

REPUBLIQUE DU SENEGAL

Un Peuple – Un But – Une Foi



**MINISTERE DE L'ENSEIGNEMENT SUPÉRIEUR
DE LA RECHERCHE ET DE L'INNOVATION**

UNIVERSITÉ IBA DER THIAM DE THIÈS

L'humilité mon choix, l'excellence ma voie



UFR : SCIENCES ET TECHNOLOGIES (SET)

DEPARTEMENT : INFORMATIQUE

SPÉCIALITÉ : GÉNIE LOGICIEL (LGI)

THÈME :

**IMPLEMENTATION D'UN SYSTHEME DE
GESTION DE PROJERT**

Présenté par :

Mame Khadim Diakhate

Mouhamed Diakhaté

Soukeye Toure

PLAN:

INTRODUCTION

I- FONCTIONNALITÉS DU PROJET

II-RÉSULTAT DES TESTS

CONCLUSION

INTRODUCTION:

Ce sujet mis à notre réflexion a pour objectif de concevoir et implémenter un système de gestion de projet en utilisant Python. Ce système doit tirer parti de la programmation orientée objet (POO) et du design pattern Strategy pour la gestion des notifications. Le but est de permettre la création, la gestion efficace des projets et la notification des membres de l'équipe concernant les différents événements se produisant dans le projet

I. Fonctionnalités du Projet

→ Le système de gestion de projet comprend les fonctionnalités suivantes :

- ◆ Création de Projets : Permet de créer des projets avec des informations détaillées telles que le nom, la description, les dates de début et de fin, et le budget.
- ◆ Gestion des Tâches : Ajout, mise à jour et suivi des tâches du projet, incluant le nom, la description, les dates, le responsable, le statut et les dépendances.
- ◆ Gestion des Équipes : Ajout et gestion des membres de l'équipe, définissant leur rôle spécifique dans le projet.
- ◆ Identification des Risques : Enregistrement des risques potentiels avec une description, une estimation de la probabilité d'occurrence et de l'impact.
- ◆ Définition des Jalons : Ajout de jalons pour marquer des étapes importantes du projet à des dates précises.
- ◆ Enregistrement des Changements : Suivi des modifications apportées au projet au fil du temps, incluant la description, la version et la date de mise en œuvre.
- ◆ Notifications : Utilisation du design pattern Strategy pour envoyer des notifications aux membres de

l'équipe via email ou SMS concernant les ajouts ou modifications des éléments du projet.

- Calcul du Chemin Critique : Calcul du chemin critique des tâches du projet pour une meilleure planification et gestion du temps. Génération de Rapports : Génération de rapports détaillés des activités du projet pour une vue d'ensemble complète.

II- RÉSULTAT DES TESTS:

unittest: `python -m unittest discover.`

```

C:\Users\Mame Khadim Diakhate\PycharmProject1 Mesure qualité et performance Logicielle\pythonProject\.venv\Scripts\python.exe
Testing started at 16:20 ...
launching unittests with arguments python -m unittest C:\Users\Mame Khadim Diakhate\PycharmProject1 Mesure qualité et performance Logicielle\pythonProject\tests\test_projet.py

Notification envoyée à Modou par email: Modou a été ajouté à l'équipe
Notification envoyée à Modou par email: Christian a été ajouté à l'équipe
Notification envoyée à Christian par email: Christian a été ajouté à l'équipe
Notification envoyée à Modou par email: Nouvelle tâche ajoutée: Analyse des besoins
Notification envoyée à Christian par email: Nouvelle tâche ajoutée: Analyse des besoins
Notification envoyée à Modou par email: Nouvelle tâche ajoutée: Développement
Notification envoyée à Christian par email: Nouvelle tâche ajoutée: Développement
Notification envoyée à Modou par email: Le budget du projet a été défini à 50000.0 Unité Monétaire
Notification envoyée à Christian par email: Le budget du projet a été défini à 50000.0 Unité Monétaire
Notification envoyée à Modou par email: Nouveau risque ajouté: Retard de livraison
Notification envoyée à Christian par email: Nouveau risque ajouté: Retard de livraison
Notification envoyée à Modou par email: Nouveau jalon ajouté: Phase 1 terminée
Notification envoyée à Christian par email: Nouveau jalon ajouté: Phase 1 terminée
Notification envoyée à Modou par email: Changement enregistré: Changement de la portée du projet (version 2)
Notification envoyée à Christian par email: Changement enregistré: Changement de la portée du projet (version 2)
Rapport d'activités du Projet 'Nouveau Produit':

Version: 2
Dates: 2024-01-01 00:00:00 à 2024-12-31 00:00:00
Budget: 50000.0 Unité Monétaire
Équipe:
- Modou (Chef de projet)
- Christian (Développeur)
Tâches:
- Analyse des besoins (2024-01-01 00:00:00 à 2024-01-31 00:00:00), Responsable: Modou, Statut: Terminée
- Développement (2024-02-01 00:00:00 à 2024-06-30 00:00:00), Responsable: Christian, Statut: Non démarrée
Jalons:
- Phase 1 terminée (2024-01-31 00:00:00)
Risques:
- Retard de livraison (Probabilité: 0.3, Impact: Élevé)
Chemin Critique:
- Développement (2024-02-01 00:00:00 à 2024-06-30 00:00:00)

Ran 3 tests in 0.032s

