

Rapport Projet :

Architecture des composants d'entreprise

Realisé par

EL ASRI MOHAMED Taha

NAJAH AHlAM

BENATTIK Amal

1. Introduction

- Aperçu du projet
- Importance de l'architecture microservices

2. Architecture Microservices

- Architecture
- Description des services
- Mécanismes de communication

3. Conception des Microservices

- Approche de conception pour chaque service

4. Conteneurisation avec Docker

- Implémentation et avantages

5. CI/CD avec Jenkins

- Processus et configuration

6. Déploiement Automatique

- Utilisation de Ngrok ou Azure Cloud

7. Intégration de SonarQube

- Configuration et bénéfices pour la qualité du code

8. Conclusion

- Résumé des accomplissements
- Perspectives future

1. Introduction

Aperçu du Projet

Le projet vise à mettre en place un système de gestion de la Taxe sur les Terrains Non Bâties (TNB) au Maroc, avec un accent particulier sur l'architecture des composants d'entreprise, tout en basant sur une base de sécurité, avec une gestion efficace des terrains, taux, taxes, redevable(user), catégorie, et une option de consultation et de paiement des Taxes.

Importance de l'Architecture Microservices

L'adoption d'une architecture micro services permet une flexibilité accrue, une évolutivité facilitée, et une maintenance simplifiée des différents modules du système, en utilisant des mécanismes de Communication synchrone et asynchrone.

2. Architecture Microservices

Architecture

Le système sera basé sur une architecture microservices, favorisant la modularité et la séparation des préoccupations. Chaque composant aura des responsabilités spécifiques, facilitant le développement, la maintenance et le déploiement.

Description des Services

Les services seront définis en fonction des exigences du cahier des charges :

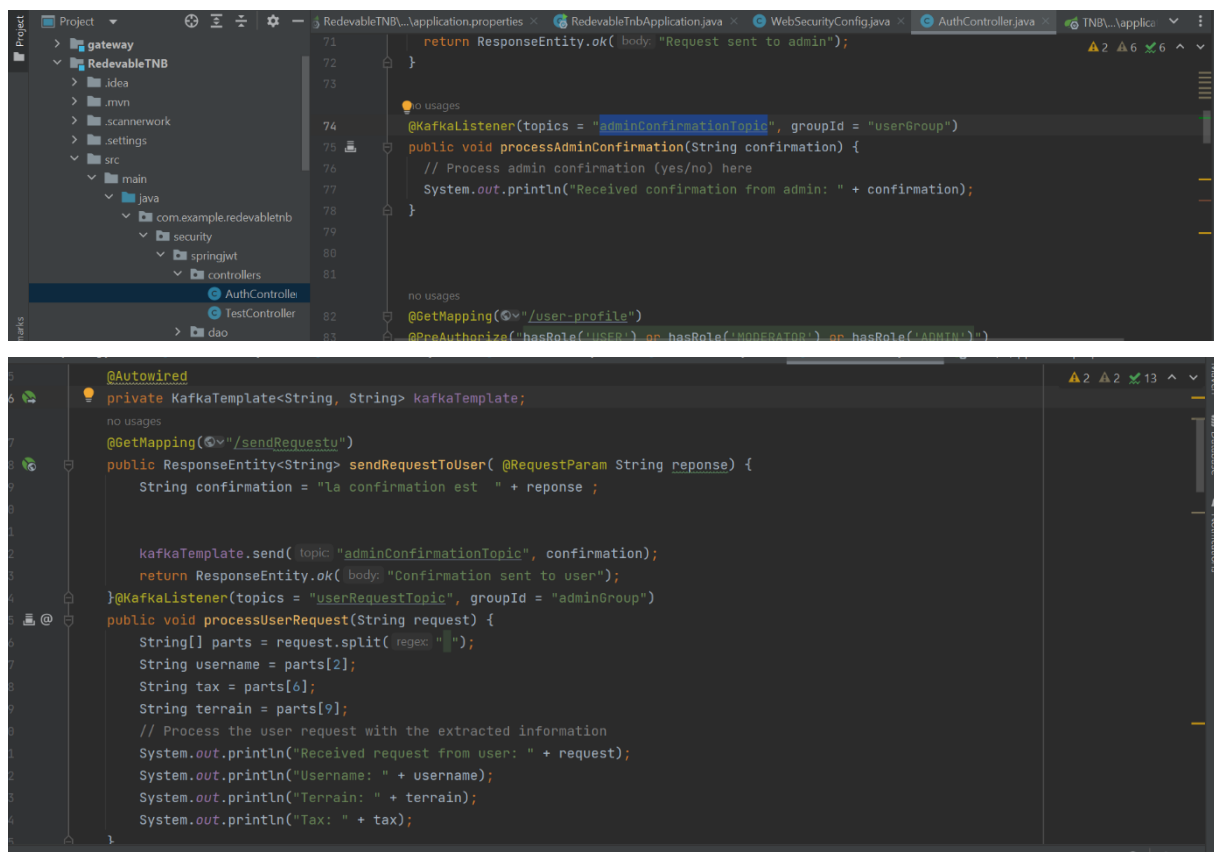
- Service de gestion des catégories de terrains
- Service de gestion des terrain
- Service de gestion des taux de terrain
- Service de calcul de la taxe TNB
- Service de gestion de redevable.

Mécanismes de Communication

Dans le cadre du projet, les mécanismes de communication choisis incluent Kafka et REST Template.

- **Kafka** : Utilisé comme système de messagerie pour permettre la communication asynchrone entre les microservices. Les événements liés aux mises à jour des taxes ou aux changements de status de paiement seront émis via Kafka, offrant une scalabilité et une extensibilité efficaces.
- **REST Template** : Adopté pour les interactions synchrones entre les services. Les appels directs, tels que la recherche par CIN, la consommation des méthode user pour spring security seront effectués via des requêtes REST, fournissant une communication légère et simple entre les services.

