Exercice sur le Low-Level Design (LLD)

Exercice 1.

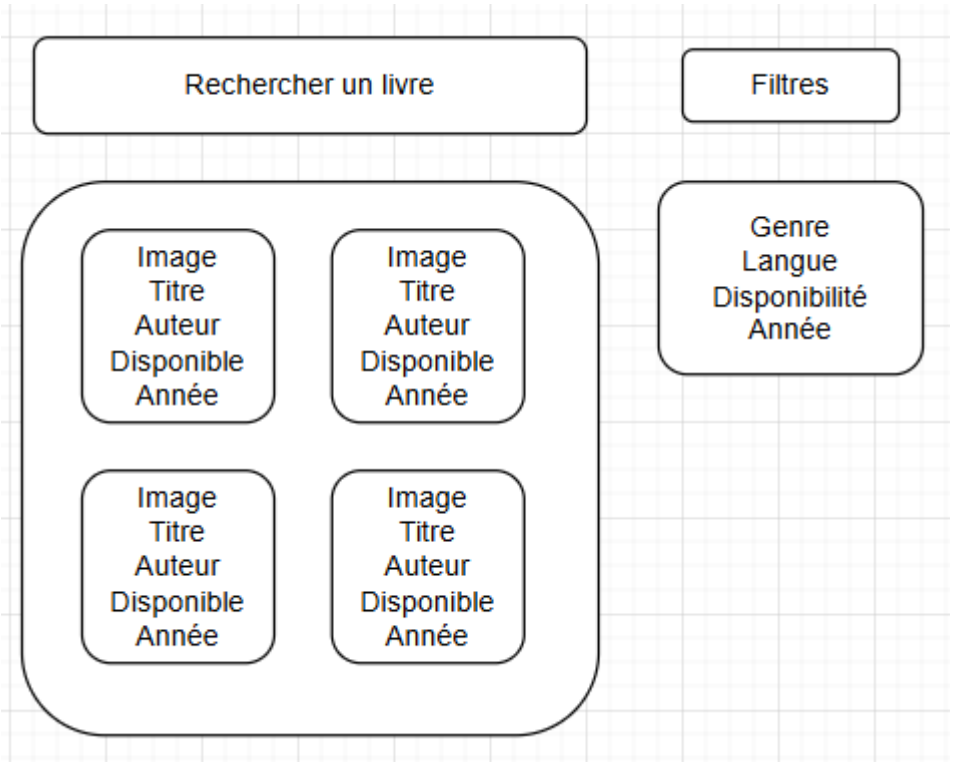
Décrivez et esquissez(maquettes) l'interface utilisateur pour les fonctionnalités suivantes :
Recherche de Livres:
Comment les utilisateurs recherchent-ils des livres ? Quels filtres peuvent-ils utiliser ?

1 - Zone de recherche principale avec :
- Champ de recherche (input), text libre a n'importe quel type de recherche, par titre auteur ou autre

2 - Filtres Avancé (optionnels qui peut etre affiché avec un bouton « filtres »
- Genre : Liste avec : science, histoire etc..
- Langue : Liste avec : français, anglais …
- Dispnibilité : Disponible ou emprunté
- Date de publication : 2 champs min et max

3 - Bouton « Rechercher »
- Déclenchement quand l'utilisateur clique sur le bouton, ou fait « Entrer » au clavier

4 - Resultat de la recherche :
Un livre par box (cliquable pour voir le détails du livre) :
Image couverture
Titre
Auteur
Disponible
Année



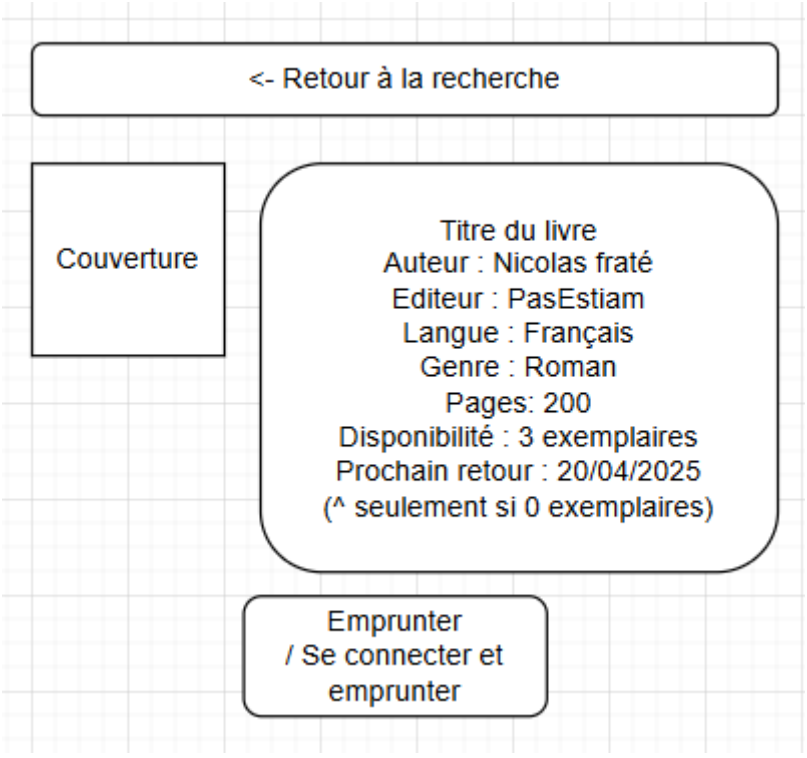
Détails du Livre: À quoi ressemble la page de détails d'un livre ? Quelles informations sont affichées ?