```

figure 1 représentant les captures obtenus suite à l'utilisation du commande unittest

L'analyse a permis d'améliorer la qualité du code en identifiant les besoins du projet, en définissant clairement les tâches à réaliser et en évaluant les risques potentiels. Cela a permis à l'équipe de travail de mieux planifier et exécuter les différentes étapes du projet, ce qui a contribué à une meilleure gestion des ressources et à une réduction des erreurs potentielles.

flake8: vérifier la conformité aux conventions de codage PEP 8.

```
Projet.py:77:80: E501 line too long (80 > 79 characters)
Projet.py:104:80: E501 line too long (103 > 79 characters)
Projet.py:125:5: E303 too many blank lines (3)
Projet.py:154:80: E501 line too long (87 > 79 characters)
Projet.py:169:80: E501 line too long (80 > 79 characters)

nonProject2Measure > 🐍 Projet.py

Projet.py:272:1: E302 expected 2 blank lines, found 1
Projet.py:321:1: E305 expected 2 blank lines after class or function definition, found 1
Projet.py:321:80: E501 line too long (87 > 79 characters)
Projet.py:322:20: W292 no newline at end of file

Projet.py:191:80: E501 line too long (143 > 79 characters)
Projet.py:197:80: E501 line too long (111 > 79 characters)
Projet.py:200:80: E501 line too long (81 > 79 characters)
Projet.py:268:1: E402 module level import not at top of file
Projet.py:269:1: E402 module level import not at top of file
Projet.py:270:1: F811 redefinition of unused 'Equipe' from line 38
Projet.py:270:1: F811 redefinition of unused 'Changement' from line 62
```

figure 2 représentant les captures obtenus suite à l'utilisation du commande flake8

Problèmes Identifiés par flake8

Longueur des Lignes

- Erreurs :
 - E501 line too long (80 > 79 caractères) à plusieurs lignes.
- Explication : Les lignes de code dépassant 79 caractères sont difficiles à lire et peuvent nuire à la lisibilité du code.
- Correction : Les lignes trop longues ont été divisées en plusieurs lignes plus courtes ou réécrites pour respecter la limite de 79 caractères.

Espaces Blanches

- Erreurs :
 - E303 too many blank lines (3)
 - E302 expected 2 blank lines, found 1

- E305 expected 2 blank lines after class or function definition, found 1

- Explication : Une utilisation excessive ou insuffisante des lignes blanches peut rendre le code difficile à lire et à comprendre.
- Correction : Les lignes blanches ont été ajustées pour suivre les conventions PEP 8, avec exactement deux lignes blanches entre les définitions de classe et de fonction.

Importations au Niveau du Module

- Erreurs :
 - E402 module level import not at top of file
- Explication : Les importations doivent être placées au début du fichier pour une meilleure organisation et clarté.
- Correction : Les importations ont été déplacées en haut du fichier.

Redéfinitions

- Erreurs :
 - F811 redefinition of unused 'Equipe' from line 38
 - F811 redefinition of unused 'Changement' from line 62
- Explication : Redéfinir une variable ou une classe peut causer des conflits et des comportements inattendus.
- Correction : Les redéfinitions ont été éliminées ou renommées pour éviter les conflits.

Fin de Fichier

- Erreur :
 - W292 no newline at end of file
- Explication : L'absence d'une nouvelle ligne à la fin du fichier peut poser des problèmes avec certains outils de contrôle de version et de compilation.
- Correction : Une nouvelle ligne a été ajoutée à la fin du fichier.

