LOG3430 - Méthodes de test et de validation du logiciel

Laboratoire 1

Couverture de test

Hisham Boulifa - 2085232

Pungtzé-Sy D. Kamdem - 2139333

Gr. : 02

11 février 2024

**Question 1**

Après avoir fait la commande *’pytest in tests’ with Coverage,* on peut apercevoir que la couverture totale du code pour *Requests* est de 87%.

Une image contenant texte, capture d’écran, document, nombre

Description générée automatiquement

Figure 1. Couverture de code de la question 1

**Question 2**

En prenant en compte les branches, la couverture de code diminue de 2% pour tomber à 85%. Ceci peut s’expliquer par le fait qu’il manque des scénarios possibles à tester. Par exemple, il se peut qu’une condition *if* soit exécutée par les tests sans qu’on puisse avoir un aperçu de toutes les issues de la condition.

Une image contenant texte, document, menu, capture d’écran

Description générée automatiquement

Figure 2. Couverture de code de la question 2

**Question 3**

Nous avons d’abord modifié *.coveragerc* afin d’omettre le fichier *compat.py* des tests.

**A close up of a text

Description automatically generated**

Figure 3. Contenu du fichier *.coveragerc* une fois modifié

Ensuite, nous avons écrit ces trois lignes de commandes :

* *coverage erase*
* *coverage run -m pytest*
* *coverage html*

ce qui a généré le fichier de couverture html suivant : (page suivante)

Une image contenant texte, menu, document, nombre

Description générée automatiquement

Figure 4. Couverture de code de la question 3

Nous pouvons apercevoir que la couverture de code est passé à 93% après avoir omis *compat.py*.

**Question 4**

Tout d’abord, si ce n’est pas déjà fait, il faut installer le module *coverage* en faisant *pip install coverage*.

Ensuite, il suffit d’exécuter la commande *coverage run -m --branch -m pytest.*

Enfin, pour visualiser le tableau de la couverture de code, nous avons exécuté la commande *coverage html.*

* La commande *coverage run* dit à coverage.py de surveiller et de mesurer la couverture du code Python lors de son exécution, ce qui inclut le suivi des appels de fonction, des lignes de code, des branches, et d'autres éléments du programme.
* L'option *-m* informe Python d'exécuter un module particulier comme s'il s'agissait d'un script autonome. Dans ce contexte, *-m* est employé pour exécuter le module pytest comme un script indépendant.
* *--branch* spécifie que la couverture de branche doit être calculée. Cela signifie que la couverture sera mesurée non seulement pour les lignes de code exécutées, mais aussi pour les branches de code, telles que les instructions conditionnelles, les boucles, etc.
* *pytest* est le module de test Python que nous utilisons pour exécuter les tests. Il recherche automatiquement les fichiers de test dans le répertoire courant et les sous-répertoires et exécute les tests qu'il trouve.

**A screenshot of a computer

Description automatically generated**

Figure 5. Couverture de code de la question 4

**Question 5**

Comme pour la question précédente, il faut installer le module *pytest-cov* en faisant *pip install pytest-cov*. Ensuite, il faut exécuter la commande suivante :

*pytest --cov=requests --cov-branch --cov-report term --cov-report html*

* *pytest* est la commande pour exécuter des tests en utilisant *pytest*.
* *--cov=requests* indique à *pytest-cov* de mesurer la couverture de code pour le paquet *requests*.
* *--cov-branch* active la mesure de la couverture de branches.
* *--cov-report term* indique à *pytest-cov* de montrer le rapport de couverture dans le terminal après l'exécution des tests.
* *--cov-report html* crée un rapport de couverture détaillé en HTML.

**A screenshot of a computer screen

Description automatically generated**

Figure 6. Couverture de code de la question 5

**Question 6**

Voici un exemple de couverture avec la méthode *coverage.py*:

**A screenshot of a computer

Description automatically generated**

Figure 7. Couverture de code de la question 6 avec la méthode *coverage.py*

Pour le même fichier, voici la couverture avec *pytest-cov*:

A screenshot of a computer

Description automatically generated

Figure 8. Couverture de code de la question 6 avec la méthode *pytest-cov*

À travers l’analyse de la couverture de ce fichier avec les deux méthodes différentes, nous pouvons remarquer que *pytest-cov* ne couvre pas les lignes d’importation de module (ex : import calendar), les lignes de déclaration de classe (ex : *class MockRequest*) ainsi que les lignes de déclaration de méthode (ex *: def \_\_init\_\_(self, request)*).

Ces résultats sont sensés, car *pytest-cov* couvre uniquement le code qu’on teste activement. Les lignes d’importation et de déclaration ne font pas partie de ces lignes désirées. En voici l’explication; les lignes d'importation servent plutôt à importer d'autres modules ou ressources nécessaires à l'exécution de votre code. Ainsi, inclure ces lignes dans les mesures de la couverture de code peut fausser les résultats, car elles ne représentent pas le fonctionnement réel de notre code. Cela s’applique également pour les lignes de déclaration.

**Question 7**

Voici les tests ajoutés pour augmenter la couverture :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 9. Tests ajoutés



Figure 10. Couverture de code de la question 7

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 11. Lignes couvertes grâce aux nouveaux tests