133 – Réaliser des applications Web en Session-Handling

Documentation de projet

Khalil Siwan - Pochon Marius

Date de création : 21.03.2025  
Version 1 du 21.03.2025



Projet du 21.03.2025 au 17.04.2025

Table des matières

[1 Introduction et contexte du projet 3](#_Toc196770774)

[1.1 Introduction et but 3](#_Toc196770775)

[1.1.1 Description détaillée 3](#_Toc196770776)

[1.1.2 Réalisation 4](#_Toc196770777)

[1.1.3 À rendre 5](#_Toc196770778)

[1.2 Notre projet 5](#_Toc196770779)

[1.3 Objectifs du projet 5](#_Toc196770780)

[2 Analyse à faire complètement avec EA -> à rendre uniquement le fichier EA 6](#_Toc196770781)

[2.1 Use case client et use case Rest 6](#_Toc196770782)

[2.2 Activity Diagram d'un cas complet navigant dans les applications avec les explications 9](#_Toc196770783)

[2.3 Maquettes du projet avec la navigation intégrée et ses explications 10](#_Toc196770784)

[2.4 Sequence System global entre les applications 12](#_Toc196770785)

[3 Conception à faire complétement avec EA -> à rendre uniquement le fichier EA 13](#_Toc196770786)

[3.1 Class Diagram complet avec les explications de chaque application 13](#_Toc196770787)

[4 Bases de données 14](#_Toc196770788)

[4.1 Modèles ER 14](#_Toc196770789)

[4.2 Modèles WorkBench MySQL 14](#_Toc196770790)

[5 Implémentation des applications <Le client Ap1> et <Le client Ap2> 15](#_Toc196770791)

[5.1 Une descente de code client 1 15](#_Toc196770792)

[5.1.1 register.js 15](#_Toc196770793)

[5.1.2 login.js 15](#_Toc196770794)

[5.1.3 AllBooks.js 15](#_Toc196770795)

[5.1.4 addBook.js 16](#_Toc196770796)

[5.1.5 BookInfo.js 16](#_Toc196770797)

[5.1.6 addAuteur.js 16](#_Toc196770798)

[5.2 Une descente de code client 2 16](#_Toc196770799)

[5.2.1 addAuteur.js 16](#_Toc196770800)

[5.2.2 addBook.js 17](#_Toc196770801)

[5.2.3 allBook.js 17](#_Toc196770802)

[5.2.4 BookInfo.js 18](#_Toc196770803)

[5.2.5 login.js 19](#_Toc196770804)

[5.2.6 register.js 19](#_Toc196770805)

[6 Implémentation de l'application <API Gateway> 22](#_Toc196770806)

[6.1 Une descente de code APIGateway 22](#_Toc196770807)

[6.1.1 GatewayController 22](#_Toc196770808)

[6.1.1.1 Contexte 22](#_Toc196770809)

[6.1.1.2 Technologies utilisées 22](#_Toc196770810)

[6.1.1.3 Architecture 22](#_Toc196770811)

[6.1.1.4 Routes définies dans le contrôleur 22](#_Toc196770812)

[7 Implémentation des applications <API élève1> et <API élève2> 25](#_Toc196770813)

[7.1 Une descente de code de l'API REST 1 25](#_Toc196770814)

[7.1.1 Auteurs.java 25](#_Toc196770815)

[7.1.2 Livres.java 26](#_Toc196770816)

[7.1.3 Controller.java 28](#_Toc196770817)

[7.1.3.1 Méthodes de la classe 30](#_Toc196770818)

[7.2 Une descente de code de l'API REST 2 32](#_Toc196770819)

[7.2.1 User.java 32](#_Toc196770820)

[7.2.2 UserController.java 33](#_Toc196770821)

[7.2.2.1 Méthodes 34](#_Toc196770822)

[7.2.3 UserDTO.java 35](#_Toc196770823)

[7.2.4 UserRepository.java 36](#_Toc196770824)

[8 Hébergement 37](#_Toc196770825)

[9 Outils, langages (versions, définitions, installations) 42](#_Toc196770826)

[10 Tests de fonctionnement du projet 43](#_Toc196770827)

[11 Auto-évaluations et conclusions 45](#_Toc196770828)

[11.1 Auto-évaluation et conclusion de Siwan 45](#_Toc196770829)

[11.2 Auto-évaluation et conclusion de Marius 45](#_Toc196770830)

# Introduction et contexte du projet

## Introduction et but

L’apprenant doit être capable, à la fin de la leçon :

* Rédiger un cahier des charges.
* Concevoir un diagramme entité-relations sur la base d'un cahier des charges
* Transcrire un schéma entité-relations en relationnel
* Alimenter une base de données Mysql
* Créer une API Rest attachée à cette base de données
* Créer une API Gateway qui gère la session pour 2 backend
* Utiliser les compétences web pour compléter le projet avec un frontend

Une image contenant texte, capture d’écran, diagramme, conception

Le contenu généré par l’IA peut être incorrect.Le projet se fait par groupe et doit démontrer vos compétences à la création de plusieurs applications WEB ainsi qu’à leur mise en relation.

### Description détaillée

**Application clientes pour chaque étudiant**

Deux applications <Le client Ap1> et <Le client Ap2> seront réalisées en HTML-CSS-JS, en séparant bien les couches. Ces deux applications clientes, réaliseront le dialogue avec l’application <APIGateway> à travers le protocole HTTP. Chacune de ces applications clientes sera hébergée au final sur l'hébergement Alphosting de l'étudiant. L’application <Le client Ap1> sera utilisable que par un Utilisateur connecté, l’autre <Le client Ap2> que par un Administrateur connecté ! L’application <Le client Ap1> connectée par un Utilisateur pourra afficher les données choisies par le groupe et naviguer dans les pages de l’application. L’application <Le client Ap2> connectée par un Administrateur pourra gérer les données des comptes utilisateurs uniquement.

**Application APIGateway en commun**

L’application <APIGateway> en JAVA jouera le rôle de contrôleur d’accès en demandant la vérification à une application JAVA Web RESTFul d’un des apprentis et permettra aux applications <<Le client Ap1>> et <<Le client Ap2>> de réaliser leur fonction. Cette application permettra de rediriger les informations vers la bonne application cliente après le contrôle du niveau de l’utilisateur.

Cette application sera hébergée en format container Docker sur un serveur cloud. Elle sera accessible depuis les 2 clients.

**Application API pour chaque étudiant**

L’application <API élève1> en JAVA s’occupera d’aller vérifier les accès dans une base de données MySQL, cette application sera mise en place en utilisant le principe des services RESTFul.   
L’application <<API élève2>> de même technologie que celle précédente aura le rôle d’aller chercher des données choisies par le groupe à donner en retour pour sa gestion.

Ces 2 applications seront hébergées en format container Docker sur un serveur cloud. Ces applications ne seront accessible que par l'APIGateway directement. Ces 2 applications auront l'accès exclusif à leur DB respective.