Les captures : kafka et resttemplate code



The image consists of two screenshots of an IDE, likely IntelliJ IDEA, showing Java code. The top screenshot shows the 'AuthController.java' file. It contains a method 'processAdminConfirmation' that uses a 'KafkaTemplate' to send a confirmation message to a Kafka topic. The bottom screenshot shows the 'TestController.java' file. It contains a method 'sendRequestToUser' that uses a 'KafkaTemplate' to send a request message to a Kafka topic. Both screenshots show the project structure on the left, with the 'AuthController' and 'TestController' classes highlighted.

```
71 return ResponseEntity.ok( body: "Request sent to admin");
72 }
73
74 @KafkaListener(topics = "adminConfirmationTopic", groupId = "userGroup")
75 public void processAdminConfirmation(String confirmation) {
76     // Process admin confirmation (yes/no) here
77     System.out.println("Received confirmation from admin: " + confirmation);
78 }
79
80
81
82 @GetMapping("/user-profile")
83 @PreAuthorize("hasRole('USER') or hasRole('ADMIN') or hasRole('AUTH')")
```

```
6 @Autowired
7 private KafkaTemplate<String, String> kafkaTemplate;
8
9 @GetMapping("/sendRequest")
10 public ResponseEntity<String> sendRequestToUser( @RequestParam String reponse) {
11     String confirmation = "la confirmation est " + reponse ;
12
13     kafkaTemplate.send( topic: "adminConfirmationTopic", confirmation);
14     return ResponseEntity.ok( body: "Confirmation sent to user");
15 }
16 @KafkaListener(topics = "userRequestTopic", groupId = "adminGroup")
17 public void processUserRequest(String request) {
18     String[] parts = request.split( regex: " ");
19     String username = parts[2];
20     String tax = parts[6];
21     String terrain = parts[9];
22     // Process the user request with the extracted information
23     System.out.println("Received request from user: " + request);
24     System.out.println("Username: " + username);
25     System.out.println("Terrain: " + terrain);
26     System.out.println("Tax: " + tax);
27 }
```

3. Conception des Microservices

Les microservices seront conçus en suivant les principes SOLID, et la conception se déclinera en quatre principaux microservices :

1. Eureka Server :

- **Responsabilités** : Enregistrement et découverte des microservices dans l'écosystème.
- **Technologies** : Spring Cloud Eureka.
- **Objectifs** : Assurer la disponibilité des microservices et faciliter leur communication.

2. Gateway :

- **Responsabilités** : Gestion des requêtes d'entrée, routage vers les microservices appropriés.
- **Technologies** : Spring Cloud Gateway.
- **Objectifs** : Centraliser le point d'entrée de l'application, gérer l'authentification et l'autorisation.

3. Redeable User :

- **Responsabilités** : Gérer les informations liées aux redevables, rechercher par CIN, et fournir les historiques de taxes annuelles.

- **Technologies** : Spring Boot, Hibernate.
- **Objectifs** : Faciliter l'accès aux données individuelles des redevables.

4. Admin :

- **Responsabilités** : Gérer les catégories de terrains, les taux de terrain, et effectuer le calcul de la taxe TNB.
- **Technologies** : Spring Boot, Hibernate.
- **Objectifs** : Centraliser les fonctionnalités administratives et assurer la cohérence des données.

The image consists of two screenshots of an IDE, likely IntelliJ IDEA, showing Java code and application logs.

Top Screenshot: The IDE shows a project named 'eurekaserver'. The main class is 'EurekaserverApplication'. The code is as follows:

```

import ...

@EnableEurekaServer
@SpringBootApplication
public class EurekaserverApplication {

    public static void main(String[] args) { SpringApplication.run(EurekaserverApplication.class, args); }
}

```

The console shows logs for the application running on port 8761. The logs indicate that the application is running successfully and that the Eureka server is up and running.

