Rapport de test

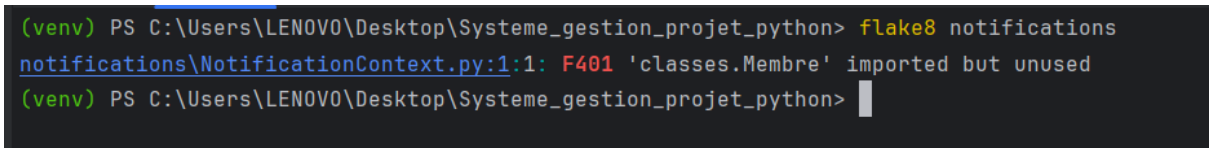
Utilisation de flake8 :

L'analyse de code par flake8 a révélé certains problèmes de formatage :

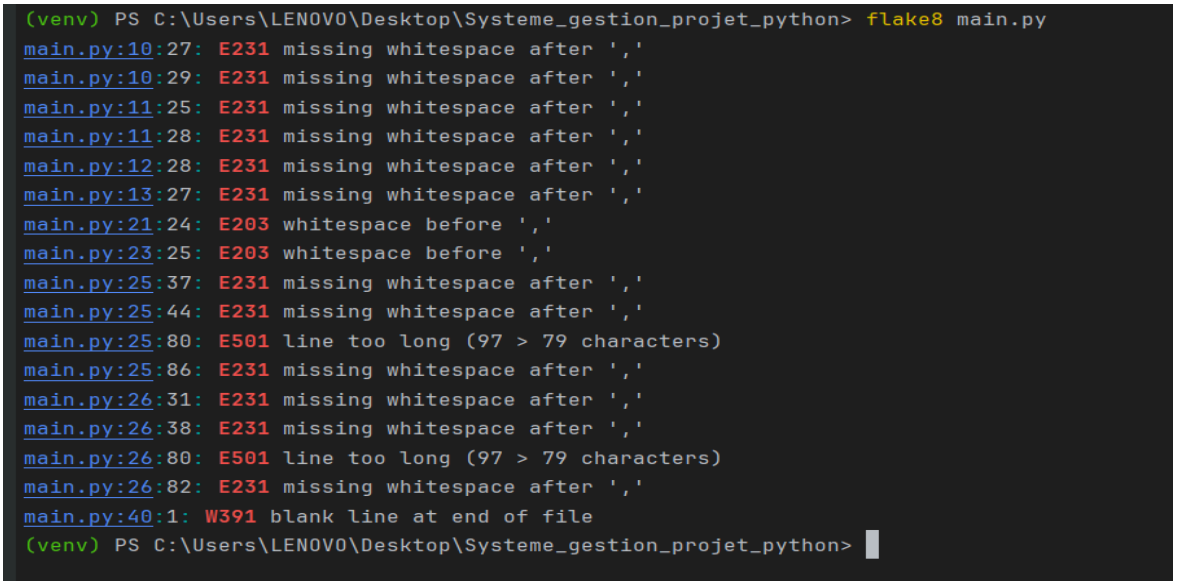
Dossier Classes : De nombreuses lignes dépassent la limite de 79 caractères (E501), des points-virgules inutiles sont présents en fin d'instruction (E703), il y a trop de lignes vides entre les fonctions/classes (E303), un espace supplémentaire est présent avant un crochet fermant] (E202), et certains fichiers manquent d'une ligne vide à la fin (W391). La correction de ces derniers nous à permis d'améliorer la lisibilité et la maintenabilité du code.

Dossier Notifications : Dans le fichier notifications/NotificationContext.py, l'erreur F401 indique que le module 'classes.Membre' est importé mais non utilisé.

