Rapport Cocktail-Companion

1 Introduction

1.1 Description du projet

Le projet Cocktail Companion consiste en la création d'une application web back-end distribuée permettant la consultation de cocktails et la gestion de feedbacks (notes/commentaires) associés.

L'architecture repose sur des microservices en Spring Boot, exposant des API REST communiquant entre eux via des appels synchrones et asynchrones. Le système intègre également une API externe pour enrichir les informations des cocktails.

1.2 Fonctionnalités

Rechercher un cocktail

- L'utilisateur peut chercher un cocktail par nom (ex. : Margarita)
- Ou simplement en obtenir un au hasard

Consulter la recette

- Une fois un cocktail trouvé, l'utilisateur voit son nom, son image, ses ingrédients, ses quantités et les instructions

Laisser un avis

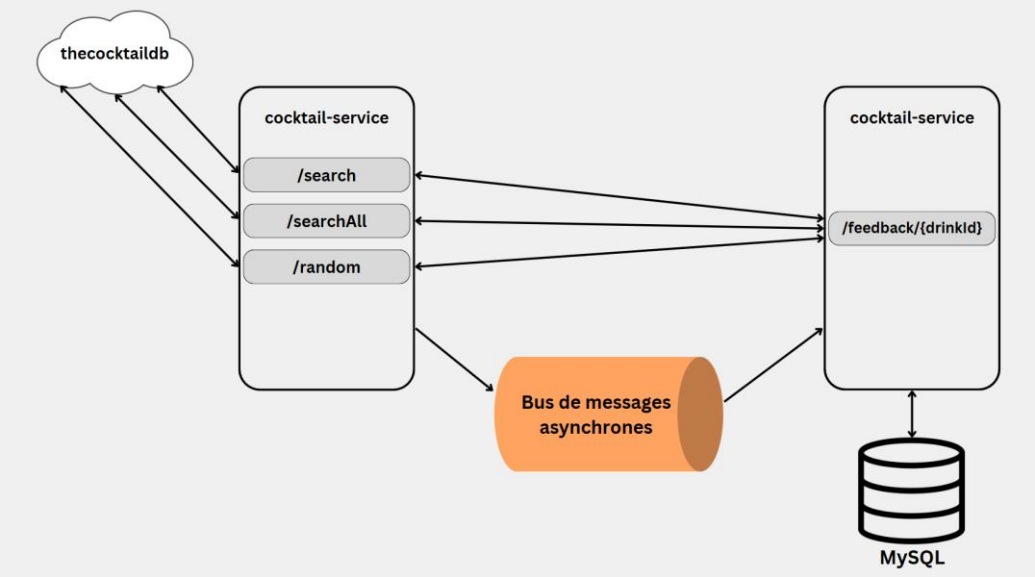
- L'utilisateur peut noter le cocktail entre 1 et 5
- Il peut aussi laisser un commentaire

Voir les avis

- Lorsqu'un utilisateur consulte un cocktail, il peut voir :
    - La moyenne des notes
    - Les commentaires et notes déposés par d'autres utilisateurs

2 Conception

2.1 Schéma général



2.2 Architecture microservices

L'application Cocktail Companion repose sur une architecture microservices composée de deux services principaux :

- ***cocktail-service*** : ce service est responsable de la recherche de cocktails, de la consultation des recettes via une API externe (TheCocktailDB), et de la gestion des feedbacks utilisateurs (envoi).
- ***feedback-service*** : ce service est dédié à la gestion des avis (feedbacks). Il les reçoit de manière asynchrone, les stocke dans une base de données MySQL, et expose une API REST pour les consulter.

Cette séparation permet une meilleure scalabilité, maintenance indépendante, et un découplage des responsabilités métier.

2.3 Communications entre services

Le système combine **deux types de communication** entre les microservices, illustrés dans le schéma ci-dessous :

Communication synchrone (REST)

Le *cocktail-service* fait des appels HTTP directs vers le *feedback-service* pour :

- récupérer les feedbacks liés à un cocktail
- calculer la note moyenne

Cela se fait via la route exposée : GET /feedback/{drinkId}

Ce type de communication est bloquant et utilisé lorsque l'on a besoin d'une réponse immédiate (ex : affichage de la fiche cocktail).

Communication asynchrone (JMS)

Lorsque l'utilisateur soumet un avis sur un cocktail, le *cocktail-service* :

1. construit un message contenant l'idDrink, la note et le commentaire
2. envoie ce message sur un bus de messages (ActiveMQ)
3. le feedback-service écoute ce bus et consomme les messages reçus
4. les feedbacks sont alors persistés en base MySQL

Ce mécanisme permet de désengorger le service principal, de ne pas bloquer la réponse à l'utilisateur, et d'assurer une meilleure résilience en cas de surcharge.

2.4 Base de données

Le *feedback-service* utilise une base MySQL pour stocker durablement tous les feedbacks associés aux cocktails.

Chaque entrée contient l'identifiant du cocktail, une note (entier), et un commentaire utilisateur.

