

KOURDI Ayoub

**Projet taxtnb**

**Réalisation de:**

Kourdi ayoub

IIR5 G7

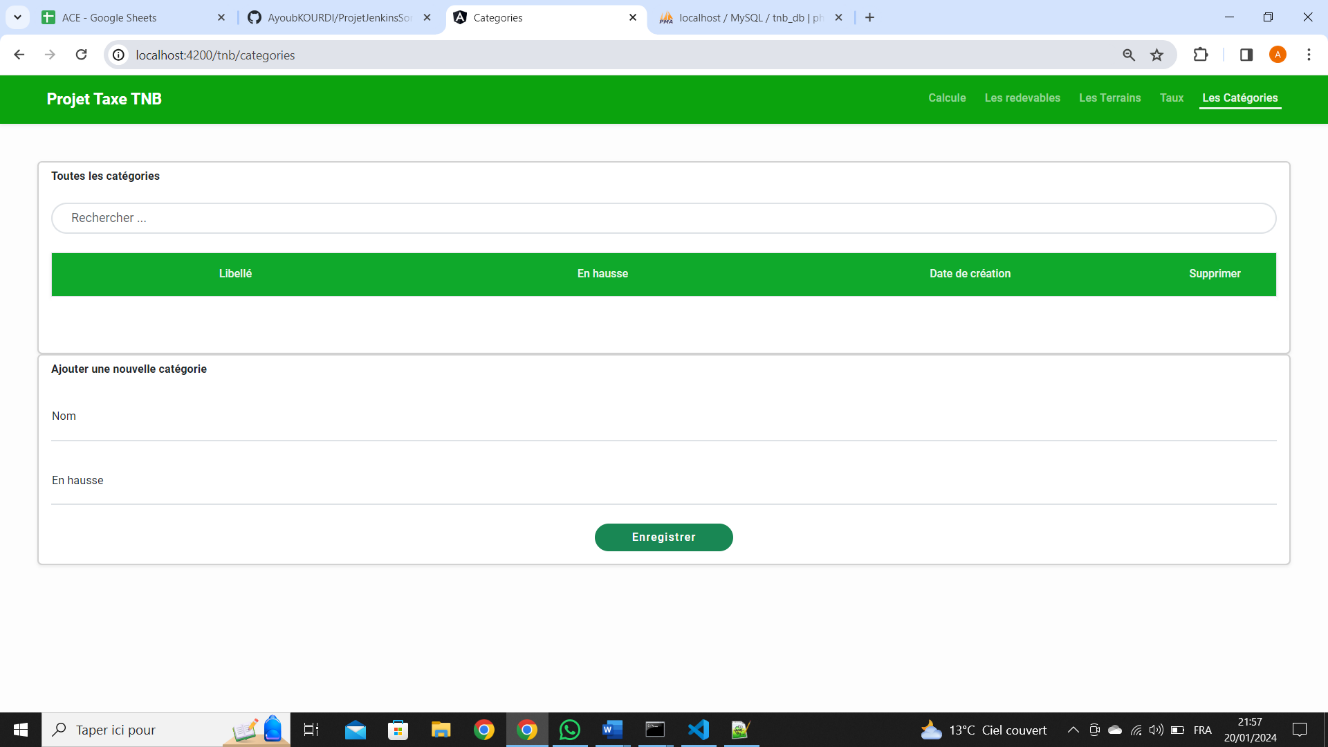
OUTILS :