Bottom Screenshot: The IDE shows a project named 'Back-End'. The main class is 'GatewayApplication'. The code is as follows:

```

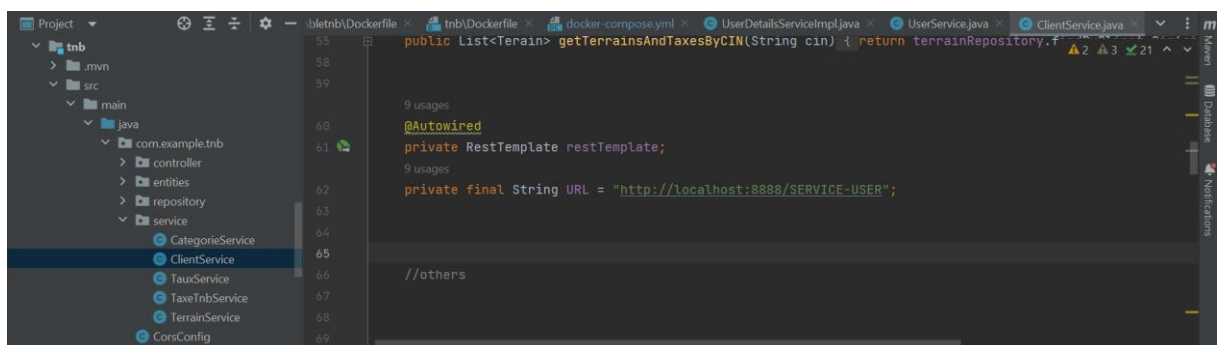
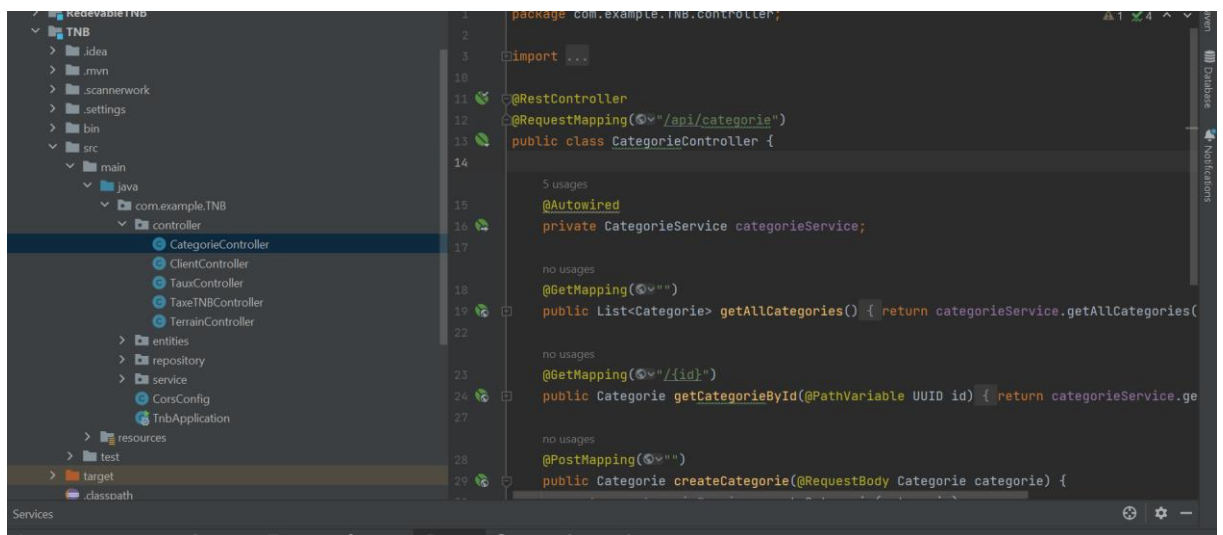
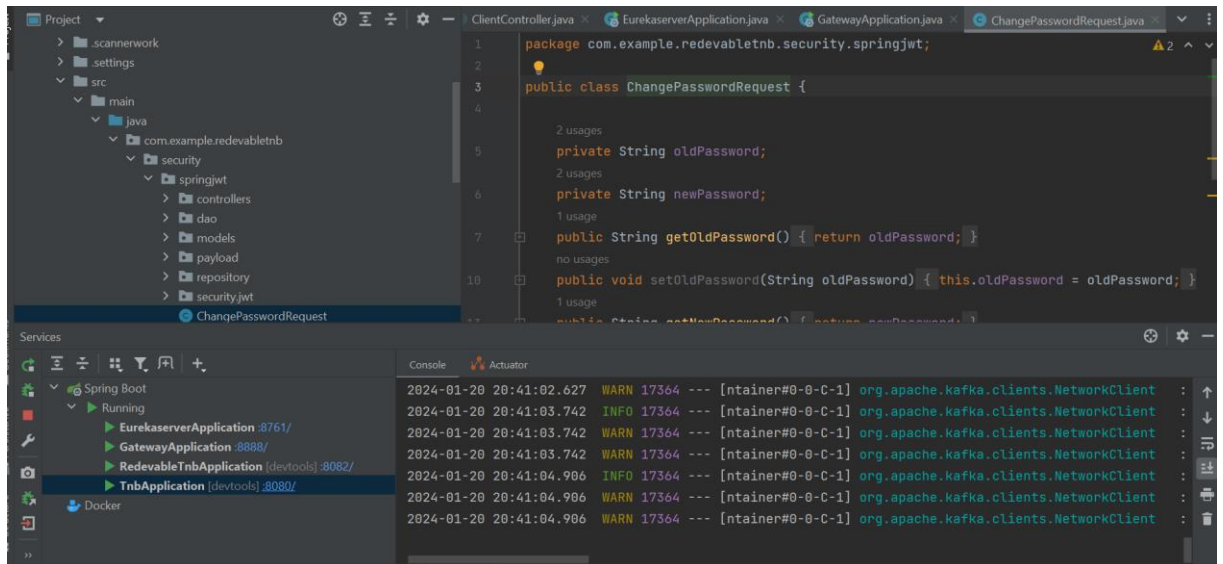
public class GatewayApplication {

    public static void main(String[] args) { SpringApplication.run(GatewayApplication.class, args); }

    @Bean
    DiscoveryClientRouteDefinitionLocator routesDynamic(
        ReactiveDiscoveryClient reactiveDiscoveryClient,
        DiscoveryLocatorProperties discoveryLocatorProperties){
        return new DiscoveryClientRouteDefinitionLocator(reactiveDiscoveryClient, discoveryLocatorProperties);
    }
}

```

The console shows logs for the application running on port 8888. The logs indicate that the application is running successfully and that the Gateway is up and running.



4. Conteneurisation avec Docker

Implémentation et Avantages

Les microservices seront conteneurisés à l'aide de Docker pour assurer la portabilité, la facilité de déploiement et la gestion des dépendances

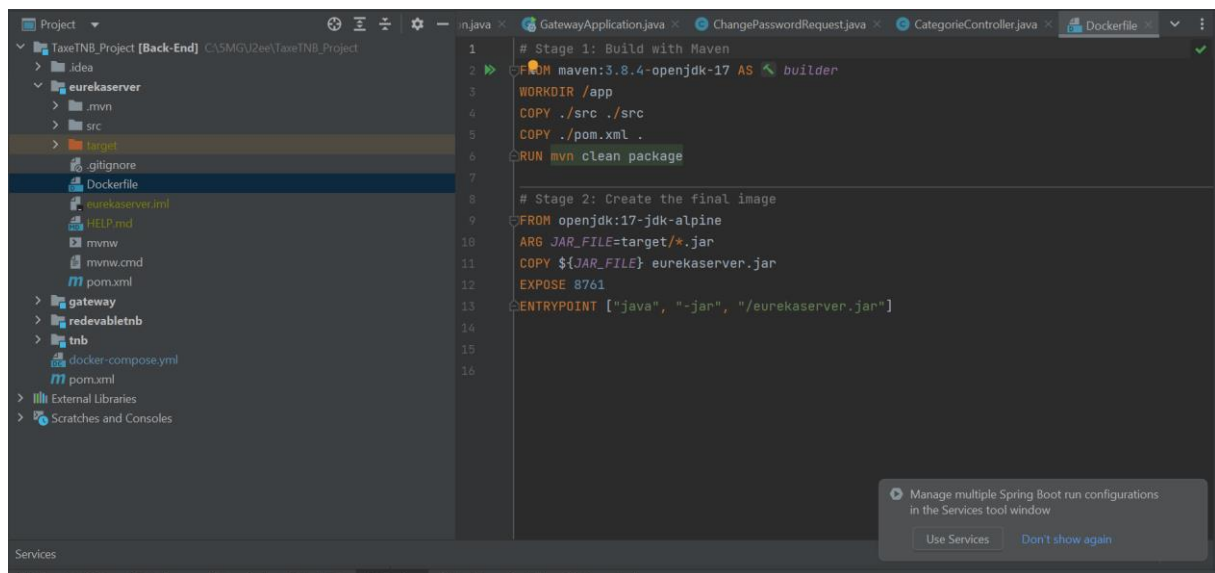
Le Dockerfile est un script qui contient des instructions pour la construction d'une image Docker. Chaque microservice dispose de son propre Dockerfile, décrivant les étapes nécessaires pour créer une image contenant l'application et ses dépendances. Voici une brève explication des sections clés d'un Dockerfile :

Stage 1: Build with Maven (ou autre outil de build) :

- Dans cette étape, l'image Maven est utilisée comme base pour construire l'application Java. Toutes les sources et les dépendances nécessaires sont copiées dans le conteneur. Ensuite, la commande **mvn clean package** est exécutée pour compiler l'application et créer le fichier JAR.

