



# Rapport de projet J2EE

3-ème année du cycle ingénieur

Filière ingénierie informatique et réseaux.

**Sous le thème :**

---

Système de Gestion des  
Bibliothèques Universitaires

**Réalisé par :**

BENDADDA Amine

SADIK Oumaima

**Encadré par :**

Pr. BENYOUSSUF Marwa

# Table des Matières

Chapitre 1 : contexte générale du projet.....	6
Introduction .....	7
1.Cadre général du projet.....	7
2.Périmètre du projet .....	8
2.1 Problématique .....	8
2.2 But.....	8
2.3 Missions .....	9
Conclusion .....	10
Chapitre 2 : Conception et analyse .....	11
Introduction .....	12
1.Cahier des charges .....	12
2.L'étude fonctionnelle.....	12
2.1 Objectifs fonctionnels .....	12
2.2 Spécifications des besoins fonctionnels.....	14
2.3 Spécification des besoins non fonctionnels .....	17
3.Conception.....	18
3.1 Acteurs (Diagramme d'organisation des acteurs) .....	18
3.2 Vue fonctionnelles (Diagramme des cas d'utilisation) .....	19
Conclusion.....	23
Chapitre 3 : Etude technique .....	24
Introduction .....	25
1.Architecture applicative.....	25
1.1 Description de l'architecture .....	25
1.2 Interaction entre les couches.....	26
2.Architecture technique.....	27
2.1 Structure technique globale.....	27
3.Outils et Framework utilises.....	28
3.1 Backend.....	29
3.2 Frontend .....	29
3.3 Base de données .....	30
3.4 Conteneurisation et Orchestration.....	30
3.5 Environnements de Développement et de Gestion .....	30

Conclusion.....	31
Chapitre 4 : Réalisation .....	32
Introduction .....	33
1.Configuration des Clients dans Keycloak .....	33
2.Configuration des Microservices dans Eureka .....	34
3.Validation d'un Token JWT sur jwt.io.....	34
4.Tests des Microservices.....	36
4.1 Test du Module : Gestion des Livres .....	36
4.2 Test du Module : Gestion des Emprunts.....	45
Conclusion.....	61
Annexes .....	62

# Liste des figures

<u>Figure 1: Diagramme d'organisation des acteurs</u> .....	16
<u>Figure 2: Diagrammes des cas d'utilisation du module gestion des comptes</u> .....	17
<u>Figure 3 : Diagrammes des cas d'utilisation du module gestion des livres</u> .....	18
<u>Figure 4 : Diagrammes des cas d'utilisation du module gestion des emprunts</u> .....	19
<u>Figure 5: Diagrammes des cas d'utilisation du module gestion des réservations</u> .....	20
<u>Figure 6: Liste des clients configurés dans Keycloak</u> .....	31
<u>Figure 7: Services enregistrés dans Eureka</u> .....	32
<u>Figure 8: Décodage et validation d'un token JWT</u> .....	33
<u>Figure 9: Test de l'API Get All Categories dans le module Gestion des Livres</u> .....	34
<u>Figure 10: Test de l'API Add Category dans le module Gestion des Livres</u> .....	35
<u>Figure 11: Test de l'API Update Category dans le module Gestion des Livres</u> .....	36
<u>Figure 12: Test de l'API Delete Category dans le module Gestion des Livres</u> .....	37
<u>Figure 13: Test de l'API Get All Livres dans le module Gestion des Livres</u> .....	38
<u>Figure 14: Test de l'API Get Livre par ID dans le module Gestion des Livres</u> .....	39
<u>Figure 15: Test de l'API Add Livre dans le module Gestion des Livres</u> .....	40
<u>Figure 16: Test de l'API Update Livre dans le module Gestion des Livres</u> .....	41
<u>Figure 17: Test de l'API Delete Livre dans le module Gestion des Livres</u> .....	42
<u>Figure 18: Test de l'API Créer un Emprunt dans le module Gestion des Emprunts</u> .....	43
<u>Figure 19: Test de l'API Consulter Mes Emprunts dans le module Gestion des Emprunts</u> .....	44
<u>Figure 20: Test de l'API Consulter Tous les Emprunts dans le module Gestion des Emprunts</u> .....	45
<u>Figure 21: Test de l'API Mise à Jour de la Pénalité dans le module Gestion des Emprunts</u> .....	46
<u>Figure 22: Test de l'API Consultation des Emprunts en Retard dans le module Gestion des Emprunts</u> .....	47
<u>Figure 23: Test de l'API Modification de la Date de Retour dans le module Gestion des Emprunts</u> ...	48
<u>Figure 24: Test de l'API Retourner un Livre dans le module Gestion des Emprunts</u> .....	49
<u>Figure 25: Test de l'API Réserver un Livre dans le module Gestion des Réservations</u> .....	50
<u>Figure 26: Test de l'API Afficher Mes Réservations dans le module Gestion des Réservations</u> .....	51
<u>Figure 27: Test de l'API Annuler une Réservation dans le module Gestion des Réservations</u> .....	52
<u>Figure 28: Test de l'API Afficher Toutes les Réservations dans le module Gestion des Réservations</u> .....	53
<u>Figure 29: Test de l'API Afficher les Livres Réservés dans le module Gestion des Réservations</u> .....	54
<u>Figure 30: Interface Utilisateur - Page de Connexion</u> .....	55
<u>Figure 31: Interface Utilisateur - Tableau de Bord Étudiant</u> .....	55
<u>Figure 32: Interface Utilisateur - Page de Recherche des Livres</u> .....	56
<u>Figure 33: Interface Utilisateur - Page des Détails d'un Livre</u> .....	56
<u>Figure 34: Interface Utilisateur - Mes Réservations</u> .....	57
<u>Figure 35: Interface Utilisateur - Tableau de Bord Administrateur</u> .....	58
<u>Figure 36: Interface Utilisateur - Ajouter un nouveau livre</u> .....	58

# Liste des tableaux

<u>Tableau 1: Les fonctionnalités par utilisateur du module gestion des comptes</u> .....	13
<u>Tableau 2 : Les règles de gestion du module gestion des comptes</u> .....	13
<u>Tableau 3: Les fonctionnalités par utilisateur du module gestion des livres</u> .....	14
<u>Tableau 4: Les règles de gestion du module gestion des livres</u> .....	14
<u>Tableau 5: Les fonctionnalités par utilisateur du module gestion des emprunts</u> .....	15
<u>Tableau 6: Les règles de gestion du module gestion des emprunts</u> .....	15
<u>Tableau 7: Les règles de gestion du module gestion des réservations</u> .....	15
<u>Tableau 8: Descriptions des cas d'utilisation du module gestion des comptes</u> .....	18
<u>Tableau 9: Descriptions des cas d'utilisation du module gestion des livres</u> .....	19
<u>Tableau 10: Descriptions des cas d'utilisation du module gestion des emprunts</u> .....	20
<u>Tableau 11 : Descriptions des cas d'utilisation du module gestion des réservations</u> .....	21

## **Chapitre 1 : contexte générale du projet**

# Introduction

Ce chapitre introductif présente notre projet ambitieux de développer un système de gestion des bibliothèques universitaires, conçu pour moderniser et automatiser les processus de gestion des ressources documentaires et des interactions utilisateurs. L'objectif principal est de répondre aux besoins croissants d'efficacité, de flexibilité et de sécurité dans la gestion des emprunts, des réservations et des notifications. Ce chapitre pose les bases nécessaires pour comprendre le cadre général du projet, en définissant les défis, les opportunités, ainsi que les missions essentielles à sa mise en œuvre réussie.

