ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

2023/2024

ARCHITECTURE DES COMPOSANTS D'ENTREPRISE

Ingénierie Informatique et Réseaux

MIAGE

Réalisé par:
Chaima Mehrach

Encadré par :
Mr LACHGAR Mohamed

Chapitre 1

INTRODUCTION

Aperçu du Projet

Dans le contexte de la rapide évolution des technologies de l'information, l'adaptation des architectures logicielles devient impérative pour répondre efficacement aux exigences croissantes des applications.

Le projet actuel, consacré à la gestion de la Taxe sur les Terrains Non Bâtis (TNB) au Maroc, embrasse pleinement cette évolution en adoptant une architecture microservices. Cette approche innovante vise à optimiser la scalabilité, la flexibilité et la maintenabilité du système, créant ainsi un environnement propice à la gestion efficace des complexités inhérentes à la fiscalité foncière.

Au cœur de cette initiative réside la volonté de moderniser et automatiser la gestion de la taxe TNB au Maroc. Ceci se matérialise par l'intégration de fonctionnalités clés, telles que la classification des terrains, la détermination dynamique des taux en fonction de la catégorie et de l'évolution temporelle, la recherche ciblée des terrains d'un redevable par le biais de sa Carte d'Identité Nationale (CIN), et enfin, le calcul précis de la taxe TNB annuelle.

Un aspect distinctif de notre approche est la focalisation sur la transparence et l'efficacité, avec une interface utilisateur intuitive permettant aux contribuables de visualiser et de comprendre clairement les détails relatifs à leurs taxes foncières.

Ce système informatisé se positionne comme une solution efficiente, transparente et évolutive, répondant de manière judicieuse aux besoins complexes de la fiscalité liée aux terrains non bâtis au Maroc.

De plus, le développement continu de cette plateforme s'inscrit dans notre engagement envers l'innovation, assurant ainsi sa pertinence à long terme dans un paysage technologique en constante évolution.

Importance de l'architecture microservices

L'architecture microservices, un élément central de l'ingénierie logicielle contemporaine, s'est imposée comme une solution efficace face aux défis des applications modernes. Cette approche novatrice va au-delà des modèles monolithiques traditionnels, offrant une flexibilité et une agilité exceptionnelles pour répondre aux exigences complexes des systèmes d'aujourd'hui. Plongeons dans l'essence cruciale de l'architecture microservices et examinons pourquoi elle est devenue indispensable dans le paysage technologique actuel.

1. Scalabilité Dynamique :

L'architecture microservices brille particulièrement par sa capacité à permettre une scalabilité dynamique. Chaque service peut évoluer indépendamment, garantissant une performance optimale même face à des fluctuations importantes de la charge de travail. Cette agilité opérationnelle constitue un atout essentiel dans un environnement où l'adaptabilité rapide aux évolutions est incontournable.

2. Flexibilité et Agilité :

La flexibilité inhérente à l'architecture microservices la rend particulièrement adaptée aux méthodologies de développement agiles. Chaque microservice peut être développé, testé et déployé de manière autonome, favorisant ainsi une mise en œuvre continue et une réponse agile aux changements requis. Cet aspect devient crucial dans un contexte où la vitesse de développement est synonyme de compétitivité.

3. Isolation des Responsabilités :

La conception modulaire des microservices permet une isolation claire des responsabilités. Chaque microservice est dédié à une fonctionnalité spécifique, facilitant la maintenance, le débogage et la mise à l'échelle. L'impact d'éventuelles erreurs se trouve ainsi limité, préservant la robustesse globale du système.

4. Réutilisabilité :

L'un des avantages significatifs des microservices réside dans leur réutilisabilité, favorisant une utilisation efficace des ressources. Cette caractéristique contribue à la construction d'une bibliothèque de services réutilisables, accélérant le processus de développement et réduisant la duplication des efforts.

5. Indépendance Technologique :

L'architecture microservices offre une liberté d'utilisation de technologies spécifiques à chaque microservice, favorisant ainsi l'indépendance technologique. Chaque service peut être développé avec les outils les mieux adaptés à sa fonction, optimisant ainsi les performances et la productivité sans compromettre la cohérence du système.

