# Contexte

Suite à une étude de santé publique sur les étudiants en cursus informatique, le ministère de la santé nous a demandé de développer une application permettant la réalisation d'un parcours virtuel en pratiquant une activité sportive dans le réel.  Elle permet de créer des parcours virtuels via un site web et via une application mobile de se déplacer en marchant, en courant ou en faisant du vélo.

Dans la mise en place de l’application, il nous a paru judicieux de déployer un outil nous permettant d’analyser la qualité du code afin d’avoir un code robuste, sans bug et présentant le moins de redondance possible. En outre, cette optimisation de code permettra d’avoir un code plus facilement maintenable.

Cette application sera déclinée sur 2 médias :

* Un site web qui servira pour la gestion des parcours par les administrateurs et pour les utilisateurs, à afficher les détails de leur progression.
* Une application mobile qui servira essentiellement aux joueurs pour avancer sur leurs parcours, mais aussi de s’inscrire à d’autres challenges

Le but du présent document est de présenter l’outil utilisé et la démarche que nous avons mise en place pour améliorer la qualité du code source de notre application.

Composition de l’équipe :

* Alabic Nemanja
* Derousseaux-Lebert Nathanael
* Metzger Nicolas
* Sacher Marilyn
* Steiner Kélio

# SonarQube

## Présentation de l’outil

SonarQube est un logiciel permettant de mesurer la qualité d’un code source en continu. Il supporte plus de 25 langages et génère des reporting sur :

* Le respect de règle de programmation
* La présence de bugs ou de faille de sécurité
* La duplication de code
* La complexité du code
* La couverture de code (d'après wikipédia, la **couverture de code** est une mesure qui décrit le taux de code source exécuté d'un programme quand une suite de test est lancée.)

L’outil permet d’avoir une évolution dans le temps et des vues différentielles de l’analyse. Il est même possible d’automatiser cette analyse avec des outils tel que Maven, Ant, ...

## Comprendre une analyse SonarQube

L’analyse SonarQube se base sur une série de règles de codage et d’indicateur défini par dans l’application. Ces règles et indicateurs sont définis par défaut pour les langages pris en charge par l’application et peuvent être complétés par l’équipe qui gère l'application.

Lors de l’analyse SonarQube va soulever des “problèmes” chaque fois qu’il rencontre un élément qui va à l’encontre des règles définies. Il y a 3 types de problèmes possibles :

1. **Bug** : une erreur dans le code qui doit être fixé immédiatement
2. **Vulnerability** : un endroit de votre code qui constitue une faille de sécurité
3. **Code Smell** : un point de votre code qui est confus et peut s’avérer difficile à maintenir

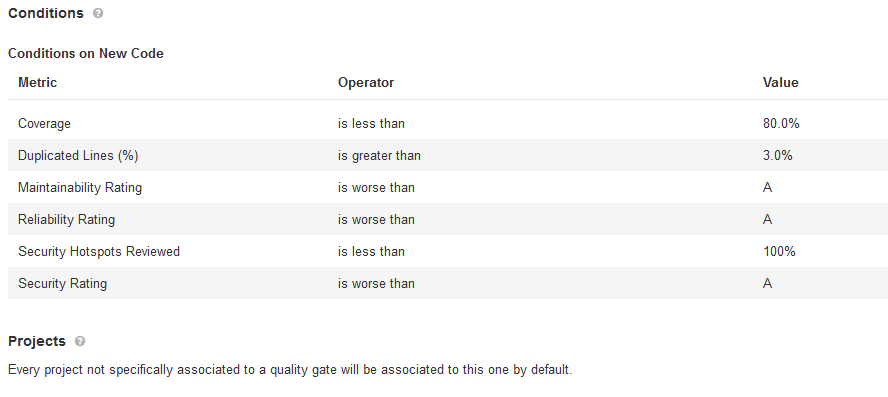
Pour chacun de ces problèmes, vous avez 5 niveaux de sévérité :

1. **Bloquant :** bug ayant un fort impact sur votre code en production qui doit être fixé immédiatement.
2. **Critique :** bug avec un faible impact qui doit être examiné.
3. **Majeur :** problème de qualité de code qui impacte la maintenabilité du code
4. **Mineur :** problème de qualité de code avec un impact faible la maintenabilité du code
5. **Info :** juste informatif

Par ailleurs, à chaque analyse SonarQube compare à la précédente analyse et affiche une analyse globale et une sur les nouveaux problèmes rencontrés depuis la précédente. L’analyse va permet d'obtenir un rapport qui affichera entre autres les informations suivantes :

* Nombre de bugs
* Nombre de code smell
* Nombre de vulnérabilités et nombre de **Security Hotspots**. Un Hotspots est d’une faille de sécurité que le développeur doit vérifier. A la différence d’une vulnérabilité, il est nécessaire de l’examiner avant de décider d'appliquer un correctif.
* Complexité : 2 mesures qui mesure la complexité simple de code, à savoir lenombre de subdivisions du code, tel que des if else,switch, case, ….. et la complexité cognitive qui représente la difficulté à suivre le « flux » du code, c-à-d le lire et le comprendre le code.
* Duplication : nombre de lignes dupliquées, nombre de fichiers contenant des duplications, nombre de lignes impliqués dans les duplications, % de duplication
* Note de maintenabilité : elle dépend du temps restant nécessaire pour corriger, par rapport à celui déjà dédié au développement
  + <=5%, la note est À
  + Entre 6 et 10 % la note est un B
  + Entre 11 et 20 % la note est un C
  + Entre 21 et 50 % la note est un D
  + Tout ce qui dépasse 50 % est un E
* Dette technique : temps nécessaire pour corriger les code smell en minutes (1 jours = 8h)
* Le statut “**Quality Gate**” : État du projet par rapport au quality gate. Les valeurs possibles sont ERROR et OK , et **Quality Gate Details** (quality\_gate\_details) : détail du quality gate des états par rapport au “**Quality Gate**”.

REM : le “**Quality Gate**” est une liste de conditions appliqués sur des mesures que l’analyse du code doit respecter, pour exemple



# Organisation du projet et technologies

Pour la mise en place de notre application, nous devons réaliser 3 parties distinctes :

* Une API qui gérera l'accès à la base de données, la création/modification des données et les diverses requêtes d'extraction des données de la base
* Un site web qui permettra la gestion des challenges par l’administrateur
* Une application mobile qui sera utilisé par le joueur pour se déplacer sur un parcours

Le projet est divisé comme suit :

