Nix Tax.

Réaliser par : Tolab Haitam,El Messoussi Mohamed Sinane,Douaibi Mostafa

**1. Introduction**

* Aperçu du projet :

Le projet vise à créer un système de gestion de la taxe sur les terrains non bâtis au Maroc (TNB). Cette taxe est calculée annuellement en fonction de la catégorie du terrain, son taux associé, et sa surface. Le système permettra de rechercher les terrains d'un redevable par son numéro d'identification national (CIN), d'afficher l'historique des taxes annuelles, de trouver le taux de terrain par catégorie, et enfin, de calculer la taxe TNB.

**Catégories de Terrains et Taux Associés :**

Les catégories de terrains, telles que Villa, Maison, Appartement, sont préalablement définies dans le système.

Chaque catégorie a un taux associé, exprimé en dirhams par mètre carré (DH/m²), qui peut être modifié au fil du temps selon les régulations en vigueur.

**Calcul de la Taxe TNB :**

Pour chaque terrain trouvé, le système calcule la taxe TNB pour une année donnée.

Le calcul se fait en multipliant le taux de terrain par la surface du terrain.

Si la taxe n'a pas encore été payée pour l'année en question, elle est ajoutée au montant total dû.

**Recherche des Terrains par CIN :**

L'utilisateur peut entrer le numéro d'identification national (CIN) pour rechercher tous les terrains associés à un redevable spécifique.

Le système renvoie les détails de chaque terrain, y compris la catégorie et la surface.

* Importance de l'architecture microservices :

L'architecture microservices joue un rôle crucial dans le domaine du développement logiciel, offrant plusieurs avantages qui répondent aux défis contemporains liés à la conception, au déploiement et à la maintenance des applications. Voici quelques-unes des raisons qui soulignent l'importance de l'architecture microservices:

**Scalabilité et Agilité :** Les microservices permettent une évolutivité horizontale, où chaque service peut être développé, déployé et mis à l'échelle indépendamment des autres. Cela facilite l'ajout de nouvelles fonctionnalités et la gestion des charges de travail variables, offrant une agilité significative.

**Facilité de Déploiement :** En raison de leur nature indépendante, les microservices peuvent être déployés de manière continue, facilitant la mise en production fréquente et la réduction des risques associés aux mises à jour logicielles.

**Isolation des Fonctionnalités :** Chaque microservice est responsable d'une fonctionnalité spécifique de l'application, favorisant une isolation claire entre les différentes parties du système. Cela facilite le développement, la maintenance et le dépannage.

**Technologies et Langages Divers :** Les équipes de développement peuvent choisir les technologies et les langages de programmation les mieux adaptés à chaque microservice. Cela permet d'utiliser les meilleurs outils pour chaque tâche, plutôt que de s'en tenir à une seule technologie pour l'ensemble de l'application.

**Facilité de Gestion :** En raison de leur taille réduite, les microservices sont plus faciles à comprendre, à gérer et à maintenir. Les équipes peuvent se concentrer sur des aspects spécifiques de l'application sans être submergées par la complexité globale.

**2. Architecture Microservices**

* Architecture :

Le projet de gestion de la taxe TNB adopte une architecture microservices avec Spring Boot pour la partie back-end et Angular pour le front-end. Le back-end utilise des composants microservices pour des fonctionnalités spécifiques telles que l'authentification, taxe-service et autre. La partie front-end, développée avec Angular, offre une interface utilisateur moderne pour une interaction intuitive avec le système. Cette approche combine la robustesse de Spring Boot et la flexibilité d'Angular pour créer un système efficace, modulaire et réactif répondant aux besoins de l'utilisation.

**Architecture Globale du Projet :**

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

Description générée automatiquement

* Description des services

**Config-server :**

Le projet config-server structuré en serveur de configuration.Le dossier "configurations" contient les fichiers de configuration pour les microservices associés. En résumé, c'est un projet bien organisé en serveur de configuration, prêt pour la gestion et le déploiement des microservices.

