

Réalisé par :

Samih Aicha

Youssef elyourizi

youssef elmouden

Rapport Projet : Architecture des composants d'entreprise

1. Introduction :

- Aperçu du projet : Le projet de calcul de la Taxe (TNB) vise à créer un système informatisé permettant le calcul automatisé des taxes, en mettant l'accent sur une architecture microservices. Cette solution se compose de deux microservices côté backend développés en utilisant Spring Boot, à savoir le microservice principal de calcul de taxe (Backend) et un microservice de gestion de la sécurité (Spring Security). Du côté frontend, l'interface utilisateur est développée en utilisant Angular.
- Description du Projet
- Architecture Microservices avec Spring Boot :
 - Le microservice principal (Backend) est chargé de la logique de calcul de la Taxe TNB. Il gère les entités telles que le propriétaire, le taux de taxe, la catégorie de bien, le type de terrain, et la taxe elle-même.
 - Le microservice Spring Security s'occupe de la gestion de la sécurité, y compris l'authentification et l'autorisation. Il garantit que l'accès aux fonctionnalités du microservice principal est sécurisé et contrôlé.

- Frontend - Angular :
- Interface Utilisateur Conviviale :

Développement d'une interface utilisateur attrayante et conviviale pour permettre aux utilisateurs de saisir les détails nécessaires pour le calcul de la taxe.

- Communication avec les Microservices :

Intégration d'Angular avec les microservices backend pour récupérer les données nécessaires pour le calcul de la taxe et afficher les résultats.

- Gestion des Utilisateurs :
Intégration avec le microservice Spring Security pour gérer l'authentification et l'autorisation des utilisateurs.

2&3 - Architecture et conception des Microservices

- Service Entités (Microservice 1) :

Responsabilités : Gère les entités, y compris l'entité "Redevable".

Technologies: Spring Boot avec API REST.

Communication : Expose des points d'API REST pour permettre aux autres services d'accéder et de manipuler les données des entités. Utilise Apache Kafka en tant que consommateur pour recevoir des événements liés à la sécurité du service Spring Security.

- Service Spring Security (Microservice 2) :

Responsabilités : Gère la sécurité de l'application, y compris les rôles des utilisateurs tels que "Admin".

Technologies: Spring Boot avec Spring Security.

Communication : Agit en tant que producteur Kafka pour informer le Service Entités des événements liés à la création de nouveaux utilisateurs (Redevables). Utilise également Apache Kafka pour recevoir des événements liés à la sécurité du service Entités.

Mécanismes de Communication :

- Communication Synchrone (REST) :

Le Service Entités expose des API REST pour permettre aux autres services d'accéder et de manipuler les données des entités.

Exemple d'interaction : Le Service Spring Security peut faire des requêtes REST au Service Entités pour obtenir des informations sur les Redevables.

- Communication Asynchrone (Apache Kafka) :

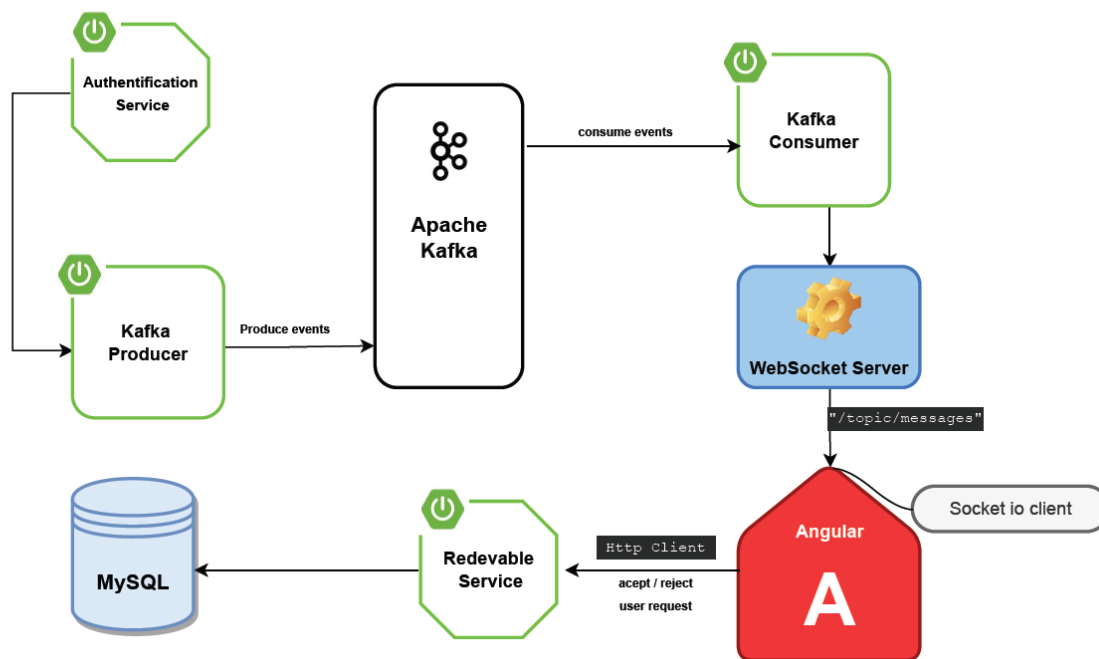
Le Service Entités agit en tant que consommateur Kafka pour traiter les événements liés à la sécurité du Service Spring Security.

