

2023/2024

5IIR /G6

Rapport Projet : Architecture des composants d'entreprise

Réalisé par :



Bougrine Mohamed

Abderrahmane Charak

Mohamed Ayman Yakhmim

Encadré par :



Mr. Lachgar Mohamed

1. Introduction

Le projet de gestion des taxes pour des terrains, vise à concevoir et développer une plateforme moderne pour la gestion fiscale immobilière. En utilisant une architecture de microservices, le backend sera développé avec Spring Boot, tandis que le frontend utilisera Angular. L'objectif principal est de créer une application robuste, évolutive et hautement modulaire qui facilitera la gestion des taxes pour les propriétaires de terrains. Les fonctionnalités incluront la déclaration, le suivi des paiements, et la génération de rapports pour assurer une gestion fiscale transparente et conviviale.

Importance de l'architecture microservices :

Nous avons choisi une architecture microservices en raison des nombreux avantages qu'elle offre en termes de flexibilité, d'évolutivité, de maintenance et de collaboration

Évolutivité et Flexibilité : L'architecture microservices permet une évolutivité horizontale, offrant la possibilité de développer, déployer et mettre à l'échelle chaque service de manière indépendante. Cette approche permet de répondre plus facilement aux changements et aux évolutions nécessaires pour l'application.

Indépendance Technologique : Chaque microservice peut être développé avec des technologies indépendantes, offrant ainsi une flexibilité quant au choix des outils et des langages adaptés à chaque service.

Facilité de Maintenance : La maintenance des microservices est simplifiée en raison de leur nature indépendante. Les mises à jour, correctifs et améliorations peuvent être appliqués à un service spécifique sans perturber les autres parties de l'application.

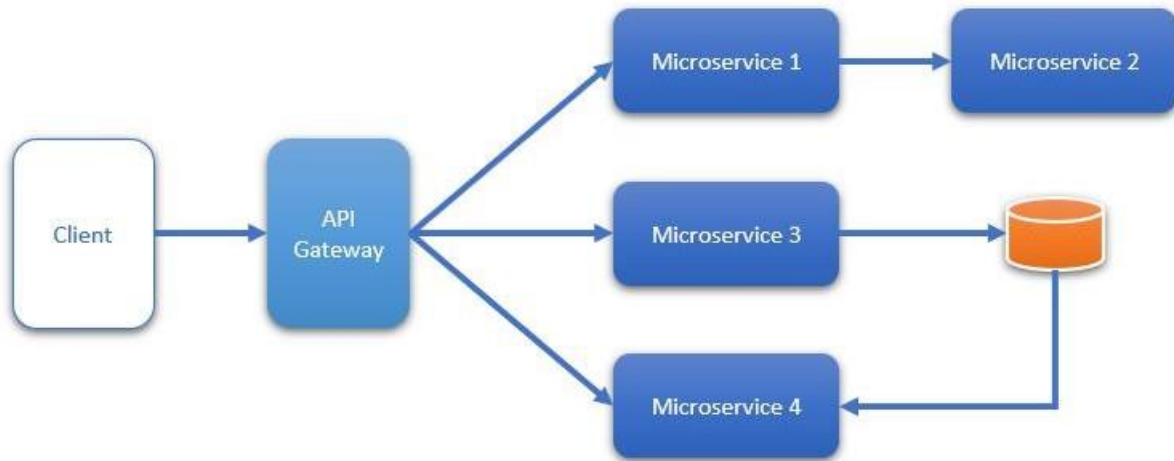
Isolation des Erreurs : Les microservices offrent une isolation des erreurs. En cas de défaillance d'un service, les autres peuvent continuer à fonctionner normalement, contribuant ainsi à créer des applications résilientes et à minimiser l'impact des bugs.

Intégration Continue (CI/CD) : Les microservices facilitent la mise en œuvre de pratiques de développement agile, de déploiement continu (CI/CD) et de livraison continue. Chaque service peut être déployé indépendamment, accélérant ainsi le cycle de développement.

Scalabilité Fine : La scalabilité fine est réalisable avec les microservices. Cela signifie que seuls les services nécessitant une augmentation de

capacité doivent être mis à l'échelle, évitant ainsi de dimensionner l'ensemble de l'application.

