EXERCICE 2

1. Conception de l’Interface Utilisateur

1.1 Recherche de Livres afin de permettre à l’utilisateur de rechercher des livres efficacement.

Fonctionnalités :

- Champ de recherche : saisie libre (titre, auteur, mots-clés)
- Filtres disponibles : catégorie (roman, science, BD, etc.), auteur, date de publication, disponibilité (Disponible / Emprunté)
- Tri : popularité, date de publication, titre (A-Z)

| Rechercher un livre...

| [Filtres 🔼]
| [Catégorie : tous 🔼]
| 🔲 roman
| 🔲 science
| 🔲 BD
| 🔲 etc
| [Auteur : tous 🔽]
| [Disponibilité : Tous 🔽]
| [Trier par: Date 🔽]

| Harry Potter à l'école des sorciers
| Auteur : J.K. Rowling | Disponible
| date : 1997 | Catégorie : Fantastique
| Voir détails

| Le Petit Prince
| Auteur: Antoine de Saint-Exupéry | Emprunté
| date : 1943 | Catégorie: Conte
| Voir détails

1.2 Afficher toutes les informations nécessaires à la décision d’emprunt.

- Titre du livre
- Auteur
- Date de publication
- Nombre de pages
- Catégorie
- État (disponible / emprunté)
- Bouton "Emprunter" (si disponible) / Dernière date de retour estimée (si emprunté)
- Suggestions de livres similaires

| Harry Potter à l'école des sorciers
| Auteur: J.K. Rowling
| Publié en 1997
| 340 pages
| Catégorie: Fantastique
| Disponibilité : Disponible

| Résumé :
| Harry découvre qu’il est un sorcier...

| 🔘 Emprunter ce livre / prochain retour estimé vers le (jj)/(mm)/(aaaa)

| Livres similaires :
| - Harry Potter et la Chambre des Secrets
| - L’école des sorciers (BD)

1.3 L’utilisateur clique sur “Emprunter ce livre” puis une fenêtre modale s’ouvre avec :

- Confirmation du prêt
- Durée du prêt (par ex. 14 jours)
- Date de retour prévue
- Bouton "Confirmer"

Enfin, l’utilisateur reçoit une confirmation et une notification.

| Emprunter "Harry Potter à l'école des sorciers"

| Durée du prêt : 14 jours
| Date de retour : 30 avril 2025
| En cas de retard, une pénalité peut s'appliquer.

| [❌ Annuler] [✅ Confirmer l'emprunt]
---

2. Définir les Modèles de Données

2.1 Service Livre

Table : books

Champ	Type	Description
id	UUID	Identifiant unique du livre
title	String	Titre du livre
author	String	Auteur
category	String	Catégorie (ex: Roman, Science, etc.)
publicationDate	Date	Date de publication
pageCount	Integer	Nombre de pages
summary	Text	Résumé du livre
status	Enum	disponible/emprunté
popularity	Integer	Score calculé (nb d’emprunts, ou likes, etc.)

2.2 Service Utilisateur

Table : users

Champ	Type	Description
id	UUID	Identifiant unique utilisateur
firstName	String	Prénom
lastName	String	Nom
email	String	Adresse email
phoneNumber	String	Numéro de téléphone
address	Text	Adresse postale
dateOfBirth	Date	Date de naissance
membershipDate	Date	Date d’adhésion
penaltyPoints	Integer	Points de pénalité cumulés
isActive	Boolean	Statut actif/inactif de l’utilisateur

2.2 Service Emprunt

Table borrowings

Champ	Type	Description
id	UUID	Identifiant unique de l'emprunt
userId	UUID	ID utilisateur (récupéré via Auth - couplage auth)
bookId	UUID	ID du livre emprunté
borrowedAt	DateTime	Date de début de l'emprunt
dueDate	DateTime	Date limite de retour
returnedAt	DateTime	Date effective de retour (nullable)
penaltyAmount	Decimal	Montant de la pénalité si applicable
status	Boolean	disponible/emprunté

Table borrowed_users

Champ	Type	Description
userId	UUID	Référence (foreign key) vers borrowings.userId
firstName	String	Prénom de l'utilisateur
lastName	String	Nom de l'utilisateur

Table borrowed_books

Champ	Type	Description
bookId	UUID	Référence (foreign key) vers borrowings.bookId
title	String	Titre du livre
author	String	Auteur du livre

3. Concevoir le Système de Gestion des Emprunts

3.1 Enregistrement des emprunts

Quand un utilisateur veut emprunter un livre, tout part de son action sur l'interface. Il clique sur le bouton "Emprunter ce livre", ce qui ouvre une petite fenêtre modale de confirmation. La durée du prêt est de 14 jours pour pas que ca soit trop court ni trop long pour pouvoir le réemprunter, la date de retour estimée, et un petit rappel qu'une pénalité pourrait s'appliquer en cas de retard.

Quand il clique sur "Confirmer", l'application envoie une requête vers le backend du service d'emprunt. Cette requête contient les informations suivantes : le livre concerné, la date du jour (date d'emprunt), et l'identifiant de l'utilisateur qui est récupéré directement depuis le token JWT (donc via l'authentification).

Côté backend, la première chose qu'on fait, c'est de vérifier que le livre est bien disponible, donc qu'il n'est pas déjà emprunté par quelqu'un d'autre. Ensuite, on vérifie que l'utilisateur est actif, et éventuellement qu'il n'a pas atteint une limite d'emprunts ou trop de pénalités accumulées.

Si tout est bon, alors on peut enregistrer l'emprunt. Une ligne est ajoutée à la table borrowings avec les infos importantes : qui emprunte quoi, à quelle date, et quand doit-il rendre le livre. À ce moment-là, on en profite aussi pour capturer un instantané de certaines données :

- Du côté utilisateur : enregistré son prénom et son nom dans la table borrowed_users, pour pouvoir les afficher même si un jour il change de nom dans le système principal.