Le Service Spring Security agit en tant que producteur Kafka pour informer le Service Entités des événements liés à la création de nouveaux Redevables.

- WebSocket :

Utilisation de WebSocket pour permettre à l'administrateur d'accepter ou de refuser la création d'un nouvel utilisateur (Redevable).

Exemple d'interaction : Lorsqu'un Redevable est créé, le Service Entités envoie une notification via WebSocket au front-end de l'administrateur. L'administrateur peut alors accepter ou refuser cette création en temps réel.



4- Conteneurisation avec Docker

1. Création de Conteneurs :

Chaque microservice (Service Entités et Service Spring Security) est encapsulé dans un conteneur Docker distinct.

Nous avons utilisé un fichier Dockerfile pour décrire les dépendances et les étapes nécessaires à la création du conteneur.

2. Images Docker :

Nous avons Construis des images Docker à partir des fichiers Dockerfile pour chaque microservice.

Nous avons utilisé des images légères et optimisées pour garantir une efficacité de l'espace disque et des performances.

3. Orchestration avec Docker Compose :

Nous avons utilisé Docker Compose pour définir et gérer l'environnement d'exécution de plusieurs conteneurs.

```
1 # Stage 1: Build with Maven
2 FROM maven:3.8.4-openjdk-17 AS builder
3 WORKDIR /app
4 COPY ./src ./src
5 COPY ./pom.xml .
6 RUN mvn clean package -DskipTests
7
8 # Stage 2: Create the final image
9
10 FROM openjdk:17-jdk-alpine
11 VOLUME /tmp
12 ARG JAR_FILE=target/*.jar
13 COPY ${JAR_FILE} app.jar
14 ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Dockerfile pour le microservice 1

```
# 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 -DskipTests

# 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"]
```

Dockerfile pour le microservice 2

Pour le fichier Docker-compose.yml :

```
version: '3'
services:
  mysql:
    image: mysql:latest
    container_name: mysql-tax
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_ALLOW_EMPTY_PASSWORD: true
      MYSQL_DATABASE: tax
    ports:
      - "3306:3306"

  backend-auth:
    container_name: tax-auth
    build:
      context: ./tax-msAuth
    ports:
      - "8090:8090"
    depends_on:
      mysql:
        condition: service_completed_successfully
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/tax?createDatabaseIfNotExist=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
```

```
backend-tax:
  container_name: tax-back
  build:
    context: ./tax
  ports:
    - "8084:8084"
  depends_on:
    mysql:
      condition: service_completed_successfully
  environment:
    SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/tax?createDatabaseIfNotExist=true
    SPRING_DATASOURCE_USERNAME: root
    SPRING_DATASOURCE_PASSWORD: root

frontend:
  container_name: tax-front
  build:
    context: ./tax-front
  ports:
    - "8095:80"
  depends_on:
    backend-auth:
      condition: service_completed_successfully
    backend-tax:

zookeeper:
  container_name: zookeeper-container
  image: confluentinc/cp-zookeeper:latest
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
  ports:
    - "22181:2181"

kafka:
  container_name: kafka-container
  image: confluentinc/cp-kafka:latest
  depends_on:
    - zookeeper
  ports:
    - "9092:9092"
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:29092
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