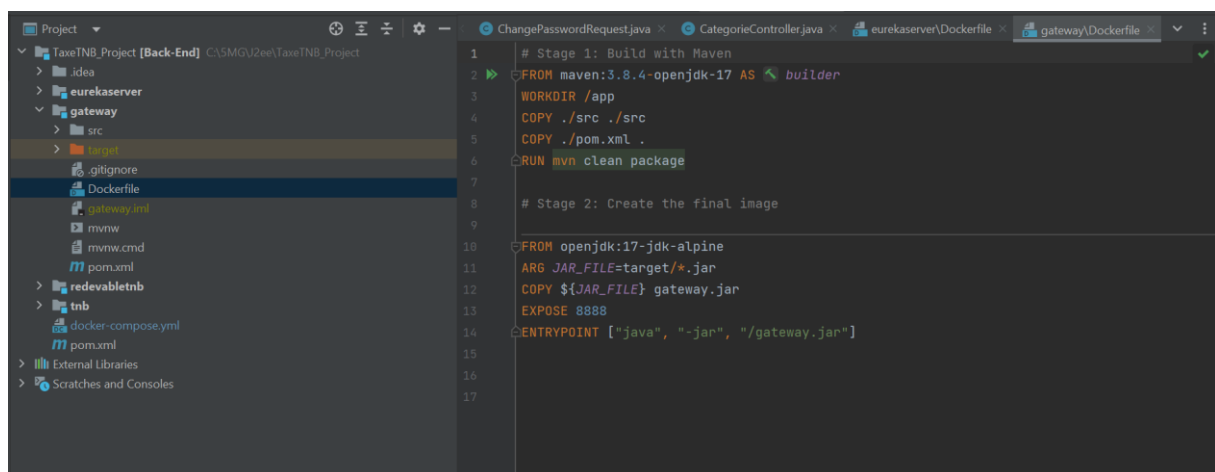
Stage 2: Create the final image :

- Une nouvelle image est créée à partir d'une image légère basée sur OpenJDK, et le fichier JAR construit à l'étape précédente est copié dans cette nouvelle image. L'**ENTRYPOINT** spécifie la commande à exécuter lorsque le conteneur démarre.



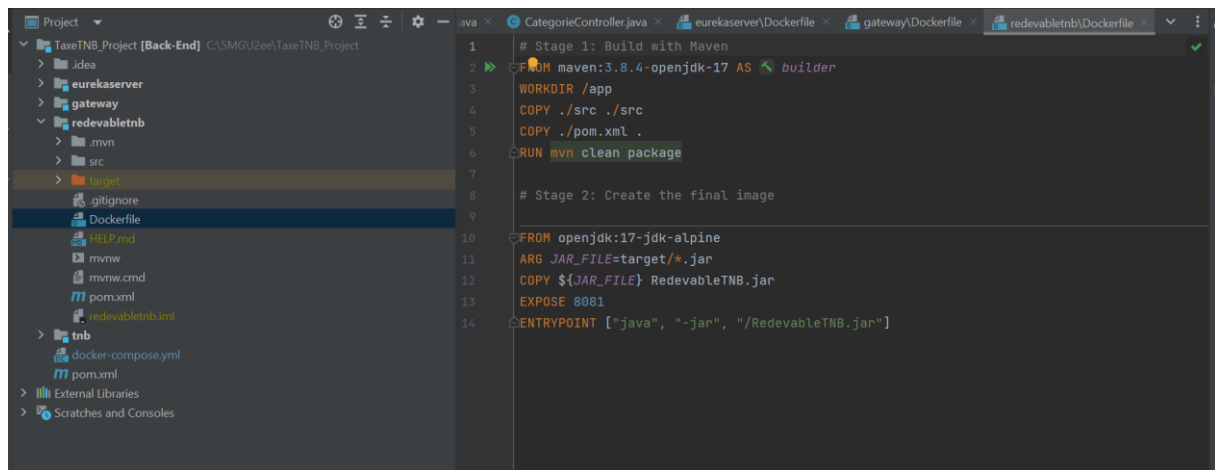
The screenshot shows an IDE with a project named 'TaxeTNB_Project [Back-End]'. The file explorer on the left shows the project structure, including a 'eurekaserver' directory with files like 'pom.xml', 'Dockerfile', and 'HELP.md'. The main editor displays the content of the 'eurekaserver/Dockerfile' file. The Dockerfile contains two stages: Stage 1 for building with Maven and Stage 2 for creating the final image.

```
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
7
8 # Stage 2: Create the final image
9 FROM openjdk:17-jdk-alpine
10 ARG JAR_FILE=target/*.jar
11 COPY ${JAR_FILE} eurekaserver.jar
12 EXPOSE 8761
13 ENTRYPOINT ["java", "-jar", "/eurekaserver.jar"]
```

