ANALYSE SONARQUBE
EQUIPE 1

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 solution permettant la réalisation d'un parcours virtuel via la pratique d'une activité sportive en réel. Elle doit permettre de créer des parcours virtuels via un site web et de se déplacer via une application mobile. Durant la mise en place de l'application, il nous a paru judicieux de déployer un outil permettant d'analyser la qualité du code afin de le rendre plus robuste, moins de redondant et facilement maintenable.

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 permet d'avoir une évolution dans le temps et des vues différentielles. Il supporte plus de 25 langages et génère des reportings 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

Comprendre une analyse SonarQube

L'analyse SonarQube se base sur une série de règles de codage et d'indicateurs défini par défaut dans l'application pour les langages pris en charge. Ces réglés et indicateurs peuvent être complétés ou modifiés par les utilisateurs.

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

ANALYSE SONARQUBE
EQUIPE 1

Par ailleurs, à chaque analyse, SonarQube compare à la précédente analyse. Il affiche une analyse globale et une sur les nouveaux problèmes rencontrés. 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 **Security Hotspots** (une faille de sécurité devant être vérifier par le développeur doit vérifier avant d'appliquer ou non un correctif).
- Complexité : simple, à savoir le nombre de subdivisions du code (ex if else,switch, …..) et cognitive, à savoir la difficulté à suivre le « flux » du code, c-à-d le lire et le comprendre.
- Duplication : nombre de lignes dupliquées, de fichiers contenant des duplications, ...
- Maintenabilité : temps 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 (Valeurs possibles ERROR et OK). Le "**Quality Gate**" : liste de conditions appliqués sur des mesures que l'analyse du code doit respecter, pour exemple

Conditions

Conditions on New Code

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

Organisation du projet et technologies

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

- Une API qui gérera l'accès et les requêtes à la base de données
- 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 un répertoire server où se trouve l'ensemble des répertoires, librairies et fichiers nécessaires
- Un répertoire frontend avec deux sous répertoires :
    - Web avec l'ensemble des répertoires, librairies et fichiers nécessaires au site web
    - Mobile avec l'ensemble des répertoires, librairies et fichiers nécessaires à l'application mobile

En ce qui concerne les technologies employées :

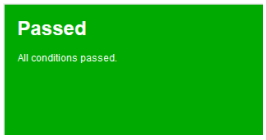
- L'API a été développé en python avec le framework de développement web, Pyramid
- Le site web et l'application sont en javascript :
    - Le site web, il est développé avec la librairie reactJS et pour la partie graphique, matérial ui
    - L'application mobile est développée avec le framework ReactNative

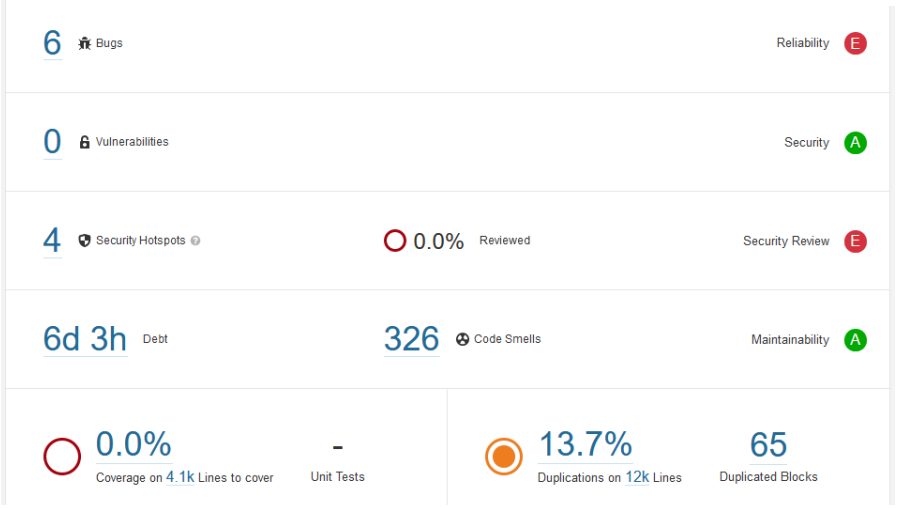
ANALYSE SONARQUBE
EQUIPE 1

Analyse SonarQube du projet

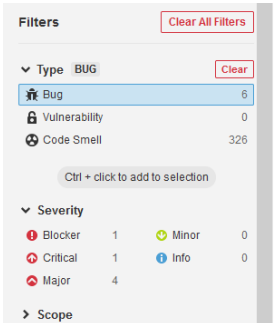
Nous avons fait une première analyse, qui a mis en évidence sur l'ensemble de l'application

