**Rapport Projet :**

**Architecture des composants d’entreprise**

**Realisé par**

**EL ASRI MOHAMED Taha**

**NAJAHI 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 Bâtis (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 payement 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 :

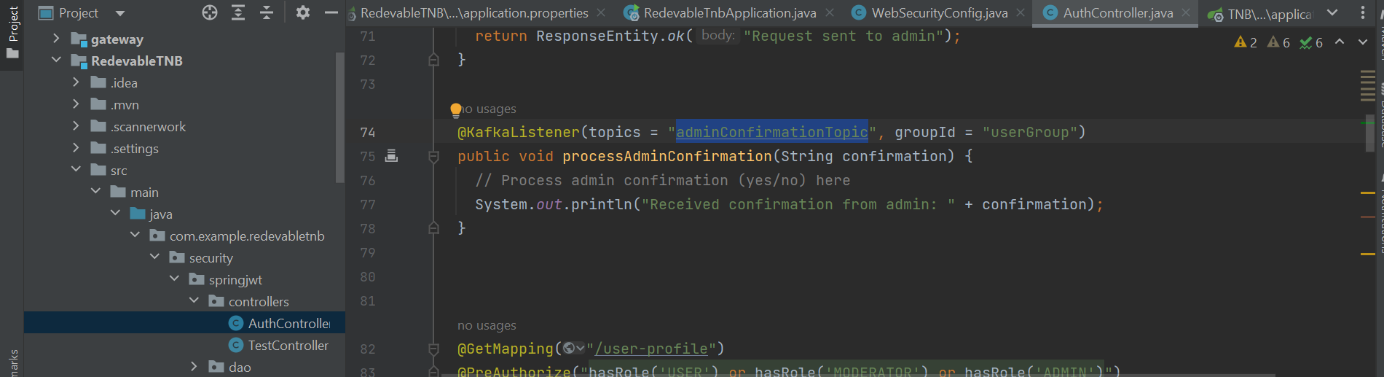
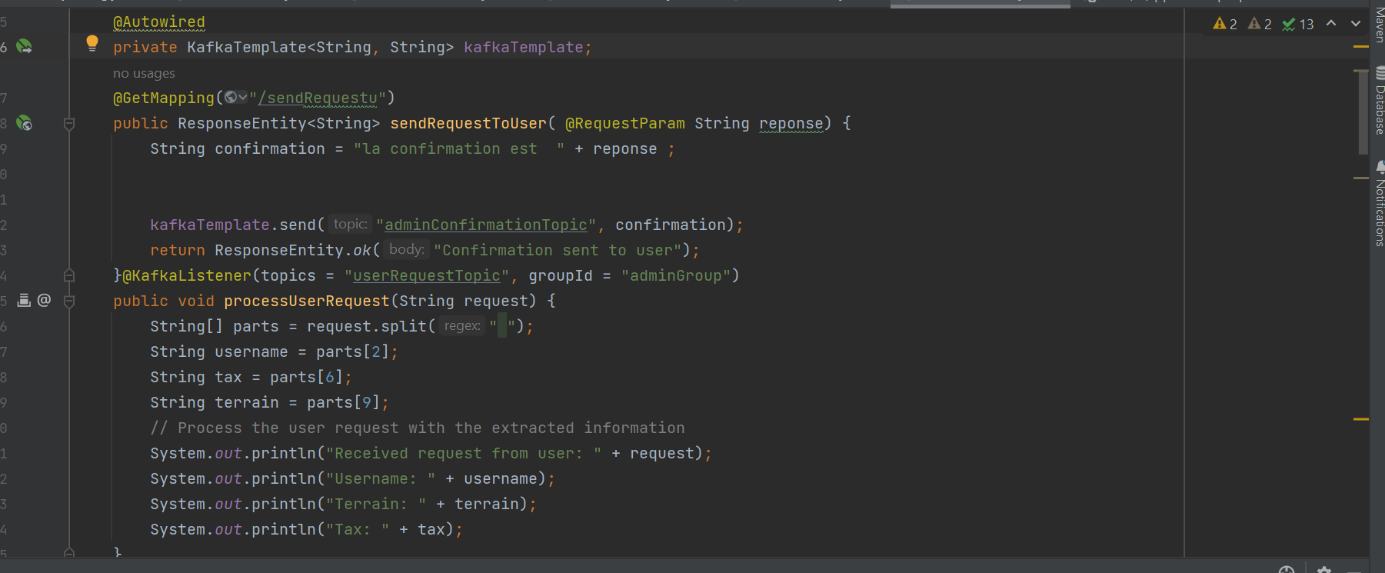
* Service de gestion des catégories de terrains
* Service de gestion des terrain
* Service de gestion des taux de terrain
* Service de calcul de la taxe TNB
* Service de gestion de redevable .

**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 payement 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 consomation des methode user pour spring security seront effectués via des requêtes REST, fournissant une communication légère et simple entre les services.

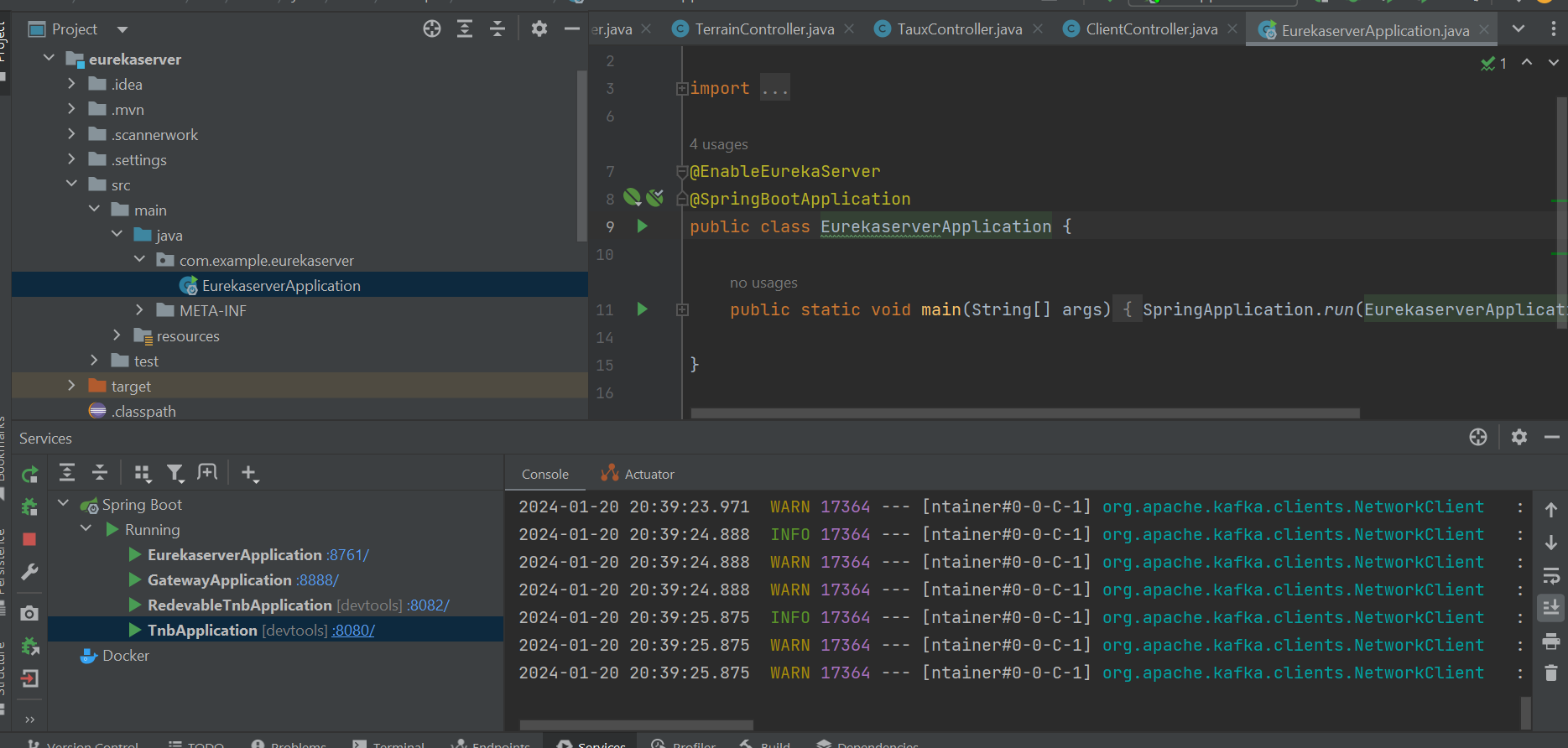
Les captures : kafka et resttemplate code

