

3293.3 JEE/Spring II · 2023-2024 · ISC3il-a

NeoLib - Un système de gestion de bibliothèque

Nima Dekhli

31 mai 2024

Table des matières

1	Introduction	2
1.1	Contexte	2
1.2	Choix du sujet	2
2	Architecture implémentée	2
2.1	book-service	2
2.2	loan-service	3
3	Flux de communications entre les services	3
3.1	Flux de communication synchrones	3
3.2	Flux de communication asynchrones	3
4	Guide de démarrage	4
4.1	Prérequis	4
4.2	Démarrage des services	4
5	Conclusion	4

1 Introduction

1.1 Contexte

Dans le cadre du cours de JEE/Spring II, il nous est demandé d'implémenter une API qui utilise le pattern *microservices*. Le sujet du projet est libre, tant qu'il respecte les contraintes minimales.

1.2 Choix du sujet

NeoLib est un système de gestion des bibliothèques. Il permet de gérer les livres à disposition et les emprunts des utilisateurs.

2 Architecture implémentée

Deux services distincts coexistent et se complètent. Le premier service, *book-service*, permet la gestion des livres, avec toutes les méta-données qui les composent. Le second service, *loan-service*, permet la gestion des emprunts par les utilisateurs.

Chacun des services possède sa propre base de données. Le service *loan-service* est dépendant du service *book-service* pour la gestion des emprunts. En effet, un emprunt ne peut être effectué que si le livre existe et est disponible. Il doit donc effectuer une requête synchrone pour interroger le service *book-service*.

Les communications synchrones sont effectuées en utilisant le protocole HTTP. Les communications asynchrones sont effectuées en utilisant le protocole AMQP. Afin d'effectuer du *load balancing*, une gateway est utilisée pour rediriger les requêtes vers les services. Tous les services s'enregistrent auprès du registre Eureka.

Sur la figure 1, on peut voir l'architecture générale du système.

2.1 book-service

Ce service est une simple base de données de livres. L'API permet de lister les livres, d'en ajouter, d'en supprimer et de les modifier. Les champs suivants sont disponibles pour chaque livre :

- ID (génééré automatiquement)
- ISBN
- Titre
- Auteur

— Statut

Le statut permet de savoir si le livre est disponible, emprunté, perdu ou bloqué.

2.2 loan-service

Ce service permet la gestion des emprunts ainsi que des utilisateurs. Un utilisateur doit exister dans la base de données pour pouvoir emprunter un livre.

L'API permet d'emprunter un livre, de le rendre, de lister les emprunts en cours (pour un utilisateur donné) et de savoir qui a emprunté un livre donné. De plus, il est possible de marquer un livre comme perdu.

3 Flux de communications entre les services

3.1 Flux de communication synchrones

Les flux de communication synchrones sont effectués en utilisant le protocole HTTP. Ils sont uniquement utilisés pour effectuer des requêtes depuis le service *loan-service* vers le service *book-service*.

Il y a deux cas d'utilisation des communications synchrones :

- Lors de l'emprunt d'un livre, le service *loan-service* doit récupérer les informations du livre (méta-données) pour compléter sa base de données locale et pour vérifier que le livre est bien disponible ;
- Lors de la perte d'un livre, le service *loan-service* doit marquer le livre comme perdu dans la base de données du service *book-service*.

Afin d'effectuer du *client-side load balancing*, on utilise un client Feign pour effectuer les requêtes HTTP.

3.2 Flux de communication asynchrones

Les flux de communication asynchrones sont effectués en utilisant le protocole AMQP. Ils sont utilisés pour la communications entres les services, dans les deux sens. Ils permettent de notifier sans attendre de réponse.

Ils sont utilisés dans les cas suivants :

- Lors de l'emprunt d'un livre, le service *loan-service* doit notifier le service *book-service* que le livre a été emprunté, pour la mise à jour du statut du livre ;

- Lors du retour d'un livre, le service *loan-service* doit notifier le service *book-service* que le livre a été rendu, pour la mise à jour du statut du livre ;
- Lorsque les méta-données d'un livre sont modifiées, le service *book-service* doit notifier le service *loan-service* pour la mise à jour des informations dans la base de données du service *loan-service*.

4 Guide de démarrage

Dans cette section, on décrit la procédure qui permet de démarrer correctement les services.

