



**ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR**

Membre de

HONORIS UNITED UNIVERSITIES

Rapport Projet : Architecture des composants d'entreprise

Realiser par:

BELAMOUD Mariam

Plan:

1. Introduction

- Aperçu du projet
- Importance de l'architecture microservices

2. Architecture Microservices

- Architecture
- Description des services
- Mécanismes de communication

3. Conception des Microservices

- Approche de conception pour chaque service

4. Conteneurisation avec Docker

- Implémentation et avantages

5. CI/CD avec Jenkins

- Processus et configuration

6. Déploiement Automatique

7. Intégration de SonarQube

- Configuration et bénéfices pour la qualité du code

8. Conclusion

- Résumé des accomplissements

1. Introduction

- Aperçu du projet

Ce projet vise à développer un système automatisé pour gérer la taxe TNB au Maroc, en mettant l'accent sur une application capable de calculer cette taxe selon différentes catégories de terrains. Il a pour objectifs de faciliter la recherche et le suivi des informations fiscales liées aux individus (via leur CIN) et aux terrains spécifiques, tout en automatisant et simplifiant le processus de calcul pour les utilisateurs et les autorités fiscales. Le système se base sur une classification des terrains (Villa, Maison, Appartement, etc.), la gestion des taux de taxe par mètre carré, et des fonctionnalités de recherche par CIN ou catégorie de terrain pour un calcul précis de la taxe.

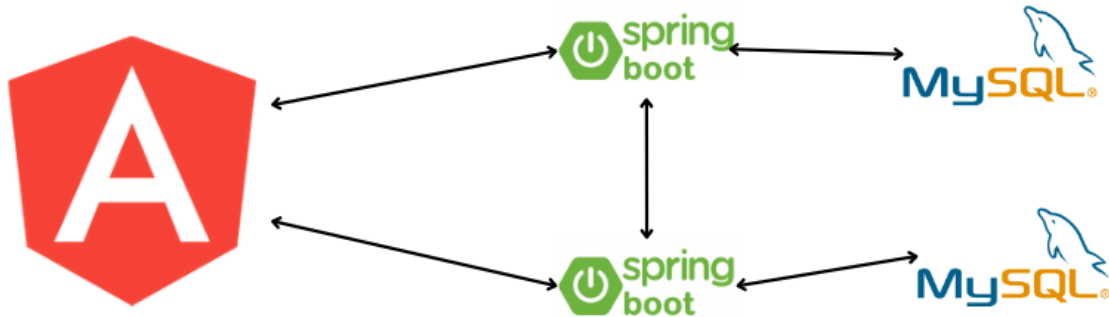
Sur le plan technique, le projet adopte une architecture basée sur des microservices pour l'authentification et la gestion fiscale, tout en employant Docker pour la containerisation des services, garantissant ainsi leur portabilité et isolation. L'intégration continue avec Jenkins facilitera le déploiement et les tests automatiques, tandis que l'utilisation de SonarQube permettra de maintenir un haut niveau de qualité et de sécurité du code. En termes d'implémentation, les utilisateurs devront fournir le CIN, l'année et des informations détaillées sur le terrain pour que le système calcule efficacement la taxe TNB selon un algorithme spécifique, tout en assurant la sécurité des données et l'authentification des utilisateurs.

- Importance de l'architecture microservices

L'importance de l'architecture microservices dans le développement logiciel s'est accrue considérablement, principalement grâce aux multiples avantages qu'elle présente par rapport aux architectures monolithiques classiques.

2. Architecture Microservices

- Architecture



- Description des services

Microservice d'Authentification

Fonction Principale :

Le microservice d'authentification est conçu pour gérer tous les aspects de l'identification et de la validation des utilisateurs. Il sert de porte d'entrée sécurisée pour l'application, assurant que seuls les utilisateurs autorisés peuvent accéder aux fonctionnalités et données sensibles.

Caractéristiques Clés :

Gestion des Identifiants : Permet aux utilisateurs de s'inscrire, de se connecter, et gère les informations d'identification comme les noms d'utilisateur et les mots de passe.

Authentification : Vérifie les identifiants des utilisateurs, souvent via des méthodes comme les mots de passe, les jetons d'authentification, ou l'authentification multi-facteurs.

Gestion des Sessions : Maintient et suit les sessions utilisateur actives, en s'assurant que les utilisateurs sont déconnectés après une période d'inactivité ou lorsqu'ils le demandent explicitement.

