ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

RAPPORT Projet

2024

Conception et réalisation d'une application pour la gestion de Board Kanban

Architecture des composants d'entreprise

Réalisé par :

MAZZOUZ Chaimae

BOUTISSANTE Issam

Groupe : 5

1. Introduction

1.1 Aperçu du projet

1.1.1 Introduction :

L'application Web Kanban à architecture microservice est un outil de gestion de projet visuel qui permet aux utilisateurs de suivre leurs tâches et projets en temps réel. L'application est conçue pour être flexible et adaptable, afin de répondre aux besoins des utilisateurs de tous types.

1.1.2 Fonctionnalités :

Ce projet consiste en une application Web Kanban développée à l'aide d'une architecture microservice. Il offre les fonctionnalités suivantes :

- Authentification sécurisée : Les utilisateurs doivent s'authentifier pour accéder à l'application, garantissant la confidentialité et la protection des données.
- Création de tableaux Kanban : Les utilisateurs peuvent créer des tableaux Kanban personnalisés pour organiser leurs tâches et projets.
- Gestion des tâches : Sur chaque tableau, les utilisateurs peuvent ajouter, modifier et supprimer des tâches. Ils peuvent également déplacer les tâches entre les différentes étapes du flux Kanban (à faire, en cours, terminé).
- Architecture microservice : L'application est conçue autour d'une architecture microservice, ce qui permet une meilleure modularité, évolutivité et facilité de maintenance.

1.1.3 Objectifs du projet :

- Concevoir et développer une application Web Kanban fonctionnelle et conviviale.
- Mettre en œuvre une architecture microservice efficace pour gérer les différents composants de l'application.
- Assurer une authentification sécurisée pour protéger les données des utilisateurs.

- Fournir aux utilisateurs un outil flexible pour la gestion de leurs tâches et projets.

1.1.4 Equipe du projet :

L'application Web Kanban à architecture microservice a été développée par une équipe de deux étudiants :

- BOUTISSANTE Issam
- MAZZOUZ Chaimae

1.1.5 Conclusion :

L'application Web Kanban à architecture microservice est un outil de gestion de projet visuel et convivial qui répond aux besoins des utilisateurs de tous types. L'application est conçue pour être flexible et adaptable, afin de s'adapter aux différentes exigences des utilisateurs.

1.2 Importance de l'architecture microservices :

L'architecture microservices est un type d'architecture d'application dans laquelle l'application est développée sous la forme d'un ensemble de services indépendants. Ces services sont généralement petits et auto-suffisants, et ils communiquent entre eux via des API.

L'architecture microservices présente de nombreux avantages, notamment :

- La modularité : Les microservices sont des composants indépendants, ce qui permet de les développer, de les déployer et de les gérer de manière indépendante. Cela rend l'application plus flexible et adaptable aux changements.
- L'évolutivité : Les microservices peuvent être mis à l'échelle de manière indépendante, ce qui permet de répondre aux besoins croissants de l'application.
- La résilience : Les microservices peuvent être isolés les uns des autres, ce qui permet de limiter les perturbations en cas de défaillance d'un microservice.
- La maintenabilité : Les microservices sont plus faciles à maintenir que les applications monolithiques, car ils sont plus petits et plus simples.

En résumé, l'architecture microservices est un choix judicieux pour les applications qui doivent être :

- Flexibles et adaptables
- Évolutives
- Résilientes
- Maintenables

L'architecture microservices est une approche de développement d'applications qui gagne en popularité, car elle offre de nombreux avantages par rapport aux architectures monolithiques traditionnelles.

2. Architecture Microservices

2.1 Architecture

Notre Figure 1 illustre l'architecture microservices de notre application de tableau Kanban.

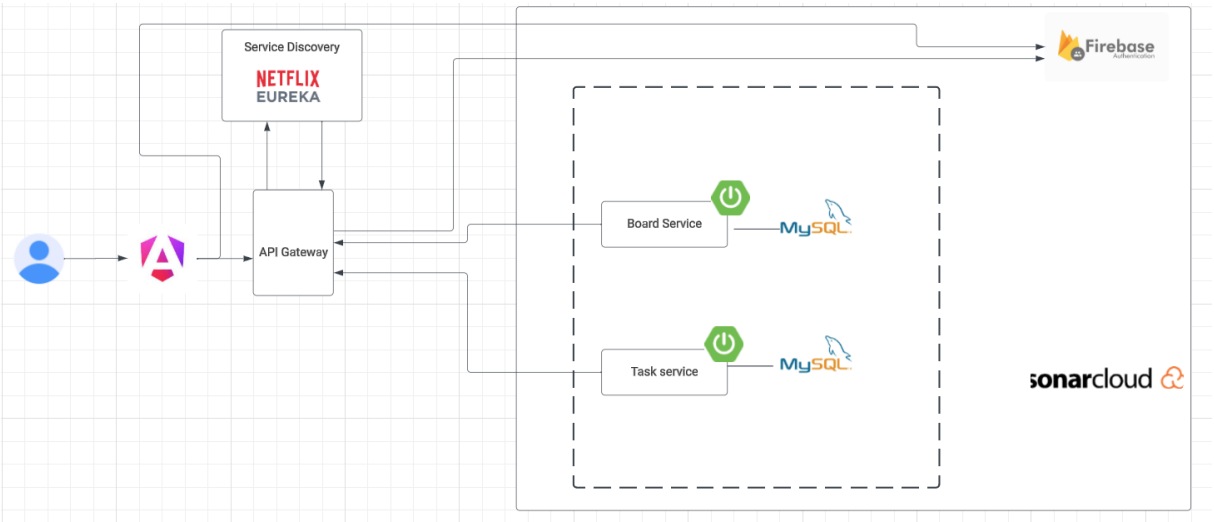


Figure 1: Architecture Microservices

2.1.1 Composants clés :

- **Application Angular :** Interface utilisateur frontale développée avec Angular, c'est là que l'utilisateur interagira avec l'application.
- **Passerelle API (API Gateway) :** Point d'entrée centralisé pour toutes les requêtes API, gérant l'authentification, le routage et la répartition de charge.
- **Firebase Auth :** Assure une authentification sécurisée des utilisateurs.
- **Service de tableaux :** Microservice dédié à la gestion des tableaux Kanban, y compris leur création, récupération, mise à jour et suppression.

- **Service de tâches :** Microservice responsable de la gestion des tâches au sein des tableaux, prenant en charge les opérations CRUD (création, lecture, mise à jour et suppression) des tâches.
- **Bases de données MySQL :** Chaque microservice (service de tableaux et service de tâches) dispose de sa propre base de données MySQL indépendante pour stocker et gérer ses données.
- **Registre de services Eureka :** Facilite la découverte de services, permettant aux microservices de se localiser et de communiquer entre eux dynamiquement.