2. Architecture Microservices



L'architecture de l'application repose sur trois microservices distincts, chacun dédié à des fonctions spécifiques. Nous avons un microservice dédié à la gestion des utilisateurs, un autre microservice dédié à la gestion des taxes, et enfin, un microservice utilisant RabbitMQ pour la gestion des messages. Chacun de ces microservices est déployé de manière indépendante sur des serveurs distincts, avec sa propre base de données pour assurer une isolation et une performance optimale.

Description des services :

Service de Gestion des Utilisateurs : Ce service est responsable de la gestion des utilisateurs au sein de l'application. Il inclut des fonctionnalités telles que l'inscription, la connexion, la mise à jour de profil et la gestion des autorisations. Les informations utilisateur sont stockées de manière sécurisée, et ce service offre des interfaces pour interagir avec la base de données dédiée aux utilisateurs.

Service de Gestion des Taxes : Le service de gestion des taxes s'occupe de la gestion des données relatives aux taxes associées aux terrains. Il offre des fonctionnalités telles que l'ajout, la modification et la suppression des informations fiscales pour chaque terrain. Ce service garantit la cohérence des données fiscales et fournit des interfaces pour l'intégration avec d'autres parties de l'application.

Service RabbitMQ : Ce microservice exploite RabbitMQ, un système de messagerie, pour la gestion des messages au sein de l'application. Il facilite la communication asynchrone entre différentes parties du système, permettant une intégration souple et réactive. Les messages peuvent être échangés entre les microservices de manière fiable, contribuant ainsi à la mise en œuvre de flux de travail distribués.

Chaque service est conçu de manière indépendante, favorisant la modularité et la maintenance simplifiée. Ces microservices interagissent entre eux au sein de l'architecture globale de l'application pour offrir une expérience utilisateur cohérente et robuste.

Mécanismes de communication:

En ce qui concerne l'interaction entre les microservices, nous avons adopté l'utilisation de RestTemplate.

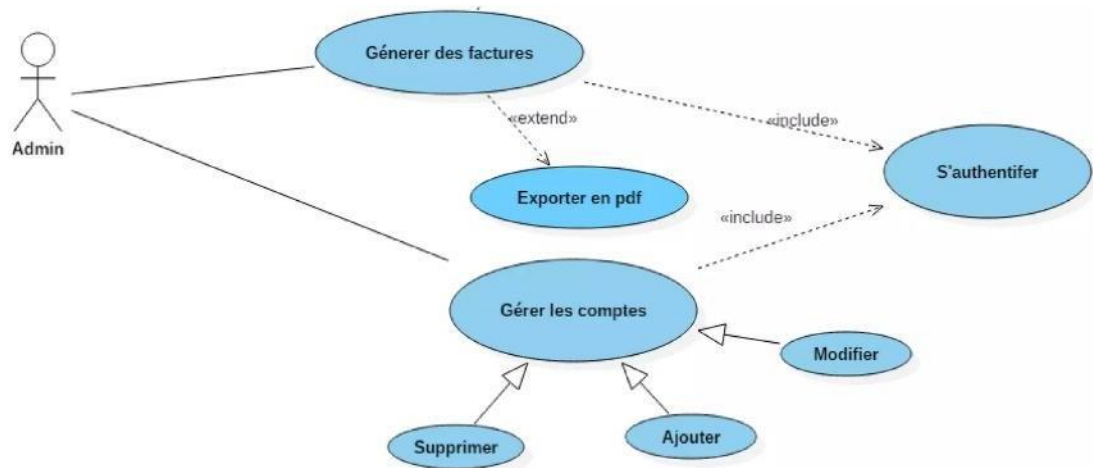
Il s'agit d'un composant essentiel permettant d'initier des appels vers des services distants au moyen de requêtes HTTP.

RestTemplate représente un choix largement adopté pour la mise en œuvre de clients HTTP synchrones dans l'écosystème Spring Boot. Il offre une interface simple et intuitive pour l'exécution de requêtes HTTP, la gestion des réponses, et prend en charge une diversité de méthodes HTTP, de types de requêtes et de réponses, ainsi que des convertisseurs de messages.