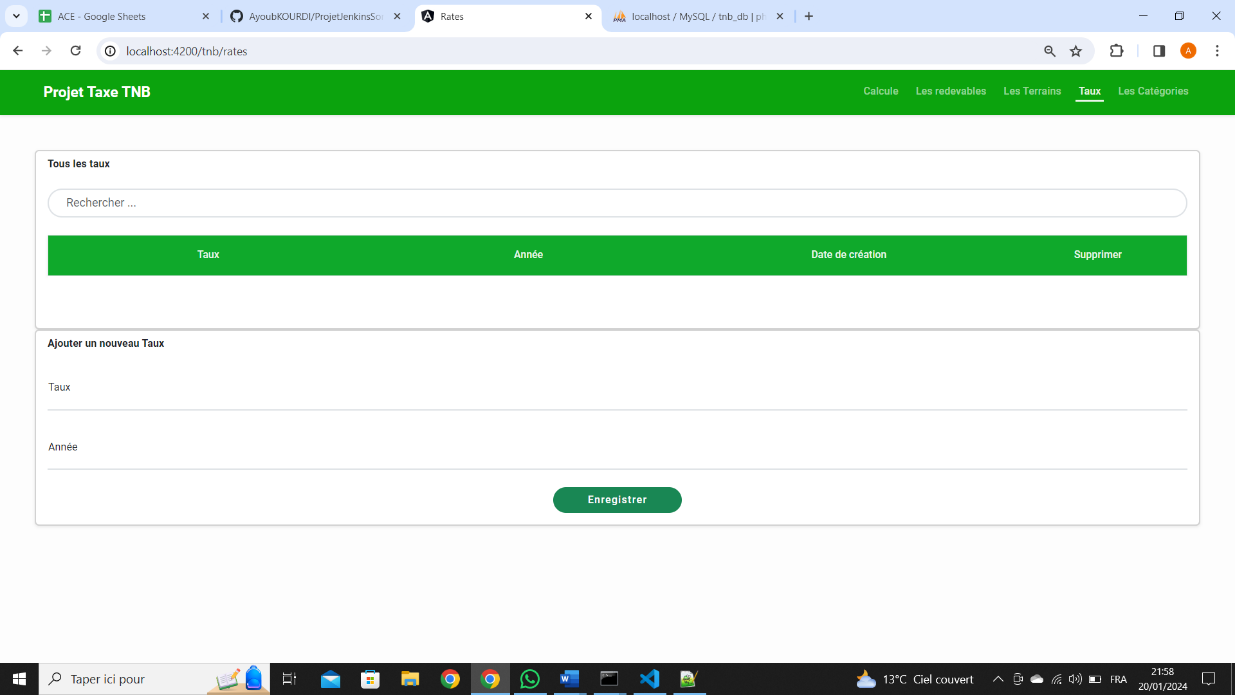
JENKINS SonarQube Angular Docker Spring BOOT