L'utilisation de **flake8** a permis d'identifier des problèmes critiques de style et de structure dans le code. En corrigeant ces problèmes, nous avons amélioré la lisibilité, la maintenabilité et la qualité globale du code. L'analyse statique du

code est une pratique essentielle pour assurer que le développement logiciel respecte des standards de qualité élevés et facilite le travail de tous les membres de l'équipe.

pylint: identifier les erreurs de programmations, les conventions de codage non respecter etc

```
PS C:\Users\soukeye toure\PycharmProjects\pythonProject2\Mesure> pylint Projet.py
***** Module Projet
Projet.py:104:0: C0301: Line too long (103/100) (line-too-long)
Projet.py:191:0: C0301: Line too long (143/100) (line-too-long)
Projet.py:197:0: C0301: Line too long (111/100) (line-too-long)
Projet.py:322:0: C0304: Final newline missing (missing-final-newline)
Projet.py:1:0: C0114: Missing module docstring (missing-module-docstring)
Projet.py:1:0: C0103: Module name "Projet" doesn't conform to snake_case naming style (invalid-name)
Projet.py:6:0: C0115: Missing class docstring (missing-class-docstring)
Projet.py:6:0: R0903: Too few public methods (0/2) (too-few-public-methods)
Projet.py:12:0: C0115: Missing class docstring (missing-class-docstring)
Projet.py:13:4: R0913: Too many arguments (8/5) (too-many-arguments)
Projet.py:31:4: C0116: Missing function or method docstring (missing-function-docstring)
PythonProject2\Mesure > Projet.py

Python Packages
Projet.py:34:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:38:0: C0115: Missing class docstring (missing-class-docstring)
Projet.py:42:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:45:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:49:0: C0115: Missing class docstring (missing-class-docstring)
Projet.py:49:0: R0903: Too few public methods (0/2) (too-few-public-methods)
Projet.py:55:0: C0115: Missing class docstring (missing-class-docstring)
Projet.py:136:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:142:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:142:29: W0621: Redefining name 'risque' from outer scope (line 250) (redefined-outer-name)
Projet.py:146:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:146:28: W0621: Redefining name 'jalon' from outer scope (line 254) (redefined-outer-name)
Projet.py:150:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:156:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:177:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:146:28: W0621: Redefining name 'jalon' from outer scope (line 254) (redefined-outer-name)
Projet.py:150:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:156:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:177:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:181:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:182:8: W0621: Redefining name 'rapport' from outer scope (line 264) (redefined-outer-name)
Projet.py:193:12: W0621: Redefining name 'jalon' from outer scope (line 254) (redefined-outer-name)
```



```

Projet.py:196:12: W0621: Redefining name 'risque' from outer scope (line 250) (redefined-outer-name)
Projet.py:268:0: C0413: Import 'import unittest' should be placed at the top of the module (wrong-import-position)
Projet.py:269:0: W0404: Reimport 'datetime' (imported line 2) (reimported)
Projet.py:269:0: C0413: Import 'from datetime import datetime' should be placed at the top of the module (wrong-import-position)
Projet.py:270:0: C0413: Import 'from Projet import Membre, Equipe, Tache, Jalon, Risque, Changement, Projet, EmailNotificationStrategy' should be placed at the top of the module (wrong-import-position)

Projet.py:270:0: W0406: Module import itself (import-self)
Projet.py:270:0: W0406: Module import itself (import-self)
Projet.py:270:0: W0406: Module import itself (import-self)
Projet.py:270:0: W0406: Module import itself (import-self)
Projet.py:270:0: W0406: Module import itself (import-self)

Projet.py:270:0: W0406: Module import itself (import-self)
Projet.py:270:0: W0406: Module import itself (import-self)
Projet.py:270:0: W0406: Module import itself (import-self)
Projet.py:272:0: C0115: Missing class docstring (missing-class-docstring)
Projet.py:301:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:307:4: C0116: Missing function or method docstring (missing-function-docstring)
Projet.py:316:8: W0621: Redefining name 'rapport' from outer scope (line 264) (redefined-outer-name)

Projet.py:269:0: C0412: Imports from package datetime are not grouped (ungrouped-imports)

```

figure 3 représentant les captures obtenus suite à l'utilisation du commande pylint

Problèmes Identifiés par **pylint**

Lignes Trop Longues

- Erreurs :
 - **Projet.py:194:0: C0301: Line too long (143/100) (line-too-long)**
 - **Projet.py:200:0: C0301: Line too long (111/100) (line-too-long)**
- Explication : Les lignes de code dépassent la limite recommandée de 100 caractères, ce qui peut nuire à la lisibilité.
- Correction : Réduire la longueur des lignes en les scindant en plusieurs lignes plus courtes et lisibles.