## 1. Cadre général du projet

Notre projet de conception d'un système de gestion des bibliothèques universitaires s'inscrit dans un contexte où les avancées technologiques redéfinissent les méthodes d'accès aux ressources documentaires et les interactions entre les utilisateurs et les bibliothèques. À l'heure où les étudiants, enseignants et chercheurs attendent des services rapides, fiables et accessibles en ligne, il devient impératif pour les institutions universitaires d'adopter des solutions modernes et performantes pour gérer leurs bibliothèques.

Le système proposé vise à répondre à plusieurs problématiques rencontrées dans les bibliothèques traditionnelles, telles que la difficulté à localiser les livres disponibles, le suivi manuel des emprunts et retours, ou encore l'absence d'un mécanisme efficace pour notifier les utilisateurs en cas de retard. Grâce à l'intégration de technologies comme les microservices, les notifications automatisées, et une interface utilisateur intuitive, ce projet offre une plateforme centralisée et sécurisée pour une gestion optimisée des ressources.

Dans ce cadre, notre objectif est de fournir une solution interactive qui simplifie l'expérience des étudiants et des administrateurs. Les étudiants pourront rechercher et réserver des livres, suivre leurs emprunts en cours, et recevoir des rappels personnalisés pour éviter des retards. Les administrateurs disposeront d'outils pour gérer efficacement les collections, les utilisateurs et les statistiques d'utilisation.

En adoptant une approche technologique robuste, le système s'inscrit dans une dynamique d'innovation et de transformation numérique des services universitaires. Dans les sections suivantes, nous détaillerons les objectifs spécifiques, les défis à relever, ainsi que les étapes

nécessaires à la réalisation de ce projet, qui vise à répondre aux besoins évolutifs des utilisateurs et des institutions.

## 2. Périmètre du projet

### 2.1 Problématique

La gestion des bibliothèques universitaires repose encore largement sur des processus traditionnels, ce qui peut entraîner plusieurs difficultés. Les étudiants et les chercheurs rencontrent souvent des problèmes pour localiser rapidement les ressources disponibles, effectuer des réservations ou suivre leurs emprunts. De leur côté, les administrateurs sont confrontés à des défis dans la gestion des collections, le suivi des emprunts, et la notification des utilisateurs en cas de retards ou d'indisponibilités.

La problématique centrale est donc de concevoir un système de gestion des bibliothèques modernes et centralisé, capable d'automatiser les tâches clés, d'optimiser la gestion des ressources, et d'offrir une expérience utilisateur fluide. Ce système doit répondre aux attentes des utilisateurs tout en s'adaptant aux besoins administratifs, le tout dans un cadre sécurisé, robuste et évolutif.

### 2.2 But

Le but du projet est de développer un système de gestion des bibliothèques universitaires qui automatise et simplifie les processus liés à la gestion des livres, des utilisateurs, des emprunts et des réservations. Cette plateforme vise à fournir une interface intuitive et conviviale pour les étudiants et les administrateurs, en intégrant des fonctionnalités avancées telles que la recherche en temps réel, la gestion des emprunts et des retards, ainsi que les réservations.

Le système cible couvre les modules fonctionnels suivants :

- **Module 1** : Gestion des livres (catalogage, disponibilité, mise à jour des informations)
- **Module 2** : Gestion des emprunts et des réservations (suivi des emprunts, rappels automatisés)
- **Module 3** : Gestion des utilisateurs (profils, rôles et autorisations)
- **Module 4** : Gestion des réservations (suivi des réservations, réserver des livres emprunté)

## 2.3 Missions

### Étude fonctionnelle

L'étude fonctionnelle consiste à identifier et analyser les besoins des utilisateurs, qu'il s'agisse des étudiants ou des administrateurs. Cette phase permet de définir les fonctionnalités clés du système, telles que la recherche et la réservation de livres, la gestion des retards, et l'envoi de notifications automatisées. Les besoins fonctionnels et non fonctionnels sont étudiés pour garantir une expérience utilisateur fluide, intuitive et sécurisée. Une attention particulière est portée à la robustesse du système et à l'interopérabilité entre les différents modules, comme la gestion des emprunts et des utilisateurs.

### Étude technique

L'étude technique vise à établir les fondations technologiques du projet en sélectionnant les outils et frameworks appropriés :

- **Backend** : Microservices avec Spring Boot et Spring Cloud (Eureka, Gateway).
- **Frontend** : Angular pour une interface utilisateur moderne et interactive.
- **Base de données** : MySQL pour la gestion des données relationnelles.
- **Sécurité** : Keycloak pour l'authentification et l'autorisation via JWT. Cette phase inclut également la conception d'une architecture évolutive pour permettre la scalabilité du système.

### Conception

La phase de conception se concentre sur la modélisation des fonctionnalités et des processus à l'aide de diagrammes UML, comme les cas d'utilisation et les séquences. Les interactions entre les utilisateurs et le système, ainsi que les flux de données entre les microservices, sont détaillés pour assurer une cohérence fonctionnelle et technique. Des maquettes d'interface sont également créées pour représenter les écrans de l'application, garantissant simplicité et ergonomie.

### Réalisation

La réalisation constitue la phase de développement du projet, où les différents modules fonctionnels prennent forme :

1. **Développement des microservices** pour gérer les livres, les emprunts, les utilisateurs et les réservations.
2. **Intégration de l'interface utilisateur Angular** pour permettre une navigation intuitive et fluide.
3. **Mise en œuvre des tâches automatisées avec Spring Batch**, comme les rappels d'emprunts en retard.
4. **Test et déploiement** pour valider la conformité du système et le rendre opérationnel dans un environnement de production.

## Conclusion

Ce chapitre a établi les bases du projet de gestion des bibliothèques universitaires en explorant son contexte, son périmètre et ses missions. Il met en lumière les problématiques liées à la gestion des livres, des utilisateurs et des réservations, tout en présentant les objectifs visant à offrir une solution moderne et centralisée. Avec des modules fonctionnels dédiés à la gestion des livres, des réservations et des utilisateurs, le projet répond aux besoins des utilisateurs en matière de simplicité et d'efficacité. Ces fondations seront approfondies dans les sections suivantes pour détailler la conception et la réalisation de la solution.