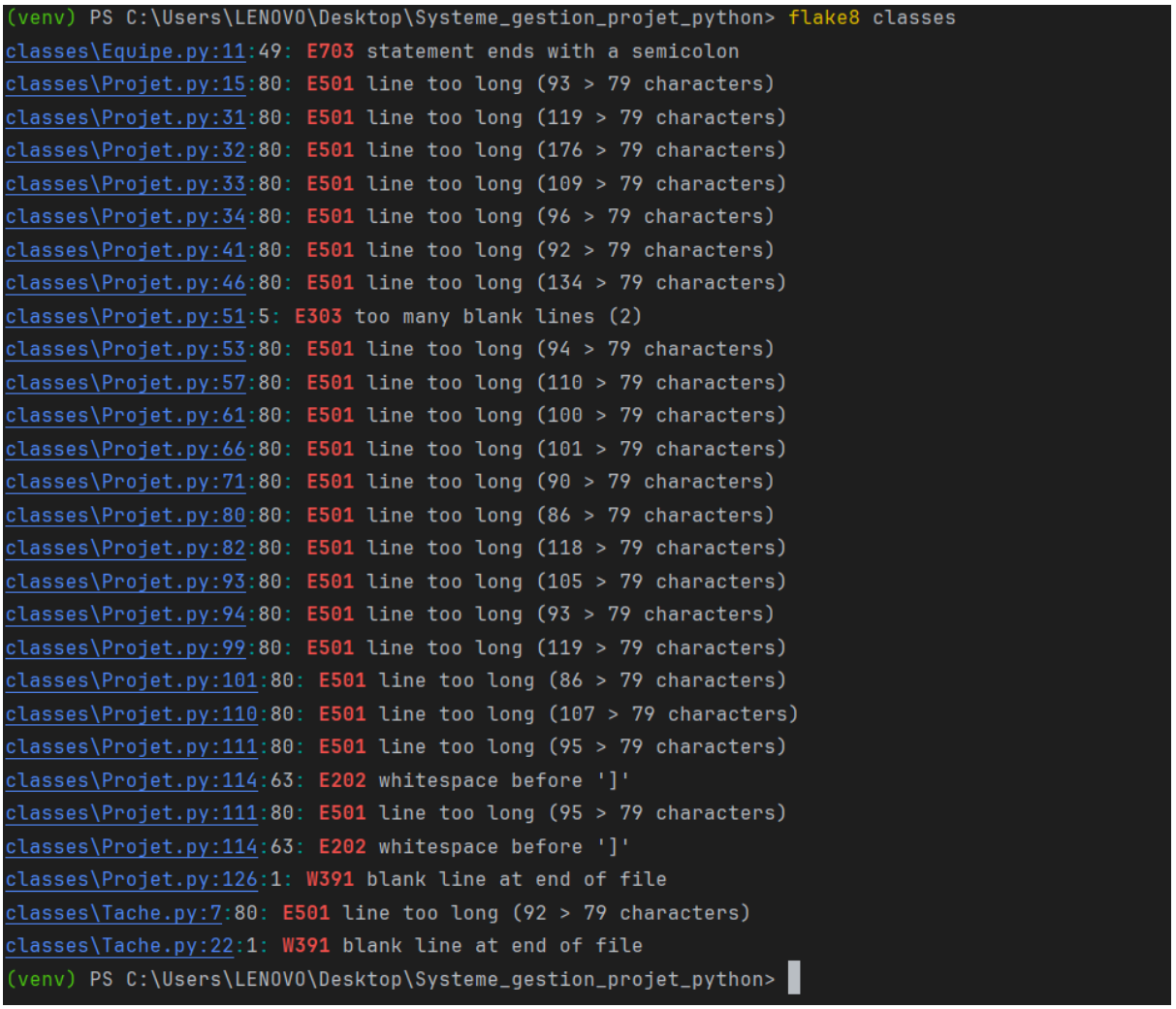
Fichier main.py : Beaucoup de problèmes de formatage ont été détectés : espace manquant après certaines virgules (E231) aux lignes 10, 11, 12, 13, 25, 26, espace inutile avant certaines virgules (E203) aux lignes 21, 23, lignes trop longues dépassant 79 caractères (E501) aux lignes 25 et 26, et absence de ligne vide à la fin du fichier (W391).



capture: dossier Notifications



capture : fichier main.py



capture : dossier classes

L'utilisation de la commande '**flake8'** nous a permis d'identifier les endroits où les conventions de codage Python n'étaient pas respectées et de corriger ces erreurs. Ces corrections ont amélioré la lisibilité, la compréhension et la maintenabilité du code.

Utilisation de pylint :

L'analyse par pylint a révélé des problèmes dans les dossiers du projet.

Dossier Classes : Violations des conventions de nommage (snake_case non respecté), docstrings manquants pour les modules et les classes, et certaines classes ayant moins de deux méthodes publiques.

Dossier Notifications : Importations de modules inexistants ou mal nommés (classes et **notifications.NotificationStrategy**), noms de modules ne respectant pas la convention snake_case, docstrings manquants, et certaines classes avec trop peu de méthodes publiques.

De plus, **NotificationContext** importe "Membre" sans l'utiliser, et l'ordre des imports est incorrect dans NotificationStrategy.py.

