**Rapport Projet : Architecture des composants d’entreprise**

Préparée par : El mardi oumaima/boutata imane/karchaou hafsa

**1. Introduction**

* Aperçu du projet

Le projet a pour objectif la conception et la mise en place d'un système de gestion intégré pour la Taxe sur les Terrains Non Bâtis (TNB) au Maroc. La TNB représente un montant annuel prélevé sur les terrains qui n'ont pas encore été développés ou construits. L'initiative de ce système vise à simplifier et à automatiser le processus de perception de la taxe, offrant une gestion plus efficace et transparente pour les autorités fiscales et les propriétaires de terrains.

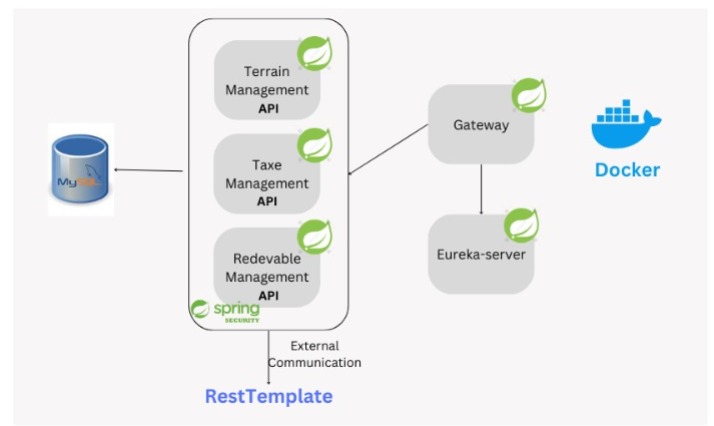
1. Suivi des Terrains des Redevables
2. Détermination des Taux de Taxe par Catégorie de Terrain
3. Calcul de la Taxe TNB Annuelle
4. Recherche par CIN et Consultation de l'Historique des Taxes
5. Interface Utilisateur Conviviale
6. Sécurité et Confidentialité

* Importance de l'architecture microservices

Une architecture microservices permet de diviser l'application en modules indépendants, chaque service se concentrant sur une fonctionnalité spécifique. Dans le cas de la gestion de la TNB, cela pourrait signifier des services distincts pour la gestion des terrains, le calcul des taxes, la recherche par CIN, etc. Cette modularité facilite le développement, la maintenance et l'évolutivité du système.

**2. Architecture Microservices**

* Architecture



* Description des services

**Redevable Service :**

La classe UserDetailsServiceImpl est un service Spring chargé de l'authentification des utilisateurs et de la récupération de données. Elle implémente l'interface UserDetailsService et utilise l'injection de dépendances pour un référentiel d'utilisateurs (UserRepository) et un RestTemplate. La classe comprend des méthodes pour charger les détails d'un utilisateur par nom d'utilisateur, trouver des terrains par numéro d'identification d'utilisateur (CIN) et récupérer l'historique fiscal d'un utilisateur. Elle communique avec un service externe pour obtenir des informations sur l'historique fiscal.

**TaxeService**

La classe TaxeTNBService est un service Spring qui gère les opérations liées à la taxe TNB. Elle utilise un RestTemplate pour communiquer avec un service externe de gestion des terrains. Les principales fonctionnalités incluent la récupération de toutes les taxes TNB, l'enregistrement d'une nouvelle taxe avec validation de l'existence du terrain associé, la récupération de l'historique des taxes pour un terrain donné, et la recherche d'une taxe par terrain et année. La classe gère également les erreurs, notamment en lançant une exception si un terrain n'est pas trouvé lors des opérations.

**TerrainService**

chargé de gérer les opérations liées aux terrains. Il inclut des opérations CRUD telles que la récupération de tous les terrains, l'enregistrement d'un nouveau terrain avec des validations, la récupération d'un terrain par ID, le calcul de la taxe en fonction de la catégorie du terrain, et plus encore. La classe utilise l'injection de dépendances pour un RestTemplate, un CategorieService, et un TerrainRepository. Elle communique avec d'autres services en utilisant RestTemplate pour obtenir des informations supplémentaires. Les méthodes gèrent les validations et le traitement des erreurs, assurant l'intégrité des données.

