ABOUEL MAKARIM ADNANE
AKHATAR EL MEHDI 5IIR - G4

Rapport du projet Ecommerce

1. Introduction

Aperçu du Projet :

Le projet décrit une architecture basée sur des microservices en utilisant Spring boot en backend et Angular js en frontend. Il est conçu pour gérer un système e-commerce comprenant des fonctionnalités telles que la gestion de produits, des facturations, des notifications, et une passerelle pour l'interface utilisateur. Chaque aspect du système est géré par un microservice distinct, favorisant une conception modulaire et évolutive.

Importance de l'Architecture Microservices :

L'adoption de l'architecture microservices revêt une importance significative dans le développement logiciel moderne. Voici quelques raisons clés pour lesquelles cette approche est privilégiée :

1. Évolutivité et Agilité : Les microservices permettent une évolutivité facile. Chaque microservice peut être développé, déployé indépendamment des autres, offrant ainsi une agilité considérable dans le développement et la maintenance des applications.
2. Déploiement Indépendant : Chaque microservice est une unité autonome, ce qui signifie qu'il peut être déployé indépendamment des autres. Cela facilite les mises à jour continues sans perturber l'ensemble du système.
3. Technologies Variées : Les microservices permettent l'utilisation de différentes technologies et langages de programmation pour

chaque service en fonction de ses besoins spécifiques. Cela favorise la flexibilité et l'optimisation des performances.

4. **Isolation des Erreurs :** L'architecture microservices isole les erreurs. Si un microservice échoue, cela n'affecte pas nécessairement l'ensemble du système. Les autres services peuvent continuer à fonctionner normalement.
5. **Facilité de Maintenance :** La maintenance et l'évolution des fonctionnalités sont plus faciles à gérer, car chaque microservice peut être développé et testé de manière indépendante.
6. **Scalabilité Fine :** La capacité de chaque microservice à être mis à l'échelle indépendamment permet une gestion plus précise des ressources en fonction des besoins spécifiques de chaque composant du système.

2. Architecture Microservices

Architecture :

L'architecture est basée sur des microservices. Elle comporte plusieurs microservices, chacun étant responsable d'une partie spécifique de la logique métier. Les services sont orchestrés via un système de conteneurisation **Docker** et un service de découverte de services **Consul**.

Description des services :

1. **Store (Gateway) :**
   - **Type d'application** : Gateway
   - **Responsabilités** : Gère l'interface utilisateur, la gestion des clients (utilisateurs), et orchestre les interactions avec d'autres services.
   - Utilise **Consul** pour la découverte de services.
   - Utilise **JWT** pour l'authentification.
2. **Product Microservice :**
   - **Type d'application** : Microservice

- **Responsabilités** : Gère les produits, catégories de produits, commandes de produits et détails des commandes.
- Utilise **Consul** pour la découverte de services.
- Utilise **JWT** pour l'authentification.

3. Invoice Microservice :
   - **Type d'application** : Microservice
   - **Responsabilités** : Gère les factures et les expéditions.
   - Utilise **Consul** pour la découverte de services.
   - Utilise **JWT** pour l'authentification.
4. Notification Microservice :
   - **Type d'application** : Microservice
   - **Responsabilités :** Gère les notifications liées aux produits.
   - Utilise **Consul** pour la découverte de services.
   - Utilise **JWT** pour l'authentification.

Mécanismes de communication :