Fichier main.py : Note de 9.38/10, indiquant un code de qualité, mais un docstring manque pour le module (C0114) et une ligne vide superflue est présente à la fin du fichier (C0305).

```
(venv) PS C:\Users\LENOVO\Desktop\Systeme_gestion_projet_python> pylint classes
************* Module classes.Changement
classes\Changement.py:1:0: C0114: Missing module docstring (missing-module-docstring)
classes\Changement.py:1:0: C0103: Module name "Changement" doesn't conform to snake_case naming style (invalid-name)
classes\Changement.py:4:0: C0115: Missing class docstring (missing-class-docstring)
classes\Changement.py:4:0: R0903: Too few public methods (0/2) (too-few-public-methods)
************* Module classes.Equipe
classes\Equipe.py:11:0: W0301: Unnecessary semicolon (unnecessary-semicolon)
classes\Equipe.py:1:0: C0114: Missing module docstring (missing-module-docstring)
classes\Equipe.py:1:0: C0103: Module name "Equipe" doesn't conform to snake_case naming style (invalid-name)
classes\Equipe.py:4:0: C0115: Missing class docstring (missing-class-docstring)
classes\Equipe.py:14:4: C0116: Missing function or method docstring (missing-function-docstring)
classes\Equipe.py:17:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module classes.Jalon
classes\Jalon.py:1:0: C0114: Missing module docstring (missing-module-docstring)
classes\Jalon.py:1:0: C0103: Module name "Jalon" doesn't conform to snake_case naming style (invalid-name)
classes\Jalon.py:4:0: C0115: Missing class docstring (missing-class-docstring)
classes\Jalon.py:4:0: R0903: Too few public methods (0/2) (too-few-public-methods)
************* Module classes.Membre
classes\Membre.py:1:0: C0114: Missing module docstring (missing-module-docstring)
classes\Membre.py:1:0: C0103: Module name "Membre" doesn't conform to snake_case naming style (invalid-name)
classes\Membre.py:1:0: C0115: Missing class docstring (missing-class-docstring)
classes\Membre.py:1:0: R0903: Too few public methods (0/2) (too-few-public-methods)
************* Module classes.Projet
classes\Projet.py:31:0: C0301: Line too long (119/100) (line-too-long)
classes\Projet.py:32:0: C0301: Line too long (176/100) (line-too-long)
classes\Projet.py:33:0: C0301: Line too long (109/100) (line-too-long)
classes\Projet.py:46:0: C0301: Line too long (134/100) (line-too-long)
classes\Projet.py:57:0: C0301: Line too long (110/100) (line-too-long)
classes\Projet.py:66:0: C0301: Line too long (101/100) (line-too-long)
```

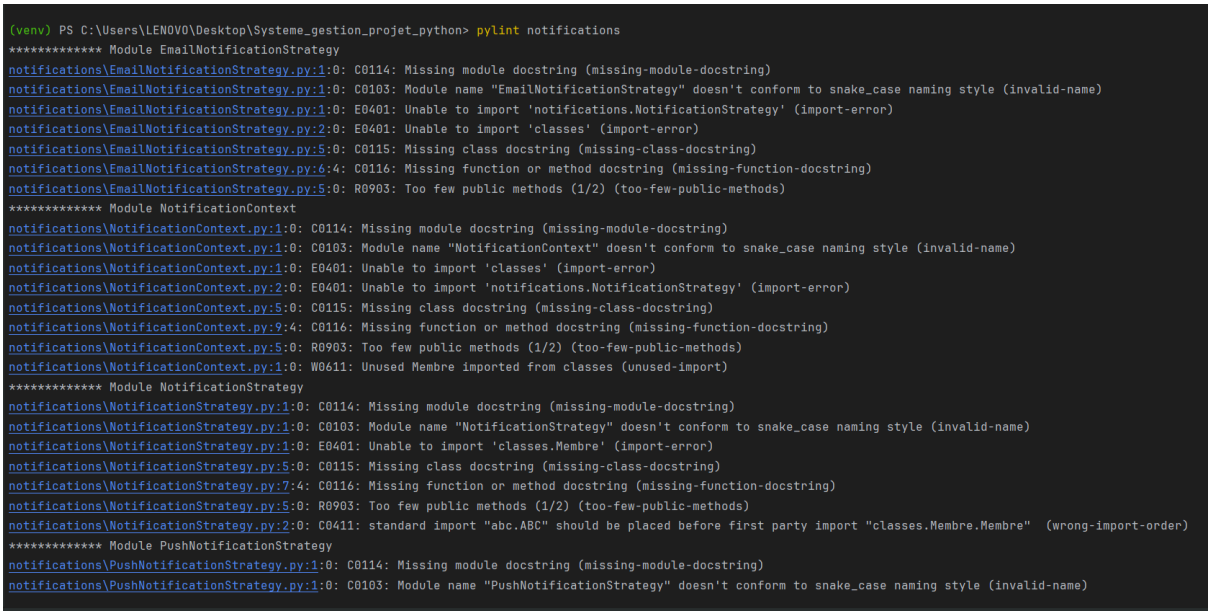
capture : dossier classes

```
(venv) PS C:\Users\LENOVO\Desktop\Systeme_gestion_projet_python> pylint main.py
************** Module main
main.py:40:0: C0305: Trailing newlines (trailing-newlines)
main.py:1:0: C0114: Missing module docstring (missing-module-docstring)

-----------------------------------
Your code has been rated at 9.38/10

(venv) PS C:\Users\LENOVO\Desktop\Systeme_gestion_projet_python>
```

