Rapport de Test

Membres de l'équipe:

Ousmane Mbathie
Omar Boune Khatabe Thiam
Mouhamed Koné

Introduction

Ce document présente le rapport de test pour le module de gestion de projet implémenté dans les fichiers `gestion_projet.py` et `test_gestion_projet.py`. L'objectif de ce rapport est de fournir une vue d'ensemble des tests effectués pour valider les fonctionnalités du module, de décrire les résultats obtenus et d'identifier les éventuelles anomalies détectées au cours du processus de test.

Contexte

Le module de gestion de projet est conçu pour faciliter la gestion des projets en offrant des fonctionnalités telles que l'ajout de membres à l'équipe, la gestion des tâches, la définition du budget, la gestion des risques, et la génération de rapports de performance. Le module utilise différentes stratégies de notification (email, SMS, push notification) pour tenir les membres de l'équipe informés des mises à jour importantes.

Objectifs des Tests

Les tests ont été effectués dans le but de :

1. Vérifier la correcte implémentation des fonctionnalités du module de gestion de projet.
2. Assurer que les différentes stratégies de notification fonctionnent comme prévu.
3. Valider que les calculs tels que le chemin critique sont exacts et conformes aux attentes.
4. Garantir que les rapports de performance générés reflètent correctement les activités du projet.

Environnement de Test

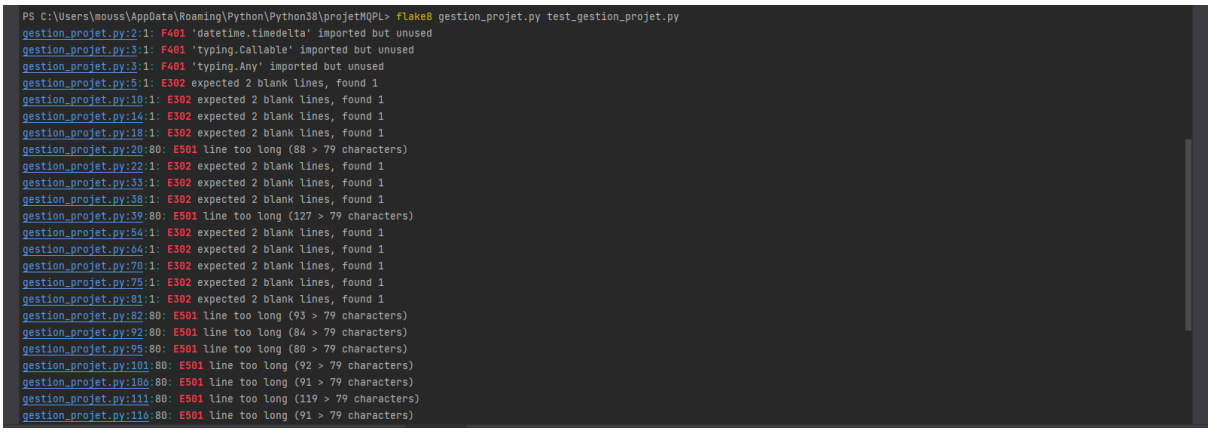
Les tests ont été réalisés dans l'environnement suivant :

- Système d'exploitation : Windows 10
- Version de Python : 3.7
- Bibliothèques et outils utilisés : unittest, flake , pylint , mypy , coverage , vulture , black , radon , pyflakes .

