Samba Diarra DIOUF
Mohamed ALASSAF

Introduction

Dans le cadre de la conception d'une application de prêt de livres, nous avons décidé de mettre en place les micro-services suivantes :

Service de gestion des lecteurs (Reader) : ce micro-service serait responsable de la gestion des informations relatives aux lecteurs. Il permet de créer/afficher/modifier/supprimer un lecteur.

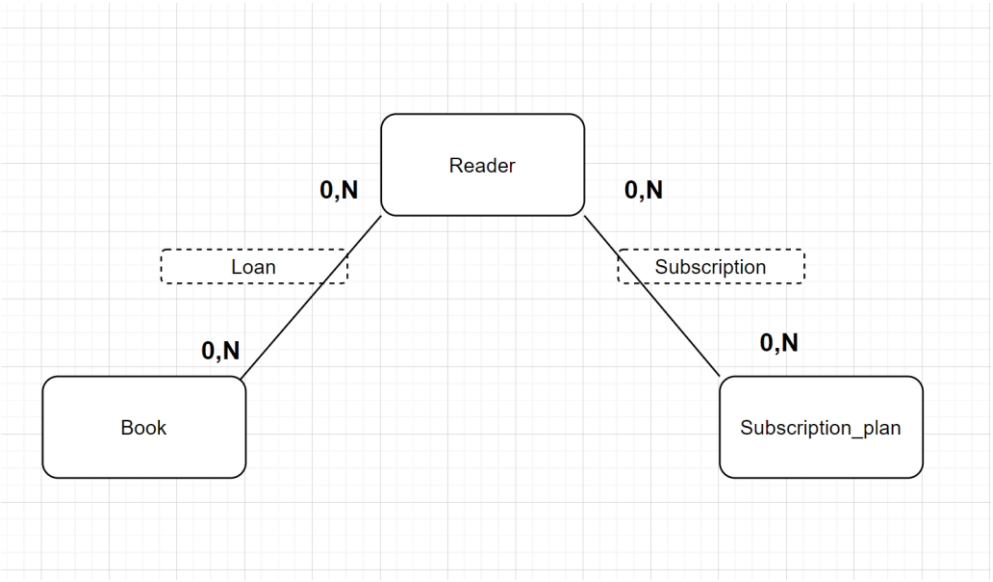
Service de gestion des livres (Book) : ce micro-service serait responsable de la gestion des informations relatives aux livres, telles que les informations de titre, d'auteur, de disponibilité, etc.

Service de gestion des plans d'abonnement (SubscriptionPlan) : ce micro-service serait responsable de la gestion des différents plans d'abonnement proposés, tels que son nom, les quotas de prêt et les périodes d'abonnement (durée de l'abonnement en mois).

Service de gestion des abonnements (Subscription) : ce micro-service serait responsable de la gestion des abonnements des lecteurs. C'est ce service qui va gérer le fait qu'un lecteur puisse s'abonner ou se désabonner à un plan d'abonnement.

Service de gestion des emprunts : ce micro-service serait responsable de la gestion des emprunts, y compris la création d'un emprunt, la vérification de la disponibilité des livres, la vérification que le lecteur a un abonnement valide et qu'il n'a pas atteint son quota d'emprunt, la gestion des dates d'emprunt et de retour, etc.

User Story :
Dans l'application on peut créer des plans d'abonnement et le lecteur devra s'abonner pour pouvoir faire un emprunt de livre. Un plan auquel le lecteur s'est abonné définie le nombre de livre qu'un lecteur peut emprunter et la durée de son abonnement.
Ainsi lorsqu'un lecteur veut faire un près on vérifie d'abord si le livre qu'il veut emprunter est disponible. Si oui, on vérifie s'il a un abonnement valide, c'est-à-dire qu'il s'est abonné à un plan et que cet abonnement est toujours en cours de validité. S'il a un abonnement on vérifie s'il n'a pas atteint le quota d'emprunt par semaine. Et si tout est bon, l'emprunt se fait et le statut du livre est mise à jour.