**CategorieService**

est un service Spring dédié à la gestion des catégories de terrains. Il fournit des fonctionnalités telles que la récupération de toutes les catégories, l'enregistrement d'une nouvelle catégorie, la recherche par ID ou nom, et la récupération du taux de taxe associé à une catégorie. La classe gère également les erreurs en vérifiant l'existence de la catégorie avant de l'enregistrer. En résumé, elle offre des opérations CRUD pour les catégories de terrains et des fonctionnalités spécifiques liées aux taux de taxe.

* Mécanismes de communication

Le mécanisme de communication repose principalement sur l'utilisation de la classe RestTemplate fournie par le framework Spring.

L'utilisation de la communication REST et de RestTemplate permet à l'application d'interagir de manière transparente avec des services distants, facilitant l'échange de données nécessaire pour les opérations métier. Cette approche soutient une architecture modulaire et distribuée, où chaque service peut demander et recevoir indépendamment des informations pertinentes provenant de composants externes.

**3. Conception des Microservices**

* Approche de conception pour chaque service :

**Service de Gestion des Taxes (TaxeService) :**

Responsabilité : Le service de gestion des taxes est chargé de gérer toutes les opérations liées aux taxes foncières (TaxeTNB). Cela inclut la création, la récupération, la modification et la suppression des taxes, ainsi que la fourniture de l'historique fiscal.

**Approche de Conception :**

API RESTful : Exposition d'une API RESTful pour permettre l'accès aux fonctionnalités du service.

Microservices Indépendants : Conception du service de manière à être indépendant des autres services. Il peut fonctionner et évoluer de manière autonome.

**Service de Gestion des Terrains (TerrainService) :**

**Responsabilité :** Ce service est chargé de gérer toutes les opérations liées aux terrains. Cela comprend la création, la récupération, la modification et la suppression des informations sur les terrains, ainsi que l'association avec les catégories de terrains.

**Approche de Conception :**

API RESTful : Exposition d'une API RESTful pour permettre aux utilisateurs d'interagir avec les fonctionnalités du service.

Communication avec d'Autres Services : Interaction avec d'autres microservices, tels que le service de gestion des taxes, pour obtenir des informations connexes lorsqu'elles sont nécessaires.

**Service de Gestion des Redevables (RedevableService) :**

**Responsabilité :** Le service de gestion des redevables est chargé de gérer toutes les opérations liées aux utilisateurs ou entités responsables du paiement des taxes. Cela inclut la création, la récupération, la modification et la suppression des informations sur les redevables, ainsi que l'association avec les terrains qu'ils possèdent.

**Approche de Conception :**

API RESTful : Exposition d'une API RESTful pour permettre l'accès aux fonctionnalités du service, notamment la gestion des redevables et de leurs informations.

Communication avec d'Autres Services : Interaction avec d'autres microservices, comme le service de gestion des terrains, pour obtenir des informations associées.

Gestion des Autorisations : Mise en place de contrôles d'autorisation pour déterminer les actions que chaque redevable est autorisé à effectuer.

**Service de Registre Eureka :**

*Responsabilité :* Eureka est un service de registre qui permet aux différents microservices de s'enregistrer et de découvrir dynamiquement les autres services disponibles dans le système distribué. Il agit comme un annuaire pour faciliter la localisation des services.

**Approche de Conception :**

Registre des Services : Les microservices s'enregistrent sur Eureka au démarrage, indiquant leur disponibilité et leurs informations de localisation.

Découverte de Services : Les microservices utilisent Eureka pour découvrir les autres services disponibles. Cela permet une communication dynamique entre les services sans nécessiter de configuration statique.

Haute Disponibilité : Eureka est conçu pour être hautement disponible, assurant la disponibilité constante du registre de services même en cas de défaillance d'un nœud.

Monitoring : Mise en place de mécanismes de surveillance pour suivre la santé des services enregistrés et pour identifier les services défaillants.

**API Gateway :**

Responsabilité : L'API Gateway sert de point d'entrée principal pour les clients qui souhaitent interagir avec le système. Il agit comme une passerelle pour rediriger les requêtes vers les microservices appropriés.

**Approche de Conception :**

Entrée Unique : Toutes les requêtes clients entrent par l'API Gateway, simplifiant ainsi le point d'entrée pour les clients externes.

Routage Dynamique : L'API Gateway utilise les informations de registre d'Eureka pour router dynamiquement les requêtes vers les microservices correspondants.