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

Description générée automatiquement

**Discovery :**

Le projet discovery plus précisément le serveur Eureka. La classe principale, "DiscoveryApplication.java", et le dossier "application.yml" contiennent les configurations cruciales pour les microservices associés au projet. Cette capture met en lumière un projet spécifiquement orienté vers la gestion et la découverte des microservices, avec un accent particulier sur le rôle du serveur Eureka.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

**Gateway :**

Le projet "gateway", avec son dossier renfermant le fichier de configuration "application.yml", représente une passerelle essentielle au sein d'une architecture microservices. Un gateway, dans ce contexte, agit comme un point d'entrée centralisé pour le système distribué, facilitant l'acheminement sécurisé des requêtes, la gestion des versions de microservices, la consolidation des réponses, et la sécurisation des communications. Le fichier "application.yml" dans ce dossier joue un rôle crucial en permettant la configuration flexible et spécifique au fonctionnement du gateway, ajustant son comportement en fonction des besoins du système.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

**Auth-service :**

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement

Le microservice "auth-service" démontre une organisation méthodique dans le répertoire "src/main/java", mettant en lumière une structure logique du projet. La présence de la classe "TokenService" dans le dossier "service" suggère une gestion spécifique des jetons d'authentification, avec probablement l'utilisation de JSON Web Tokens (JWT). Les JWT, en tant que méthode, renforcent la sécurité en facilitant la création, la validation et le rafraîchissement sécurisés des jetons d'authentification. En résumé, le microservice "auth-service" est astucieusement structuré, mettant l'accent sur une gestion sécurisée des jetons d'authentification via la classe dédiée "TokenService".

**Taxe-service :**

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquementUne image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Le microservice "taxe-service".Les dossiers, tels que "controller", "feign", et "model", sont organisés par fonction, indiquant clairement leurs rôles spécifiques dans le projet. Les dossiers "repository" et "service" révèlent des fonctions liées à l'accès aux données et à la prestation de services aux contrôleurs. Les différentes fonctions importantes du projet comme le calcul de la taxe et les catégories de terrain son bien structuré.

* Mécanismes de communication :

Les protocoles HTTP/RESTful jouent un rôle central dans l'architecture moderne des services web, offrant une approche standardisée et flexible pour la communication entre applications. Voici une élaboration sur ces protocoles :

HTTP (Hypertext Transfer Protocol) : C'est le protocole de communication de base sur le World Wide Web. Il fonctionne sur un modèle de requête-réponse, où un client envoie une requête à un serveur, et le serveur renvoie une réponse. Les méthodes couramment utilisées incluent GET pour récupérer des données, POST pour soumettre des données, PUT pour mettre à jour des données existantes, et DELETE pour supprimer des données.

REST (Representational State Transfer) : REST est un style d'architecture qui repose sur les principes fondamentaux de l'HTTP. Il propose une approche uniforme pour l'accès aux ressources via des URI (Uniform Resource Identifiers). Les services RESTful sont construits autour des opérations standard HTTP, en utilisant souvent les méthodes mentionnées ci-dessus. Les données sont généralement échangées au format JSON ou XML, offrant une flexibilité dans la représentation des informations.

La popularité des services HTTP/RESTful découle de leur simplicité, de leur extensibilité et de leur conformité aux principes de l'architecture web. Ils offrent une approche standardisée pour la communication entre services, favorisant ainsi l'interopérabilité et la facilité d'intégration. De plus, l'utilisation du format JSON ou XML pour la représentation des données permet une flexibilité dans le transfert d'informations structurées.

1. **Conception des Microservices**

La conception des microservices pour le projet TNB implique la division des fonctionnalités en trois microservices distincts : auth-service, taxe-service. Chaque microservice est conçu pour répondre à des besoins spécifiques, tout en interagissant avec les autres pour former une application cohérente de gestion de la taxe sur les terrains non bâtis.

**Microservice auth-service (Authentification et Autorisation) :**

* **Responsabilités :**
* **Authentification et Autorisation :**

Gère l'authentification des utilisateurs du système TNB.

Fournit des tokens JWT pour valider l'identité des utilisateurs et garantir l'accès sécurisé aux autres microservices.

* **Gestion des Utilisateurs :**

Stocke et gère les informations d'identification des redevables.

Implémente des mécanismes de gestion des autorisations d'accès aux ressources du système TNB.

* **Architecture et Technologies :**

**Framework :** Utilise un framework léger et adapté à la gestion d'authentification, comme Spring Boot (Java).

**Stockage :** Utilise une base de données (PostgreSQL) pour stocker les informations des utilisateurs et les tokens JWT.