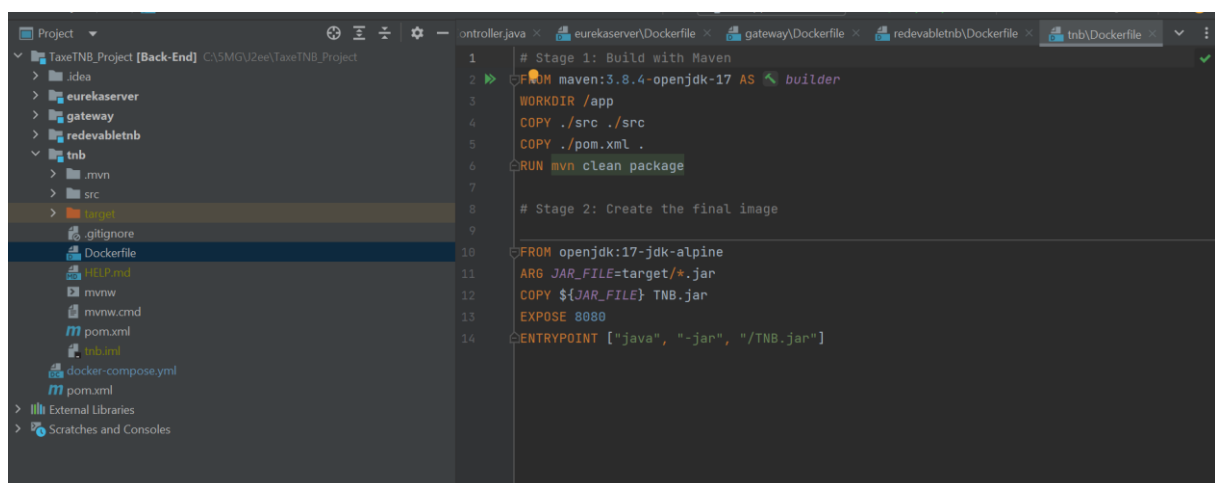


The screenshot shows the same IDE with the project structure. The file explorer now shows the 'gateway' directory. The main editor displays the content of the 'gateway/Dockerfile' file. The Dockerfile follows the same two-stage structure as the previous one, but with a different port and entry point.

```
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
7
8 # Stage 2: Create the final image
9 FROM openjdk:17-jdk-alpine
10 ARG JAR_FILE=target/*.jar
11 COPY ${JAR_FILE} gateway.jar
12 EXPOSE 8888
13 ENTRYPOINT ["java", "-jar", "/gateway.jar"]
```



```
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
7
8 # Stage 2: Create the final image
9
10 FROM openjdk:17-jdk-alpine
11 ARG JAR_FILE=target/*.jar
12 COPY ${JAR_FILE} RedevableTNB.jar
13 EXPOSE 8081
14 ENTRYPOINT ["java", "-jar", "/RedeableTNB.jar"]
```



```
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
7
8 # Stage 2: Create the final image
9
10 FROM openjdk:17-jdk-alpine
11 ARG JAR_FILE=target/*.jar
12 COPY ${JAR_FILE} TNB.jar
13 EXPOSE 8080
14 ENTRYPOINT ["java", "-jar", "/TNB.jar"]
```

Description du Fichier docker-compose.yml

Services

1. Eureka Server

- **Build** : Utilise le contexte du répertoire **./eureka-server** pour construire l'image.
- **Ports** : Expose le port 8761 pour accéder à l'interface d'Eureka Server.
- **Réseaux** : Connecté au réseau **my-network**.

2. Gateway

- **Build** : Utilise le contexte du répertoire **./gateway** pour construire l'image.
- **Ports** : Expose le port 8888 pour l'accès au Gateway.
- **Réseaux** : Connecté au réseau **my-network**.
- **Dépendances** : Attend que le service Eureka Server soit opérationnel.

3. RedeableTNB

- **Build** : Utilise le contexte du répertoire **./RedeableTNB** pour construire l'image.
- **Ports** : Expose le port 8086 pour l'accès au service RedeableTNB.

- **Réseaux** : Connecté au réseau **my-network**.
- **Dépendances** : Attend que le service Eureka Server soit opérationnel.

4. TNB

- **Build** : Utilise le contexte du répertoire **./TNB** pour construire l'image.
- **Ports** : Expose le port 8087 pour l'accès au service TNB.
- **Réseaux** : Connecté au réseau **my-network**.
- **Dépendances** : Attend que le service Eureka Server soit opérationnel.

Bases de Données

5. mysql_redevabletnb

- **Image** : Utilise l'image MySQL:latest.
- **Variables d'Environnement** : Configure le mot de passe et la base de données pour le service RedevableTNB.
- **Ports** : Mappe le port 3306 pour l'accès à la base de données.
- **Réseaux** : Connecté au réseau **my-network**.

6. mysql_tnb

- **Image** : Utilise l'image MySQL:latest.
- **Variables d'Environnement** : Configure le mot de passe et la base de données pour le service TNB.
- **Ports** : Mappe le port 3307 pour l'accès à la base de données.
- **Réseaux** : Connecté au réseau **my-network**.

Outils de Gestion de Base de Données

7. phpmyadmin_redevabletnb

- **Image** : Utilise l'image phpMyAdmin/phpMyAdmin.
- **Environnement** : Configure l'hôte et le port pour se connecter à la base de données du service RedevableTNB.
- **Ports** : Mappe le port 8082 pour l'accès à phpMyAdmin.
- **Réseaux** : Connecté au réseau **my-network**.
- **Dépendances** : Attend que le service mysql_redevabletnb soit opérationnel.

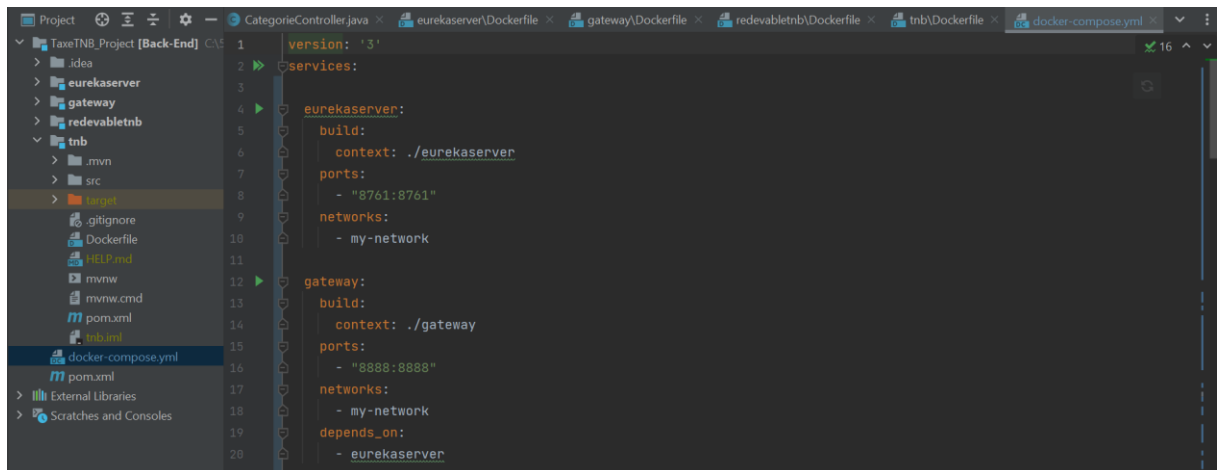
8. phpmyadmin_tnb

- **Image** : Utilise l'image phpMyAdmin/phpMyAdmin.
- **Environnement** : Configure l'hôte et le port pour se connecter à la base de données du service TNB.
- **Ports** : Mappe le port 8083 pour l'accès à phpMyAdmin.
- **Réseaux** : Connecté au réseau **my-network**.