Sécurité : Implémente des protocoles de sécurité robustes pour protéger les données d'identification des utilisateurs contre les accès non autorisés ou les fuites.

Intégration avec d'Autres Services : Fournit des jetons ou des clés d'authentification aux autres microservices pour valider les demandes des utilisateurs.

Microservice de Gestion Fiscale (Taxe Management)

Fonction Principale :

Ce microservice est responsable de la gestion de tous les aspects liés aux taxes et aux obligations fiscales. Il calcule les taxes, suit les paiements, et gère les informations fiscales liées aux utilisateurs ou aux entités.

Caractéristiques Clés :

Calcul de Taxe : Calcule les montants de taxe dus en fonction de divers paramètres comme la catégorie de terrain, la surface, et les taux de taxation applicables.

Intégration avec les Données Immobilières : Interagit avec d'autres services pour obtenir des informations sur les propriétés et terrains pour le calcul précis des taxes.

- **Mécanismes de communication**

J'ai structuré les mécanismes de communication de mon architecture en me basant sur le framework Spring Boot pour le développement des microservices, tout en intégrant Spring Cloud pour la gestion des interactions entre ces services. Spring Boot sert de base solide et modulable pour la création de microservices indépendants, tandis que Spring Cloud facilite leur découverte et leur communication. Pour le microservice d'authentification, j'ai mis en œuvre Spring Security pour assurer une gestion sécurisée de l'authentification et de l'autorisation, incorporant JWT (JSON Web Tokens) pour une gestion efficace des sessions utilisateur. Concernant le microservice de gestion fiscale, j'ai utilisé Spring Boot pour élaborer des interfaces RESTful dédiées au traitement des requêtes fiscales. L'intégration entre ces microservices est réalisée grâce à Eureka, un service de découverte de Spring Cloud, qui assure une localisation et une communication aisées entre les services. Cette configuration, alliant Spring Boot et Spring Cloud, permet une communication fluide et performante, autorisant le microservice d'authentification à gérer les identifiants utilisateurs et le microservice de gestion fiscale à calculer et gérer les taxes de manière précise et sécurisée.

3. Conception des Microservices

Dans l'architecture microservices que j'ai mise en place, j'ai adopté une approche de conception distincte pour chaque service afin de répondre de manière optimale à leurs responsabilités uniques.

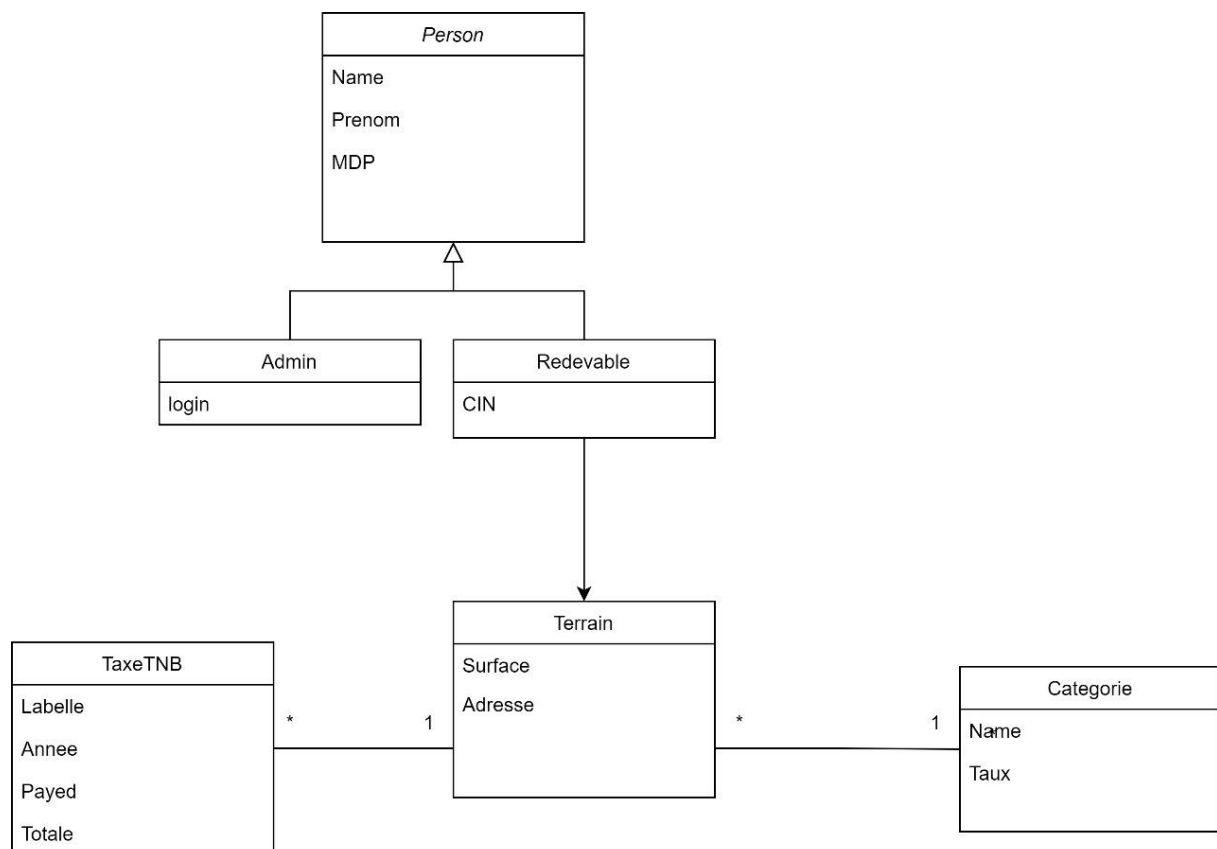
Microservice d'Authentification :