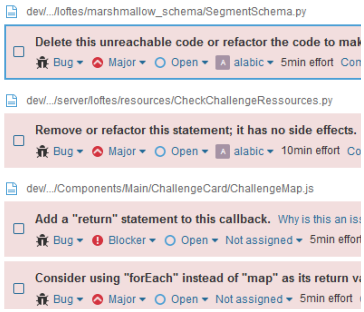
- 6 bugs
- 4 Sécurity HotSpots
- 326 code smell
- 0% coverage
- 13,7 % de duplication





Nous avons commencé par corriger les 6 bugs en appliquant les correctifs nécessaires :





Pour exemple :

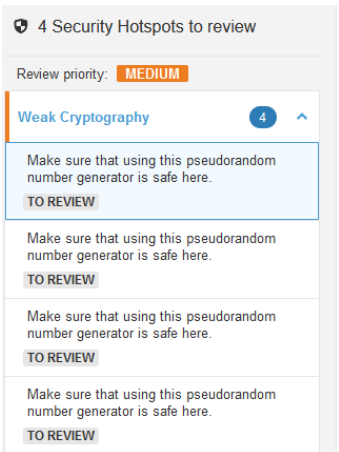
```
            else:
1               raise ValueError("The segment's coordinates must be of the type array.")

            data["coordinates"] = json.dumps(data["coordinates"])
```

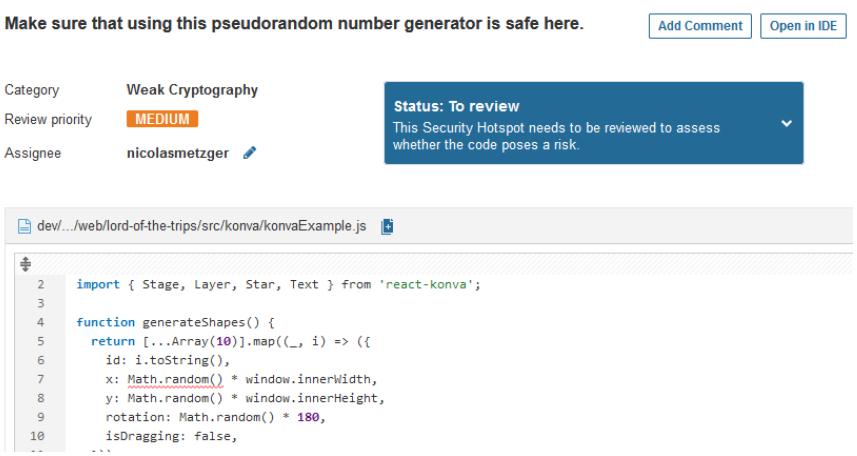
Delete this unreachable code or refactor the code to make it reachable. Why is this an issue? 2 mo
Bug Major Open alabic 5min effort Comment

Correction

```
103             else:
104                 raise ValueError("The segment's coordinates must be of the type arra
105
106         return data
107
108     def check_json(self, data, segment, **kwargs):
```

Nous avons ensuite examiné les « Sécurity Hotspots », mais il s'agissait de code de tests amené à disparaître

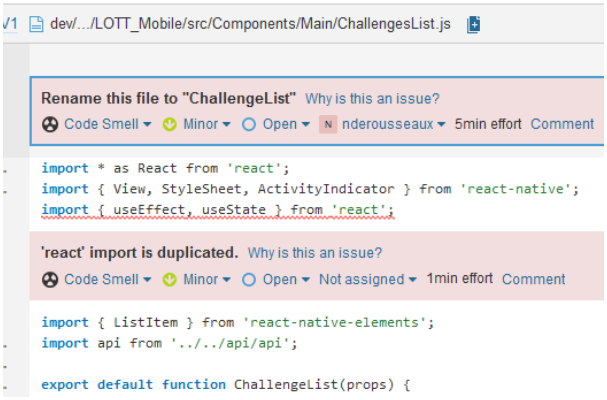




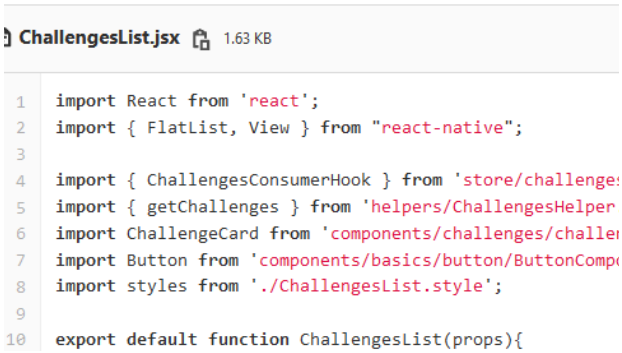
ANALYSE SONARQUBE
EQUIPE 1

Puis, les codes smells nous ont permis de mettre en place des optimisations de codes en identifiant :