2.1.2 Flux architectural :

1. **Interaction utilisateur :** L'utilisateur interagit avec l'application Angular, initiant des actions telles que la connexion, la création de tableaux, la gestion des tâches, etc.
2. **Passerelle API :** plaque tournante centrale : L'application Angular envoie des requêtes API à la passerelle API. La passerelle API, quant à elle :
    - Authentifie l'utilisateur à l'aide de Firebase Auth.
    - Route les requêtes vers les microservices appropriés en fonction de leurs points de terminaison.
    - Agrège les réponses de plusieurs microservices si nécessaire.
3. **Microservices et bases de données :**
    - Service de tableaux :
        - Traite les requêtes liées aux tableaux.
        - Interagit avec sa base de données MySQL dédiée pour stocker et récupérer les données des tableaux.
    - Service de tâches :
        - Traite les requêtes liées aux tâches.
        - Interagit avec sa propre base de données MySQL pour gérer les données des tâches.

4. **Découverte de services avec Eureka :** Les microservices s'enregistrent auprès d'Eureka au démarrage. La passerelle API interroge Eureka pour localiser les microservices nécessaires au routage des requêtes.
5. **Flux de données** : Les microservices échangent des données entre eux si nécessaire pour répondre aux requêtes des utilisateurs. Chaque microservice interagit avec sa base de données respective pour stocker et récupérer des données.

2.1.3 Conclusion :

Cette architecture microservices prend efficacement en charge les fonctionnalités de l'application de tableau Kanban et offre des avantages en termes de modularité, de scalabilité, de résilience et d'indépendance technologique.

2.2 Description des services

2.2.1 Service de tâches

Le service de tâches est un microservice responsable de la gestion des tâches au sein des tableaux Kanban. Il prend en charge les opérations CRUD (création, lecture, mise à jour et suppression) des tâches.

2.2.1.1 Fonctionnalités

Les principales fonctionnalités du service de tâches sont les suivantes :

- Créer une nouvelle tâche
- Lire les détails d'une tâche existante
- Mettre à jour les détails d'une tâche existante
- Supprimer une tâche existante

2.2.1.2 Flux de données

Le flux de données du service de tâches est le suivant :

1. L'application Angular envoie une requête API au service de tâches.
2. Le service de tâches valide la requête et effectue l'opération demandée.
3. Le service de tâches renvoie la réponse à l'application Angular.

2.2.1.3 Exemples d'utilisation

Voici quelques exemples d'utilisation du service de tâches :

- Un utilisateur crée une nouvelle tâche en entrant un titre, une description et une date de création.
- Un utilisateur met à jour les détails d'une tâche en modifiant son titre ou sa description.
- Un utilisateur supprime une tâche.

2.2.2 Service de tableaux

Le service de tableaux est un microservice responsable de la gestion des tableaux Kanban. Il prend en charge les opérations CRUD (création, lecture, mise à jour et suppression) des tableaux.

2.2.2.1 Fonctionnalités

Les principales fonctionnalités du service de tableaux sont les suivantes :

- Créer un nouveau tableau
- Lire les détails d'un tableau existant
- Mettre à jour les détails d'un tableau existant
- Supprimer un tableau existant

2.2.2.2 Flux de données

Le flux de données du service de tableaux est le suivant :

1. L'application Angular envoie une requête API au service de tableaux.
2. Le service de tableaux valide la requête et effectue l'opération demandée.
3. Le service de tableaux renvoie la réponse à l'application Angular.

2.2.2.3 Exemples d'utilisation

Voici quelques exemples d'utilisation du service de tableaux :

- Un utilisateur crée un nouveau tableau en entrant un titre et une description.
- Un utilisateur met à jour les détails d'un tableau en modifiant son titre ou sa description.
- Un utilisateur supprime un tableau.

2.2.3 Conclusion

Les services de tâches et de tableaux sont des éléments essentiels de l'architecture microservices de l'application de tableau Kanban. Ils permettent de gérer efficacement les tâches et les tableaux, en offrant une grande flexibilité et évolutivité.

2.3 Mécanismes de communication

Mécanismes de communication entre les services : Indépendance et cohérence des données. Bien que les services de tâches et de tableaux soient conçus pour fonctionner de manière indépendante, il est essentiel de garantir la cohérence des données entre eux. Pour ce faire, nous adoptons les mécanismes suivants :

1. Couplage par données :

- Chaque service gère sa propre base de données, mais les données sont logiquement liées par des identifiants partagés.
- Le service de tableaux stocke l'identifiant de l'utilisateur (userid) pour associer chaque tableau à un utilisateur spécifique.
- Le service de tâches stocke l'identifiant du tableau (boardid) pour associer chaque tâche au tableau correspondant.

2. Orchestration centralisée par la passerelle API :

- La passerelle API agit comme un chef d'orchestre, garantissant la coordination des requêtes et la cohérence des données entre les services.

- Lors de la récupération des tâches d'un tableau, la passerelle API effectue des requêtes simultanées aux services de tableaux et de tâches, puis fusionne les résultats de manière cohérente.
- Ceci permet de présenter à l'utilisateur une vue unifiée des données, même si elles proviennent de deux services distincts.

3. Validation des données et gestion des erreurs :

- Des règles de validation strictes sont appliquées au niveau de chaque service afin de préserver l'intégrité des données.
- La passerelle API gère les erreurs potentielles de communication ou d'incohérence de données, en renvoyant des messages d'erreur appropriés à l'application Angular.

En conclusion, cette approche garantit une architecture microservices découplée et évolutive, tout en maintenant la cohérence des données entre les services de tâches et de tableaux.

3. Conception des Microservices

3.1 Approche de conception pour chaque service

3.1.1 Service de tableaux

3.1.1.1 Responsabilités :

- Gérer la création, la lecture, la mise à jour et la suppression des tableaux Kanban.
- Associer chaque tableau à un utilisateur spécifique.

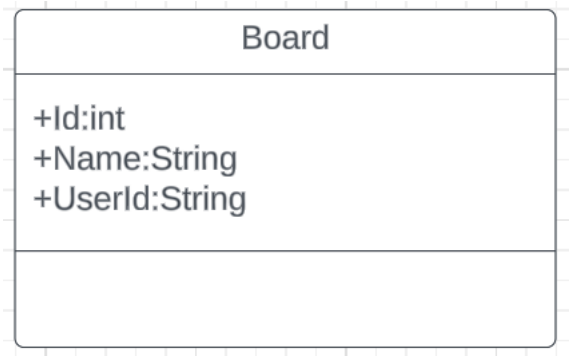


Figure 2 : Table Board

3.1.1.2 Données :

- Id (int) : Identifiant unique du tableau.
- Name (String) : Nom du tableau.
- UserId (String) : Identifiant de l'utilisateur propriétaire du tableau.