Cela assure une communication efficace entre les microservices, favorisant ainsi une architecture modulaire et réactive au sein de l'application globale.

3. Conception des Microservices

Diagramme de cas d'utilisation :



4. Conteneurisation avec Docker

a. Avantages de la conteneurisation de l'application :

Isolation des applications : Les conteneurs Docker offrent une isolation efficace pour les applications et leurs dépendances, garantissant une exécution cohérente indépendamment de l'environnement de déploiement. Cette approche prévient les conflits entre les dépendances, assurant ainsi une portabilité accrue des applications.

Portabilité : Les conteneurs Docker encapsulent l'application ainsi que toutes ses dépendances, assurant une portabilité sans effort. Cette flexibilité permet d'exécuter le même conteneur sur divers environnements tels que le développement local, les serveurs de test ou le cloud, sans se préoccuper des variations environnementales.

Rapidité des déploiements : La création et le déploiement des conteneurs sont des opérations rapides. Grâce à la facilité de partage des images de conteneurs via Docker Hub ou d'autres registres, les déploiements deviennent extrêmement efficaces.

Gestion des ressources : Docker propose une gestion optimisée des ressources système en isolant les conteneurs. Cette approche maximise l'utilisation des ressources, améliorant ainsi les performances tout en évitant les conflits.

Évolutivité : Les conteneurs peuvent être aisément mis à l'échelle horizontalement pour faire face à des charges de travail en expansion. La gestion automatisée des conteneurs simplifie le déploiement, réduisant ainsi le délai de mise sur le marché.

b. Création des images docker :

```

docker-compose.yml  Dockerfile x
1  FROM openjdk:17
2
3  WORKDIR /App
4
5  COPY /target/taxe_tnb-0.0.1-SNAPSHOT.jar .
6  💡
7  EXPOSE 9090
8
9  ENTRYPOINT ["java", "-jar", "taxe_tnb-0.0.1-SNAPSHOT.jar"]
  
```

```

er-compose.yml  micro1\Dockerfile  microservices-messaging-consumer\Dockerfile  micro2\Dockerfile x
1  FROM openjdk:17
2
3  WORKDIR /App
4
5  COPY /target/vol-0.0.1-SNAPSHOT.jar .
6
7  EXPOSE 8081
8  💡
9  ENTRYPOINT ["java", "-jar", "vol-0.0.1-SNAPSHOT.jar"]
  
```

```

docker-compose.yml  micro1\Dockerfile  microservices-messaging-consumer\Dockerfile x
1  FROM openjdk:17
2
3  WORKDIR /App
4
5  COPY /target/messaging-consumer-0.0.1-SNAPSHOT.jar .
6
7  EXPOSE 9111
8  💡
9  ENTRYPOINT ["java", "-jar", "messaging-consumer-0.0.1-SNAPSHOT.jar"]
  
```

```
ker-compose.yml  micro1\Dockerfile  micro2\Dockerfile  microservices-messaging-producer\Dockerfile × ▾
1  FROM openjdk:17
2
3  WORKDIR /App
4
5  COPY /target/messaging-0.0.1-SNAPSHOT.jar .
6
7  EXPOSE 8082
8  ⚡
9  ENTRYPOINT ["java", "-jar", "messaging-0.0.1-SNAPSHOT.jar"]
```

```
docker-compose.yml ×  micro1\Dockerfile  micro2\Dockerfile
1  version: '3.8'
2  services:
3  rabbitmq:
4    container_name: rabbitmq
5    image: rabbitmq:management
6    ports:
7      - "5672:5672"
8      - "15672:15672"
9    networks:
10     - backend
11  db:
12    image: mysql:8.0
13    restart: always
14    environment:
15      MYSQL_DATABASE: terrain
16      MYSQL_ALLOW_EMPTY_PASSWORD: "yes"
17    ports:
18      - "3316:3306"
```



```

docker-compose.yml × micro1\Dockerfile micro2\Dockerfile
17   ports:
18     - "3316:3306"
19   volumes:
20     - ./db:/var/lib/mysql
21   networks:
22     - backend
23 db1:
24   image: mysql:8.0
25   restart: always
26   environment:
27     MYSQL_DATABASE: terrain_auth
28     MYSQL_ALLOW_EMPTY_PASSWORD: "yes"
29   ports:
30     - "3317:3306"
31   volumes:
32     - ./db1:/var/lib/mysql
33   networks:
34     - backend
  
