

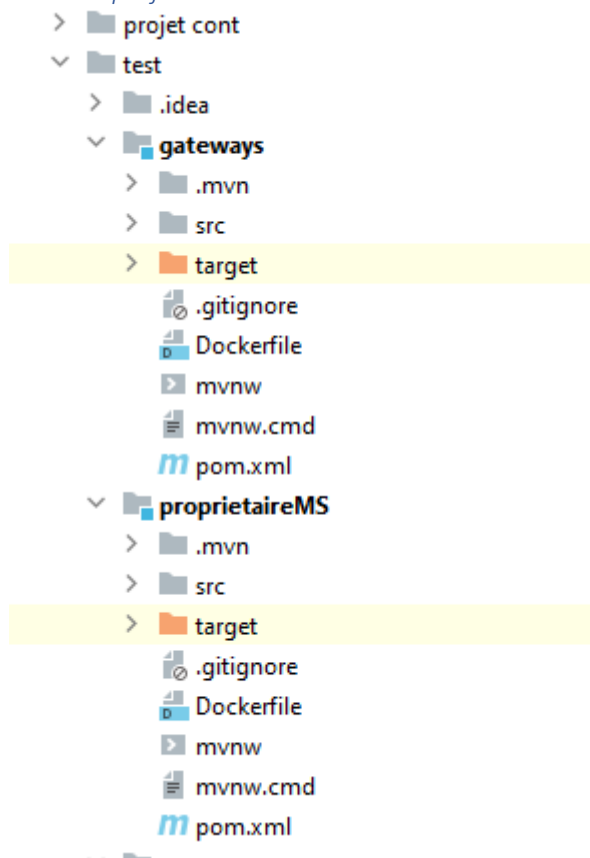
Rapport Projet : Architecture des composants d'entreprise

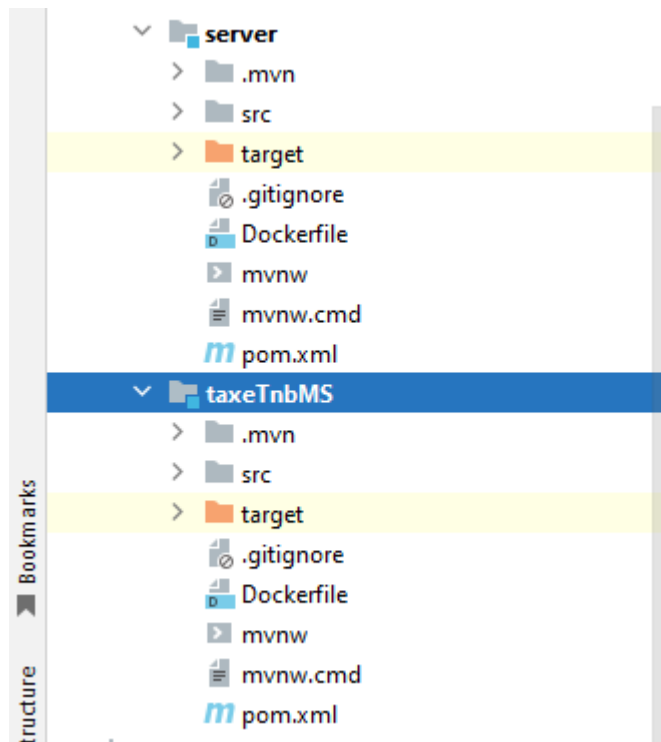
SOUHAIB CHAYMAA

1. Introduction

Aperçu du projet

Structure du projet :





Importance de l'architecture microservices

L'architecture microservices revêt une grande importance dans le domaine du développement logiciel en raison de plusieurs avantages qu'elle offre. Voici quelques-unes des raisons pour lesquelles l'architecture microservices est considérée comme importante :

