

**1. Introduction**

**Aperçu du Projet**

Ce projet vise à développer un système pour calculer la taxe sur les terrains non bâtis. Il utilise une architecture microservices avec des services spécifiques comme Server (Eureka), Gateway, Tnb\_Taxe et Propriétaire, en plus de base de données MySQL.

**Importance de l'Architecture Microservices**

**L'architecture microservices**, adoptée dans notre projet, offre une gestion flexible et isolée de chaque composant du système. Cette approche décompose le système en services plus petits et indépendants, chacun s'exécutant dans son propre processus. Elle facilite la mise à l'échelle de parties spécifiques du système en réponse à des demandes changeantes, sans affecter l'ensemble de l'infrastructure. De plus, cette isolation améliore significativement la facilité de maintenance, permettant des mises à jour et des corrections individuelles sans perturber les autres services. Cette architecture favorise ainsi une évolution et une expansion agiles du système, tout en assurant une haute disponibilité et une réactivité accrue aux besoins changeants du projet.

**2. Architecture Microservices**

**Architecture**

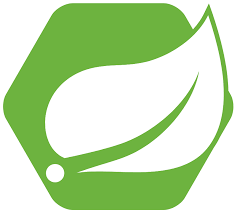


Figure : logo de Eureka Server

En employant **Eureka** comme service de découverte au cœur de notre architecture microservices, nous garantissons une communication fluide et dynamique entre les divers composants. Eureka sert de registre centralisé où chaque microservice s'enregistre et peut localiser d'autres services nécessaires pour fonctionner. Cette intégration permet à chaque composant de découvrir et de communiquer avec les autres services de manière efficace et transparente, sans configuration rigide ou dépendance directe. Cette flexibilité est essentielle pour le maintien d'une architecture réactive et adaptable, facilitant les interactions entre services même en cas d'évolution ou de modification de l'infrastructure sous-jacente. En somme, Eureka contribue à une architecture robuste, capable de gérer dynamiquement la localisation et la disponibilité des services dans un environnement distribué.

**Description des Services**

**Notre projet microservices** s'articule autour de **quatre services** clés, chacun jouant un rôle spécifique et essentiel dans l'architecture globale. Ces services interagissent de manière harmonieuse pour assurer une gestion efficace et optimisée des données relatives à la taxation des terrains non bâtis.

**Voici une présentation détaillée de chaque service :**

**Server (Eureka) - Service de Découverte :** Eureka agit comme le cœur de notre architecture microservices. En tant que service de découverte, il permet à chaque microservice de s'enregistrer et de découvrir d'autres services nécessaires à son fonctionnement. Cette centralisation facilite une communication fluide et une gestion efficace des services dans notre écosystème distribué, permettant ainsi une meilleure scalabilité et résilience du système.

**Gateway - Point d'Entrée pour les Requêtes :** Le Gateway fonctionne comme le point d'entrée unifié pour toutes les requêtes externes. Il dirige les requêtes entrantes vers les microservices appropriés, assurant ainsi une gestion sécurisée et efficace du trafic. Cette approche centralisée du traitement des requêtes facilite la surveillance, la gestion des erreurs et la mise en œuvre de politiques de sécurité cohérentes.

**Tnb\_Taxe - Gestion des Données de Taxation :** Ce service est dédié à la gestion des entités clés du projet telles que Catégorie, Propriétaire, Taux, Taxe\_Tnb et Terrain. Il manipule toutes les opérations liées au calcul et à l'enregistrement des taxes pour les terrains non bâtis, assurant ainsi l'intégrité et l'exactitude des données fiscales.

**Propriétaire - Gestion des Informations des Propriétaires :** Ce service gère les informations détaillées des propriétaires de terrains. Il s'occupe de l'enregistrement, de la mise à jour et de la consultation des données relatives aux propriétaires, jouant un rôle crucial dans l'association des terrains à leurs propriétaires respectifs et dans la gestion des informations liées à la propriété.

En complément de ces services, nous utilisons **MySQL** comme système de base de données pour stocker et gérer toutes les données de l'application. MySQL offre une solution de stockage fiable et efficace, essentielle pour gérer les grandes quantités de données générées et traitées par nos services. Sa robustesse, sa performance et sa facilité d'intégration avec les autres composants de notre architecture microservices en font un choix idéal pour notre projet.

**Chaque service est conçu pour fonctionner de manière autonome tout en collaborant étroitement avec les autres, formant ainsi une structure cohérente et efficace, essentielle pour répondre aux exigences de notre application de calcul de taxe sur les terrains non bâtis.**

**3. Conception des Microservices**

Une image contenant texte, diagramme, Police, capture d’écran

Description générée automatiquement

La conception de notre système de calcul de la taxe sur les terrains non bâtis (TNB) au Maroc est méticuleusement représentée dans le diagramme de classes ci-joint. Ce diagramme illustre la structure et les relations entre les différentes entités du système :

* **Catégorie :** Représente les diverses classifications des terrains, telles que Villa, Maison, Appartement, etc., et est identifiée par un libellé unique.
* **Terrain :** Détient les informations spécifiques à chaque parcelle de terrain, y compris son type, nom, surface, et sa catégorie correspondante. Chaque terrain est associé à un propriétaire, désigné ici par l'entité 'Redevable'.
* **Redevable :** Correspond au propriétaire du terrain, identifié par son CIN, nom, prénom, et adresse. Cette entité permet de relier un terrain à son propriétaire légitime et de retrouver l'historique fiscal associé.
* **TaxeTnb :** Entité qui enregistre le montant de la taxe TNB due pour chaque terrain, en se basant sur le taux applicable et la surface du terrain.
* **Taux :** Spécifie le taux de taxe actuel pour chaque catégorie de terrain, qui peut évoluer au fil du temps.