6. Facilitation de la Collaboration :

L'architecture microservices encourage la collaboration entre les équipes de développement. En attribuant des microservices spécifiques à des équipes dédiées, cela favorise une responsabilité claire et une collaboration efficace, conduisant à une meilleure productivité et à des cycles de développement plus rapides.

7. Déploiement Indépendant :

Chaque microservice peut être déployé indépendamment des autres, minimisant ainsi les interruptions de service globales. Cette capacité de déploiement indépendant permet aux équipes de publier des mises à jour de manière continue, améliorant la flexibilité et la réactivité aux changements.

8. Gestion de la Complexité :

La décomposition des fonctionnalités en microservices plus petits simplifie la gestion de la complexité. Les équipes peuvent se concentrer sur des tâches spécifiques sans se soucier des dépendances excessives, facilitant la compréhension du code, la maintenance et le débogage.

9. Résilience et Tolérance aux Pannes :

L'architecture microservices favorise la résilience du système. En cas de défaillance d'un microservice, les autres peuvent continuer à fonctionner, minimisant ainsi l'impact des pannes et améliorant la disponibilité globale du système.

10. Évolutivité Indépendante :

Chaque microservice peut évoluer indépendamment des autres. Cela signifie qu'une fonctionnalité spécifique peut être mise à jour ou étendue sans affecter l'ensemble du système, offrant une flexibilité essentielle pour répondre aux évolutions des besoins métier.

En somme, l'adoption de l'architecture microservices représente une avancée significative pour relever les défis complexes du développement logiciel contemporain. Ses avantages en termes de scalabilité dynamique, flexibilité, isolation des responsabilités, réutilisabilité et indépendance technologique en font une approche incontournable pour les projets d'envergure, offrant une efficacité, une maintenance et une évolutivité optimales. Adopter cette architecture, c'est embrasser une nouvelle ère de développement logiciel, où l'innovation et la performance convergent harmonieusement.

Chapitre 2

ARCHITECTURE MICROSERVICES

Architecture

L'architecture microservices adoptée pour ce projet repose sur une approche modulaire et distribuée, où chaque composant du système est traité comme un service indépendant. Cette décomposition permet une gestion plus efficace des fonctionnalités, favorisant la flexibilité, la scalabilité et la maintenance.

Composants Principaux :

Composant	Desccription	Rôle
Eureka Server	Comme un registre des services. Il permet à chaque microservice de s'enregistrer, facilitant ainsi la découverte dynamique des services disponibles dans notre application.	Fournit une coordination centralisée pour la localisation des services.
Configuration du Serveur	Gère la configuration des microservices. Il permet la modification dynamique des paramètres sans nécessiter de redémarrage des services.	Gestion centralisée et dynamique des configurations.
Gateway	Point d'entrée central pour toutes les requêtes client. Il gère la redirection vers les microservices appropriés en fonction de la demande.	Routage des requêtes clients vers les services adéquats.

Composant	Desccription	Rôle
test	Responsables de la gestion des taxes sur les terrains non bâtis. Il peut traiter des fonctionnalités spécifiques, telles que la classification des terrains, le calcul des taxes, et la gestion des historiques.	Gestion des opérations liées aux taxes sur les terrains non bâtis.
Clients Service	Gère les informations relatives aux redevables. Il permet la recherche des terrains d'un redevable par le biais de sa Carte d'Identité Nationale (CIN).	Gestion des données relatives aux redevables et à leurs terrains.

Mécanismes de Communication

La communication entre les services est orchestrée de manière efficace pour garantir la cohérence et la fiabilité du système. Voici quelques éléments clés de cette gestion des communications :

Protocole HTTP/RESTful :

Les services interagissent principalement au moyen de requêtes HTTP RESTful, favorisant un échange de données léger et une intégration flexible.