**Sécurité :** Implémente des protocoles de sécurité robustes, tels que OAuth 2.0, pour assurer la confidentialité et l'authenticité.

**Microservice taxe-service (Gestion des Terrains et Calcul de la Taxe) :**

* **Responsabilités :**
* **Gestion des Terrains :**

Permet la recherche des terrains d'un redevable par son CIN.

Maintient l'historique des taxes annuelles de chaque terrain.

Calcul de la Taxe TNB :

Calcule la taxe TNB en fonction de la catégorie du terrain, son taux associé, et sa surface.

Gère le suivi des paiements de taxe pour éviter les doubles paiements.

* **Architecture et Technologies :**

**Framework :** Utilise un framework robuste pour le développement de l'API, tel que Spring Boot (Java) avec ces controller.

**Stockage :** Utilise une base de données pour stocker les détails des terrains, les historiques de taxes et les paiements.

**Communication :** Interagit avec auth-service pour garantir l'authentification avant l'accès aux données sensibles.

La conception des microservices pour le projet TNB offre une approche modulaire et scalable pour la gestion de la taxe sur les terrains non bâtis au Maroc. Le microservice auth-service assure une authentification sécurisée et une gestion efficace des autorisations, renforçant ainsi la sécurité globale du système. D'autre part, le microservice taxe-service gère de manière optimale la recherche des terrains, le calcul de la taxe TNB, et maintient un suivi précis. L'utilisation de frameworks modernes comme Spring Boot et une architecture basée sur des microservices facilitent la maintenance et l'évolution du système, tout en garantissant une performance optimale. Ensemble, ces microservices contribuent à la création d'une application robuste et évolutive pour la gestion efficace de la taxe sur les terrains non bâtis.

1. **Conteneurisation avec Docker**

* Implémentation et avantages :

La conteneurisation avec Docker simplifie le déploiement des applications en empaquetant tous les éléments nécessaires dans des conteneurs légers et autonomes. L'implémentation implique la création de conteneurs à l'aide de Dockerfiles, la gestion des images dans des registres comme Docker Hub, et l'utilisation de Docker Compose pour orchestrer des applications multi-conteneurs. Les avantages comprennent la portabilité, l'isolation, la rapidité, la facilité de déploiement, l'évolutivité, et la gestion efficace des dépendances. En somme, Docker offre une approche efficace et standardisée pour relever les défis du déploiement logiciel.

**Dockerfile(taxe-service) :**

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

Description générée automatiquement

**Dockerfile(auth-service) :**

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

Description générée automatiquement

**Dockerfile(discovery) :**

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

Description générée automatiquement

**Dockerfile(config-server) :**

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

Description générée automatiquement

**Dockerfile(config-server) :**

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

Description générée automatiquement

* Avantages de la Conteneurisation avec Docker :

Portabilité : Les conteneurs Docker encapsulent une application et ses dépendances, garantissant une portabilité maximale entre différents environnements. Un conteneur Docker fonctionnera de manière cohérente quel que soit l'endroit où il est déployé.

Isolation : Les conteneurs offrent une isolation légère entre les applications. Chaque conteneur a son propre système de fichiers, ses processus et son espace réseau, assurant l'isolation des dépendances et des configurations.

Rapidité : Les conteneurs partagent le noyau du système d'exploitation hôte, ce qui les rend plus légers et plus rapides à démarrer par rapport aux machines virtuelles.

Facilité de Déploiement : Les conteneurs peuvent être déployés rapidement et facilement, réduisant le temps nécessaire pour mettre en service une application. Ils facilitent également le déploiement sur des environnements cloud.

Évolutivité : La conteneurisation permet l'évolutivité horizontale, où de multiples instances de conteneurs peuvent être déployées et gérées de manière coordonnée pour répondre à la demande croissante.

Gestion des Dépendances : Les dépendances de l'application sont encapsulées dans le conteneur, éliminant ainsi les conflits potentiels avec d'autres applications sur le même système.

En résumé, la conteneurisation avec Docker simplifie le processus de développement, de déploiement et de gestion des applications, offrant une solution puissante pour répondre aux exigences modernes en matière de déploiement logiciel.

