



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

Membre de 
HONORIS UNITED UNIVERSITIES

Rapport projet

Architecture des composants d'entreprise

Réalisé par :

ELMADARI CHADI

Table des matières

I. Introduction.....	3
1. Aperçu du projet :	3
2. Importance de l'architecture microservices :	3
II. Architecture Microservices.....	4
1. Architecture :	4
2. Description des services :	5
3. Mécanismes de communication :	6
III. Conception des Microservices :	6
IV. Conteneurisation avec Docker :	7
V. CI/CD avec Jenkins :	9
VI. Intégration de SonarQube	13
1. Configuration.....	13
2. bénéfices pour la qualité du code :	15
VIII. conclusion.....	18

I. Introduction

1. Aperçu du projet :

Le projet a pour objectif de concevoir et mettre en œuvre une architecture basée sur des microservices pour la gestion de l'authentification, la gestion des taxes sur le terrain (TNB), et un frontend Angular. Cette approche modulaire vise à améliorer la flexibilité, la scalabilité, tout en facilitant la maintenance.

2. Importance de l'architecture microservices :

L'importance de l'architecture microservices réside dans plusieurs avantages qui répondent aux défis auxquels font face les développeurs et les entreprises dans le domaine du développement logiciel. Voici quelques points clés mettant en évidence l'importance de l'architecture microservices :

1. Scalabilité et Flexibilité :

- Les microservices permettent une scalabilité granulaire, car chaque service peut être développé, déployé et mis à l'échelle indépendamment des autres.
- Les équipes peuvent travailler sur des services distincts, facilitant ainsi la flexibilité et l'adaptabilité aux changements requis par l'évolution des exigences métier.

2. Découplage des Services :

- Chaque microservice fonctionne de manière autonome, avec sa propre base de code, sa base de données et sa logique métier.
- Le découplage permet aux équipes de développement de travailler de manière indépendante, ce qui simplifie la gestion des versions, des mises à jour et des déploiements.

3. Facilité de Déploiement :

- La nature modulaire des microservices facilite le déploiement continu et l'intégration continue (CI/CD), accélérant ainsi le cycle de vie du développement logiciel.
- Les mises à jour peuvent être effectuées sans affecter l'ensemble de l'application, minimisant ainsi les temps d'arrêt.

4. Technologies et Langages Divers :

- Les équipes peuvent choisir les technologies et les langages de programmation les plus adaptés à chaque service, plutôt que d'être limitées à une seule pile

technologique pour l'ensemble de l'application.

- Cela favorise l'utilisation des meilleures technologies pour des tâches spécifiques et permet d'adopter de nouvelles technologies sans perturber l'ensemble du système.

5. Résilience et Tolérance aux Pannes :

- En cas de défaillance d'un microservice, les autres services continuent de fonctionner, améliorant la résilience globale de l'application.
- La conception distribuée permet une meilleure gestion des pannes et une récupération plus rapide.

6. Évolutivité Organique :

- Les nouvelles fonctionnalités peuvent être ajoutées en développant de nouveaux microservices plutôt qu'en modifiant une monolithique, facilitant l'évolutivité organique de l'application.

7. Meilleure Gestion des Équipes :

- Les équipes peuvent être organisées autour des microservices, favorisant la responsabilité et la propriété de bout en bout, ce qui améliore la communication et la collaboration.