**Toutes les applications seront basées sur le pattern MVC.**

### Réalisation

1. Les applications doivent se réaliser selon la norme apprise à l’école. Chaque membre du groupe est responsable d’une partie. Un membre du groupe sera responsable de son application cliente, de son application API et en commun avec l’autre apprenti responsable de l’application ApiGateway.
2. La réalisation du projet doit être documentée selon le modèle fourni, doc\_Gxy\_30023z\_133.docx. Chaque membre du groupe est responsable de documenter sa partie. Les membres du groupe sont responsables des parties communes de la documentation.
3. Toutes les analyses et les conceptions des différentes applications devront être documentées techniquement en UML dans un projet unique UML\_EA\_\_Gxy\_30023z\_133.eap, organisé selon les étapes de projet données dans le fichier de base.
4. Le projet sera présenté et expliqué au formateur une fois terminé. Chaque membre du groupe présentera sa partie du travail de groupe.

Le projet de chaque groupe est audité semaine après semaine !

Une ToDoList des tâches à faire pour chacun, en deux colonnes, sera à gérer de semaine en semaine.

### À rendre

Les applications, les projets, le projet EA, la documentation de chacun, les scripts des bases de données, la ToDoList de travail du groupe bien séparé par membre avec le total d’heures chacun, etc. doivent se retrouver bien organisée sur le repository Github.

Arborescence projet Github :

|  |  |
| --- | --- |
| Readme.md | Descriptif du projet avec les liens (url) des clients et de l'APIGateway |
| rapportEleve1.pdf | Rapport de l'élève1 |
| rapportEleve2.pdf | Rapport de l'élève2 |
| Client1 | Dossier du code du client de l'élève1 |
| Client2 | Dossier du code du client de l'élève2 |
| ApiGateway | Dossier du code de l'APIGateway en commun |
| Rest1 | Dossier du code de l'API Rest de l'élève1 |
| Rest2 | Dossier du code de l'API Rest de l'élève2 |

## Notre projet

Notre projet est le suivant : une bibliothèque Web.

**Marque-PAF** (Projet à faire) est une application web qui propose aux utilisateurs de gérer une bibliothèque. Il repose sur une architecture avec 2 applications front-end, une application qui ferait la passerelle vers 2 applications backend avec une base de données SQL chacun.

## Objectifs du projet

* Permettre aux utilisateurs de **se connecter** ou de **s’enregistrer**.
* Une fois connecter, l’utilisateur a accès à une interface pour **consulter la liste des livres**.
* Permettre aux administrateurs de **gérer** (ajouter, supprimer) les livres.
* Centraliser les interactions via un **API Gateway** pour une meilleure gestion des requêtes.

# Analyse à faire complètement avec EA -> à rendre uniquement le fichier EA

## Use case client et use case Rest

Use Case client 1

Une image contenant texte, capture d’écran, diagramme, cercle

Le contenu généré par l’IA peut être incorrect.

Use case client 2

Une image contenant texte, capture d’écran, diagramme, cercle

Le contenu généré par l’IA peut être incorrect.

Use case Rest 1

Une image contenant texte, capture d’écran, diagramme, ligne

Le contenu généré par l’IA peut être incorrect.

Use case Rest 2

Une image contenant capture d’écran, diagramme, cercle, texte

Le contenu généré par l’IA peut être incorrect.

## Activity Diagram d'un cas complet navigant dans les applications avec les explications

Une image contenant texte, diagramme, capture d’écran, Parallèle

Le contenu généré par l’IA peut être incorrect.

## Maquettes du projet avec la navigation intégrée et ses explications

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, nombre, Police

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

## Sequence System global entre les applications

Une image contenant texte, capture d’écran, Parallèle, nombre

Le contenu généré par l’IA peut être incorrect.

# Conception à faire complétement avec EA -> à rendre uniquement le fichier EA

## Class Diagram complet avec les explications de chaque application

Voir ce fichier pour les diagrammes :

 

# Bases de données

## Modèles ER

Une image contenant diagramme, cercle, ligne, conception

Le contenu généré par l’IA peut être incorrect.

Une image contenant capture d’écran, ligne, diagramme, cercle

Le contenu généré par l’IA peut être incorrect.

## Modèles WorkBench MySQL

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

# Implémentation des applications <Le client Ap1> et <Le client Ap2>

## Une descente de code client 1

### register.js

* **Quand ?** : Quand un utilisateur remplit le formulaire d'inscription.
* **Que fait-il ?**
  + Récupère le username, password, confirmPassword, et l'état de la case à cocher "isAdmin".
  + Vérifie que les deux mots de passe correspondent.
  + Crée un JSON { username, password, role } où role = "admin" ou "user".
  + **Envoie un POST** vers http://localhost:8080/register avec ce JSON.
  + Si succès : il redirige vers index.html.
  + Si erreur : il affiche un message d'erreur sous le formulaire.

### login.js

* **Quand ?** : Quand l'utilisateur se connecte.
* **Que fait-il ?**
  + Récupère le username et password.
  + Envoie un **POST** vers http://localhost:8080/login.
  + Attend la réponse texte :
    - Si text === "admin", il autorise l'accès à AllBooks.html.
    - Sinon, il affiche : **"Accès refusé : vous n'êtes pas administrateur"**.
  + Si problème serveur ou identifiants invalides : message d'erreur.

### AllBooks.js

* **Quand ?** : Dès que AllBooks.html est chargé.
* **Que fait-il ?**
  + Fait un **GET** vers http://localhost:8080/client1/getLivres.
  + Reçoit un **tableau de livres** en JSON.
  + Pour chaque livre :
    - Crée un élément <li> qui contient :
      * Le titre du livre sous forme de lien cliquable (BookInfo.html?id=xxx).
      * Un bouton "poubelle" pour supprimer le livre.
  + Initialise les icônes avec lucide.createIcons().

### addBook.js

**Objectif** : Permettre d'**ajouter un livre**.

* Au chargement :
  + Remplit un <select> avec les auteurs existants via un GET client1/getAuteurs.
* Lors du submit :
  + Récupère : title, genre, publicationYear, authorId.
  + Envoie une requête POST **en application/x-www-form-urlencoded** vers :

/client1/addLivre

* + Redirige vers AllBooks.html en cas de succès.

### BookInfo.js

**Objectif** : **Afficher les détails** d'un livre.

* Récupère id dans l'URL (BookInfo.html?id=...).
* Fait un GET vers :

/client1/getLivre?id={id}

* Affiche :
  + Titre
  + Nom de l’auteur
  + Année de publication
  + Genre

En cas d'erreur ou d'ID manquant : message d'erreur affiché.

### addAuteur.js

**Objectif** : Permettre d'**ajouter un auteur** avant d'ajouter un livre.

* Lors du submit du formulaire auteurForm :
  + Récupère : name, birthYear, nationality.
  + Envoie une requête POST **x-www-form-urlencoded** vers :

/client1/addAuteur

* Si succès : redirection automatique vers addBook.html pour ajouter ensuite un livre.

## Une descente de code client 2

### addAuteur.js

**Objectif** : Permettre d'ajouter un **auteur**.

• **Lors du submit** :

* Empêche le comportement par défaut du formulaire.
* Récupère : name, birthYear, nationality.
* Envoie une requête POST en application/x-www-form-urlencoded vers :

/client1/addAuteur

* Redirige vers addBook.html en cas de succès.

### addBook.js

**Objectif** : Permettre d'ajouter un **livre**.

• **Au chargement** :

* Récupère l'élément <select> des auteurs.
* Effectue une requête GET vers :

/client1/getAuteurs

* Remplit dynamiquement le <select> avec les auteurs existants (id et name).