5. CI/CD avec Jenkins

- Processus et configuration

Jenkins

Jenkins (<https://www.jenkins.io/>) est un outil open source d'automatisation des tâches, principalement utilisé pour automatiser la construction, les tests et le déploiement de logiciels. Il offre une intégration continue et peut être utilisé pour automatiser toute tâche récurrente liée au processus de développement logiciel.

- Intégration Continue (CI) : Jenkins permet d'automatiser le processus d'intégration continue en détectant automatiquement les modifications dans le code source et en lançant des builds automatiques et des tests;
- Gestion des Builds et Déploiements : Jenkins prend en charge la gestion des builds de logiciels, la gestion des dépendances, et peut également être utilisé pour déployer automatiquement des applications;

Description du Pipeline Jenkins

Le pipeline Jenkins fourni est un script en langage Groovy utilisé pour automatiser les étapes du cycle de vie du développement logiciel. Voici une explication détaillée de chaque étape du pipeline :

1. Git Clone (Étape de clonage Git) :

- Cette étape utilise le plugin GitSCM pour cloner le dépôt GitHub :

<https://github.com/youssefelyou/TaxTNB.git>

- Le script récupère la branche principale (main) du dépôt.

2. Build (Étape de construction) : pour build ici on fait le build des micro service et partie front end aussi

- Cette étape utilise Maven (outil de gestion de projet Java) pour effectuer une construction propre et une installation du projet.
- La commande Maven exécutée est : `mvn clean install`.

3. Create Docker Image (Étape de création de l'image Docker) :

- Cette étape utilise Docker pour créer une image du projet.
- La commande Docker exécutée est : `'docker-compose build'`

4. Run (Étape d'exécution) :

- Cette étape utilise Docker pour exécuter le conteneur de l'application

La commande Docker exécutée est : `bat "docker run --name tax-msAuth -d -p 8090:8090 elmoudene/taxtnb-backend-auth"`

- L'option `-d` signifie que le conteneur s'exécute en arrière-plan, et l'option `-p` mappe le port 8090 du conteneur sur le port 8090 de l'hôte.

Ce pipeline automatise le processus de récupération du code source depuis GitHub, la construction du projet avec Maven, la création d'une image Docker, et enfin, l'exécution de l'application dans un conteneur Docker. Il s'agit d'un exemple de

pipeline d'intégration continue (CI) et de déploiement continu (CD) pour un projet Java avec Docker
voilà le script :

```
pipeline {
  agent any
  tools {
    maven 'maven'
  }
  stages {
    stage('Git Clone') {
      steps {
        script {
          checkout([$class: 'GitSCM', branches: [[name: 'main']],
userRemoteConfigs: [[url: 'https://github.com/youssefelyou/TaxTNB.git']]])
        }
      }
    }
    stage('Build tnbAuth') {
      steps {
        script {
          dir('tax-msAuth') {
            bat './mvnw clean install'
          }
        }
      }
    }
    stage('Build tax') {
      steps {
        script {
          dir('tax') {
            bat './mvnw clean install -DskipTests'
          }
        }
      }
    }
    stage('Build tax-front') {
      steps {
        script {
          dir('tax-front') {
            bat 'npm install'
            bat 'npm run build'
          }
        }
      }
    }
  }
}
```



```

    }
  }
}

stage('Build Docker Images') {
  steps {
    script {
      bat 'docker-compose build'

    }
  }
}

```

```

stage('Run') {
  steps {
    script {
      bat "docker run --name tax-msAuth -d -p 8090:8090 elmoudene/taxtnb-backend-auth"
      bat "docker run --name tax-front -d -p 8095:80 elmoudene/taxtnb-frontend"
      bat "docker run --name tax -d -p 8084:8084 elmoudene/taxtnb-backend-tax"
    }
  }
}
}}}