- Du côté livre : gardé le titre et l'auteur dans borrowed_books, histoire de ne pas être impacté si le titre ou les métadonnées du livre sont modifiées ailleurs.

Enfin, on n'oublie pas de mettre à jour le statut du livre en borrowed, pour que personne d'autre ne puisse le réserver entre-temps.

3.2 Calcul de la date de retour

La date de retour est très simple à déterminer. Elle dépend d'une règle de base : l'utilisateur a 14 jours pour lire le livre. Cette durée peut être un paramètre configurable dans le système, mais dans notre cas, on part sur 14 jours.

Donc, si l'utilisateur emprunte le livre aujourd'hui, on fait simplement une addition de 14 jours à la date actuelle pour obtenir la date limite de retour.

Par exemple, si on emprunte un livre le 17 avril 2025, la date de retour attendue est le 1er mai 2025.

3.3 Gestion des retours et des pénalités

Lorsqu'un utilisateur rend un livre, l'application envoie une autre requête pour dire : "ce livre a été rendu". Cette action va mettre à jour l'enregistrement de l'emprunt, en remplissant la colonne "returnedAt" avec la date du jour.

C'est à ce moment-là qu'on regarde si l'utilisateur est en retard. On compare la date de retour "returnedAt" avec la date limite "dueDate" :

- Si le livre a été rendu dans les temps, tout va bien, pas de pénalité.

- Si le livre a été rendu en retard, alors une pénalité s'applique. Le système calcule combien de jours de retard ont été accumulés, et applique un tarif de 0.50 € par jour (à voir).

Ce montant est ensuite enregistré dans le champ penaltyAmount de l'emprunt. En parallèle, ce serait de pouvoir incrémenter les points de pénalité de l'utilisateur dans le service utilisateur.

Une fois le retour enregistré et la pénalité calculée s'il y en a une, le statut du livre est remis à disponible, ce qui permet à d'autres lecteurs de l'emprunter à leur tour.

4. Interfaces et Services

Pour permettre à l'application de fonctionner efficacement, chaque service doit interagir de manière fluide avec sa propre base de données, mais aussi dialoguer avec les autres services. Par exemple, lorsqu'un utilisateur cherche un livre, le frontend envoie une requête au service livres. Ce service interroge sa propre base de données pour retrouver les ouvrages correspondant à la recherche, en prenant en compte les filtres, ainsi que les tris. C'est lui qui renvoie toutes les données nécessaires pour l'affichage de la liste de résultats ou des détails d'un livre.

Quand un utilisateur veut emprunter un livre, les choses deviennent un peu plus coordonnées. Le frontend envoie une requête au service d'emprunt, contenant l'identifiant du livre et l'ID utilisateur. Le service d'emprunt commence par vérifier que le livre est bien disponible en consultant directement l'état dans la table borrowings. Ensuite, il enregistre les données de l'emprunt dans sa table borrowings, les données du livre équivalents dans borrowed_books et pareil pour les données de l'utilisateur dans borrowed_users. Cette duplication légère permet d'avoir un historique cohérent même si les données du livre ou de l'utilisateur changent par la suite.

Toutes ces données sont mises à jour dans les bases respectives à travers des requêtes. Le service des livres, par exemple, met à jour le statut du livre passant de "disponible" à "emprunté", tandis que le service d'emprunt enregistre les dates de début et de fin prévues de l'emprunt en plus du statut.

Mais le système ne s'arrête pas là : pour offrir une expérience fluide, il doit aussi prévenir l'utilisateur quand la date de retour approche ou lorsqu'il est en retard. C'est là qu'intervient

un système de notifications. Celui-ci est un microservice indépendant. Il fonctionne en analysant régulièrement les emprunts en cours : par exemple, chaque nuit, une tâche automatisée passe en revue tous les emprunts actifs pour voir lesquels approchent de leur date limite. Si un emprunt est proche de l'échéance, une notification est préparée et envoyée par mail, SMS ou via une interface utilisateur. Si un emprunt dépasse sa date de retour sans avoir été marqué comme retourné, alors une autre notification est envoyée pour avertir l'utilisateur et le pousser à régulariser la situation. En parallèle, le système peut aussi calculer automatiquement le montant de la pénalité à appliquer.

Les interactions entre services se font de manière asynchrone, via un bus de messages. Lorsqu'un emprunt est enregistré, un message est publié sur un canal spécifique. Ce message est capté par les autres services concernés, qui réagissent et mettent à jour leurs bases de données respectives. Par exemple, le service des livres reçoit l'information d'un emprunt et met à jour le statut du livre en "emprunté". Ce modèle permet à chaque service d'agir de manière autonome et de mettre à jour sa base de données sans dépendre directement d'une réponse immédiate des autres services, ce qui évite de bloquer l'enregistrement de l'emprunt. Ainsi, les opérations se déroulent de manière fluide et les services peuvent réagir à l'événement quand cela est nécessaire.

En cas d'échec ponctuel d'un service, par exemple si le service de livres est temporairement indisponible, le système ne bloque pas entièrement l'opération : les microservices étant découplés, chaque service peut gérer localement l'échec, consigner l'erreur, et éventuellement déclencher une tentative de reprise ou alerter un autre composant. Ce type de résilience est essentiel pour garantir la disponibilité globale du système, même en cas de panne partielle, ce qui démontre le gros avantage du microservice.

Tout cela repose sur un découplage efficace des responsabilités : chaque service connaît uniquement ce dont il a besoin, mais tous communiquent entre eux par des API et/ou via des messages asynchrones, favorisant ainsi la scalabilité et la flexibilité du système.