- **Dépendances** : Attend que le service mysql_tnb soit opérationnel.

Réseaux

- **my-network** : Réseau personnalisé pour isoler le trafic entre les services.



5. CI/CD avec Jenkins

Processus et Configuration

Un pipeline CI/CD sera établi avec Jenkins pour automatiser les processus de construction, de test et de déploiement, garantissant la stabilité du système.

Script PipeLine :

```

1 pipeline {
2   agent any
3   tools {
4     maven 'maven'
5   }
6
7   stages {
8     stage('Git Clone') {
9       steps {
10        script {
11          checkout([$class: 'GitSCM', branches: [[name: 'main']], userRemoteConfigs: [[url: 'https://github.com:
12        }
13      }
14    }
15
16    stage('Build eureka') {
17      steps {
18        script {
19          // Navigate into the project directory before running Maven commands
20          dir('eureka-server') {
21            bat 'mvn clean install -DskipTests'
22          }
23        }
24      }
25    }
26
27    stage('Build redevable') {
28      steps {
29        script {
30          // Navigate into the project directory before running Maven commands
31          dir('RedeableTNB') {
32            bat 'mvn clean install -DskipTests'
33          }
34        }
35      }
36    }
37  }
38 }

```

```

38 stage('Build gateway') {
39   steps {
40     script {
41       // Navigate into the project directory before running Maven commands
42       dir('gateway') {
43         bat 'mvn clean install -DskipTests'
44       }
45     }
46   }
47 }
48
49 stage('Build tnb') {
50   steps {
51     script {
52       // Navigate into the project directory before running Maven commands
53       dir('TNB') {
54         bat 'mvn clean install -DskipTests'
55       }
56     }
57   }
58 }
59
60
61 stage('Run') {
62   steps {
63     script {
64
65
66       bat "docker-compose up -d"
67     }
68   }
69 }
70 }
71 }

```

	Declarative: Tool Install	Git Clone	Build eureka	Build redevable	Build gateway	Build tnb	Run
3s: is)	426ms	4s	43s	2min 7s	28s	33s	18s
	365ms	2s	23s	33s	23s	31s	9s

Explication :

Agent et Outils Maven :

- Utilise n'importe quel agent disponible.
- Utilise l'outil Maven configuré dans Jenkins.

Stages :

- **Git Clone :**
 - Clone le référentiel Git depuis le lien spécifié (https://github.com/AhatIrsale/TNB_Microservices.git).
 - Branche utilisée : main.
- **Build eureka :**
 - Compile le microservice Eureka Server.
 - Utilise Maven pour exécuter la phase clean et install.
 - Ignorer les tests (-DskipTests).
- **Build redevable :**
 - Compile le microservice RedevableTNB.
 - Utilise Maven pour exécuter la phase clean et install.
 - Ignorer les tests (-DskipTests).
- **Build gateway :**
 - Compile le microservice Gateway.
 - Utilise Maven pour exécuter la phase clean et install.
 - Ignorer les tests (-DskipTests).
- **Build tnb :**
 - Compile le microservice TNB.
 - Utilise Maven pour exécuter la phase clean et install.
 - Ignorer les tests (-DskipTests).
- **Run :**

Configuration

Script ?

```
1 pipeline {
2     agent any
3     tools {
4
5         maven 'maven'
6
7         scanner 'sonarscanner'
8     }
9 }
```

On ajoute la déclaration de sonnar scanner sur Jenkins

```
stage('SonarQube Analysis') {
    steps {
        script {
            withSonarQubeEnv('sonarscanner') {
                // Adjust the SonarQube analysis command as needed
                bat "mvn sonar:sonar -Dsonar.projectKey=organisation-1_jenkinssonar -Dsonar.projectName=JenkinsSonar"
            }
        }
    }
}
```

Stage 'SonarQube Analysis' :

- **Steps** : Les étapes à exécuter dans ce stage.
 - **Script** : Bloc script où les commandes spécifiques à SonarQube sont exécutées.
 - **withSonarQubeEnv('sonarscanner')** : Utilisation de l'environnement SonarQube configuré comme 'sonarscanner'.
 - **Commande SonarQube** : Exécute la commande Maven pour l'analyse SonarQube.
 - **mvn sonar:sonar -Dsonar.projectKey=organisation-1_jenkinssonar -Dsonar.projectName=JenkinsSonar**
 - **sonar.projectKey** : Clé unique du projet dans SonarQube.
 - **sonar.projectName** : Nom du projet dans SonarQube.

Bénéfices pour la Qualité du Code

SonarQube sera intégré pour surveiller la qualité du code, détecter les erreurs potentielles, et assurer la conformité aux normes de codage définies.

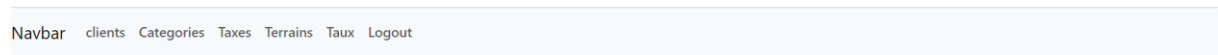
L'intégration de SonarQube dans le pipeline d'intégration continue offre des avantages tels que **la détection précoce des erreurs, la mesure de la qualité du code, l'application des normes de codage, la sensibilisation à la sécurité, et la réduction des défauts**. Cela permet une amélioration continue, une rétroaction rapide, une visibilité globale de la qualité du code, et la génération automatique de rapports détaillés, contribuant ainsi à maintenir un code robuste et fiable.

8. Conclusion

Résumé des Accomplissements

Le projet a atteint avec succès ses objectifs en mettant en œuvre une architecture microservices pour gérer la taxe sur les terrains non bâtis au Maroc. L'utilisation de microservices, tels que Eureka Server,

L'adoption de Docker pour la conteneurisation a facilité le déploiement, offrant une portabilité et une gestion des dépendances efficaces. L'intégration continue avec Jenkins a automatisé les processus d'intégration, de construction et de déploiement, renforçant la cohérence et la fiabilité du cycle de développement.