3.1.1.3 API :

- GET /boards : Récupérer la liste de tous les tableaux d'un utilisateur.
- GET /boards/{id} : Récupérer les détails d'un tableau spécifique.
- POST /boards : Créer un nouveau tableau.
- PUT /boards/{id} : Mettre à jour les détails d'un tableau existant.
- DELETE /boards/{id} : Supprimer un tableau.

3.1.2 Service de tâches :

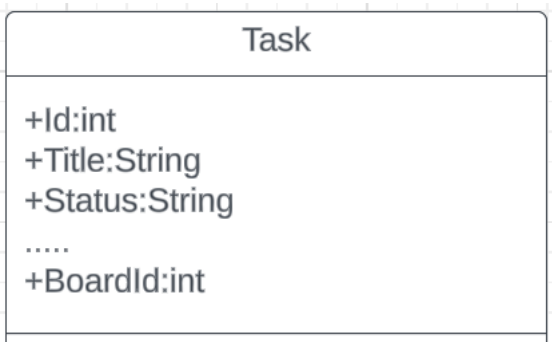


Figure 3 : Table Task

3.1.2.1 Responsabilités :

- Gérer la création, la lecture, la mise à jour et la suppression des tâches au sein des tableaux.
- Associer chaque tâche au tableau correspondant.

3.1.2.2 Données :

- Id (int): Identifiant unique de la tâche.
- Title (String): Titre de la tâche.

- Status (String): Statut de la tâche (à faire, en cours, terminée).
- BoardId (int): Identifiant du tableau auquel la tâche est associée.

3.1.2.3 API :

- GET /tasks/{boardId} : Récupérer la liste des tâches d'un tableau spécifique.
- GET /tasks/{id} : Récupérer les détails d'une tâche spécifique.
- POST /tasks : Créer une nouvelle tâche.
- PUT /tasks/{id} : Mettre à jour les détails d'une tâche existante.
- DELETE /tasks/{id} : Supprimer une tâche.

3.1.3 Considérations générales de conception :

- **Indépendance :** Chaque service est conçu pour fonctionner de manière autonome, avec sa propre base de données et son propre ensemble d'API.
- **Couplage par données** : Les services sont liés par des identifiants partagés (UserId et BoardId) pour assurer la cohérence des données.

4. Conteneurisation avec Docker

4.1 Implémentation et avantages

4.1.1 Implémentation

Api gateway :

```
Maven image to build the application
FROM maven:3.8.5-openjdk-17 as builder
WORKDIR /app

# Copy the pom.xml and source code
COPY pom.xml .
COPY src ./src

# Build the application
RUN mvn clean package -DskipTests

# Step 2: Use OpenJDK image to run the application
FROM openjdk:17-jdk-slim
```

```
WORKDIR /app

# Copy the built application from the builder stage
COPY --from=builder /app/target/api-gateway-0.0.1-SNAPSHOT.jar /app/api-
gateway.jar

# Expose the port the application runs on
EXPOSE 8888

# Command to run the application
CMD ["java", "-jar", "api-gateway.jar"]
```

Discovery Server

```
# Step 1: Use Maven image to build the application
FROM maven:3.8.5-openjdk-17 as builder
WORKDIR /app

# Copy the pom.xml and source code
COPY pom.xml .
COPY src ./src

# Build the application
RUN mvn clean package -DskipTests

# Step 2: Use OpenJDK image to run the application
FROM openjdk:17-jdk-slim
WORKDIR /app

# Copy the built application from the builder stage
COPY --from=builder /app/target/board-service-0.0.1-SNAPSHOT.jar /app/board-
service.jar

# Expose the port the application runs on
EXPOSE 8082

# Command to run the application
CMD ["java", "-jar", "board-service.jar"]
```

Service Board:

```
# Step 1: Use Maven image to build the application
FROM maven:3.8.5-openjdk-17 as builder
WORKDIR /app

# Copy the pom.xml and source code
COPY pom.xml .
COPY src ./src

# Build the application
RUN mvn clean package -DskipTests

# Step 2: Use OpenJDK image to run the application
FROM openjdk:17-jdk-slim
WORKDIR /app

# Copy the built application from the builder stage
COPY --from=builder /app/target/board-service-0.0.1-SNAPSHOT.jar /app/board-service.jar

# Expose the port the application runs on
EXPOSE 8082

# Command to run the application
CMD ["java", "-jar", "board-service.jar"]
```

Service Task:

```
# Dockerfile for Spring Boot app
FROM maven:3.8.4-openjdk-17 as build
WORKDIR /app
# No need to navigate into kanban-board-microservices/task-service, as we are already in the task-service directory
COPY pom.xml .
COPY src ./src
RUN mvn clean package -DskipTests

FROM openjdk:17-jdk-alpine
WORKDIR /app
# Make sure to copy from the correct build directory, assuming JAR file is in target/ under WORKDIR
COPY --from=build /app/target/task-service-*.jar /app/task-service.jar
EXPOSE 8084
ENTRYPOINT ["java","-jar","task-service.jar"]
```