Samba Diarra DIOUF
Mohamed ALASSAF

Base de données

Dans notre application chaque micro-services à sa propre base de données.
Nous avons décidé de faire ainsi pour les raisons suivantes :

Isolation : En utilisant des bases de données distinctes pour chaque micro-service, vous assurez l'isolation des données. Cela signifie que si un micro-service est compromis ou qu'une erreur se produit, cela n'affectera pas les autres micro-services ou les données qu'ils gèrent.

Scalabilité : Les bases de données des micro-services sont souvent conçues pour être scalables de manière indépendante. Cela signifie que si un micro-service nécessite plus de ressources de base de données, vous pouvez simplement ajouter des ressources à la base de données de ce micro-service sans affecter les autres micro-services.

Autonomie : En permettant à chaque micro-service d'avoir sa propre base de données, cela permet à chaque équipe de développement de travailler de manière autonome et indépendante, sans affecter les autres équipes.

Documentation technique :

Choix techniques

Dans notre application nous avons utilisé une base de données Postgres. Assurer vous donc d'avoir installé Postgres sur votre ordinateur.

Dans notre application nous avons mis en place un Gateway afin d'agréger les appels aux différents services. Au lieu que chaque client fasse des appels directs à chaque service, le gateway va regrouper les appels nécessaires pour une seule requête, ce qui réduit le nombre d'appels nécessaires et peut améliorer les performances.
Afin de mettre en place le Gateway, nous avons utilisé Eureka.
Eureka est un service de découverte de micro-services qui permet de localiser dynamiquement les instances de services dans un environnement de micro-services.
On doit s'assurer que le gateway peut découvrir les instances des différents services qu'il doit contacter. Eureka sert donc à fournir cette fonctionnalité de découverte de services pour le gateway.
Dans notre application nous avons donc créer un projet server-eureka qui est donc un service de découverte locale.

Nous avons mis en place aussi en place un système de cache (Redis) au niveau du gateway.
Ce système de cache permet de stocker temporairement les données fréquemment accédées, ce qui peut aider à réduire le temps de réponse et à améliorer les performances de l'application.

Nous avons utilisé docker pour notre application aussi. Nous avons créé un fichier Dockerfile au sein du répertoire racine de chaque projet(service). Ensuite nous avons créé un fichier docker-compose afin de mettre ces services sur le même réseau et de permettre la communication de ces services.

Nous avons essayé aussi de déployer notre application sur Minikube. Mais malheureusement nous n'avons pas comprendre comment configurer les fichiers service.yml et deployment.yml de chaque

service afin de pouvoir déployer sur Minikube.

Exécuter le projet :

Faire un git clone de notre application.
Pour lancer notre application en locale il faut créer les bases de données suivantes dans votre pgAdmin de postgreSQL.
- projet_microservice

- projet_microservice_reader

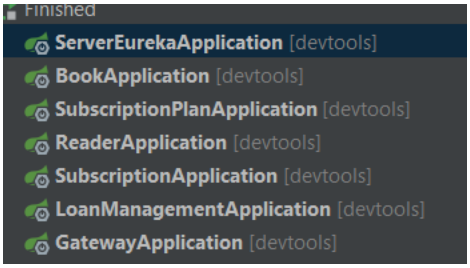
- projet_microservice_plan

- projet_microservice_loan

- projet_microservice_subscription

Pensez à modifier les application.yml de chaque projet pour remplacer les identifiants de connexion à la base de données par les votre.

Si vous voulez lancer notre application sans docker, vous devez lancer les applications dans l'ordre suivante :



Pour utiliser docker. Il faut au préalable installer docker sur votre machine. Nous avons publié l'image de nos services sur le docker hub. Vous devez donc les récupérer avant de lancer le docker-compose de notre application.