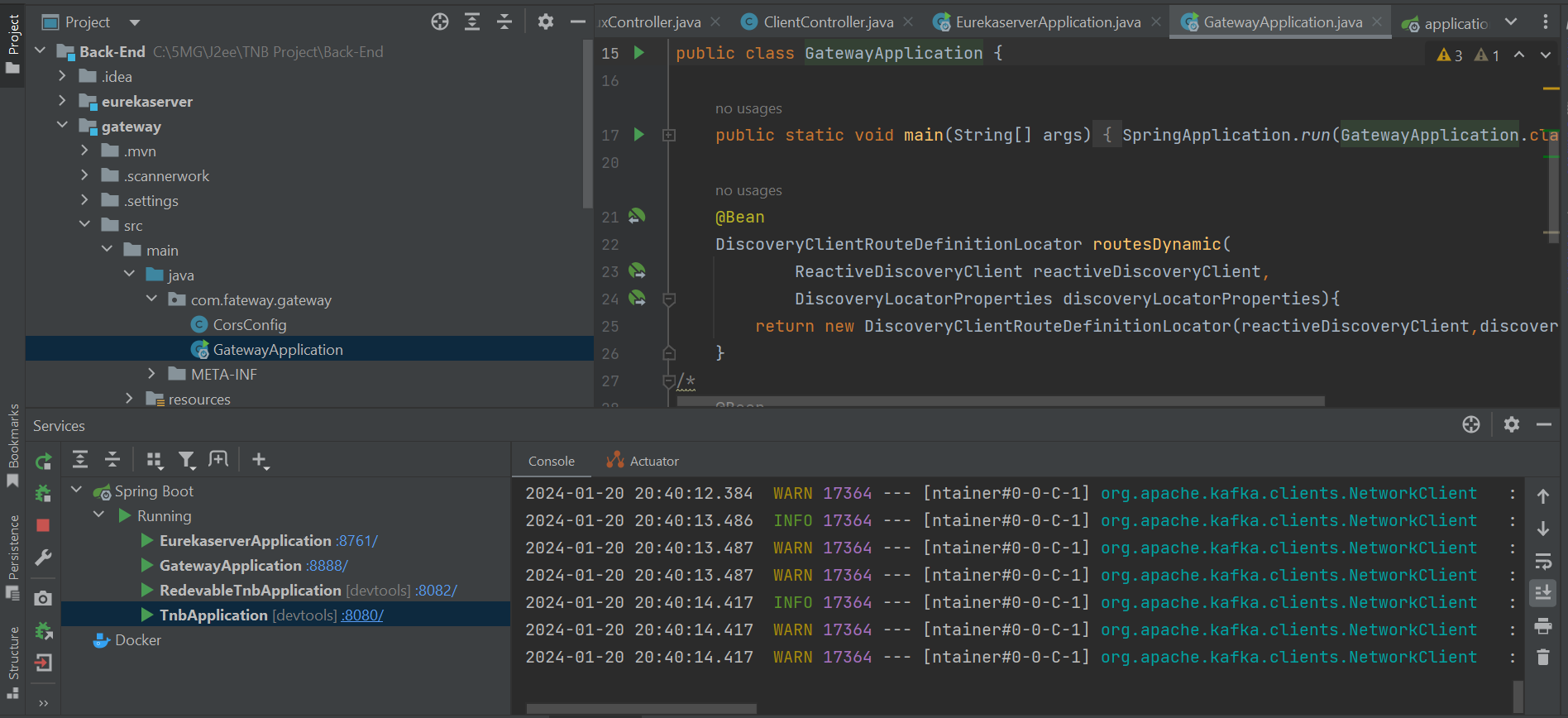


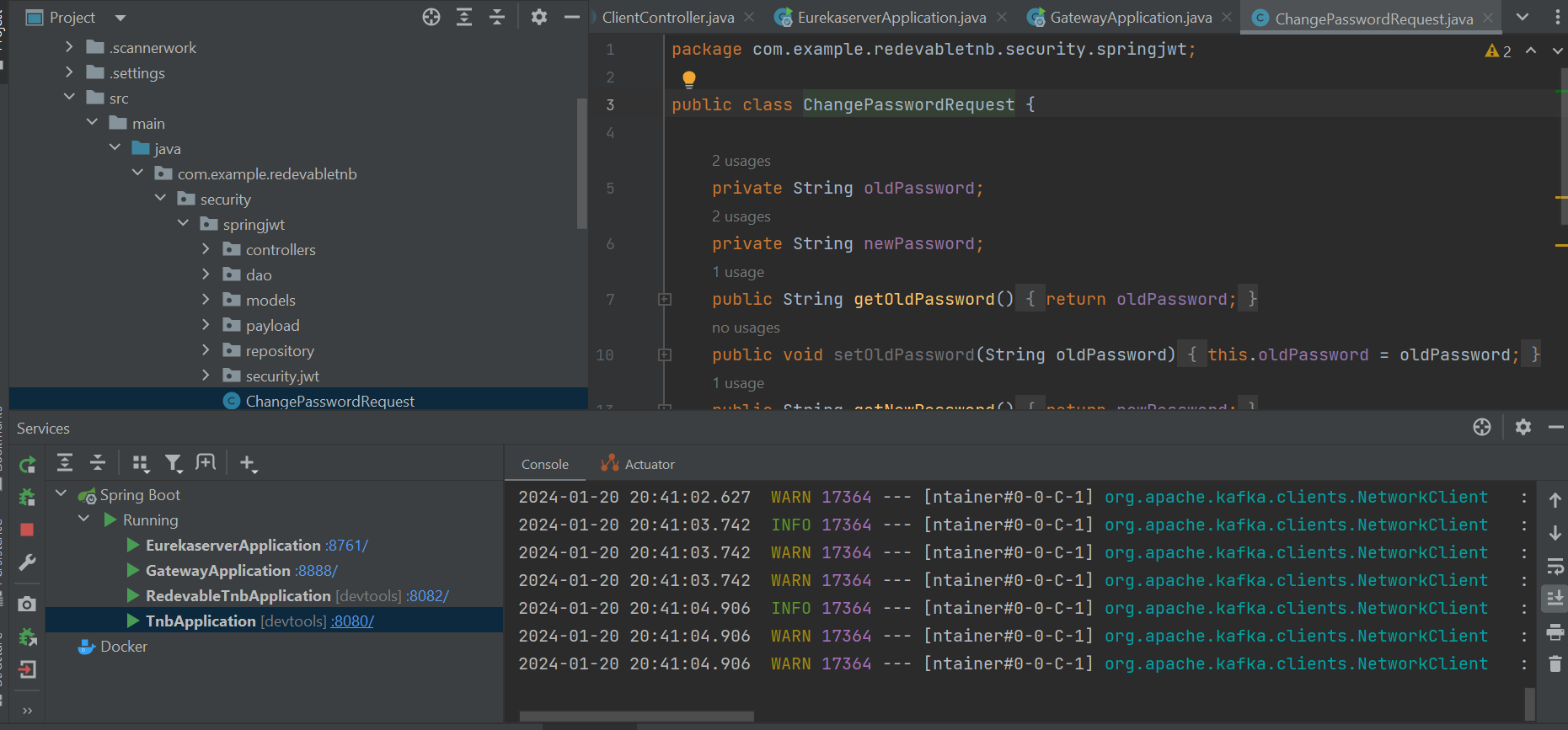
**3. Conception des Microservices**

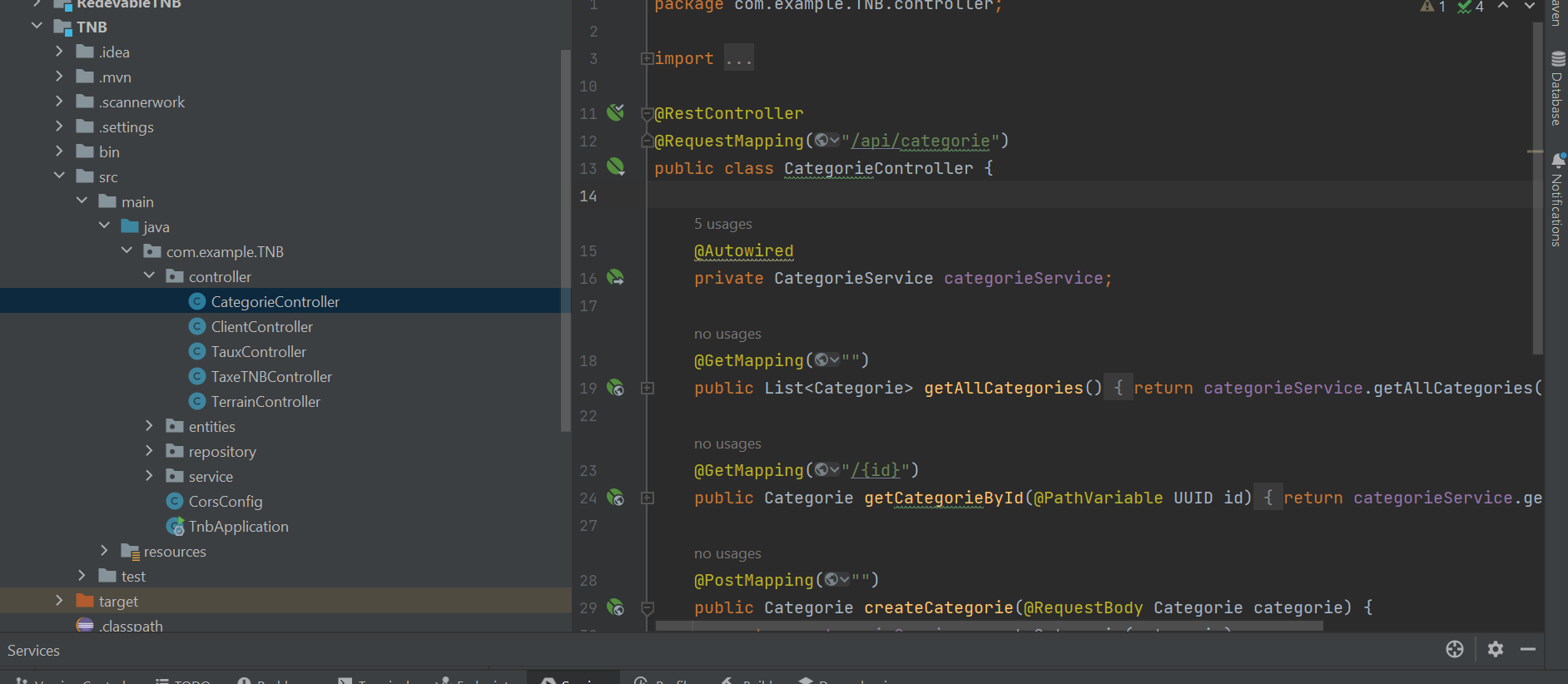
Les microservices seront conçus en suivant les principes SOLID, et la conception se déclinera en quatre principaux microservices :