capture : fichier main.py



capture : dossier notifications

La commande Python **‘pylint’** nous a aidés à améliorer la documentation au niveau des méthodes, des modules, et d'autres éléments du code.

Utilisation de mypy:

L 'exécution de mypy a révélé plusieurs problèmes de typage dans le code :

Dossier Classe : Types incorrects dans Tache.py (lignes 7 et 8) avec datetime et classes.Membre, variables non annotées dans Projet.py (taches, risques), et une erreur d'affectation dans Projet.py (ligne 37) où un objet NotificationContext est assigné à une variable censée contenir None.

Dossier Notifications : Avec l'option --explicit-package-bases, mypy a trouvé des références invalides à classes.Membre et un accès incorrect à l'attribut nom.

Fichier main.py : À la ligne 15, les arguments sont de type date au lieu de datetime. Erreurs communes dans d'autres fichiers incluent l'utilisation incorrecte de datetime et classes.Membre comme types, ainsi que des variables non annotées dans Projet.py.

```
(venv) PS C:\Users\LENOVO\Desktop\Systeme_gestion_projet_python> mypy classes
classes\Tache.py:7: error: Module "datetime" is not valid as a type  [valid-type]
classes\Tache.py:7: note: Perhaps you meant to use a protocol matching the module structure?
classes\Tache.py:8: error: Module "classes.Membre" is not valid as a type  [valid-type]
classes\Tache.py:8: note: Perhaps you meant to use a protocol matching the module structure?
classes\Tache.py:15: error: Need type annotation for "dependances" (hint: "dependances: list[<type>] = ...")  [var-annotated]
classes\Jalon.py:5: error: Module "datetime" is not valid as a type  [valid-type]
classes\Jalon.py:5: note: Perhaps you meant to use a protocol matching the module structure?
classes\Changement.py:5: error: Module "datetime" is not valid as a type  [valid-type]
classes\Changement.py:5: note: Perhaps you meant to use a protocol matching the module structure?
classes\Projet.py:20: error: Need type annotation for "taches" (hint: "taches: list[<type>] = ...")  [var-annotated]
classes\Projet.py:23: error: Need type annotation for "risques" (hint: "risques: list[<type>] = ...")  [var-annotated]
classes\Projet.py:24: error: Need type annotation for "jalons" (hint: "jalons: list[<type>] = ...")  [var-annotated]
classes\Projet.py:26: error: Need type annotation for "changements" (hint: "changements: list[<type>] = ...")  [var-annotated]
classes\Projet.py:27: error: Need type annotation for "chemin_critique" (hint: "chemin_critique: list[<type>] = ...")  [var-annotated]
classes\Projet.py:37: error: Incompatible types in assignment (expression has type "NotificationContext", variable has type "None")  [assignment]
Found 11 errors in 4 files (checked 8 source files)
(venv) PS C:\Users\LENOVO\Desktop\Systeme_gestion_projet_python>
```

capture : dossier classes

```
(venv) PS C:\Users\LENOVO\Desktop\Systeme_gestion_projet_python> mypy --explicit-package-bases notifications
notifications\SMSNotificationStrategy.py:6: error: Module "classes.Membre" is not valid as a type  [valid-type]
notifications\SMSNotificationStrategy.py:6: note: Perhaps you meant to use a protocol matching the module structure?
notifications\SMSNotificationStrategy.py:7: error: Membre? has no attribute "nom"  [attr-defined]
notifications\PushNotificationStrategy.py:6: error: Module "classes.Membre" is not valid as a type  [valid-type]
notifications\PushNotificationStrategy.py:6: note: Perhaps you meant to use a protocol matching the module structure?
notifications\PushNotificationStrategy.py:7: error: Membre? has no attribute "nom"  [attr-defined]
notifications\EmailNotificationStrategy.py:6: error: Module "classes.Membre" is not valid as a type  [valid-type]
notifications\EmailNotificationStrategy.py:6: note: Perhaps you meant to use a protocol matching the module structure?
notifications\EmailNotificationStrategy.py:7: error: Membre? has no attribute "nom"  [attr-defined]
Found 6 errors in 3 files (checked 5 source files)
(venv) PS C:\Users\LENOVO\Desktop\Systeme_gestion_projet_python>
```