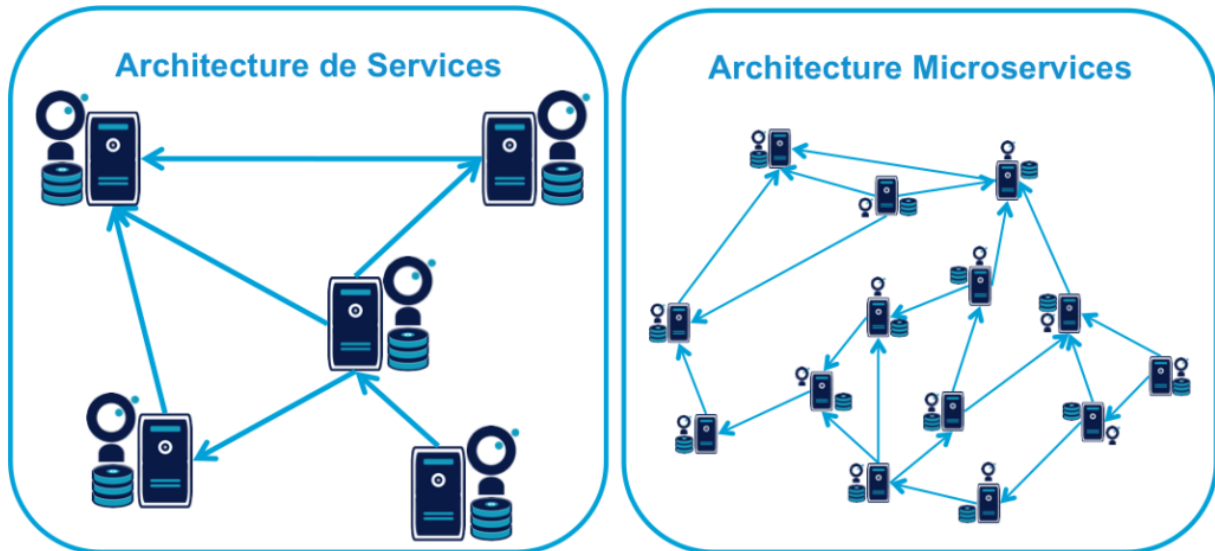
8. Analyse et Mesure Précises :

- Les microservices facilitent la collecte de données spécifiques à chaque service, offrant une meilleure visibilité et des mesures précises des performances et de l'utilisation.

II. Architecture Microservices

1. Architecture :

Un micro-service est une petite application destinée à faire une seule fonctionnalité. Par exemple, une application qui envoie un texte à une adresse mail peut être un micro-service. C'est une seule brique de l'application globale, une brique indépendante ayant une seule responsabilité indépendante. Une personne, voire une équipe peut travailler de façon libre et autonome sur un micro-service, le concevoir à leur choix, le coder avec un langage de programmation qu'ils sélectionnent eux-mêmes et le déploient sur le serveur qu'ils veulent. Puis ils fournissent le micro-service en tant qu'API par exemple. L'application globale sera par la suite la combinaison et l'intégration de tous les micro-services.



2. Description des services :

Service d'Authentification

Le service d'authentification est conçu pour répondre aux besoins liés à l'identification et à la gestion des utilisateurs. Voici une description détaillée de ses composants :

- Processus d'Inscription
- Processus de Connexion
- Gestion des Identités Utilisateur
-

Service TNB (Taxes sur le Terrain)

Le service TNB est dédié à la gestion des opérations liées aux taxes foncières. Sa conception est axée sur la manipulation efficace des données fiscales, et voici une vue détaillée de ses fonctionnalités :

- Manipulation des Taxes Foncières
- Manipulation des Terrains
- Manipulation des Taux
- Manipulation des Taxes Redevables

Frontend Angular

Le frontend Angular agit comme une interface utilisateur intuitive permettant l'interaction avec les microservices. Ses fonctionnalités sont conçues pour offrir une expérience utilisateur optimale

3. Mécanismes de communication :

Apache Kafka est une plateforme de streaming de données open source. À l'origine, elle fut développée en interne par LinkedIn en guise de queue de messagerie. Toutefois, cet outil largement évolué et ses cas d'usage se sont multipliés.

Cette plateforme est écrite en Scala et Java. Elle est toutefois compatible avec une large variété de langages de programmation.

À la différence de queues de messagerie traditionnelles comme RabbitMQ, Kafka retient les messages après qu'ils aient été consommés pendant une certaine période de temps. Les messages ne sont pas supprimés immédiatement après confirmation de réception.