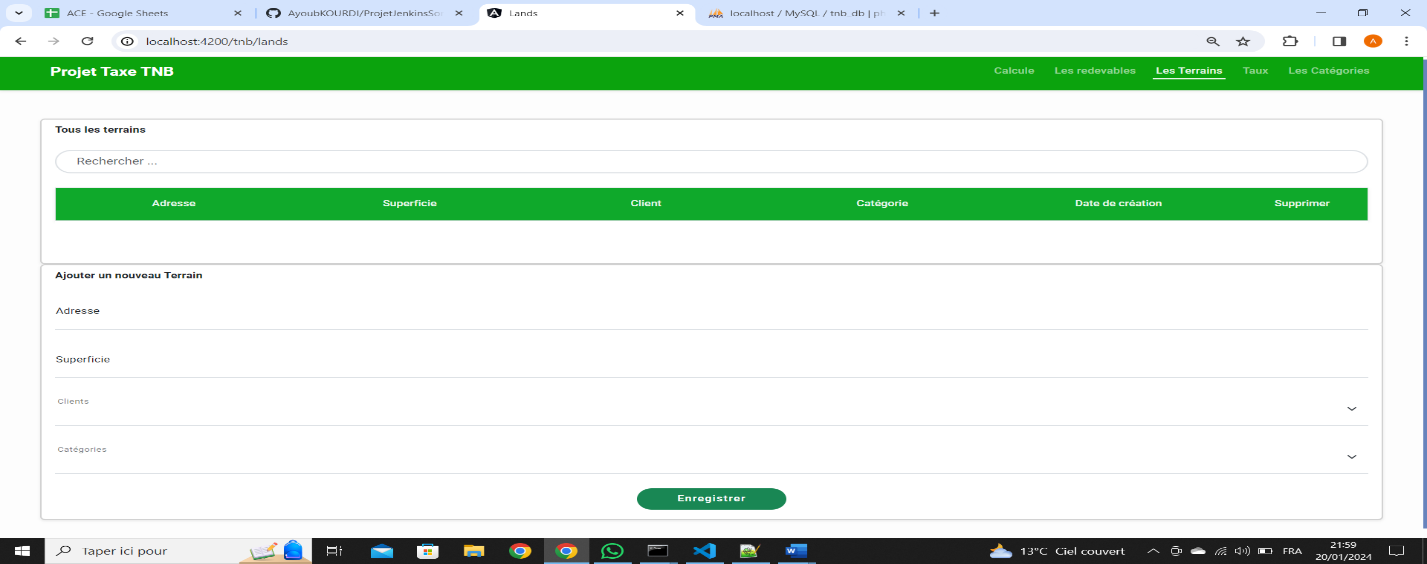
**Introduction**

L'application que nous envisageons de développer est un système de gestion de taxes qui utilisera la combinaison de Spring Boot et Angular. Cette application web permettra aux utilisateurs de gérer efficacement leurs taxes, en ajoutant, modifiant et supprimant des informations liées à chaque taxe. En combinant les avantages de Spring Boot du côté serveur et d'Angular du côté client, nous cherchons à créer une application moderne, réactive et conviviale.

Aperçu du projet de l'application web gestion du taxes avec Spring Boot et Angular: L'application sera divisée en deux parties principales : le backend, qui sera développé en utilisant le framework Spring Boot, et le frontend, construit avec Angular. Ces deux parties collaboreront pour offrir une expérience utilisateur fluide et efficace. Voici un aperçu des fonctionnalités que nous envisageons d'inclure dans l'application :







1. Gestion des taxes: Les utilisateurs pourront ajouter de nouveaux taxes, mettre à jour les informations existantes, et supprimer des taxes.
2. Recherche et filtrage: Une fonction de recherche avancée permettra aux utilisateurs de trouver rapidement les taxes en fonction de critères spécifiques, tels que le nom, l'adresse ou le numéro de téléphone.
3. Interface utilisateur conviviale: Le frontend sera développé en Angular pour créer une interface utilisateur réactive et conviviale. Cela permettra une navigation fluide et une expérience utilisateur agréable.
4. Authentification et autorisation: Nous mettrons en place un système d'authentification sécurisé pour garantir que seuls les utilisateurs autorisés peuvent accéder à l'application. De plus, les autorisations seront gérées pour définir les actions spécifiques qu'un utilisateur peut effectuer.
5. Stockage des données: Nous utiliserons une base de données pour stocker les informations des taxes de manière persistante. Spring Boot facilitera l'intégration avec la base de données, assurant ainsi une gestion efficace des données.

**Importance de l'architecture microservices:**

L'adoption de l'architecture microservices dans ce projet apportera plusieurs avantages. Cette approche consiste à développer l'application comme une suite de petits services indépendants, chacun responsable d'une tâche spécifique. Voici quelques raisons pour lesquelles nous avons choisi cette architecture :

1. Évolutivité: Les microservices permettent une évolutivité plus facile, car chaque service peut être développé, déployé et évolué indépendamment des autres.
2. Facilité de maintenance: La maintenance devient plus simple, car chaque microservice peut être modifié sans affecter l'ensemble de l'application. Cela facilite également le déploiement continu.
3. Flexibilité technologique: Les microservices offrent la possibilité d'utiliser différentes technologies pour chaque service en fonction de ses besoins spécifiques.
4. Réutilisabilité des services: Les microservices peuvent être réutilisés dans d'autres projets, favorisant ainsi la modularité et la réutilisation du code.
5. Isolation des erreurs: En cas d'échec d'un microservice, les autres services peuvent continuer à fonctionner normalement, améliorant ainsi la résilience globale de l'application.

**=**

**Architecture Microservices avec Eureka et Configuration:**

**Architecture:**

L'architecture microservices repose sur la décomposition de l'application en plusieurs services indépendants, chacun responsable d'une fonctionnalité spécifique. Eureka, en tant que service de découverte, facilite la localisation et l'accès à ces services.