Veuillez exécuter les commandes suivantes :
docker login (pour vous connecter à votre compte docker hub. Si vous n'en avez pas il faut en créer un)

docker pull samba914/eureka-image:latest

docker pull samba914/book-image:latest

docker pull samba914/reader-image:latest

docker pull samba914/plan-image:latest

docker pull samba914/subscription-image:latest

docker pull samba914/loan-image:latest

docker pull samba914/gateway-image:latest

Samba Diarra DIOUF
Mohamed ALASSAF

Ainsi toutes les images récupérées, vous pouvez lancer au niveau de notre répertoire racine(la ou se trouver le docker-compose.yml) la commande :
docker-compose up

PS : Veuillez vérifier que vous port 6379 et 5432 sont bien libre car docker les utilisent.

Veuillez attendre quelques minutes afin que toutes les applications spring boot se lancent et s'enregistre au serveur eureka.

Après le lancement de toutes les applications, vous pouvez donc maintenant faire des tests.
Nous vous avons mis dans notre répertoire racine un fichier collection postman « Librairie.postman_collection » à importer dans votre postman. Ainsi vous avez un ensemble de requêtes à lancer pour tester notre application.
Il faut lancer les requêtes dans l'ordre (du haut vers le bas) afin que les requêtes soient cohérentes.

Bilan :

Nous avons apprécié la liberté de conception qu'offre le développement de microservices, ainsi que la possibilité de travailler sur plusieurs parties du système en même temps. Nous avons également apprécié l'utilisation de technologies modernes telles que Spring Boot, Docker, Redis.

Nous avons appris beaucoup de choses au cours de ce projet. Tout d'abord, comment concevoir et développer des microservices, faire dialoguer plusieurs applications entre eux. Mais aussi travailler avec des outils tels que Git, Maven, Docker.

L'un des plus grands succès du projet a été la mise en place d'une architecture de microservices efficace qui a permis une plus grande flexibilité et une meilleure évolutivité de l'application.

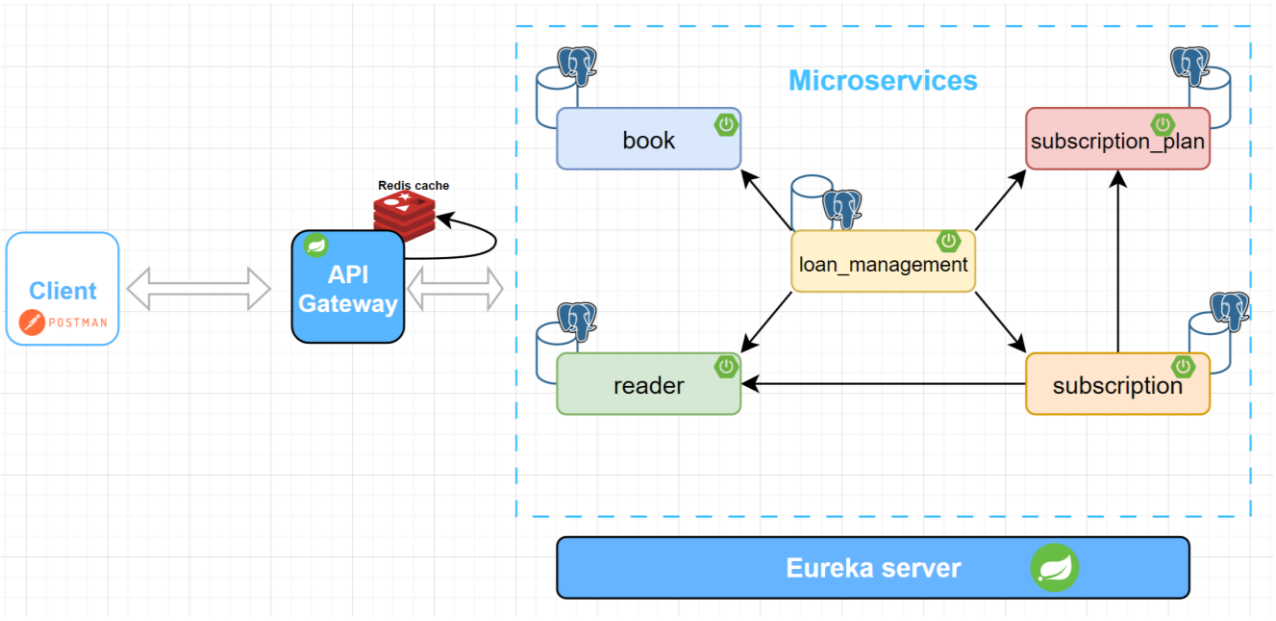
Il y a eu des moments de frustration lors des tests qui fonctionnait une fois sur deux. On a rencontré également quelques difficultés à comprendre le fonctionnement de Docker et la logique derrière. Nous n'avons pas eu de problème à concevoir l'application en tant que telle mais il a eu quelques difficultés lors de la mise en place du cache notamment l'utilisation des annotations (tel que : @Cacheable, @CacheEvict, @CachePut).