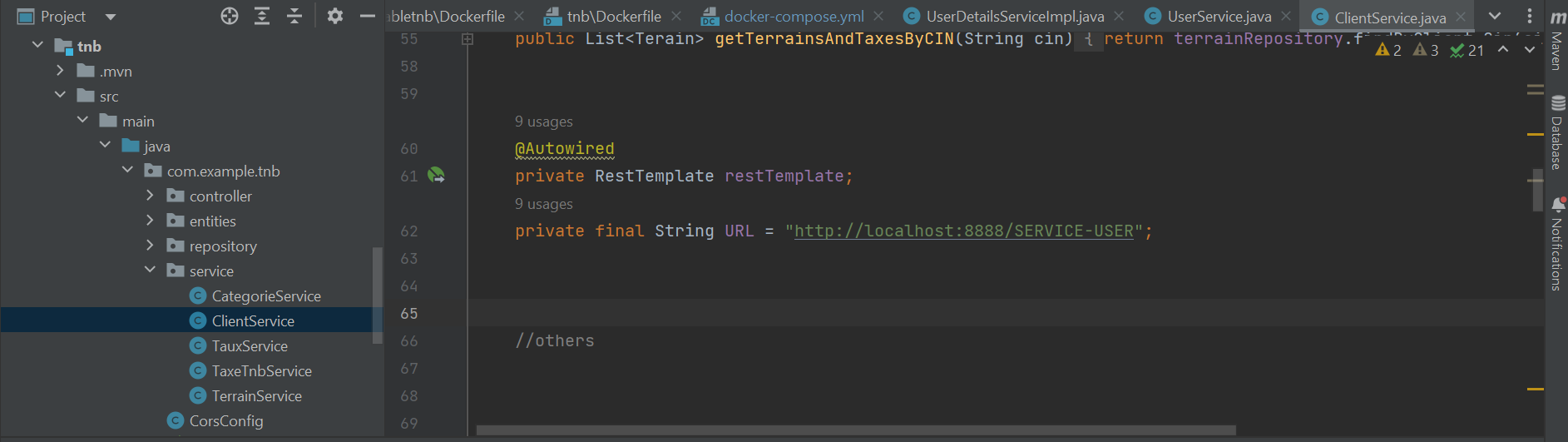
1. **Eureka Server :**
   * **Responsabilités :** Enregistrement et découverte des microservices dans l'écosystème.
   * **Technologies :** Spring Cloud Eureka.
   * **Objectifs :** Assurer la disponibilité des microservices et faciliter leur communication.
2. **Gateway :**
   * **Responsabilités :**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











**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

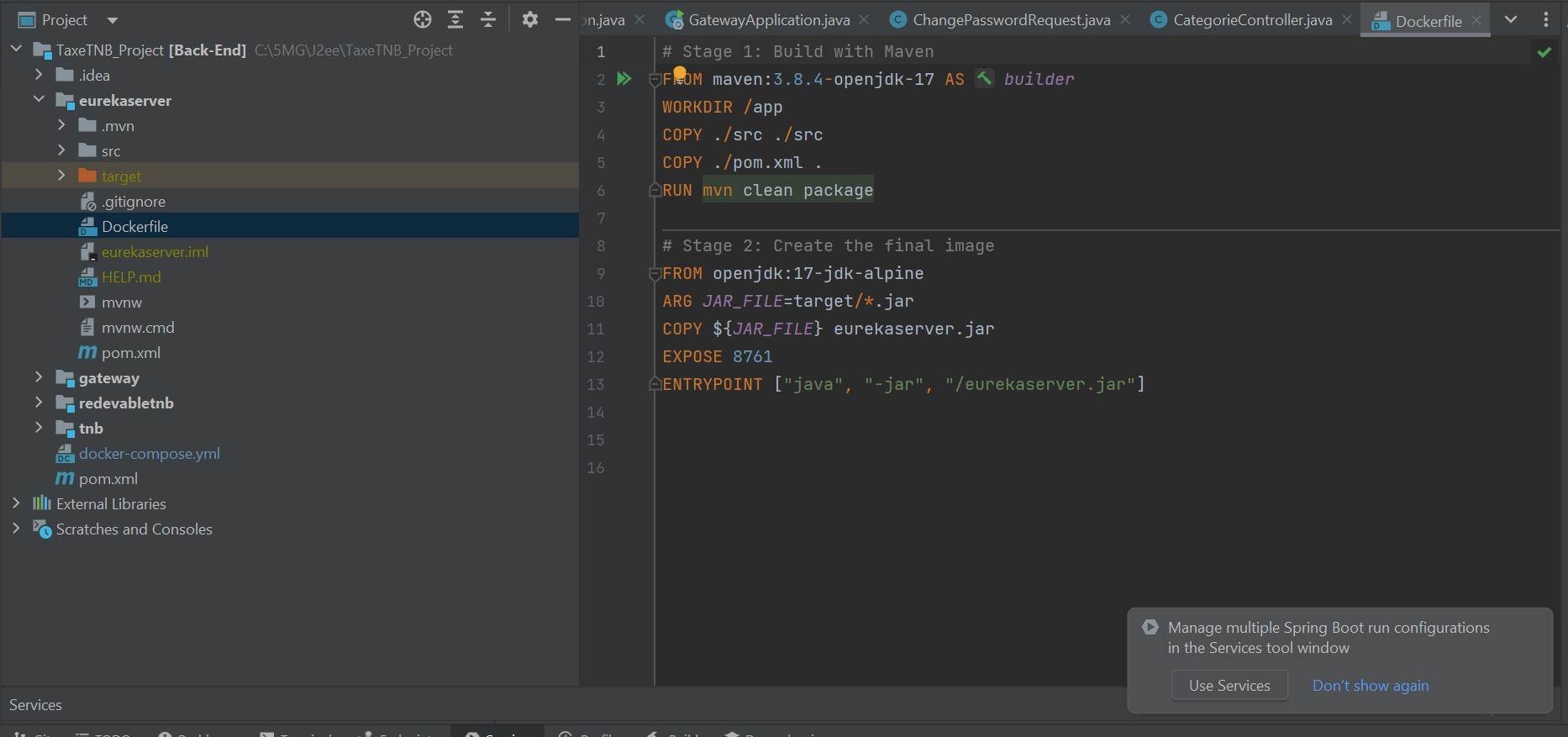
Le Dockerfile est un script qui contient des instructions pour la construction d'une image Docker. Chaque microservice dispose de son propre Dockerfile, décrivant les étapes nécessaires pour créer une image contenant l'application et ses dépendances. Voici une brève explication des sections clés d'un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