```

```

docker-compose.yml × micro1\Dockerfile micro2\Dockerfile
32     - ./db1:/var/lib/mysql
33   networks:
34     - backend
35 db2:
36   image: mysql:8.0
37   restart: always
38   environment:
39     MYSQL_DATABASE: taxetnbusers
40     MYSQL_ALLOW_EMPTY_PASSWORD: "yes"
41   ports:
42     - "3318:3306"
43   volumes:
44     - ./db2:/var/lib/mysql
45   networks:
46     - backend
47 networks:
48   backend:
49     driver: bridge
  
```


5. CI/CD avec Jenkins

Pour automatiser le clone du projet, le build, la création de l'image docker et la création et le lancement des conteneurs, on crée un pipeline pour le projet :

Tableau de bord > projet > Configuration

Configure

projet micro service

Plain text [Prévisualisation](#)

☐ Ce build a des paramètres ?

☐ Do not allow concurrent builds

☐ Do not allow the pipeline to resume if the controller restarts

☒ GitHub project

Project url ?

Avancé ▾

☐ Pipeline speed/durability override ?

Script qu'on a utilisé :

```

1 pipeline {
2   agent any
3   tools {
4     maven 'maven'
5   }
6   stages {
7     stage('Git Clone') {
8       steps {
9         script {
10          checkout([$class: 'GitSCM', branches: [[name: 'main']],
11          userRemoteConfigs: [[url:
12            'https://github.com/hamzamoussebbih/mic.git']]
13          ])
14        }
15      }
16    }
17  }
18  stage('Build') {
19    steps {
20      script {
21        dir('micro1') {
22          bat 'mvn clean install'
23        }
24      }
25    }
26  }
27  stage('Build2') {
28    steps {
29      script {
30        dir('micro2') {
31          bat 'mvn clean install'
32        }
33      }
34    }
35  }
36 }

```

```

36 stage('Build3') {
37   steps {
38     script {
39       dir('micro3/microservices-messaging-consumer') {
40         bat 'mvn clean install'
41       }
42     }
43   }
44 }
45 stage('Build4') {
46   steps {
47     script {
48       dir('micro4/microservices-messaging-producer') {
49         bat 'mvn clean install'
50       }
51     }
52   }
53 }

```

```

55 stage('Create Docker Image') {
56   steps {
57     script {
58       bat 'docker build -t micro1 ./micro1'
59       bat 'docker build -t micro2 ./micro2'
60       bat 'docker build -t micro3 ./micro3/microservices-messaging-consumer'
61       bat 'docker build -t micro4 ./micro4/microservices-messaging-producer'
62     }
63   }
64 }

```

Script ?

```

60       bat 'docker build -t micro3 ./micro3/microservices-messaging-consumer'
61       bat 'docker build -t micro4 ./micro4/microservices-messaging-producer'
62     }
63   }
64 }
65
66 stage('Run') {
67   steps {
68     bat 'docker run --name test-micro1 -d -p 8585:9090 micro1'
69     bat 'docker run --name test-micro2 -d -p 8686:8081 micro2'
70     bat 'docker run --name test-micro3 -d -p 8787:9111 micro3'
71     bat 'docker run --name test-micro4 -d -p 8888:8082 micro4'
72   }
73 }
74
75 }
76
77 }

```

☒ Use Groovy Sandbox ?

Pipeline Syntax

Sauver

Apply

Git Clone:

Description : Cette étape effectue la récupération du code source depuis un dépôt Git. Elle cible la branche principale ("main") du dépôt

'https://github.com/hamzamoussebbih/mic.git'.

Objectif : Assurer que le code source le plus récent est récupéré pour le processus de build.