Front Angular

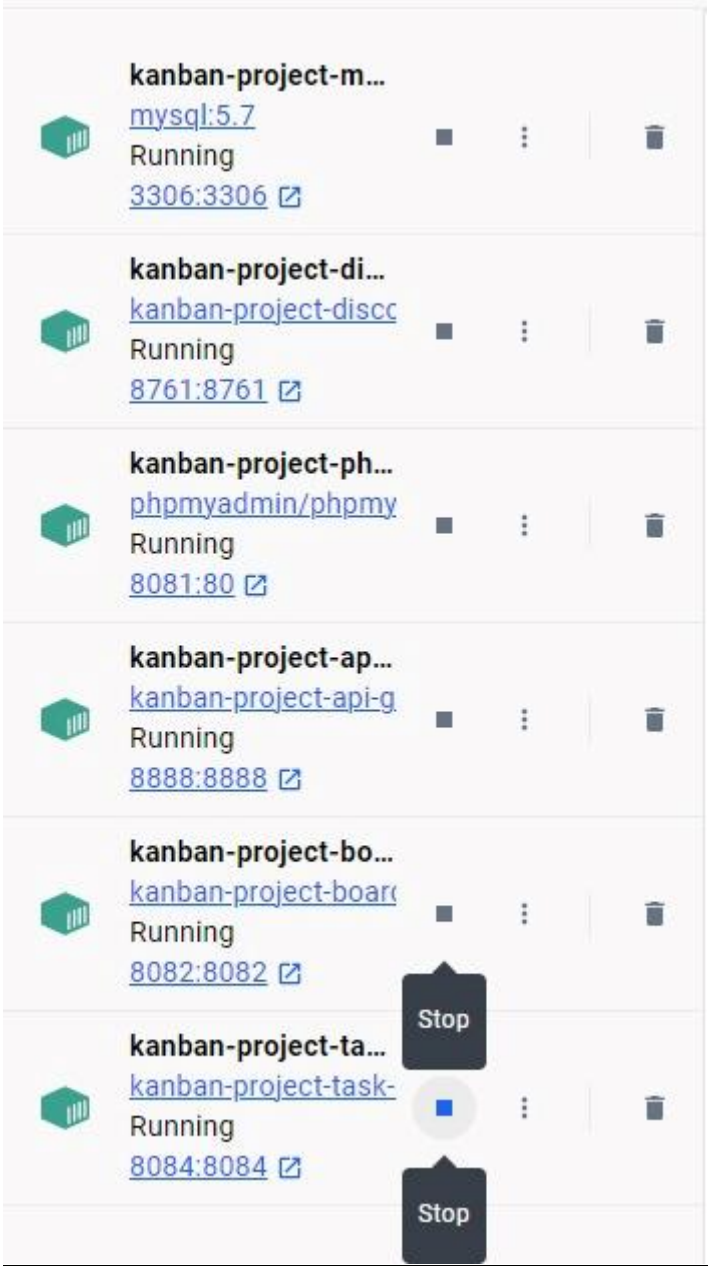
```
# Dockerfile for Angular app
FROM node:latest as build
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
RUN npm run build

# nginx state for serving content
FROM nginx:alpine
# Copy the build output to replace the default nginx contents.
COPY --from=build /app/dist/kanban-board /usr/share/nginx/html
# Expose port 4200 to the outside
EXPOSE 4200
# Use the default nginx.conf provided by the nginx image
COPY ./.angular/nginx.conf /etc/nginx/nginx.conf
CMD ["nginx", "-g", "daemon off;"]
```

Docker compose :

```
version: '3.8'

services:
  discovery-server:
    build:
      context: ./discovery-server
    ports:
      - "8761:8761"
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8761/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  api-gateway:
    build:
      context: ./api-gateway
    ports:
      - "8888:8888"
    depends_on:
      - discovery-server

  board-service:
    build:
      context: ./board-service
    ports:
```

```
      - "8082:8082"
    depends_on:
      discovery-server:
        condition: service_healthy
      mysql:
        condition: service_healthy
    environment:
      -
SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/board_service_db?allowPublicKeyR
etrieval=true&useSSL=false
      - SPRING_DATASOURCE_USERNAME=root
      - SPRING_DATASOURCE_PASSWORD=
      - SPRING_JPA_HIBERNATE_DDL_AUTO=create
      - SPRING_JPA_SHOW_SQL=true

  task-service:
    build:
      context: ./task-service
    ports:
      - "8084:8084"
    depends_on:
      discovery-server:
        condition: service_healthy
      mysql:
        condition: service_healthy
    environment:
      -
SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/task_service_db?allowPublicKeyRe
trieval=true&useSSL=false
      - SPRING_DATASOURCE_USERNAME=root
      - SPRING_DATASOURCE_PASSWORD=
      - SPRING_JPA_HIBERNATE_DDL_AUTO=create
      - SPRING_JPA_SHOW_SQL=true

  mysql:
    image: mysql:5.7
    ports:
      - "3306:3306"
    environment:
      - MYSQL_ALLOW_EMPTY_PASSWORD=yes
    volumes:
      - mysql-data:/var/lib/mysql
      - ./mysql-scripts:/docker-entrypoint-initdb.d
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-u", "root", "--
silent"]
      interval: 30s
      timeout: 10s
      retries: 5
```

```yaml
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
      - '8081:80'
    environment:
      - PMA_HOST=mysql
      - PMA_USER=root
      - PMA_PASSWORD=
    depends_on:
      - mysql

  kanban-board:
    build:
      context: ./kanban-board
      dockerfile: ./kanban-board/Dockerfile
    ports:
      - "4200:4200"
    depends_on:
      board-service:
        condition: service_healthy
      task-service:
        condition: service_healthy

volumes:
  mysql-data:
```



4.1.2 Avantages

La conteneurisation avec Docker offre de nombreux avantages, notamment :

- **Portabilité :** Les conteneurs sont portables, ce qui signifie qu'ils peuvent être exécutés sur n'importe quel système qui exécute Docker.
- **Scalabilité :** Les conteneurs peuvent être mis à l'échelle horizontalement en créant de nouveaux conteneurs.
- **Isolation :** Les conteneurs sont isolés les uns des autres, ce qui permet de réduire les risques de sécurité et de performance.

- **Simplicité :** La conteneurisation est une technique simple à mettre en œuvre et à gérer.

4.1.3 Conclusion

La conteneurisation avec Docker est une technologie puissante qui peut être utilisée pour améliorer la portabilité, la scalabilité, l'isolation et la simplicité des applications.

5. CI/CD avec Jenkins

5.1 Processus et configuration

Configuration:

Nous avons installé les plugins nécessaires dans Jenkins :

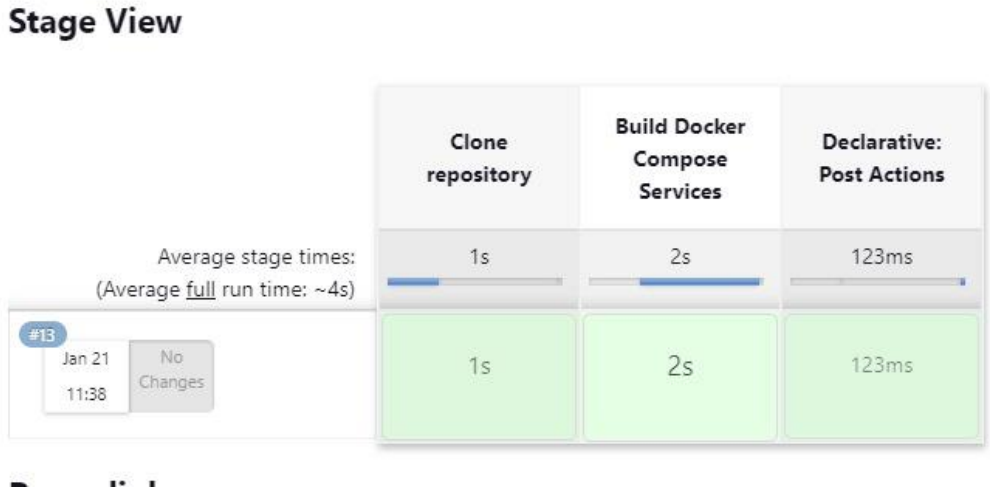
- Docker plugin
- Docker Pipeline plugin