• **Lors du submit** :

* Empêche le comportement par défaut du formulaire.
* Récupère : title, genre, publicationYear, authorId.
* Envoie une requête POST en application/x-www-form-urlencoded vers :

/client1/addLivre

* Redirige vers AllBooks.html en cas de succès.

### allBook.js

**Objectif** : Permettre d'**afficher tous les livres** et **supprimer un livre**.

• **Au chargement** (DOMContentLoaded) :

* Effectue une requête GET vers :

/client1/getLivres

* Vide le <ul> ou <div> ayant l'ID bookList.
* Pour chaque livre reçu :
  + Crée un <li> avec un style Tailwind (flex, justify-between, etc.).
  + Ajoute un titre cliquable (lien vers BookInfo.html?id=<id>).
  + Ajoute un bouton **poubelle** (icône Lucide) pour la suppression.
* Met à jour les icônes avec :

lucide.createIcons();

• **Lors du clic sur la poubelle** :

* Demande confirmation :

"Voulez-vous vraiment supprimer ce livre ?"

* Si confirmé :
  + Effectue une requête DELETE vers :

/client1/deleteLivre/{id}

* + Rafraîchit la page (location.reload()) si succès.
  + Affiche une alerte si échec.

### BookInfo.js

**Objectif** : Afficher **les détails** d'un livre.

• **Au chargement** (DOMContentLoaded) :

* Récupère l'id du livre à partir de l'URL :

window.location.search

→ via URLSearchParams.

* **Si un bookId est présent** :
  + Effectue une requête GET vers :

/client1/getLivre?id=<bookId>

* + Si le livre est trouvé (data non null) :
    - Affiche :
      * **Titre** (data.title)
      * **Auteur** (data.auteur.name)
      * **Année de publication** (data.publicationYear)
      * **Genre** (data.genre)
    - Dans l'élément ayant l'id bookDetails.
  + Sinon :
    - Affiche un message :

Livre non trouvé.

* **Si aucune id n'est présente** dans l'URL :
  + Affiche un message :

ID du livre manquant dans l'URL.

• **Gestion des erreurs** :

* Si une erreur réseau survient, affiche :

Une erreur est survenue lors de la récupération des informations du livre.

### login.js

**Objectif** : Gérer la **connexion** d'un utilisateur.

• **Au chargement** (submit sur le formulaire) :

* Récupère les **valeurs** de username et password :

const username = document.getElementById('username').value;

const password = document.getElementById('password').value;

* **Envoi d'une requête POST** vers l'API login avec les **données JSON** :

fetch('http://localhost:8080/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ username, password })

})

• **Réponse du serveur** :

* Si la connexion est **OK** (response.ok) :
  + Si le serveur répond 'user' :
    - Affiche un message d'alerte avec la réponse :

alert(text);

* + - Redirige vers AllBooks.html :

window.location.href = './html/AllBooks.html';

* + Sinon, si un autre type de réponse est reçu (ex : admin ou autre), affiche un message d'erreur :

errorMsg.textContent = "Accès refusé : vous n'êtes pas un utilisateur.";

* Si la requête échoue :
  + Affiche le message d'erreur retourné par le serveur ou un message générique :

errorMsg.textContent = text || 'Erreur de connexion';

• **Gestion des erreurs** :

* En cas d'exception (erreur réseau, serveur non disponible), un message est affiché :

errorMsg.textContent = 'Erreur serveur';

### register.js

**Objectif** : Permettre à un utilisateur de **s'inscrire** en tant qu'**utilisateur** ou **administrateur**.

* 1. Récupérer les données du formulaire lors du submit :
* **Nom d'utilisateur** :

const username = document.getElementById('username').value;

* **Mot de passe** :

const password = document.getElementById('password').value;

* **Confirmation du mot de passe** :

const confirmPassword = document.getElementById('confirmPassword').value;

* **Vérification du rôle** (Admin ou utilisateur) :

const isAdmin = document.getElementById('isAdmin').checked;

const role = isAdmin ? "admin" : "user";

* 2. Validation des mots de passe :
* Si le mot de passe et sa confirmation ne correspondent pas :

if (password !== confirmPassword) {

errorMsg.textContent = 'Les mots de passe ne correspondent pas';

return;

}

* 3. Envoi de la requête d'inscription :
* **Corps de la requête** : Contient le **nom d'utilisateur**, le **mot de passe**, et le **rôle** (utilisateur ou admin).

fetch('http://localhost:8080/register', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ username, password, role })

})

* 4. Gestion de la réponse du serveur :
* Si la requête **réussit** (response.ok), un message de succès est affiché et l'utilisateur est redirigé :

alert(text);

window.location.href = '../index.html';

* Si la requête échoue, l'erreur retournée est affichée :

errorMsg.textContent = text || 'Erreur de connexion';

* 5. Gestion des erreurs réseau :
* En cas d'erreur (connexion réseau ou serveur), un message d'erreur générique est affiché :

errorMsg.textContent = 'Erreur serveur';

# Implémentation de l'application <API Gateway>

## Une descente de code APIGateway

### GatewayController

#### Contexte

Le contrôleur GatewayController fait partie du projet d'API Gateway qui agit en tant que middleware entre le front-end (interface utilisateur) et plusieurs services REST internes (gestion des utilisateurs, des livres, et des auteurs). L'objectif est de centraliser la gestion des requêtes provenant du client, les diriger vers les services appropriés et retourner les réponses.

#### Technologies utilisées

* **Spring Boot** : Framework utilisé pour créer l'API Gateway et gérer les requêtes HTTP.
* **RestTemplate** : Utilisé pour effectuer des appels HTTP vers les services REST externes.
* **Session HTTP** : Permet de gérer l'état de l'utilisateur (connexion, rôle).
* **JSON** : Format des données échangées entre le client et les services REST.

#### Architecture

Le contrôleur GatewayController gère plusieurs aspects de l'application :

* **Gestion des utilisateurs** : Connexion, inscription.
* **Gestion des livres et des auteurs** : Création, récupération et suppression de livres et d'auteurs.
* **Passerelle (proxy)** : Redirige les requêtes du front-end vers les services REST appropriés (REST1 pour la bibliothèque, REST2 pour les utilisateurs).