Méthodologie

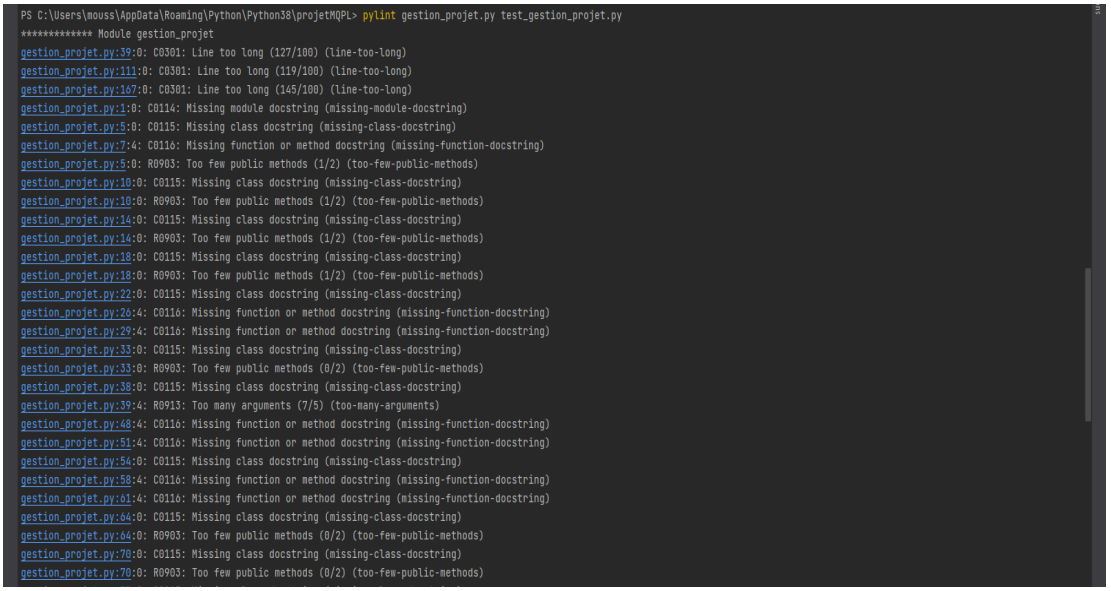
Les tests unitaires ont été écrits et exécutés à l'aide du module `unittest` de Python. Les scripts de test sont situés dans le fichier `test_gestion_projet.py`. Les tests couvrent les principaux composants du module de gestion de projet, en vérifiant à la fois les cas positifs (fonctionnement attendu) et les cas négatifs (gestion des erreurs).

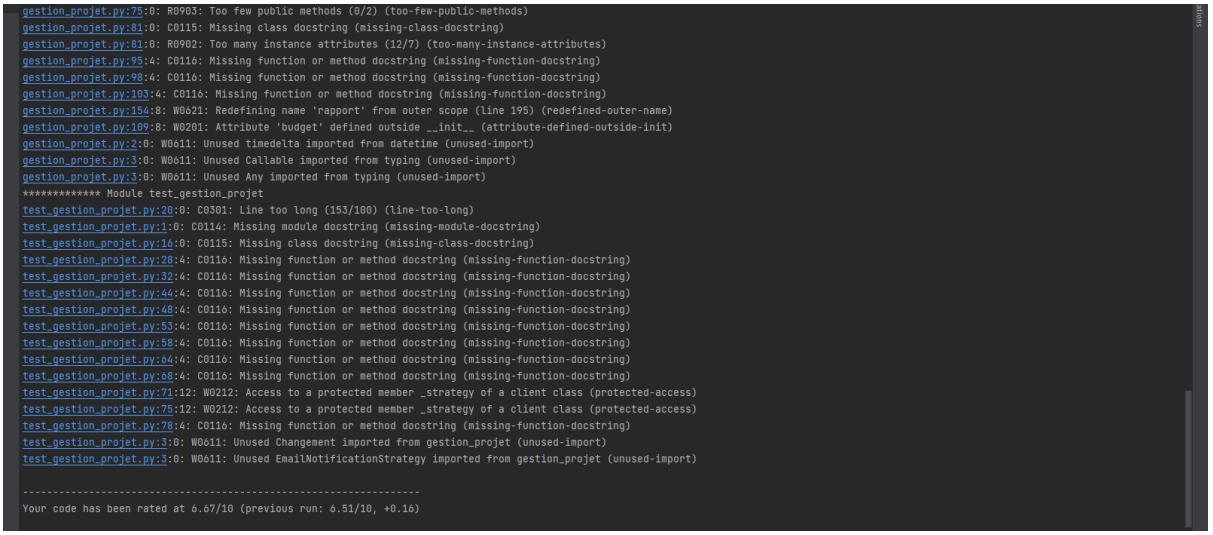
Résultats :



