**Meeting Room Management**

**Projet de Réservation de Salles de Réunion - Architecture Microservices**

**1. Introduction**

Ce projet a pour objectif de développer une application de réservation de salles de réunion en utilisant une architecture microservices. L'architecture choisie permet une meilleure scalabilité, une maintenance simplifiée et une résilience accrue, éléments cruciaux pour les entreprises modernes. En encapsulant chaque partie fonctionnelle dans des services distincts, nous favorisons une approche modulaire du développement.

**Getting Started**

Follow these steps to set up and run the microservices locally:

1. **Clone the repository:**

git clone https://github.com/your-username/Meeting-room-reservation.git

cd Meeting-room-reservation

1. **Build the project using Maven:**

mvn clean install

1. **Run the Docker containers using Docker Compose:**

docker-compose up

**2. Architecture Microservices**

Le système est divisé en cinq microservices principaux, chacun s'exécutant dans son propre conteneur Docker. La communication entre les services s'effectue à l'aide du service de découverte Eureka et via le gateway, qui agit comme un point d'entrée unique pour les requêtes entrantes.

**2.1 Architecture**

L'architecture est basée sur des services indépendants communiquant via des appels de procédures à distance (RPC) ou des messages HTTP REST. Chaque service est responsable d'une partie distincte de la logique métier et peut être déployé, mis à jour, mis à l'échelle et redémarré indépendamment des autres services.

**2.2 Description des services**

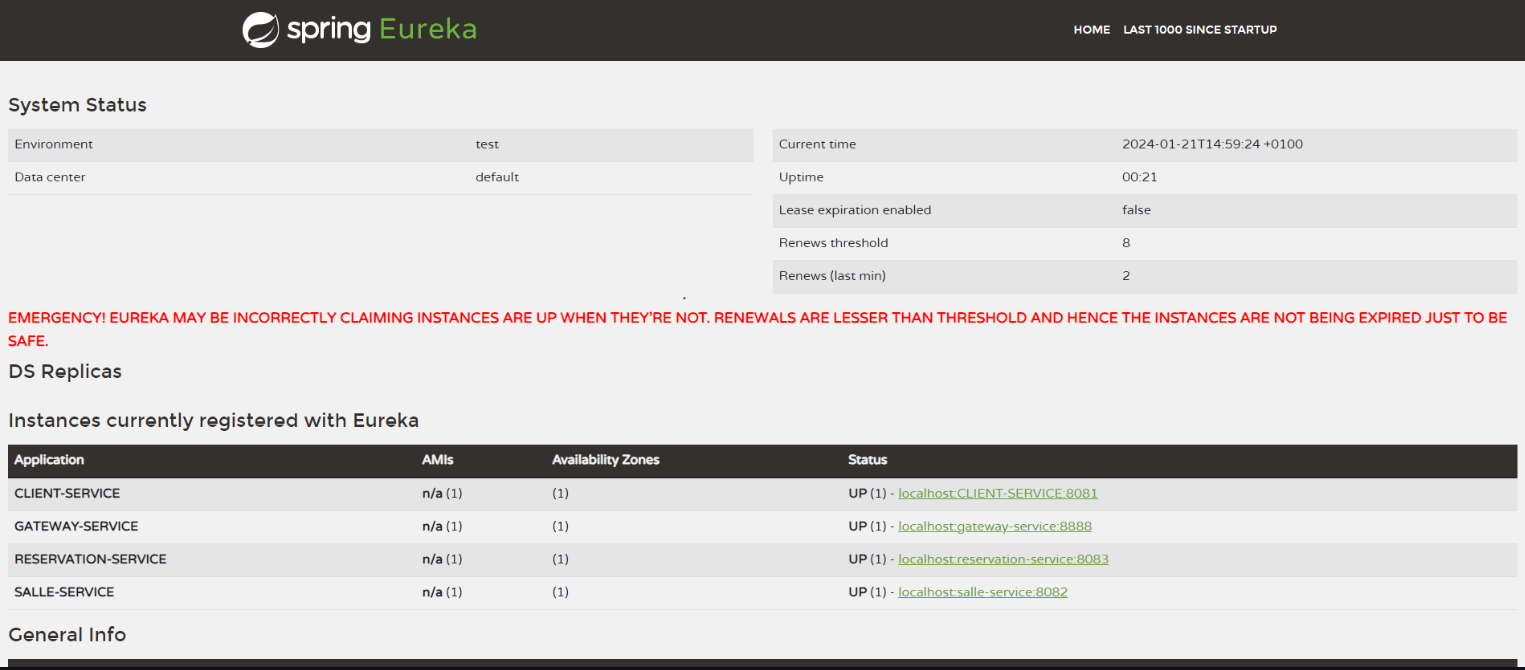
* **eureka-discovery-service** : Permet la découverte des services et leur enregistrement pour une localisation et une communication aisées entre les services.
* **gateway-service** : Fournit un point d'entrée unifié pour les requêtes externes, en les acheminant vers le service approprié.
* **service-salle** : Ce service est au cœur de la gestion des salles. Il stocke les informations relatives aux salles de réunion et permet leur gestion via des opérations CRUD standard. Il gère les types de salles et fournit des détails tels que l'équipement disponible la climatisation et la capacité.
* **client-service** : Ce service gère les données des utilisateurs. Il permet de créer et de gérer les informations de profil des utilisateurs qui réservent les salles.
* **reservation-service** : Ce service est responsable de la prise en charge des réservations des salles par les utilisateurs. Il enregistre les créneaux horaires réservés sans gérer la disponibilité en temps réel.