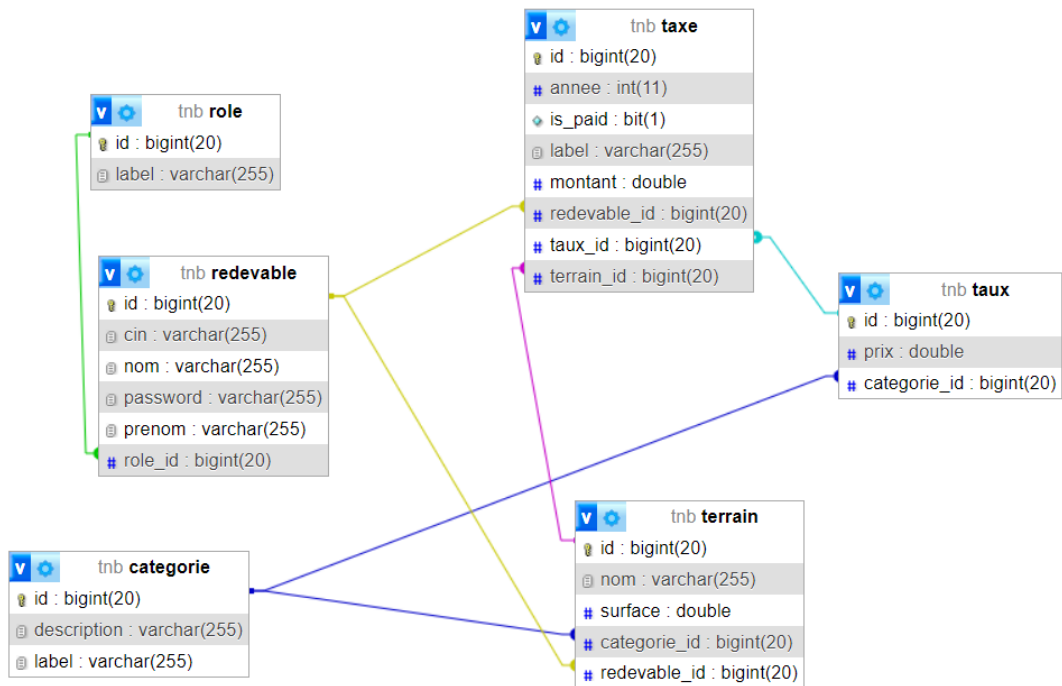
En outre, les queues de messageries sont d'ordinaire conçues pour s'étendre verticalement via l'ajout de puissance à une machine. De son côté, Kafka s'étend horizontalement grâce à l'ajout de nœuds supplémentaires au cluster de serveurs.

Il faut savoir que Kafka est distribué. Cela signifie que ses capacités sont élastiques. Il suffit d'ajouter des nœuds, c'est-à-dire des serveurs, à un cluster pour l'étendre.

Une autre particularité de Kafka est sa faible latence. Cela signifie qu'elle peut prendre en charge le traitement de nombreuses données en temps réel.

III. Conception des Microservices :

Le service d'authentification gère les données utilisateur, le service TNB gère les informations fiscales, et le frontend interagit avec ces services de manière transparente



IV. Conteneurisation avec Docker :

Grâce à Docker, les conteneurs deviennent des machines virtuelles très légères et modulaires qui vous offrent une grande flexibilité pour créer, déployer, copier des conteneurs et les déplacer d'un environnement à un autre. Vos applications sont ainsi optimisées pour le cloud.

Dockerfile 1 :

```

FROM maven:3.8.4-openjdk-17 AS builder
WORKDIR /app
COPY ./src ./src
COPY ./pom.xml .
RUN mvn clean package
  
```

```

FROM openjdk:17-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} auth-service.jar
ENTRYPOINT ["java","-jar","/auth-service.jar"]
  
```

Dockerfile 2 :

```

FROM maven:3.8.4-openjdk-17 AS builder
WORKDIR /app
COPY ./src ./src
COPY ./pom.xml .
RUN mvn clean package
  
```

```

FROM openjdk:17-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=target/*.jar
  
```

```
COPY ${JAR_FILE} tnb-service.jar  
ENTRYPOINT ["java","-jar","/tnb-service.jar"]
```