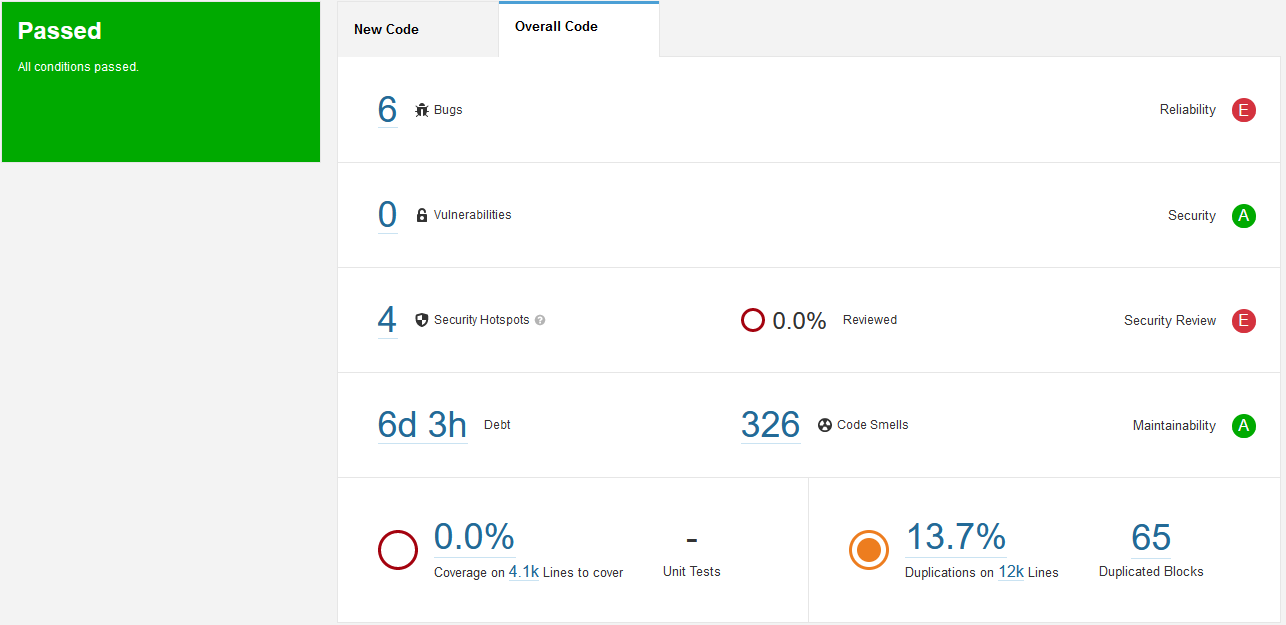
* Un répertoire backend dans lequel se trouve server où se trouve l’ensemble des répertoires, librairies et fichiers nécessaires
* Un répertoire frontend avec deux sous répertoires :
  + Web où se trouve l’ensemble des répertoires, librairies et fichiers nécessaires au site web
  + Mobile où se trouve l’ensemble des répertoires, librairies et fichiers nécessaires à l’application mobile

En ce qui concerne les technologies employées :

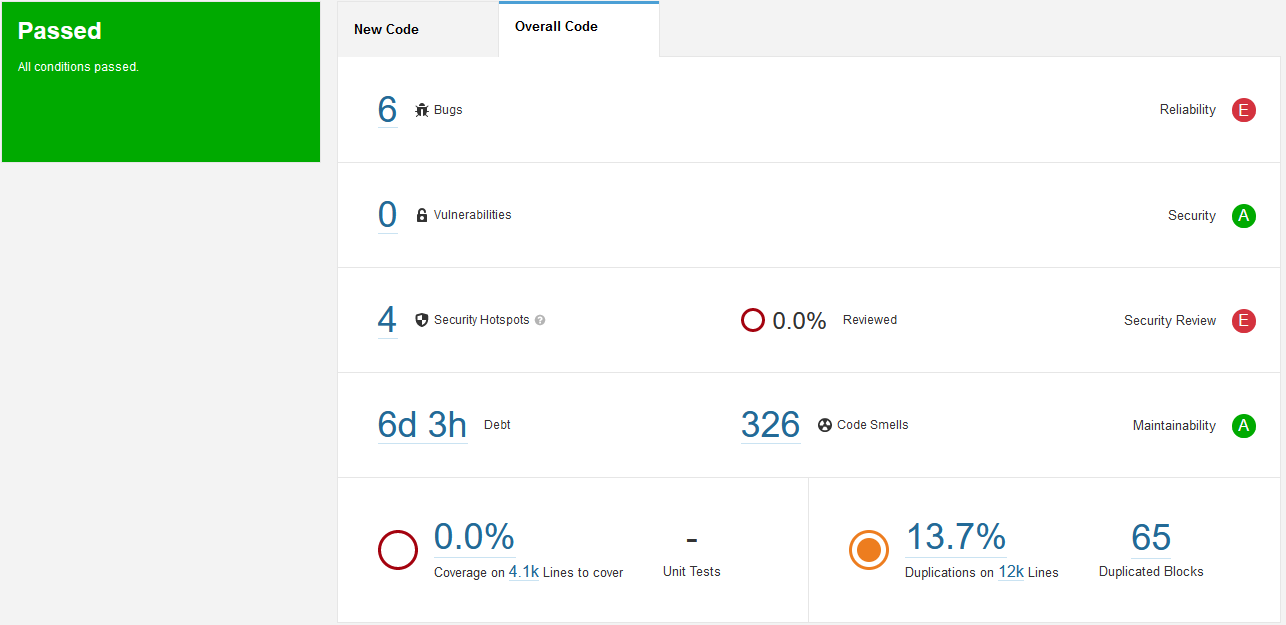
* L’API a été développé en python avec le framework de développement web, Pyramid
* Le site web et l’application sont en javascript :
  + Le site web, il est développé avec la librairie reactJS et pour la partie graphique, matérial ui
  + L’application mobile est développée avec le framework ReactNative

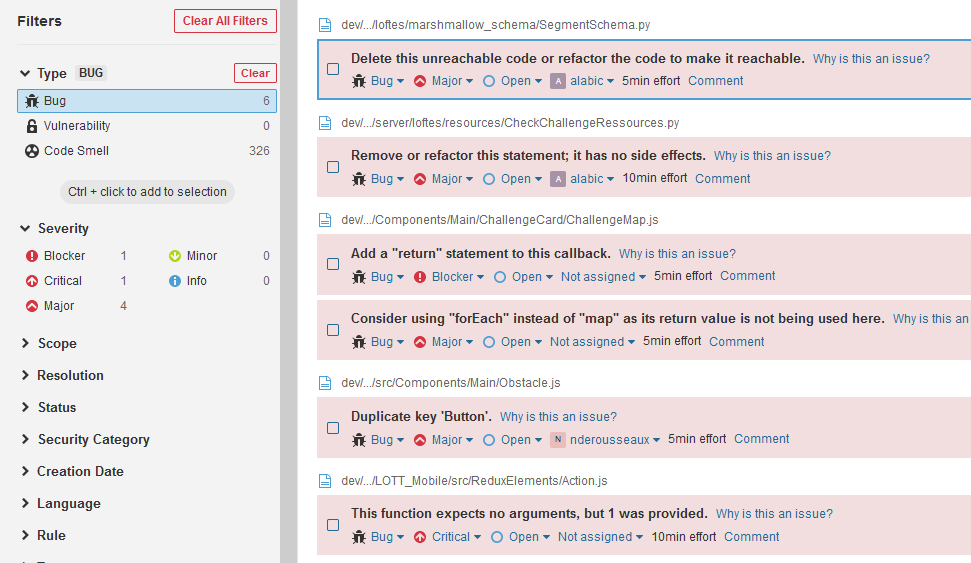
# Analyse SonarQube du projet

Nous avons fait une première analyse, qui a mis en évidence sur l’ensemble de l’application

