République Algérienne Démocratique et Populaire Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

2ème année Cycle Supérieur (2CS)

Option : Systèmes Informatiques et Logiciels (**SIL1**)

Rapport d'analyse avec SonarQube

Réalisé par :

- **DJERFI Fatma**
- **MELLITI Abdelmalek**

Encadré par Mr. Mokaddem *Hakim*

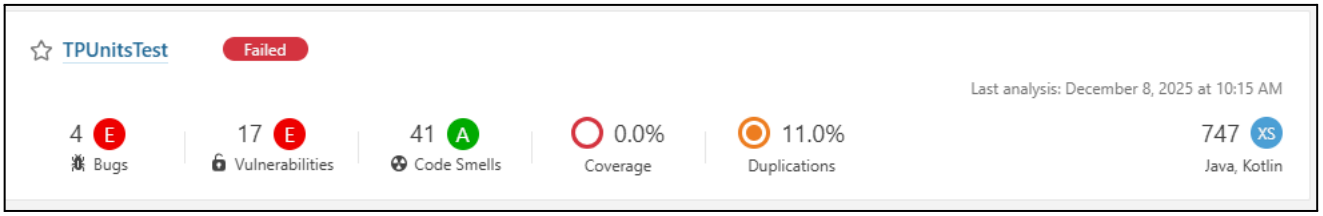
Promotion 2025-2026

I. Introduction :

L'analyse du code source de notre programme a été réalisée à l'aide de **SonarQube**, dans le cadre de notre démarche d'**amélioration continue de la qualité logicielle**. Cette étude nous a permis d'évaluer l'état initial du code, avant toute optimisation ou correction.

Notre programme, développé en **Java**, présente les informations générales suivantes dans sa version non améliorée :

- **4 bugs** identifiés, susceptibles d'affecter le comportement ou la stabilité de l'application.
- **17 vulnérabilités**, représentant des failles potentielles pouvant compromettre la sécurité du logiciel.
- **41 code smells**, c'est-à-dire des fragments de code pouvant nuire à la lisibilité, à la maintenabilité ou à la performance.
- Un **taux de duplication de 11%**, indiquant la présence de portions de code répétitives qui pourraient être refactorisées pour améliorer la qualité globale.



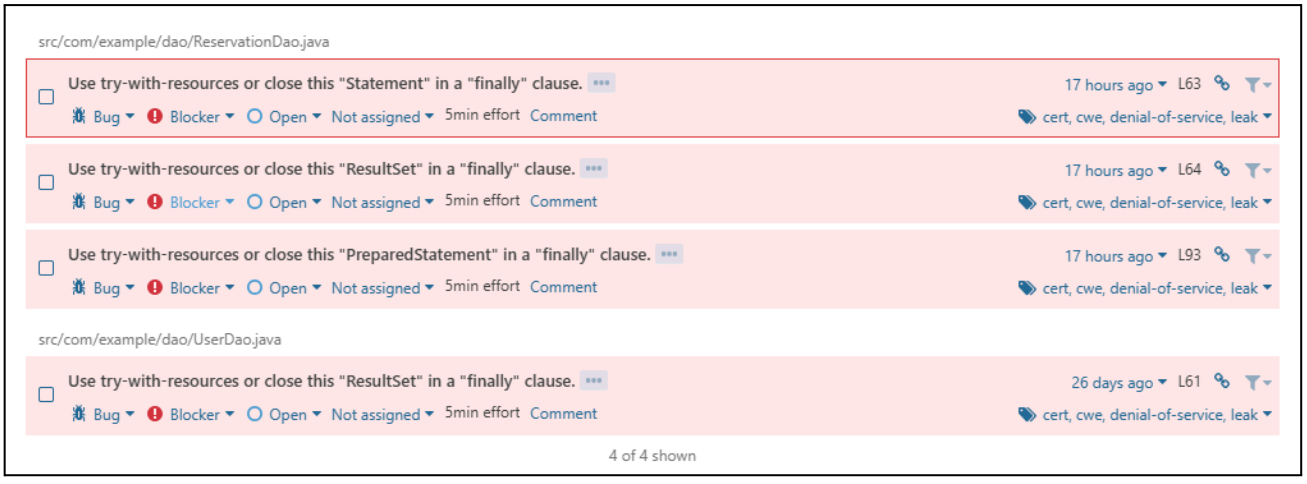
Cette analyse constitue un point de départ pour **prioriser les corrections et les améliorations**, et pour assurer que le logiciel respecte les bonnes pratiques de développement.