2.5 API externe

Le service *cocktail-service* consomme l'API publique TheCocktailDB pour récupérer :

- la liste des cocktails (recherche par nom)
- les détails d'un cocktail aléatoire
- les ingrédients, instructions et visuels

Ces appels permettent de proposer un contenu riche et réaliste à l'utilisateur sans maintenir soi-même une base de cocktails.

3 Implémentation

3.1 Architecture logicielle

L'architecture interne des deux services suit un découpage classique de projet Spring Boot, en couches controller, service, model, repository, et, dans le cas du *feedback-service*, un listener JMS.

cocktail-service

- **Controller**
  - *CocktailController* : expose les routes REST /search, /searchAll, /random pour interroger l'API externe, et /feedback (POST) pour publier un avis utilisateur.
- **Service**
  - *CocktailService* : interroge l'API externe TheCocktailDB.
  - *CocktailFeedbackService* : communique avec le feedback-service pour enrichir les cocktails avec les feedbacks.
  - *JmsSender* : envoie les messages de feedback dans une file JMS.
- **Model (package resource)**
  - *Drink* : modèle représentant un cocktail tel que retourné par TheCocktailDB.
  - *JmsMessage* : structure utilisée pour transporter un feedback en message JMS.

feedback-service

- **Controller**
  - *FeedbackController* : expose l'endpoint /feedback/{drinkId} qui retourne la liste des avis et la moyenne des notes pour un cocktail donné.
- **Service**
  - *FeedbackService* : contient la logique métier pour récupérer et traiter les feedbacks.
- **Model**
  - *Feedback* : entité JPA stockée en base (cocktailId, note, commentaire).
  - *JmsMessage* : modèle de réception d'un message JMS, identique à celui de cocktail-service.
- **Repository**
  - *FeedbackRepository* : interface Spring Data JPA pour accéder à la base MySQL.
- **Listener**
  - *FeedbackListener* : écoute la file JMS pour recevoir les feedbacks et les stocker.
- **Config**
  - *SwaggerConfig* : génère automatiquement la documentation Swagger de l'API.

3.2 Liste des routes

cocktail-service

Méthode	Route	Description
GET	/search?name={nom}	Recherche un cocktail par nom
GET	/searchAll?name={nom}	Recherche tous les cocktails contenant le nom
GET	/random	Récupère un cocktail aléatoire
POST	/feedback	Envoie un avis (note + commentaire)

Feedback-service

Méthode	Route	Description
GET	/feedback/{drinkId}	Retourne la liste des avis et la note moyenne d'un cocktail

3.3 Communication asynchrone : intégration technique

La gestion des messages asynchrones est réalisée via JMS avec ActiveMQ comme broker. Deux composants clés interviennent :

Dans *cocktail-service*

- La classe *JmsSender* est responsable de l'envoi des messages.
  Elle utilise un *JmsTemplate* pour publier un objet *JmsMessage* sérialisé en JSON dans une file nommée *feedback-queue*.
- Ce composant est appelé depuis le *CocktailController* lorsque l'utilisateur envoie un feedback.

Dans feedback-service

- La classe *FeedbackListener* est annotée avec @JmsListener, ce qui lui permet d'écouter automatiquement la file *feedback-queue*.
- Lorsqu'un message arrive :
  - Il est désérialisé en *JmsMessage* via *ObjectMapper*
  - Un objet *Feedback* est construit
  - L'avis est persisté via *FeedbackRepository*

Cette mise en œuvre permet un découplage complet entre l'envoi (*cocktail-service*) et la réception (*feedback-service*), tout en maintenant un flux fiable et non bloquant.

3.4 Utilisation de Docker

Pour simplifier la gestion de l'infrastructure, un fichier docker-compose.yaml est utilisé afin de déployer automatiquement les conteneurs nécessaires, notamment :

- une base de données MySQL pour stocker les feedbacks,
- un serveur Apache ActiveMQ pour la gestion des messages asynchrones (JMS).

3.5 Documentation API Swagger

Pour garantir une documentation claire et à jour de l'API REST, une documentation interactive Swagger est générée automatiquement.

Accès à la documentation

Une fois les applications démarrées, la documentation Swagger est accessible aux adresses suivantes :

- http://localhost:8080/swagger-ui/index.html (feedback-service)
- http://localhost:8081/swagger-ui/index.html (cocktail-service)

4 Conclusion

4.1 Problèmes rencontrés et solutions

Aucun problème majeur n'a été rencontré.

Le projet s'est appuyé sur les concepts déjà vus en cours, notamment ceux du projet pays / pays-stats, concernant les microservices et la communication via un bus de messages JMS.

4.2 Bilan et évolutions

Le projet a permis de mettre en œuvre :

- une architecture microservices claire, avec séparation des responsabilités,
- la consommation d'une API externe publique (TheCocktailDB),
- la communication entre services via REST (synchronisation) et JMS (messagerie asynchrone),
- la persistance des feedbacks en base de données,
- une documentation automatique de l'API via Swagger.

Évolutions possibles :

- Ajouter une authentification des utilisateurs (OAuth2 avec JWT),
- Gérer les utilisateurs et associer les feedbacks à un auteur,
- Améliorer le suivi des logs et des erreurs (tracabilité, monitoring, etc.).