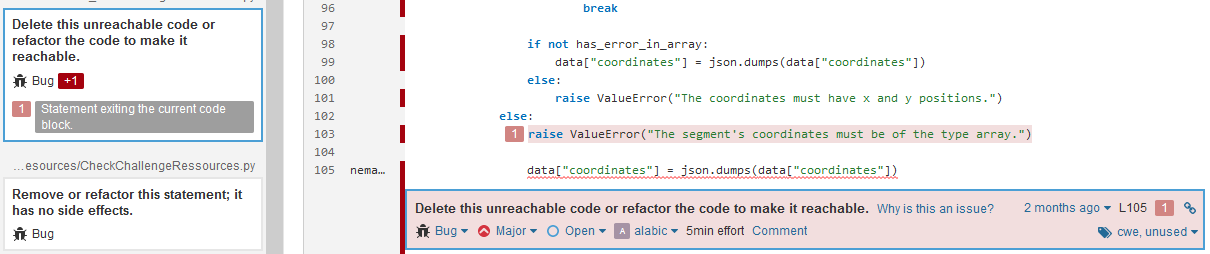


* 6 bugs
* 4 Sécurity HotSpots
* 326 code smell
* 0% coverage
* 13,7 % de duplication

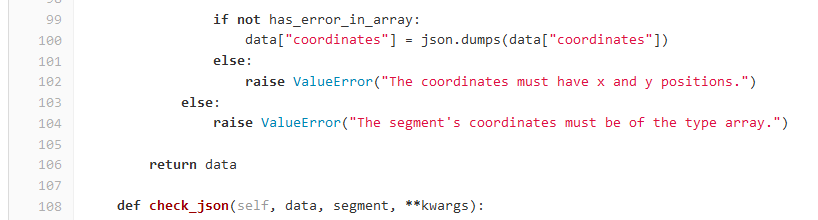


Nous avons commencé par corrigé les 6 bugs

Selon les cas les correctifs appliqués ont été différents, par exemple des correctifs sont appliqués sur le code en erreur



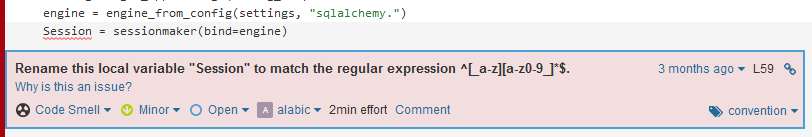
Correction



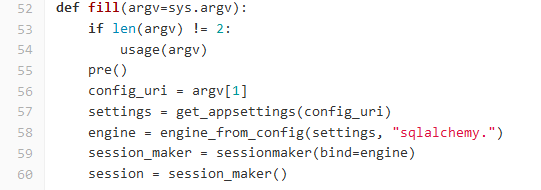
Mais dans certains, nous avons aussi refactoriser le code de façon plus globale, à la fois pour gérer les bugs mais aussi pour corriger des code smells ou des duplications

Les codes smells nous ont aussi permis de mettre en place des optimisations de codes en éditifiant

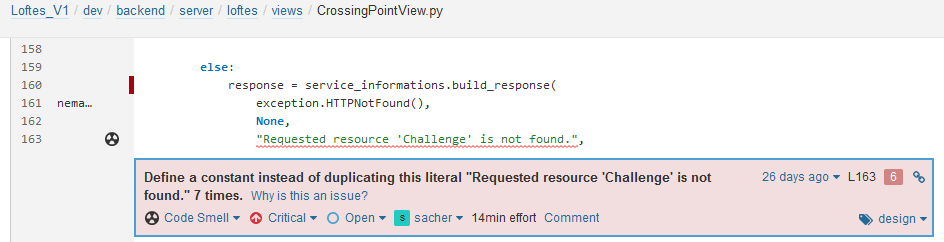
Des variables ne respectant pas les normes de nommage



Correction

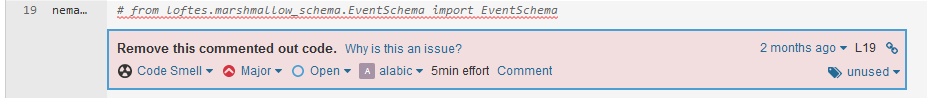


Des textes en dur dans le code qui se répété et pouvait être mutualisé



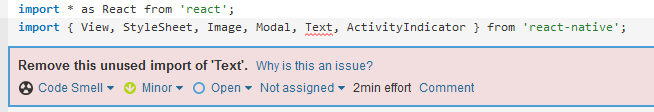
* Quand cela est possible création d’un fichier de constante avec le contenu des messages, plus facile à maintenir

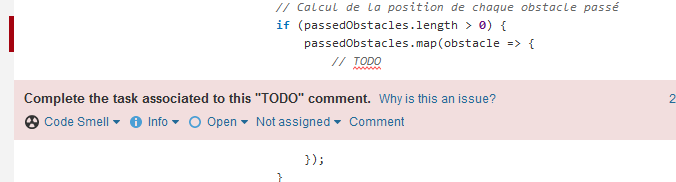
Du codes en commentaires qui surchargés inutilement le code

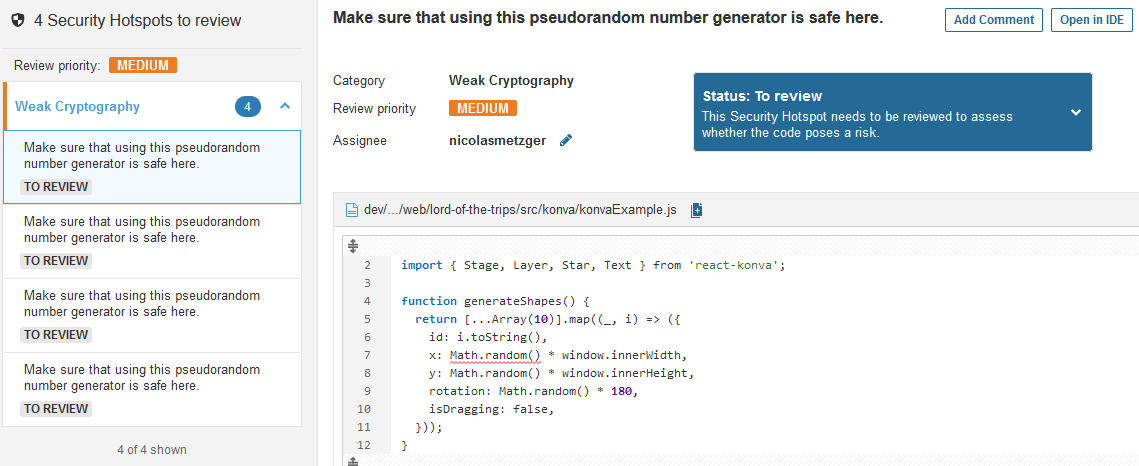


* Suppression du code commenté

Des imports inutiles de données





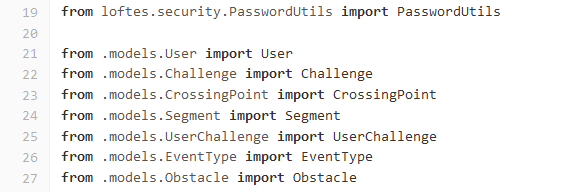


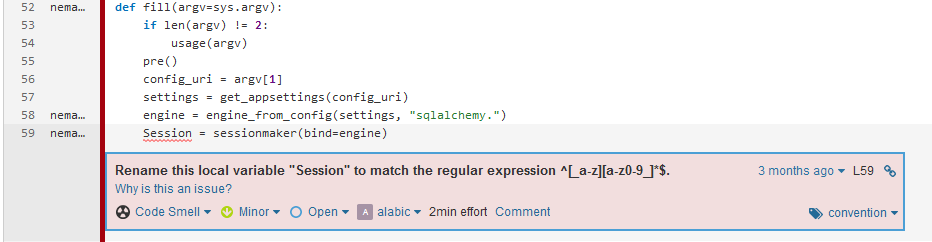
+> le code a été supprimé car il s’agit d’exemple pour le développement

