**Rapport Projet : Architecture des composants d’entreprise**

**Ingénierie Informatique et Réseaux**



**Réalisé par :**

Oukattou othmane

Chenani imane

Saydtahiri abderrahman

1. **Introduction**

L'architecture microservices joue un rôle crucial dans le paysage du développement logiciel moderne en offrant une approche modulaire qui simplifie la complexité des applications. En fragmentant une application en microservices autonomes et indépendants, cette architecture permet une évolutivité efficace, une résilience accrue, et des déploiements indépendants facilités. Chaque microservice peut être développé, déployé, et évolué de manière distincte, offrant ainsi une flexibilité et une agilité considérables. En favorisant l'utilisation de différentes technologies adaptées à chaque service spécifique, les microservices encouragent l'innovation et la mise en œuvre des dernières technologies. Malgré les défis potentiels liés à la gestion de la communication inter-microservices, l'architecture microservices demeure une approche incontournable, alignée sur les principes agiles, permettant des cycles de développement plus rapides, une meilleure gestion des risques, et une évolution harmonieuse des applications à mesure que les besoins évoluent.

# 

# **Chapitre 1 :**

# 

**Architecture Microservices**

**1.Architecture :**

Une image contenant texte, diagramme, ligne, capture d’écran

Description générée automatiquement

**2.Description des services**

**2.1.Service TNB :**

un service TNB peut être défini comme un composant logiciel dédié au calcul, à la gestion et à la manipulation des données relatives aux Travaux Non Bâtis. Ce service offre une modularité et une autonomie, permettant le développement indépendant, le déploiement séparé et l'évolution spécifique des fonctionnalités liées aux Travaux Non Bâtis. En regroupant ces fonctionnalités au sein d'un microservice dédié, l'application garantit une flexibilité accrue dans la gestion des projets d'infrastructures non bâties, tout en facilitant la maintenance, la mise à l'échelle et l'intégration de nouvelles fonctionnalités. La conception d'un service TNB s'inscrit ainsi dans une approche microservices, favorisant une architecture modulaire et une gestion efficiente des opérations liées aux Travaux Non Bâtis au sein de l'application.

**2.2.Service User :**

un service User peut être défini comme un composant logiciel dédié à la gestion des utilisateurs. Ce service offre des fonctionnalités telles que l'authentification, l'autorisation, la création de comptes utilisateur, la gestion des profils et d'autres opérations liées aux utilisateurs de l'application. En adoptant une approche microservices, le service User fonctionne de manière indépendante, permettant le développement, le déploiement et l'évolution autonomes de ses fonctionnalités. L'architecture microservices favorise ainsi une gestion flexible et modulaire des aspects liés aux utilisateurs, améliorant la scalabilité, la sécurité et la maintenance de l'application. La création d'un service User répond à la nécessité de centraliser et de rationaliser les opérations associées aux utilisateurs, offrant une solution dédiée pour la gestion efficace de la partie utilisateur au sein de l'écosystème de l'application.

**3.Mécanismes de communication :**

Dans l'architecture microservices de notre application, le mécanisme de communication entre les services repose sur l'utilisation du framework Angular. Angular offre une solution élégante pour établir des interactions fluides entre les différentes parties de l'application. Il facilite la communication asynchrone entre les services côté client en utilisant des requêtes HTTP ou d'autres mécanismes tels que les websockets.

Lorsqu'un utilisateur interagit avec l'interface utilisateur développée en Angular, des requêtes HTTP sont générées et envoyées aux services correspondants pour effectuer des opérations spécifiques, que ce soit pour calculer des Travaux Non Bâtis (TNB) ou gérer les utilisateurs. Ces requêtes sont ensuite traitées par les microservices correspondants, qui renvoient les données nécessaires en réponse. La gestion des appels réseau est simplifiée grâce à Angular, qui offre des fonctionnalités telles que les Observables pour gérer les flux de données asynchrones de manière efficace.

L'utilisation d'Angular pour la communication entre services garantit une expérience utilisateur réactive et fluide, tout en permettant une séparation claire des responsabilités entre les composants côté client et les microservices côté serveur. Cette approche favorise la modularité, l'évolutivité et la maintenabilité de notre application, offrant ainsi une architecture robuste et adaptée à la nature distribuée de notre système basé sur microservices.

# **Chapitre 2 :**

**Conception Microservices**

**MicroService User :**

**Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Description générée automatiquement**

**MicroService TNB :**

**Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement**

# **Chapitre 3 :**

**Conterneurisation Docker**

**Implémentation :**

Docker compose :

**Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement**

**Après build maven on utilise le docker build pour chaque service**

Service TNB :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Service User :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

**Avantages :**

**1. Isolation des Environnements :**

Docker utilise des conteneurs légers qui encapsulent une application et ses dépendances. Chaque conteneur est isolé, assurant que les différences entre les environnements de développement, de test et de production sont minimisées.