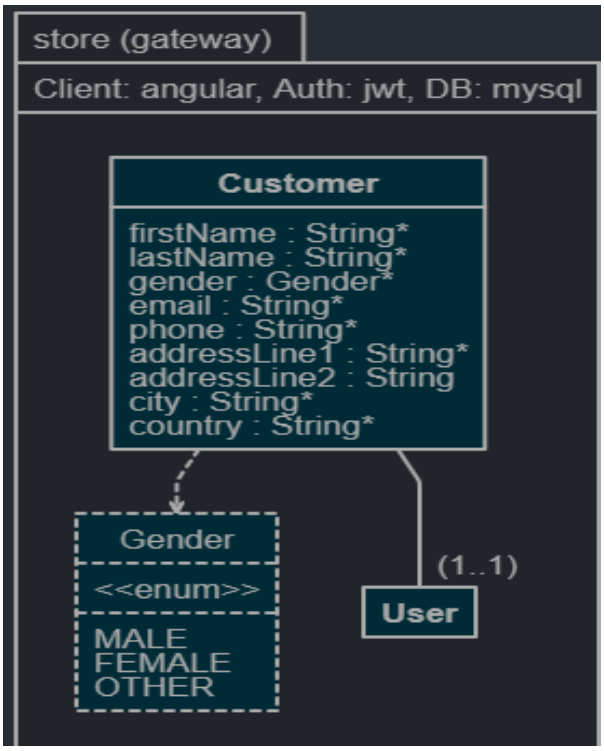
- La **communication** entre microservices se fait principalement par le biais de services **RESTful** exposés via **HTTP.**
- Les microservices utilisent **Consul** comme service de découverte pour localiser d'autres services.
- **JWT** est utilisé comme mécanisme d'authentification pour sécuriser les communications entre les microservices.

3. Conception des Microservices

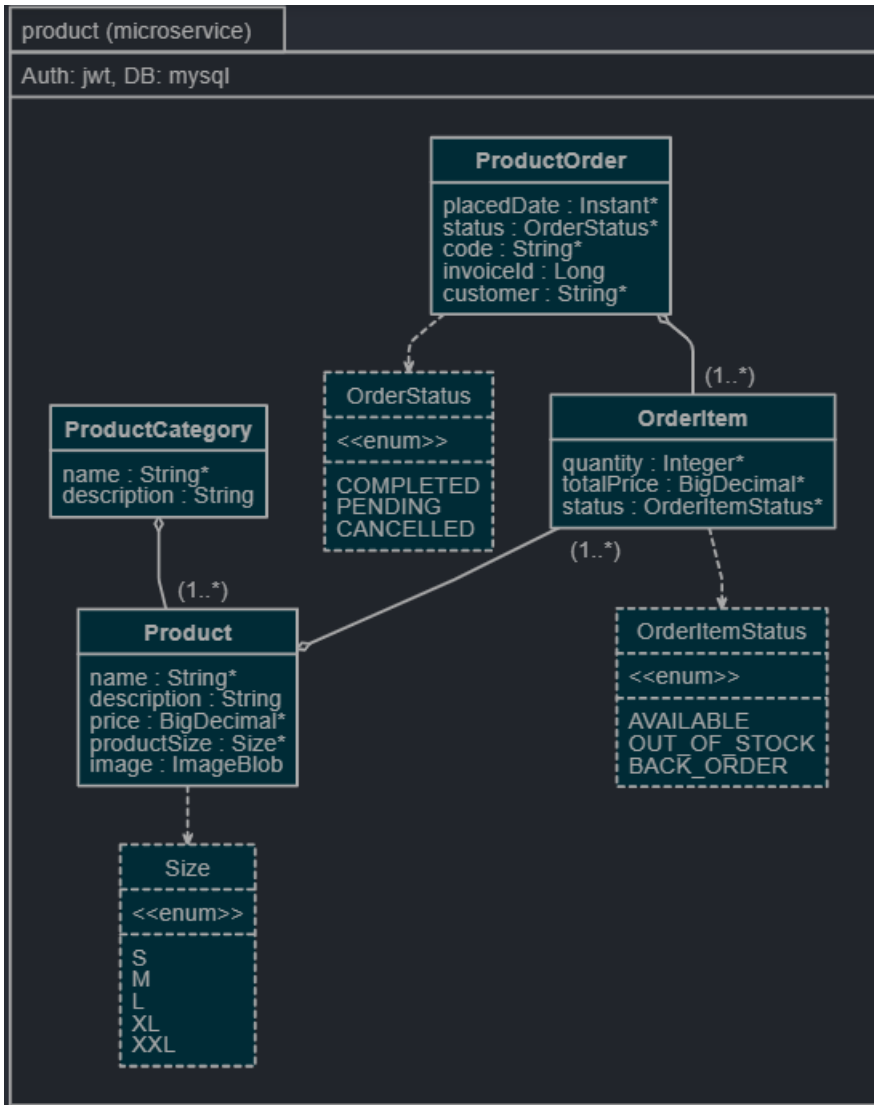
Approche de conception pour chaque service :