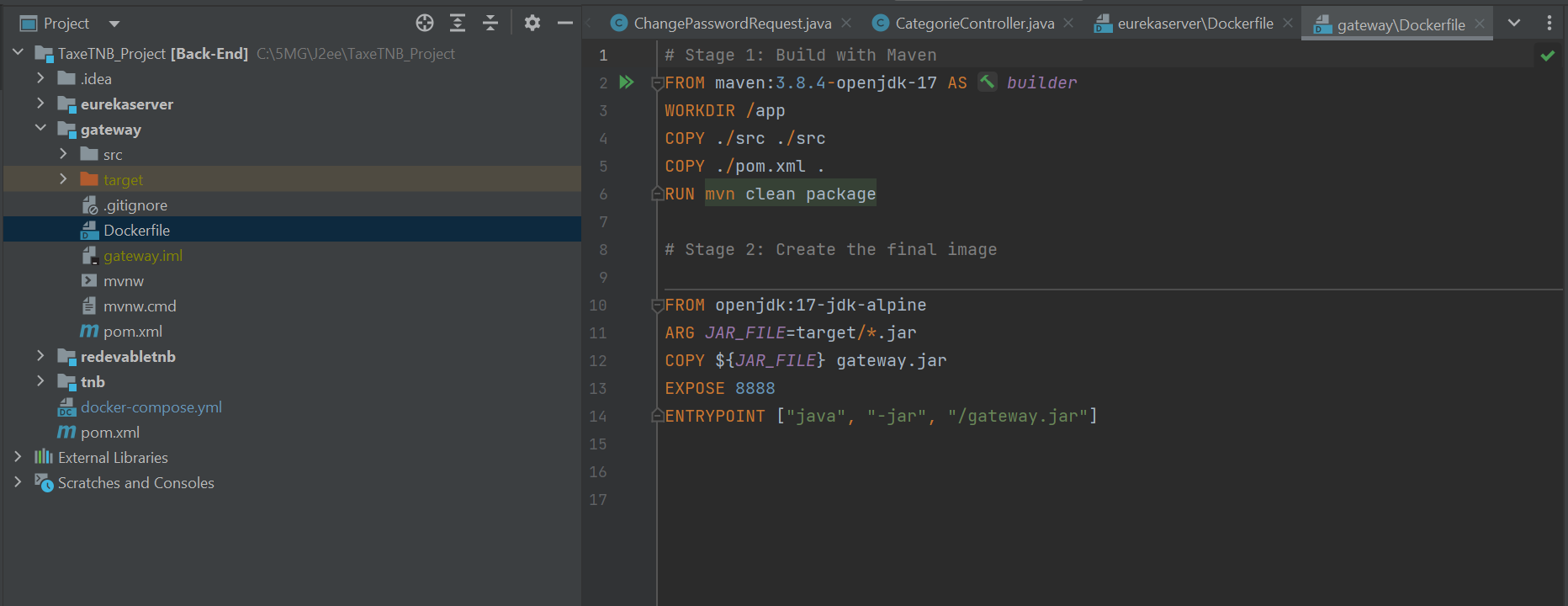
**Stage 1: Build with Maven (ou autre outil de build) :**

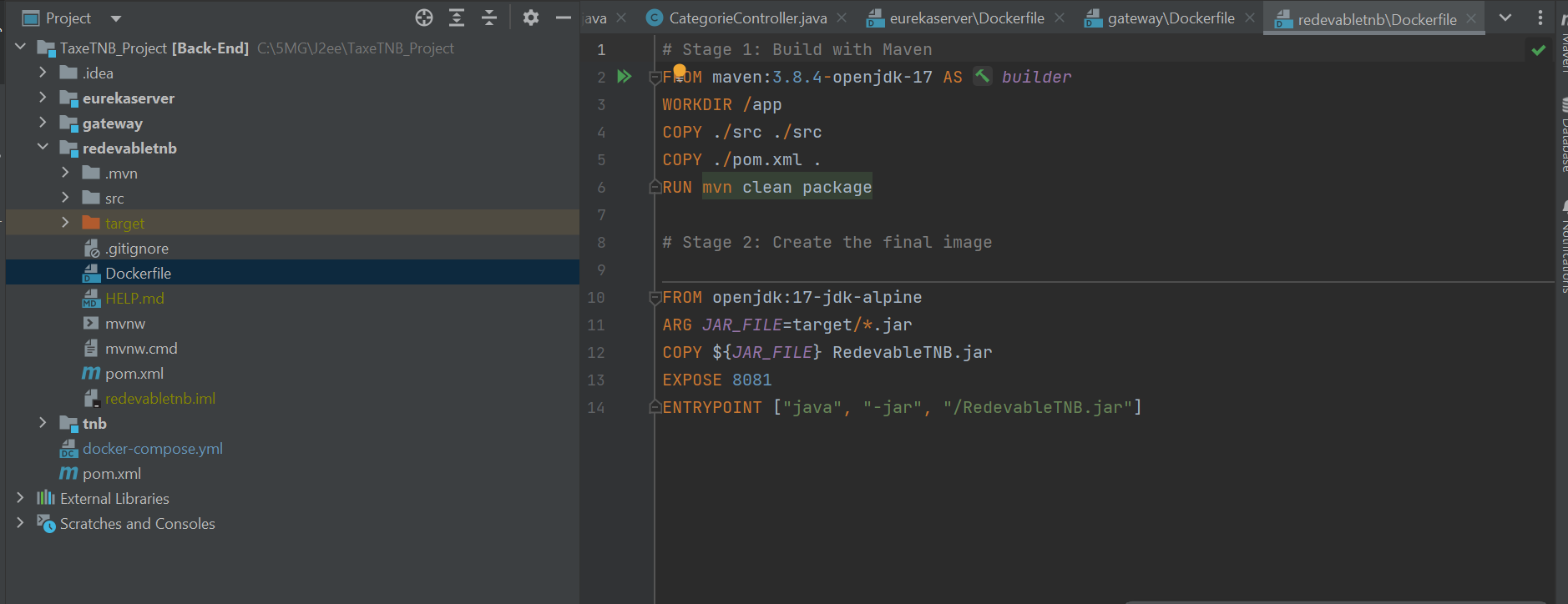
* + Dans cette étape, l'image Maven est utilisée comme base pour construire l'application Java. Toutes les sources et les dépendances nécessaires sont copiées dans le conteneur. Ensuite, la commande **mvn clean package** est exécutée pour compiler l'application et créer le fichier JAR.

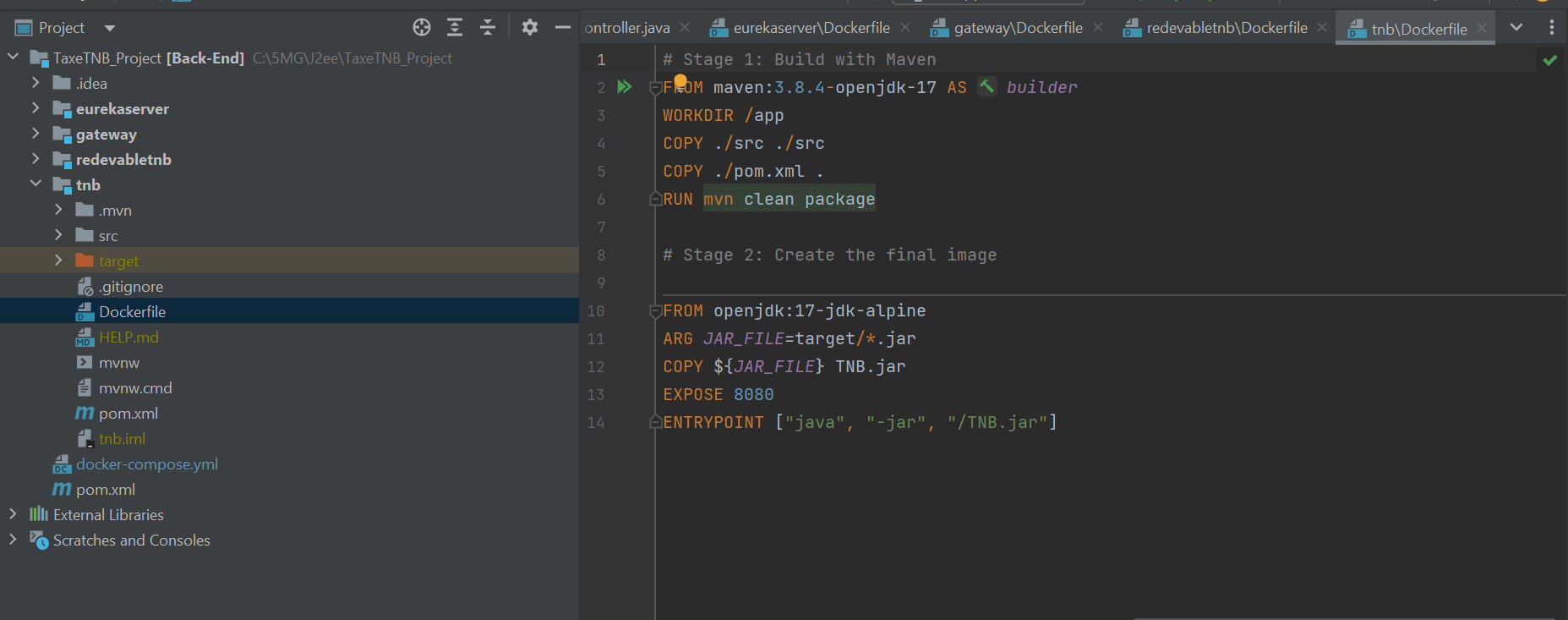
**Stage 2: Create the final image :**

* + 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.