Pour les améliorations, nous pouvons intégrer Kubernetes pour la gestion des Container. Également, des caches supplémentaires sur certains microservices ainsi que l'utilisation de technologies comme Kafka pour mettre à jour les caches lors de la modification des données. Enfin, des applications de monitoring auront été les bienvenues pour avoir une vue globale de l'application et enregistrer les logs.

En somme, ce projet de développement d'une application microservice en Java a été une expérience enrichissante, malgré quelques difficultés. Nous avons appris beaucoup de choses et acquis de nouvelles compétences qui me seront utiles dans mes projets futurs.

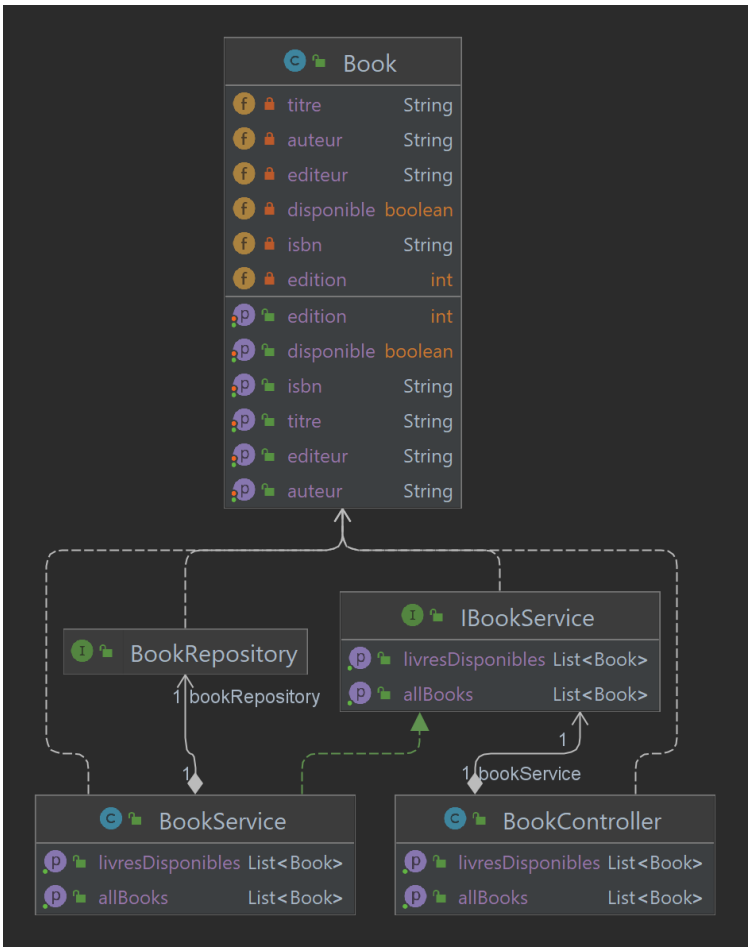
Samba Diarra DIOUF
Mohamed ALASSAF

Architecture du projet :