**4. Conteneurisation avec Docker**

* Implémentation et avantages

DockerFiles:

* **Eureka :**
* FROM openjdk:17-alpine  
    
  WORKDIR /app  
    
  COPY target/eureka-server-0.0.1-SNAPSHOT.jar /app/eureka-server.jar  
    
  EXPOSE 8761  
    
  CMD ["java", "-jar", "/app/eureka-server.jar"]
* **Gateway :**
* FROM openjdk:17-alpine  
    
  WORKDIR /app  
    
  COPY target/gateway-0.0.1-SNAPSHOT.jar /app/gateway.jar  
    
  EXPOSE 8888  
    
  CMD ["java", "-jar", "/app/gateway.jar"]
* **Redevable Management :**
* FROM openjdk:17-alpine  
    
  WORKDIR /app  
    
  COPY target/spring-boot-security-jwt-0.0.1-SNAPSHOT.jar /app/spring-boot-security-jwt.jar  
    
  EXPOSE 8094  
    
  CMD ["java", "-jar", "/app/spring-boot-security-jwt.jar"]
* **Taxe Management:**
* FROM openjdk:17-alpine  
    
  WORKDIR /app  
    
  COPY target/TaxeManagement-0.0.1-SNAPSHOT.jar /app/TaxeManagement.jar  
    
  EXPOSE 8085  
    
  CMD ["java", "-jar", "/app/TaxeManagement.jar"]
* **Terrain Management:**
* FROM openjdk:17-alpine  
    
  WORKDIR /app  
    
  COPY target/TerrainManagement-0.0.1-SNAPSHOT.jar /app/TerrainManagement.jar  
    
  EXPOSE 8084  
    
  CMD ["java", "-jar", "/app/TerrainManagement.jar"]
* **DOCKER COMPOSE:**
* version: '3.8'  
    
  services:  
    
   eurekaproject:  
   build:  
   context: ./eureka-server  
   ports:  
   - "8761:8761"  
    
    
   terrainmanagement:  
   build:  
   context: ./TerrainManagement  
   ports:  
   - "8084:8084"  
   depends\_on:  
   - mysql  
   environment:  
   SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql:3306/taxe?createDatabaseIfNotExist=true&characterEncoding=utf-8  
   SPRING\_DATASOURCE\_USERNAME: root  
   SPRING\_DATASOURCE\_PASSWORD: root  
    
    
    
    
   spring-boot-security-jwt:  
   build:  
   context: ./RedevableManagement  
   ports:  
   - "8094:8094"  
   depends\_on:  
   - mysql  
   environment:  
   SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql:3306/taxe?createDatabaseIfNotExist=true&characterEncoding=utf-8  
   SPRING\_DATASOURCE\_USERNAME: root  
   SPRING\_DATASOURCE\_PASSWORD: root  
   BEZKODER\_APP\_JWTSECRET: bezKoderSecretKey  
   BEZKODER\_APP\_JWTEXPIRATIONMS: 86400000  
    
   taxemanagement:  
   build:  
   context: ./TaxeManagement  
   ports:  
   - "8085:8085"  
   depends\_on:  
   - mysql  
   environment:  
   SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql:3306/taxe?createDatabaseIfNotExist=true&characterEncoding=utf-8  
   SPRING\_DATASOURCE\_USERNAME: root  
   SPRING\_DATASOURCE\_PASSWORD: root  
    
    
    
   gateway:  
   build:  
   context: ./gateway  
   ports:  
   - "8888:8888"  
   depends\_on:  
   - mysql  
   environment:  
   SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql:3306/taxe?createDatabaseIfNotExist=true&characterEncoding=utf-8  
   SPRING\_DATASOURCE\_USERNAME: root  
   SPRING\_DATASOURCE\_PASSWORD: root  
    
    
   phpmyadmin:  
   image: phpmyadmin/phpmyadmin  
   environment:  
   PMA\_HOST: mysql  
   PMA\_PORT: 3306  
   MYSQL\_ROOT\_PASSWORD: root  
   ports:  
   - "8080:8080"  
    
   mysql:  
   image: mysql:latest  
   environment:  
   MYSQL\_ROOT\_PASSWORD: root  
   MYSQL\_DATABASE: taxe  
   ports:  
   - "3306:3306"

**AVANTAGE:**

l'utilisation de Docker présente plusieurs avantages spécifiques :

**Isolation des Services** :

Docker permet d'isoler chaque service dans son propre conteneur, garantissant que les dépendances, les bibliothèques, et les configurations spécifiques à chaque service sont encapsulées. Cela évite les conflits potentiels entre les différentes parties de l'application.