Code smell

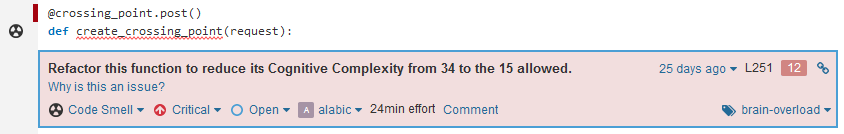


* Specifier import





* Renommage de la variable

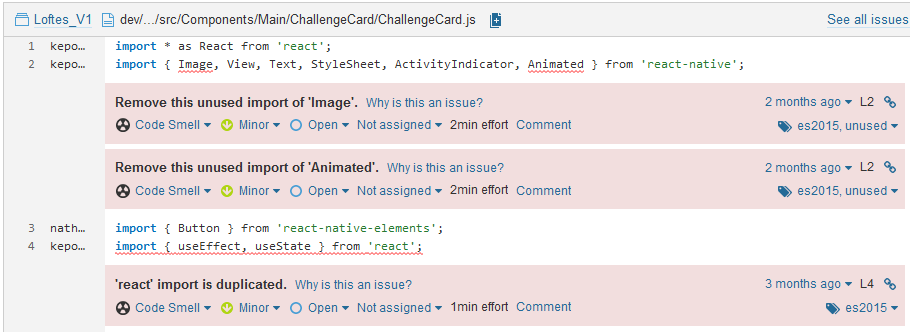


* Pas de correction apportée car il n’est pas toujours facile de refactoriser certains fonctions

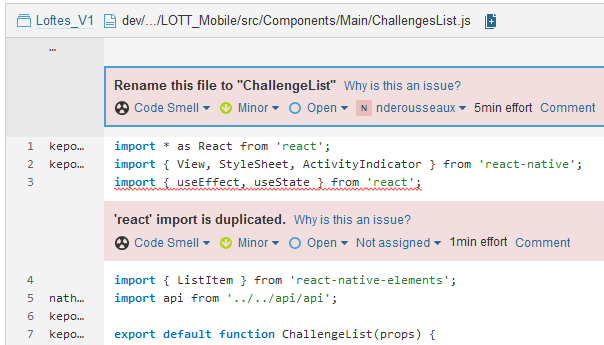


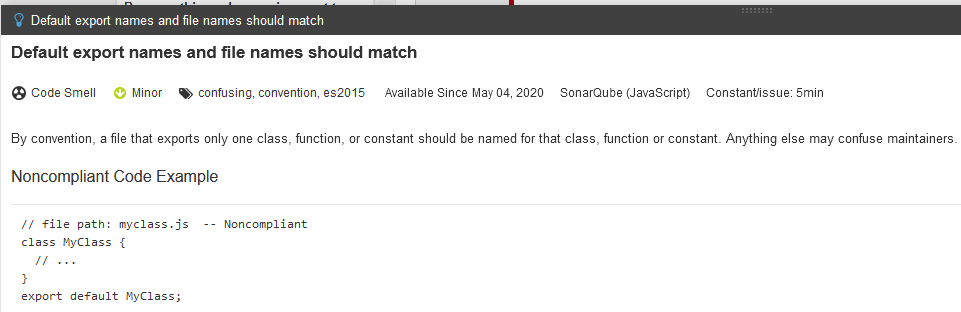
Js





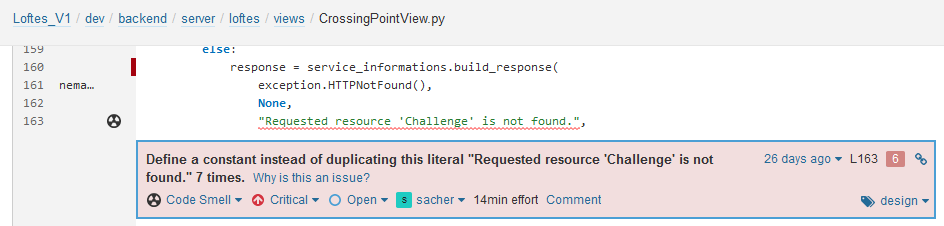
Suppression des variables importés inutilement