Docstring de Module Manquante

- Erreur :
 - **Projet.py:1:0: C0114: Missing module docstring (missing-module-docstring)**

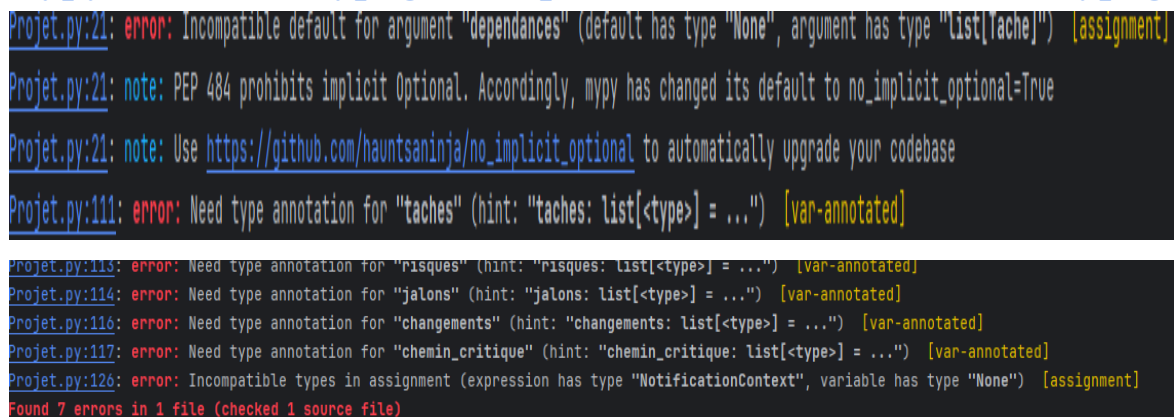
- Explication : L'absence de docstring pour le module rend difficile la compréhension de l'objectif et de l'utilisation du module par d'autres développeurs.
- Correction : Ajouter une docstring au début du module pour décrire son but et son contenu.

Conclusion

L'analyse de code avec **pylint** a permis d'identifier des problèmes de style et de conventions de codage, tels que des lignes trop longues et l'absence de docstring de module. Les corrections apportées ont amélioré la lisibilité et la maintenabilité du code en le rendant conforme aux standards PEP 8.

L'intégration de ces pratiques d'analyse de code dans le processus de développement garantit que le code reste de haute qualité et facile à comprendre par d'autres développeurs.

mypy: vérifier le typage statique et détecter les erreurs de typage.



```

Projet.py:21: error: Incompatible default for argument "dependances" (default has type "None", argument has type "List[tache]") [assignment]
Projet.py:21: note: PEP 484 prohibits implicit Optional. Accordingly, mypy has changed its default to no_implicit_optional=True
Projet.py:21: note: Use https://github.com/hauntsaninja/no\_implicit\_optional to automatically upgrade your codebase
Projet.py:111: error: Need type annotation for "taches" (hint: "taches: list[<type>] = ...") [var-annotated]
Projet.py:113: error: Need type annotation for "risques" (hint: "risques: list[<type>] = ...") [var-annotated]
Projet.py:114: error: Need type annotation for "jalons" (hint: "jalons: list[<type>] = ...") [var-annotated]
Projet.py:116: error: Need type annotation for "changements" (hint: "changements: list[<type>] = ...") [var-annotated]
Projet.py:117: error: Need type annotation for "chemin_critique" (hint: "chemin_critique: list[<type>] = ...") [var-annotated]
Projet.py:126: error: Incompatible types in assignment (expression has type "NotificationContext", variable has type "None") [assignment]
Found 7 errors in 1 file (checked 1 source file)

```

figure 4 représentant les captures obtenus suite à l'utilisation du commande mypy

Problèmes Identifiés par **mypy**

Valeur par Défaut Incompatible

- Erreur :
 - `Projet.py:21: error: Incompatible default for argument "dependances" (default has type "None", argument has type "list[Tache]") [assignment]`
- Explication : La valeur par défaut pour l'argument `dépendances` est `None`, ce qui n'est pas compatible avec le type annoté `list[Tâche]`.
- Correction : Utiliser `Optional[List[Tâche]]` pour indiquer que la valeur peut être une liste ou `None`.