Facilité de Déploiement :

Chaque service peut être déployé indépendamment dans son propre conteneur Docker. Cela simplifie le déploiement des microservices, facilitant ainsi la gestion des mises à jour et des versions sans impacter les autres services.

**Gestion des Dépendances :**

Les fichiers Dockerfile associés à chaque service définissent clairement les dépendances et les configurations nécessaires. Cela simplifie le processus de développement en assurant que chaque service dispose de son propre environnement d'exécution.

**Réplication et Évolutivité :**

Docker facilite la réplication de services pour gérer une charge accrue. L'évolutivité horizontale devient plus simple en répliquant les conteneurs Docker des services qui nécessitent une mise à l'échelle.

**Gestion des Versions et des Environnements :**

Chaque version d'un service peut être encapsulée dans une image Docker spécifique. Cela permet de gérer facilement les versions des microservices et d'assurer une cohérence entre les environnements de développement, de test et de production.

**Développement Local Simplifié :**

Les développeurs peuvent travailler sur chaque service individuellement en exécutant des conteneurs Docker localement. Cela élimine les problèmes liés aux différences d'environnement entre les machines de développement et de production.

**Sécurité :**

L'isolation des conteneurs Docker contribue à renforcer la sécurité en limitant l'impact potentiel d'une faille de sécurité à un conteneur spécifique plutôt qu'à l'ensemble du système.

**Gestion des Configurations :**

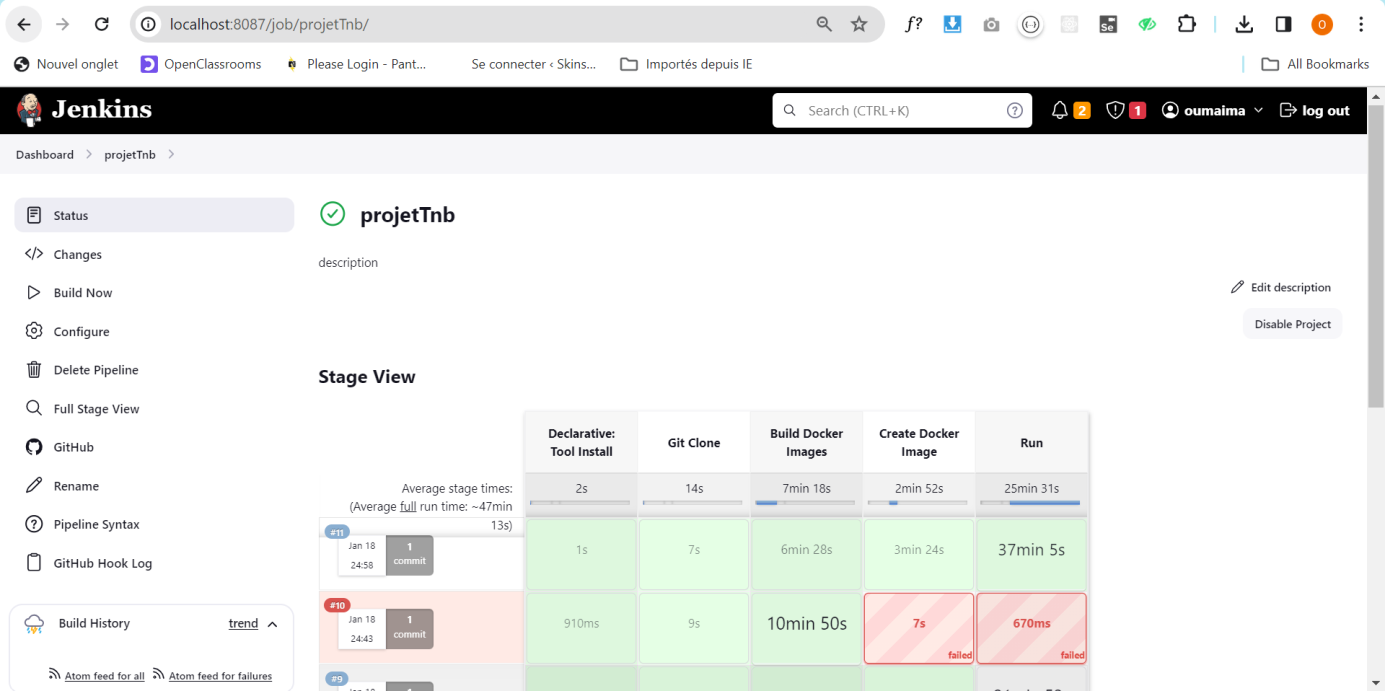
Les fichiers de configuration Docker Compose peuvent être utilisés pour définir et orchestrer l'ensemble des services nécessaires à l'application. Cela simplifie la gestion des configurations interdépendantes entre les services.