****

****

****

**Description du Fichier docker-compose.yml**

**Services**

1. Eureka Server

* **Build :** Utilise le contexte du répertoire **./eurekaserver** 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. RedevableTNB

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

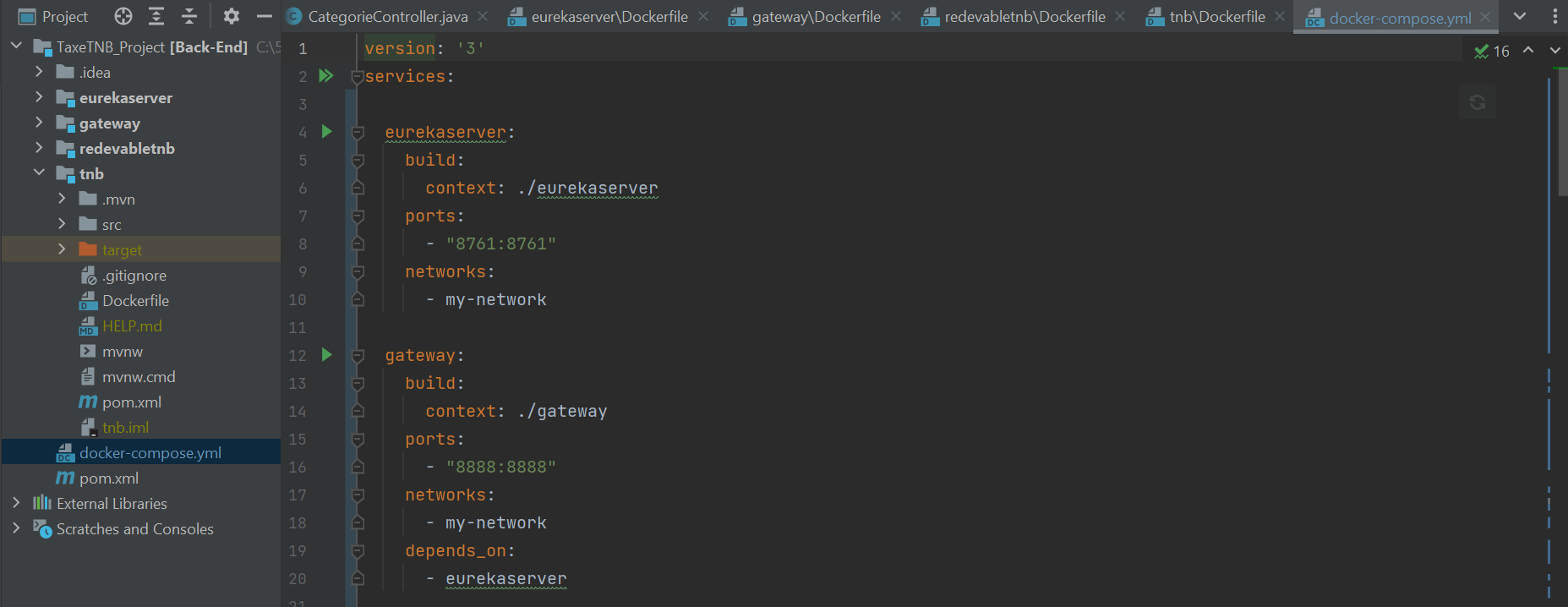
* **Image :** Utilise l'image phpMyAdmin/phpMyAdmin.
* **Environnement :** Configure l'hôte et le port pour se connecter à la base de données du service RedevableTNB.
* **Ports :** Mappe le port 8082 pour l'accès à phpMyAdmin.
* **Réseaux :** Connecté au réseau **my-network**.
* **Dépendances :** Attend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

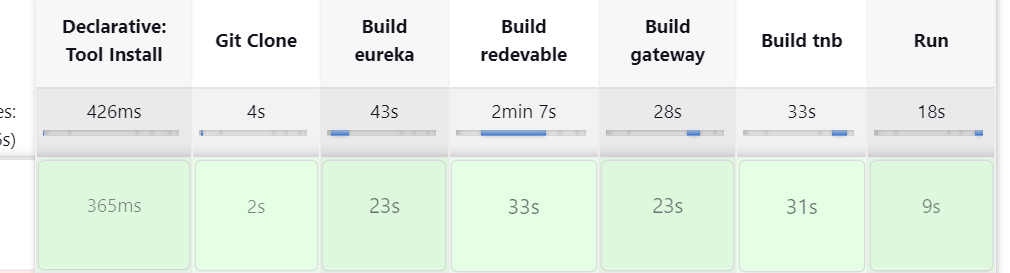
**Processus et Configuration**

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 :**







**Explication :**

**Agent et Outils Maven :**

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

**Stages :**

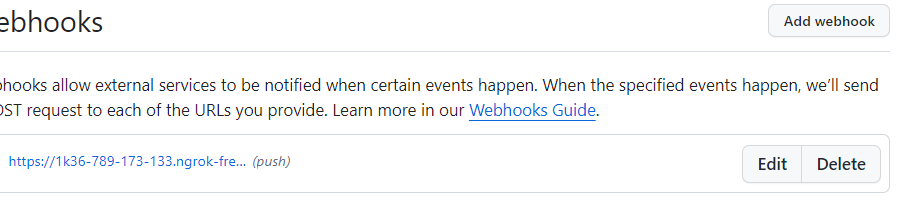
* + **Git Clone :**
    - Clone le référentiel Git depuis le lien spécifié (<https://github.com/AhatIrsale/TNB_Microservices.git>).
    - Branche utilisée : main.
  + **Build eureka :**
    - Compile le microservice Eureka Server.
    - Utilise Maven pour exécuter la phase clean et install.
    - Ignorer 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 :**
    - Lance les conteneurs Docker définis dans le fichier docker-compose.yml à la racine du projet.
    - Utilise la commande docker-compose up -d pour démarrer les conteneurs en mode détaché.

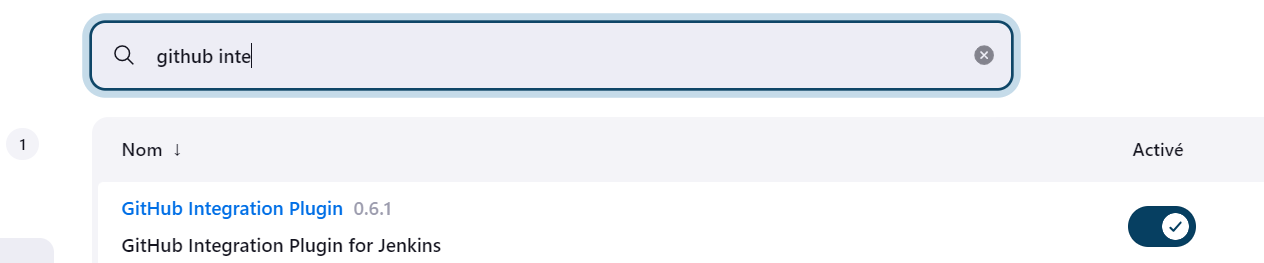
**6. Déploiement Automatique**

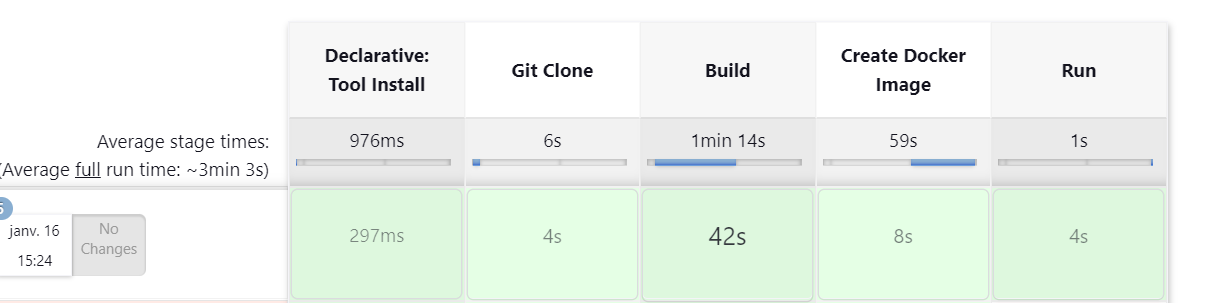
**Utilisation de Ngrok ou Azure Cloud**

Le déploiement automatique sera facilité en utilisant des outils tels que Ngrok pour un environnement de développement, ou Azure Cloud pour des déploiements en production.