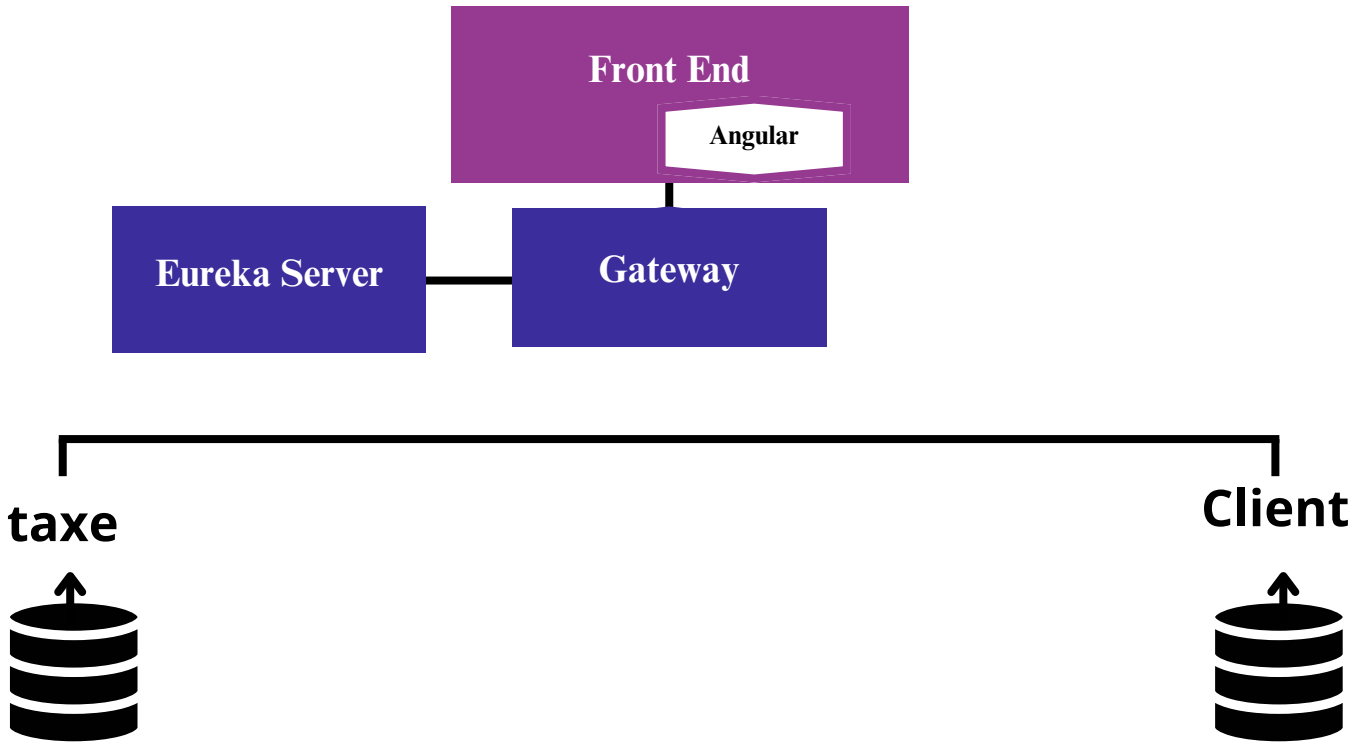
Eureka Server pour la Découverte de Services :

Les microservices s'enregistrent auprès du serveur Eureka. Cette approche facilite la localisation dynamique des services à travers l'ensemble du système, assurant une gestion efficace des dépendances.

Gateway pour le Routage :

Le Gateway assume la responsabilité du routage des requêtes clients vers les services appropriés. Cela garantit une distribution équilibrée des charges et une répartition efficace des tâches, contribuant ainsi à une performance optimale du système.

Cette architecture microservices offre une flexibilité inégalée, permettant une évolutivité facile, une maintenance simplifiée et une gestion efficace des fonctionnalités de notre application dédiée à la Taxe sur les Terrains Non Bâtis au Maroc.



Chapitre 3

CONCEPTION DES MICROSERVICES

Approche de Conception Générale :

1. Microservices Architecture :

-Service d'Authentification (Client) :

- Utilisation de Spring Security pour gérer l'authentification et l'autorisation.
- Stockage sécurisé des informations d'identification.
- Exposition d'une API sécurisée pour le frontend et d'autres services.

- Service de Gestion de Taxe :
  - Définition claire des fonctionnalités liées à la gestion de taxe.
  - Base de données pour stocker les informations fiscales.
  - L'admin peut gérer les taxe et les categories de terrain.