**Tests et Intégration Continue :**

Les conteneurs Docker facilitent l'intégration continue et les tests automatisés. Chaque service peut être testé individuellement dans un environnement isolé, améliorant la qualité du code et la détection précoce des erreurs.

En somme, l'utilisation de Docker dans le contexte de ces services permet une gestion plus efficace, évolutive, et sécurisée de l'ensemble de l'architecture de microservices, en mettant l'accent sur l'isolation, la portabilité, et la simplification des opérations de déploiement.

**5. CI/CD avec Jenkins**

****

* Processus et configuration

**PIPELINE:**

**pipeline {**

**agent any**

**tools {**

**maven 'maven'**

**}**

**stages {**

**stage('Git Clone') {**

**steps {**

**script {**

**checkout([$class: 'GitSCM', branches: [[name: 'main']], userRemoteConfigs: [[url: 'https://github.com/ImaneBoutata/Projet-TNB.git']]])**

**}**

**}**

**}**

**stage('Build Docker Images') {**

**steps {**

**script {**

**def microservice\_names = [**

**'eureka-server',**

**'gateway',**

**'TerrainManagement',**

**'TaxeManagement',**

**'RedevableManagement'**

**]**

**dir('backend') {**

**for (microservice in microservice\_names) {**

**dir(microservice) {**

**echo "Building Docker image for ${microservice}"**

**script {**

**bat 'mvn clean install -DskipTests'**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**stage('Create Docker Image') {**