**Gitwebhook First :**

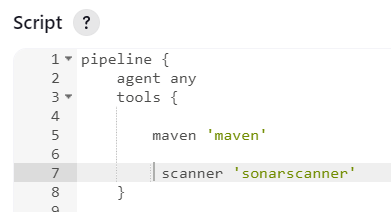






**7. Intégration de SonarQube**

**Configuration**



On ajoute la déclaration de sonnar scanner sur Jenkins



**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.

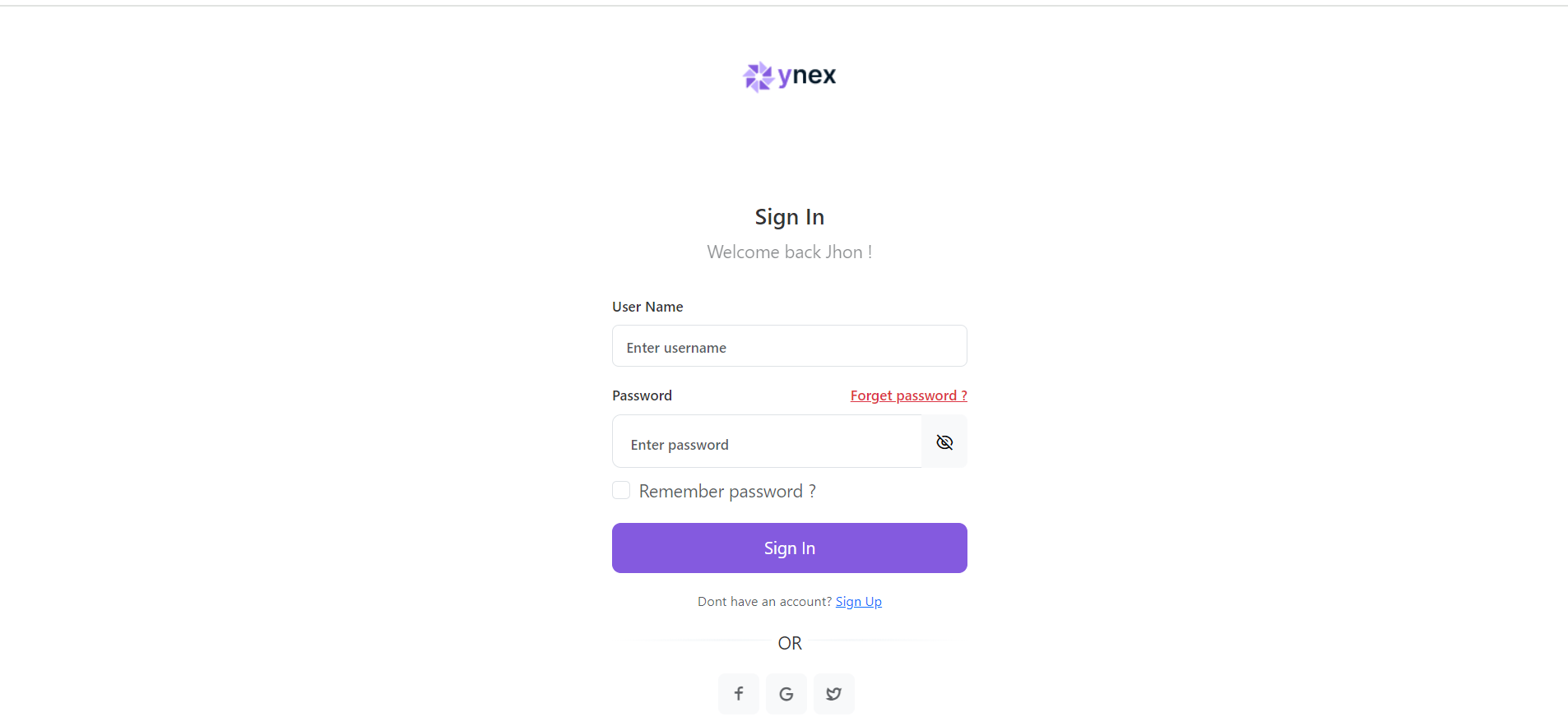
L'intégration de SonarQube dans le pipeline d'intégration continue offre des avantages tels que **la détection précoce des erreurs**, **la mesure de la qualité du code**, **l'application des normes de codage**, **la sensibilisation à la sécurité**, et **la réduction des défauts.** Cela permet une amélioration 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.