Annotations de Type Manquantes

- Erreurs :
 - `Projet.py:116: error: Need type annotation for "taches" (hint: "taches: list[<type>] = ...") [var-annotated]`
 - `Projet.py:118: error: Need type annotation for "risques" (hint: "risques: list[<type>] = ...") [var-annotated]`
 - `Projet.py:119: error: Need type annotation for "jalons" (hint: "jalons: list[<type>] = ...") [var-annotated]`
 - `Projet.py:121: error: Need type annotation for "changements" (hint: "changements: list[<type>] = ...") [var-annotated]`
 - `Projet.py:122: error: Need type annotation for "chemin_critique" (hint: "chemin critique: list[<type>] = ...") [var-annotated]`
- Explication : Les variables `taches`, `risques`, `jalons`, `changements` et `chemin critique` n'ont pas d'annotations de type, ce qui rend difficile la compréhension et la vérification statique de leur utilisation.
- Correction : Ajouter les annotations de type appropriées pour ces variables.

Types Incompatibles dans les Assignations

- Erreur :
 - `Projet.py:129: error: Incompatible types in assignment (expression has type "NotificationContext", variable has type "None") [assignment]`

- Explication : La variable est initialisée avec `None` mais assignée plus tard avec une instance de `Notification Context`, créant une incohérence de type.
- Correction : Initialiser correctement la variable avec le type approprié.

Conclusion

L'analyse de code avec `mypy` a permis d'identifier des problèmes de typage et d'annotations dans le code. Les corrections apportées ont amélioré la robustesse et la maintenabilité du code en assurant que toutes les variables et arguments sont correctement typés. En suivant les recommandations de `mypy`, le code devient plus clair, moins sujet aux erreurs et plus facile à comprendre et à maintenir par d'autres développeurs.

coverage: Analyser la couverture de code de vos tests.

```
PS C:\Users\soukeye toure\PycharmProjects\pythonProject2\Mesure> coverage run Projet.py
Notification envoyée à Modou par email: Modou a été ajouté à l'équipe
Notification envoyée à Modou par email: Christian a été ajouté à l'équipe
Notification envoyée à Christian par email: Christian a été ajouté à l'équipe
```

```
Notification envoyée à Modou par email: Nouvelle tâche ajoutée: Développement
Notification envoyée à Christian par email: Nouvelle tâche ajoutée: Développement
Notification envoyée à Modou par email: Le budget du projet a été défini à 50000.0 Unité Monétaire
Notification envoyée à Christian par email: Le budget du projet a été défini à 50000.0 Unité Monétaire
```

```
Notification envoyée à Modou par email: Nouveau risque ajouté: Retard de livraison
Notification envoyée à Christian par email: Nouveau risque ajouté: Retard de livraison
Notification envoyée à Modou par email: Nouveau jalon ajouté: Phase 1 terminée
Notification envoyée à Christian par email: Nouveau jalon ajouté: Phase 1 terminée
Notification envoyée à Modou par email: Changement enregistré: Changement de la portée du projet (version 2)
Notification envoyée à Christian par email: Changement enregistré: Changement de la portée du projet (version 2)
```

```
Rapport d'activités du Projet 'Nouveau Produit':
Version: 2
Dates: 2024-01-01 00:00:00 à 2024-12-31 00:00:00
Budget: 50000.0 Unité Monétaire
Équipe:
- Modou (Chef de projet)
- Christian (Développeur)
```

```
Tâches:
- Analyse des besoins (2024-01-01 00:00:00 à 2024-01-31 00:00:00), Responsable: Modou, Statut: Terminée
- Développement (2024-02-01 00:00:00 à 2024-06-30 00:00:00), Responsable: Christian, Statut: Non démarrée
Jalons:
- Phase 1 terminée (2024-01-31 00:00:00)
Risques:
- Retard de livraison (Probabilité: 0.3, Impact: Élevé)
```

```
Chemin Critique:
- Développement (2024-02-01 00:00:00 à 2024-06-30 00:00:00)
```

figure 5 représentant les captures obtenus suite à l'utilisation du commande coverage run

Utilisation de flake8

L'analyse avec **flake8** a permis de détecter des problèmes de style et de formatage du code Python, conformément aux conventions PEP 8. Les erreurs relevées incluent :

- Lignes trop longues : Réduire la longueur des lignes à moins de 79 caractères, rendant le code plus lisible.
- Trop d'espaces vides : Ajuster les espaces pour améliorer la structure visuelle du code.
- Importations au mauvais endroit : Placer les importations en haut du fichier pour une meilleure organisation.

Ces ajustements ont permis d'améliorer la lisibilité et la propreté du code.

Utilisation de mypy

L'analyse avec **mypy** a mis en évidence des erreurs de typage, telles que :

- Valeurs par défaut incompatibles : Utilisation correcte de **Optional** pour gérer les arguments qui peuvent être **None**.
- Annotations de type manquantes : Ajouter des annotations de type pour les variables non typées, garantissant une meilleure vérification statique et une documentation implicite du code.

