01/02/2025

Mise en œuvre d'une application micro-services de gestion des emprunts d'une librairie

Mettre en place d'une application web microservices d'une application avec son application cliente React.

MAHDI KELLOUCH
ISMO TETOUAN

Mini-Projet : Système de Gestion des Emprunts d'une Librairie en Microservices

Énoncé du Mini-Projet

Dans le but d'optimiser la gestion des emprunts de livres, une librairie souhaite mettre en place une **architecture microservices**. Ce système devra permettre une gestion indépendante et efficace des différentes fonctionnalités tout en assurant une communication fluide entre les services.

Chaque microservice doit être :

- **Indépendant** et autonome dans son fonctionnement.
- **Bien défini** avec des responsabilités claires.
- **Connecté** aux autres services via un canal de communication efficace.

Fonctionnalités du système

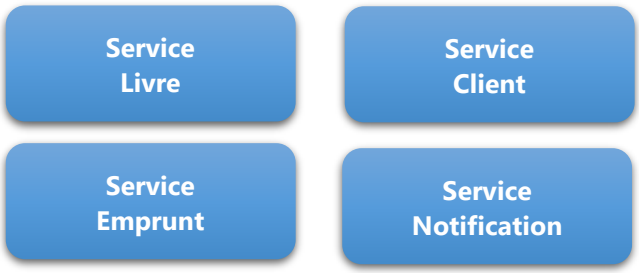
Le système offrira les fonctionnalités suivantes :

- **Gestion des livres** (stockage, ajout, suppression, modification).
- **Gestion des emprunts** (suivi des emprunts et retours de livres).
- **Gestion des clients** (enregistrement, modification et suppression).
- **Service de notifications** (alerte aux clients sur la disponibilité des livres).
- **Service de paiement** (gestion des frais d'emprunt si nécessaire).

Architecture du projet

Le projet est basé sur **quatre microservices principaux**, chacun ayant une responsabilité bien définie :

1. **Service Livre** 📚
    - Gère l'inventaire des livres.
    - Expose une API REST pour récupérer, ajouter, modifier et supprimer des livres.
2. **Service Client** 👤
    - Gère les informations des clients de la librairie.
    - Expose une API REST pour gérer les données clients.
3. **Service Emprunt** 🔄