Pipeline Script:

```
pipeline {
    agent any

    stages {
        stage('Clone repository') {
            steps {
                git 'https://github.com/issamBoutissante/kanban-project.git'
            }
        }

        stage('Build and Run Docker Compose') {
            steps {
                // Run Docker Compose up
                bat 'docker-compose -f docker-compose.yml up -d'
            }
        }
    }
```

```
    post {
        always {
            // Cleanup after pipeline execution
            bat 'docker-compose -f docker-compose.yml down'
        }
    }
}
```

Stage View

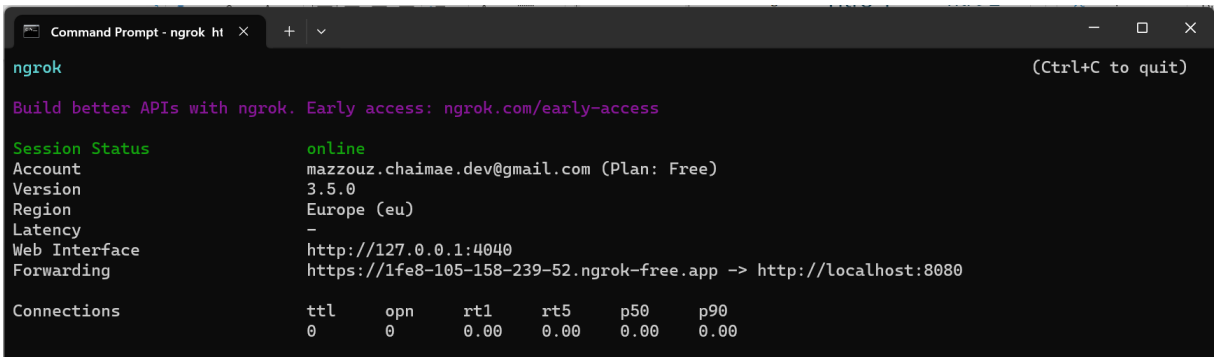


6. Déploiement Automatique

6.1 Utilisation de Ngrok

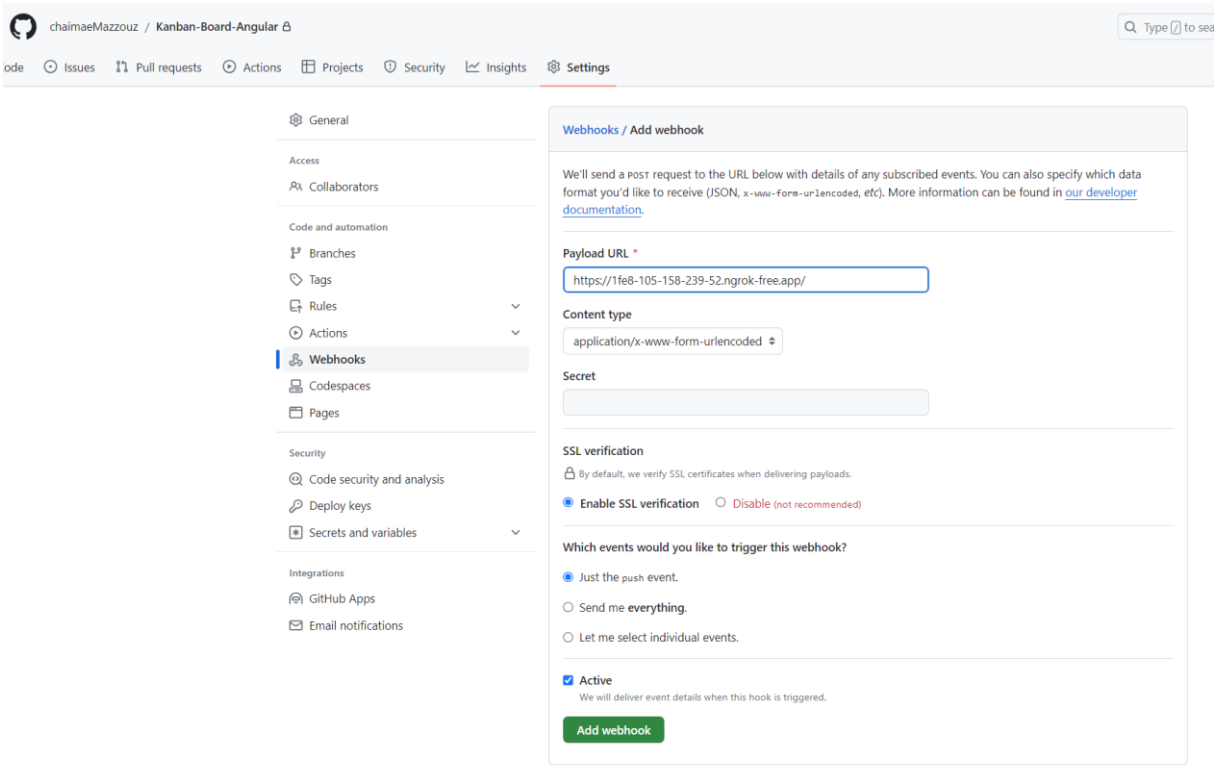
Étape 1 : Utiliser Ngrok

Démarrez Ngrok et créez un tunnel pour votre instance Jenkins :



URL publique générée par Ngrok :

Étape 2 : Créer un webhook dans GitHub



Étape 3 : Ajouter un Webhook GitHub dans Jenkins

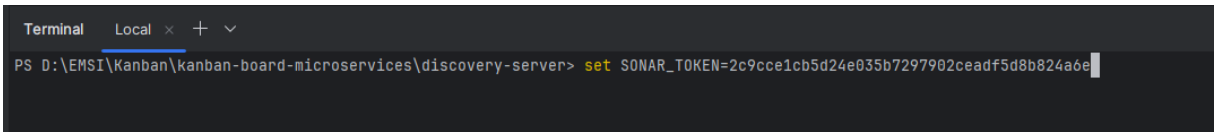
Dans l'URL Jenkins publiée, nous avons ajouté le lien vers le dépôt.

