**Atelier13**

*On souhaite mettre en place une architecture microservice pour un système de* ***gestion des emprunts d’une librairie****.*

*L’objectif de cet exemple est d’identifier les services composant ce système ;*

*Chaque service doit être :*

* + *Défini clairement*
  + *Autonome*
  + *Possède un canal de communication claire*

**Gestion des emprunts d’une librairie :**

Le système offrira les fonctionnalités suivantes :

* Gestion des livres
* Gestion des emprunts
* Gestion des clients
* Envoi des notification (Exemple: Quand un client indique qu’il a besoin d’un livre, le système lui enverra une notification une fois ce livre est retourné)
* Paiement

**Gestion d’une librairie :**

* On peut distinguer 4 services :

**Service**

**Livre**

**Service**

**Emprunt**

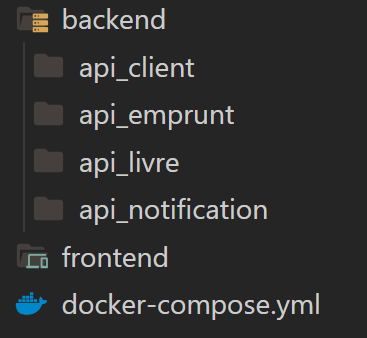
**Service**

**Client**

**Service**

**Notification**

- Le projet sera structuré de la façon suivante :



Application React

Service Notification

Service Livre

Service Emprunt

Service Client

Front End (React)

Service

Livre

Service

Emprunt

Service

Client

REST

REST

REST

DB

DB

DB

Service

Notification

**Queue**

**Travail a réaliser :**

* **Le service « Livre » :**
* Permet de gérer le stock des livres de la librairie.
* Dispose d’une base de données.
* Les données sur les livres que nous voulons enregistrées sont : code, titre, description, auteur
* Ne dépend pas des autres services (On n’a pas besoin des autres services pour que ce service fonctionne correctement)
* Expose une API Rest :

|  |  |  |
| --- | --- | --- |
| **Fonctionnalité** | **Méthode / Chemin** | **Code retourné** |
| Retourner les informations d’un livre donné | GET /api/v1/livre/{idLivre} | 200 OK  404 Non trouvé |
| Ajouter un nouveau livre | POST /api/v1/livre | 200 OK |
| Modifier un livre | PUT /api/v1/livre/{idLivre} | 200 OK  404 Non trouvé |
| Supprimer un livre | DELETE /api/v1/livre/{idLivre} | 200 OK  404 Non trouvé |

* **Le service « Client » :**
* Permet de gérer les clients de la librairie.
* Dispose d’une base de données.
* Les données sur les livres que nous voulons enregistrées sont : nom, prénom, email.
* Ne dépend pas des autres services (On n’a pas besoin des autres services pour que ce service fonctionne correctement)
* Expose une API Rest :

|  |  |  |
| --- | --- | --- |
| **Fonctionnalité** | **Méthode / Chemin** | **Code retourné** |
| Retourner les informations d’un client donné | GET /api/v1/client/{idClient} | 200 OK  404 Non trouvé |
| Ajouter un nouveau client | POST /api/v1/client | 200 OK |
| Modifier un client | PUT /api/v1/client/{idClient} | 200 OK  404 Non trouvé |
| Supprimer un client | DELETE /api/v1/client/{idClient} | 200 OK  404 Non trouvé |

* **Le service « Emprunt » :**
* Permet de gérer les emprunts des livres
* Dispose d’une base de données
* Pour ajouter un emprunt :
  1. L’utilisateur doit fournir le code de livre et le nom de client.
  2. Les informations du client seront récupérées du service Client.
  3. De même pour le livre á emprunter, ses informations seront récupérées du service Livre.
  4. La date d’emprunt est renseignée par la date du jour.
  5. La date retour est renseigné par la valeur null.
* Les données sur les emprunts que nous voulons enregistrées sont : livre, client, date d’emprunt et date de retour.
* Expose une API REST :

|  |  |  |
| --- | --- | --- |
| **Fonctionnalité** | **Méthode / Chemin** | **Code retourné** |
| **Ajouter un nouveau emprunt** | **POST /api/v1/emprunt** | **200 OK** |
| **Retourner un livre** | **POST /api/v1/emprunt** | **200 OK** |
| **Retourner les emprunts d’un client donné** | **GET /api/v1/emprunt/{idClient}** | **200 OK**  **404 Non trouvé** |

* **Le service « Notification » :**
* Est utilisé pour envoyer des notifications aux clients de la librairie (Ex. un livre a été retourné, nouveaux livres ont été ajoutés …)
* Asynchrone : Les services appelant n’attendent pas une réponse immédiate)

(Ex. Le service livre veut envoyer une notification à 1000 clients suite à un ajout d’un nouveau livre)

* La communication est assuré par le biais des « Queues »: il s’agit d’une diffusion d’événement.
* Ce package utilise principalement le package nodemailer pour envoyer des emails (notification).
* **Front End :**

- Proposer une application Front End a l’aide du framework React pour utiliser les services déclarés.

* **Docker-compose :**

- Conteneuriser les modules de l’application.