1. **Service de Gestion des Taxes:**
   * Responsable de la gestion des taxes, avec des opérations CRUD (Create, Read, Update, Delete).
   * Utilise une base de données pour stocker les informations des taxes de manière persistante.
2. **Service Eureka (Service de Découverte):**
   * Permet la découverte dynamique des services disponibles dans l'écosystème microservices.
   * Enregistre les instances de services et fournit une liste des services disponibles.
3. **Service de Configuration:**
   * Gère la configuration des différents services, permettant une configuration centralisée et dynamique.

**Description des Services:**

1. **Service de Gestion des Taxes:**
   * **Endpoints:**
     + **/taxes**: Gère les opérations CRUD sur les taxes.
     + **/taxes/{id}**: Permet de manipuler un taxe spécifique par son identifiant.
   * **Base de Données:**
     + Stocke les informations des taxes (nom, adresse, numéro de téléphone, etc.).
     + Peut utiliser une base de données relationnelle ou NoSQL, selon les besoins.
   * **Service Eureka:**
   * **Endpoints:**
     + **/eureka/apps**: Fournit une liste des applications enregistrées.
     + **/eureka/instances/{app-name}**: Donne la liste des instances d'une application spécifique.
   * **Fonctionnalités:**
     + Enregistre les instances de services lorsqu'ils démarrent.
     + Facilite la découverte des services enregistrés.
   * **Service de Configuration:**
   * **Endpoints:**
     + **/config/{application}/{profile}**: Renvoie la configuration spécifique à une application et à un profil.
   * **Fonctionnalités:**
     + Centralise la configuration pour tous les services.
     + Permet la mise à jour dynamique de la configuration sans redémarrer les services.