## **Chapitre 2 : Conception et analyse**

# **Introduction**

Ce chapitre se concentre sur la conception et l'analyse du projet de gestion des bibliothèques universitaires. Il a pour objectif de définir et structurer les aspects essentiels à la réalisation du système. À travers le cahier des charges, nous identifierons les besoins fonctionnels et non fonctionnels afin de garantir une solution conforme aux attentes des utilisateurs et aux exigences techniques. L'étude fonctionnelle précisera les objectifs, les règles de gestion et les spécifications des blocs fonctionnels dédiés à la gestion des livres, des réservations et des comptes utilisateurs. Enfin, la phase de conception proposera une vue fonctionnelle à l'aide de diagrammes de cas d'utilisation, illustrant les interactions entre les différents acteurs et le système. Ce chapitre constitue une étape clé pour établir les fondations solides du développement du projet.

## **1. Cahier des charges**

Le cahier des charges joue un rôle fondamental dans la réalisation de notre projet de système de gestion des bibliothèques universitaires. Il définit les besoins et les objectifs du projet, tout en prenant en compte les contraintes spécifiques à notre contexte. Cette section présentera une analyse détaillée des besoins fonctionnels et non fonctionnels du système, en mettant en lumière les fonctionnalités essentielles à implémenter pour garantir une expérience utilisateur optimale.

Elle abordera également les contraintes liées aux choix technologiques, aux performances attendues et à la sécurité des données. En somme, cette partie constitue une base solide pour orienter la conception et le développement de notre système, tout en assurant qu'il répond efficacement aux attentes des utilisateurs finaux et des administrateurs.

## **2. L'étude fonctionnelle**

### **2.1 Objectifs fonctionnels**

Les objectifs fonctionnels du système de gestion des bibliothèques universitaires sont conçus pour répondre aux besoins des utilisateurs tout en garantissant une expérience fluide et intuitive. Ces objectifs se répartissent en quatre modules principaux : gestion des livres, gestion des emprunts, gestion des réservations (pour les livres déjà empruntés) et gestion des comptes utilisateurs.

#### **Gestion des livres :**

- Permettre aux utilisateurs de parcourir le catalogue des livres disponibles.

- Fournir une recherche avancée par titre, auteur, catégorie ou mots-clés.
- Afficher les détails d'un livre, tels que le titre, l'auteur, l'édition, la catégorie et sa disponibilité.
- Autoriser les administrateurs à ajouter, modifier ou supprimer des livres du catalogue.

#### **Gestion des emprunts :**

- Offrir une fonctionnalité permettant aux utilisateurs d'emprunter des livres disponibles.
- Assurer un suivi des emprunts en cours, avec les dates d'emprunt et de retour prévues.
- Calculer automatiquement les retards pour les livres non retournés dans les délais impartis, avec des indications claires pour l'utilisateur.
- Permettre aux administrateurs de consulter l'historique des emprunts, de gérer les retards et de mettre en place des pénalités ou des rappels.

#### **Gestion des réservations :**

- Permettre aux utilisateurs de réserver des livres qui sont actuellement empruntés par d'autres.
- Informer les utilisateurs dès qu'un livre réservé devient disponible.
- Permettre aux utilisateurs d'annuler une réservation en cas de besoin.
- Offrir aux administrateurs une vue centralisée des réservations pour gérer les priorités et éviter les conflits.

#### **Gestion des comptes utilisateurs :**

- Faciliter l'inscription et la connexion des utilisateurs avec des mécanismes d'authentification sécurisés.
- Permettre aux utilisateurs de mettre à jour leurs informations personnelles, comme le nom, l'email et le mot de passe.
- Offrir aux administrateurs la possibilité de gérer les rôles des utilisateurs (ex. : étudiant, administrateur).

- Inclure une fonctionnalité permettant aux utilisateurs de désactiver ou supprimer leur compte.

Ces objectifs fonctionnels garantissent une plateforme robuste, efficace et adaptée aux besoins des utilisateurs, tout en introduisant des mécanismes automatisés comme le calcul des retards pour simplifier la gestion des emprunts.

## 2.2 Spécifications des besoins fonctionnels

### 2.2.1 Blocs fonctionnels : Gestion des comptes

- **Fonctionnalités par utilisateur**

Utilisateur	Fonctionnalités
Visiteur	<p>Créer un compte utilisateur en fournissant des informations personnelles.</p> <p>Se connecter avec une adresse email et un mot de passe.</p>
Utilisateur authentifié	<p>Mettre à jour ses informations personnelles (nom, email, mot de passe).</p> <p>Consulter son profil utilisateur.</p>
Administrateur	<p>Gérer les comptes utilisateurs (activer, désactiver, ou supprimer un compte).</p> <p>Assigner des rôles (étudiant, enseignant, ou administrateur).</p>

Tableau 1: Les fonctionnalités par utilisateur du module gestion des comptes

- **Règles de gestion**

Règle de gestion	Description
RG01	Chaque utilisateur doit disposer d'une adresse email unique pour créer un compte.
RG02	Les mots de passe doivent respecter un format sécurisé (minimum 8 caractères, avec chiffres et lettres).
RG03	Les administrateurs ne peuvent pas supprimer leur propre compte.
RG04	Un utilisateur désactivé ne peut plus accéder au système jusqu'à réactivation par un administrateur.
RG05	Seuls les administrateurs peuvent modifier les rôles des utilisateurs.

Tableau 2 : Les règles de gestion du module gestion des comptes

## 2.2.2 Bloc Fonctionnels : Gestion des livres

- **Fonctionnalités par utilisateur**

Utilisateur	Fonctionnalités
Utilisateur authentifié	Consulter la liste des livres disponibles dans le catalogue.
	Rechercher des livres par titre, auteur, catégorie ou mots-clés.
	Consulter les détails d'un livre (titre, auteur, édition, catégorie, disponibilité).
Administrateur	Ajouter un nouveau livre au catalogue en renseignant les informations nécessaires.
	Modifier les informations d'un livre existant (titre, auteur, disponibilité, etc.).
	Supprimer un livre du catalogue.

Tableau 3: Les fonctionnalités par utilisateur du module gestion des livres

- **Règles de gestion**

Règle de gestion	Description
RG07	Chaque livre doit contenir un titre, un auteur, une catégorie, et un statut de disponibilité.
RG08	Un livre ne peut être supprimé du catalogue s'il est associé à un emprunt ou à une réservation en cours.
RG09	La recherche de livres doit supporter des critères multiples (titre, auteur, catégorie).
RG10	Seuls les administrateurs peuvent modifier ou supprimer les informations d'un livre.