2. Communication entre Microservices :

- Utilisation d'Eureka pour la découverte et l'enregistrement des services.
- Mise en place d'un API Gateway pour acheminer les requêtes du frontend vers les services appropriés.
-

3. Frontend (Angular) :

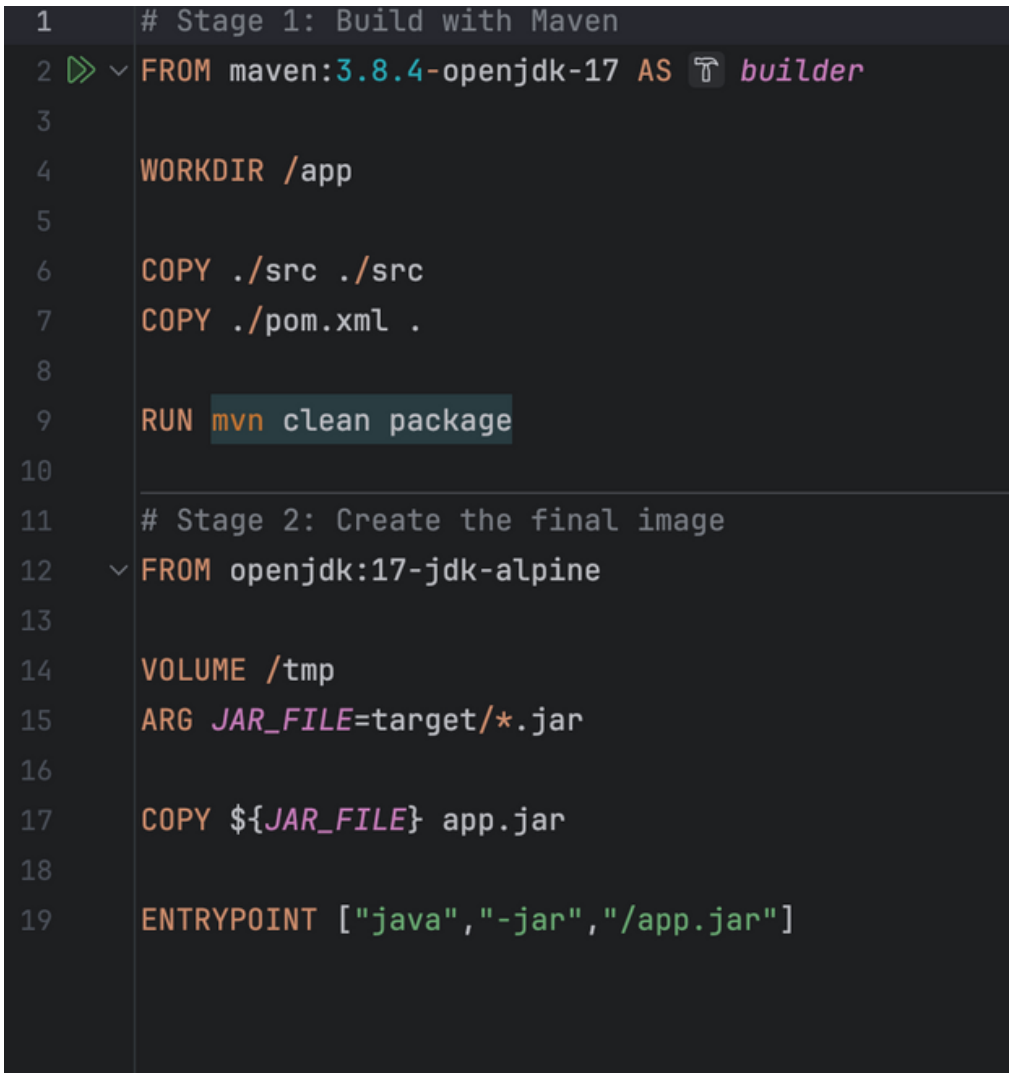
- Développement d'une interface utilisateur réactive pour interagir avec les microservices.
- Consommation des APIs exposées par les services backend.
- Gestion des opérations asynchrones et des erreurs de manière efficace.
- Utilisation de mécanismes de sécurité côté client pour protéger les informations sensibles.