Build (micro1) :

Description : Compilation du premier microservice ('micro1') à l'aide de Maven. La commande 'mvn clean install' nettoie le projet et construit le package.

Objectif : S'assurer que le microservice 'micro1' est compilé avec succès.

Build2 (micro2) :

Description : Compilation du deuxième microservice ('micro2') à l'aide de Maven. La commande 'mvn clean install' nettoie le projet et construit le package.

Objectif : S'assurer que le microservice 'micro2' est compilé avec succès.

Build3 (micro3) :

Description : Compilation du troisième microservice ('micro3/microservices-messaging-consumer') à l'aide de Maven. La commande 'mvn clean install' nettoie le projet et construit le package.

Objectif : S'assurer que le microservice 'micro3' est compilé avec succès.

Build4 (micro4) :

Description : Compilation du quatrième microservice ('micro4/microservices-messaging-producer') à l'aide de Maven. La commande 'mvn clean install' nettoie le projet et construit le package.

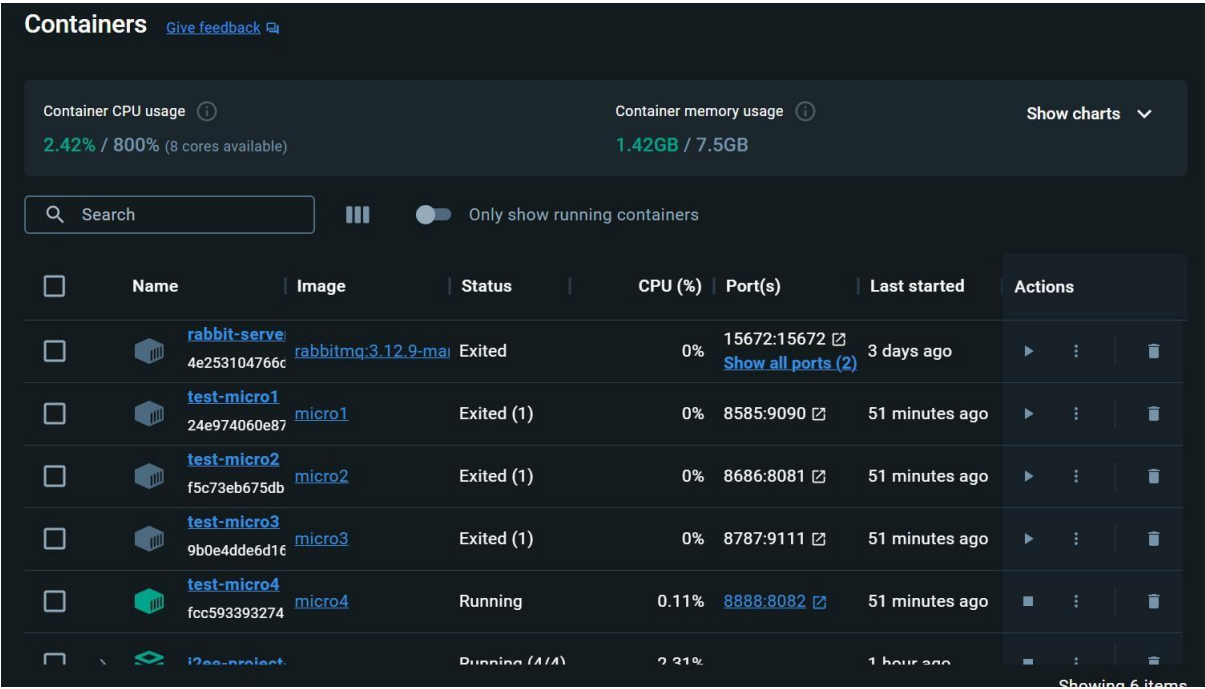
Objectif : S'assurer que le microservice 'micro4' est compilé avec succès.









Create Docker Image :

Description : Création des images Docker pour chaque microservice.

Chaque image est étiquetée avec le nom correspondant ('micro1', 'micro2', 'micro3', 'micro4').