**2. Portabilité :**

Les conteneurs Docker sont portables, ce qui signifie qu'ils peuvent être exécutés de manière cohérente sur n'importe quel système prenant en charge Docker, qu'il s'agisse d'un environnement de développement local, d'un serveur de test ou d'un cluster de production.

**3. Déploiement Rapide :**

Docker permet un déploiement rapide des applications en encapsulant tous les éléments nécessaires dans un conteneur. Cela simplifie le déploiement sur différentes plates-formes et réduit les éventuels conflits de configuration.

**4. Gestion des Dépendances :**

Les conteneurs Docker incluent toutes les dépendances requises, éliminant ainsi les conflits potentiels entre les versions des bibliothèques et des frameworks. Cela facilite la gestion des dépendances et réduit les problèmes liés à la compatibilité des versions.

**5. Scalabilité Horizontale :**

La nature légère des conteneurs Docker permet une mise à l'échelle horizontale efficace. Vous pouvez déployer plusieurs instances de conteneurs pour gérer des charges de travail croissantes de manière flexible et rapide.

**6. Intégration Continue et Déploiement Continu (CI/CD) :**

Docker s'intègre bien avec les pipelines CI/CD, automatisant ainsi les processus de construction, de test et de déploiement. Cela accélère le cycle de développement et garantit une livraison continue des applications.

# **Chapitre 4 :**

**CI/CD**

**Processus :**

### CI/CD avec Jenkins et GitLab :

**1. Dépôt de Code (GitLab) :**

Le code source de l'application est stocké dans un référentiel GitLab.

**2. Configuration du Projet dans Jenkins :**

Configurez un nouveau projet dans Jenkins, en liant le projet à votre référentiel GitLab.

**3. Webhooks GitLab-Jenkins :**

Configurez des webhooks dans GitLab pour déclencher Jenkins à chaque modification du code source.

**4. Build (CI) :**

Configurez une tâche de build dans Jenkins pour compiler le code source et générer un artefact exécutable (comme un fichier JAR, WAR, etc.).

**5. Déploiement (CD) :**

Configurez des tâches de déploiement dans Jenkins pour déployer l'application sur des environnements de test ou de production. Cela peut inclure le déploiement sur des serveurs, des conteneurs Docker, ou même sur des plates-formes cloud.

**Configuration :**

**Une image contenant texte, document, capture d’écran, reçu

Description générée automatiquement**

**Conclusion**

**Résumé des Accomplissements :**

Au cours du développement de notre application basée sur microservices, plusieurs accomplissements significatifs ont été réalisés. L'adoption de l'architecture microservices, avec des services dédiés pour les Travaux Non Bâtis (TNB) et la gestion des utilisateurs, a permis une conception modulaire et évolutive. L'utilisation de Spring Boot pour les microservices et Angular pour l'interface utilisateur a facilité le développement rapide et la création d'une application réactive. De plus, l'intégration de Docker dans le processus de déploiement a amélioré la portabilité, l'isolation et la gestion des dépendances, optimisant ainsi l'efficacité du déploiement.

**Perspectives Futures :**

Pour les perspectives futures de l'application, plusieurs pistes d'amélioration peuvent être envisagées. Tout d'abord, l'intégration de tests automatisés, y compris des tests unitaires, d'intégration et de bout en bout, renforcera la qualité du code et la stabilité de l'application. En parallèle, l'exploration de mécanismes avancés de gestion des erreurs, la mise en place de fonctionnalités de suivi des performances en temps réel et l'optimisation des requêtes entre microservices peuvent contribuer à une expérience utilisateur encore plus robuste.

Sur le plan de la sécurité, la mise en œuvre d'une stratégie complète d'authentification et d'autorisation, ainsi que l'intégration de la gestion des certificats, peuvent renforcer la protection des données sensibles. En outre, la mise en place de mécanismes de sauvegarde et de reprise après sinistre peut assurer la disponibilité continue de l'application, même dans des scénarios défavorables.

Enfin, en tenant compte de l'évolutivité, envisager l'intégration de technologies émergentes, l'exploration de nouveaux cas d'utilisation pour les microservices existants, et l'évaluation constante de l'architecture pour garantir sa pertinence face aux évolutions du domaine d'application. Dans l'ensemble, ces perspectives futures visent à renforcer l'application, à répondre aux besoins changeants de manière agile et à maintenir une expérience utilisateur exceptionnelle.