1. **Évolutivité et Flexibilité** : Les microservices permettent une évolutivité plus aisée de l'application. Chaque service peut être développé, déployé, mis à l'échelle et mis à jour indépendamment des autres. Cela donne une grande flexibilité pour répondre aux besoins changeants et évoluer rapidement.
2. **Facilité de Déploiement** : En raison de leur nature indépendante, les microservices peuvent être déployés de manière séparée. Cela facilite le déploiement continu (continuous deployment) et réduit le risque d'impact sur l'ensemble du système lors de mises à jour.
3. **Gestion Simplifiée** : Chaque microservice est responsable d'une tâche spécifique. Cela simplifie la gestion, le développement et la maintenance, car chaque équipe peut se concentrer sur son propre service sans avoir à se soucier des détails internes des autres.
4. **Technologies Diverses** : Différents microservices peuvent être développés en utilisant des technologies différentes, adaptées à leurs besoins spécifiques. Cela permet de choisir les meilleurs outils pour chaque tâche, favorisant ainsi l'innovation et l'utilisation des technologies les plus appropriées.
5. **Résilience et Tolérance aux Pannes** : En cas de défaillance d'un microservice, les autres continuent de fonctionner. Cela améliore la résilience globale du système, car une panne dans une partie de l'application n'impacte pas nécessairement l'ensemble.
6. **Facilitation du Travail en Équipe** : Les équipes peuvent travailler de manière indépendante sur des microservices spécifiques. Cela facilite le travail collaboratif et accélère le développement, car différentes équipes peuvent progresser simultanément sur différentes parties de l'application.
7. **Adaptabilité aux Technologies Futures** : L'architecture microservices permet d'adopter plus facilement de nouvelles technologies. Si une technologie évolue, il est possible de

mettre à jour uniquement le service concerné sans avoir à modifier l'ensemble de l'application.

8. **Meilleure Utilisation des Ressources** : Les microservices peuvent être déployés de manière élastique, ce qui signifie qu'ils peuvent s'ajuster dynamiquement à la demande. Cela permet une utilisation plus efficace des ressources et une optimisation des coûts d'infrastructure.

2. Architecture Microservices

- Architecture

Description des services

Service server

```
@EnableEurekaServer
@SpringBootApplication
public class ServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ServerApplication.class, args);
    }
}
```

ce code configure une application Spring Boot en tant que serveur Eureka. Lorsque cette application est lancée, elle agira comme un service de registre Eureka, permettant aux autres microservices de s'enregistrer et de découvrir les services disponibles dans l'écosystème.

microService gateways

```
@SpringBootApplication
public class GatewaysApplication {

    public static void main(String[] args) {
        SpringApplication.run(GatewaysApplication.class, args);
    }
    @Bean
    DiscoveryClientRouteDefinitionLocator
    routesDynamique(ReactiveDiscoveryClient rdc,
    DiscoveryLocatorProperties dlp){
        return new DiscoveryClientRouteDefinitionLocator(rdc,
    dlp);
    }
}
```

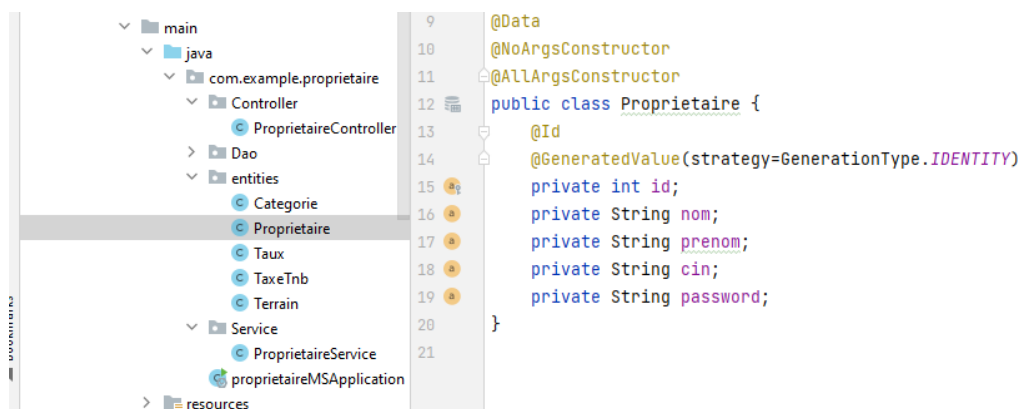
configurée en tant que passerelle (gateway) utilisant Spring Cloud Gateway et intégrant la découverte des services à l'aide de Spring Cloud Discovery.