#### Routes définies dans le contrôleur

* 1. Connexion de l'utilisateur
* **Route** : POST /login
* **Fonction** : Permet à un utilisateur de se connecter en envoyant ses informations (username, password).
* **Description** :
  + Envoie une requête POST au service REST2 (/login).
  + Si la connexion est réussie, l'utilisateur est redirigé et ses informations de session sont enregistrées.
  + Retourne un message de succès ou d'erreur.
* **Exemple de requête** :

{

"username": "user1",

"password": "password123"

}

* 2. Inscription de l'utilisateur
* **Route** : POST /register
* **Fonction** : Permet de créer un nouvel utilisateur.
* **Description** :
  + Envoie une requête POST au service REST2 (/register).
  + Retourne un message de succès ou d'erreur selon l'issue de l'inscription.
* **Exemple de requête** :

{

"username": "newUser",

"password": "newPassword",

"role": "user"

}

* 3. Gestion des livres
* **Route** : GET /client1/getLivres
* **Fonction** : Récupère tous les livres.
* **Description** : Envoie une requête GET au service REST1 pour obtenir la liste des livres.
* **Réponse** : Liste de livres au format JSON.
* **Route** : GET /client1/getLivre?id={id}
* **Fonction** : Récupère un livre par son ID.
* **Description** : Envoie une requête GET pour récupérer un livre spécifique depuis REST1.
* **Réponse** : Détails du livre (titre, auteur, genre, année de publication).
* **Route** : POST /client1/addLivre
* **Fonction** : Ajoute un livre dans la base de données.
* **Description** : Envoie une requête POST pour ajouter un livre (titre, genre, année, ID de l'auteur).
* **Exemple de requête** :

title=Title&genre=Fiction&publication\_year=2022&author\_id=1

* **Route** : DELETE /client1/deleteLivre/{id}
* **Fonction** : Supprime un livre en fonction de son ID.
* **Description** : Envoie une requête DELETE pour supprimer un livre.
* 4. Gestion des auteurs
* **Route** : GET /client1/getAuteurs
* **Fonction** : Récupère la liste des auteurs.
* **Description** : Envoie une requête GET pour obtenir tous les auteurs.
* **Route** : POST /client1/addAuteur
* **Fonction** : Ajoute un auteur dans la base de données.
* **Description** : Envoie une requête POST pour ajouter un auteur (nom, année de naissance, nationalité).
* **Exemple de requête** :

name=AuthorName&birth\_year=1980&nationality=French

# Implémentation des applications <API élève1> et <API élève2>

## Une descente de code de l'API REST 1

### Auteurs.java

* Contexte

La classe Auteurs est un modèle de données représentant un auteur dans le système. Elle est utilisée par le service REST1 pour gérer les informations relatives aux auteurs, comme leur nom, année de naissance et nationalité. Elle est également persistée dans une base de données relationnelle via JPA (Java Persistence API).

* Technologies utilisées
* **JPA (Java Persistence API)** : Permet de mapper cette classe Java à une table de base de données pour la persistance des données.
* **Lombok** : Utilisé pour générer automatiquement les méthodes getters et setters, ce qui simplifie le code.
* **Spring Boot** : Framework utilisé pour gérer la logique métier et la persistance des entités.
* Structure de la classe Auteurs

@Entity

@Getter

@Setter

@Table(name = "authors")

public class Auteurs {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

@Column(name = "birth\_year")

private Integer birthYear;

private String nationality;

}

* Annotations
* **@Entity** : Indique que la classe est une entité JPA et sera mappée à une table dans la base de données.
* **@Table(name = "authors")** : Spécifie le nom de la table dans la base de données à laquelle cette entité sera mappée. Ici, la table est nommée "authors".
* **@Id** : Désigne le champ id comme clé primaire de la table.
* **@GeneratedValue(strategy = GenerationType.IDENTITY)** : La stratégie de génération de l'identifiant de l'auteur. Ici, GenerationType.IDENTITY indique que la valeur de id est auto-incrémentée par la base de données.
* **@Column(name = "birth\_year")** : Spécifie le nom de la colonne birth\_year dans la table de base de données. Ceci permet de personnaliser le nom de la colonne dans la base de données tout en gardant un champ de classe en Java appelé birthYear.
* Champs de la classe
* **id (Integer)** : Identifiant unique de l'auteur. Il est généré automatiquement par la base de données.
* **name (String)** : Nom de l'auteur.
* **birthYear (Integer)** : Année de naissance de l'auteur. Ce champ est mappé à la colonne birth\_year dans la base de données.
* **nationality (String)** : Nationalité de l'auteur.
* Utilisation

La classe Auteurs est utilisée pour stocker, récupérer et manipuler les informations des auteurs dans la base de données. Grâce à l'annotation @Entity, elle permet une gestion automatique de la persistance via Spring Data JPA (ou une autre implémentation JPA).

Voici l'utilisation d’une classe dans un repository Spring Data :

package com.example.rest1.model;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface AuteursRepository extends JpaRepository<Auteurs, Integer> {

}

Cela permet d'effectuer des opérations courantes sur la table authors, comme ajouter, supprimer, ou rechercher des auteurs par nom, année de naissance, ou nationalité.

### Livres.java

* Contexte

La classe Livres représente un livre dans le système. Elle est utilisée par le service REST1 pour gérer les informations relatives aux livres, comme leur titre, année de publication, genre, et leur auteur. Elle est également persistée dans une base de données relationnelle via JPA.

* Technologies utilisées
* **JPA (Java Persistence API)** : Permet de mapper cette classe Java à une table de base de données pour la persistance des données.
* **Lombok** : Utilisé pour générer automatiquement les méthodes getters et setters, ce qui simplifie le code.
* **Spring Boot** : Framework utilisé pour gérer la logique métier et la persistance des entités.
* Structure de la classe Livres

@Entity

@Getter

@Setter

@Table(name = "books")

public class Livres {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String title;

@Column(name = "publication\_year")

private Integer publicationYear;

private String genre;

@ManyToOne

@JoinColumn(name = "author\_id")

private Auteurs auteur;

}

* Annotations
* **@Entity** : Indique que la classe est une entité JPA et sera mappée à une table dans la base de données.
* **@Table(name = "books")** : Spécifie le nom de la table dans la base de données à laquelle cette entité sera mappée. Ici, la table est nommée "books".
* **@Id** : Désigne le champ id comme clé primaire de la table.
* **@GeneratedValue(strategy = GenerationType.IDENTITY)** : La stratégie de génération de l'identifiant du livre. Ici, GenerationType.IDENTITY indique que la valeur de id est auto-incrémentée par la base de données.
* **@Column(name = "publication\_year")** : Spécifie que le champ publicationYear est mappé à la colonne publication\_year dans la table de la base de données.
* **@ManyToOne** : Indique une relation de type "many-to-one" entre un livre et un auteur. Un auteur peut être lié à plusieurs livres.
* **@JoinColumn(name = "author\_id")** : Spécifie le nom de la colonne dans la table books qui sert de clé étrangère pointant vers l'auteur. Ici, la colonne est appelée author\_id et elle fait référence à l'identifiant de l'auteur dans la table authors.
* Champs de la classe
* **id (Integer)** : Identifiant unique du livre. Il est généré automatiquement par la base de données.
* **title (String)** : Titre du livre.
* **publicationYear (Integer)** : Année de publication du livre. Ce champ est mappé à la colonne publication\_year dans la base de données.
* **genre (String)** : Genre du livre (par exemple, fiction, biographie, science-fiction, etc.).
* **auteur (Auteurs)** : L'auteur du livre. Ce champ fait référence à une instance de la classe Auteurs et utilise une relation de type "many-to-one", ce qui signifie qu'un auteur peut avoir plusieurs livres.
* Utilisation

La classe Livres est utilisée pour stocker, récupérer et manipuler les informations des livres dans la base de données. Grâce à l'annotation @Entity, elle permet une gestion automatique de la persistance via Spring Data JPA (ou une autre implémentation JPA).

Voici l'utilisation d’une classe dans un repository Spring Data :

package com.example.rest1.model;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface LivresRepository extends JpaRepository<Livres, Integer> {

}

Cela permet d'effectuer des opérations courantes sur la table books, comme ajouter, supprimer, ou rechercher des livres par titre, genre, année de publication ou auteur.