RG11	Les informations d'un livre modifié doivent être mises à jour en temps réel pour tous les utilisateurs.
------	---------------------------------------------------------------------------------------------------------

Tableau 4: Les règles de gestion du module gestion des livres

### 2.2.3 Bloc Fonctionnels : Gestion des emprunts

- **Fonctionnalités par utilisateur**

Utilisateur	Fonctionnalités
Utilisateur authentifié	Emprunter un livre disponible en ligne ou via un point de retrait.
	Consulter la liste des emprunts en cours avec les dates d'échéance.
	Consulter l'historique des emprunts passés.
Administrateur	Suivre les emprunts en cours pour tous les utilisateurs.
	Visualiser les retards et appliquer des pénalités en fonction des règles définies.
	Prolonger la durée d'un emprunt sous certaines conditions

Tableau 5: Les fonctionnalités par utilisateur du module gestion des emprunts

- **Règles de gestion**

Règle de gestion	Description
RG12	Un livre ne peut être emprunté que s'il est marqué comme "disponible".
RG13	Le calcul des retards doit être automatique et basé sur la date d'échéance définie lors de l'emprunt.
RG14	Les pénalités pour retard doivent être enregistrées dans le compte utilisateur et accessibles pour l'administration.

**RG15**

Les emprunts prolongés doivent être approuvés par l'administrateur si le livre est déjà réservé par un autre utilisateur.

Tableau 6: Les règles de gestion du module gestion des emprunts

#### 2.2.4 Bloc Fonctionnels : Gestion des réservations

- Fonctionnalités par utilisateur**

Utilisateur	Fonctionnalités
Utilisateur authentifié	Réserver un livre déjà emprunté en s'ajoutant à une file d'attente.
	Consulter la liste de ses réservations en cours avec les dates estimées de disponibilité.
	Annuler une réservation en cas de changement de besoin.
Administrateur	Visualiser toutes les réservations en cours pour tous les livres et utilisateurs.
	Visualiser tous les livres en réservation.

Tableau 7: Les règles de gestion du module gestion des réservations

### 2.3 Spécification des besoins non fonctionnels

Les besoins non fonctionnels définissent les caractéristiques qualitatives du système, garantissant son bon fonctionnement, sa sécurité, sa performance, et son évolutivité.

#### Sécurité

- Les données des utilisateurs, notamment les informations personnelles et les emprunts, doivent être protégées contre tout accès non autorisé.
- Les mots de passe des utilisateurs doivent être stockés de manière sécurisée avec un algorithme de hachage (ex. : BCrypt).
- Les communications entre les différents services et les utilisateurs doivent être sécurisées via HTTPS et des tokens JWT.

#### Performance

- Le temps de réponse pour rechercher un livre, consulter ses détails ou effectuer une réservation doit être inférieur à 2 secondes dans des conditions normales.

- Le système doit supporter un grand nombre d'utilisateurs simultanés, avec un minimum de dégradation des performances, notamment en période de pointe (ex. : en début de semestre universitaire).
- Les calculs, comme ceux des retards ou de la disponibilité des livres, doivent être traités rapidement pour éviter des ralentissements.

## **Disponibilité**

- Le système doit être accessible 24 heures sur 24, 7 jours sur 7, pour les étudiants et les administrateurs.
- Les interruptions pour maintenance planifiée ou mises à jour doivent être limitées à des créneaux prédefinis et communiqués aux utilisateurs.
- Les indisponibilités imprévues doivent être minimisées grâce à des mécanismes de surveillance et des solutions de récupération en cas de panne.

## **Extensibilité**

- Le système doit être conçu pour intégrer facilement de nouveaux modules ou fonctionnalités (ex. : ajout de nouvelles catégories de livres ou d'emprunts numériques).
- L'architecture microservices doit permettre une montée en charge sans modification significative de l'infrastructure.

## **Convivialité**

- L'interface utilisateur doit être intuitive, avec des fonctionnalités claires pour les étudiants et les administrateurs.
- Les messages d'erreur doivent être informatifs et guider les utilisateurs pour résoudre les éventuels problèmes (ex. : "Le mot de passe doit contenir au moins 8 caractères").
- Une aide en ligne ou une documentation utilisateur doit être disponible pour faciliter la prise en main du système.

## **3. Conception**

UML (Unified Modeling Language, que l'on peut traduire par « langage de modélisation unifié ») est une notation permettant de modéliser un problème de façon standard. Ce langage est né de la fusion de plusieurs méthodes existantes auparavant, et est devenu désormais la

référence en termes de modélisation objet. UML est destiné à faciliter la conception des documents nécessaires au développement d'un logiciel orienté objet, comme standard de modélisation de l'architecture logicielle.

### 3.1 Acteurs (Diagramme d'organisation des acteurs)

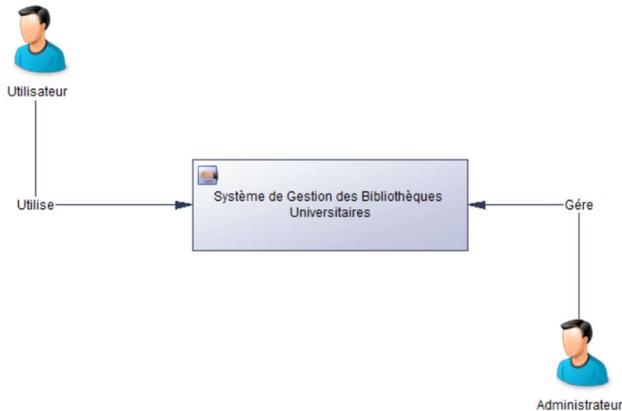


Figure 1: Diagramme d'organisation des acteurs

#### Description :

Le diagramme d'organisation des acteurs du système de gestion des bibliothèques universitaires illustre les différentes entités interagissant avec le système et leurs relations respectives. Il met en évidence deux rôles clés : les utilisateurs (étudiants et enseignants) qui utilisent le système pour emprunter, réserver ou consulter des livres, et les administrateurs, qui gèrent les différentes fonctionnalités du système, notamment la gestion des utilisateurs, des livres, des emprunts et des réservations. Ce diagramme offre une vue d'ensemble claire des responsabilités de chaque acteur et des interactions avec le système, servant ainsi de base pour mieux comprendre les besoins et attentes des utilisateurs dans le cadre de ce projet.