Pour ce service, j'ai mis l'accent sur la sécurité et la gestion des identités. J'ai implémenté des fonctionnalités d'authentification avancées, y compris la gestion des sessions et des jetons d'accès. La conception s'appuie sur des principes de sécurité robustes et des stratégies d'authentification multi-niveaux pour assurer la protection des données utilisateur. Les endpoints API sont soigneusement développés pour une interaction sécurisée avec le frontend, garantissant un contrôle d'accès strict aux fonctionnalités sensibles.

Microservice de Gestion Fiscale :

Ce service est axé sur le calcul et la gestion des données fiscales. J'ai conçu des algorithmes spécifiques pour calculer précisément les taxes selon divers critères et normes réglementaires. Ce service intègre aussi des fonctionnalités pour la gestion des comptes fiscaux des

utilisateurs. Les API sont conçues pour une récupération efficace et sécurisée des informations fiscales et assurent une intégration transparente avec d'autres parties de l'application, telles que le service d'authentification et le frontend.



4. Conteneurisation avec Docker

La conteneurisation avec Docker est une pratique essentielle dans le déploiement d'applications modernes, qui encapsule les applications dans des environnements isolés appelés conteneurs. Ces conteneurs intègrent non seulement l'application et son code, mais aussi toutes les dépendances nécessaires, assurant une cohérence entre les environnements de développement, de test et de production. L'implémentation commence par l'installation de Docker et la création d'un Dockerfile, qui définit les paramètres de l'application et ses dépendances. Avec la commande `docker build`, une image Docker est construite, servant de base pour exécuter des conteneurs indépendants de l'environnement sous-jacent.

- **Implémentation et avantages**

Premier Microservice (Authentification) :













Le microservice d'authentification encapsule la logique d'authentification des utilisateurs et la gestion des sessions. Il est conteneurisé avec Docker, où un Dockerfile spécifique définit toutes les dépendances et configurations nécessaires, assurant ainsi une implémentation cohérente et sécurisée.

Deuxième Microservice (Gestion Fiscale) :

Pour la gestion fiscale, un deuxième microservice est isolé dans son propre conteneur Docker. Son Dockerfile dédié établit l'environnement d'exécution, incluant les outils spécifiques au calcul et à la gestion des taxes, tout en maintenant l'indépendance opérationnelle.

Frontend (Angular) :