1. Store (Gateway) :

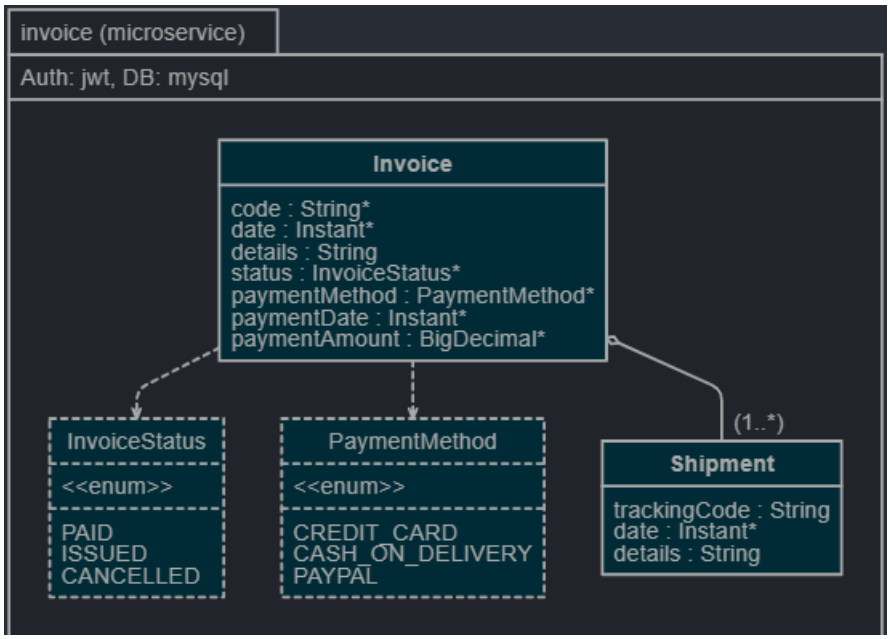
- Gère les interactions avec l'utilisateur.
- Utilise le service de découverte pour localiser les microservices backend.
- Utilise JWT pour l'authentification des utilisateurs.



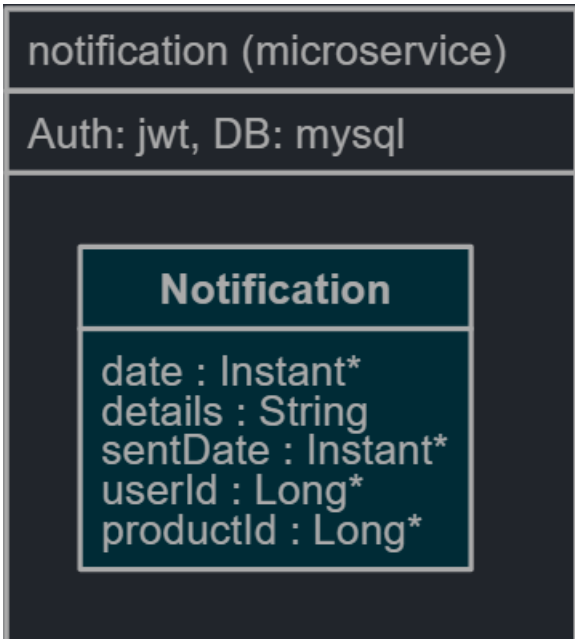
2. Product Microservice :
    - Gère la logique métier liée aux produits, catégories et commandes.
    - Utilise Consul pour la découverte de services.
    - Utilise JWT pour l'authentification des requêtes.



3. Invoice Microservice :
    - Gère la logique métier liée aux factures et expéditions.
    - Utilise Consul pour la découverte de services.
    - Utilise JWT pour l'authentification des requêtes.



4. Notification Microservice :
    - Gère la logique métier liée aux notifications de produits.
    - Utilise Consul pour la découverte de services.
    - Utilise JWT pour l'authentification des requêtes.