**steps {**

**script {**

**dir('backend') {**

**bat 'docker-compose build'**

**}**

**}**

**}**

**}**

**stage('Run') {**

**steps {**

**script {**

**dir('backend') {**

**bat 'docker-compose up'**

**}**

**}**

**}**

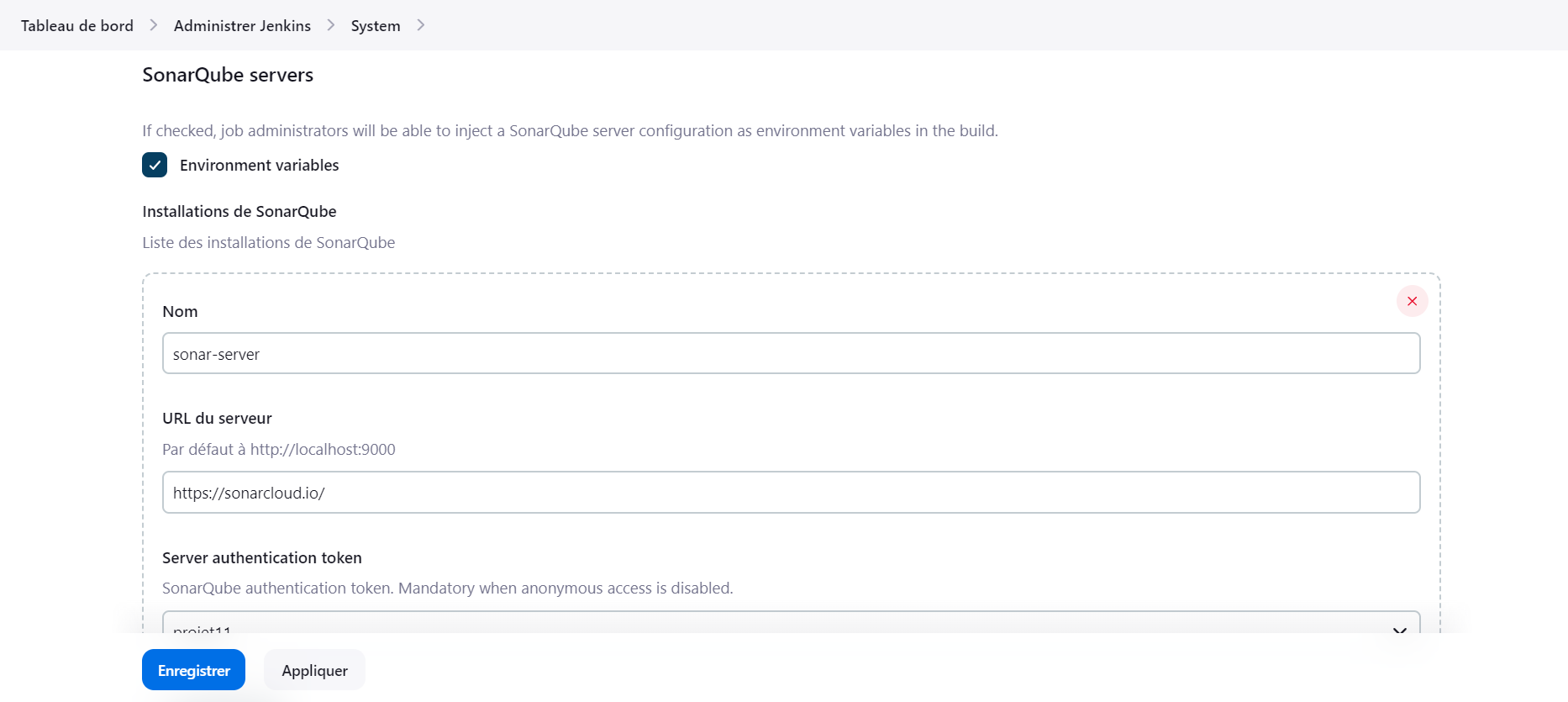
**}**

**}**

**}**

**6.Intégration de SonarQube**

1. Configuration et bénéfices pour la qualité du code
2. sonar.projectKey=sonarqubetest1234\_projet11  
   sonar.sources=.  
   sonar.login=96d0b985d84a0933719416411cef5d9a1adb6e45  
   sona.host.url=https://sonarcloud.io  
   sonar.sourceEncoding=UTF-8  
   sonar.organization=sonarqubetest1234  
   # Eureka Server  
   sonar.java.binaries=eureka-server/target/classes  
     
   # Gateway  
   sonar.java.binaries=gateway/target/classes  
     
   # TerrainManagement  
   sonar.java.binaries=TerrainManagement/target/classes  
     
   # TaxeManagement  
   sonar.java.binaries=TaxeManagement/target/classes  
     
   # RedevableManagement  
   sonar.java.binaries=RedevableManagement/target/classes
   * Configuration de sonarqube dans jenkins :



**Pipeline :**

node {

stage('SCM') {

steps {

script {

checkout([$class: 'GitSCM', branches: [[name: 'main']], userRemoteConfigs: [[url: 'https://github.com/ImaneBoutata/Projet-TNB.git']]])