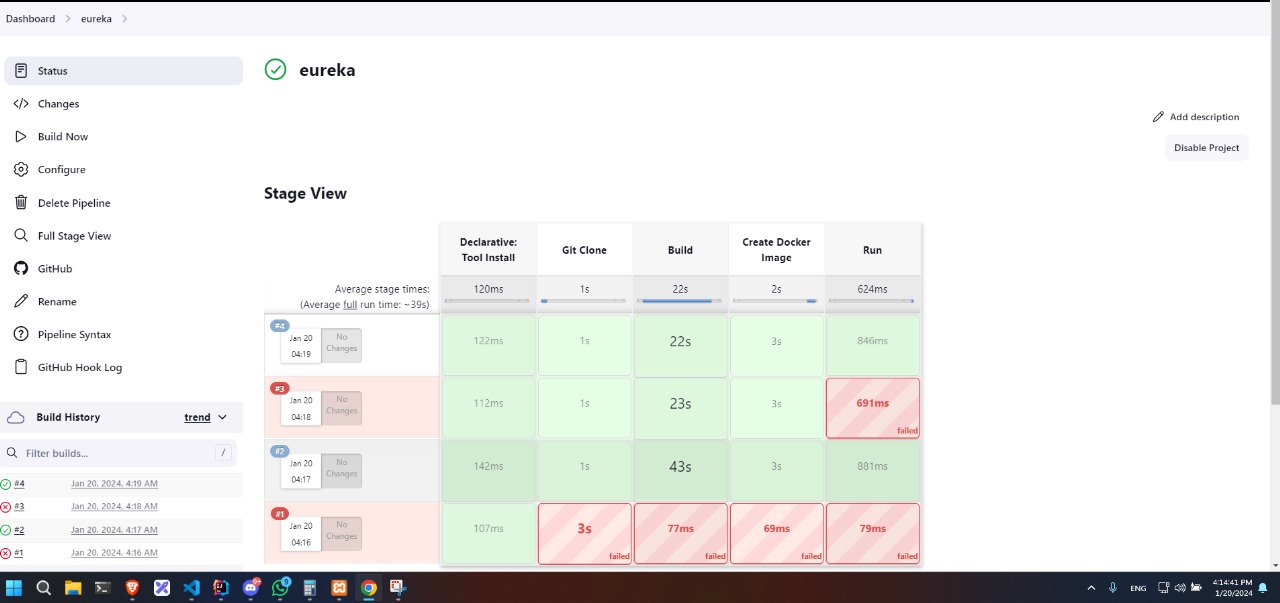
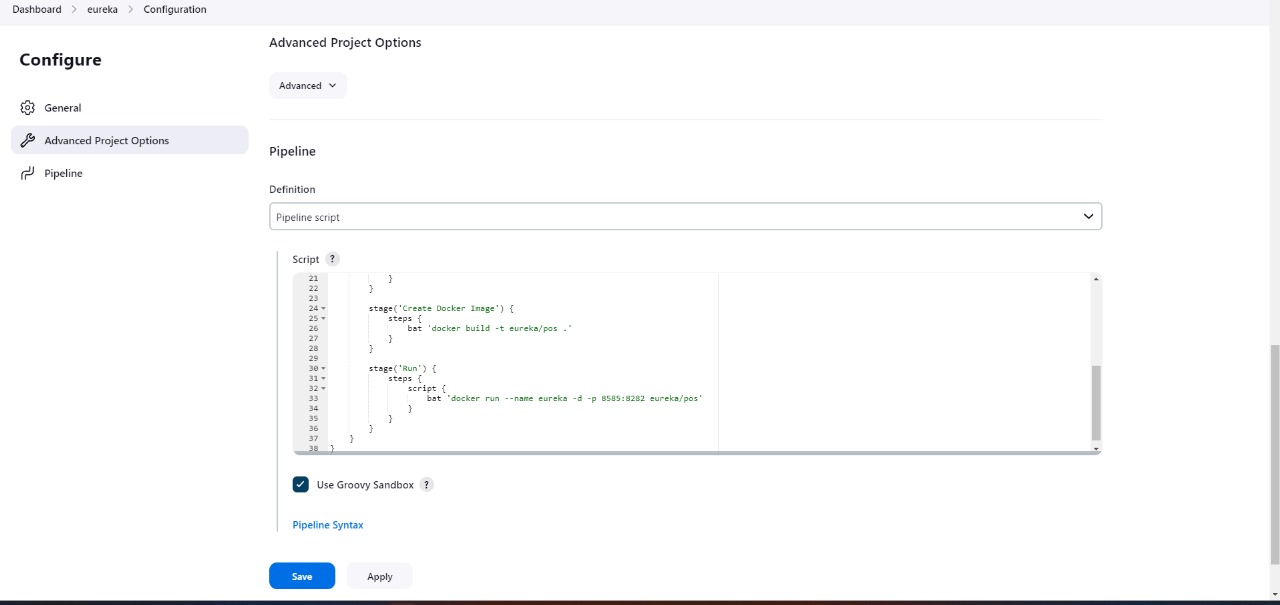