Voici une brève explication du code :

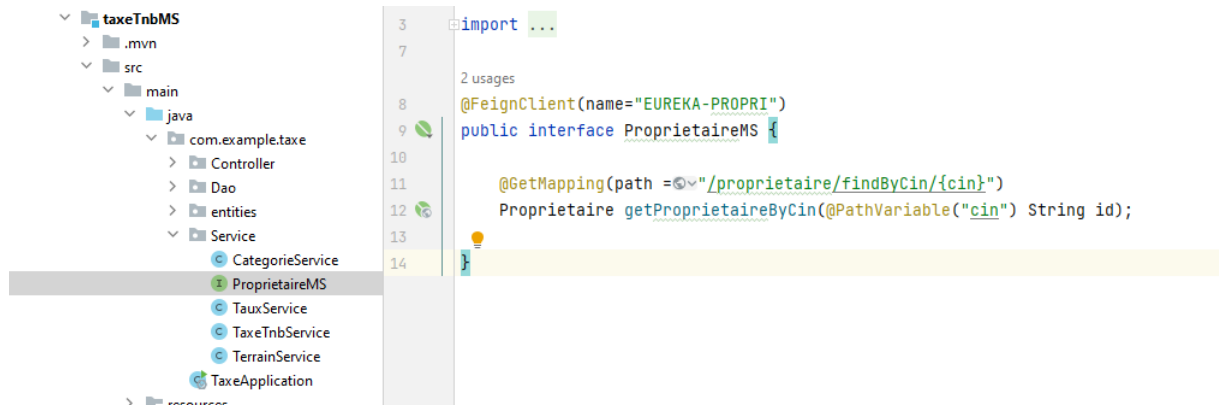
1. **@SpringBootApplication** :

- Cette annotation indique que c'est une application Spring Boot. Elle active la configuration automatique, la gestion des composants, et d'autres fonctionnalités spécifiques à Spring Boot.
2. **public class GatewaysApplication :**
 - C'est la classe principale de votre application.
 3. **public static void main(String[] args) :**
 - Cette méthode est le point d'entrée principal de l'application. Elle utilise **SpringApplication.run** pour démarrer l'application Spring Boot. La classe **GatewaysApplication.class** est spécifiée comme classe de configuration principale.
 4. **@Bean :**
 - Cette annotation est utilisée pour définir un bean Spring. Dans ce cas, vous définissez un bean de type **DiscoveryClientRouteDefinitionLocator** nommé **routesDynamique**.
 5. **DiscoveryClientRouteDefinitionLocator :**
 - Cette classe fait partie de Spring Cloud Gateway et est utilisée pour définir des routes dynamiques basées sur la découverte des services. Elle utilise un **ReactiveDiscoveryClient** pour obtenir des informations sur les services enregistrés, et **DiscoveryLocatorProperties** probablement pour configurer des propriétés liées à la découverte des services.
 6. **routesDynamique(ReactiveDiscoveryClient rdc, DiscoveryLocatorProperties dlp) :**
 - Cette méthode est le point où vous configurez le bean **DiscoveryClientRouteDefinitionLocator**. Vous fournissez un **ReactiveDiscoveryClient** et un **DiscoveryLocatorProperties** comme paramètres. Le **ReactiveDiscoveryClient** est utilisé pour découvrir les services, et le **DiscoveryLocatorProperties** est probablement utilisé pour configurer des propriétés spécifiques liées à la découverte des services.