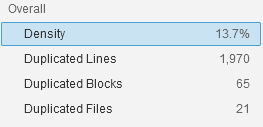


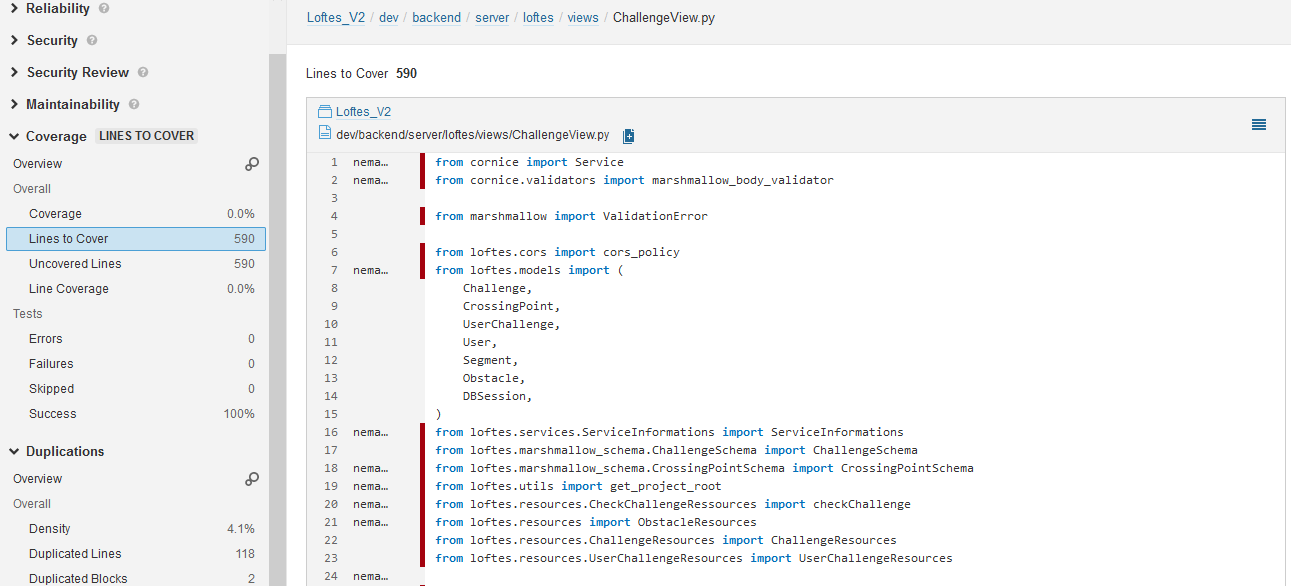


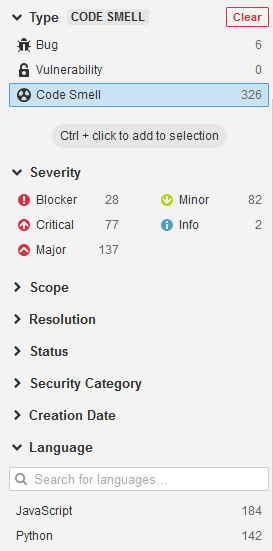
Duplicated line



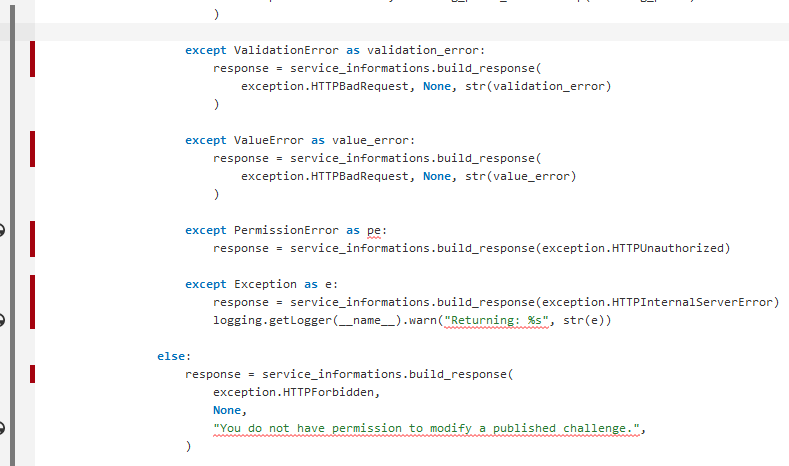
* Création de message



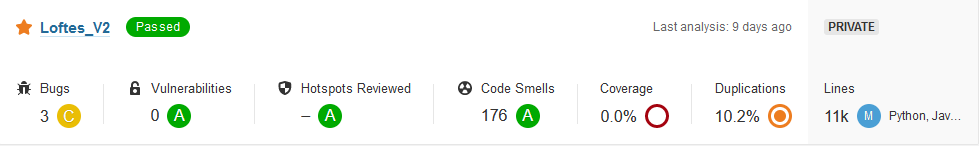


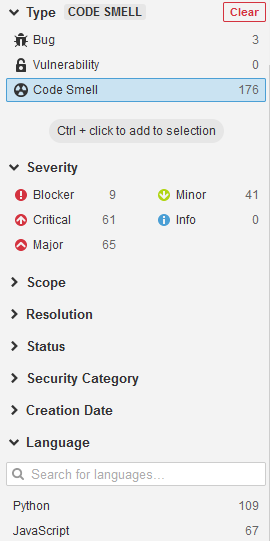


Code dupliqué

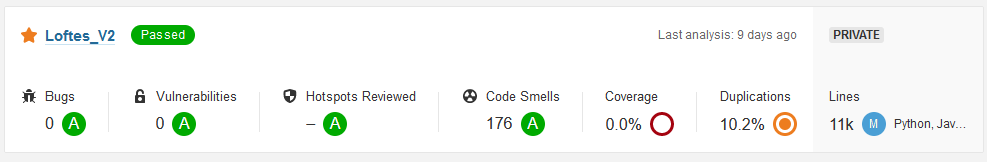


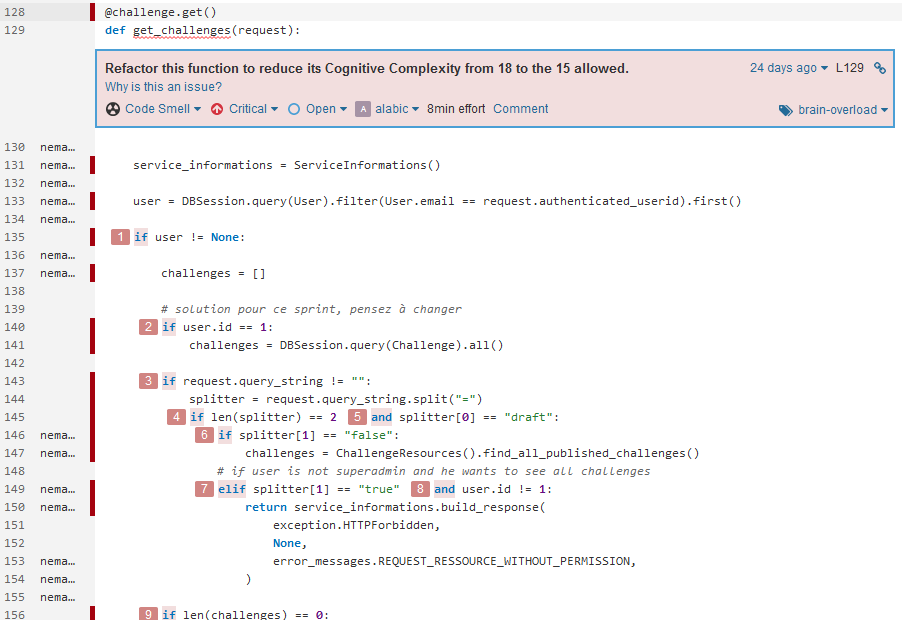
2nde analyse

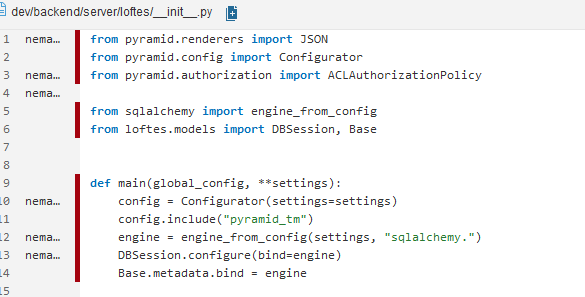


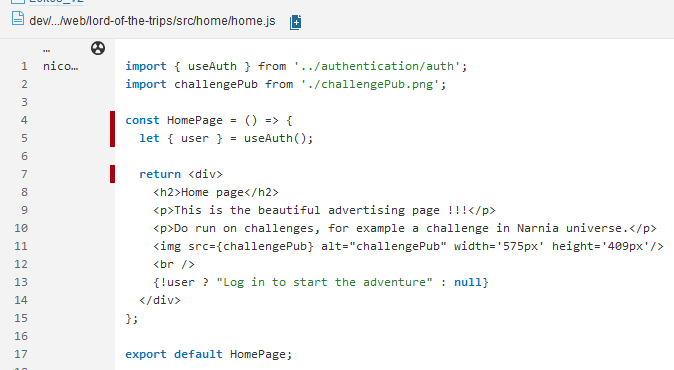




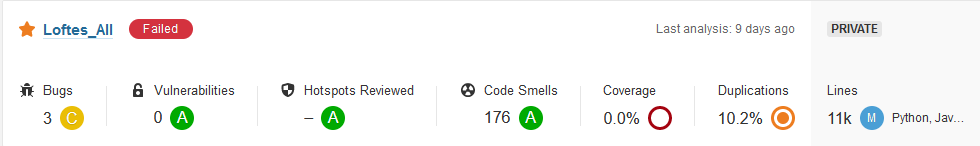


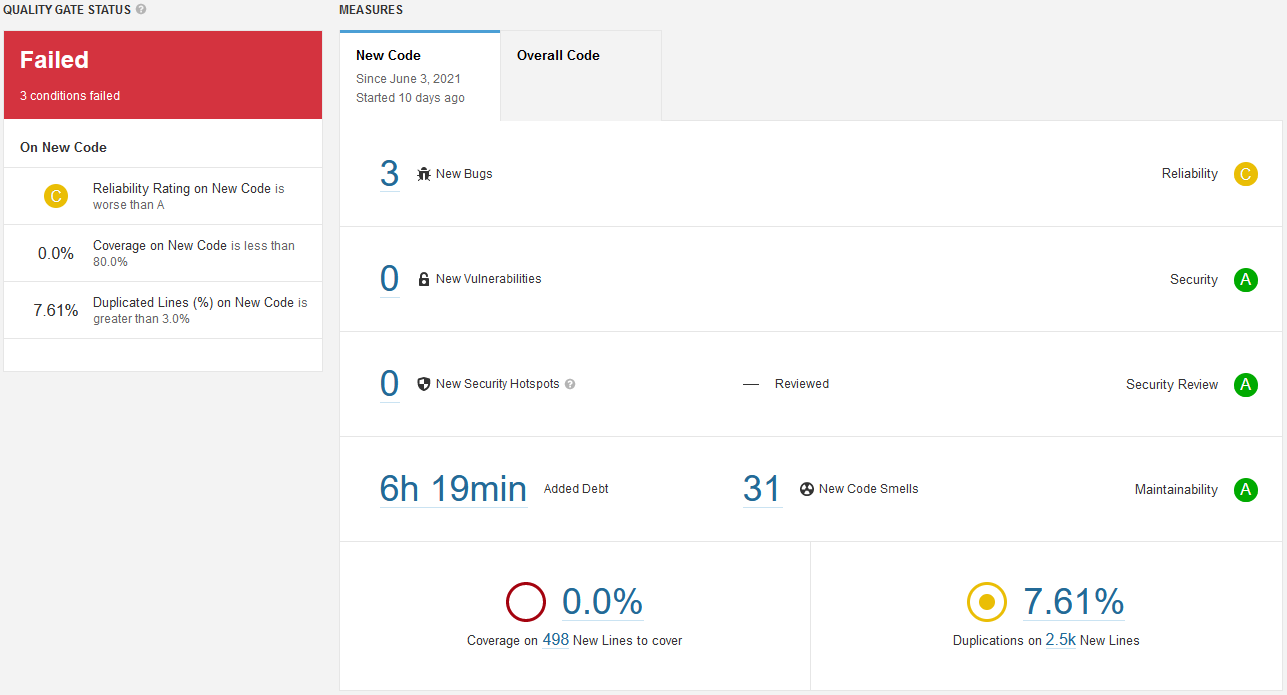


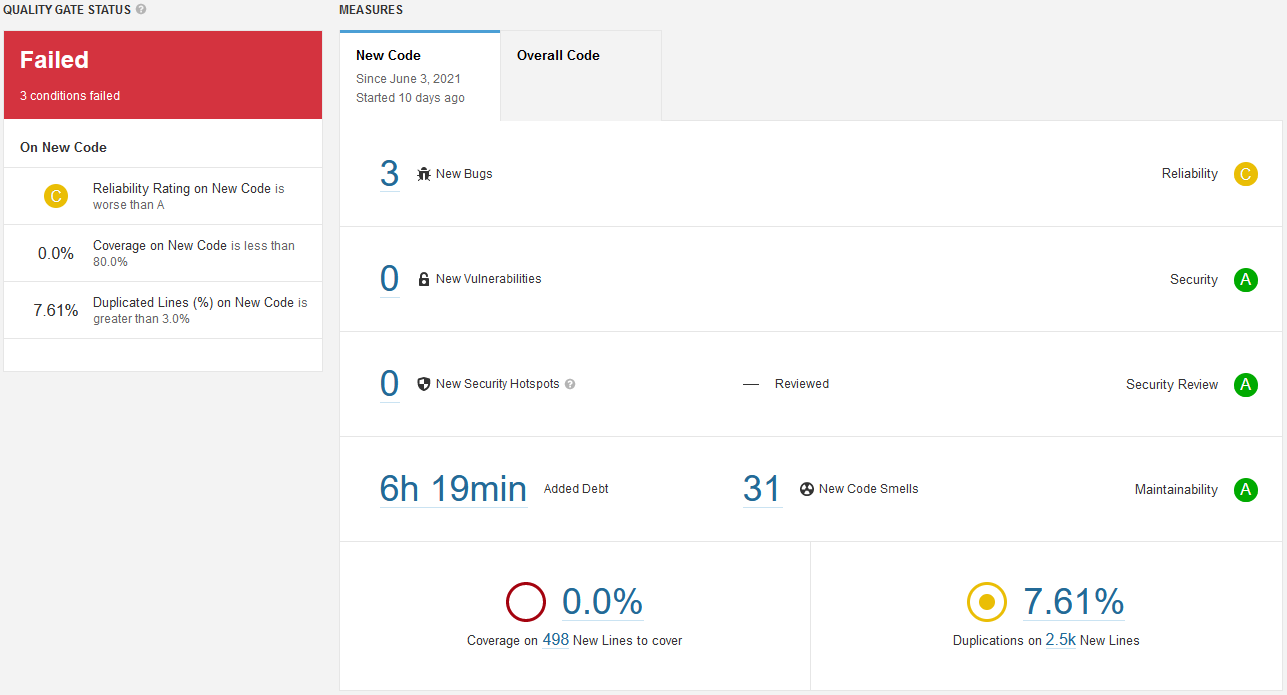


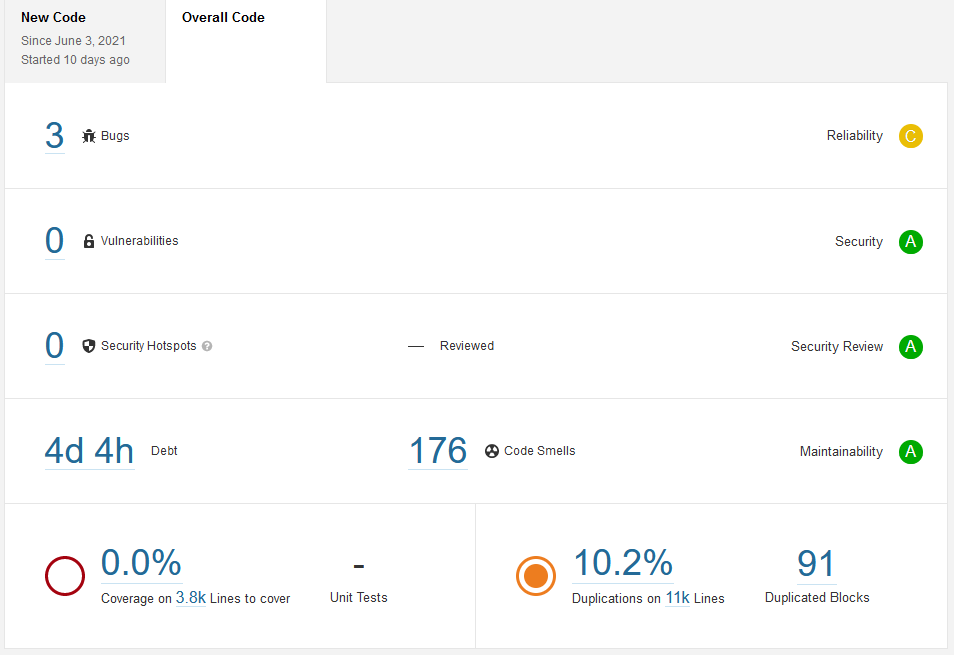


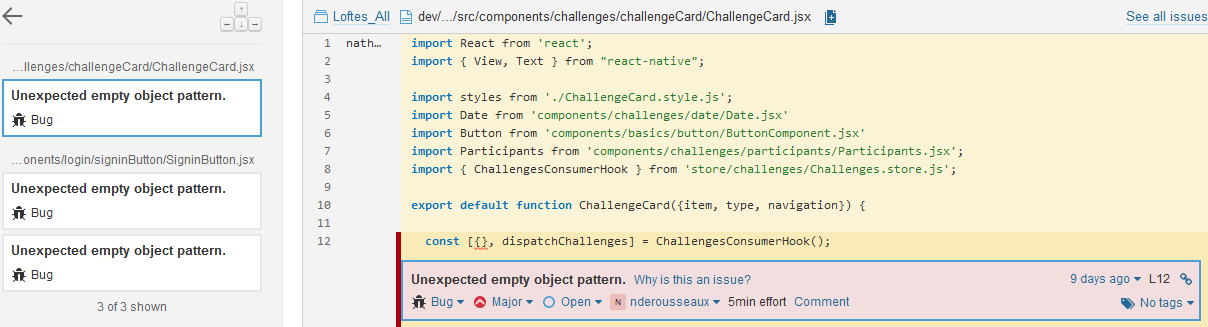
Evolution

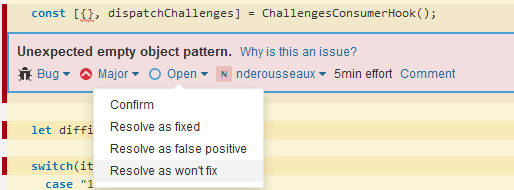


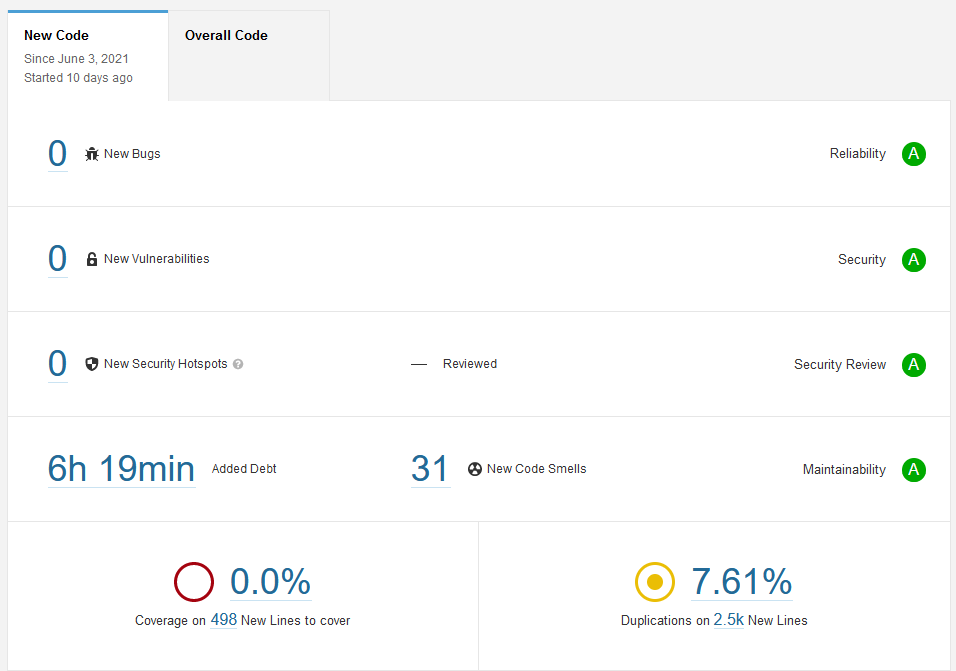












# En conclusion

Bug intéressant

Mutualisation de code => OK

Mais coverage pas forcement utile et aussi import µ parfois nécessiare

# Contexte

*Suite à une étude de santé publique sur les étudiants en cursus informatique, le ministère de la santé nous a demandé de développer une application permettant la réalisation d'un parcours virtuel en pratiquant une activité sportive dans le réel.*

*L’application doit permettre de*

* *Créer des parcours et de gérer des parcours, ainsi que les événements se déroulant sur ceux-ci.*