### Controller.java

* Contexte

La classe Controller agit en tant que contrôleur pour les opérations sur les livres et les auteurs dans ton application. Elle fournit des endpoints pour effectuer des actions sur la base de données, comme récupérer des livres, ajouter de nouveaux livres ou auteurs, et supprimer des livres. Cette classe est essentielle pour gérer les interactions entre l'utilisateur (ou d'autres services) et la base de données.

* Technologies utilisées
* **Spring Boot** : Framework utilisé pour créer des applications Java avec une architecture basée sur des microservices. Ici, il est utilisé pour gérer les contrôleurs REST.
* **JPA (Java Persistence API)** : Permet de gérer la persistance des objets Java dans une base de données relationnelle.
* **Spring Data JPA** : Fournit une abstraction pour interagir avec les données en utilisant des repositories et les entités.
* **ResponseEntity** : Permet de personnaliser les réponses HTTP avec des codes de statut, des en-têtes et des corps de réponse.
* Structure de la classe Controller

@RestController

public class Controller {

@Autowired

private LivresRepository livresRepository;

@Autowired

private AuteursRepository auteursRepository;

// Handler pour GET

@GetMapping("/getLivres")

public ResponseEntity<List<Livres>> getLivres() { ... }

@GetMapping("/getLivre")

public ResponseEntity<Livres> getLivre(@RequestParam Integer id) { ... }

// Handler pour POST Livre

@PostMapping("/addLivre")

public ResponseEntity<String> addNewLivre(

@RequestParam String title,

@RequestParam String genre,

@RequestParam Integer publication\_year,

@RequestParam Integer author\_id) { ... }

@PostMapping("/addAuteur")

public ResponseEntity<String> addNewAuteur(

@RequestParam String name,

@RequestParam Integer birth\_year,

@RequestParam String nationality) { ... }

@DeleteMapping("/deleteLivre/{id}")

public ResponseEntity<String> deleteLivre(@PathVariable Integer id) { ... }

@GetMapping("/getAuteur")

public ResponseEntity<List<Auteurs>> getAllAuteurs() { ... }

@GetMapping("/getAuteurs")

public List<Auteurs> getAuteurs() { ... }

}

* Annotations utilisées
* **@RestController** : Indique que cette classe est un contrôleur REST qui répond aux requêtes HTTP. Elle est dédiée à la gestion des endpoints de l'application.
* **@Autowired** : Injecte les dépendances, comme les repositories, automatiquement.
* **@GetMapping** : Utilisé pour définir un endpoint GET qui récupère des informations (livres ou auteurs).
* **@PostMapping** : Utilisé pour définir un endpoint POST qui crée de nouveaux objets (livres ou auteurs).
* **@DeleteMapping** : Utilisé pour définir un endpoint DELETE pour supprimer des objets de la base de données.

#### Méthodes de la classe

* 1. GET /getLivres

Retourne la liste de tous les livres stockés dans la base de données.

* **Réponse** : Une liste de tous les livres sous forme d'une réponse HTTP 200 OK.
* **Exemple de retour** :

[

{"id": 1, "title": "Book Title", "publicationYear": 2022, "genre": "Fiction", "auteur": {"id": 1, "name": "Author Name", "birthYear": 1970, "nationality": "French"}},

{"id": 2, "title": "Another Book", "publicationYear": 2020, "genre": "Non-Fiction", "auteur": {"id": 2, "name": "Another Author", "birthYear": 1985, "nationality": "British"}}

]

* 2. GET /getLivre

Retourne un livre spécifique en fonction de son id.

* **Paramètre** : id (ID du livre).
* **Réponse** : Le livre correspondant ou une réponse HTTP 404 si le livre n'est pas trouvé.
* **Exemple de retour en cas de succès** :

{"id": 1, "title": "Book Title", "publicationYear": 2022, "genre": "Fiction", "auteur": {"id": 1, "name": "Author Name", "birthYear": 1970, "nationality": "French"}}

* **Exemple de retour en cas d'échec** :

{

"message": "Livre non trouvé"

}

* 3. POST /addLivre

Ajoute un nouveau livre à la base de données.

* **Paramètres** :
  + title : Le titre du livre.
  + genre : Le genre du livre.
  + publication\_year : L'année de publication.
  + author\_id : L'ID de l'auteur (doit correspondre à un auteur existant).
* **Réponse** : Confirme l'ajout du livre ou retourne une erreur si l'auteur n'est pas trouvé.
* **Exemple de retour en cas de succès** :

{

"message": "Livre ajouté avec succès : Book Title"

}

* 4. POST /addAuteur

Ajoute un nouvel auteur à la base de données.

* **Paramètres** :
  + name : Le nom de l'auteur.
  + birth\_year : L'année de naissance.
  + nationality : La nationalité de l'auteur.
* **Réponse** : Confirme l'ajout de l'auteur.
* **Exemple de retour** :

{

"message": "Auteur ajouté avec succès : Author Name"

}

* 5. DELETE /deleteLivre/{id}

Supprime un livre de la base de données.

* **Paramètre** : id (ID du livre à supprimer).
* **Réponse** : Confirme la suppression du livre ou retourne une erreur si le livre n'est pas trouvé.
* **Exemple de retour en cas de succès** :

{

"message": "Livre supprimé avec succès"

}

* **Exemple de retour en cas d'échec** :

{

"message": "Livre non trouvé"

}

* 6. GET /getAuteur

Retourne la liste de tous les auteurs.

* **Réponse** : Une liste de tous les auteurs stockés dans la base de données.
* **Exemple de retour** :

[

{"id": 1, "name": "Author Name", "birthYear": 1970, "nationality": "French"},

{"id": 2, "name": "Another Author", "birthYear": 1985, "nationality": "British"}

]

* 7. GET /getAuteurs

Retourne également la liste de tous les auteurs, mais sans le format ResponseEntity. Cette méthode retourne directement la liste des auteurs.

## Une descente de code de l'API REST 2

### User.java

* Contexte

La classe User est utilisée pour représenter un utilisateur dans le système. Elle contient des informations comme le nom d'utilisateur, le mot de passe hashé et le rôle de l'utilisateur (administrateur ou utilisateur standard). Cette classe est une entité JPA qui sera persistée dans la base de données.

* Technologies utilisées
* **JPA (Java Persistence API)** : Permet de mapper cette classe Java à une table relationnelle dans une base de données.
* **Lombok** : Utilisé pour générer automatiquement les méthodes getters et setters pour les champs de la classe.
* **Spring Data JPA** : Fournit des fonctionnalités supplémentaires pour gérer les entités en base de données.
* **EnumType.STRING** : Permet de persister un énumérateur en base de données sous forme de chaîne de caractères.
* Annotations utilisées
* **@Entity** : Spécifie que la classe est une entité JPA.
* **@Table(name = "users")** : Définit le nom de la table dans la base de données. Ici, la table s'appelle users.
* **@Id** : Indique que ce champ est la clé primaire de l'entité.
* **@GeneratedValue(strategy = GenerationType.IDENTITY)** : Spécifie que l'ID sera généré automatiquement par la base de données.
* **@Column** : Utilisé pour spécifier des détails supplémentaires sur la colonne correspondante dans la table de la base de données (ex : la longueur du champ, sa contrainte nullable, etc.).
* **@Enumerated(EnumType.STRING)** : Spécifie que l'énumération Role doit être stockée en tant que chaîne de caractères dans la base de données.
* Propriétés de la classe User

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

@Column(name = "username", nullable = false, unique = true, length = 50)

private String username;

@Column(name = "password\_hash", nullable = false, length = 255)

private String passwordHash;

@Enumerated(EnumType.STRING)

@Column(name = "role", nullable = false, columnDefinition = "ENUM('admin', 'user')")

private Role role;

* Enumération Role

public enum Role {

admin, user

}

* **Rôle admin** : Utilisé pour les utilisateurs ayant des privilèges d'administrateur dans l'application.
* **Rôle user** : Utilisé pour les utilisateurs ayant des privilèges standards.