Microservice proprietaireMS



Microservice taxeTNB



interface **ProprietaireMS** est une interface utilisée pour définir un client HTTP pour interagir avec microservice "EUREKA-PROPRI"

Test

The screenshot shows a web browser with the address bar displaying "localhost:8761". The page title is "DS Replicas". Below the title, it says "Instances currently registered with Eureka". The page contains a table with the following data:

Application	AMIs	Availability Zones	Status
EUREKA-PROPRI	n/a (1)	(1)	UP (1) - DESKTOP-4IT023H:EUREKA-PROPRI:8088
GATEWAY	n/a (1)	(1)	UP (1) - localhost:Gateway:8888
SERVICE-TAXE	n/a (1)	(1)	UP (1) - localhost:SERVICE-TAXE:8089

Test sur postman :

GET

⌵

http://localhost:8888/EUREKA-PROPRI/proprietaire/findByCin/{a12}

Params

Authorization

Headers (8)

Body ●

Pre-request Script

Tests

Settings

Query Params

	Key	Value
	Key	Value

Body

Cookies

Headers (5)

Test Results

⌐

Pretty

Raw

Preview

Visualize

Text ⌵

≡

1

Conteneurisation avec Docker

Implémentation

Définition du 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
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} taxeTnbMS.jar
EXPOSE 8081
ENTRYPOINT ["java", "-jar", "/taxeTnbMS.jar"]
```

Construction de l'image Docker :

docker build

Exécution du Conteneur :

docker run

<input type="checkbox"/>	 wizardly_sanderson 9e59a31db1d9 	springapi	Running	8089:8080 	1 minute ago	  
<input type="checkbox"/>	 peaceful_austin eb344cd225d1 	springapi	Running	8088:8080 	1 minute ago	  
<input type="checkbox"/>	 agitated_murdock 2df4673075ea 	springapi	Running	8761:8080 	1 minute ago	  
<input type="checkbox"/>	 thirsty_hamilton f8081ca98cdc 	springapi	Running	8888:8080 	1 minute ago	  

Les avantages

Portabilité :

Les conteneurs Docker encapsulent l'application et toutes ses dépendances, assurant une portabilité maximale. Un conteneur peut être exécuté de manière cohérente sur différents environnements, que ce soit sur un ordinateur de développement, un serveur de test ou en production.

Isolation :

Chaque conteneur fonctionne de manière isolée, partageant le noyau du système d'exploitation hôte mais possédant son propre système de fichiers, ses bibliothèques et ses processus. Cela garantit l'isolation des applications et évite les conflits de dépendances.

Facilité de Déploiement :

Les conteneurs peuvent être déployés rapidement et facilement. Les images Docker peuvent être partagées via des registres (comme Docker Hub), permettant une distribution efficace des applications et de leurs dépendances.

Évolutivité :

La conteneurisation facilite l'évolutivité horizontale, où plusieurs instances de conteneurs peuvent être déployées et gérées de manière dynamique pour répondre à la demande croissante. Les orchestrateurs de conteneurs, tels que Kubernetes, facilitent la gestion de l'évolutivité.

Gestion des Dépendances :

Les conteneurs encapsulent toutes les dépendances nécessaires, évitant ainsi les problèmes de conflits de versions et de compatibilité. Cela simplifie la gestion des dépendances et garantit une exécution cohérente de l'application.

Rapidité de Démarrage :

Les conteneurs ont des temps de démarrage très rapides par rapport aux machines virtuelles, car ils partagent le noyau de l'hôte. Cela permet un déploiement rapide et une mise à l'échelle dynamique.

Sécurité :

La conteneurisation renforce la sécurité en isolant les applications et en réduisant la surface d'attaque. Les conteneurs offrent également la possibilité de définir des politiques de sécurité spécifiques.

5. CI/CD avec Jenkins

CI/CD (Intégration Continue / Déploiement Continu) avec Jenkins est une pratique courante dans le développement logiciel qui vise à automatiser les processus d'intégration, de tests et de déploiement. Jenkins est un outil d'automatisation open source largement utilisé pour implémenter des pipelines CI/CD.

Intégration Continue (CI) :



The image shows a Jenkins configuration interface for a GitHub project. It includes a checked checkbox for 'GitHub project', a 'Project url' field with a help icon, and a text input containing the URL 'https://github.com/chaymaa2001/projet2.git'. Below the input is an 'Avancé' dropdown menu.

Definition du script pipeline


```

pipeline {

    agent any

    tools {

        maven 'maven'

    }

    stages {

        stage('Git Clone') {

            steps {

                script {

                    checkout([$class: 'GitSCM', branches: [[name: 'main']], userRemoteConfigs: [[url:
'https://github.com/chaymaa2001/projet2.git']]])