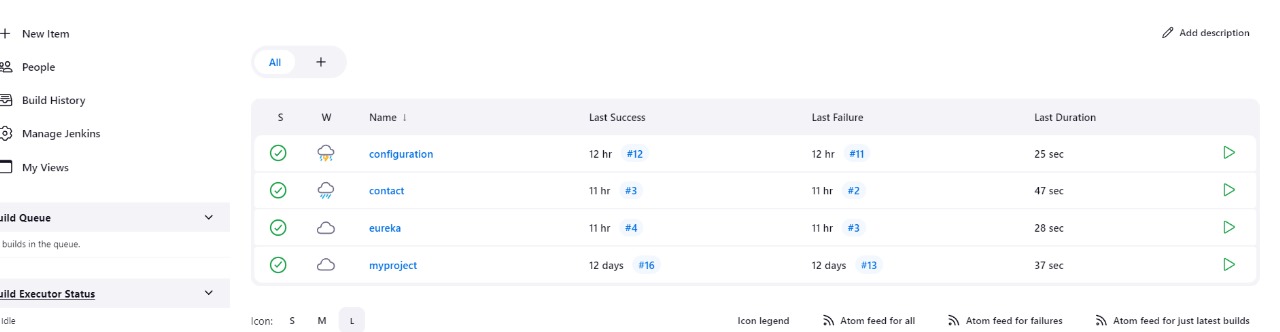
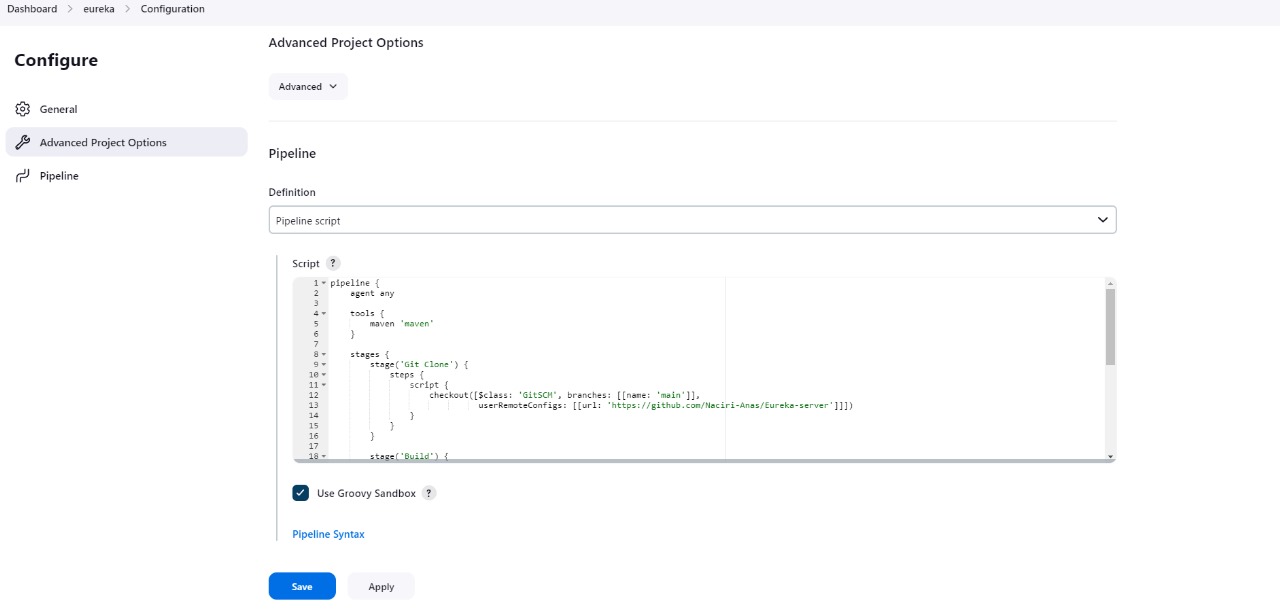
**Mécanismes de Communication:**