4. Conteneurisation Docker (Implémentation)

```
services:
  store:
    image: store
    environment:
      - _JAVA_OPTIONS=-Xmx512m -Xms256m
      - SPRING_PROFILES_ACTIVE=prod,api-docs
      - MANAGEMENT_PROMETHEUS_METRICS_EXPORT_ENABLED=true
      - SPRING_CLOUD_CONSUL_HOST=consul
      - SPRING_CLOUD_CONSUL_PORT=8500
      - SPRING_R2DBC_URL=r2dbc:mysql://store-mysql:3306/store?useUnicode=true&characterEncoding=utf8&useSSL=false&useLegacyDatetimeCode=false&createDatabaseIfNotExist=true
      - SPRING_LIQUIBASE_URL=jdbc:mysql://store-mysql:3306/store?useUnicode=true&characterEncoding=utf8&useSSL=false&useLegacyDatetimeCode=false&createDatabaseIfNotExist=true
    ports:
      - "8080:8080"
    healthcheck:
      test:
        - CMD
        - curl
        - "-f"
        - http://localhost:8080/management/health
      interval: 5s
      timeout: 5s
      retries: 40
    depends_on:
      store-mysql:
        condition: service_healthy
  store-mysql:
    image: mysql:8.2.0
```

```
  volumes:
    - ./config/mysql:/etc/mysql/conf.d
  environment:
    - MYSQL_ALLOW_EMPTY_PASSWORD=yes
    - MYSQL_DATABASE=store
  command: mysqld --lower_case_table_names=1 --skip-ssl --character_set_server=utf8mb4 --explicit_defaults_for_timestamp
  healthcheck:
    test:
      - CMD
      - mysql
      - "-e"
      - SHOW DATABASES;
    interval: 5s
    timeout: 5s
    retries: 10

product:
  image: product
  environment:
    - _JAVA_OPTIONS=-Xmx512m -Xms256m
    - SPRING_PROFILES_ACTIVE=prod,api-docs
    - MANAGEMENT_PROMETHEUS_METRICS_EXPORT_ENABLED=true
    - SPRING_CLOUD_CONSUL_HOST=consul
    - SPRING_CLOUD_CONSUL_PORT=8500
    - SPRING_DATASOURCE_URL=jdbc:mysql://product-mysql:3306/product?useUnicode=true&characterEncoding=utf8&useSSL=false&useLegacyDatetimeCode=false&createDatabaseIfNotExist=true
    - SPRING_LIQUIBASE_URL=jdbc:mysql://product-mysql:3306/product?useUnicode=true&characterEncoding=utf8&useSSL=false&useLegacyDatetimeCode=false&createDatabaseIfNotExist=true
```

```
  healthcheck:
    test:
      - CMD
      - curl
      - "-f"
      - http://localhost:8081/management/health
    interval: 5s
    timeout: 5s
    retries: 40
  depends_on:
    product-mysql:
      condition: service_healthy
product-mysql:
  image: mysql:8.2.0
  volumes:
    - ./config/mysql:/etc/mysql/conf.d
  environment:
    - MYSQL_ALLOW_EMPTY_PASSWORD=yes
    - MYSQL_DATABASE=product
  command: mysqld --lower_case_table_names=1 --skip-ssl --character_set_server=utf8mb4 --explicit_defaults_for_timestamp
  healthcheck:
    test:
      - CMD
      - mysql
      - "-e"
      - SHOW DATABASES;
    interval: 5s
    timeout: 5s
```

```
    retries: 10

 invoice:
   image: invoice
   environment:
     - _JAVA_OPTIONS=-Xmx512m -Xms256m
     - SPRING_PROFILES_ACTIVE=prod,api-docs
     - MANAGEMENT_PROMETHEUS_METRICS_EXPORT_ENABLED=true