Flake8

- **Commande :** `flake8 gestion_projet.py test_gestion_projet.py`
- **Amélioration :**
    - **Détection des Erreurs de syntaxe :** Flake8 détecte les erreurs de syntaxe dans le code, comme les parenthèses manquantes ou les erreurs d'indentation.
    - **Conformité PEP8 :** Il vérifie la conformité du code aux conventions de style PEP8, assurant une uniformité et une lisibilité accrue.
    - **Identification des Problèmes de Complexité :** Flake8 détecte les segments de code trop complexes, suggérant des simplifications possibles.





Commande : `pylint gestion_projet.py test_gestion_projet.py`
Amélioration :

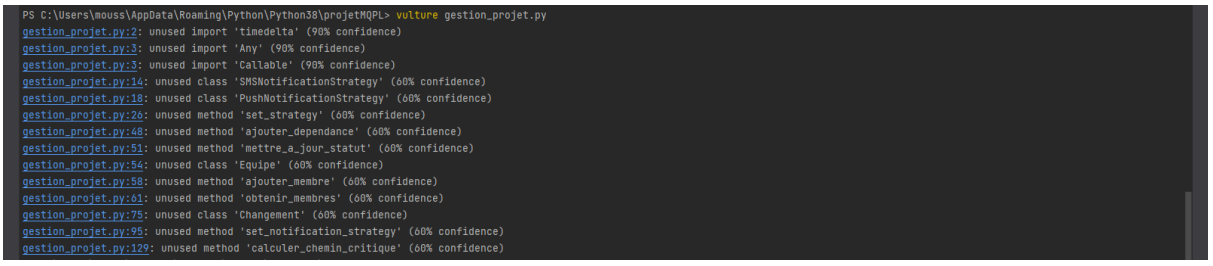
- **Analyse de la Qualité du Code :** Pylint fournit un score de qualité pour le code et des recommandations pour l'améliorer.
- **Détection des Erreurs :** Il détecte des erreurs de logique, comme l'utilisation de variables non définies.
- **Suggestions de Refactorisation :** Pylint suggère des améliorations pour rendre le code plus propre et plus maintenable.

Mypy

- **Commande :** `mypy gestion_projet.py test_gestion_projet.py`
- **Amélioration :**
    - **Vérification des Types :** Mypy vérifie la conformité du code avec les annotations de type, aidant à prévenir les erreurs de type à l'exécution.
    - **Détection des Types Manquants :** Il signale les variables et fonctions sans annotations de type, encourageant une documentation explicite des types.

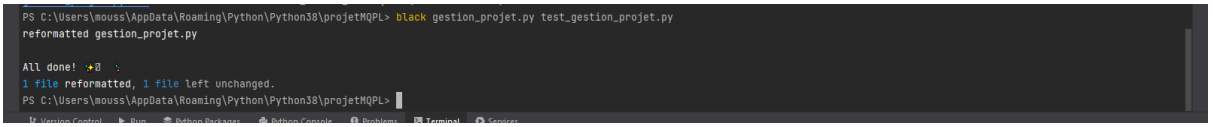
Commande : `coverage run -m unittest discover` suivi de `coverage report`
Amélioration :

- **Mesure de la Couverture de Test :** Coverage montre quelles parties du code sont couvertes par les tests, identifiant les segments non testés.
- **Amélioration de la Couverture :** Il aide à augmenter la couverture de test en écrivant des tests supplémentaires pour les parties du code non couvertes.