}

}

}

stage('SonarQube analysis') {

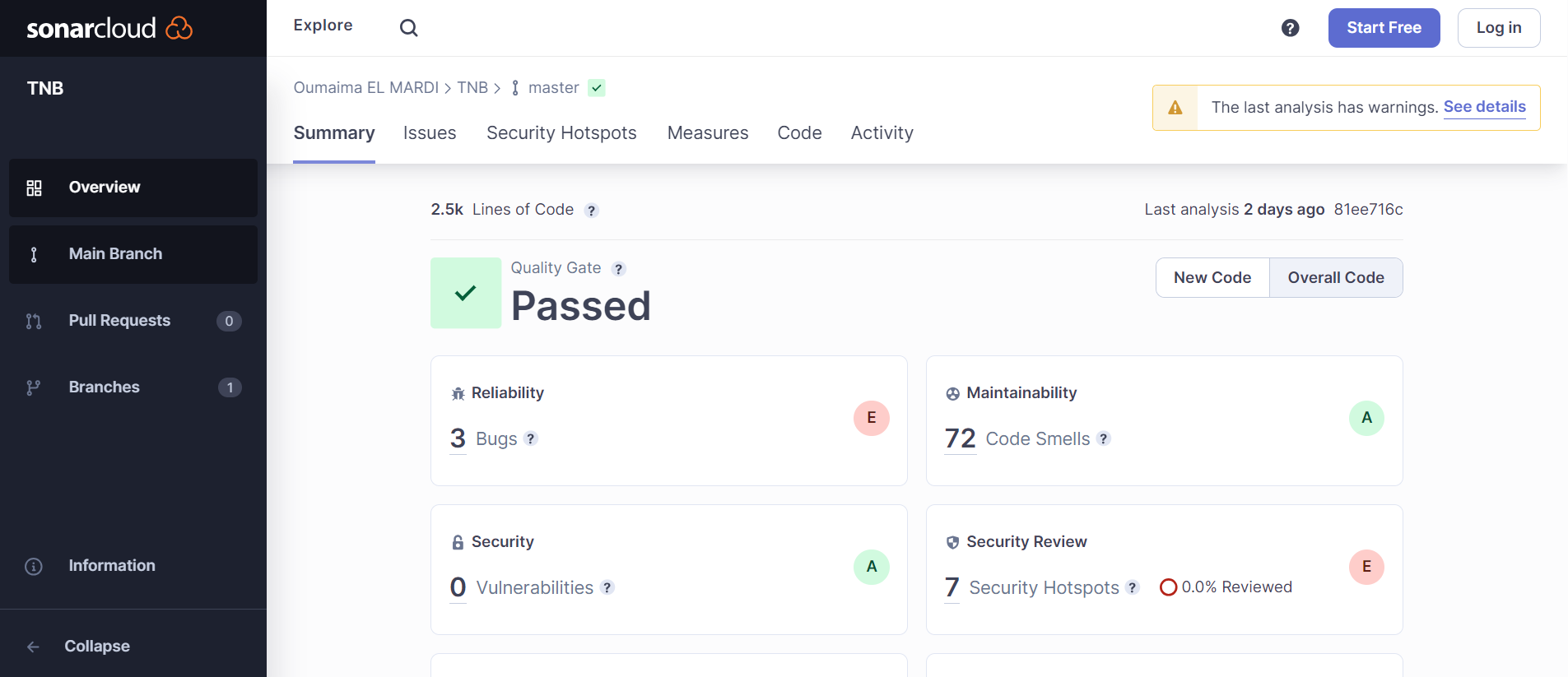
withSonarQubeEnv(credentialsId: '6ec3079ec5eae421f9b090f7c6abcfc2f29ef9ed', installationName: 'sonar-server')

bat 'mvn org.sonarsource.scanner.maven:sonar-maven-plugin:3.6.0.1398:sonar'

}

}

}



**7. Conclusion**

* **Résumé des accomplissements**

L'intégration réussie de Jenkins, Docker et SonarQube a considérablement amélioré notre processus de développement logiciel. Grâce à l'utilisation de Jenkins, nous avons automatisé le cycle de vie de notre application, de la compilation à la livraison, garantissant ainsi des déploiements rapides et cohérents. L'adoption de Docker a introduit une gestion efficace des dépendances et assuré la portabilité de nos applications, simplifiant ainsi le déploiement sur différents environnements. L'intégration de SonarQube a permis une analyse approfondie de notre code source, identifiant les vulnérabilités, les erreurs de codage et les pratiques non optimales. Cette démarche a renforcé la qualité globale de notre code et facilité la détection précoce des problèmes. Globalement, ces réalisations ont amélioré notre agilité, notre efficacité opérationnelle et notre capacité à maintenir des normes élevées de qualité du code dans un environnement de développement dynamique.

* **Perspectives futures**

À mesure que nous continuons à évoluer dans notre approche de développement logiciel, plusieurs perspectives futures sont envisagées pour renforcer notre infrastructure et optimiser davantage notre processus de livraison :

**Automatisation Avancée :**

Exploration de l'automatisation avancée en intégrant des outils complémentaires à notre pipeline CI/CD pour automatiser des tâches spécifiques, améliorant ainsi l'efficacité opérationnelle.

**Container Orchestration :**

Considération de l'intégration de solutions de gestion de conteneurs, telles que Kubernetes, pour une orchestration plus avancée, une gestion des mises à l'échelle et une haute disponibilité accrue.

**Déploiement dans le Cloud :**

Évaluation de la possibilité de migrer vers des solutions de déploiement dans le cloud, permettant une flexibilité accrue et une gestion simplifiée des ressources.

**Tests Automatisés Étendus :**

Mise en place de tests automatisés étendus, y compris des tests de performance, des tests de sécurité et des tests d'intégration continue, pour garantir la robustesse de l'application.