7. Intégration de SonarQube

7.1 Configuration et bénéfices pour la qualité du code

1er étape :

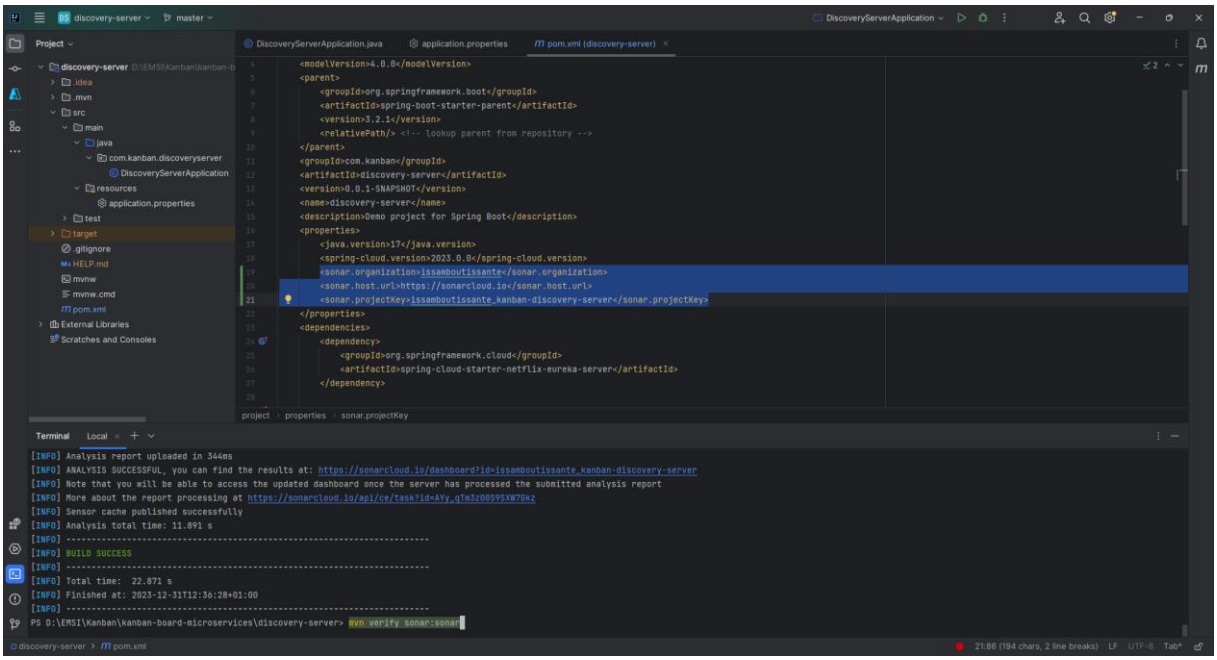
Après la création des projets pour les différents services dans SonarCloud, nous avons défini le sonarToken localement.



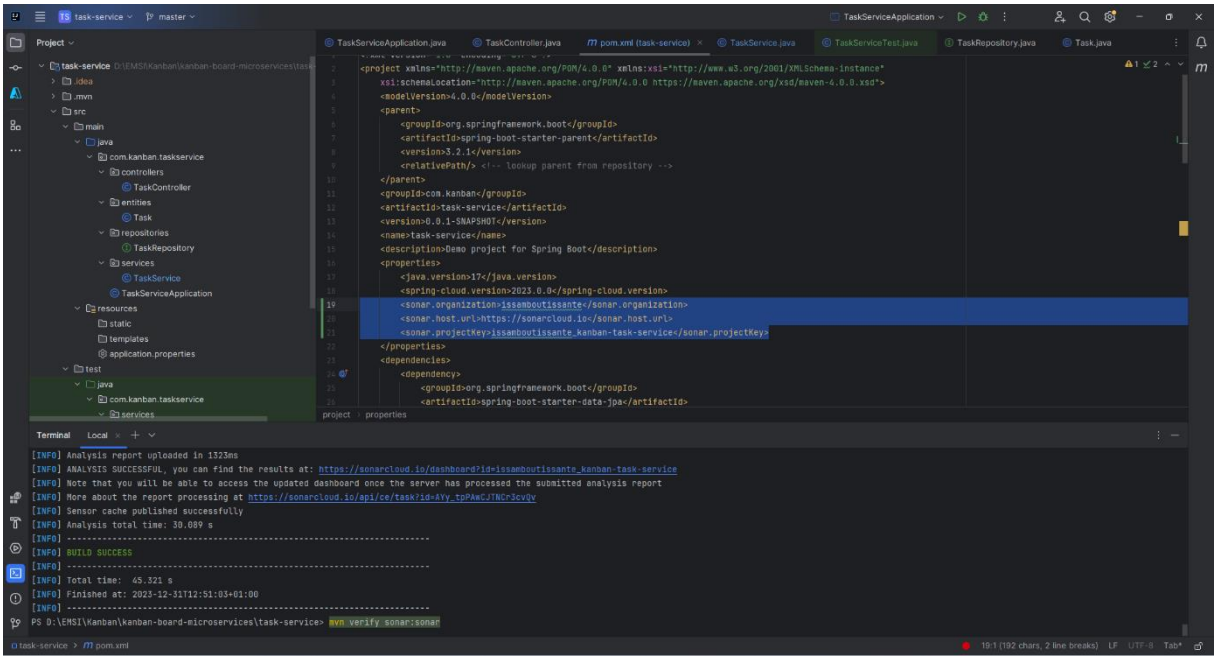
2ème étape :

Puis, nous avons ajouté les propriétés pour chaque service. Ensuite, nous avons exécuté la commande "mvn verify sonar:sonar" pour chaque service.