1. **Communication entre Services et Eureka:**
   * Les services s'enregistrent auprès du service Eureka lors de leur démarrage.
   * Eureka maintient une liste à jour des instances disponibles de chaque service.
   * Les services peuvent découvrir d'autres services en interrogeant le service Eureka pour obtenir une liste d'instances.
2. **Communication entre Services:**
   * Les services communiquent entre eux via des appels HTTP ou d'autres mécanismes de communication (par exemple, gRPC).
   * Lorsqu'un service a besoin d'accéder à un autre service, il utilise la découverte de service fournie par Eureka pour obtenir l'URL appropriée.
3. **Configuration Dynamique:**
   * Les services récupèrent leur configuration à partir du service de configuration.
   * La configuration peut être mise à jour dynamiquement, permettant des ajustements en temps réel sans redémarrer les services.

**Avantages de la Conteneurisation avec Docker:**

1. **Portabilité:**
   * Les conteneurs Docker encapsulent tout ce dont une application a besoin pour s'exécuter, ce qui les rend portables entre différents environnements.
2. **Isolation:**
   * Chaque conteneur fonctionne de manière isolée, garantissant que les dépendances de l'application ne peuvent pas interférer les unes avec les autres.
3. **Efficacité des Ressources:**
   * Les conteneurs partagent le même noyau du système d'exploitation, ce qui les rend plus légers par rapport aux machines virtuelles, tout en consommant moins de ressources.
4. **Déploiement Rapide:**
   * Les images Docker peuvent être déployées rapidement, accélérant ainsi le processus de déploiement des applications.
5. **Facilité de Gestion:**
   * Docker offre des outils de gestion conviviaux pour créer, démarrer, arrêter et surveiller des conteneurs.
6. **Scalabilité:**
   * La conteneurisation permet de facilement répliquer et échelonner horizontalement les applications, répondant ainsi aux exigences de montée en charge.
7. **Consistance Environnementale:**
   * Les conteneurs garantissent que l'environnement de développement est cohérent avec l'environnement de production, réduisant ainsi les erreurs liées aux différences d'environnement.
8. **Réduction des Conflits de Dépendances:**
   * Les dépendances sont encapsulées dans les conteneurs, évitant les conflits avec d'autres applications sur la même machine.