     - SPRING_CLOUD_CONSUL_HOST=consul
     - SPRING_CLOUD_CONSUL_PORT=8500
     - SPRING_DATASOURCE_URL=jdbc:mysql://invoice-mysql:3306/invoice?useUnicode=true&characterEncoding=utf8&useSSL=false&useLegacyDatetimeCode=false&createDatabaseIfNotExist=true
     - SPRING_LIQUIBASE_URL=jdbc:mysql://invoice-mysql:3306/invoice?useUnicode=true&characterEncoding=utf8&useSSL=false&useLegacyDatetimeCode=false&createDatabaseIfNotExist=true
   healthcheck:
     test:
       - CMD
       - curl
       - "-f"
       - http://localhost:8082/management/health
     interval: 5s
     timeout: 5s
     retries: 40
   depends_on:
     invoice-mysql:
       condition: service_healthy
 invoice-mysql:
   image: mysql:8.2.0
   volumes:
     - ./config/mysql:/etc/mysql/conf.d
   environment:
     - MYSQL_ALLOW_EMPTY_PASSWORD=yes
```

```
    - MYSQL_DATABASE=invoice
  command: mysqld --lower_case_table_names=1 --skip-ssl --character_set_server=utf8mb4 --explicit_defaults_for_timestamp
  healthcheck:
    test:
      - CMD
      - mysql
      - "-e"
      - SHOW DATABASES;
    interval: 5s
    timeout: 5s
    retries: 10

notification:
  image: notification
  environment:
    - _JAVA_OPTIONS=-Xmx512m -Xms256m
    - SPRING_PROFILES_ACTIVE=prod,api-docs
    - MANAGEMENT_PROMETHEUS_METRICS_EXPORT_ENABLED=true
    - SPRING_CLOUD_CONSUL_HOST=consul
    - SPRING_CLOUD_CONSUL_PORT=8500
    - SPRING_DATASOURCE_URL=jdbc:mysql://notification-mysql:3306/notification?useUnicode=true&characterEncoding=utf8&useSSL=false&useLegacyDatetimeCode=false&createDatabaseIfNotExist=true
    - SPRING_LIQUIBASE_URL=jdbc:mysql://notification-mysql:3306/notification?useUnicode=true&characterEncoding=utf8&useSSL=false&useLegacyDatetimeCode=false&createDatabaseIfNotExist=true
  healthcheck:
    test:
      - CMD
      - curl
      - "-f"
      - http://localhost:8083/management/health
    interval: 5s
    timeout: 5s
```

```
      retries: 40
    depends_on:
      notification-mysql:
        condition: service_healthy
  notification-mysql:
    image: mysql:8.2.0
    volumes:
      - ./config/mysql:/etc/mysql/conf.d
    environment:
      - MYSQL_ALLOW_EMPTY_PASSWORD=yes
      - MYSQL_DATABASE=notification
    command: mysqld --lower_case_table_names=1 --skip-ssl --character_set_server=utf8mb4 --explicit_defaults_for_timestamp
    healthcheck:
      test:
        - CMD
        - mysql
        - "-e"
        - SHOW DATABASES;
      interval: 5s
      timeout: 5s
      retries: 10

  consul:
    image: docker.io/bitnami/consul:1.16.2
    ports:
      - 8300:8300
      - 8500:8500
      - 8600:8600
    command: consul agent -dev -ui -client 0.0.0.0 -log-level=INFO
  consul-config-loader:
```

```
image: jhipster/consul-config-loader:v0.4.1
volumes:
  - ./central-server-config:/config
environment:
  - INIT_SLEEP_SECONDS=5
  - CONSUL_URL=consul
  - CONSUL_PORT=8500
```