Objectif : Préparer les images Docker nécessaires pour le déploiement.



	j2ee-project-main... mysql:8.0 Running 3317:3306 ↗	■	⋮	
	j2ee-project-main... mysql:8.0 Running 3318:3306 ↗	■	⋮	
	j2ee-project-main... mysql:8.0 Running 3316:3306 ↗	■	⋮	
	rabbitmq rabbitmq:manageme Running 15672:15672 ↗ Show all ports (2)	■	⋮	

7. Intégration de SonarQube :

Configure the `SONAR_TOKEN` environment variable

- 1 Name of the environment variable: `SONAR_TOKEN` [↗](#)
- 2 Value of the environment variable: `9a4f7ff781c5aa58b2880c60f252e4f7fe2cce33` [↗](#) [✎](#)

Execute the SonarScanner from your computer
Run the following command in your project's folder.

```
sonar-scanner.bat \
-D"sonar.organization=msbhsonarcloud" \
-D"sonar.projectKey=msbhsonarcloud_micro" \
-D"sonar.sources=." \
-D"sonar.host.url=https://sonarcloud.io"
```

[Copy](#)

```

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
    <plugin>
      <groupId>org.sonarsource.scanner.maven</groupId>
      <artifactId>sonar-maven-plugin</artifactId>
      <version>3.9.1.2184</version>
    </plugin>
  </plugins>
</build>

```

```

<description>taxe_tnb</description>
<properties>
  <java.version>17</java.version>
  <sonar.organization>msbhsonarcloud</sonar.organization>
  <sonar.host.url>https://sonarcloud.io</sonar.host.url>
  <sonar.login>9a4f7ff781c5aa58b2880c60f252e4f7fe2cce33</sonar.login>
</properties>
<dependencies> Edit Starters...
<dependency>