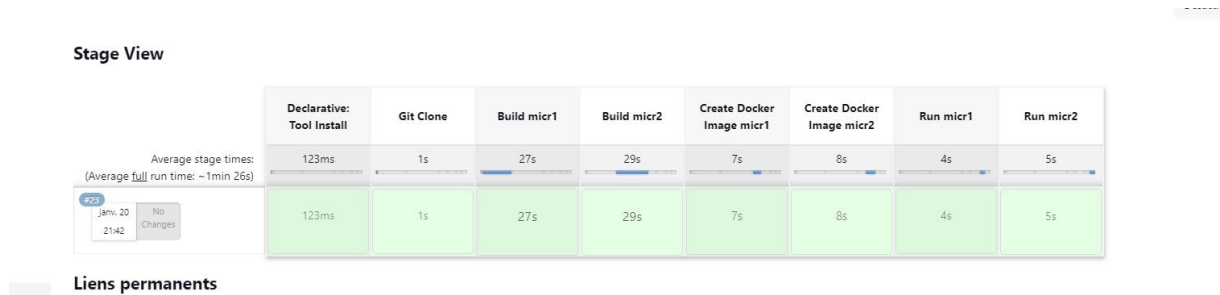
Le frontend de l'application, construit avec Angular, bénéficie aussi de la conteneurisation. Le Dockerfile correspondant contient les étapes pour assembler un conteneur Docker autonome, qui sert à déployer l'interface utilisateur Angular. Cela garantit que le frontend peut être exécuté dans n'importe quel environnement avec une grande facilité et sans nécessiter de configuration spécifique de l'hôte.

<input type="checkbox"/>	 auth a79699558e2a 	auth	Exited (1)	8081:8081 	22 minutes ago  	
<input type="checkbox"/>	 taxe 545b57d4f142 	taxe	Exited (1)	8082:8082 	6 minutes ago  	

5. CI/CD avec Jenkins

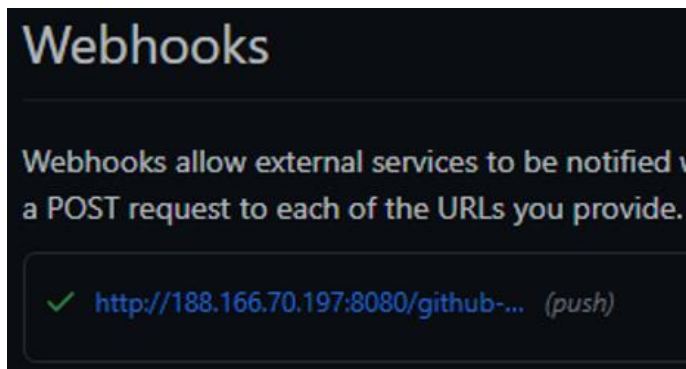
```
jenkins X
C: > Users > mariam > Desktop > jenkins

1 pipeline {
2   agent any
3   tools {
4     maven 'maven'
5   }
6
7   stages {
8     stage('Git Clone') {
9       steps {
10        script {
11          checkout([$class: 'GitSCM', branches: [[name: 'main']], userRemoteConfigs: [[url: 'https://github.com/Mariam81/ACE-Projet.git']]])
12        }
13      }
14    }
15
16    stage('Build micr1'){
17      steps {
18        script {
19          dir('C:\\Users\\mariam\\Documents\\SIIR\\ACE\\C1\\taxemanagment-spring') {
20            bat 'mvn clean install'
21          }
22        }
23      }
24
25      stage('Build micr2'){
26        steps {
27          script {
28            dir('C:\\Users\\mariam\\Documents\\SIIR\\ACE\\C1\\auth-spring') {
29              bat 'mvn clean install'
30            }
31          }
32        }
33      }
34
35      stage('Create Docker Image micr1') {
36        steps {
37          script {
38            // Navigate into the project directory before running Maven commands
39            dir('C:\\Users\\mariam\\Documents\\SIIR\\ACE\\C1\\taxemanagment-spring') {
40              bat 'docker build -t tax .'
41            }
42          }
43        }
44      }
45
46      stage('Create Docker Image micr2') {
47        steps {
48          script {
49            // Navigate into the project directory before running Maven commands
50            dir('C:\\Users\\mariam\\Documents\\SIIR\\ACE\\C1\\auth-spring') {
51              bat 'docker build -t authif .'
52            }
53          }
54        }
55      }
56
57      stage('Run micr1') {
58        steps {
59          script {
60            bat "docker rm -f tax || true"
61            bat "docker run --name tax -d -p 8082:8082 tax"
62          }
63        }
64      }
65
66      stage('Run micr2') {
67        steps {
68          script {
69            bat "docker rm -f authif || true"
70            bat "docker run --name authif -d -p 8081:8081 authif"
71          }
72        }
73      }
74
75      stage('Clean Docker') {
76        steps {
77          script {
78            bat "docker rm -f tax || true"
79            bat "docker rm -f authif || true"
80          }
81        }
82      }
83    }
84  }
85}
```

6. Déploiement Automatique

Un serveur privé virtuel (VPS) hébergé chez DigitalOcean est configuré pour exécuter Jenkins. Des webhooks ont été établis pour chaque dépôt sur GitHub, permettant une communication automatique et instantanée avec Jenkins à chaque push ou pull request.