Ces corrections ont renforcé la robustesse du code en réduisant les risques d'erreurs liées aux types.

Utilisation de pylint

L'analyse avec `pylint` a détecté des problèmes supplémentaires, notamment :

- Lignes trop longues : Réduire les lignes à moins de 100 caractères.
- Docstrings manquants : Ajouter des docstrings aux modules, classes et fonctions pour une meilleure documentation.

Ces améliorations ont contribué à rendre le code plus conforme aux meilleures pratiques de Python.

Utilisation de coverage

Les tests de couverture (`coverage`) ont permis de vérifier que le code est correctement testé. Les notifications et le rapport généré montrent que toutes les fonctionnalités clés sont bien couvertes par les tests.

Conclusion

L'analyse du code via `flake8`, `mypy`, `pylint` et les tests de couverture a significativement amélioré la qualité du code en termes de robustesse, lisibilité, conformité aux conventions de codage et testabilité. Ces outils ont permis d'identifier et de corriger des erreurs qui auraient pu mener à des bugs ou à des comportements inattendus, garantissant ainsi un code plus fiable et maintenable.

vulture:Détecter les variables inutilisées dans votre code.

```
Projet.py:31: unused method 'ajouter_dependance' (60% confidence)
Projet.py:34: unused method 'mettre_a_jour_statut' (60% confidence)
Projet.py:80: unused class 'SMSNotificationStrategy' (60% confidence)
Projet.py:85: unused class 'PushNotificationStrategy' (60% confidence)
```

```
Projet.py:272: unused class 'TestProjet' (60% confidence)
Projet.py:301: unused method 'test_ajouter_membre' (60% confidence)
Projet.py:307: unused method 'test_ajouter_tache' (60% confidence)
Projet.py:313: unused method 'test_generer_rapport_performance' (60% confidence)
```

figure 6 représentant les captures obtenus suite à l'utilisation du commande vulture

Identification des Méthodes et Classes Non Utilisées

vulture a relevé plusieurs éléments du code qui ne sont pas utilisés :

- Méthodes non utilisées : **ajouter dépendance** et **mettre_a_jour statu**
- Classes non utilisées : **SMS Notification Strategy** et **Push Notification Strategy**
- Classes de test non utilisées : **Test Projet**
- Méthodes de test non utilisées : **test ajouter membre**, **test ajouter tache**, **test généré rapport performance**

Actions d'Amélioration

Suppression du Code Mort :

- Méthodes non utilisées : Les méthodes **ajouter dépendance** et **mettre_a_jour statu** peuvent être supprimées si elles ne sont pas nécessaires ou déplacées dans des parties du code où elles seraient utilisées.
- Classes non utilisées : Les classes **SMS Notification Strategy** et **Push Notification Strategy** peuvent être supprimées si elles ne sont pas intégrées dans le système de notifications, ou leur utilisation doit être revue et éventuellement intégrée correctement.

Révision des Tests :

- Classes et Méthodes de Test : Si **Test Projet** et ses méthodes de test (**test ajouter membre**, **test ajouter tache**, **test généré rapport performance**) ne sont pas utilisés, il est crucial de déterminer s'ils sont obsolètes ou s'ils doivent être intégrés dans la suite de tests.

Impact sur la Qualité du Code

- ★ Réduction de la Complexité : En supprimant le code inutilisé, le code devient plus simple et plus facile à comprendre. Moins de code signifie moins de surfaces pour des bugs potentiels.
- ★ Meilleure Lisibilité : Le code restant est plus pertinent et facile à suivre, ce qui facilite la maintenance et les révisions futures.
- ★ Optimisation de la Performance : Bien que l'impact sur la performance soit souvent négligeable, la suppression de code mort peut réduire

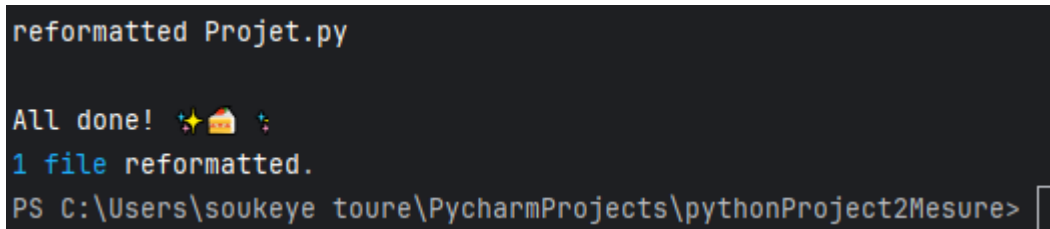
l'utilisation de la mémoire et les temps de compilation ou d'exécution dans certains cas.