1. **CI/CD avec Jenkins**

* Processus et configuration

L'intégration continue (CI) et le déploiement continu (CD) avec Jenkins sont des pratiques essentielles dans le développement logiciel moderne. Voici une élaboration sur le processus et la configuration de CI/CD avec Jenkins :

**Processus :**

Intégration Continue (CI) :

Les développeurs soumettent leur code à un référentiel (comme Git).

Jenkins surveille en permanence le référentiel pour détecter les modifications.

Dès qu'une modification est détectée, Jenkins déclenche un processus de construction.

Construction (Build) :

Jenkins récupère le code source depuis le référentiel.

Il exécute un ensemble de commandes définies dans un fichier de configuration (par exemple, un fichier Maven pour un projet Java).

Jenkins compile le code, exécute des tests unitaires et génère un artefact (comme un fichier JAR ou WAR).

Tests Automatisés :

Jenkins peut être configuré pour exécuter des tests automatisés après la construction.

Les résultats des tests sont enregistrés et présentés dans l'interface Jenkins.

Déploiement Continu (CD) :

Si les tests réussissent, Jenkins peut déclencher le déploiement automatique vers un environnement de test.

Si le déploiement sur l'environnement de test réussit, Jenkins peut déclencher le déploiement sur des environnements de production successifs.

**Configuration :**

Création d'un Projet Jenkins :

Créer un nouveau projet Jenkins pour surveiller le référentiel (Freestyle project ou Pipeline).

Configuration du Référentiel :

Lier Jenkins à votre référentiel (Git, SVN, etc.).

Configurer les déclencheurs pour surveiller les modifications du référentiel.

Configuration de la Construction :

Définir les étapes de la construction dans le fichier de configuration (Jenkinsfile pour les pipelines).

Configurer les dépendances, les outils nécessaires, et les actions post-construction.

Configuration des Tests Automatisés :

Intégrer les tests automatisés dans le processus de construction.

Configurer les rapports de tests pour une analyse ultérieure.

Configuration du Déploiement :

Configurer les étapes de déploiement dans le fichier Jenkinsfile (pour les pipelines) ou dans la configuration du projet.

Paramétrer les environnements de déploiement.

L'approche CI/CD avec Jenkins automatise le processus de développement, améliorant la qualité du code et la rapidité des déploiements. Cette configuration garantit une intégration transparente et une livraison continue des applications.

Une image contenant texte, capture d’écran, nombre, logiciel

Description générée automatiquement

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

Description générée automatiquement

Une image contenant texte, capture d’écran

Description générée automatiquement

Une image contenant texte, capture d’écran, document

Description générée automatiquement Une image contenant texte, capture d’écran, document

Description générée automatiquement Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

Le fichier de pipeline Jenkins, aussi appelé Jenkinsfile, est une composante essentielle du processus d'automatisation du déploiement continu. Il offre une approche déclarative pour définir et orchestrer les différentes phases du pipeline, du téléchargement du code source à son déploiement sur l'environnement de production. Typiquement écrit en syntaxe Groovy, ce script détaille les étapes telles que la construction, les tests, et le déploiement, permettant à Jenkins d'automatiser le processus d'intégration continue. Les stages du pipeline, tels que "Checkout", "Build", "Test" et "Deploy", organisent de manière logique les différentes actions à effectuer. En intégrant des outils comme Maven pour la construction et des mécanismes de notification, le fichier de pipeline devient un pilier central pour garantir un flux de travail fluide, fiable et reproductible tout au long du cycle de vie du développement logiciel.

1. **Déploiement Automatique :**

Le déploiement automatique avec Ngrok simplifie la mise en production des applications en développement, offrant un accès en ligne pour des tests collaboratifs en temps réel. Cette approche rapide améliore l'efficacité du développement, permettant des retours immédiats et des tests approfondis avant le déploiement sur l'environnement de production. Ngrok crée un tunnel sécurisé vers le service local, rendant l'application accessible en ligne pour les tests et la validation en temps réel. Cette approche facilite la collaboration entre les membres de l'équipe de développement et les parties prenantes externes, offrant une visibilité immédiate sur les changements apportés. En résumé, le déploiement automatique avec Ngrok offre une solution agile, collaborative et efficiente pour la mise en production des applications.