Dockercompose :

```
version: '3'  
  
services:  
  mysql:  
    image: mysql:latest  
    container_name: mysql-container2  
    environment:  
      MYSQL_ROOT_PASSWORD: root  
    ports:  
      - "3306:3306"  
    networks:  
      - microservices-network  
  
  client-service2:  
    build:  
      context: ./tnb-maroc  
    ports:  
      - "8090:8090"  
    depends_on:  
      - mysql  
    networks:  
      - microservices-network  
    environment:  
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/tnb?createDatabaseIfNotExist=true  
      SPRING_DATASOURCE_USERNAME: root  
      SPRING_DATASOURCE_PASSWORD:  
    healthcheck:  
      test: "/usr/bin/mysql --user=root --password=root --execute \"SHOW DATABASES;\""   
      interval: 5s  
      timeout: 2s  
      retries: 100  
  
  auth-service:  
    build:  
      context: ./authentification-service  
    ports:  
      - "8099:8099"  
    depends_on:  
      - mysql  
  
    networks:  
      - microservices-network  
    environment:
```



```
SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/tnb?createDatabaseIfNotExist=true
SPRING_DATASOURCE_USERNAME: root
SPRING_DATASOURCE_PASSWORD:
healthcheck:
  test: "/usr/bin/mysql --user=root --password=root --execute \"SHOW DATABASES;\""
  interval: 5s
  timeout: 2s
  retries: 100

phpmyadmin:
  image: phpmyadmin/phpmyadmin
  environment:
    PMA_HOST: mysql
    PMA_PORT: 3306
    MYSQL_ROOT_PASSWORD: root
  ports:
    - "8081:80"
  networks:
    - microservices-network

networks:
  microservices-network:
    driver: bridge
```

V. CI/CD avec Jenkins :

1. Démarrage de Jenkins :

Une fois l'installation terminée, démarrez le service Jenkins.

Accédez à l'interface utilisateur de Jenkins à l'aide de votre navigateur en ouvrant l'URL `http://localhost:8080` (par défaut).

2. Déverrouillage initial :

Lors de la première exécution, Jenkins affichera une clé générée pour déverrouiller l'installation. Suivez les instructions pour trouver cette clé dans les logs de Jenkins.

3. Configuration initiale :

Configurez Jenkins en fournissant les informations nécessaires telles que le nom de l'administrateur, le mot de passe, et les paramètres du serveur.

4. Installation de plugins :

Choisissez les plugins nécessaires pour votre environnement. Les plugins couramment utilisés incluent ceux pour Git, Docker, Maven, etc.

Accédez à "Gérer Jenkins" -> "Gérer les plugins" -> "Disponible" pour installer de nouveaux plugins.

5. Configuration des outils :

Allez à "Gérer Jenkins" -> "Configurer le système".

Configurez les outils tels que JDK, Git, Maven, Docker, etc., en ajoutant les chemins d'installation ou les configurations nécessaires.

6. Création d'un nouvel emploi (Job) :

Cliquez sur "Nouvel emploi" sur le tableau de bord de Jenkins.

Donnez un nom à l'emploi et sélectionnez le type de projet (freestyle, pipeline, etc.).

7. Configuration de l'emploi :

Configurez les détails spécifiques à l'emploi, tels que la source de code (Git, SVN), les déclencheurs de build, les actions post-build, etc.

8. Configuration du Build :

Ajoutez les étapes de build en fonction de votre projet (ex. : compilation, tests, emballage, etc.).

Utilisez des outils tels que Maven, Gradle, ou des scripts shell pour les étapes de build.

9. Configuration des notifications :

Configurez les notifications de build en cas de succès ou d'échec. Vous pouvez envoyer des e-mails, des notifications Slack, etc.

10. Configuration des déclencheurs de build :

Choisissez quand déclencher un build, que ce soit après chaque commit, à intervalles réguliers, ou manuellement.

11. Enregistrement des modifications :

Sauvegardez toutes les configurations en cliquant sur "Enregistrer" ou "Appliquer".