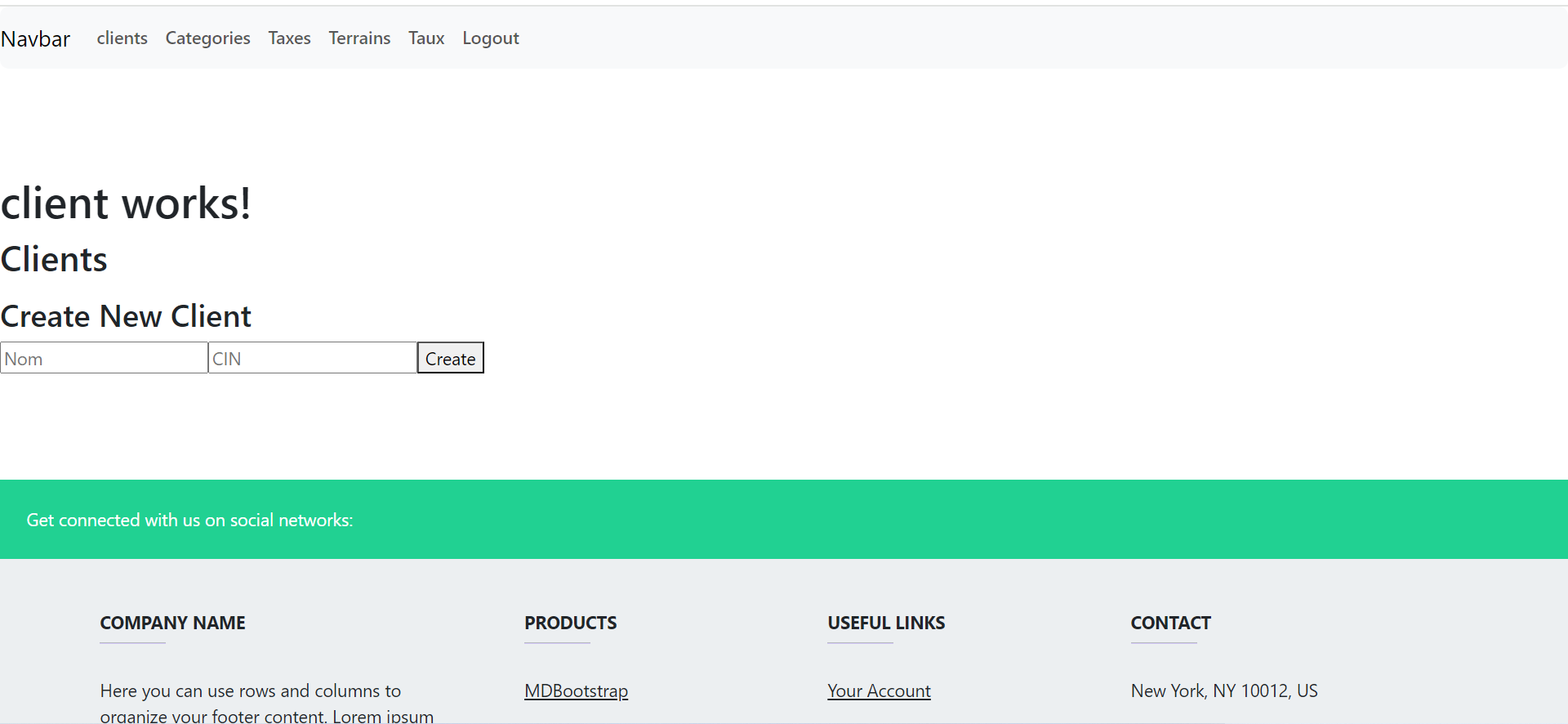
**8. Conclusion**

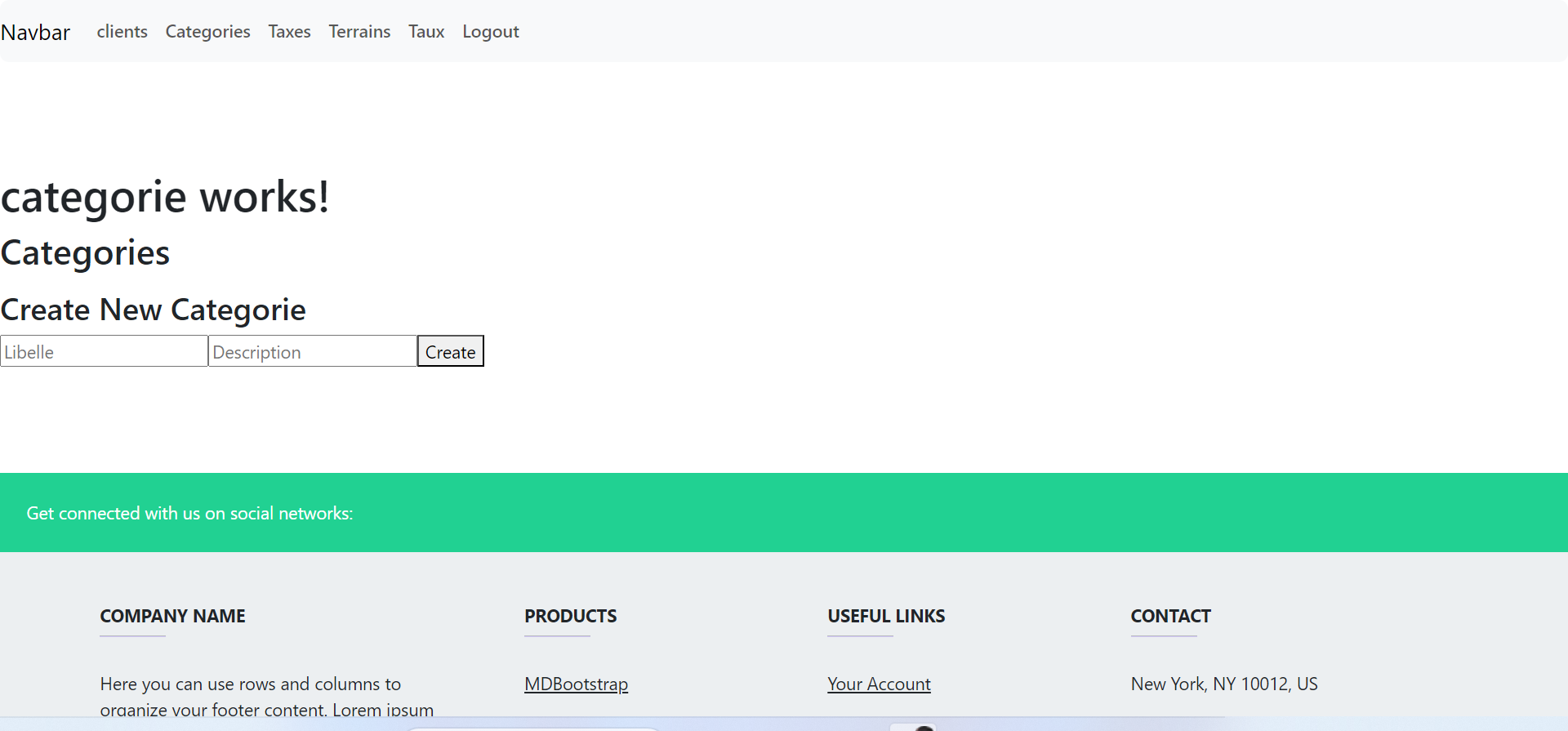
**Résumé des Accomplissements**

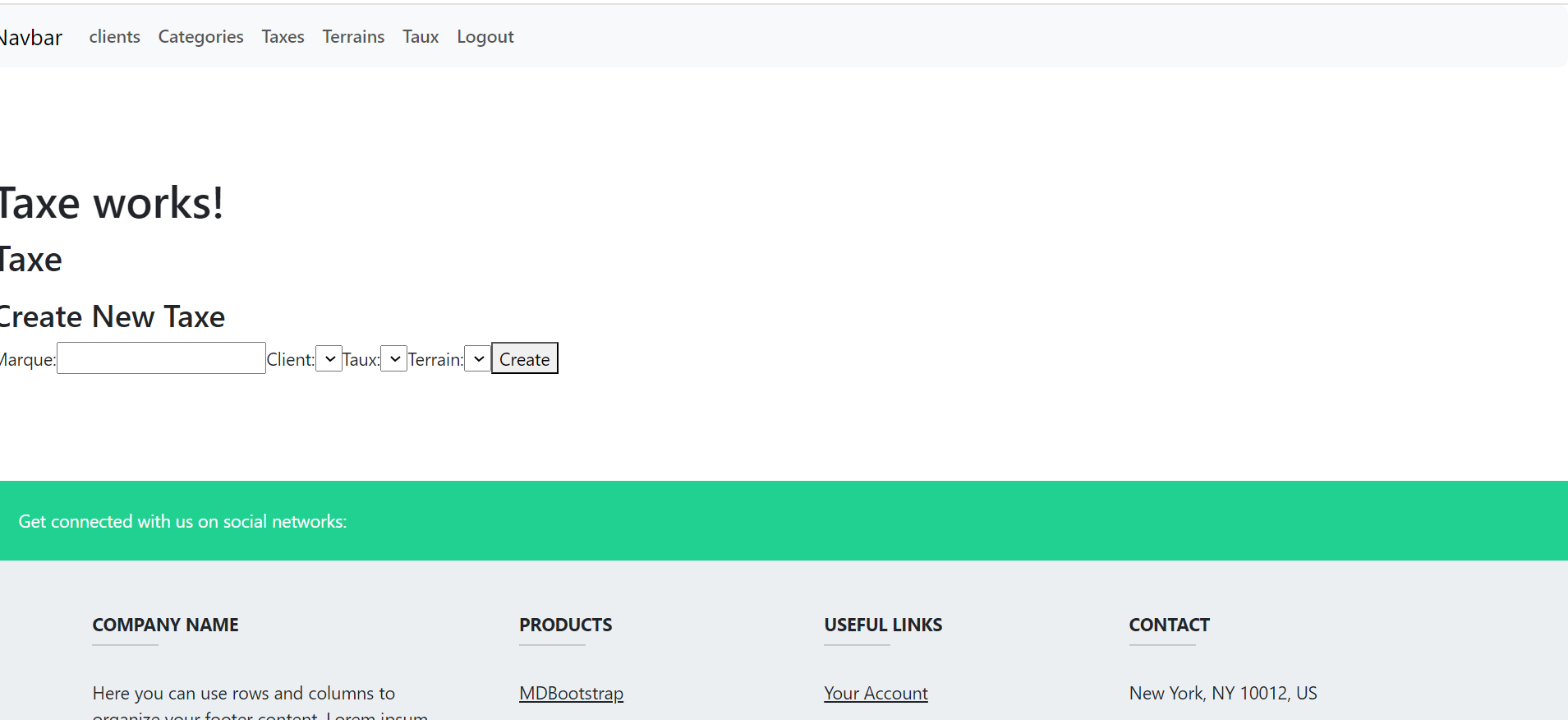
Le projet a atteint avec succès ses objectifs en mettant en œuvre une architecture microservices pour gérer la taxe sur les terrains non bâtis au Maroc. L'utilisation de microservices, tels que Eureka Server, Gateway, RedevableTNB, et TNB, a permis une conception modulaire, favorisant la maintenabilité et la scalabilité du système. Les mécanismes de communication, notamment Kafka et REST Template, ont été intégrés pour assurer une communication efficace entre les microservices.

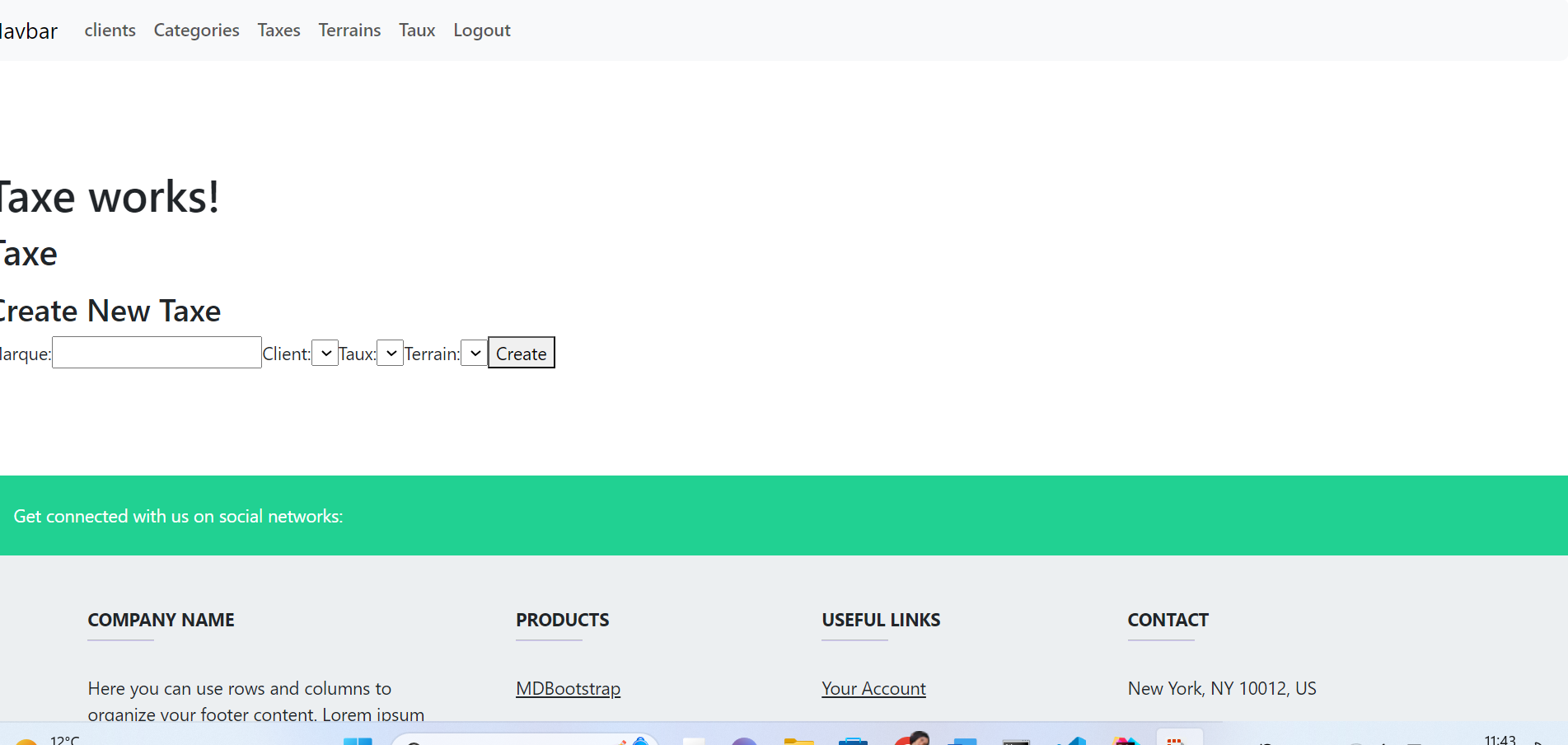
L'adoption de Docker pour la conteneurisation a facilité le déploiement, offrant une portabilité et une gestion des dépendances efficaces. L'intégration continue avec Jenkins a automatisé les processus d'intégration, de construction et de déploiement, renforçant la cohérence et la fiabilité du cycle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