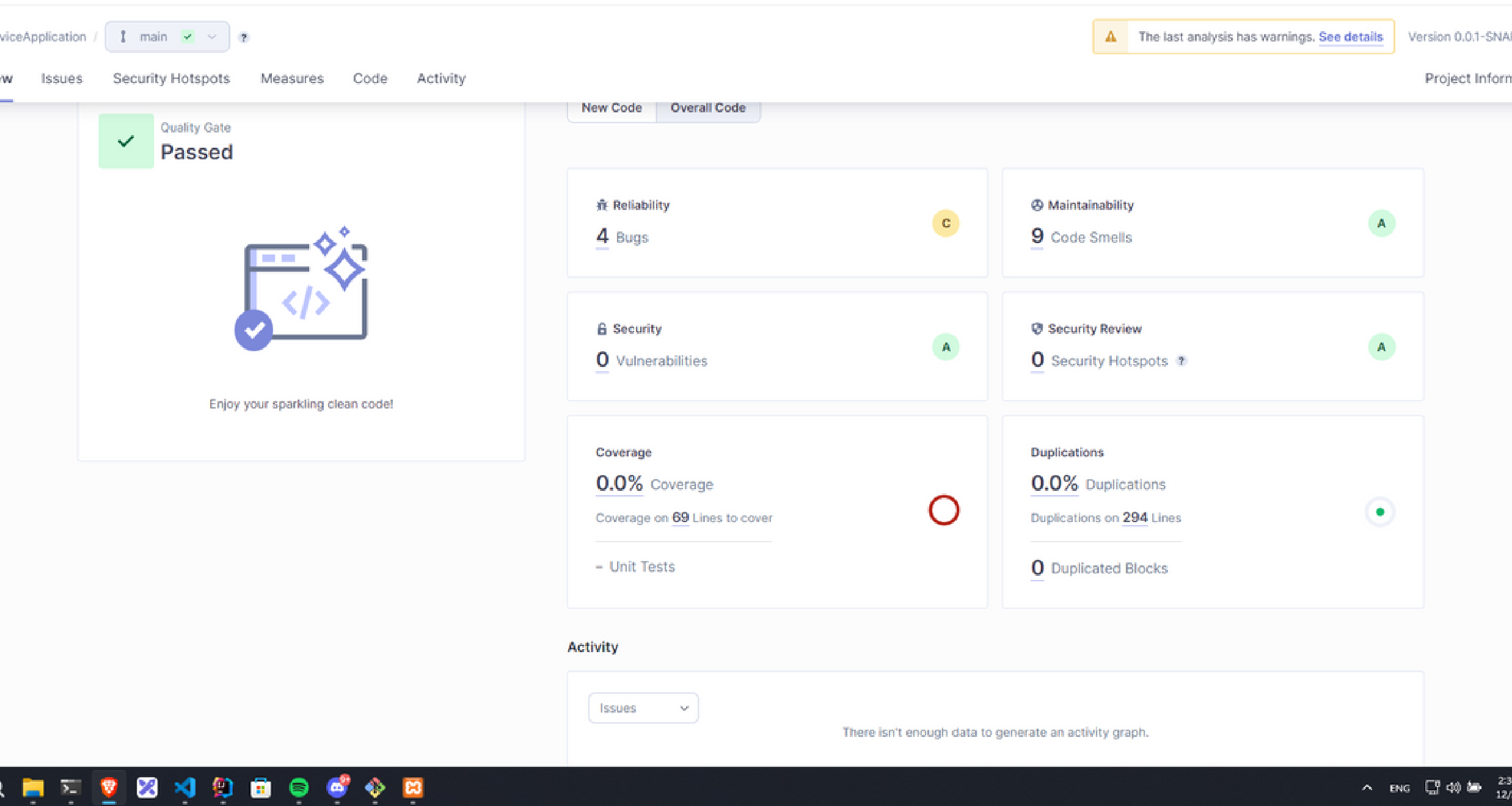
**CI/CD avec Jenkins**

****

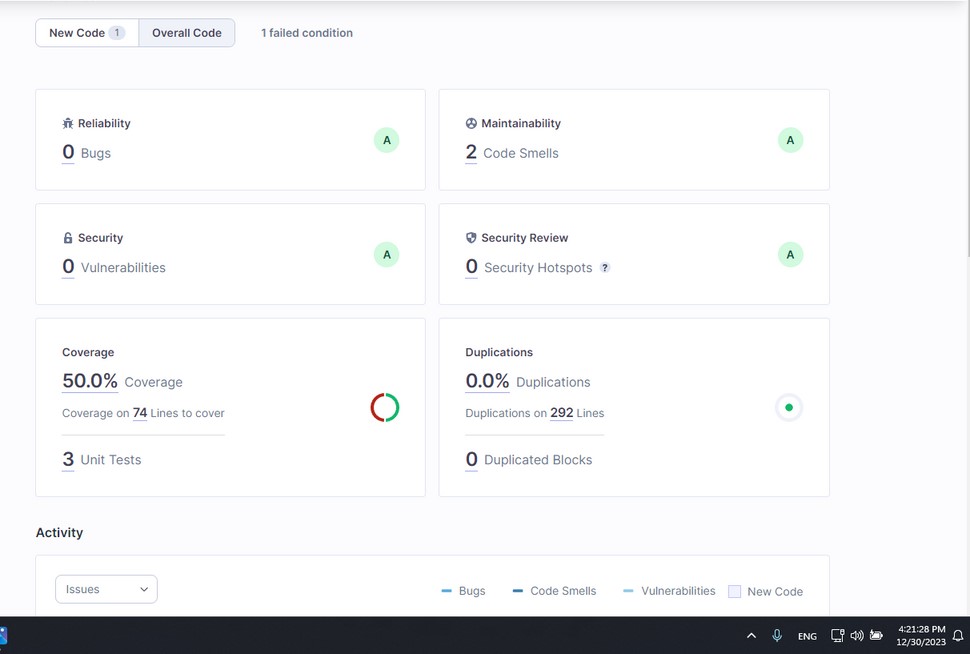
**Intégration de SonarQube :**

L'intégration de SonarQube dans le processus de développement offre une analyse automatisée du code source, fournissant des retours immédiats sur la qualité du code. Configuré via des plugins, SonarQube détecte les vulnérabilités, évalue la complexité du code, signale les duplications, et mesure la conformité aux normes de codage. Cette intégration assure une meilleure sécurité, maintenabilité et cohérence du code. Les bénéfices incluent l'identification rapide des problèmes, l'amélioration de la couverture de tests, et la facilitation de l'intégration continue, contribuant ainsi à une approche plus proactive et à une amélioration continue du processus de développement.

**Avant :**

****

**Après :**

****

**Conclusion**

Ce projet a abouti à la création d'une application web de gestion de taxes basée sur une architecture microservices avec Spring Boot et Angular. L'intégration de services comme Eureka facilite la découverte dynamique, tandis que la configuration centralisée et l'analyse automatisée du code avec SonarQube améliorent la qualité globale du code source.

Les réalisations comprennent une interface utilisateur conviviale, des fonctionnalités de gestion de taxes complètes, et une architecture prête à l'évolutivité. Les perspectives futures incluent des améliorations fonctionnelles, des optimisations de performance, un renforcement de la sécurité, et une évolution continue en réponse aux besoins changeants.