capture : dossier notifications

```
(venv) PS C:\Users\LENOVO\Desktop\Systeme_gestion_projet_python> mypy main.py
classes\Changement.py:5: error: Module "datetime" is not valid as a type  [valid-type]
classes\Changement.py:5: note: Perhaps you meant to use a protocol matching the module structure?
notifications\EmailNotificationStrategy.py:6: error: Module "classes.Membre" is not valid as a type  [valid-type]
notifications\EmailNotificationStrategy.py:6: note: Perhaps you meant to use a protocol matching the module structure?
notifications\EmailNotificationStrategy.py:7: error: Membre? has no attribute "nom"  [attr-defined]
classes\Jalon.py:5: error: Module "datetime" is not valid as a type  [valid-type]
classes\Jalon.py:5: note: Perhaps you meant to use a protocol matching the module structure?
classes\Tache.py:7: error: Module "datetime" is not valid as a type  [valid-type]
classes\Tache.py:7: note: Perhaps you meant to use a protocol matching the module structure?
classes\Tache.py:8: error: Module "classes.Membre" is not valid as a type  [valid-type]
classes\Tache.py:8: note: Perhaps you meant to use a protocol matching the module structure?
classes\Tache.py:15: error: Need type annotation for "dependances" (hint: "dependances: list[<type>] = ...")  [var-annotated]
classes\Projet.py:20: error: Need type annotation for "taches" (hint: "taches: list[<type>] = ...")  [var-annotated]
classes\Projet.py:23: error: Need type annotation for "risques" (hint: "risques: list[<type>] = ...")  [var-annotated]
classes\Projet.py:24: error: Need type annotation for "jalons" (hint: "jalons: list[<type>] = ...")  [var-annotated]
classes\Projet.py:26: error: Need type annotation for "changements" (hint: "changements: list[<type>] = ...")  [var-annotated]
classes\Projet.py:27: error: Need type annotation for "chemin_critique" (hint: "chemin_critique: list[<type>] = ...")  [var-annotated]
classes\Projet.py:37: error: Incompatible types in assignment (expression has type "NotificationContext", variable has type "None")  [assignment]
main.py:15: error: Argument 3 to "Projet" has incompatible type "date"; expected "datetime"  [arg-type]
main.py:15: error: Argument 4 to "Projet" has incompatible type "date"; expected "datetime"  [arg-type]
Found 15 errors in 6 files (checked 1 source file)
(venv) PS C:\Users\LENOVO\Desktop\Systeme_gestion_projet_python>
```

capture : dossier mypy

La commande **‘mypy’** en Python a identifié des erreurs de typage dans certains attributs des classes et des erreurs d'importation de modules, nous permettant ainsi de corriger ces problèmes.

Utilisation de coverage :

```
(venv) PS C:\Users\LENOVO\Desktop\certificat_phyton\Systeme_gestion_projet_python> coverage run --source=classes,notifications -m unittest discover -s C:\Users\LENOVO\Desktop\certi
ficat_phyton\Systeme_gestion_projet_python\test
.................
----------------------------------------------------------------------
Ran 17 tests in 0.018s

OK
(venv) PS C:\Users\LENOVO\Desktop\certificat_phyton\Systeme_gestion_projet_python>
```

Cette capture montre que l'exécution des tests par le module unittest se sont déroulées correctement. Et que toutes les méthodes de la classe test marchent très bien dans ce cas si les tests sont aux nombres de 17.

```
(venv) PS C:\Users\LENOVO\Desktop\certificat_phyton\Systeme_gestion_projet_python> coverage report
Name                                          Stmts   Miss  Cover
-----------------------------------------------------------------
classes\Changement.py                             6      0   100%
classes\Equipe.py                                13      4    69%
classes\Jalon.py                                  5      0   100%
classes\Membre.py                                 4      0   100%
classes\Projet.py                                92     12    87%
classes\Risque.py                                 5      0   100%
classes\Tache.py                                 16      0   100%
classes\__init__.py                               0      0   100%
notifications\EmailNotificationStrategy.py        5      0   100%
notifications\NotificationContext.py              7      0   100%
notifications\NotificationStrategy.py             6      1    83%
notifications\PushNotificationStrategy.py         5      0   100%
notifications\SMSNotificationStrategy.py          5      0   100%
-----------------------------------------------------------------
TOTAL                                           169     17    90%
(venv) PS C:\Users\LENOVO\Desktop\certificat_phyton\Systeme_gestion_projet_python>
```

Le rapport de couverture :

- Stmts (Statements) : Nombre total de lignes de code exécutables dans le fichier.
- Miss (Missed Statements) : Nombre de lignes de code qui n'ont pas été exécutées.
- Cover (Coverage) : Pourcentage de couverture des tests.