## 3.2 Vue fonctionnelles (Diagramme des cas d'utilisation)

### 3.2.1 Blocs fonctionnels : Gestion des comptes

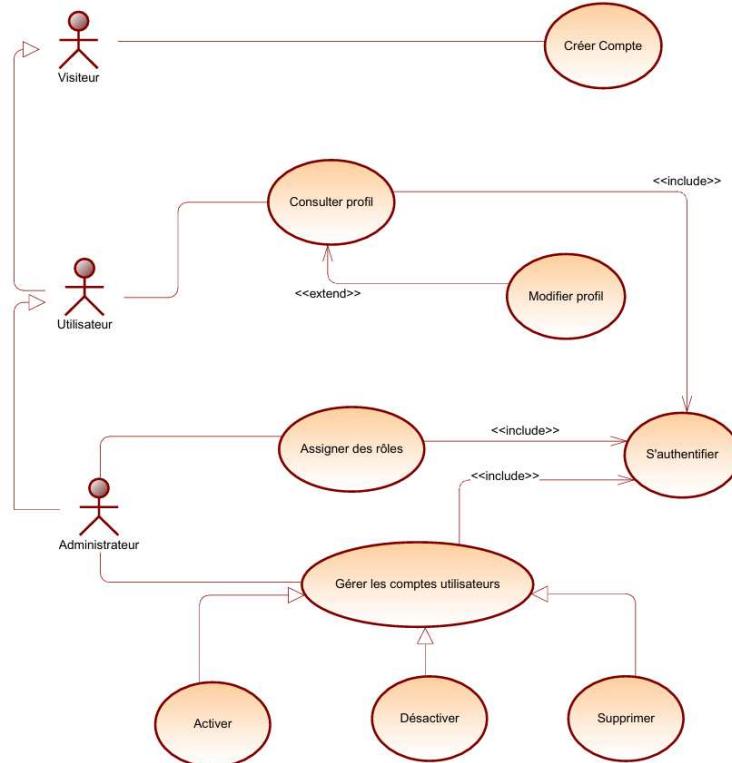


Figure 2: Diagrammes des cas d'utilisation du module gestion des comptes

Code	Libellé	Acteur	Description
V-01	Créer Compte	Visiteur	Permet au visiteur de créer un compte sur la plateforme en fournissant ses informations personnelles.
USR-01	Consulter Profil	Utilisateur	Permet à l'utilisateur connecté de consulter son profil, affichant ses informations personnelles.
USR-02	Modifier Profil	Utilisateur	Permet à l'utilisateur de modifier les informations de son profil, telles que le nom ou l'email.
USR-03	S'authentifier	Utilisateur	Permet au visiteur de s'authentifier pour accéder aux fonctionnalités réservées aux utilisateurs connectés.
ADM-01	Gérer Comptes Utilisateurs	Administrateur	Permet à l'administrateur de gérer les comptes des utilisateurs, notamment leur activation, désactivation ou suppression.
ADM-02	Assigner des rôles	Administrateur	Permet à l'administrateur d'assigner ou de modifier les rôles des utilisateurs (étudiant, enseignant, etc.).

Tableau 8: Descriptions des cas d'utilisation du module gestion des comptes

### 3.2.2 Blocs fonctionnels : Gestion des livres

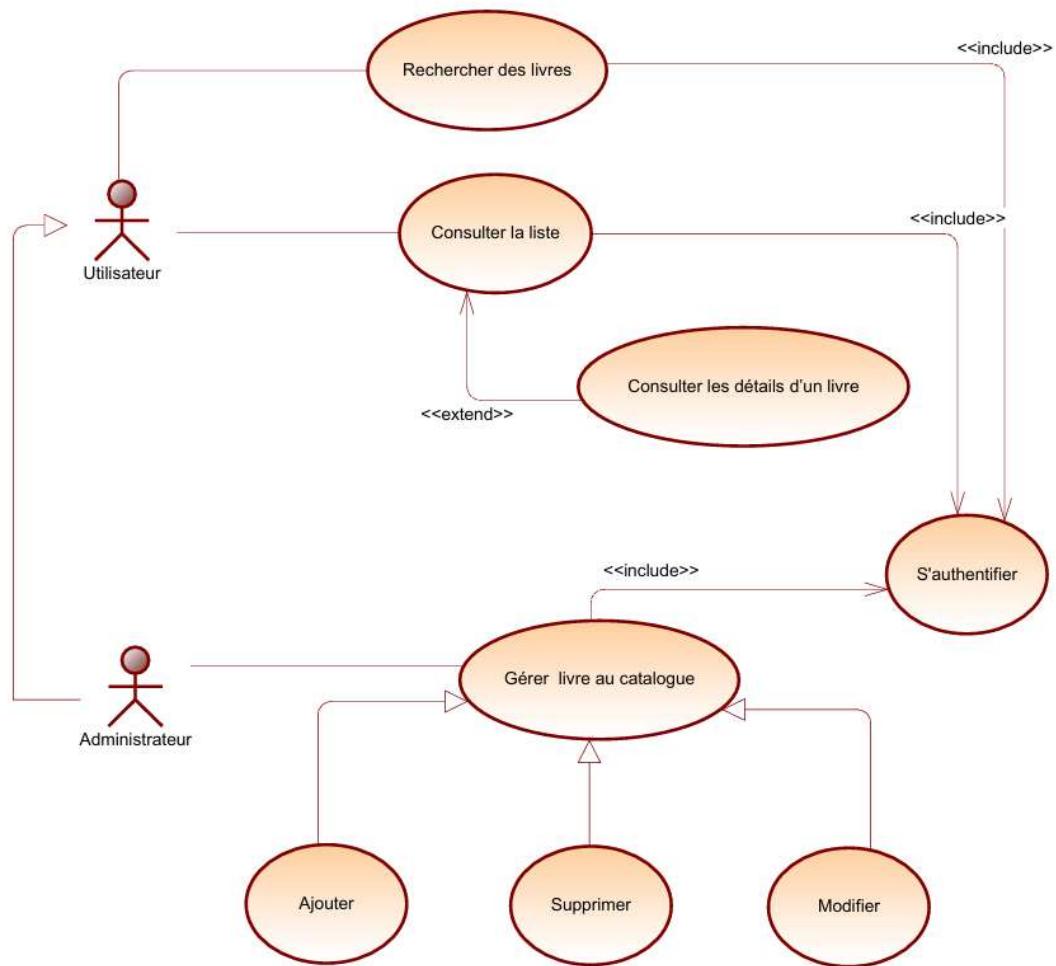


Figure 3 : Diagrammes des cas d'utilisation du module gestion des livres

Code	Libellé	Acteur	Description
<b>USR-04</b>	Rechercher des livres	Utilisateur	Permet à l'utilisateur de rechercher des livres dans le catalogue par titre, auteur, ou catégorie.
<b>USR-05</b>	Consulter la liste	Utilisateur	Permet à l'utilisateur de visualiser la liste des livres disponibles selon les résultats de recherche.
<b>USR-06</b>	Consulter les détails d'un livre	Utilisateur	Permet à l'utilisateur de consulter les informations détaillées d'un livre (titre, auteur, disponibilité, etc.).
<b>ADM-03</b>	Gérer livre au catalogue	Administrateur	Permet à l'administrateur de gérer les livres présents dans le catalogue (Ajouter, Supprimer, Modifier).