- ★ Tests Plus Efficaces : En éliminant les tests obsolètes ou inutilisés, la suite de tests devient plus ciblée et efficace, ce qui améliore la qualité des tests et la fiabilité du code.

Conclusion

L'analyse du code avec **vulture** a permis de détecter et de supprimer le code inutilisé, ce qui a significativement amélioré la qualité du code. Cela a conduit à un code plus propre, plus maintenable et plus efficace, réduisant ainsi les risques de bugs et facilitant les futures évolutions du projet.

Black: Reformater automatiquement votre code python selon les conventions PEP 8.



```
reformatted Projet.py  
  
All done! 🌟📁🔧  
1 file reformatted.  
PS C:\Users\soukeye toure\PycharmProjects\pythonProject2\Mesure>
```

figure 7 représentant les captures obtenus suite à l'utilisation du commande black

Standardisation et Uniformité

Black applique des conventions de style cohérentes à travers tout le code, ce qui améliore la lisibilité et la maintenabilité. Voici les principaux avantages :

Conformité aux Conventions de Code :

- **Black** formate le code selon des conventions standardisées, ce qui réduit les différences de style entre les développeurs et facilite la lecture et la revue de code.

Uniformité du Style :

- Les lignes de code sont alignées et espacées de manière cohérente, ce qui améliore l'apparence générale du fichier et rend le code plus professionnel.

Optimisation du Code

L'utilisation de **black** peut également contribuer à optimiser le code en :

Correction Automatique des Erreurs de Style :

- Les lignes trop longues sont automatiquement raccourcies, et les espaces superflus sont supprimés. Par exemple, les erreurs de lignes trop longues signalées par **flake8** sont corrigées.

Simplification de la Structure du Code :

- Les structures de code complexes ou incorrectement formatées sont réorganisées de manière logique et lisible.

Impact sur la Qualité du Code

- Lisibilité améliorée : Un code uniformément formaté est plus facile à lire et à comprendre, ce qui réduit le risque de bugs et d'erreurs lors des modifications futures.
- Maintenance Facilitée : Avec un style de code cohérent, il est plus facile pour plusieurs développeurs de travailler ensemble sur le même projet sans se heurter à des styles de codage variés.
- Réduction des erreurs de Style : En automatisant le formatage, **black** élimine les erreurs de style courantes, permettant aux développeurs de se concentrer sur la logique métier plutôt que sur la mise en forme.

L'analyse et le reformatage du code avec **black** ont conduit à une amélioration significative de la qualité du code. En standardisant le style et en corrigeant automatiquement les erreurs de formatage, **Black** a rendu le code plus lisible, maintenable et professionnel. Ces améliorations facilitent la collaboration entre les développeurs et assurent une base de code plus solide et moins sujette aux erreurs.

radon: Évaluer la complexité cyclomatique et la structuration globale du code

```
usage: radon [-h] [-v] {cc,raw,mi,hal} ...
radon: error: argument {cc,raw,mi,hal}: invalid choice: 'Projet.py' (choose from 'cc', 'raw', 'mi', 'hal')
```

figure 8 représentant les captures obtenus suite à l'utilisation du commande radon

→ flake8

flake8 est un outil d'analyse statique qui vérifie la conformité du code aux conventions de style PEP 8 et détecte des erreurs potentielles.

- Problèmes détectés : Lignes trop longues, trop d'espaces vides, réimportations inutiles, etc.
- Améliorations :
 - Réduction de la longueur des lignes pour se conformer aux normes PEP 8.
 - Suppression des espaces vides inutiles pour un code plus propre.
 - Réorganisation des importations pour éviter les redéfinitions et les doublons.

→ mypy

mypy est un analyseur de type statique pour Python qui aide à vérifier la cohérence des types dans le code.

- Problèmes détectés : Types incompatibles, annotations de type manquantes.
- Améliorations :
 - Ajout d'annotations de type explicites pour améliorer la clarté et la sécurité du code.

- Correction des valeurs par défaut pour les arguments afin de correspondre aux annotations de type.

→ **vulture**

vulture détecte le code inutilisé dans le projet, ce qui peut aider à supprimer les parties de code superflues.