* *Jouer sur ces mêmes parcours, en utilisant trois mode de déplacements : marche, course et vélo, et aussi de suivre la progression sur les parcours*

*Cette application sera déclinée sur 2 médias :*

* *Un site web qui servira pour la gestion des parcours par les administrateurs et pour les utilisateurs, à afficher les détails de leur progression.*
* *Une application mobile qui servira essentiellement aux joueurs pour avancer sur leurs parcours, mais aussi de s’inscrire à d’autres challenges*

*Le but du présent document est de présenter certaines étapes de la conception qui nous ont permis d’aboutir à une application fonctionnelle répondant à la demande du client.*

*Composition de l’équipe :*

* *Alabic Nemanja*
* *Derousseaux-Lebert Nathanael*
* *Metzger Nicolas*
* *Sacher Marilyn*
* *Steiner Kélio*

Contexte

Travail demandé

Vous êtes invités à constituer un dossier synthétique (3 - 4 pages) portant sur la qualité du code de votre projet afin d'établir les indicateurs du tableau de bord Sonarqube (duplication, "code smell", vulnérabilités, "dette technique", "complexité"...), suivant les pluggins de langages disponibles.

1) présentez brièvement votre projet (architecture technique, framewoks, langages ...)

2) mettez en oeuvre l'outil Sonarqube mis à disposition sur les postes de l'IUT : <https://si.iutrs.unistra.fr/2020/05/13/sonarqube-presentation-de-loutil/>

3) présentez et commentez les résultats obtenus en montrant leur évolution suivant les modifications de versions effectuées à partir des recommandations de l'outil Sonarqube

4) concluez sur votre perception des apports et des limites de l'outil pour votre projet (périmètre couvert - pluggins, facilité d'utilisation, recommandations ...), puis en élargissant votre point de vue des avantages et inconvénients pour une utilisation en environnement professionnel Organisation –

Modalités pratiques de livraison Travail à réaliser par équipe projet, un dossier par équipe, deux séances consacrées à cette étude Vous déposerez votre rendu en PDF sur Moodle pour le Vendredi 18 juin 2021 au plus tard, une section ayant été mise en place sur l’espace de travail à cet effet (un seul dépôt par équipe).

Les metric sonarqube

## **Issues lifecycle**

### **Statuses**

After creation, issues flow through a lifecycle, taking one of five possible statuses:

* **Open** - set by SonarQube on new issues
* **Confirmed** - set manually to indicate that the issue is valid
* **Resolved** - set manually to indicate that the next analysis should Close the issue
* **Reopened** - set automatically by SonarQube when a Resolved issue hasn't actually been corrected
* **Closed** - set automatically by SonarQube for automatically created issues.