4.1 Prérequis

Il est nécessaire d'avoir installé les logiciels suivants :

- Java 17
- Maven 3.9
- Docker 25
- Docker Compose 2.24

4.2 Démarrage des services

Tout d'abord, lancer le Docker Compose pour démarrer RabbitMQ et les bases de données mySQL :

```
1 docker-compose up
```

Ensuite, lancer le registre Eureka :

```
1 cd registry-service
2 mvn spring-boot:run
3 cd ..
```

5 Conclusion

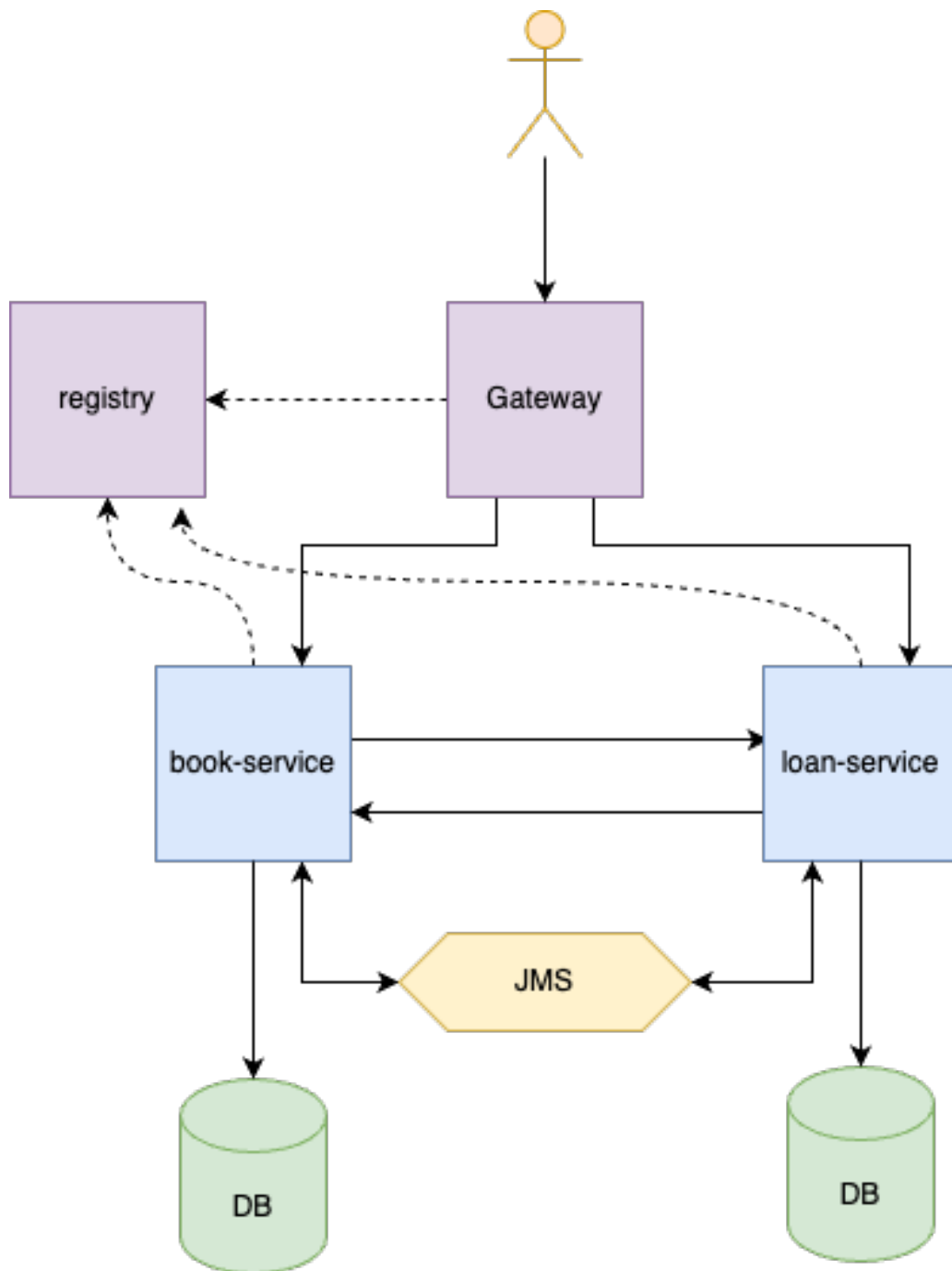


FIGURE 1 – Architecture générale du système