Avantages

La conteneurisation de cette application avec Docker présente plusieurs avantages, en particulier dans le contexte d'une architecture basée sur des microservices. Voici quelques-uns des avantages spécifiques à la conteneurisation avec Docker pour cette application :

1. Isolation : Chaque microservice peut être encapsulé dans son propre conteneur Docker, assurant ainsi une isolation complète. Cela signifie que les dépendances, bibliothèques et configurations spécifiques à chaque microservice sont encapsulées, évitant ainsi les conflits potentiels avec d'autres services.
2. Portabilité : Les conteneurs Docker sont portables, ce qui signifie que l'application peut être exécutée de manière cohérente sur n'importe quel environnement qui prend en charge Docker. Cela facilite le déploiement sur différents environnements, que ce soit en développement, en test ou en production.
3. Facilité de Déploiement : La création, la distribution et le déploiement des conteneurs Docker sont rapides et efficaces. Cela simplifie considérablement le processus de déploiement des microservices, permettant des mises à jour rapides et la gestion de versions.
4. Gestion des Dépendances : Les conteneurs Docker permettent de spécifier et de gérer de manière explicite les dépendances et les versions des logiciels nécessaires à chaque microservice. Cela évite les problèmes potentiels liés à des différences d'environnement.

5. **Gestion des Ressources** : Docker offre des outils pour surveiller et gérer efficacement l'utilisation des ressources par chaque conteneur. Cela permet une gestion fine des performances et une meilleure optimisation des ressources système.
6. **Intégration avec l'Écosystème** : Docker s'intègre bien avec d'autres outils et services, facilitant ainsi l'intégration continue, la livraison continue (CI/CD), et la gestion des configurations.

5. CI/CD avec Jenkins

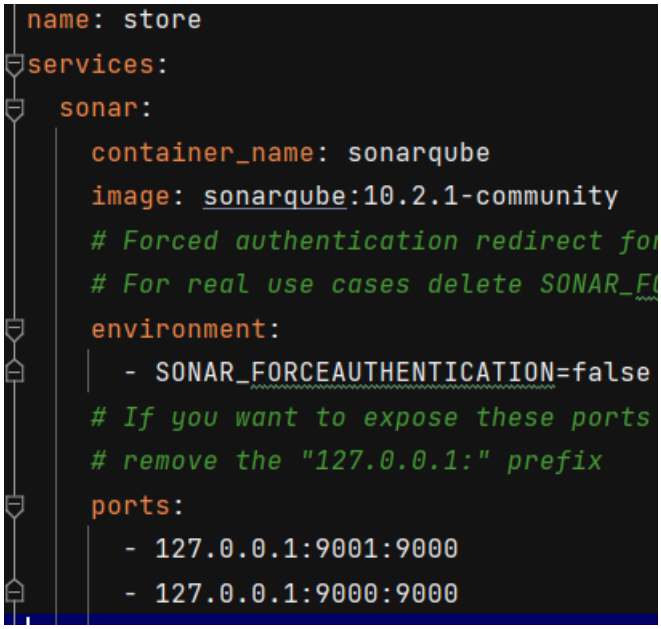
Configuration

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                script {
                    checkout([$class: 'GitSCM', branches: [[name: 'main']], userRemoteConfigs: [[url: 'https://github.com/RevSt0/Ecommerce
                }
            }
        }

        stage('Build and Deploy Microservices') {
            steps {
                script {
                    def microservices = ['store', 'notification', 'invoice', 'product']

                    dir('store'){
                        bat 'git config --global user.email "jenkins@pipe.com"'
                        bat 'git config --global user.name "Jenkins"'
                        bat 'git init'
                        bat 'git add .'
                        //bat 'git commit -m "commit"'
                        //bat 'npm install'
                        //bat 'npm install -g @angular/cli'
                        //bat 'npm run webapp:test'
                    }