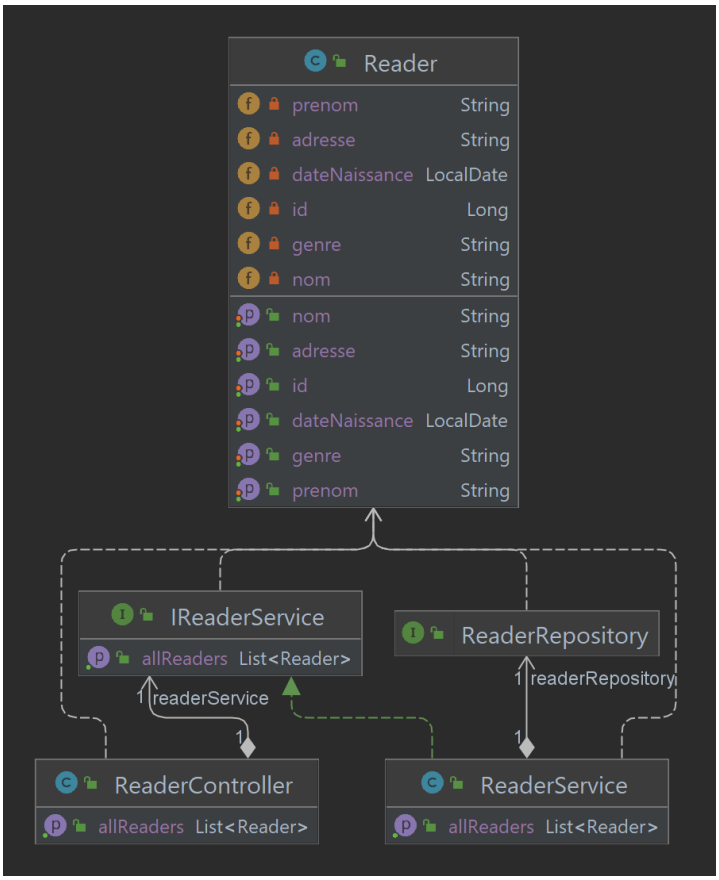
Digrammes de classes :