**Le système est conçu pour répondre aux exigences fonctionnelles suivantes :**

* Classification et gestion des terrains selon des catégories définies, avec un taux de taxe associé qui peut varier avec le temps.
* Recherche de terrains et de l'historique des taxes TNB par CIN du redevable.
* Recherche du taux applicable par catégorie de terrain.
* Calcul de la taxe TNB en multipliant le taux de taxe par la surface du terrain, pourvu que la taxe n'ait pas déjà été payée.

L'objectif est de fournir un système où les entrées (CIN, année, terrain) permettent de déterminer le montant de la taxe TNB due, assurant ainsi une gestion fiscale précise et efficiente. Ce diagramme de classes sert de fondement à la conception logicielle et guide le développement en structurant les données et les fonctionnalités requises pour le calcul de la taxe TNB.

**4. Conteneurisation avec Docker**

Une image contenant Graphique, dessin humoristique, clipart, conception

Description générée automatiquement

Figure : Docker logo

Docker encapsule chaque service de notre architecture microservices, y compris Jenkins, SonarQube et MySQL, dans des conteneurs distincts. Cette isolation garantit une uniformité de l'environnement d'exécution, quel que soit le déploiement. Docker facilite également l'intégration et le déploiement continus (CI/CD) via Jenkins, en permettant des tests et des déploiements cohérents et automatisés. L'intégration de SonarQube dans Docker permet une analyse continue de la qualité du code, assurant des pratiques de codage sûres et efficaces. La légèreté et la portabilité des conteneurs Docker renforcent la flexibilité et la scalabilité de notre système.

**5. CI/CD avec Jenkins**

Une image contenant dessin humoristique, Dessin animé, illustration, clipart

Description générée automatiquement

Figure : Jenkins logo

Jenkins, au cœur de notre système, automatise le pipeline CI/CD, intégrant développement, tests et déploiement. Cet outil clé standardise notre flux de travail, optimisant l'efficacité et la cohérence. Il minimise les erreurs humaines, accélérant ainsi le cycle de vie du développement. Jenkins permet une intégration continue des modifications de code et un déploiement continu des applications, garantissant ainsi une mise en production rapide et fiable des nouvelles fonctionnalités et mises à jour. Cette automatisation assure une réponse agile aux besoins changeants du projet.

Une image contenant texte, Police, nombre, ligne

Description générée automatiquement

Figure : PipeLine du premier MicroService -TaxeTNB-

Une image contenant texte, Police, nombre, ligne

Description générée automatiquement

Figure : PipeLine du deuxième MicroService -Propriètaire-

**6. Déploiement Automatique et Gestion des Services**

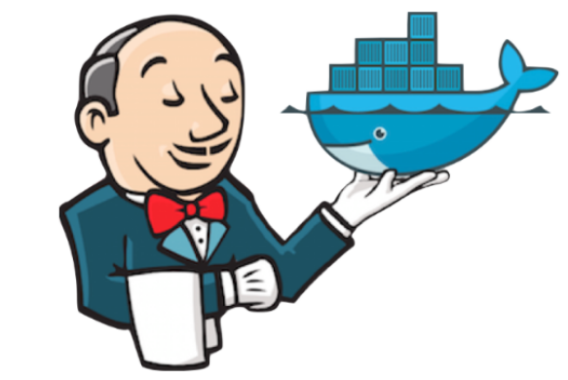


Figure : Jenkins + Docker

Le déploiement automatique dans notre projet est orchestré grâce à Jenkins et Docker. Jenkins gère le CI/CD, facilitant des déploiements rapides et efficaces. Docker assure la conteneurisation, garantissant la cohérence des environnements de déploiement. Eureka, notre service de découverte, gère dynamiquement les instances de service, permettant leur localisation et interaction fluide. Cette synergie entre Jenkins, Docker et Eureka crée un écosystème robuste pour nos microservices, assurant déploiement rapide, fiabilité et adaptabilité face aux exigences évolutives de l'application.

**7. Intégration de SonarQube**



SonarQube, intégré dans notre environnement Docker, joue un rôle essentiel dans notre pipeline CI/CD. Il analyse en continu le code source pour détecter les vulnérabilités, les bugs et les mauvaises pratiques de codage. Cette intégration assure non seulement la qualité et la sécurité de notre code, mais contribue également à améliorer constamment les standards de codage et la maintenabilité du projet. SonarQube fournit ainsi une visibilité claire sur la santé du code et guide l'équipe vers des améliorations ciblées.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Figure : Rapport SonarQube

**8. Conclusion**

Notre projet a brillamment établi une architecture microservices à la fois robuste, flexible et évolutive. L'intégration harmonieuse d'outils de développement, de déploiement et d'assurance qualité, notamment Jenkins, Docker et SonarQube, a joué un rôle crucial dans ce succès. Cette combinaison a permis de créer un environnement où la continuité, l'efficacité et la qualité du développement sont maintenues à un niveau élevé. L'adaptabilité de l'architecture assure sa pertinence face aux évolutions technologiques futures, posant ainsi les bases d'une croissance soutenue et d'une amélioration continue. Ce projet est un exemple remarquable de la manière dont la technologie moderne peut être utilisée pour construire des systèmes complexes, mais gérables, qui répondent aux exigences dynamiques du monde des affaires actuel.