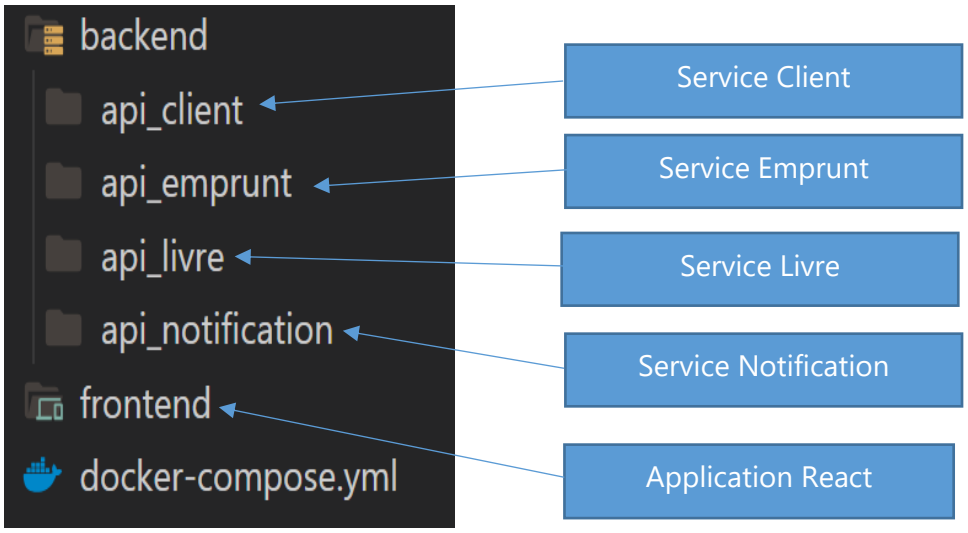


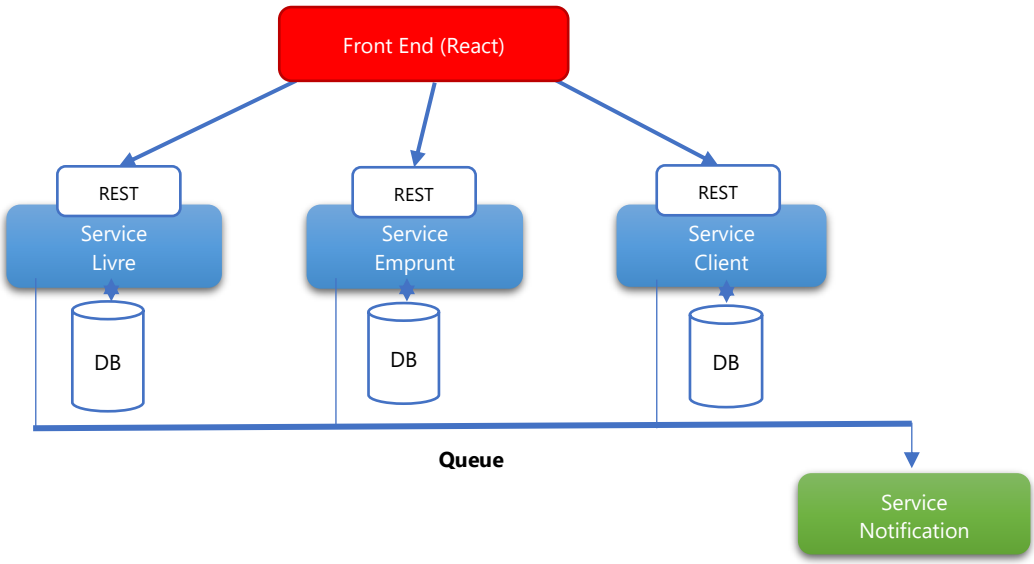
- Gère les emprunts et retours de livres.
- Communique avec les services Client et Livre pour valider les emprunts.

4. **Service Notification** 📢
   - Notifie les clients lorsque des livres sont disponibles ou en cas de mises à jour.
   - Utilise une communication **asynchrone** via un système de messagerie (ex. RabbitMQ).

Structure du projet

L'application est organisée comme suit :





Travail à réaliser

✓ **Service Livre**

- Gère le stock de la librairie.

- Dispose d'une base de données contenant : code, titre, description, auteur.
- **Exposition d'une API REST :**

Fonctionnalité	Méthode	URL	Code retour
Obtenir les détails d'un livre	GET	/api/v1/livre/{idLivre}	200 OK / 404 Not Found
Ajouter un nouveau livre	POST	/api/v1/livre	200 OK
Modifier un livre	PUT	/api/v1/livre/{idLivre}	200 OK / 404 Not Found
Supprimer un livre	DELETE	/api/v1/livre/{idLivre}	200 OK / 404 Not Found

✓ **Service Client**

- Gère les clients de la librairie.
- Dispose d'une base de données avec nom, prénom, email.
- **Exposition d'une API REST :**

Fonctionnalité	Méthode	URL	Code retour
Obtenir les détails d'un client	GET	/api/v1/client/{idClient}	200 OK / 404 Not Found
Ajouter un nouveau client	POST	/api/v1/client	200 OK
Modifier un client	PUT	/api/v1/client/{idClient}	200 OK / 404 Not Found
Supprimer un client	DELETE	/api/v1/client/{idClient}	200 OK / 404 Not Found

✓ **Service Emprunt**

- Gère les emprunts des livres.
- Dispose d'une base de données contenant livre, client, date d'emprunt, date de retour.
- Fonctionnement :
    1. Un client fournit le **code du livre** et son **nom** pour un emprunt.
    2. Le service récupère les détails du client et du livre via leurs APIs respectives.
    3. La date d'emprunt est enregistrée automatiquement.
    4. La date de retour est initialement null et mise à jour au moment du retour.
- **Exposition d'une API REST :**

Fonctionnalité	Méthode	URL	Code retour

Ajouter un emprunt	POST	/api/v1/emprunt	200 OK
Retourner un livre	POST	/api/v1/emprunt/retour	200 OK
Lister les emprunts d'un client	GET	/api/v1/emprunt/{idClient}	200 OK / 404 Not Found

✓ **Service Notification**

- Gère l'envoi d'alertes aux clients de la librairie.
- **Asynchrone** (pas de réponse immédiate attendue).
- Fonctionnement basé sur un **système de messagerie (ex. RabbitMQ)** pour diffuser les événements.
- Utilisation du package **nodemailer** pour l'envoi d'e-mails.

✓ **Front-End (React)**

- Développer une **interface utilisateur intuitive** pour consommer les services backend.
- Afficher la **liste des livres**, gérer les **emprunts**, consulter les **notifications**.
- Utiliser **Axios** pour interagir avec les APIs REST.

✓ **Dockerisation (Docker-Compose)**

- Conteneuriser chaque microservice.
- Configurer un docker-compose.yml pour orchestrer les conteneurs.