12. Exécution d'un Build :

Démarrez manuellement le build pour tester la configuration.

13. Configuration de la CI/CD avec Jenkinsfile (Pipeline) :

Pour une approche plus avancée, créez un Jenkinsfile pour définir votre pipeline en tant que code. Stockez-le dans votre référentiel de code.

14. Configuration des plugins additionnels :

Si nécessaire, configurez des plugins additionnels pour des fonctionnalités spécifiques, tels que des rapports de test, des analyses statiques de code, etc.

15. Sécurité et gestion des utilisateurs :

Configurez la sécurité de Jenkins et gérez les utilisateurs en accédant à "Gérer Jenkins" ->

"Configurer la sécurité".

Configuration :

```
pipeline {
```

```
    agent any
```

```
    tools {
```

```
        maven 'maven'
```

```
    }
```

```
    stages {
```

```
        stage('Git Clone') {
```

```
            steps {
```

```
                script {
```

```
                    checkout([$class: 'GitSCM', branches: [[name: 'master']],
```

```
userRemoteConfigs: [[url: 'https://github.com/zinebnaciri/TNBprojectJ.git']]])
```

```
                }
```

```
            }
```

```
        }
```

```
        stage('Build authentication-service') {
```

```
            steps {
```

```
                script {
```

```
                    dir('authentification-service') {
```

```
                        bat 'mvn clean install'
```

```
                    }
```

```
                }
```

```
            }
```

```
        }
```

```
        stage('SonarQube Analysis Auth') {
```

```
            steps {
```

```
                script {
```

```
                    dir('authentification-service') {
```

```
        withSonarQubeEnv('sq1') {  
            bat 'mvn sonar:sonar'  
        }  
    }  
}  
}
```

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

```
stage('SonarQube Analysis Tnb') {  
    steps {  
        script {  
            dir('tnb-maroc') {  
                withSonarQubeEnv('sq1') {  
                    bat 'mvn sonar:sonar'  
                }  
            }  
        }  
    }  
}
```

```
stage('Create Docker Images') {  
    steps {  
        script {  
            bat 'docker build -t catmilk/auth-service authentication-service'        }  
    }  
}
```

```

    bat 'docker build -t catmilk/tnb-service tnb-maroc'
  }
}
}

stage('Run') {
  steps {
    script {
      bat "docker run --name Auth -d -p 8090:8090 catmilk/auth-service"
      bat "docker run --name Tnb -d -p 8080:8080 catmilk/tnb-service"
    }
  }
}
}
}

```

Stage View

	Declarative: Tool Install	Git Clone	Build authentication- service	SonarQube Analysis Auth	Build tnb- maroc	SonarQube Analysis Tnb	Create Docker Images	Run
Average stage times: (Average full run time: ~13min 30s)	926ms	13s	1min 33s	2min 5s	35s	49s	58s	52s
#16 janv. 18 08:54 2 commits	1s	11s	1min 24s	1min 52s	1min 44s	2min 28s	2min 56s	2min 36s
#15								

VI. Intégration de SonarQube

1. Configuration

Tableau de bord > Administrer Jenkins > System >

SonarQube servers

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

☒ Environment variables

Installations de SonarQube

Liste des installations de SonarQube

Nom

URL du serveur
Par défaut à http://localhost:9000


Server authentication token
SonarQube authentication token. Mandatory when anonymous access is disabled.

+ Ajouter

Activate Windows
Go to Settings to activate


Enregistrer Appliquer


```
stage('SonarQube Analysis Tnb') {
    steps {
        script {
            dir('tnb-maroc') {
                withSonarQubeEnv('sq1') {
                    bat 'mvn sonar:sonar'
                }
            }
        }
    }
}
```





Jenkins


Tableau de bord > tnbProject >


 Status


 **tnbProject**


 Changes


 Lancer un build


 Configurer


 Supprimer Pipeline


 Full Stage View


 GitHub

 SonarQube

 SonarQube

 Renommer

 Pipeline Syntax

 GitHub Hook Log

Stage View

Average stage
(Average full run time: ~

#16	janv. 18 08:54	2 commits
#15	janv. 18 08:47	No Changes

2. bénéfices pour la qualité du code :

L'utilisation de SonarQube offre plusieurs avantages significatifs pour garantir la qualité du code au sein d'un projet logiciel. Voici quelques-uns des bénéfices clés associés à l'utilisation de SonarQube :

Détection Précoce des Problèmes de Code :

SonarQube analyse le code source et identifie rapidement les problèmes potentiels, tels que

les violations des règles de codage, les erreurs, les duplications de code, etc.