- Des variables ou des noms de fonctions ne respectant pas les normes de nommage

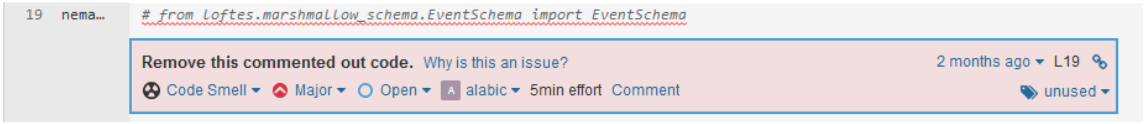


Correction

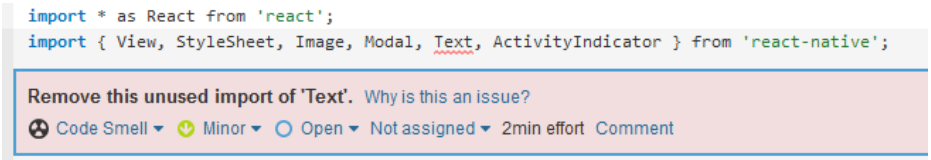


- Des textes en dur répétés dans le code qui pouvait être mutualisé, et pour lequel nous avons mis en place un fichier de constantes avec le contenu des messages

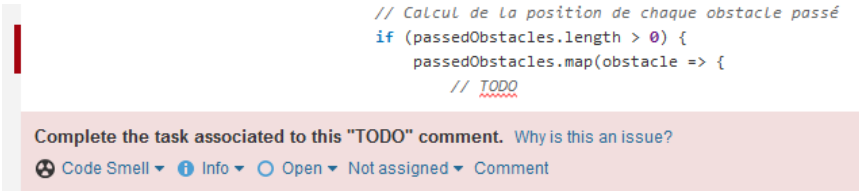
- Du code en commentaires qui surchargés inutilement les fonctions que nous avons supprimé



- Des imports inutiles de données ou des variables inutilisées qui ont été supprimés



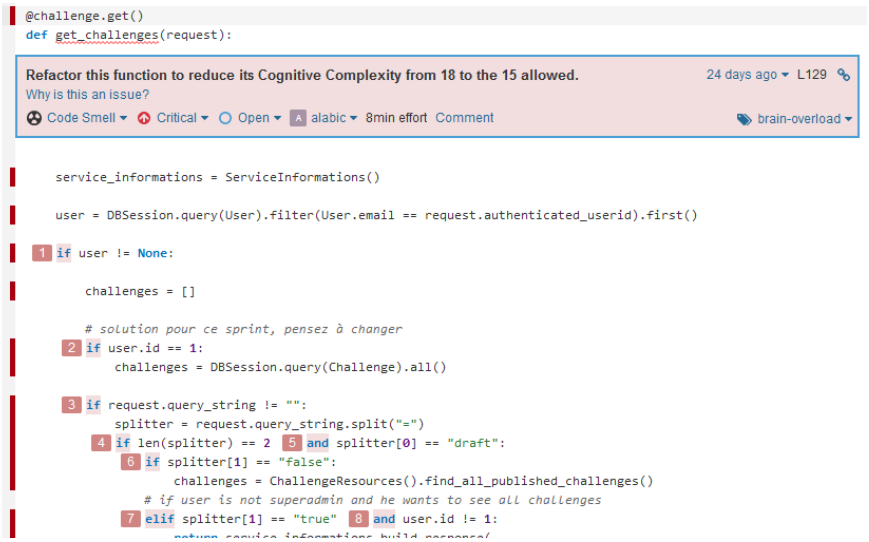
- Des portions de code non encore écrite via la détection du texte TODO/todo



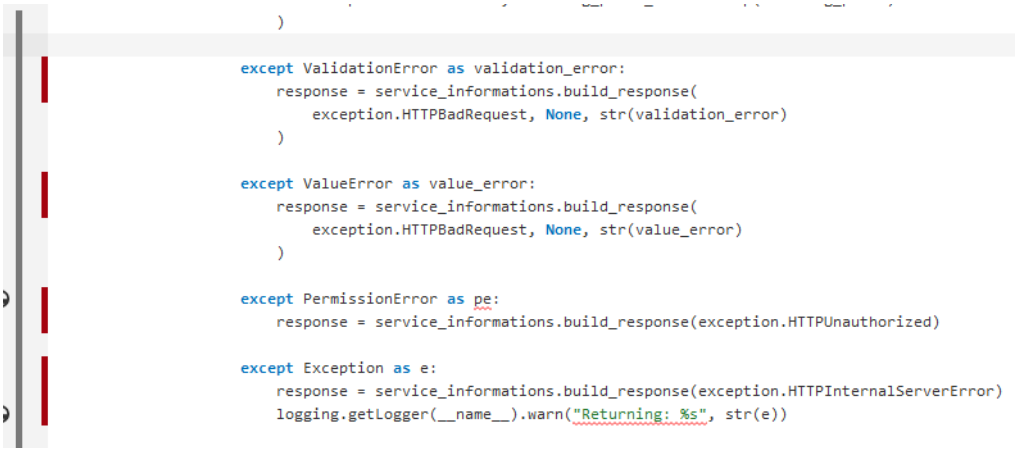
Nous avons aussi été confronté à des problèmes dont la résolution n'était pas toujours aisé, voir pour lesquelles nous avons préféré ne pas intervenir car les modifications auraient pu entraîner des erreurs dans le fonctionnement de l'application en production, c'est le cas notamment :