                }

            }

        }

        stage(' Build projet cont') {

            steps {

                script {

                    dir('taxeTnbMS') {

                        bat './mvnw clean install'

                    }

                }

            }

        }

        stage('Build tax') {

            steps {

                script {

                    dir('proprietaireMS') {

                        bat './mvnw clean install'

                    }

                }

            }

        }

    }

}

```

```
    }  
  }  
}  
}
```

```
stage('Build server') {  
  steps {  
    script {  
      dir('server') {  
        bat './mvnw clean install'  
      }  
    }  
  }  
}
```

```
stage('Build gateway') {  
  steps {  
    script {  
      dir('gateways') {  
        bat './mvnw clean install'  
      }  
    }  
  }  
}
```

```
stage('Create Docker Images') {  
  steps {
```

```

script {

    bat 'docker.build -t projet cont/test/proprietaireMS'

    bat 'docker.build -t projet cont/test/taxTnbMS'

    bat 'docker.build -t projet cont/test/server'

    bat 'docker.build -t projet cont/test/gateways'

}

}

}

}

```

7. Intégration de SonarQube

Configuration

```

stage('SonarQube Analysis') {

    steps {

        withSonarQubeEnv('SonarQubeServer') {

            sh 'mvn sonar:sonar'

        }

    }

}

```

Les bénéfices de sonarqube

Bénéfices pour la Qualité du Code :

1. Détection Précoce des Problèmes :

- SonarQube identifie les problèmes de qualité du code dès le stade de développement, permettant une correction précoce et évitant l'accumulation de défauts.

2. Analyse Automatique :

- L'analyse statique du code est automatisée dans le pipeline Jenkins, ce qui garantit que chaque commit ou chaque build est évalué.

3. Mesure de la Dette Technique :

- SonarQube fournit des mesures de la dette technique, indiquant les zones du code qui nécessitent une amélioration en termes de complexité, de duplications, et d'autres critères.
4. **Recommandations de Correctifs :**
 - SonarQube offre des recommandations pour corriger les problèmes détectés, aidant les développeurs à améliorer la qualité du code.
 5. **Intégration avec le Processus d'Intégration Continue :**
 - L'intégration de SonarQube dans le pipeline Jenkins fait partie d'un processus d'intégration continue, assurant que la qualité du code est constamment évaluée et maintenue.
 6. **Rapports et Tableaux de Bord :**
 - SonarQube génère des rapports et des tableaux de bord visuels pour aider les équipes de développement à suivre l'évolution de la qualité du code au fil du temps.

Conclusion

Résumé des accomplissements

- Architecture Microservices :
- Mise en place d'une architecture basée sur des microservices pour favoriser la scalabilité, la flexibilité et la maintenabilité de l'application.
- Utilisation de Spring Cloud :
- Intégration de Spring Cloud pour la gestion de la configuration, la découverte des services, et la gestion des passerelles (gateway).
- Conteneurisation avec Docker :
- Adoption de Docker pour la conteneurisation des microservices, facilitant le déploiement et la gestion des applications dans des environnements variés.
- CI/CD avec Jenkins :
- Mise en place d'un pipeline CI/CD automatisé avec Jenkins, accélérant le cycle de développement, améliorant la qualité du code, et facilitant les déploiements continus.
- Intégration de SonarQube :
- Intégration de SonarQube dans le pipeline pour l'analyse statique du code, permettant une évaluation automatique de la qualité du code et la détection précoce des problèmes.

Perspectives futures

1. **Optimisation de la Performance :**
 - Examiner les performances de l'application pour identifier les opportunités d'optimisation. Utiliser des outils de profilage et de surveillance pour résoudre les goulots d'étranglement.

2. **Sécurité :**

- Renforcer la sécurité en implémentant des pratiques de développement sécurisées, en effectuant des analyses de sécurité régulières, et en intégrant des mécanismes de protection dans l'architecture.

3. **Évolutivité Continue :**

- Continuer à améliorer l'évolutivité de l'application en optimisant les microservices existants, en adoptant des technologies émergentes, et en ajustant l'architecture en fonction des besoins.