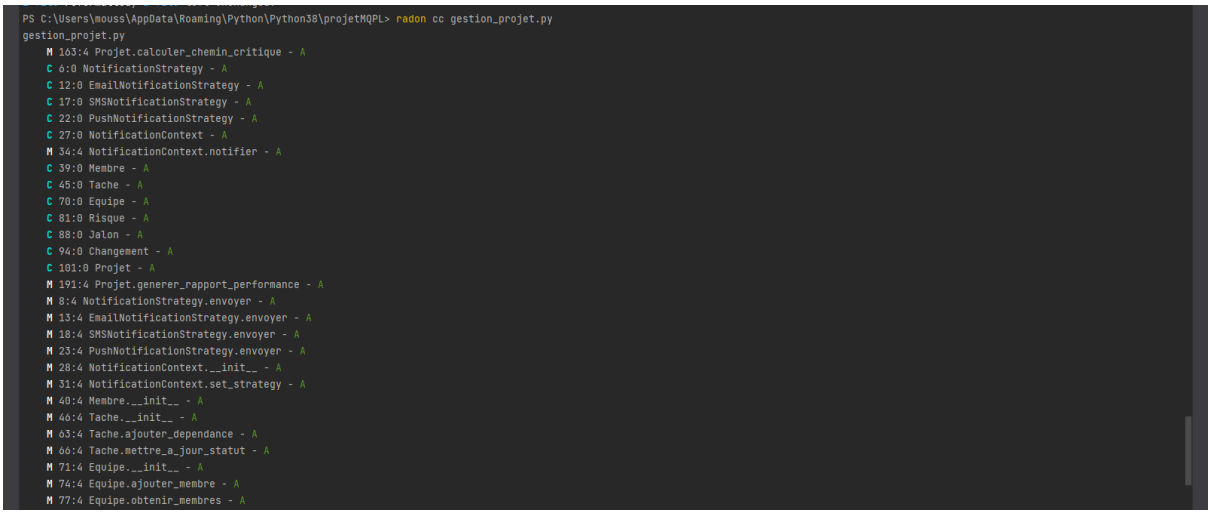
Commande : `vulture gestion_projet.py`
Amélioration :

- **Détection de Code Mort :** Vulture identifie les parties du code qui ne sont jamais exécutées (code mort), suggérant leur suppression pour réduire le volume du code et améliorer la lisibilité.



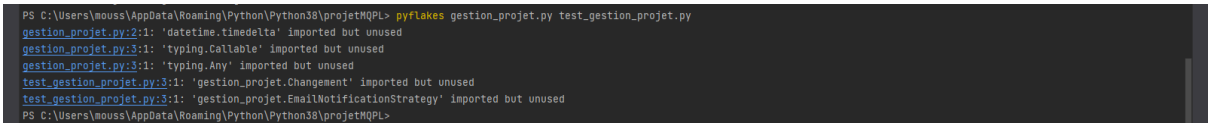
Black

- **Commande :** `black gestion_projet.py test_gestion_projet.py`
- **Amélioration :**
  - **Formatage automatique :** Black formate automatiquement le code pour qu'il soit conforme aux conventions PEP8, améliorant la cohérence et la lisibilité.
  - **Uniformité du Code :** Il assure un style de code uniforme dans tout le projet, facilitant la collaboration entre développeurs.



Radon

- **Commande :** `radon cc gestion_projet.py` et `radon mi gestion_projet.py`
- **Amélioration :**
  - **Analyse de la Complexité Cyclomatique :** Radon calcule la complexité cyclomatique des fonctions, aidant à identifier celles qui sont trop complexes et nécessitent une simplification.
  - **Mesure de la Maintenabilité :** Il évalue la maintenabilité du code, offrant un indice de maintenabilité et suggérant des améliorations pour le rendre plus facile à maintenir.



Pyflakes

- **Commande :** `pyflakes gestion_projet.py test_gestion_projet.py`
- **Amélioration :**
    - **Détection des erreurs et Avertissements :** Pyflakes détecte les erreurs syntaxiques, les imports inutilisés et les variables inutilisées.
    - **Nettoyage du Code :** Il aide à maintenir un code propre en éliminant les éléments superflus qui n'affectent pas directement le style mais peuvent causer des erreurs.

Conclusion:

L'intégration régulière de ces outils dans le cycle de développement permet d'identifier rapidement les problèmes potentiels et d'assurer que le code reste propre, maintenable et conforme aux bonnes pratiques de programmation.