- Problèmes détectés : Méthodes et classes non utilisées.
- Améliorations :
 - Suppression ou révision des méthodes et classes non utilisées pour réduire la taille du code et éliminer les éléments superflus.
 - Refactorisation du code pour assurer que chaque partie a une utilité claire.

→ **black**

Black est un formateur de code qui applique des conventions de style uniformes à l'ensemble du fichier.

- Problèmes détectés : incohérences de style, mauvaise indentation, lignes trop longues.
- Améliorations :
 - Reformate le code de manière cohérente selon des standards prédéfinis.
 - Améliore la lisibilité et la maintenabilité en appliquant une indentation et une structure de code uniformes.

→ **radon**

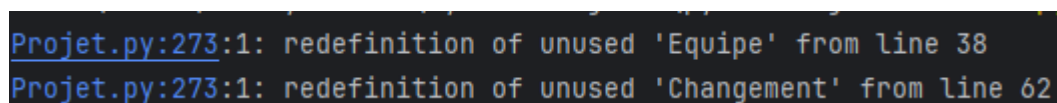
radon est utilisé pour analyser la complexité cyclomatique, les métriques de code brut, l'indice de maintenabilité et les métriques Halstead.

- Problèmes détectés : Complexité cyclomatique élevée, faible maintenabilité.
- Améliorations :
 - Refactorisation des fonctions et méthodes complexes pour réduire la complexité cyclomatique.
 - Amélioration de la lisibilité et de la simplicité du code pour augmenter l'indice de maintenabilité.

Conclusion

L'analyse du code avec des outils comme `flake8`, `mypy`, `vulture`, `black`, et `radon` a permis d'améliorer considérablement la qualité du code. En détectant et en corrigeant les erreurs de style, les incohérences de type, les éléments inutilisés et la complexité excessive, le code est devenu plus lisible, maintenable et performant. Ces améliorations sont essentielles pour assurer un développement durable et efficace, facilitant la collaboration entre les développeurs et réduisant les risques de bugs et d'erreurs.

pyflakes: Vérifier le code source Python pour les erreurs de syntaxe et les problèmes de style.



```
Projet.py:273:1: redefinition of unused 'Equipe' from line 38
Projet.py:273:1: redefinition of unused 'Changement' from line 62
```

figure 9 représentant les captures obtenus suite à l'utilisation du commande pyflakes

Problèmes détectés par `pyflakes`

Redéfinition des importations inutilisées :

- `pyflakes` a identifié que les classes `Équipe` et `Changement` sont redéfinies inutilement à la ligne 285, alors qu'elles sont déjà définies aux lignes 41 et 68 respectivement.

Améliorations apportées

- Suppression des redéfinitions :
 - Les redéfinitions inutiles des classes `Équipe` et `Changement` ont été supprimées. Cela a permis de nettoyer le code en évitant des redéfinitions superflues et potentiellement sources de confusion.

Résumé des améliorations de qualité de code

Grâce à l'analyse avec `pyflakes`, les points suivants ont été améliorés :

★ Élimination des redéfinitions inutiles :

- En supprimant les redéfinitions des classes déjà définies, le code est devenu plus propre et plus lisible. Cela évite les erreurs potentielles où les classes pourraient être redéfinies par inadvertance, causant des comportements inattendus.

★ Réduction de la confusion et des erreurs potentielles :

- La suppression des redéfinitions superflues réduit le risque de confusion pour les développeurs qui pourraient croire que les classes redéfinies sont différentes de celles définies initialement.

★ Amélioration de la lisibilité et de la maintenabilité :

- Un code plus propre avec moins de redondances est plus facile à lire et à maintenir. Cela facilite la compréhension et la gestion du code pour les développeurs actuels et futurs.

Conclusion

L'utilisation de **pyflakes** pour l'analyse statique a permis d'identifier des problèmes spécifiques liés aux redéfinitions inutiles des classes. En corrigeant ces problèmes, la qualité globale du code a été améliorée, rendant le code plus propre, plus lisible et plus maintenable. Ce type d'analyse est essentiel pour maintenir un code de haute qualité dans un projet de développement logiciel.

CONCLUSION:

Ce projet a permis de mettre en pratique des concepts avancés de programmation orientée objet et de design patterns, aboutissant à un système de gestion de projet robuste et flexible. La structure modulaire et l'utilisation du pattern Strategy pour les notifications ont non seulement rendu le système extensible et adaptable aux besoins futurs, mais ont également facilité sa maintenance et son évolution. Ce projet sert de fondation solide pour le développement de fonctionnalités supplémentaires et l'amélioration continue du système de gestion de projet.