                    for (microservice in microservices) {
                        echo "Building and deploying ${microservice}"
                        dir("${microservice}") {
                            bat './mvnw -ntp -Pprod verify jib:dockerBuild'
                            bat 'start java -jar target/${microservice}.jar'
                        }
                    }
                }
            }
        }
    }
```

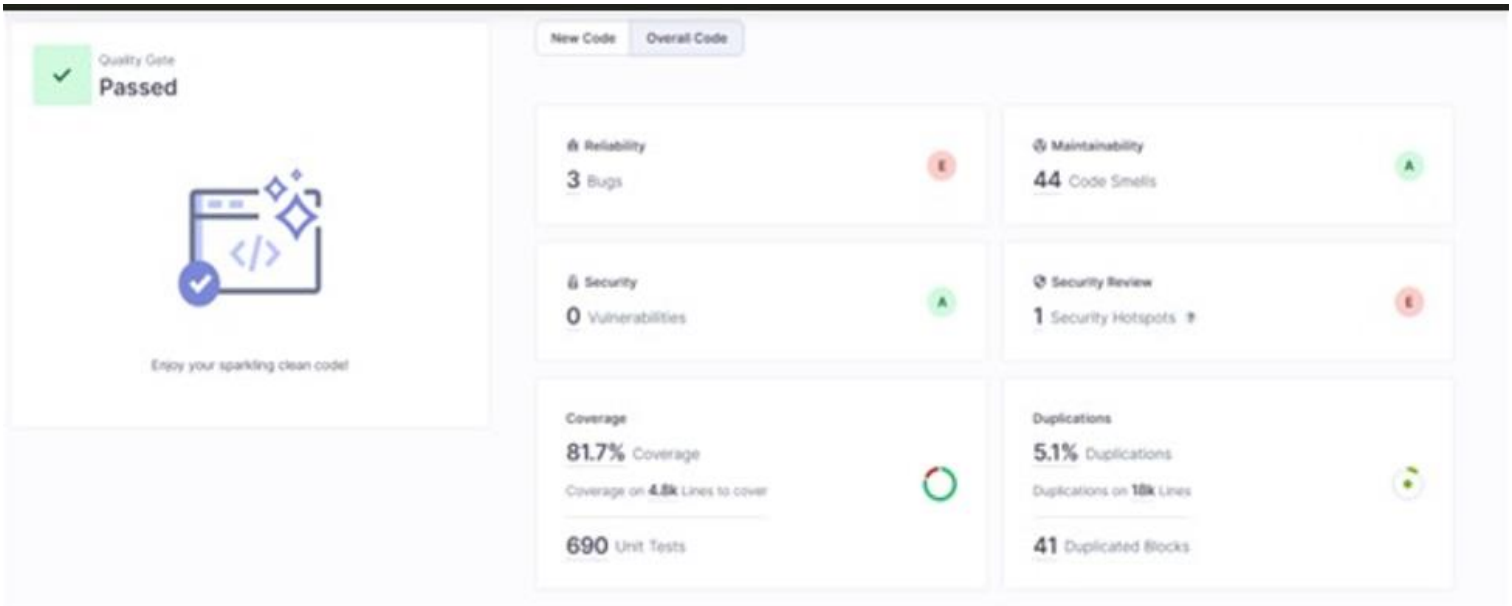
```
        stage('Build Angular') {
            steps {
                dir('store') {
                    bat 'npm install'
                    bat 'ng build'
                }
            }
        }
    }

    post {
        success {
            echo 'Le pipeline a été exécuté avec succès! Félicitations!'
        }

        failure {
            echo 'Le pipeline a échoué. Veuillez vérifier les logs pour plus d\'informations.
        }
    }
}
```

7. Intégration de SonarQube

7.1 Configuration (sonar.yml)



7.2 Rapport SonarQube



8. Conclusion

Résumé des Accomplissements :

Le projet a abouti à la création d'une architecture microservices, avec une mise en œuvre spécifique pour un système e-commerce.

1. **Microservices :** Les différentes parties fonctionnelles de l'application sont isolées dans des microservices distincts, favorisant la modularité et la gestion indépendante des fonctionnalités.
2. **Conteneurisation avec Docker :** L'application est conteneurisée avec Docker, ce qui offre une isolation efficace, une portabilité et une gestion simplifiée des déploiements.
3. **Service de Découverte avec Consul :** La communication entre les microservices est facilitée grâce à l'utilisation de Consul en tant que service de découverte, simplifiant la localisation des services.
4. **Authentification avec JWT :** La sécurité est renforcée grâce à l'authentification basée sur JWT (JSON Web Token), garantissant des échanges sécurisés entre les microservices.

5. **Utilisation de Divers Services** : Les microservices gèrent divers aspects tels que les produits, les factures, les notifications, et la passerelle pour l'interface utilisateur, reflétant une architecture complète pour un système e-commerce.

Perspectives Futures :

1. **Optimisation des Performances** : Explorer des stratégies d'optimisation des performances, y compris la mise en cache plus avancée, pour garantir une réponse rapide aux demandes des utilisateurs.
2. **Évolutions Fonctionnelles** : Ajouter de nouvelles fonctionnalités en réponse aux besoins du marché ou des utilisateurs, tout en continuant à maintenir la modularité des microservices.
3. **Tests et Assurance Qualité** : Renforcer les pratiques de tests automatisés pour garantir la fiabilité et la stabilité de l'application à mesure qu'elle évolue.
4. **Monitoring et Maintenance** : Mettre en place des solutions de monitoring pour surveiller la santé des microservices en production, permettant une maintenance proactive.