- Des problème de complexité cognitive, dans ce cas les modifications peuvent générer des effets de bord non désiré ou provoquer un dysfonctionnement de la fonction impacté, et parfois le code ne parait pas forcement complexe

ANALYSE SONARQUBE
EQUIPE 1



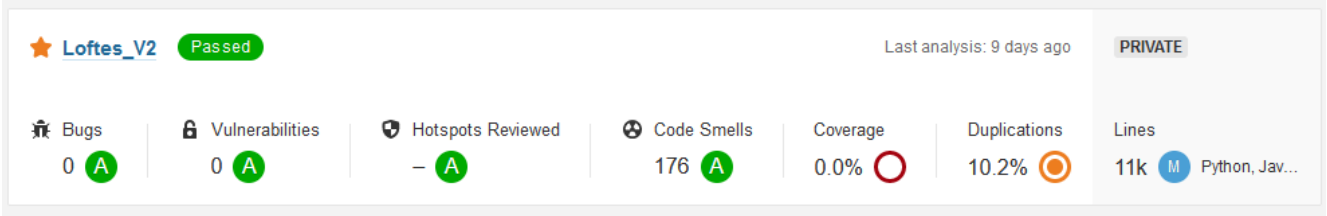
- get_obstacle_for_validationDes blocks dupliqués qu'il est complique d'extraites pour mutualisation



Tout en gérant les différents problèmes rencontrés, nous avons plus globalement modifier et refactorisé le code de façon augmenter sa maintenabilité et sa clarté.

Suite au modifications effectués, nous avons relancé une 2nde analyse qui a montré une amélioration sensible par rapport la précédente :

- Plus de bugs et de Hotspots
- Environ 46% de code smell en moins
- 3 points de moins en lignes dupliqué, et une réduction du nombre de lignes globales de l'application
- Et presque 2 jours (16h) de dettes techniques en moins



Ancienne Dette technique	Nouvelle Dette technique
6d 3h Debt	4d 4h Debt

ANALYSE SONARQUBE
EQUIPE 1

En conclusion

L'outil sonarQube a été très intéressant à utiliser. Pour nous, il nous a permis de découvrir des bugs qu'une simple lecture du code n'aurait pas suffi à détecter. Mais le plus grand avantage de cet outil, c'est qu'il permet d'imposer une régularité dans la syntaxe du code. En effet, dans un travail collaboratif, tout le monde n'applique pas les mêmes normes de nommage, et cela peut rendre difficile la relecture du code, et ainsi réduire sa maintenabilité. Avec sonarQube, le code de notre projet a pu avoir une véritable cohérence.

Précisément, chez nous, il nous a permis de nous alerter sur certains morceaux de code dupliqués dans notre projet.

Cependant, nous avons pu voir certaines limites à l'outil. En effet, l'analyse nous a fait remonter beaucoup de faux positifs, par exemple certaines variables qui étaient initialisées à 'null' étaient considérées comme problématique. Tous ces faux positifs auraient pu être réglés par un paramétrage en amont de sonarQube, afin de définir les normes de codage que l'on aurait décidé d'appliquer à notre code. Mais cela aurait nécessité un vrai travail, et nous n'avions pas le temps. Nous pensons qu'il est là le vrai défaut de sonarQube : en entreprise, cela peut nécessiter un énorme investissement en temps de travail pour poser toutes les règles de code. Ensuite, si l'outil intègre une application longtemps après le début de son développement, il est probable que l'on se retrouve avec une énorme dette technique.

A notre avis, et pour conclure, SonarQube est un outil très utile et puissant, qui permet de faciliter le travail collaboratif sur un projet et d'améliorer la maintenabilité de son code. Il a aussi comme avantage de pouvoir repérer des petits bugs. Il se doit d'être utilisé dans tous les projets ambitieux. Cependant, l'outil demande un véritable travail avant de pouvoir le faire fonctionner correctement, et celui-ci n'a peut-être pas sa place dans des projets plus petit, ou à la durée de vie plus courte.