Dans la dernière ligne du rapport, on a la moyenne de la couverture du code. Ici c’est **90 %.** Une bonne couverture de test, supérieure à 80 %, est signe d'un **projet bien testé** et auquel il est plus facile d'ajouter de nouvelles fonctionnalités.

Utilisation de vulture :

```
(venv) PS D:\Porjet\Systeme_gestion_projet_python> vulture classes
classes\Projet.py:13: unused class 'Projet' (60% confidence)
classes\Projet.py:36: unused method 'set_notification_strategy' (60% confidence)
classes\Projet.py:39: unused method 'ajouter_tache' (60% confidence)
classes\Projet.py:51: unused method 'ajouter_membre_equipe' (60% confidence)
classes\Projet.py:55: unused method 'definir_budget' (60% confidence)
classes\Projet.py:59: unused method 'ajouter_risque' (60% confidence)
classes\Projet.py:69: unused method 'ajouter_jalon' (60% confidence)
classes\Projet.py:79: unused method 'enregistrer_changement' (60% confidence)
classes\Projet.py:84: unused method 'calculer_chemin_critique' (60% confidence)
classes\Tache.py:17: unused method 'ajouter_dependance' (60% confidence)
classes\Tache.py:20: unused method 'mettre_a_jour_statut' (60% confidence)
(venv) PS D:\Porjet\Systeme_gestion_projet_python> vulture notifications
notifications\EmailNotificationStrategy.py:5: unused class 'EmailNotificationStrategy' (60% confidence)
notifications\NotificationContext.py:5: unused class 'NotificationContext' (60% confidence)
notifications\NotificationContext.py:9: unused method 'notifier' (60% confidence)
notifications\PushNotificationStrategy.py:5: unused class 'PushNotificationStrategy' (60% confidence)
notifications\SMSNotificationStrategy.py:5: unused class 'SMSNotificationStrategy' (60% confidence)
(venv) PS D:\Porjet\Systeme_gestion_projet_python>
```

Ce test nous a permis d'identifier les méthodes dans notre code qui ne sont jamais appelées. Cela inclut des méthodes au sein des classes qui ne sont référencées nulle part dans le code.

Utilisation de Black :

Contrairement aux outils de test, Black ne fournit pas d'analyse sur l'utilisation ou la qualité du code. L'utilisation de Black nous a permis de formater de manière cohérente tous les fichiers de code du projet. Cela facilite la lecture et la maintenance du code par tous les membres de l'équipe.

```
Terminal   Local  ×  +  ⌄
(venv) PS D:\Porjet\Systeme_gestion_projet_python> black classes
reformatted D:\Porjet\Systeme_gestion_projet_python\classes\Tache.py
reformatted D:\Porjet\Systeme_gestion_projet_python\classes\Projet.py

All done! ✨🍰 ✨
2 files reformatted, 6 files left unchanged.
(venv) PS D:\Porjet\Systeme_gestion_projet_python> black notifications
All done! ✨🍰 ✨
5 files left unchanged.
(venv) PS D:\Porjet\Systeme_gestion_projet_python>
```

Utilisation de pyflakes :

Pyflakes, après avoir analysé le code source, n'a détecté aucune erreur syntaxique ni problème de style, à l'exception du fichier *"**NotificationContext.py**"* où il a relevé qu'une classe "Membre" est importée mais jamais utilisée.