Payload URL *

`http://188.166.70.197:8080/github-webhook/`

Content type

`application/json`

Secret

Which events would you like to trigger this webhook?

☒ Just the push event.

☐ Send me **everything**.

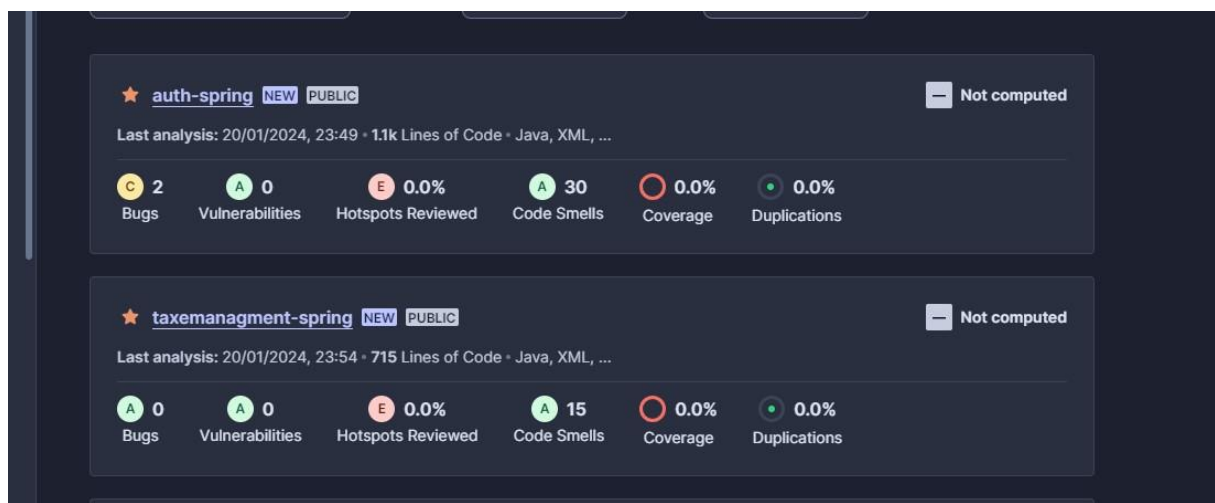
☐ Let me select individual events.

☒ **Active**
We will deliver event details when this hook is triggered.

Update webhook **Delete webhook**

7. Intégration de SonarQube

L'intégration de SonarQube dans le processus de développement est une pratique stratégique pour améliorer la qualité du code. SonarQube est un outil d'analyse statique qui examine le code source pour détecter les vulnérabilités, les bugs et les mauvaises pratiques de codage. Il fournit également des métriques sur la complexité du code, la duplication du code, et le respect des normes de codage.



8. Conclusion

- Résumé des accomplissements

Au cours de ce projet, une série d'accomplissements notables a été réalisée, marquant des avancées significatives dans le développement et la gestion de l'application. Premièrement, l'adoption d'une architecture microservices a permis de créer des services distincts pour l'authentification et la gestion fiscale, ce qui a apporté une modularité, une flexibilité et une facilité de maintenance exceptionnelles. Ensuite, la conteneurisation de l'ensemble de l'application, incluant les microservices et le frontend Angular, a été effectuée avec Docker, garantissant ainsi une portabilité sans faille et une uniformité entre les environnements de développement et de production.

L'intégration et l'automatisation du processus CI/CD ont été réalisées avec succès grâce à Jenkins, optimisant les délais de livraison et renforçant l'efficacité du pipeline de développement. Par ailleurs, l'analyse de la qualité du code a été grandement améliorée par l'intégration de SonarQube, qui a joué un rôle clé dans l'identification et la correction proactive des vulnérabilités, bugs et mauvaises pratiques de codage. Enfin, une infrastructure robuste et réactive a été mise en place grâce à l'utilisation d'un VPS sous DigitalOcean, complétée par la configuration de webhooks GitHub, assurant une intégration fluide et des réponses rapides aux événements du dépôt. Ensemble, ces réalisations représentent une avancée majeure dans le développement de l'application, mettant en place une base solide pour des améliorations et expansions futures.