### UserController.java

* Contexte

La classe UserController est un contrôleur REST qui gère les requêtes liées à l'authentification des utilisateurs, ainsi que leur inscription dans le système. Elle fournit deux principaux endpoints : un pour la connexion (/login) et un pour l'enregistrement (/register). Les informations des utilisateurs sont récupérées à partir du UserRepository, et les sessions HTTP sont utilisées pour gérer les connexions.

* Technologies utilisées
* **Spring Web** : Permet de créer des API REST avec Spring MVC.
* **Spring Data JPA** : Utilisé pour interagir avec la base de données via le repository UserRepository.
* **HttpSession** : Utilisé pour stocker temporairement les informations de l'utilisateur lors de la connexion.
* Annotations utilisées
* **@RestController** : Indique que cette classe est un contrôleur REST, prête à traiter des requêtes HTTP et à renvoyer des réponses au format JSON.
* **@RequestMapping("/session")** : Définit la racine de l'URL pour ce contrôleur. Tous les endpoints commenceront par /session.
* **@PostMapping** : Indique que la méthode doit traiter les requêtes HTTP POST.
* **@RequestParam** : Utilisé pour extraire les paramètres de la requête HTTP.
* **@RequestBody** : Utilisé pour lier les données JSON envoyées dans le corps de la requête à un objet Java.

#### Méthodes

* 1. login

@PostMapping("/login")

public ResponseEntity<String> login(

@RequestParam(required = false) String username,

@RequestParam(required = false) String password,

@RequestBody(required = false) UserDTO userDTO,

HttpSession session)

* **Description** : Cette méthode permet de connecter un utilisateur en vérifiant ses identifiants (nom d'utilisateur et mot de passe). Elle supporte à la fois les données envoyées via les paramètres de requête (formulaire) ou en format JSON.
* **Paramètres** :
  + username : Le nom d'utilisateur (optionnel, sera pris si les données sont envoyées en paramètre).
  + password : Le mot de passe (optionnel, sera pris si les données sont envoyées en paramètre).
  + userDTO : Un objet DTO (Data Transfer Object) contenant le nom d'utilisateur et le mot de passe en format JSON.
  + session : La session HTTP pour enregistrer les informations de l'utilisateur une fois connecté.
* **Processus** :
  + Récupération des données de connexion (username et password).
  + Vérification que les paramètres sont valides.
  + Recherche de l'utilisateur dans la base de données.
  + Vérification des identifiants.
  + Si l'authentification réussit, les informations de l'utilisateur sont stockées dans la session HTTP.
* **Réponses possibles** :
  + **200 OK** : Si la connexion est réussie, renvoie le rôle de l'utilisateur.
  + **400 Bad Request** : Si les paramètres username ou password sont manquants.
  + **401 Unauthorized** : Si les identifiants sont incorrects.
* 2. register

@PostMapping("/register")

public ResponseEntity<String> register(@RequestBody UserDTO userDTO)

* **Description** : Cette méthode permet d'enregistrer un nouvel utilisateur. Elle prend un objet UserDTO contenant le nom d'utilisateur, le mot de passe et le rôle de l'utilisateur.
* **Paramètres** :
  + userDTO : L'objet DTO contenant les informations d'enregistrement de l'utilisateur.
* **Processus** :
  + Vérification que le nom d'utilisateur et le mot de passe sont présents.
  + Vérification que le nom d'utilisateur est unique.
  + Vérification que le rôle est valide (admin ou user).
  + Création de l'utilisateur et sauvegarde dans la base de données.
* **Réponses possibles** :
  + **200 OK** : Si l'enregistrement réussit, un message de confirmation est renvoyé.
  + **400 Bad Request** : Si les champs obligatoires sont manquants, ou si le nom d'utilisateur est déjà utilisé, ou si le rôle est invalide.

### UserDTO.java

* Contexte

La classe UserDTO est un **Data Transfer Object (DTO)** qui contient les données nécessaires pour l'enregistrement ou la connexion d'un utilisateur dans le système. Elle est utilisée pour transférer les informations entre le client et le serveur, spécifiquement dans les requêtes de connexion (/login) et d'enregistrement (/register) du contrôleur UserController.

* Technologies utilisées
* **Java POJO** : La classe UserDTO est une simple POJO (Plain Old Java Object), qui contient des attributs avec leurs getters et setters. Elle est utilisée pour transférer des données entre la couche de présentation et la couche de service.
* Attributs
* **username** (String) : Le nom d'utilisateur de l'utilisateur. Il doit être unique dans le système.
* **password** (String) : Le mot de passe de l'utilisateur, en texte clair (avant traitement).
* **role** (String) : Le rôle de l'utilisateur, pouvant être admin ou user.
* Constructeurs
* **Constructeur par défaut** : Un constructeur sans paramètres, permettant de créer un objet UserDTO vide.
* **Constructeur paramétré** : Un constructeur avec des paramètres pour initialiser directement les attributs username, password, et role.

public UserDTO(String username, String password, String role) {

this.username = username;

this.password = password;

this.role = role;

}

* Méthodes
* **Getters et Setters** :
  + getUsername() et setUsername(String username) : Permettent de récupérer et de définir le nom d'utilisateur.
  + getPassword() et setPassword(String password) : Permettent de récupérer et de définir le mot de passe.
  + getRole() et setRole(String role) : Permettent de récupérer et de définir le rôle de l'utilisateur.

### UserRepository.java

* Contexte

UserRepository est un **repository** qui étend l'interface CrudRepository fournie par Spring Data JPA. Cette interface permet d'effectuer des opérations de base de données (CRUD : Create, Read, Update, Delete) sur l'entité User. Elle contient des méthodes pour interagir avec la table des utilisateurs dans la base de données.

* Technologies utilisées
* **Spring Data JPA** : Fournit des mécanismes pour accéder à la base de données de manière abstraite et simplifiée. L'interface CrudRepository permet de réaliser des opérations CRUD sans avoir à implémenter explicitement les méthodes.
* **Optional** : L'interface Optional est utilisée pour encapsuler une valeur qui pourrait être présente ou absente. Elle est utilisée dans la méthode findByUsername pour gérer les cas où un utilisateur n'est pas trouvé dans la base de données.
* Méthodes
* **findByUsername(String username)** : Cette méthode permet de trouver un utilisateur en fonction de son nom d'utilisateur (username). Elle retourne un Optional<User>, ce qui signifie que le résultat peut être soit un utilisateur trouvé, soit un utilisateur absent (dans ce cas, le Optional sera vide).