Clients

Nom	CIN	Create
-----	-----	--------

COMPANY NAME	PRODUCTS	USEFUL LINKS	CONTACT
Here you can use rows and columns to organize your footer content. Lorem ipsum	MDBBootstrap	Your Account	New York, NY 10012, US

categorie works!

Categories

cat11 - cat11 [Edit](#) [Delete](#)

cat2 - cat2 [Edit](#) [Delete](#)

Create New Categorie

Libelle

Description

Create

Taxe works!

Taxe

marque 3 - Taha - 2	Edit Delete
marque 2 - TTTT - 2	Edit Delete
marque 3 - Taha - 2001	Edit Delete
marque 1 - Taha - 1	Edit Delete

Create New Taxe

Marque: Client: Taux:

Terrain:

[Create](#)

Terrain works!

Terrains

TERRAIN2 - 5 - cat11 - TTTTT			Edit	Delete
TERRAIN5 - 400 - cat11 - TTTTT			Edit	Delete
aaaa - 100 - cat2 - Taha			Edit	Delete
test - 22 - cat11 - Taha			Edit	Delete

Nom:

test

Surface:

100

Catégorie:

cat2

Client:

Taha

Save

Cancel

Taux works!

Taux

Anne	Valeur	Catégorie	Actions
2001	5	cat11	Edit Delete
2	2	cat11	Edit Delete
1	1	cat11	Edit Delete

Create New Taux

Anne:

Valeur:

Catégorie:

Create

terrain-taxes works!

Liste des Terrains

ID Terrain	Nom	Surface	Catégorie	Utilisateur	Action
Vous n'avez pas de terrain.					

terrain-taxes works!

Liste des Terrains

ID Terrain	Nom	Surface	Catégorie	Utilisateur	Action
02000000-0000-0000-0000-000000000000	TERRAIN2	5	cat11	taha	<button>Afficher les taxes</button>
06000000-0000-0000-0000-000000000000	TERRAIN5	400	cat11	taha	<button>Afficher les taxes</button>
823640dd-5a79-4938-93fe-bde442ad62e5	test	22	cat11	taha	<button>Afficher les taxes</button>

Liste des Terrains

ID Terrain	Nom	Surface	Catégorie	Utilisateur	Action
02000000-0000-0000-0000-000000000000	TERRAIN2	5	cat11	taha	<button>Afficher les taxes</button>
06000000-0000-0000-0000-000000000000	TERRAIN5	400	cat11	taha	<button>Afficher les taxes</button>
823640dd-5a79-4938-93fe-bde442ad62e5	test	22	cat11	taha	<button>Afficher les taxes</button>

Taxes du Terrain

ID Taxe	Marque	Valeur	Action
---------	--------	--------	--------

Aucune taxe pour ce terrain pour le moment.

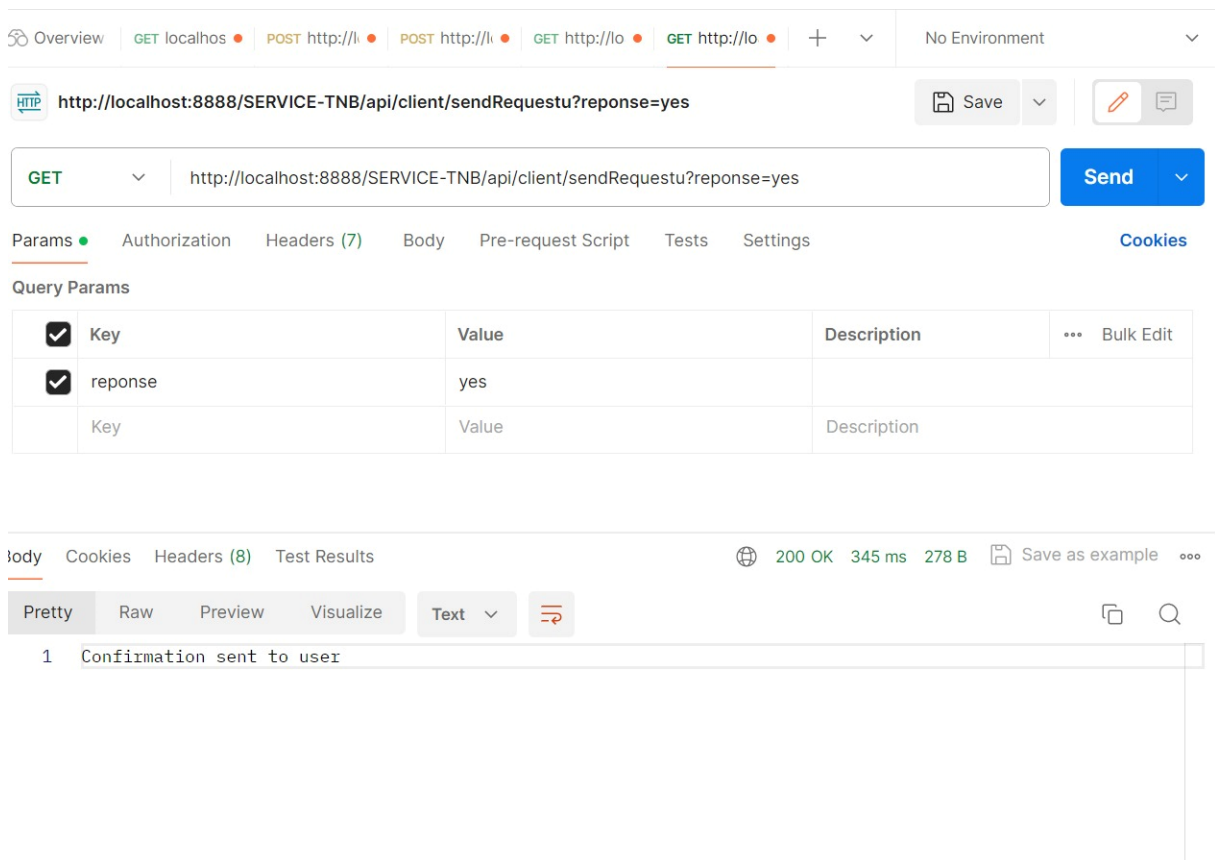
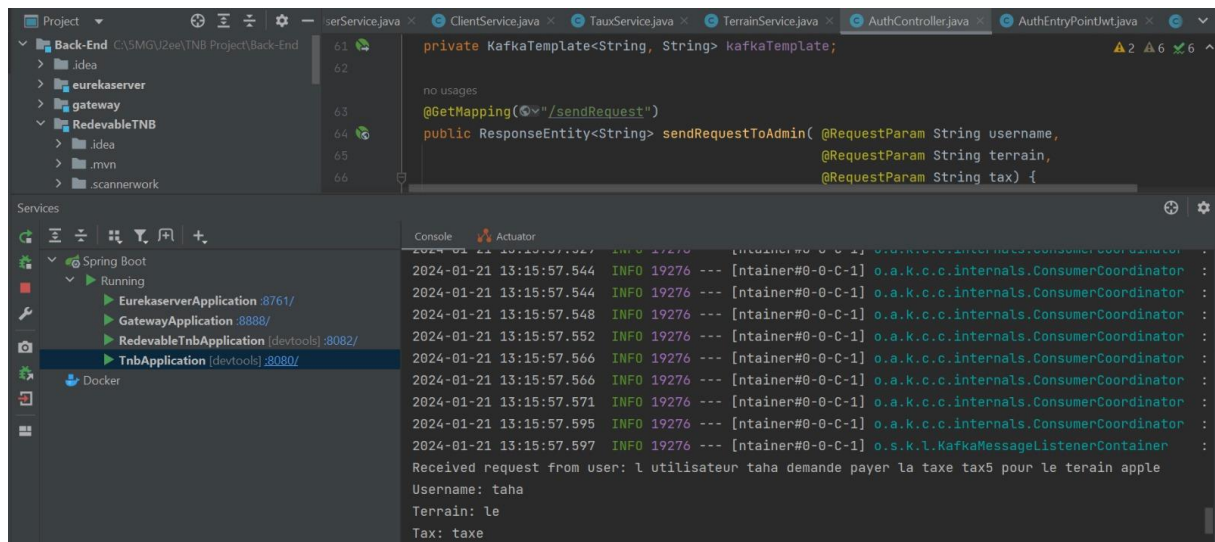
in-taxes works!