La détection précoce permet de corriger les problèmes avant qu'ils ne se propagent ou ne deviennent plus difficiles à résoudre.

Standardisation du Code :

SonarQube applique des règles de qualité prédéfinies, ce qui favorise la standardisation du code au sein de l'équipe de développement.

L'application cohérente de règles de codage améliore la lisibilité du code, facilite la maintenance et réduit les risques d'erreurs.

Mesure de la Complexité du Code :

SonarQube fournit des métriques de complexité du code, telles que la complexité cyclomatique, permettant d'évaluer la facilité de compréhension et de maintenance du code. Cela aide à identifier les parties du code nécessitant une attention particulière et à éviter la surcomplexité.

Gestion des Duplications de Code :

SonarQube identifie les duplications de code, facilitant la réduction de la redondance et garantissant une base de code plus propre et plus facile à entretenir.

Amélioration de la Conformité aux Standards Industriels :

En intégrant des règles basées sur des normes de codage (comme MISRA, CWE, etc.), SonarQube permet de garantir la conformité aux meilleures pratiques et aux normes industrielles.

Intégration dans le Processus CI/CD :

SonarQube peut être intégré dans les pipelines d'intégration continue (CI) pour automatiser l'analyse du code à chaque modification.

L'intégration continue des analyses contribue à maintenir une qualité élevée du code tout au long du développement.

Tableau de Bord et Rapports Visuels :

SonarQube propose des tableaux de bord interactifs et des rapports visuels fournissant une vue globale de la qualité du code.

Ces rapports facilitent la communication au sein de l'équipe et avec les parties prenantes.

Analyse de la Sécurité du Code :

SonarQube peut identifier les vulnérabilités de sécurité dans le code, contribuant ainsi à renforcer la sécurité du logiciel.

Les analyses statiques de sécurité aident à prévenir les problèmes de sécurité potentiels avant le déploiement.

Traçabilité des Évolutions du Code :

SonarQube conserve un historique des analyses, permettant de suivre l'évolution de la qualité du code au fil du temps.

La traçabilité facilite l'identification des tendances et la mesure de l'impact des modifications sur la qualité du code.

VII. Conclusion :

Durant la phase du développement, on conclut l'objectif visé à travers de ce rapport est de présenter le système de gestion d'organisation des événements réalisé au cours de notre projet de fin d'année.

Ce projet nous a permis de compléter nos connaissances en programmation informatique et surtout les nouvelles technologies et langages de développement pour réussir notre projet.

Le projet comme il a été démontré s'est articulé autour de quatre chapitres principaux. Le premier chapitre de ce rapport a été consacré à une présentation générale du contexte du projet et la description des besoins fonctionnels. Et pour le deuxième chapitre, nous avons présenté une analyse et conception du système suivant la démarche de langage UML. Ce qui concerne le dernier chapitre, nous avons montré dans une première étape les différents outils et technologies évoluées qui ont permis d'implémenter notre système, et dans une seconde étape, nous avons présenté notre projet accompagné des exemples d'interfaces élaborées avec des captures d'écran.

Enfin, la réalisation de ce projet a été très bénéfique, car elle nous a permis d'apprendre des nouvelles connaissances (surtout techniques) en programmation informatique et de découvrir le domaine de travail.

VII. Conclusion

Durant la phase du développement, on conclut l'objectif visé à travers de ce rapport est de présenter le système de gestion d'organisation des événements réalisé au cours de notre projet de fin d'année.