1 – Couverture du livre
Image affiché en haut a gauche, pouvoir cliquer pour agrandir l'image

2 – Informations sur le livre
Titre, Auteur(s), Date, editeur, langue, nombre de pages, genre, theme, résumé

3 – Disponibilité
Disponible ou non, si oui le nombre d'exemplaires du livre, et date du dernier retour qui est prévu si non

4 – Actions
bouton emprunter, si le livre est disponnible,
bouton se connecter et emprunter si le livre n'est pas disponnible
bouton retour pour revenir aux recherches



Processus d'Emprunt: Décrivez les étapes que l'utilisateur doit suivre pour emprunter un livre

1 – Connexion à l’application (Non nécésseraiment au départ si l’utilisateur clique sur « se connecter et emprunter »)
connexion via un formulaire de login, un token jwt est généré et stocké localement

2 – Recherche un livre
l’utilisateur utilise la page de recherche, clique sur un livre
(interface → API Gateway → Service de recherche)

3 – Consultation de la fiche du livre
accède a la page de détails
si le bouton Emprunter est actif → clique et emprunte
si le bouton « se connecter et emprunter » est actif → clique page de connexion et emprunte

4 – Clic sur le bouton « Emprunter »
L’interface envoie une requête POST avec l’identifiant du livre et le token
(interface → API GW → Validation du jwt via le service d’auth
→ Redirection vers l’emprunt (post)

5 – Traitement coté composant emprunt
Valide les règles (pas emprunté, limite du nombre de livre emprunté par l’utilisateur pas dépassé)
Créer un emprunt avec : livre_id, user_id, date_emprunt, date_retour_prevue
Met a jour la disponibilité du livre
(composant emprunt + bdd locale associé)

6 – Réponse
Réponse de succès si le livre est emprunté

Exercice 2

Définir les Modèles de Données

USERS	
integer	id
string	email
string	password_hash
string	first_name
string	last_name
string	role
penalty_points	integer

EMPRUNT	
integer	id
integer	user_id
integer	book_id
string	book_title
date	emprunt_date
date	due_date
date	return_date
penalty	float

BOOKS	
integer	id
string	title
string	author
string	publisher
date	date
string	language
string	genre
string	summary
string	cover_url
integer	nb_copies

Précisions :

1 - Le modèle books est géré par le service de recherche des livres et est partiellement dupliqué coté service emprunt (juste pour book_id, avec sa propre table books dans emprunt)
- nb_copies : nombre d'exemplaire du livre disponible (donc pas empruntés)

2 - Le modèle utilisateur est géré par le service authentification et est partiellement dupliqué coté service emprunt (juste pour user_id, avec sa propre table users dans emprunt)
- role : admin ou user
- penalty_points : score cumulé des pénalités de l'utilisateur

3 - Le modèle emprunt est géré par le composant emprunt.
- due_date : date a laquelle l'utilisateur doit rendre le livre, prévue automatiquement lors de l'emprunt (14 jours)
- return_date qui est la date de retour réel de l'utilisateur qui reste nul jusqu'a que l'utilisateur retourne le livre.
- penalty : pénalité spécifique à un emprunt

Exercice 3

Concevoir le Système de Gestion des Emprunts

Enregistrement des Emprunts: Comment le système enregistre-t-il un nouvel emprunt ?

Requête POST sur /emprunt avec :

book_id et le jwt dans l'entête

Vérifications :
- Le livre est disponibles
- l'utilisateur n'emprunte pas déjà le livre
- n'a pas dépassé la limite d'emprunts actif

Création de l'enregistrement d'emprunts
- insertion dans la base emprunter de la ligne correspondant a l'emprunt

Mise à jour de la disponibilité
- update dans la table books, en réduisant le nombre de copie de 1 du livre emprunté

Calcul des Dates de Retour: Comment la date de retour est-elle déterminée pour chaque emprunt ?

Durée d'emprunt standard fixée par défaut à 14 jours.
Logique : Au moment de l'emprunt du livre, la date actuelle + 14 jours est calculé et mise à due_date dans la table emprunt

Variantes possibles :
Prolongation manuelle au moment de la réservation, avec une valeur a enregistrer dans la table emprunt.