Optional<User> findByUsername(String username);

# Hébergement

En ce qui concerne l’hébergement, nous n’avons pas pu complétement le mettre en place. Lors des tests en local, le site était entièrement fonctionnel. Malheureusement, lors du déploiement du back-end, nous avons eu des erreurs. Le front-end est tout de même disponible à l’adresse suivante :

<https://khalils.emf-informatique.ch/133/>

Le principe du déploiement était le suivant :

**BACKEND**

Modification du code sur l’API Gateway :

@CrossOrigin(origins = { "https://khalils.emf-informatique.ch" }, allowCredentials = "true")

Et la config du CORS :

package com.example.marque\_paf.config;

import org.springframework.context.annotation.Bean;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

public class ConfigCORS {

    @Bean

    public WebMvcConfigurer corsConfigurer() {

        return new WebMvcConfigurer() {

            @Override

            public void addCorsMappings(CorsRegistry registry) {

                registry.addMapping("/\*\*")

                        .allowedOrigins("https://khalils.emf-informatique.ch")

                        .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")

                        .allowedHeaders("\*")

                        .allowCredentials(true)

                        .maxAge(3600);

            }

        };

    }

}

Ensuite mise en place de DOCKER sur le linux :

sudo apt-get update

sudo apt-get install docker

sudo apt-get install docker-compose-plugin

Téléchargement des fichiers backend :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Lancement du docker-compose

root@gr07-srvdock-bis-239:~/backend-133# docker compose up -d

[+] Running 13/13

✔ db2 Pulled 18.8s

✔ db1 Pulled 18.8s

✔ cea172a6e83b Pull complete 2.8s

✔ 6cfd9ff0e16b Pull complete 2.8s

✔ 6cdb1d882a76 Pull complete 2.8s

✔ 29c08134121b Pull complete 3.1s

✔ bcc2e96984e1 Pull complete 3.2s

✔ a5636d731401 Pull complete 3.2s

✔ fb0ff87df77d Pull complete 8.1s

✔ bb9c2ecc3352 Pull complete 8.1s

✔ fbde96d8818e Pull complete 17.0s

✔ 26bcb3921fdb Pull complete 17.0s

✔ 069dfe57230c Pull complete 17.0s

Compose can now delegate builds to bake for better performance.

To do so, set COMPOSE\_BAKE=true.

[+] Building 8.4s (20/20) FINISHED docker:default

=> [servicerest2 internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 403B 0.0s

=> [servicerest1 internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 403B 0.0s

=> [api-gateway internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 420B 0.0s

=> [api-gateway internal] load metadata for docker.io/library/openjdk:17-jdk-alpine 1.3s

=> [servicerest2 internal] load .dockerignore 0.0s

=> => transferring context: 383B 0.0s

=> [api-gateway internal] load .dockerignore 0.0s

=> => transferring context: 383B 0.0s

=> [servicerest1 internal] load .dockerignore 0.0s

=> => transferring context: 383B 0.0s

=> [servicerest1 internal] load build context 1.8s

=> => transferring context: 53.03MB 1.8s

=> [servicerest2 1/2] FROM docker.io/library/openjdk:17-jdk-alpine@sha256:4b6abae565492dbe9e7a894137c966a7485154238902f2f25e9dbd9784383d81 5.8s

=> => resolve docker.io/library/openjdk:17-jdk-alpine@sha256:4b6abae565492dbe9e7a894137c966a7485154238902f2f25e9dbd9784383d81 0.0s

=> => sha256:a996cdcc040704ec6badaf5fecf1e144c096e00231a29188596c784bcf858d05 951B / 951B 0.0s

=> => sha256:264c9bdce361556ba6e685e401662648358980c01151c3d977f0fdf77f7c26ab 3.48kB / 3.48kB 0.0s

=> => sha256:5843afab387455b37944e709ee8c78d7520df80f8d01cf7f861aae63beeddb6b 2.81MB / 2.81MB 0.6s

=> => sha256:53c9466125e464fed5626bde7b7a0f91aab09905f0a07e9ad4e930ae72e0fc63 928.44kB / 928.44kB 0.6s

=> => sha256:d8d715783b80cab158f5bf9726bcada5265c1624b64ca2bb46f42f94998d4662 186.80MB / 186.80MB 2.4s

=> => sha256:4b6abae565492dbe9e7a894137c966a7485154238902f2f25e9dbd9784383d81 319B / 319B 0.0s

=> => extracting sha256:5843afab387455b37944e709ee8c78d7520df80f8d01cf7f861aae63beeddb6b 0.5s

=> => extracting sha256:53c9466125e464fed5626bde7b7a0f91aab09905f0a07e9ad4e930ae72e0fc63 0.2s

=> => extracting sha256:d8d715783b80cab158f5bf9726bcada5265c1624b64ca2bb46f42f94998d4662 3.0s

=> [api-gateway internal] load build context 1.3s

=> => transferring context: 20.87MB 1.2s

=> [servicerest2 internal] load build context 1.1s

=> => transferring context: 53.14MB 0.9s

=> [servicerest1 2/2] COPY target/rest1-0.0.1-SNAPSHOT.jar rest1.jar 0.8s

=> [api-gateway 2/2] COPY target/marque-paf-0.0.1-SNAPSHOT.jar marquepaf.jar 0.8s

=> [servicerest2 2/2] COPY target/rest2-0.0.1-SNAPSHOT.jar rest2.jar 0.8s

=> [servicerest2] exporting to image 0.3s

=> => exporting layers 0.3s

=> => writing image sha256:6983baa1d2006a23f0fb805afcf62e2f20f109b373c2b0db7124a41f115d7692 0.0s

=> => naming to docker.io/library/backend-133-servicerest2 0.0s

=> [api-gateway] exporting to image 0.2s

=> => exporting layers 0.2s

=> => writing image sha256:e0a0f3708a6abe17c2ede5eeb30c2b901adff9aa3df55c7091a80edd46d168dd 0.0s

=> => naming to docker.io/library/backend-133-api-gateway 0.0s

=> [servicerest1] exporting to image 0.4s

=> => exporting layers 0.4s

=> => writing image sha256:00be9478f3c063633e24618f584a56f2b8b73fd7d72daa5748d0c300264f3edc 0.0s

=> => naming to docker.io/library/backend-133-servicerest1 0.0s

=> [api-gateway] resolving provenance for metadata file 0.0s

=> [servicerest2] resolving provenance for metadata file 0.0s

=> [servicerest1] resolving provenance for metadata file 0.0s

[+] Running 11/11

✔ api-gateway Built 0.0s

✔ servicerest1 Built 0.0s

✔ servicerest2 Built 0.0s

✔ Network library-network Created 0.1s

✔ Volume "backend-133\_db1-data" Created 0.0s

✔ Volume "backend-133\_db2-data" Created 0.0s

✔ Container backend-133-api-gateway-1 Started 0.5s

✔ Container backend-133-db1-1 Started 0.6s

✔ Container backend-133-db2-1 Started 0.6s

✔ Container backend-133-servicerest2-1 Started 1.2s

✔ Container backend-133-servicerest1-1 Started 1.0s

root@gr07-srvdock-bis-239:~/backend-133#

**FRONT END**

Modification du code des fichiers JS :

fetch("https://133.khalils.emf-informatique.ch/laRequete ")

Mise en ligne sur CPANEL des fichiers HTML / CSS / JAVASCRIPT avec un redirection vers le backend

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

# Outils, langages (versions, définitions, installations)

* Java (version 17 ou supérieure)
* **Définition** : Java est un langage de programmation orienté objet, utilisé pour les applications serveur et les applications mobiles.
* **Installation** : Télécharge et installe le JDK 17 à partir de [la page officielle d'Oracle](https://www.oracle.com/java/technologies/javase-jdk17-downloads.html) ou utilise un gestionnaire de paquets comme brew (macOS) ou apt (Ubuntu).