Tableau 9: Descriptions des cas d'utilisation du module gestion des livres

### 3.2.3 Bloc Fonctionnels : Gestion des emprunts

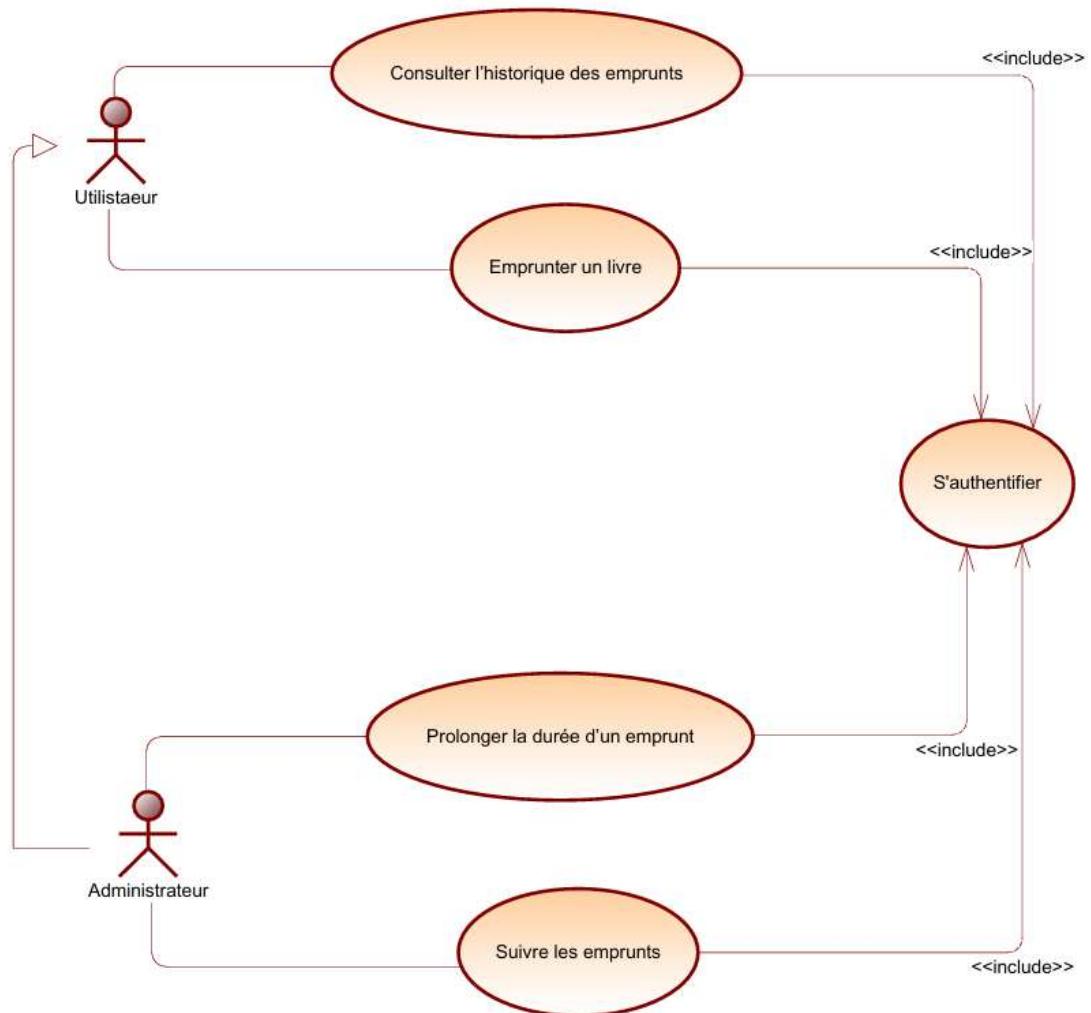


Figure 4 : Diagrammes des cas d'utilisation du module gestion des emprunts

Code	Libellé	Acteur	Description
<b>USR-07</b>	Consulter l'historique des emprunts	Utilisateur	Permet à l'utilisateur de consulter l'historique de ses emprunts passés, avec les dates et les livres empruntés.
<b>USR-08</b>	Emprunter un livre	Utilisateur	Permet à l'utilisateur d'emprunter un livre disponible dans le catalogue après authentification.
<b>ADM-04</b>	Prolonger la durée d'un emprunt	Administrateur	Permet à l'administrateur de prolonger la durée d'un emprunt pour un utilisateur.

<b>ADM-05</b>	Suivre les emprunts	Administrateur	Permet à l'administrateur de suivre tous les emprunts en cours et les retards.
---------------	---------------------	----------------	--------------------------------------------------------------------------------

Tableau 10: Descriptions des cas d'utilisation du module gestion des emprunts

### 3.2.4 Bloc Fonctionnels : Gestion des réservations

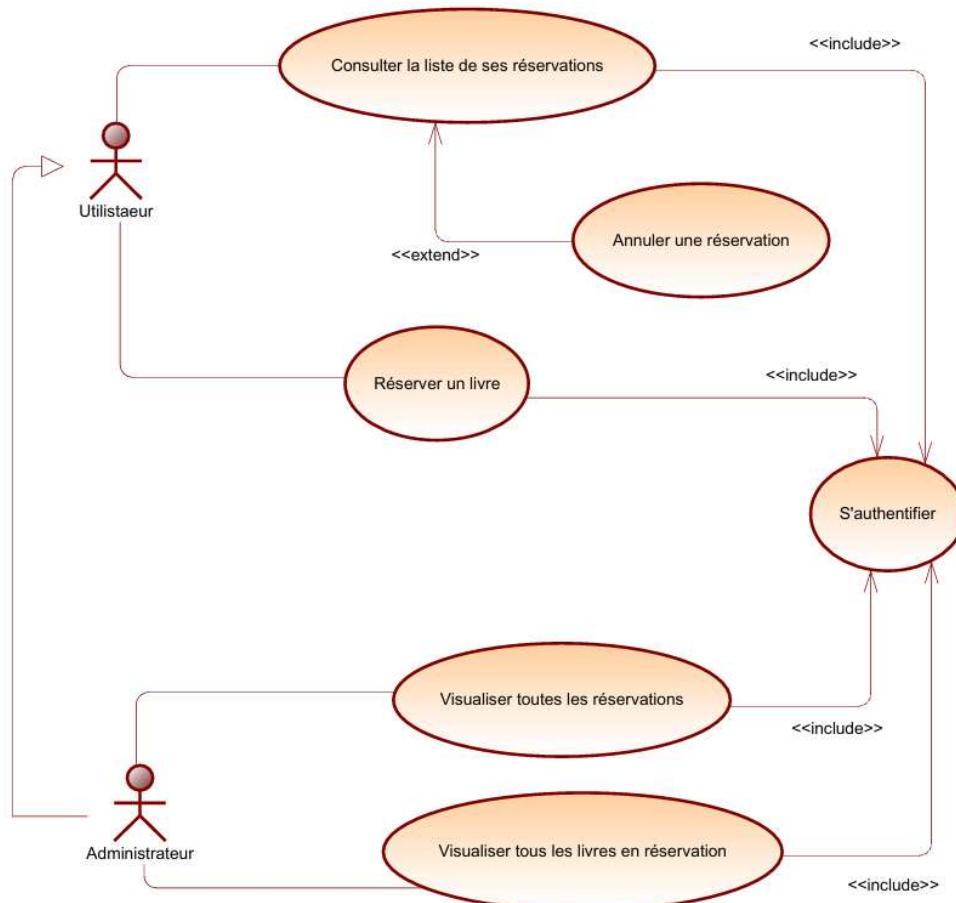


Figure 5: Diagrammes des cas d'utilisation du module gestion des réservations

Code	Libellé	Acteur	Description
<b>USR-08</b>	Réserver un livre	Utilisateur	Permet à l'utilisateur de réserver un livre déjà emprunté en s'ajoutant à une file d'attente.
<b>USR-09</b>	Consulter la liste de ses réservations	Utilisateur	Permet à l'utilisateur de consulter toutes ses réservations en cours avec les informations associées.
<b>USR-10</b>	Annuler une réservation	Utilisateur	Permet à l'utilisateur d'annuler une réservation en cas de changement de besoin.