On encore des durées différentes selon le genre du livre :
Exemples :
- BD = 7 jours
- Roman = 14 jours

Donc avoir une donnée loan_duration_days dans la table books.
L'utilisateur clique sur Emprunter,
le frontend envoie : book_id = 65,
le service emprunt a sa propre copie de book_id = 65 avec

```
{
        book_id : '65'
        loan_duration_days : '14'
        …
}
```

Il récupère loan_duration_days, calcule avec la date actuelle par exemple :
16/04/2025 + 14 jours = 30/04/2025
et il l'enregistre dans due_date :

```
{
        user_id : '25',
        book_id : '65',
        emprunt_date : '2025-04-16',
        due_date : '2025-04-30'
}
```

Gestion des Pénalités: Comment les pénalités sont-elles calculées pour les retours en retard ?

Moment ou les pénalité sont déclenché :

- Les pénalités seront évaluées à la date de retour (return_date)
- et automatiquement, en tache de fond, pour détecter les retards en cours, et recalculer ceux actuels (via un job planifié comme avec un cron, ça sert à exécuter des taches à un moment précis)

Les étapes a faire pour les gérer :

- on compare la date de retour (ou la date du jour si le livre n'a pas été rendu) avec la date limite (due_date)
- si il y a bien un retard alors une pénalité est calculée

puis calculer :

- On compte le nombre de jour de dépassement, avec par exemple 0,50€ par jour de retard

avec des conséquences éventuelles :

- de 0 à 9€ aucune conséquences
- de 10 à 20 → avertissement visuel
- de 21 à 30 → suspension temporaire des emprunts pendant 7 jours
- 31+ → blocage du compte

Exercice 4

Interfaces et Services

Interagir avec la Base de Données: Comment les informations sont-elles récupérées et mises à jour dans la base de données ?

Chaque microservice possède sa propre base de données, et y accède via une interface dédiée

Donc pour chaque composant on a :

Recherche des livres :
- récupère les livres avec filtres
- recherche textuelle
- met à jour le stock total

Emprunt :
- Créer une transaction d'emprunts
- mettre à jour return_date, status et penalty
- récupère les emprunts actifs pour un utilisateurs
- vérifier les retards

Auth :
- récupère un profil
- mettre à jour penalty_points et status

Chaque service métier possède ses logique métier, et n'écrit pas directement dans la base :
- Il demande a son interface (repository) par exemple : « donne-moi tous les emprunts de cet utilisateur »
- Le repository sert a traduire une demande en requête SQL, parler à la base et retourner des objects propre au service

Donc :

- Chaque service a donc une interface interne d'accès à sa base. Et ces interfaces sont isolées du code métier pour que ça rend le service testable et propre.
- Donc l'idée est de ne pas mélanger la logique métier et les requetes SQL, pour organiser le code avec des services d'un coté, et des repositories de l'autres.

Notifier les Utilisateurs: Comment les utilisateurs sont-ils notifiés de l'approche de la date de retour ou des pénalités dues ?

Les notifications vont prévenir de l’approche d’une date de retour, alerter un retard/pénalité et informer un compte bloqué

Par exemple :

Type de la notif	Contenue de la notif	Fréquence
Rappel de retour	« votre livre est à rendre dans 2 jours »	48h avant la date à rendre (due_date)
retard	« votre livre est en retard depuis 3 jours	Chaque jour jusqu’au retour du livre
suspension	« votre compte est bloqué »	Dès que l’utilisateur se connecte, et a chaque fois
Pénalité	« une pénalité de 2,50€ a été ajoutée »	Dès que le livre est rendu (pas besoin de faire chaque jours car la partie retard le fait déjà

Fonctionnement :

1 – Service notifications

- Je n’y ai pas pensé au début, mais pour gérer toute la partie notif, faire un composant notifications semble être une bonne idée.
- Il reçoit donc des événements émis par d’autres services liée au future notifications : book_due_soon, book_late, panalty_apply ...
- Avec des envoie de mails, sms, ou notifications en push

2 – Un système de messagerie interservices

- pour que ça fonctionne quand même pour du microservice, il faut ajouter un composant intermédiaire, donc un message broker. Ça permet a qu’un service soit down sans bloquer les autres.
- Par exemple quand le service emprunt détecte un emprunt en retard, il publie un événement du style :

{
type : ‘book_late’
user_id : ‘25’,
book_title : ‘titre du super livre’ (par contre il faudra mettre book_title dans la bdd emprunt)
due_date : ‘3’
}

- le service notification écoute ce type d’événement via le broker
- et quand il le reçoit il envoie un email, sms ou notif push
- si le service notif est down, pas grave le message est dans la file et il le traitera quand il revient.