```

mvn verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -
 Dsonar.projectKey=msbhsonarcloud_micro

```

PS C:\Users\info\Downloads\J2EE-Project-main\micro1> mvn -v
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dccc295161ae)
Maven home: C:\Users\info\Downloads\apache-maven-3.9.6-bin\apache-maven-3.9.6
Java version: 17.0.8, vendor: Eclipse Adoptium, runtime: C:\Users\info\jdk-17.0.8+7
Default locale: fr_FR, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
PS C:\Users\info\Downloads\J2EE-Project-main\micro1> mvn verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -
Dsonar.projectKey=msbhsonarcloud_micro
[INFO] Scanning for projects...
[INFO] -----< org.emsi:taxe_tnb >-----
[INFO] Building taxe_tnb 0.0.1-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO] --- resources:3.3.1:resources (default-resources) @ taxe_tnb ---
[INFO] Copying 2 resources from src\main\resources to target\classes
[INFO] Copying 0 resource from src\main\resources to target\classes
[INFO] --- compiler:3.11.0:compile (default-compile) @ taxe_tnb ---
[INFO] Changes detected - recompiling the module! :input tree
[INFO] Compiling 27 source files with javac [debug release 17] to target\classes
[INFO] --- resources:3.3.1:testResources (default-testResources) @ taxe_tnb ---
[INFO] skip non existing resourceDirectory C:\Users\info\Downloads\J2EE-Project-main\micro1\src\test\resources
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ taxe_tnb ---

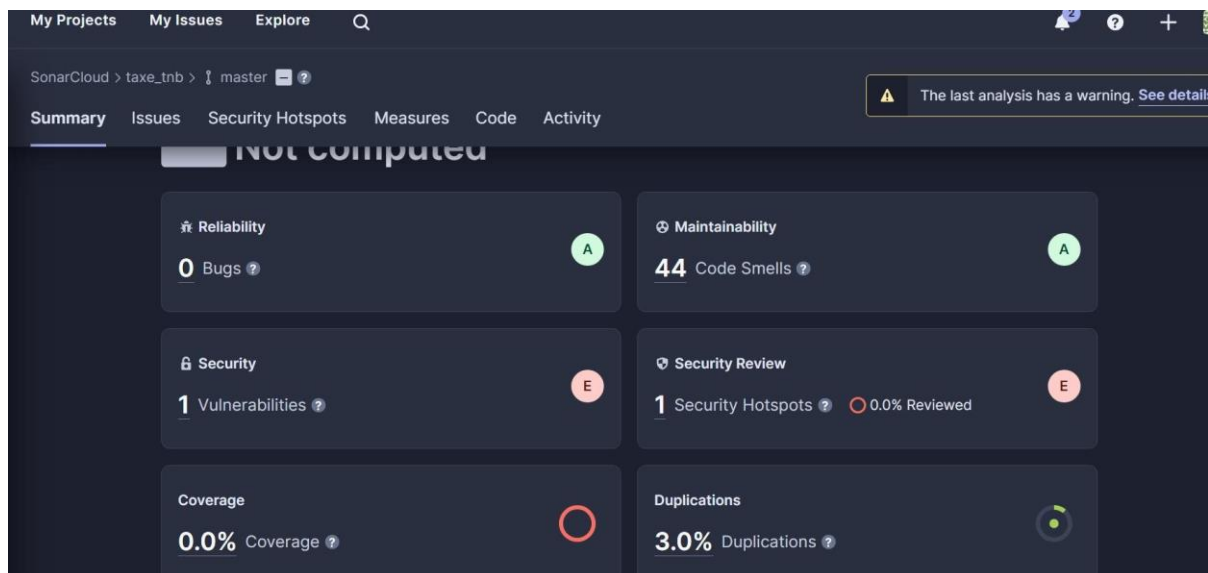
```



```

[INFO] ----- Run sensors on project
[INFO] Sensor Zero Coverage Sensor
[INFO] Sensor Zero Coverage Sensor (done) | time=25ms
[INFO] Sensor Java CPD Block Indexer
[INFO] Sensor Java CPD Block Indexer (done) | time=38ms
[INFO] SCM Publisher SCM provider for this project is: git
[INFO] SCM Publisher 29 source files to be analyzed
[INFO] SCM Publisher 28/29 source files have been analyzed (done) | time=867ms
[WARNING] Missing blame information for the following files:
[WARNING] * pom.xml
[WARNING] This may lead to missing/broken features in SonarCloud
[INFO] CPD Executor 7 files had no CPD blocks
[INFO] CPD Executor Calculating CPD for 20 files
[INFO] CPD Executor CPD calculation finished (done) | time=17ms
[INFO] Analysis report generated in 288ms, dir size=310 KB
[INFO] Analysis report compressed in 146ms, zip size=110 KB
[INFO] Analysis report uploaded in 632ms
[INFO] ANALYSIS SUCCESSFUL, you can find the results at: https://sonarcloud.io/dashboard?id=msbhsonarcloud_micro
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at https://sonarcloud.io/api/ce/task?id=AY0uXA629Gd7d8rK5vKE
[INFO] Sensor cache published successfully
[INFO] Analysis total time: 23.267 s
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 44.866 s
[INFO] Finished at: 2024-01-22T00:30:04+01:00
[INFO] -----

```



8. Conclusion

La mise en place réussie de notre pipeline CI/CD pour une application de blog a abouti à des réalisations significatives. En utilisant Jenkins, nous avons automatisé le processus de récupération du code depuis le référentiel Git, construit chaque module avec Maven, créé des images Docker pour chaque microservice, et coordonné le déploiement avec Docker Compose. L'incorporation de l'analyse statique de code via SonarQube a renforcé la qualité du code en identifiant les vulnérabilités, les erreurs potentielles, et en fournissant des métriques de qualité. De plus, nous avons élargi notre pipeline pour inclure Angular et déployer une interface utilisateur attrayante. Grâce au déploiement sur le cloud, nous avons gagné en flexibilité et en élasticité, avec la possibilité d'utiliser des services cloud tels qu'AWS, Azure ou GCP. Ces réalisations posent les bases d'une gestion efficace du cycle de vie de l'application, couvrant l'automatisation de la construction et du déploiement jusqu'à l'amélioration continue de la qualité du code. Pour l'avenir, nous envisageons d'explorer davantage d'améliorations, notamment l'adoption de l'intégration continue/déploiement.

continu (CI/CD), l'orchestration de conteneurs avec Kubernetes, et l'optimisation des aspects opérationnels afin d'assurer une application robuste et évolutive.