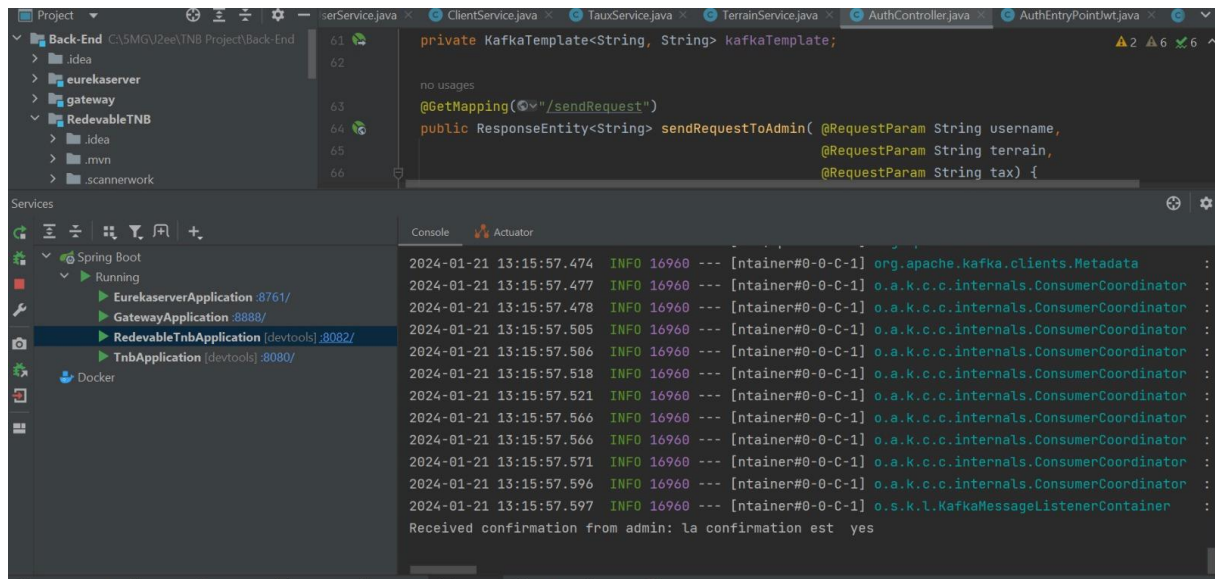
Liste des Terrains

ID Terrain	Nom	Surface	Catégorie	Utilisateur	Action
02000000-0000-0000-0000-000000000000	TERRAIN2	5	cat11	taha	<button>Afficher les taxes</button>
06000000-0000-0000-0000-000000000000	TERRAIN5	400	cat11	taha	<button>Afficher les taxes</button>
823640dd-5a79-4938-93fe-bde442ad62e5	test	22	cat11	taha	<button>Afficher les taxes</button>

Taxes du Terrain

ID Taxe	Marque	Valeur	Action
4f791557-462b-4842-a1da-e136b1652d8f	marque 1	1	<button>Payer</button>





Perspectives Futures

Au-delà des améliorations techniques, les perspectives futures pour le système pourraient inclure des fonctionnalités et des optimisations axées sur l'amélioration de l'expérience utilisateur, la conformité aux réglementations en constante évolution et l'adaptabilité aux nouvelles technologies.

1. Intégration avec les Services Gouvernementaux :

- Explorer des possibilités d'intégration avec d'autres services gouvernementaux ou plates-formes pour faciliter l'échange d'informations et garantir la conformité aux exigences gouvernementales.

2. Automatisation des Rapports Financiers :

- Mettre en place des mécanismes d'automatisation pour la génération de rapports financiers liés aux taxes TNB, facilitant ainsi la transparence et la conformité aux normes comptables.

3. Intelligence Artificielle et Analytique de Données :

- Explorer la possibilité d'intégrer des technologies émergentes telles que l'intelligence artificielle et l'analytique de données pour identifier des tendances, prévoir les besoins futurs et optimiser les processus décisionnels.

4. Formation Continue et Support Utilisateur :

- Mettre en place des programmes de formation continue pour les utilisateurs et le personnel, ainsi qu'un support utilisateur robuste pour garantir une adoption efficace et une utilisation optimale du système.