Chapitre 4

CONTENEURISATION AVEC DOCKER

L'utilisation de la conteneurisation à travers Docker est une méthode largement adoptée dans le développement logiciel contemporain. Docker offre une plateforme accessible qui autorise les développeurs à concevoir, déployer et faire fonctionner des applications au sein de conteneurs. Voici une succincte explication de la mise en œuvre de la conteneurisation via Docker ainsi que de ses bénéfices.

Docker File

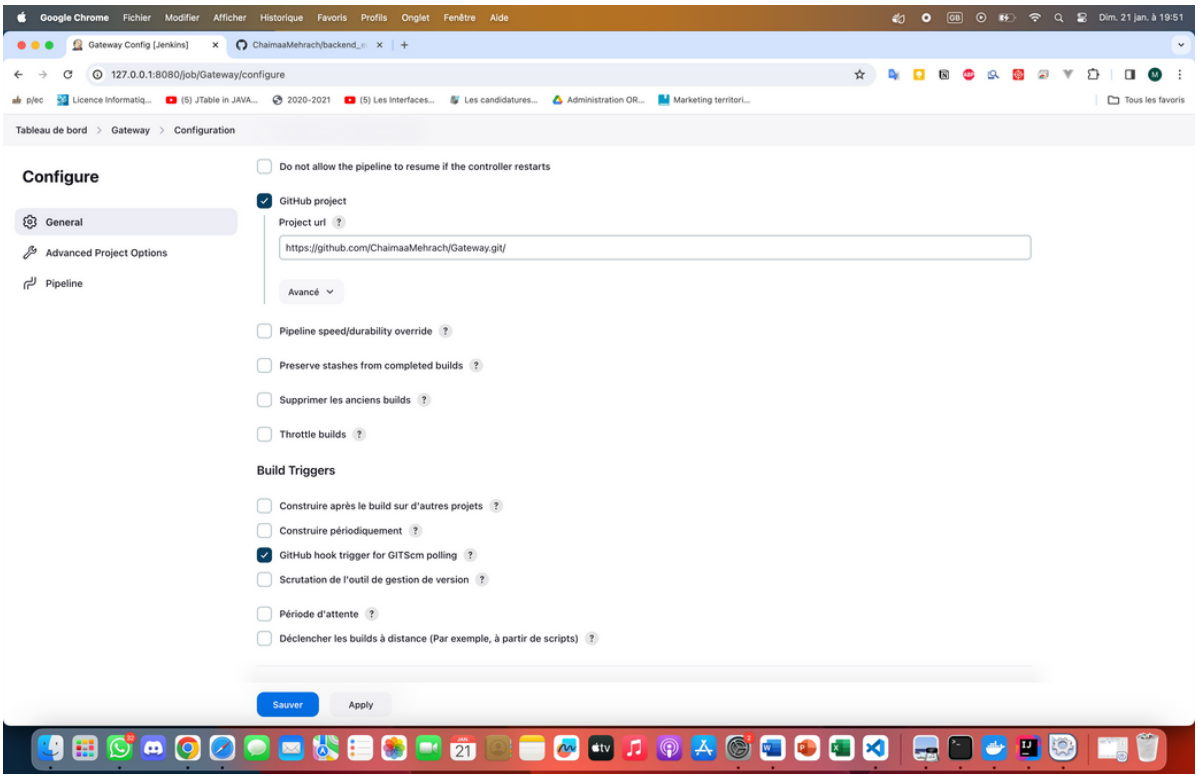


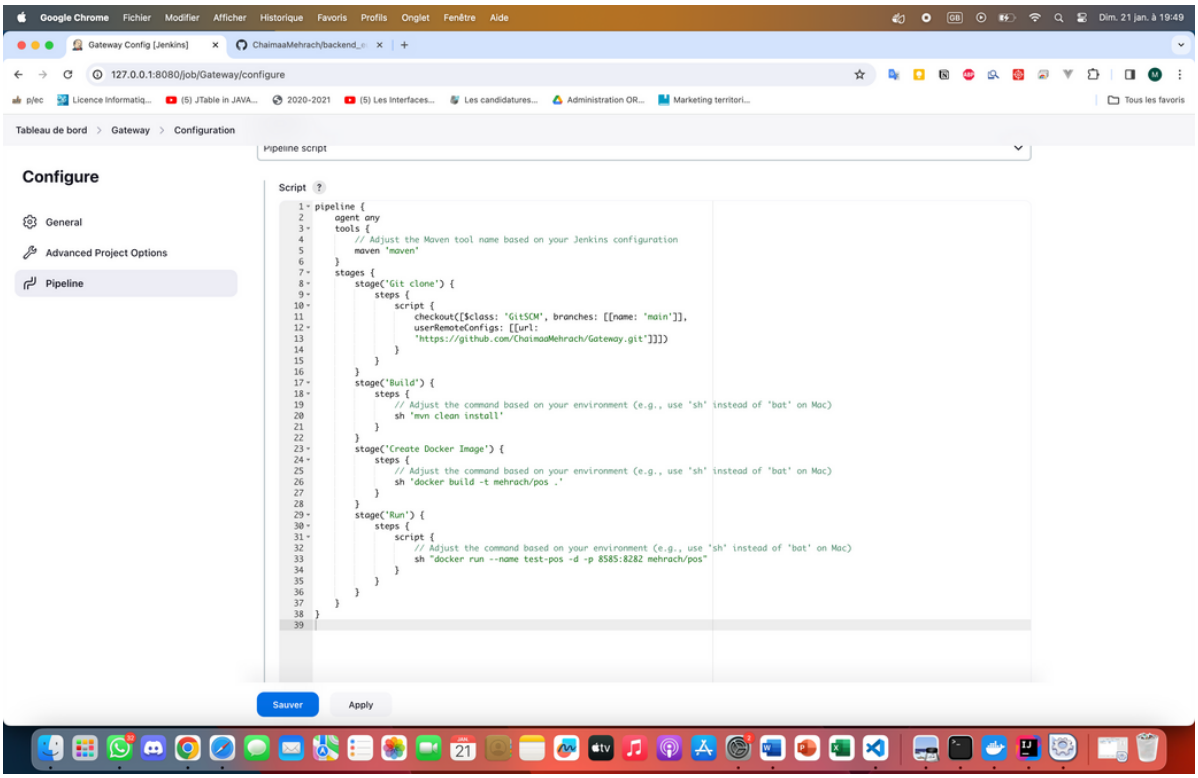
```dockerfile
# Stage 1: Build with Maven
FROM maven:3.8.4-openjdk-17 AS builder

WORKDIR /app

COPY ./src ./src
COPY ./pom.xml .

RUN mvn clean package

# Stage 2: Create the final image
FROM openjdk:17-jdk-alpine

VOLUME /tmp
ARG JAR_FILE=target/*.jar

COPY ${JAR_FILE} app.jar

ENTRYPOINT ["java","-jar","/app.jar"]
```

Isolation Efficient:
Les conteneurs assurent une isolation légère en encapsulant non seulement l'application mais aussi ses dépendances, avec chaque conteneur possédant son propre système de fichiers, processus, et espace mémoire distinct.
Portabilité Unifiée:
Les conteneurs offrent une uniformité opérationnelle sur divers environnements, assurant une cohérence dans le comportement de l'application, qu'elle soit exécutée en local, sur un serveur de développement, ou dans le cloud.
Déploiement Simplifié:
La standardisation des images Docker simplifie grandement le déploiement sur une gamme variée d'infrastructures, allant des serveurs locaux aux clusters Kubernetes et aux services cloud.
Réactivité et Agilité:
Du fait de leur légèreté, les conteneurs démarrent rapidement, facilitant des déploiements et des mises à l'échelle rapides.
Gestion Efficace des Dépendances:
Les dépendances de l'application sont encapsulées au sein du conteneur, éliminant ainsi les conflits et les problèmes associés aux versions divergentes des bibliothèques.
Évolutivité Horizontale Facilitée:
Les applications conteneurisées peuvent être aisément étendues horizontalement par l'ajout ou la suppression d'instances de conteneurs.
Contrôle des Versions Simplifié:
Les images Docker offrent une gestion plus efficace des versions de l'application, facilitant le suivi des modifications et la rétrogradation en cas de besoin.
Écosystème Étendu:
Docker bénéficie d'un écosystème étendu d'outils qui simplifient le cycle de vie du développement, du déploiement, et de la gestion des applications conteneurisées.
Docker File