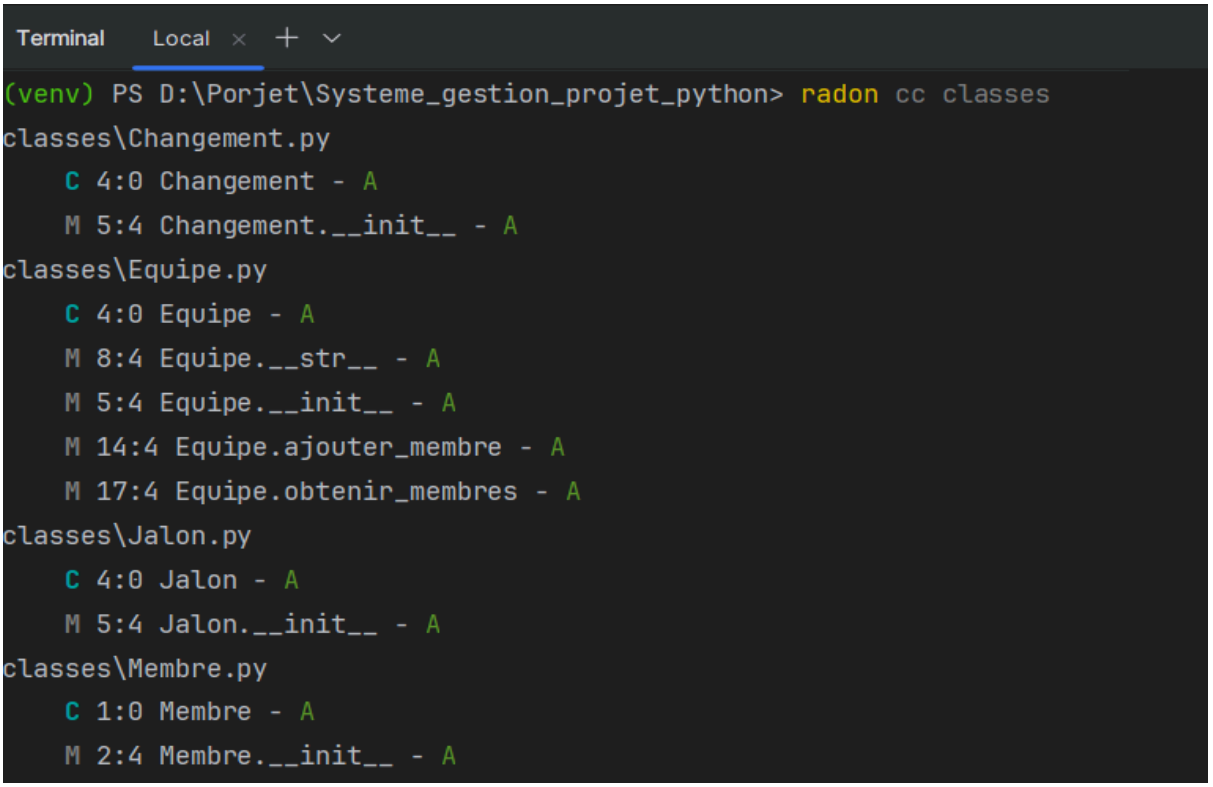
```
(venv) PS D:\Porjet\Systeme_gestion_projet_python> pyflakes classes
(venv) PS D:\Porjet\Systeme_gestion_projet_python> pyflakes notifications
notifications\NotificationContext.py:1:1: 'classes.Membre' imported but unused
(venv) PS D:\Porjet\Systeme_gestion_projet_python>
```

Utilisation de radon:

L'analyse Radon montre que les modules "notifications" et "classes" sont bien structurés et maintenables, ce qui est un excellent indicateur de la qualité de notre code source. Au cours de cette analyse, Radon a mesuré la maintenabilité (mi) ainsi que la complexité cyclomatique (cc).