<b>ADM-06</b>	Visualiser toutes les réservations	Administrateur	Permet à l'administrateur de consulter toutes les réservations effectuées par les utilisateurs.
<b>ADM-07</b>	Visualiser tous les livres en réservation	Administrateur	Permet à l'administrateur de consulter les livres actuellement réservés, avec les détails des utilisateurs associés.

*Tableau 11 : Descriptions des cas d'utilisation du module gestion des réservations*

## Conclusion

Ce chapitre a permis de définir les spécifications fonctionnelles et non fonctionnelles du système de gestion des bibliothèques universitaires, ainsi que les aspects conceptuels nécessaires à sa mise en œuvre. Nous avons détaillé les objectifs fonctionnels et les règles de gestion pour chaque bloc fonctionnel (gestion des comptes, des livres, des emprunts et des réservations), tout en illustrant les interactions entre les acteurs et le système à travers des diagrammes de cas d'utilisation. De plus, l'étude des besoins non fonctionnels a permis de garantir que le système respectera les exigences en matière de performance, de sécurité, de disponibilité et de convivialité. Ces bases solides serviront de guide pour la phase de développement technique, afin de concevoir un système performant et conforme aux attentes des utilisateurs.

## **Chapitre 3 : Etude technique**

# Introduction

Dans ce chapitre, nous examinerons les aspects techniques et technologiques qui soutiennent le développement du système de gestion des bibliothèques universitaires. L'architecture technique est un élément clé pour garantir un système robuste, performant et évolutif, capable de répondre aux besoins des utilisateurs. Nous commencerons par décrire l'architecture applicative, en présentant les différentes couches qui structurent le système et leur rôle dans le fonctionnement global. Ensuite, nous analyserons l'architecture technique, en mettant en évidence les composants matériels et logiciels nécessaires pour assurer un déploiement efficace et fiable. Enfin, nous détaillerons les outils et frameworks utilisés pour concevoir, développer et tester le système. Ces éléments techniques forment la base indispensable pour assurer la réussite du projet et sa conformité aux exigences définies.

## 1 . Architecture applicative

L'architecture applicative du système de gestion des bibliothèques universitaires repose sur une structure bien définie, permettant de garantir la modularité, la maintenabilité et l'évolutivité du système. Cette architecture suit une approche basée sur les microservices, où chaque service est conçu pour gérer un domaine fonctionnel spécifique, tout en s'intégrant harmonieusement avec les autres composants.

### 1.1 Description de l'architecture

Le système est structuré selon une architecture en couches, comprenant les éléments suivants :

#### Couche de présentation (Frontend) :

- Cette couche est responsable de l'interaction entre les utilisateurs (étudiants, enseignants et administrateurs) et le système.
- Elle est développée en Angular, offrant une interface utilisateur intuitive et réactive pour faciliter les recherches de livres, les emprunts et la gestion des comptes.
- Elle communique avec les microservices via des appels API REST sécurisés.

#### Couche métier (Backend) :

Cette couche gère la logique métier à travers des microservices dédiés. Chaque microservice couvre un domaine fonctionnel précis :

- **Service des comptes** : Gère l’authentification, l’autorisation et la gestion des rôles des utilisateurs.
- **Service des livres** : Permet la gestion du catalogue de livres (ajout, modification, suppression, disponibilité).
- **Service des emprunts** : Gère les emprunts des livres, les retards et les pénalités.
- **Service des réservations** : Permet de gérer les réservations des livres déjà empruntés.

Ces services sont développés en Java avec Spring Boot pour une gestion efficace des transactions et une intégration facile des fonctionnalités.

#### **Couche de données (Base de données) :**

- Cette couche est responsable du stockage des données des utilisateurs, des livres, des emprunts et des réservations.
- Une base de données relationnelle (MySQL) est utilisée pour garantir l’intégrité et la consistance des données.
- Chaque microservice dispose de sa propre base de données pour respecter le principe d’isolation des données dans une architecture microservices.

#### **Couche de communication (API Gateway) :**

- Un API Gateway centralisé est utilisé pour acheminer les requêtes des utilisateurs vers les microservices appropriés.
- Il assure également la gestion des authentifications via JWT (JSON Web Tokens), la sécurité des communications et la mise en place des politiques de routage.

### **1.2 Interaction entre les couches**

- Les utilisateurs interagissent avec le système via l’interface Angular (frontend), qui envoie des requêtes HTTP aux microservices.
- Les microservices traitent les requêtes en appliquant la logique métier appropriée, puis accèdent à leurs bases de données respectives pour lire ou écrire des données.
- L’API Gateway agit comme un intermédiaire, gérant les authentifications et redirigeant les requêtes vers les services concernés.

## II .Architecture technique

L'architecture technique du système de gestion des bibliothèques universitaires vise à répondre aux besoins de performance, de fiabilité et de sécurité. Elle s'appuie sur des choix technologiques et des outils adaptés pour garantir une infrastructure robuste et évolutive.

### 2.1 Structure technique globale

#### **Backend : Automatisation et sécurité**

Le backend est conçu pour gérer les traitements métier complexes, l'automatisation des processus et la sécurité globale du système.

##### **Automatisation avec Spring Batch :**

Spring Batch est utilisé pour exécuter des traitements en masse tels que :

- Le calcul périodique des retards d'emprunt.
- La génération de rapports d'utilisation des livres (emprunts, réservations).
- La gestion de processus critiques comme la mise à jour des pénalités dues aux retards.

Ces tâches sont exécutées à des intervalles programmés, garantissant leur efficacité et leur impact minimal sur les performances du système.

##### **Gestion centralisée de la sécurité avec Keycloak :**

**Authentification et autorisation** : Keycloak est intégré pour gérer l'accès utilisateur via OAuth2 et JWT (JSON Web Tokens).

**Gestion des rôles** : Keycloak assure la distinction entre les rôles d'utilisateur (**ROLE\_USER**) et d'administrateur (**ROLE\_ADMIN**) pour appliquer des restrictions adaptées à chaque fonctionnalité.

**Conformité et extensibilité** : Grâce à Keycloak, l'authentification centralisée permet de gérer facilement de nouveaux types d'utilisateurs et de renforcer les exigences de sécurité à l'avenir.

