le navigateur

Pour régler cela, nous allons maintenant remplacer les mocks préalablement créés par nos implémentations.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Pokedex.java		100 %		100 %	0	12	0	21	0	8	0	1
PokemonMetadataProvider.java		100 %		100 %	0	6	0	14	0	2	0	1
PokemonComparators.java		100 %		n/a	0	3	0	8	0	3	0	1
PokemonFactory.java		100 %		100 %	0	3	0	12	0	2	0	1
Pokemon.java		100 %		n/a	0	6	0	12	0	6	0	1
Team.java		100 %		n/a	0	1	0	4	0	1	0	1
PokemonMetadata.java		100 %		n/a	0	6	0	12	0	6	0	1
PokemonTrainerFactory.java		100 %		n/a	0	2	0	6	0	2	0	1
PokemonTrainer.java		100 %		n/a	0	4	0	8	0	4	0	1
PokedexFactory.java		100 %		n/a	0	2	0	2	0	2	0	1
PokedexException.java		100 %		n/a	0	1	0	2	0	1	0	1
Total	0 of 417	100 %	0 of 18	100 %	0	46	0	101	0	37	0	11

Figure 12. Les rapports dans le navigateur

4 Conclusion

Dans cet tp, il était question de calculer la couverture de test de notre projet mais aussi d'implémenter les différentes méthodes de nos interfaces.

Ce travail a été réalisé en utilisant la plateforme **Codecov** mais aussi la bibliothèque **Jacoco**.

Références

[1] Daniel SALAS. *TP4 : Code coverage*. Université Avignon. 2023. url : https://github.com/Youkoulanda/ceri-m1-techniques-de-test/blob/master/TPs/TP4.md.

[2] LAMAH Michel Marie. *TP4 : Code coverage [master]*. Université Avignon. 2023. url : https://github.com/MichelMarieLamah13/ceri-m1-techniques-de-test/tree/master.

[3] LAMAH Michel Marie. *TP4 : Code coverage [TP04]*. Université Avignon. 2023. url : https://github.com/MichelMarieLamah13/ceri-m1-techniques-de-test/tree/TP04.