II. Les bugs et les problèmes de sécurité :

a. <u>Les bugs :</u>

Les bugs trouvés en utilisant SonarQube :

On remarque que ces 4 bugs sont de la même nature et renvoient presque le même message, la solution de chacun est donc la suivante:



- <u>**Solution suggérée pour résoudre les bugs :**</u>

  Il faut mettre la déclaration des variables qui posent problème, en l'occurrence **"pstmt"** et **"rs"** respectivement associées aux classes **"PreparedStatement"** et **"ResultSet"**, dans la

clause try-with-ressources et elle s'occupera de fermer les connexions (au lieu d'appeler la méthode **close( )** de l'objet). Pour la variable **"stmt"** qui était déclarée dans la méthode **getReservationById(int idReservation)** dans la classe **ReservationDao**, nous l'avons remplacée par **pstmt** (PreparedStatement) car les requêtes préparées évitent les injections **SQL**, au lieu de mettre directement l'id de la réservation dans la requête. (Cela a été traité dans la partie de résolution des problèmes de sécurité).

b. Les problèmes de sécurité :

L'analyse en utilisant SonarQube nous a permis d'extraire **17 problèmes de sécurité**, qui se résument en ces **trois** types:

1. Credentials should not be hard-coded :

- On déclare une variable d'environnement **"DB_PASSWORD"** avec la valeur du mot de passe ici **"root"** à l'aide de la commande à saisir dans le terminal : **"setx DB_PASSWORD root"**.
- Pour récupérer le password dans la classe Home, il faut le déclarer comme attribut de cette façon : **public static final String PASSWORD = System.getenv("DB_PASSWORD")**;

2. SQL binding mechanisms should be used :

- C'est un problème de type **Critical** qui peut être résolu de la manière suivante:
  1. Préparation de la requête.
  2. Paramétrage de la valeur (**idReservation**).
  3. Exécution de la requête.
- Cette approche contribue à la sécurité et à la performance de l'accès à la base de données en utilisant des requêtes paramétrées et des objets PreparedStatement pour éviter les attaques par injection SQL.

3. Throwable.printStackTrace(...) should not be called:

- C'est un problème de type Minor qui peut être résolu en ajoutant des Loggers avec un message clair décrivant l'erreur. Pour ce faire, nous avons utilisé la classe "Logger" du package "java.util.logging" en le déclarant comme suit sous forme d'un attribut dans les classes correspondantes : **"private static final Logger LOGGER = Logger.getLogger(Nom_De_La_Classe.class.getName());"**. Pour l'utiliser dans le bloc **catch** : **"LOGGER.log(Level.SEVERE, message, variable_d'erreur);"**.
- Ce problème affecte la sécurité du code car la fonction **printStackTrace( )** peut divulguer des informations sensibles comme : les chemins d'accès aux fichiers, le nom des packages et

classes,...etc. Grâce aux loggers, on peut personnaliser les messages d'erreurs et définir leur niveau de criticisme.

III. Les faux positifs identifiés par SonarQube :

Dans le cadre de l'analyse de notre code, **aucun** faux positif n'a été détecté. Les résultats obtenus par SonarQube reflètent précisément **les problèmes existants** dans le code source, confirmant ainsi la fiabilité de l'outil dans notre contexte spécifique.

IV. Les problèmes du code non identifiés par SonarQube :

Sachant que SonarQube ne permet pas d'identifier certains types de problèmes, notamment les erreurs dynamiques, les erreurs algorithmiques ou logiques, ainsi que les problèmes de maintenabilité, de conception ou d'architecture nous avons relevé les points suivants dans notre code :

- Dans la méthode **getReservationById**, Si aucune réservation n'est trouvée (rs.next() retourne false), la méthode retourne **reservation null** , donc il est préférable de lever une **exception** ou de retourner une valeur spécifique pour indiquer que la réservation n'a pas été trouvée.
- La méthode **main** dans la classe Home ne ferme pas la connexion (**connection**). Cela pourrait entraîner des **problèmes de ressources non libérées**.
- On remarque une **duplication** de code dans la gestion des exceptions (try-catch-finally), on peut proposer comme solution la **factorisation** de ces exceptions dans une une classe Exception qui contient toutes les méthodes qui gère les types d'exceptions levées.

V. L'impact du code dupliqué sur la maintenabilité du code :

L'influence du **code dupliqué** sur la **maintenabilité** du logiciel constitue un point central de notre analyse. En effet, lorsqu'une modification ou une correction doit être effectuée, il est nécessaire de l'appliquer à **toutes les parties dupliquées**, ce qui **allonge considérablement le temps de développement**.

Cette duplication augmente également le **risque d'erreurs**, car il est facile d'oublier de modifier une copie, ce qui peut provoquer des **incohérences** dans le fonctionnement du programme. Par ailleurs, le code dupliqué rend le programme plus **long et moins lisible**, ce qui complique sa compréhension et sa maintenance par les développeurs.

Compte tenu de ces effets négatifs, il est essentiel d'adopter des mesures pour **réduire la duplication du code**, afin d'assurer une **qualité durable** et une **maintenance plus efficace** du logiciel.

Dans l'exemple de notre code, il y avait une duplication de code au niveau de la classe **Home** et des classes de tests, le code dupliqué créait des objets et les insérait dans la base de données. Si on souhaite ajouter des models plus tard et les insérer dans la base on doit modifier le code dans tous ces fichiers. La maintenabilité devient difficile avec le temps.

VI. Conclusion :

Le code review avec SonarQube nous a permis d'identifier efficacement les bugs, Code Smell, vulnérabilités,...etc. Le résultat obtenu est un code propre et lisible respectant les normes de codage et facilement maintenable. Cela démontre l'importance d'usage de cet outil durant le développement et surtout avant le déploiement en production.