**2.3 Mécanismes de communication**

La communication entre les services s'effectue via REST pour les requêtes synchrones, et peut être étendue pour utiliser des systèmes de messagerie asynchrones pour des cas d'usage plus complexes.

**3. Conception des Microservices**

Les microservices communiquent les uns avec les autres en utilisant la découverte de services Eureka. Ci-dessous, une capture d'écran du tableau de bord Eureka montrant les services enregistrés :



 Chaque microservice est conçu suivant les principes SOLID pour assurer une forte cohésion et un faible couplage. Le design pattern de chaque service est choisi pour répondre au mieux à sa responsabilité unique au sein du système.

**4. Conteneurisation avec Docker**

La conteneurisation est réalisée à l'aide de Docker, qui encapsule chaque microservice dans son propre conteneur, permettant une portabilité et une cohérence de l'environnement à travers les différentes étapes du développement.

**4.1 Implémentation et avantages**

Chaque service contient un Dockerfile qui décrit les étapes pour créer une image Docker, qui peut ensuite être exécutée de manière isolée. Les avantages incluent la facilité de déploiement, la gestion des dépendances et la scalabilité.

A screenshot of a computer

Description automatically generated

* **client-service**: [Docker Hub Link](https://hub.docker.com/r/your-username/client-service)
* **eureka-discovery-service**: [Docker Hub Link](https://hub.docker.com/r/your-username/eureka-discovery-service)
* **gateway-service**: [Docker Hub Link](https://hub.docker.com/r/your-username/gateway-service)
* **reservation-service**: [Docker Hub Link](https://hub.docker.com/r/your-username/reservation-service)
* **salle-service**: [Docker Hub Link](https://hub.docker.com/r/your-username/salle-service)

**5. CI/CD avec Jenkins**

Jenkins est utilisé pour automatiser l'intégration continue (CI) et le déploiement continu (CD), en définissant des pipelines qui testent, construisent et déploient automatiquement les microservices à chaque commit.

**5.1 Processus et configuration**

Des Jenkinsfile sont présents dans chaque microservice pour décrire les étapes du pipeline, de la construction des images Docker à leur déploiement sur le serveur de production.

**7. Intégration de SonarQube**

SonarQube est intégré pour réaliser l'analyse statique du code, ce qui permet d'améliorer la qualité du code en détectant les vulnérabilités et les codes smells.

**7.1 Configuration et bénéfices pour la qualité du code**

La configuration de SonarQube est définie dans les fichiers sonar-project.properties de chaque service, permettant un suivi et une amélioration continus de la qualité du code.

**8. Conclusion**

Le projet de réservation de salles de réunion illustre l'efficacité de l'architecture microservices dans un contexte d'entreprise. Nous envisageons d'ajouter des fonctionnalités d'intelligence artificielle pour des recommandations de salles et d'optimiser davantage les performances à l'aide de caches distribués.

Pour démarrer avec le projet, veuillez suivre les instructions de mise en place et d'exécution fournies dans le README de chaque microservice.

Nous vous souhaitons un codage productif et satisfaisant ! 🚀