Book **:**

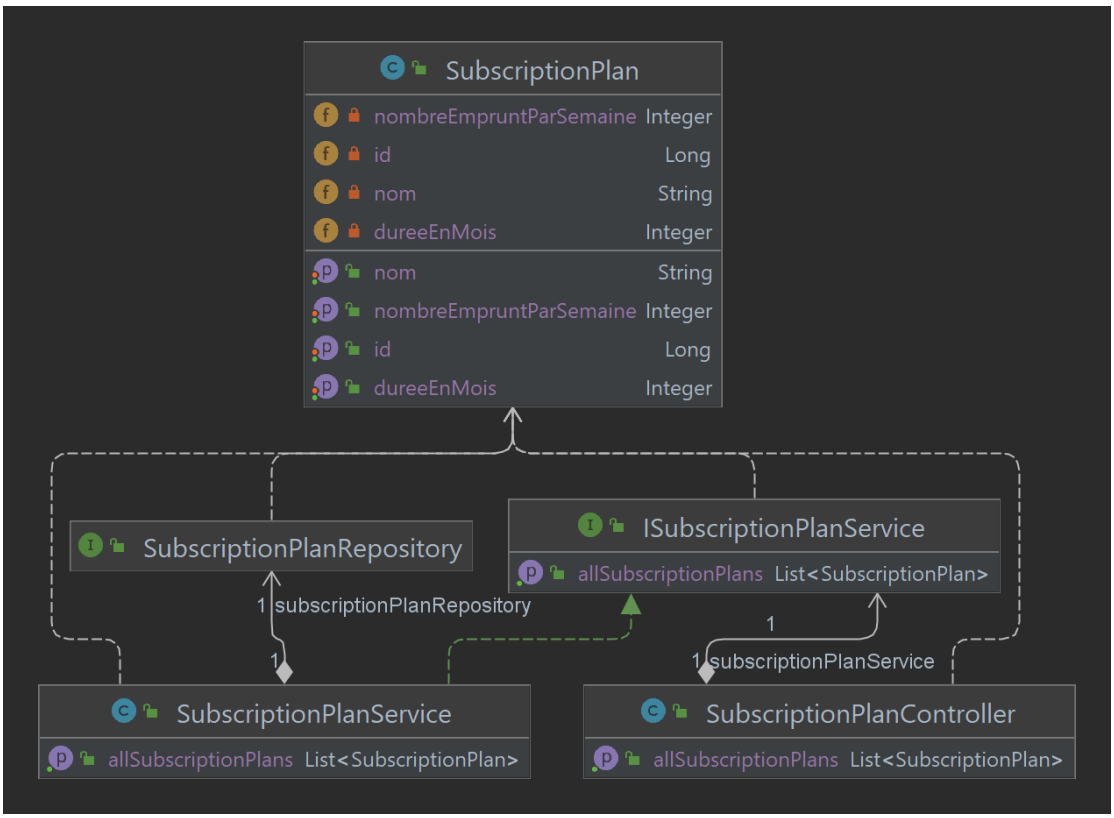


Samba Diarra DIOUF
Mohamed ALASSAF

Reader :

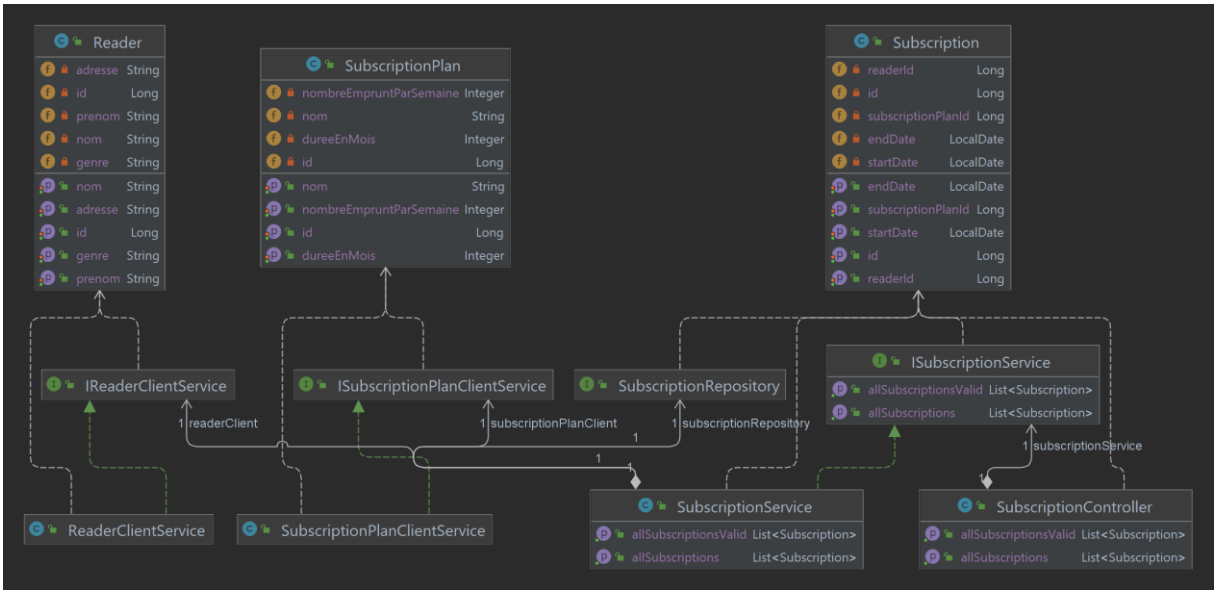


SubscriptionPlan :



Samba Diarra DIOUF
Mohamed ALASSAF

Subscription :



Loan :