Installation sur Ubuntu :

sudo apt install openjdk-17-jdk

* Spring Boot (version 3.x)
* **Définition** : Spring Boot est un framework Java permettant de créer des applications Java avec une configuration minimale.
* **Installation** : Installe Spring Tool Suite (STS) ou utilise un IDE comme IntelliJ IDEA ou VS Code avec des plugins Spring.
* Pour ajouter Spring Boot à un projet, il faut utiliser le pom.xml pour Maven
* Maven (version 3.8 ou supérieure)
* **Définition** : Maven est un outil de gestion de projet pour Java, utilisé pour gérer les dépendances et automatiser les tâches de build.
* **Installation** : Télécharge et installe Maven à partir de [son site officiel](https://maven.apache.org/).

Installation sur Ubuntu :

sudo apt install maven

* Docker (version 20.x ou supérieure)
* **Définition** : Docker est une plateforme permettant de créer, déployer et gérer des conteneurs d'applications.
* **Installation** : Télécharge et installe Docker en suivant les instructions sur le site officiel de Docker.

Installation sur Ubuntu :

sudo apt install docker.io

* Node.js

**Définition** : Node.js est un environnement d'exécution JavaScript, souvent utilisé pour le développement côté serveur, mais aussi pour gérer des applications front-end avec des frameworks comme React ou Angular.

* **Installation** : Télécharge et installe Node.js à partir de son site officiel.

Installation sur Ubuntu :

sudo apt install nodejs

sudo apt install npm

# Tests de fonctionnement du projet

Nous avons tous simplement utilisé la commande :

mvn test

et voilà le résultat :

root@PC-Siwan:/home/module-133-classe-300231-MariusPochon/projet/apiGateway/marque-paf# mvn test

[INFO] Scanning for projects...

[INFO]

[INFO] -----------------------< com.example:marque-paf >-----------------------

[INFO] Building marque-paf 0.0.1-SNAPSHOT

[INFO] --------------------------------[ jar ]---------------------------------

[INFO]

[INFO] --- maven-resources-plugin:3.3.1:resources (default-resources) @ marque-paf ---

[INFO] Copying 1 resource from src/main/resources to target/classes

[INFO] Copying 0 resource from src/main/resources to target/classes

[INFO]

[INFO] --- maven-compiler-plugin:3.13.0:compile (default-compile) @ marque-paf ---

[INFO] Recompiling the module because of added or removed source files.

[INFO] Compiling 5 source files with javac [debug parameters release 17] to target/classes

[INFO]

[INFO] --- maven-resources-plugin:3.3.1:testResources (default-testResources) @ marque-paf ---

[INFO] skip non existing resourceDirectory /home/module-133-classe-300231-MariusPochon/projet/apiGateway/marque-paf/src/test/resources

[INFO]

[INFO] --- maven-compiler-plugin:3.13.0:testCompile (default-testCompile) @ marque-paf ---

[INFO] Recompiling the module because of changed dependency.

[INFO] Compiling 1 source file with javac [debug parameters release 17] to target/test-classes

[INFO]

[INFO] --- maven-surefire-plugin:3.5.2:test (default-test) @ marque-paf ---

[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider

[INFO]

[INFO] -------------------------------------------------------

[INFO] T E S T S

[INFO] -------------------------------------------------------

[INFO] Running com.example.marque\_paf.MarquePafApplicationTests

17:22:41.448 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -- Could not detect default configuration classes for test class [com.example.marque\_paf.MarquePafApplicationTests]: MarquePafApplicationTests does not declare any static, non-private, non-final, nested classes annotated with @Configuration.

17:22:41.667 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper -- Found @SpringBootConfiguration com.example.marque\_paf.MarquePafApplication for test class com.example.marque\_paf.MarquePafApplicationTests

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v3.4.4)

2025-04-28T17:22:43.289+02:00 INFO 12614 --- [marque-paf] [ main] c.e.m.MarquePafApplicationTests : Starting MarquePafApplicationTests using Java 17.0.12 with PID 12614 (started by root in /home/module-133-classe-300231-MariusPochon/projet/apiGateway/marque-paf)

2025-04-28T17:22:43.295+02:00 INFO 12614 --- [marque-paf] [ main] c.e.m.MarquePafApplicationTests : No active profile set, falling back to 1 default profile: "default"

2025-04-28T17:22:45.537+02:00 INFO 12614 --- [marque-paf] [ main] c.e.m.MarquePafApplicationTests : Started MarquePafApplicationTests in 3.43 seconds (process running for 5.207)

Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 6.278 s -- in com.example.marque\_paf.MarquePafApplicationTests

[INFO]

[INFO] Results:

[INFO]

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 13.109 s

[INFO] Finished at: 2025-04-28T17:22:47+02:00

[INFO] ------------------------------------------------------------------------

# Auto-évaluations et conclusions

## Auto-évaluation et conclusion de Siwan

**Auto-évaluation** : J'ai travaillé sur client 1, rest 1 et l'API Gateway. J'ai assuré la gestion des livres et des auteurs, et configuré l'API Gateway pour centraliser les requêtes. Cela m’a permis de renforcer mes compétences en gestion des API et de la communication entre services.

**Conclusion** : Le projet m’a permis de bien comprendre la gestion des microservices. Je suis satisfait du travail accompli, mais je vois des opportunités d'amélioration, notamment dans la gestion de la sécurité et des tests.

## Auto-évaluation et conclusion de Marius

**Auto-évaluation** : J'ai travaillé sur le client 2, rest 2 : j’ai géré l’authentification des utilisateurs et leur connexion. J’ai passé pas mal de temps sur l’Api Gateway aussi. Durant les vacances, j’ai beaucoup travaillé sur le code et Siwan sur le doc. Je me suis rendu compte que notre projet n’était pas réaliste alors je les revu et je recode une bonne partie. Au final, tout fonctionnait et, malgré des manques de certains concepts, je suis content de ce que nous avons réussi à faire.

**Conclusion** : Comme dis plus haut, j’ai passé beaucoup de temps durant les vacances sur le projet. Ce temps ma permis de bien comprendre le code : au début, je m’aidais beaucoup de ChatGPT sans connaitre le code. Cette façon de faire est illusoire et irréalisable sur le long therme. On se perds, on ne comprend pas les erreurs, etc.  
**mais durant les vacances, j’ai commencé à bien comprendre le code et j’ai compris une des erreurs que nous avions faites :**

**Le client 1 s’occupait des livres et le client 2 uniquement des logins.**

**Au final, le projet n’avait pas de sens et la gestion des utilisateurs était (quasi) impossible (dure de partager des informations sur 2 front end différent) alors j’ai décidé de revoir le projet.**