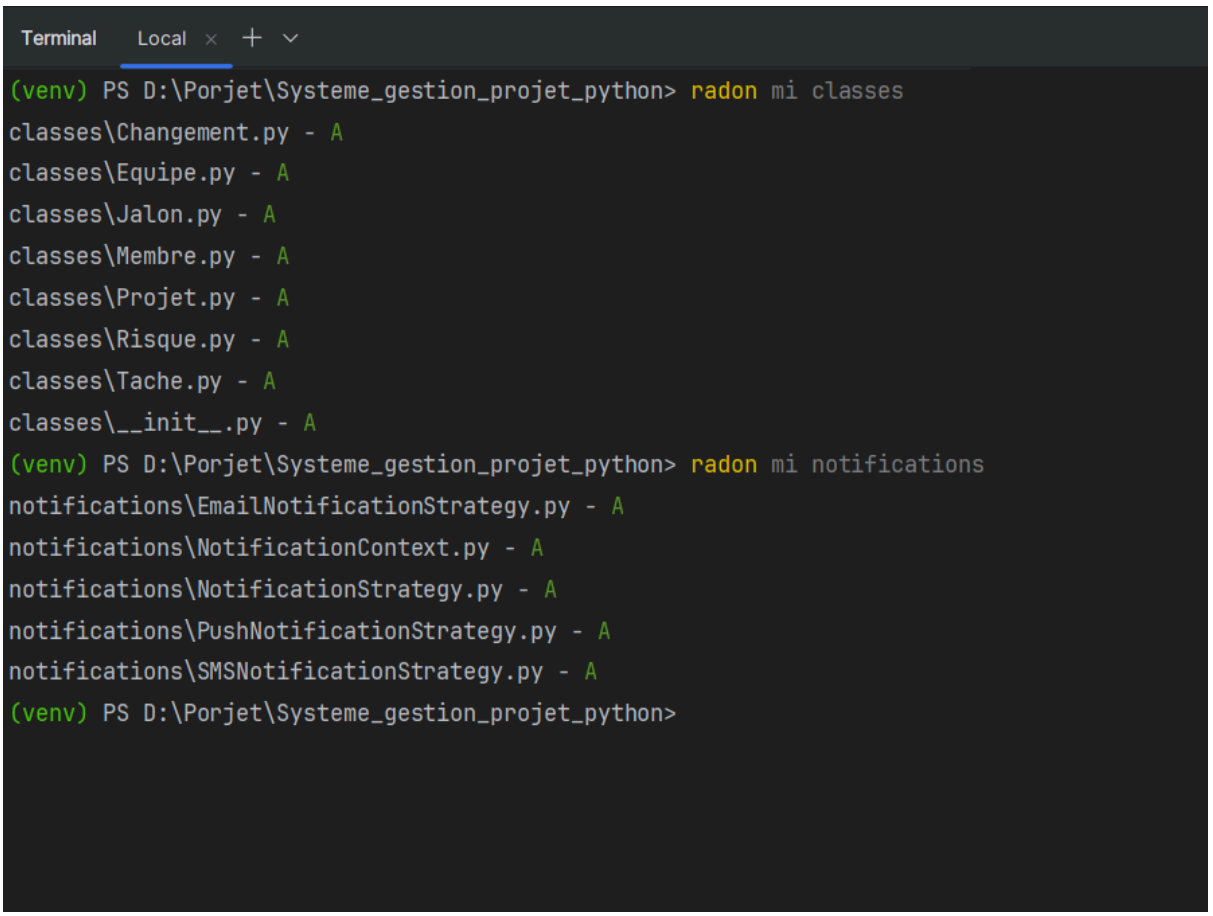
À l'exception de la méthode "***Projet.calculer_chemin_critique***", qui a reçu une note "C" indiquant une complexité modérée, toutes les classes et méthodes ont obtenu une note "A". Cela signifie que ces fichiers ont un excellent niveau de maintenabilité ou encore sont simples à comprendre.

- **Mesure de la complexe cyclomatique**


Terminal
Local
(venv) PS D:\Porjet\Systeme_gestion_projet_python> radon cc classes
classes\Changement.py
C 4:0 Changement - A
M 5:4 Changement.__init__ - A
classes\Equipe.py
C 4:0 Equipe - A
M 8:4 Equipe.__str__ - A
M 5:4 Equipe.__init__ - A
M 14:4 Equipe.ajouter_membre - A
M 17:4 Equipe.obtenir_membres - A
classes\Jalon.py
C 4:0 Jalon - A
M 5:4 Jalon.__init__ - A
classes\Membre.py
C 1:0 Membre - A
M 2:4 Membre.__init__ - A


classes\Projet.py
M 84:4 Projet.calculer_chemin_critique - C
C 13:0 Projet - A
M 43:4 Projet.afficher_taches - A
M 63:4 Projet.afficher_risques - A
M 73:4 Projet.afficher_jalons - A
M 116:4 Projet.afficher_chemin_critique - A
M 122:4 Projet.notifier - A
M 15:4 Projet.__init__ - A
M 30:4 Projet.__str__ - A
M 36:4 Projet.set_notification_strategy - A
M 39:4 Projet.ajouter_tache - A
M 51:4 Projet.ajouter_membre_equipe - A
M 55:4 Projet.definir_budget - A
M 59:4 Projet.ajouter_risque - A
M 69:4 Projet.ajouter_jalon - A
M 79:4 Projet.enregistrer_changement - A
classes\Risque.py
C 1:0 Risque - A
M 2:4 Risque.__init__ - A
classes\Tache.py
C 6:0 Tache - A
M 7:4 Tache.__init__ - A
M 17:4 Tache.ajouter_dependance - A
M 20:4 Tache.mettre_a_jour_statut - A

- **Mesure de la maintenabilité**



En résumé, l'utilisation de ces commandes nous a permis d'analyser la qualité de notre code, d'y apporter des modifications, d'identifier et de corriger plusieurs problèmes, de supprimer des méthodes non utilisées, améliorant ainsi la lisibilité, la maintenabilité et la qualité globale du projet.