Chapitre 5

CI/CD AVEC JENKINS


Google Chrome Fichier Modifier Afficher Historique Favoris Profils Onglet Fenêtre Aide
Dim. 21 jan. à 19:51
Gateway Config [Jenkins]
ChaimaaMehrach/backend_e
127.0.0.1:8080/job/Gateway/configure
Tableau de bord > Gateway > Configuration
Configure
General
Advanced Project Options
Pipeline
Do not allow the pipeline to resume if the controller restarts
GitHub project
Project url
https://github.com/ChaimaaMehrach/Gateway.git/
Avancé
Pipeline speed/durability override
Preserve stashes from completed builds
Supprimer les anciens builds
Throttle builds
Build Triggers
Construire après le build sur d'autres projets
Construire périodiquement
GitHub hook trigger for GITScm polling
Scrutation de l'outil de gestion de version
Période d'attente
Déclencher les builds à distance (Par exemple, à partir de scripts)
Sauver
Apply

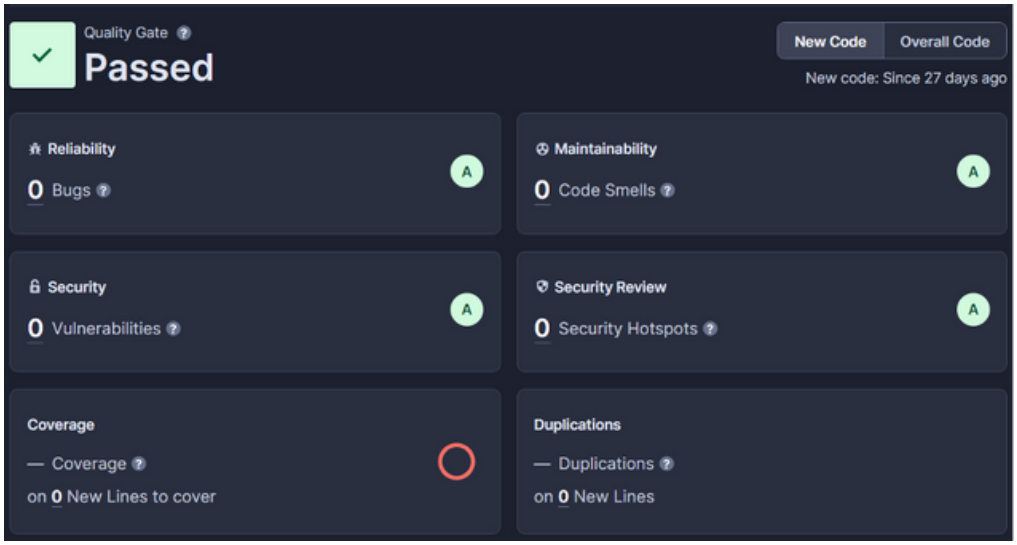

Google Chrome Fichier Modifier Afficher Historique Favoris Profils Onglet Fenêtre Aide
Dim. 21 jan. à 19:49
Gateway Config [Jenkins]
ChaimaaMehrach/backend_e
127.0.0.1:8080/job/Gateway/configure
Tableau de bord > Gateway > Configuration
Configure
General
Advanced Project Options
Pipeline
Pipeline script
Script
pipeline {
agent any
tools {
// Adjust the Maven tool name based on your Jenkins configuration
maven 'maven'
}
stages {
stage('Git clone') {
steps {
script {
checkout([$class: 'GitSCM', branches: [[name: 'main']],
userRemoteConfigs: [[url:
'https://github.com/ChaimaaMehrach/Gateway.git']]])
}
}
}
stage('Build') {
steps {
// Adjust the command based on your environment (e.g., use 'sh' instead of 'bat' on Mac)
sh 'mvn clean install'
}
}
stage('Create Docker Image') {
steps {
// Adjust the command based on your environment (e.g., use 'sh' instead of 'bat' on Mac)
sh 'docker build -t mehrach/pos .'
}
}
stage('Run') {
steps {
script {
// Adjust the command based on your environment (e.g., use 'sh' instead of 'bat' on Mac)
sh "docker run --name test-pos -d -p 8585:8282 mehrach/pos"
}
}
}
}
}
Sauver
Apply