### **Resolutions**

Closed issues will have one of two resolutions:

* **Fixed** - set automatically when a subsequent analysis shows that the issue has been corrected or the file is no longer available (removed from the project, excluded or renamed)
* **Removed** - set automatically when the related rule is no longer available. The rule may not be available either because it has been removed from the Quality Profile or because the underlying plugin has been uninstalled.

Resolved issues will have one of two resolutions:

* **False Positive** - set manually
* **Won't Fix** - set manually

### **Issue Workflow**

Issues are automatically closed (status: Closed) when:

* an issue (of any status) has been properly fixed => Resolution: Fixed
* an issue no longer exists because the related coding rule has been deactived or is no longer available (ie: plugin has been removed) => Resolution: Removed

Issues are automatically reopened (status: Reopened) when:

* an issue that was manually Resolved as Fixed(but Resolution is not False positive) is shown by a subsequent analysis to still exist

## **Understanding which Issues are "New"**

To determine the creation date of an issue, an algorithm is executed during each analysis to determine whether an issue is new or existed previously. This algorithm relies on content hashes (excluding whitespace) for the line the issue is reported on. For multi-line issues, the hash of the first line is used. For each file (after detection of file renaming), the algorithm takes the base list of issues from the previous analysis, and tries to match those issues with the raw issue list reported by the new analysis. The algorithm tries to first match using the strongest evidence, and then falls back to weaker heuristics.

* if the issue is on the same rule, with the same line number and with the same line hash (but not necessarily with the same message) > MATCH
* detect block move inside file, then if the issue is on the same (moved) line and on the same rule (but not necessarily with the same message) > MATCH
* on the same rule, with the same message and with the same line hash (but not necessarily with the same line) > MATCH
* on the same rule, with the same message and with the same line number (but not necessarily with the same line hash) > MATCH
* on the same rule and with the same line hash (but not the same message and not the same line) > MATCH
* is there a matching **CLOSED** issue > MATCH and Reopen

Unmatched "base" issues are closed as fixed.

Unmatched "raw" issues are new.

## Arborescences

### Mobile

### Web

https://miro.com/app/board/o9J\_lOS-cTw=/

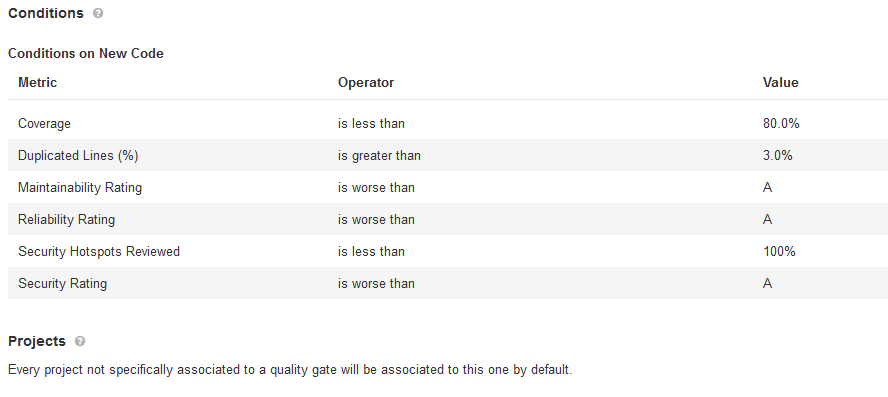
## Définition des métriques

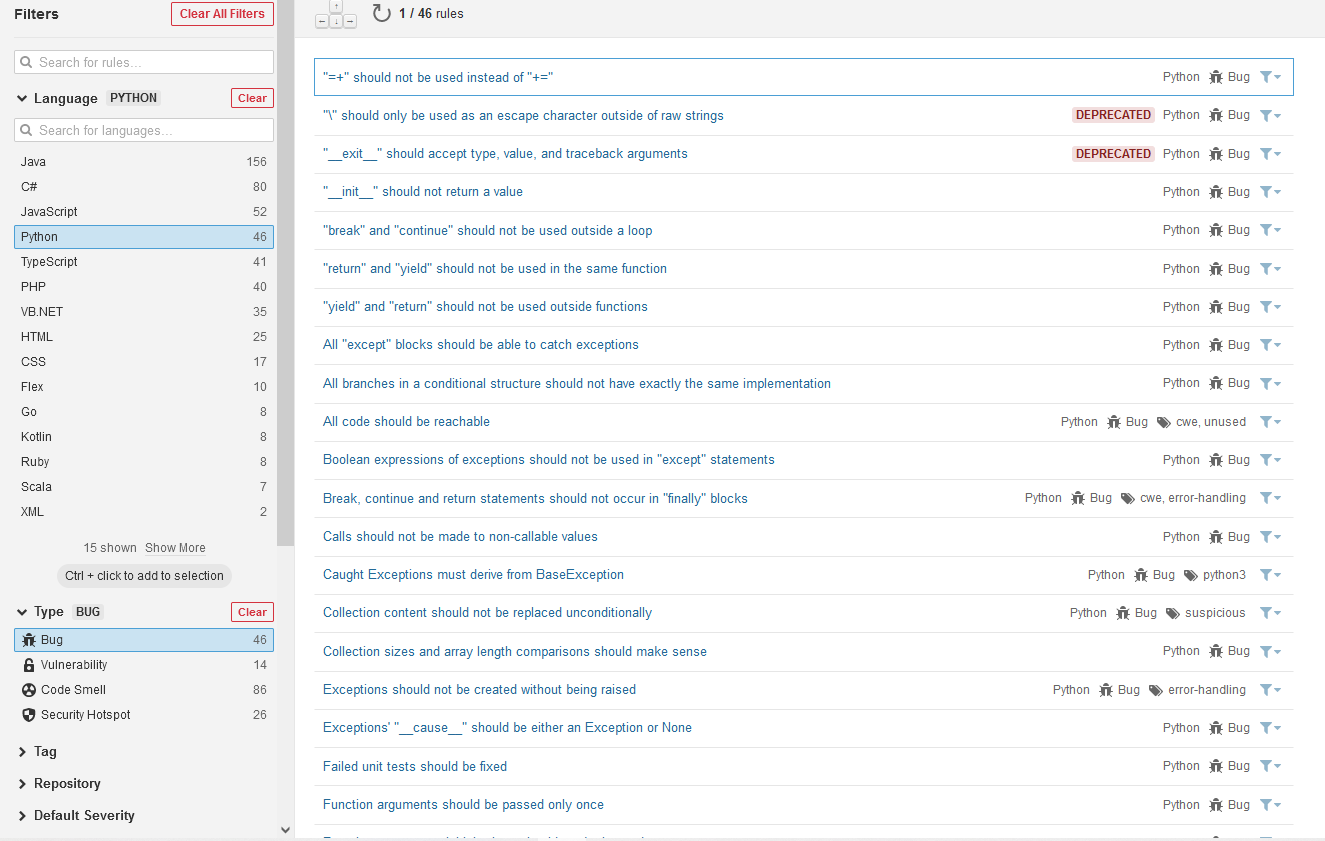
https://docs.sonarqube.org/latest/user-guide/metric-definitions/

## Annexes

Voir ces pages pour du blabla

Quality gates du sonarQubes de l’école





https://sonar.iutrs.unistra.fr/coding\_rules?languages=py&types=BUG