```

Jenkins 1 samih aicha se déconnecter

Tableau de bord > test2 >

Status

</> Changes

Lancer un build

Configurer

Supprimer Pipeline

Full Stage View

GitHub

Renommer

Pipeline Syntax

GitHub Hook Log

✓ **test2**

microservices with jenkins

modifier la description

Désactiver le projet

Stage View

Average stage times: (Average full run time: ~5min 10s)

	Declarative: Tool Install	Git Clone	Build tnbAuth	Build tax	Build tax-front	Build Docker Images	Run
Average	191ms	2s	16s	15s	2min 23s	3min 18s	5s
#32 Janv. 18 09:35 No Changes	194ms	2s	15s	14s	2min 9s	1min 3s	4s

Historique des builds test2

6. Déploiement Automatique

Le déploiement automatique peut être réalisé de différentes manières en fonction de la technologie que vous utilisez. Vous avez mentionné l'utilisation de Ngrok ou Azure Cloud, qui sont deux solutions différentes. Voici comment vous pourriez aborder le déploiement automatique avec chacune de ces options :

Déploiement Automatique avec Ngrok :

Installation de Ngrok :

Assurez-vous d'installer Ngrok sur le serveur où votre application est hébergée.

Configuration de Ngrok :

Utilisez la commande Ngrok pour exposer votre application locale via un tunnel sécurisé. Par exemple : `ngrok http 3000`


Ngrok générera une URL publique qui redirige vers votre application locale.

Intégration dans le Pipeline :

Dans votre pipeline de déploiement, ajoutez une étape pour lancer Ngrok avec la commande appropriée.

Stockez l'URL générée par Ngrok en tant que variable d'environnement.

7. Intégration de SonarQube :



The screenshot shows the Jenkins web interface. At the top, the Jenkins logo and a search bar are visible. Below the header, the breadcrumb trail reads 'Tableau de bord > Administrer Jenkins > Plugins'. The main section is titled 'Plugins' and contains a search bar with 'sonar' entered. On the left, there are filters for 'Mises à jour' (17), 'Plugins disponibles', and 'Plugins installés'. The 'Plugins installés' filter is selected. The main list shows the 'SonarQube Scanner for Jenkins' plugin, version 2.17.1, which is marked as 'Activé'. A description of the plugin is provided, along with a link to 'Report an issue with this plugin'.

On commence d'abord par install plugins ' sonarQube Scanner for jenkins' et restart jenkins

Et voilà le resultat de sonarQube et 0 code smells

 [Youssef Elyourizi](#) / [taxTNB](#) NEW PUBLIC ✓ Passed

Last analysis: 1/17/2024, 9:43 PM • 721 Lines of Code • Java, XML

A 0
Bugs

A 0
Vulnerabilities

A 100%
Hotspots Reviewed

A 0
Code Smells

0 0.0%
Coverage

0 0.0%
Duplications

Tableau de bord > Administrer Jenkins > System >

Nom

taxtnb

URL du serveur

Par défaut à http://localhost:9000

http://localhost:9000/

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

token-tnb

+ Ajouter ▾


Avancé ▾

Enregistrer

Appliquer

Et on ajoute dans le script de pipeline :

```
stage('SonarQube Tnb') {  
    steps {  
        script {  
            dir('tax') {  
                withSonarQubeEnv('sq1') {  
                    bat 'mvn sonar:sonar'  
                }  
            }  
        }  
    }  
}
```


Jenkins



 1
  samih aïcha
 [se déconnecter](#)

Tableau de bord

>

test2

Status

Changes

Lancer un build

Configurer

Supprimer Pipeline


Full Stage View

GitHub

Renommer

Pipeline Syntax

GitHub Hook Log


test2

microservices with jenkins

[modifier la description](#)
[Désactiver le projet](#)

Stage View

Average stage times:

(Average full run time: ~3min 45s)

	Declarative: Tool Install	Git Clone	Build tnbAuth	Build tax	Build tax-front	SonarQube Tnb	Build Docker Images	Run
	206ms	2s	22s	18s	2min 24s	679ms	14s	929ms
#35 janv. 21 00:42	121ms	1s	11s	13s	2min 29s	540ms	44s	2s

- Résumé des accomplissements

- Perspectives futures

Déploiement Automatique AVEC Ngrok