Chapitre 6

SONARQUBE

Intégrer SonarQube à Jenkins représente une stratégie clé pour assurer la qualité du code à toutes les étapes du développement. En paramétrant Jenkins pour déclencher l'analyse de SonarQube à chaque construction, nous instaurons une évaluation continue de la qualité du code. SonarQube identifie promptement des problèmes tels que les bugs, les vulnérabilités, et les violations de normes, offrant aux développeurs la possibilité de résoudre ces problèmes à un stade précoce du cycle de développement. Les métriques fournies par SonarQube, comme le taux de duplication et la complexité cyclomatique, permettent une évaluation approfondie de la qualité du code. L'intégration harmonieuse dans Jenkins garantit la cohérence dans l'application des normes de codage. En unissant Jenkins et SonarQube, nous renforçons la solidité et la fiabilité de notre code, soutenant ainsi une approche d'amélioration continue au sein de notre processus de déploiement.

```
stage('SonarQube') {
    steps {
        script {
            def scannerHome = tool 'sonar-scanner'
            bat "${scannerHome}/bin/sonar-scanner -Dsonar.organization=mbk-moughit -Dsonar.projectKey=mbk-moughit_sonar-jenkins -Dsonar.sources=. -Dsonar.host.url=https://so
        }
    }
}
```



Chapitre 7

CONCLUSION

Résumé des Accomplissements :

Au cours de ce projet, j'ai contribué à la conception et à la mise en œuvre de deux microservices distincts. Le premier microservice était dédié à la gestion des clients, avec une attention particulière portée sur l'implémentation sécurisée de l'authentification grâce à Spring Security. Ceci garantit une protection robuste des données des utilisateurs et une gestion sécurisée des autorisations.

Le second microservice, axé sur la gestion des taxes, a été développé pour fournir des fonctionnalités spécifiques à la manipulation des données fiscales. Cette séparation des préoccupations a permis de créer des services spécialisés, favorisant ainsi la modularité et la maintenabilité du système.

Le frontend, développé en Angular, a consommé les APIs exposées par les microservices, assurant une expérience utilisateur fluide et réactive. L'utilisation d'Eureka et de Gateway a facilité la découverte et la gestion des microservices, renforçant ainsi l'architecture orientée services.

Le recours à HttpRequestFactory a enrichi l'interactivité du système en permettant des communications HTTP flexibles et adaptatives, contribuant ainsi à l'efficacité globale de l'application.

Perspectives Futures :

Pour aller de l'avant, il serait judicieux d'explorer davantage les opportunités d'amélioration et d'optimisation. Voici quelques pistes à considérer pour les perspectives futures :

1. Optimisation de la Sécurité : Continuer à surveiller et à renforcer les mécanismes de sécurité, en s'adaptant aux évolutions des meilleures pratiques et des menaces potentielles.

2. Évolutivité des Microservices :Penser à l'évolutivité des microservices pour répondre aux besoins futurs. Cela pourrait impliquer l'ajout de nouvelles fonctionnalités ou l'optimisation des performances.

3. Amélioration de l'Expérience Utilisateur : Explorer des moyens d'améliorer l'interface utilisateur en tirant parti des dernières fonctionnalités d'Angular et en garantissant une expérience utilisateur intuitive.

4. Surveillance et Gestion des Performances : Mettre en place des mécanismes de surveillance pour évaluer les performances du système en temps réel et identifier les opportunités d'optimisation.

5. Documentation et Formation : Documenter rigoureusement le système et fournir des sessions de formation pour l'équipe, facilitant ainsi la maintenance et l'intégration de nouveaux membres.

En résumé, les accomplissements actuels ont jeté les bases d'une application robuste et modulaire. Les perspectives futures s'orientent vers l'amélioration continue, la flexibilité, et l'adaptation aux besoins changeants du domaine et de l'utilisateur final.