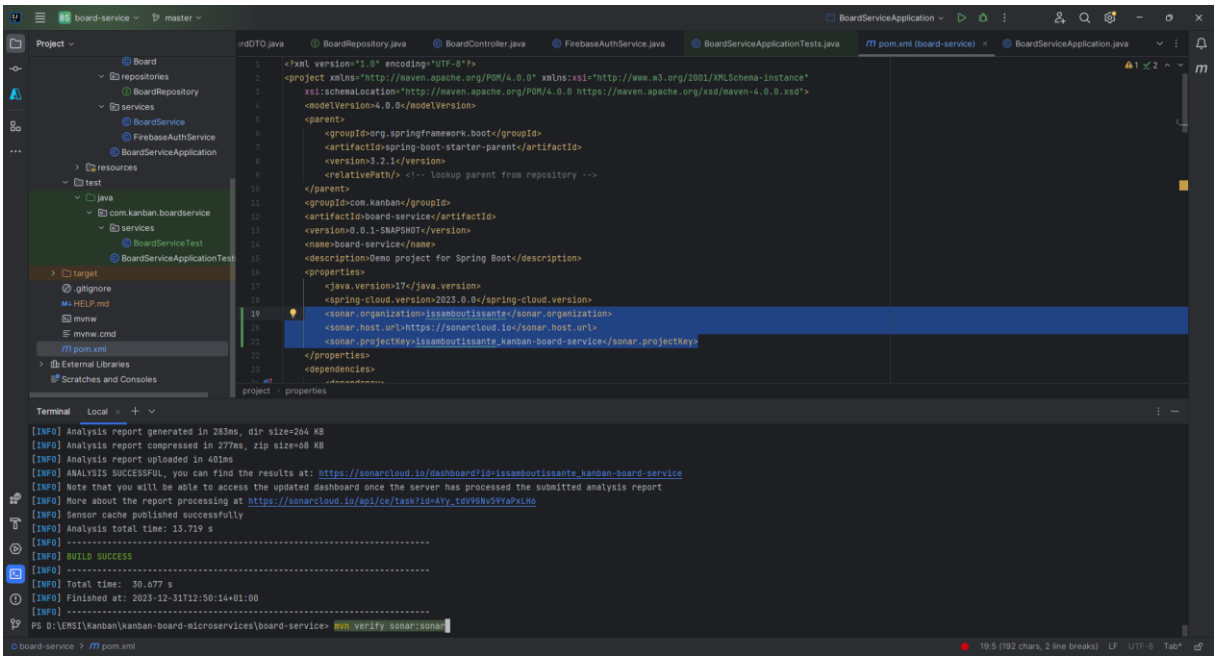
➔ **Pour discovery-server**



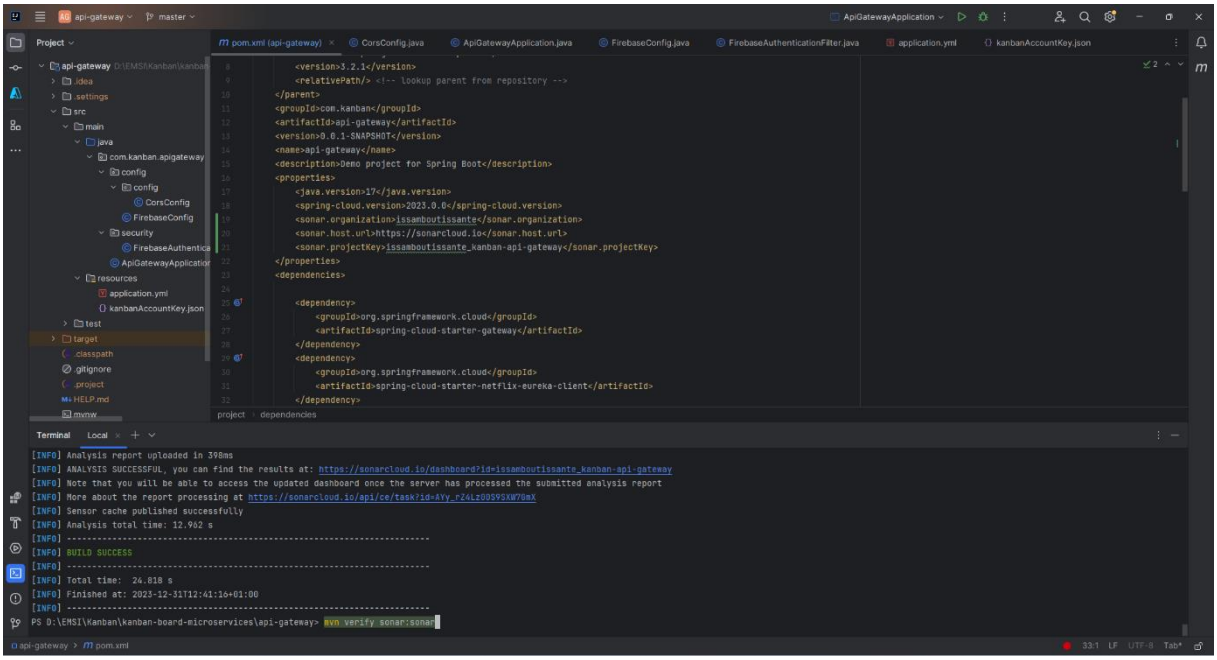
➔ **Pour Task-service**



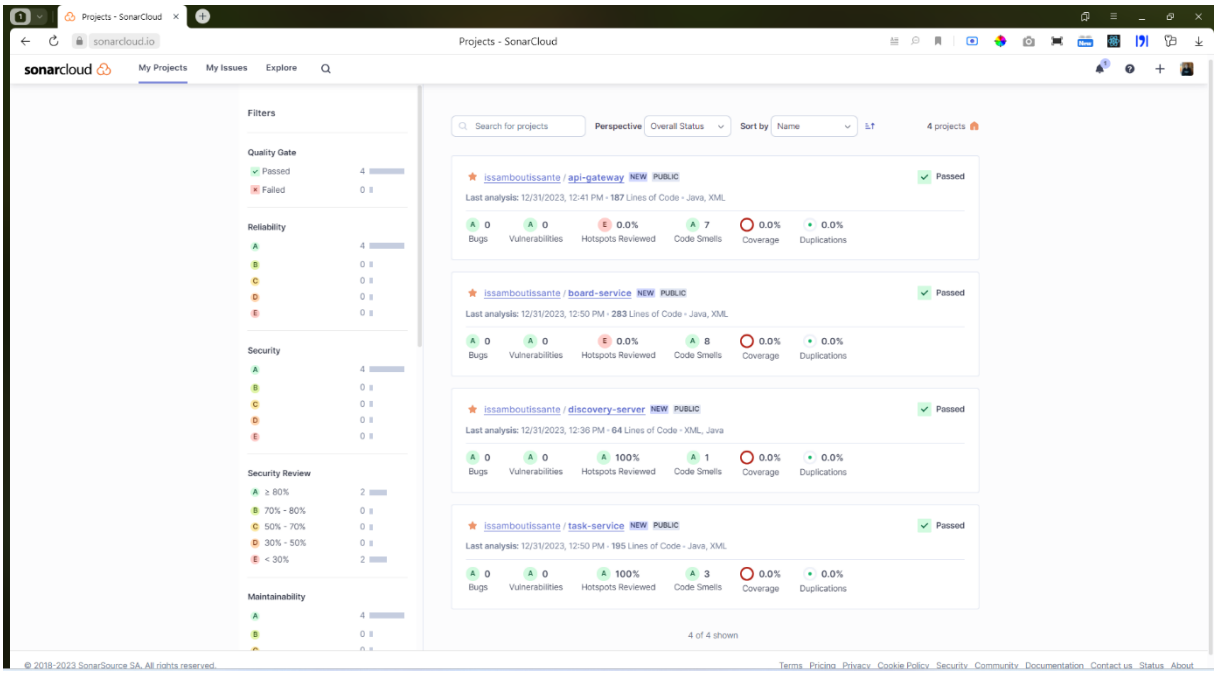
➔ **Pour Board-service**



➔ **Pour Api-gateway**



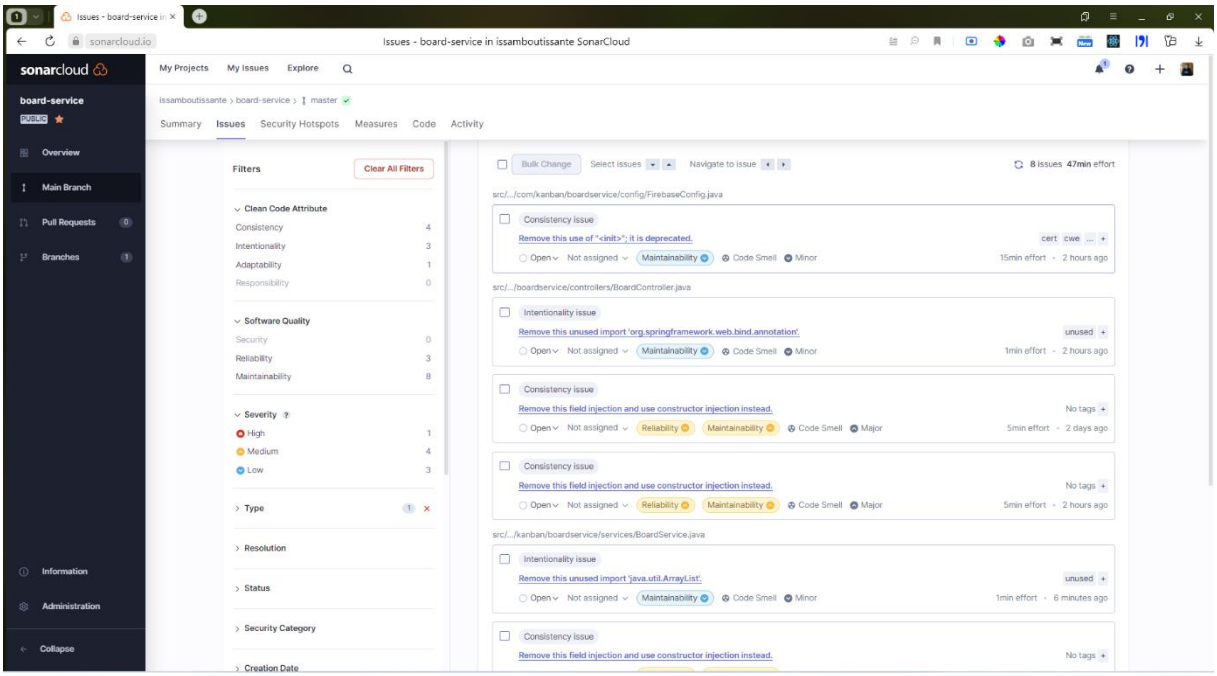
Nos projets dans SonarCloud



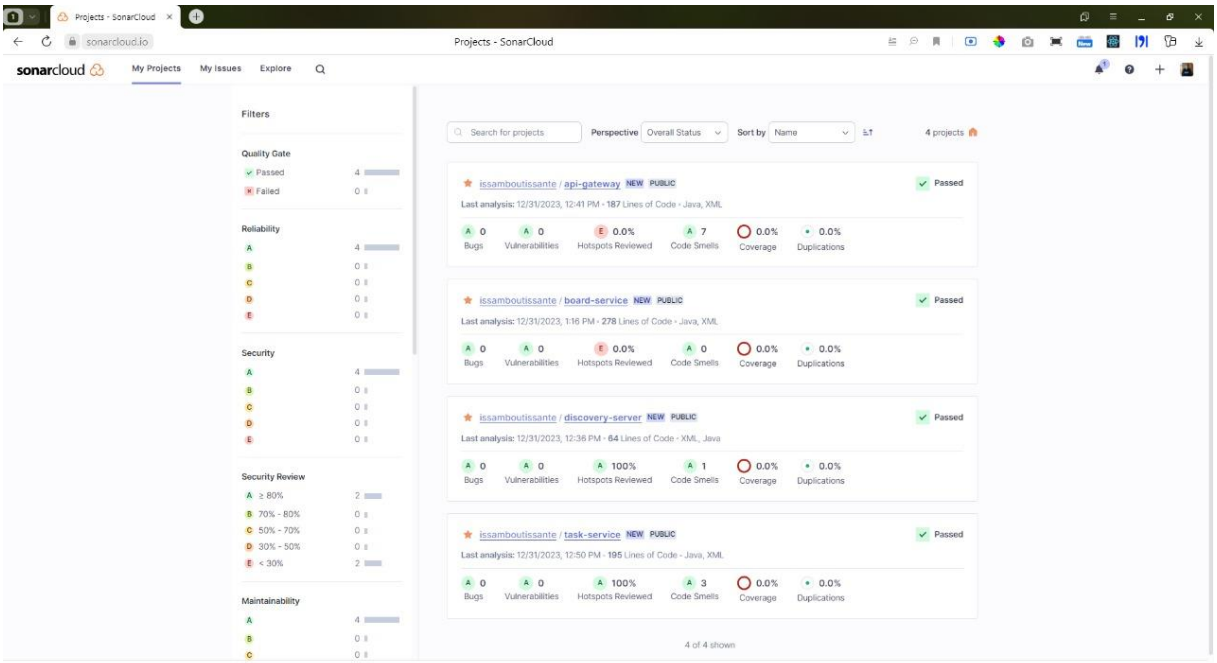
3ème étape

Nous avons corrigé les code smells dans le service board

➔ **Avant correction**



➔ **Après correction**



8. Conclusion et Perspectives

Le développement de l'application Web Kanban à architecture microservice a été couronné de succès, offrant aux utilisateurs un outil de gestion de projet visuel et convivial. L'ensemble des fonctionnalités mises en œuvre, telles que l'authentification sécurisée, la création de tableaux Kanban personnalisés et la gestion des tâches, répondent de manière efficace aux objectifs du projet.

La décision d'adopter une architecture microservice s'est avérée judicieuse, permettant une modularité accrue, une évolutivité facilitée et une maintenance plus efficace de l'application. La collaboration harmonieuse de l'équipe composée de BOUTISSANTE Issam et MAZZOUZ Chaimae a été un élément clé de ce succès, démontrant la synergie et l'efficacité d'une équipe bien coordonnée.

En regardant vers l'avenir, il est possible d'explorer des améliorations continues et des évolutions fonctionnelles pour enrichir davantage l'expérience des utilisateurs. Les retours des utilisateurs peuvent être pris en compte pour ajuster et améliorer les fonctionnalités existantes. De plus, des fonctionnalités supplémentaires, telles que l'intégration avec d'autres outils de gestion de projet ou l'ajout de fonctionnalités avancées, pourraient être envisagées pour répondre aux besoins émergents.

En conclusion, l'application Web Kanban à architecture microservice constitue une réalisation significative, et son succès ouvre la voie à des perspectives passionnantes pour l'amélioration continue et l'expansion des fonctionnalités, contribuant ainsi à l'évolution constante de cet outil de gestion de projet novateur.