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

Description générée automatiquement Une image contenant texte, logiciel, Icône d’ordinateur, Page web

Description générée automatiquement

1. **Intégration de SonarQube**

* Configuration et bénéfices pour la qualité du code

L'intégration de SonarQube offre une amélioration significative de la qualité du code en identifiant rapidement les erreurs, en maintenant la cohérence, en renforçant la sécurité, et en optimisant les performances du code source. En configurant SonarQube dans le pipeline CI/CD, les analyses continues du code deviennent une pratique régulière, assurant une détection précoce des vulnérabilités et des points d'amélioration. Les bénéfices se traduisent par une maintenance efficace, des cycles de développement plus rapides, et une qualité logicielle globalement renforcée.

Détection Précoce des Problèmes : SonarQube identifie rapidement les problèmes potentiels dans le code source dès le stade de développement. Cela permet aux développeurs de corriger les erreurs au fur et à mesure qu'elles surviennent, évitant ainsi l'accumulation de défauts.

Maintien de la Cohérence du Code : En appliquant des règles prédéfinies, SonarQube assure la cohérence du code au sein d'un projet. Cela garantit que toutes les parties du code suivent des normes de codage définies, facilitant ainsi la maintenance et la compréhension du code.

L’intégration de SonarQube, associée à une configuration adaptée, apporte des bénéfices significatifs en termes de qualité du code, de sécurité et d'efficacité du processus de développement. C'est un outil essentiel pour les équipes de développement cherchant à maintenir des normes élevées et à assurer la robustesse de leurs applications.

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

Description générée automatiquement

Une image contenant texte, capture d’écran

Description générée automatiquement

1. **Conclusion**

* Résumé des accomplissements :

Le projet de mise en place d'un système de gestion de la taxe sur les terrains non bâtis (TNB) au Maroc a abouti à d'importants accomplissements. En utilisant une architecture de microservices avec Angular pour le front-end, Spring Boot pour le back-end, et PostgreSQL comme base de données, le projet offre une solution robuste et évolutive. Les microservices tnb-auth, tnb-core et tnb-analytics ont été conçus pour répondre à des besoins spécifiques, favorisant la scalabilité, la flexibilité et la maintenance indépendante.

L'authentification sécurisée via tnb-auth assure la protection des données, tandis que tnb-core gère efficacement la recherche des terrains, le calcul de la taxe TNB, et le suivi des paiements. En parallèle, tnb-analytics offre des analyses approfondies et des rapports pour faciliter la prise de décision.

De plus, l'adoption de Jenkins pour l'intégration continue et de Docker pour la conteneurisation a joué un rôle crucial dans l'automatisation et la gestion du cycle de vie du développement. Jenkins a permis la mise en place de pipelines d'intégration continue, assurant des déploiements réguliers et fiables des microservices. Cette automatisation a considérablement réduit les erreurs humaines potentielles, accélérant le processus de développement.

Le déploiement automatique avec Ngrok a simplifié la mise en production, améliorant l'efficacité du développement et permettant des tests collaboratifs en temps réel. L'intégration de SonarQube garantit la qualité du code en détectant les bogues, les défauts de conception, et les vulnérabilités.

En conclusion, le projet TNB représente une solution complète et moderne pour la gestion de la taxe sur les terrains non bâtis au Maroc.

* Perspectives futures :

Les perspectives du projet TNB s'orientent vers une amélioration continue et une adaptation proactive aux évolutions du paysage fiscal au Maroc. En mettant l'accent sur l'évolutivité fonctionnelle, le système pourrait intégrer des fonctionnalités avancées telles que des outils de visualisation sophistiqués et des analyses prédictives, améliorant ainsi l'expérience des utilisateurs.

L'optimisation continue des performances assurera une réponse rapide aux besoins croissants, tandis que le renforcement des mesures de sécurité garantira la protection des données sensibles. L'internationalisation pourrait être envisagée pour élargir l'applicabilité du système à d'autres contextes législatifs. En investissant dans la formation et le support utilisateur, le projet favorisera une utilisation optimale du système. Enfin, rester vigilant face aux évolutions législatives permettra au TNB de demeurer conforme aux réglementations en constante évolution, assurant ainsi sa pertinence et son efficacité à long terme.