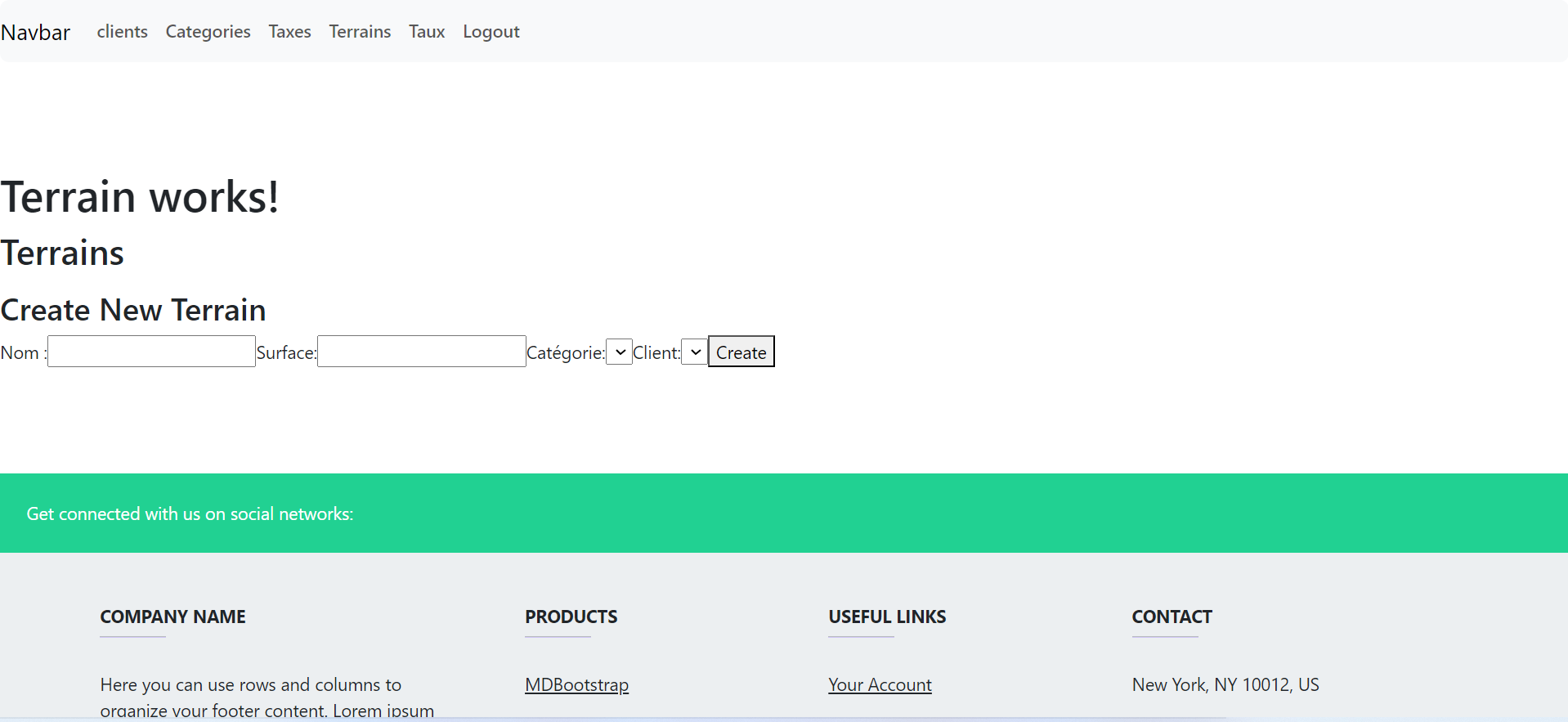


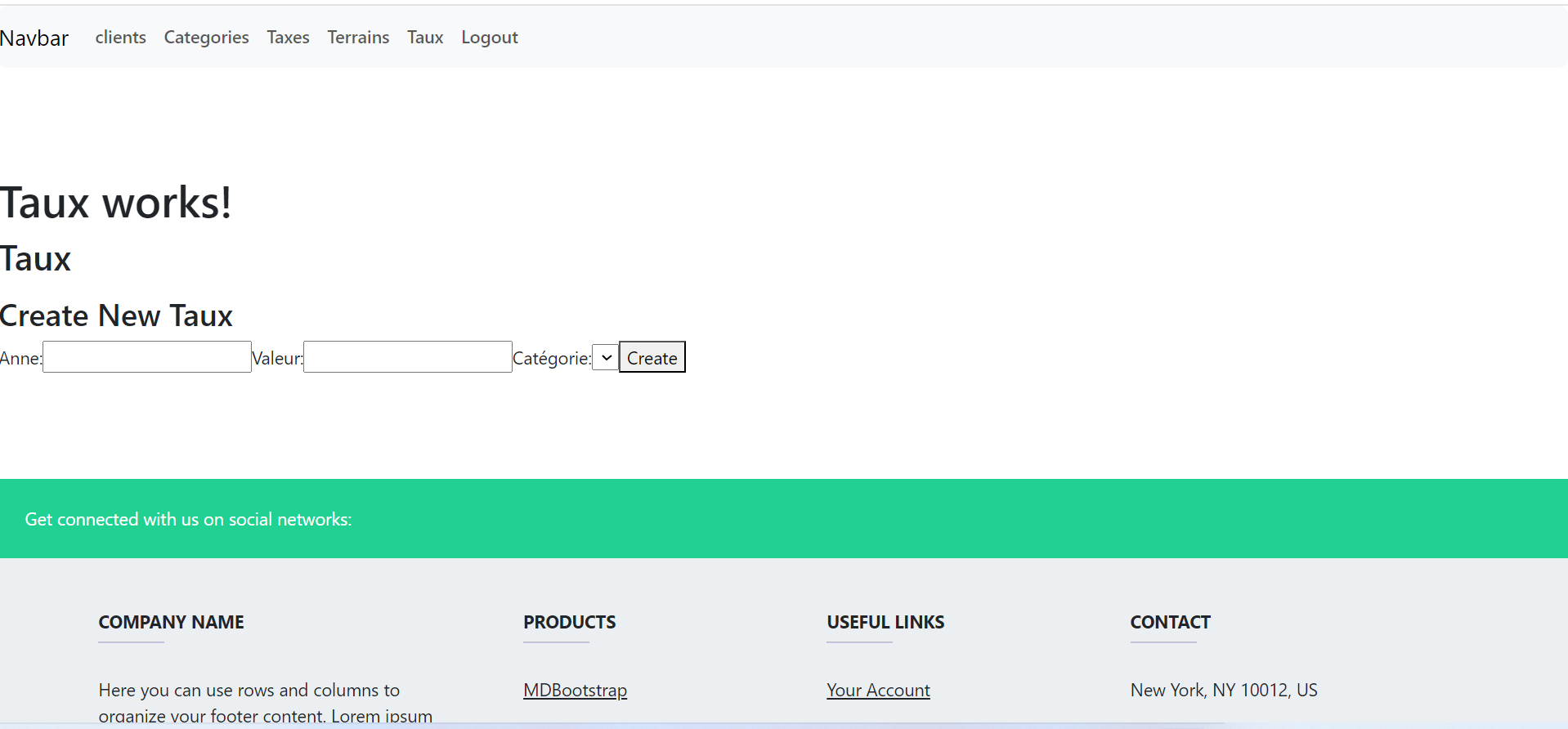












**Perspectives Futures**

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.