#### **Frontend : Intégration et expérience utilisateur**

Le frontend, développé en Angular, se concentre sur l'optimisation de l'expérience utilisateur et la communication efficace avec le backend.

- **Sécurisation des échanges** : Toutes les requêtes API envoyées depuis Angular vers le backend sont sécurisées par HTTPS et accompagnées de tokens JWT générés par Keycloak.
- **Interopérabilité** : Angular consomme les API REST exposées par les microservices, permettant une interface utilisateur fluide pour des fonctionnalités telles que la recherche de livres ou la gestion des emprunts et réservations.

### **Base de données : Cohérence et isolation des données**

La base de données relationnelle MySQL est utilisée pour stocker et organiser les informations critiques du système.

- **Partitionnement par microservices** : Chaque microservice dispose de sa propre base de données pour respecter le principe d'isolation des données, minimiser les dépendances, et faciliter la scalabilité.
- **Intégrité des données** : Les relations complexes, telles que celles entre utilisateurs, livres, emprunts et réservations, sont gérées via des contraintes de clé étrangère, assurant la cohérence globale des données.
- **Optimisation des performances** : L'indexation est appliquée sur les colonnes fréquemment consultées (par exemple, titre, auteur, ID utilisateur) pour garantir une réponse rapide aux requêtes.

## **II . Outils et Framework utilisés**

Le développement du système de gestion des bibliothèques universitaires s'appuie sur un ensemble d'outils et de frameworks modernes qui assurent une efficacité optimale, une sécurité renforcée, et une expérience utilisateur fluide. Cette section détaille les principaux outils et frameworks utilisés dans le projet.

### 3.1 Backend



#### Spring Boot

- Framework principal utilisé pour le développement des microservices backend.
- Offre des modules intégrés tels que Spring Data pour l'accès à la base de données, Spring Security pour la gestion des utilisateurs, et Spring Cloud pour l'architecture distribuée.



#### Spring Batch

Utilisé pour automatiser les traitements en masse, tels que :

- Le calcul des retards d'emprunts.
- La gestion des pénalités.
- La génération de rapports périodiques.

Garantit des performances élevées pour les tâches répétitives et planifiées.



#### Keycloak

- Fournit l'authentification et l'autorisation basées sur OAuth2 et JWT.
- Centralise la gestion des rôles et des utilisateurs tout en renforçant la sécurité des interactions entre les composants.

### 3.2 Frontend



#### Angular

- Framework utilisé pour développer une interface utilisateur dynamique et réactive.
- Fournit des outils avancés pour la création de composants, la gestion des formulaires, et la navigation.
- Permet une intégration fluide avec les API REST exposées par le backend.



#### HTML CSS JS

- Technologies de base utilisées pour la conception et le développement de l'interface utilisateur.

### 3.3 Base de données



#### MySQL

- Système de gestion de base de données relationnel choisi pour sa robustesse et sa performance.
- Permet de gérer efficacement les relations complexes entre les entités, telles que les utilisateurs, les livres, les emprunts et les réservations.
- Utilisé avec des mécanismes d'indexation pour optimiser les performances des requêtes.

### 3.4 Conteneurisation et Orchestration



#### Docker

- Utilisé pour conteneuriser les microservices, l'API Gateway, et Keycloak, garantissant une portabilité et une uniformité des environnements.

### 3.5 Environnements de Développement et de Gestion



#### IntelliJ IDEA

- IDE (Environnement de Développement Intégré) utilisé pour le développement des microservices backend en Java.



#### Visual Studio Code

- IDE utilisée pour le développement des composants frontend en Angular.



#### Postman

- Utilisé pour tester et déboguer les API REST exposées par les microservices.



#### Git

- Système de gestion de version utilisé pour collaborer sur le code et assurer un suivi des modifications.



#### GitHub

- Plateformes utilisées pour héberger le code source et gérer les pipelines CI/CD (Intégration et Déploiement Continus).

## Conclusion

Ce chapitre a permis d'explorer les aspects techniques et technologiques qui soutiennent le développement du système de gestion des bibliothèques universitaires. Nous avons présenté une structure technique globale détaillant les interactions entre le backend, le frontend, la base de données, ainsi que les composants clés tels que Keycloak pour la sécurité et Spring Batch pour l'automatisation des tâches critiques.

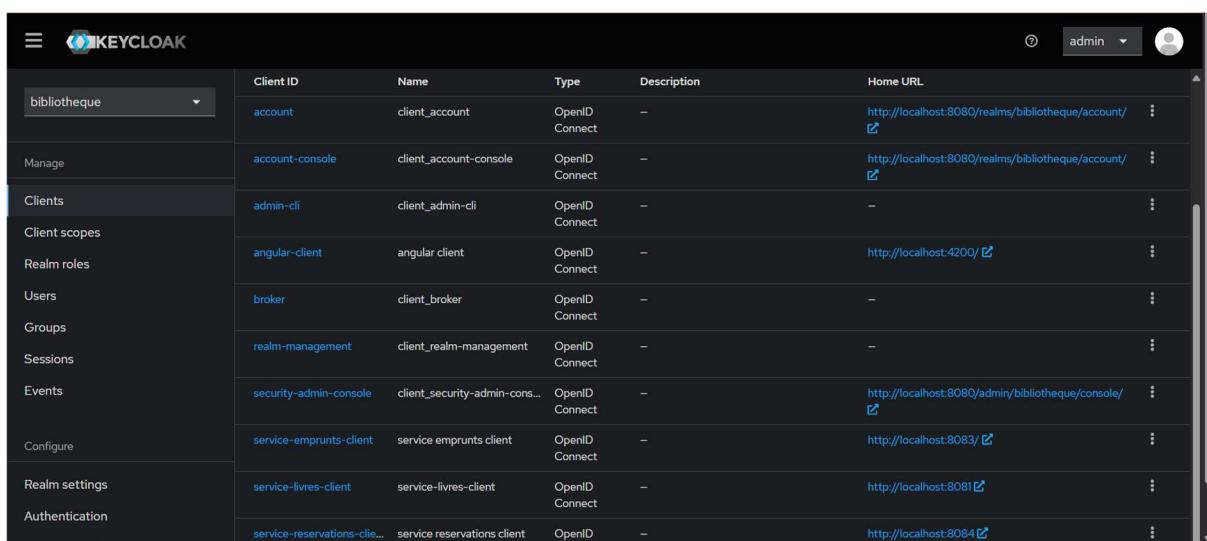
En outre, les outils et frameworks utilisés, comme Spring Boot, Angular, Docker, et Kubernetes, ont été décrits pour souligner leur rôle dans la mise en œuvre d'un système performant, sécurisé et évolutif. Ces choix technologiques garantissent la robustesse et la fiabilité de l'architecture tout en offrant une base solide pour le développement et l'intégration des fonctionnalités. Cette fondation technique constitue un pilier essentiel pour passer à la phase de développement et de mise en œuvre pratique du système.

## **Chapitre 4 : Réalisation**

# Introduction

Ce chapitre présente la phase de réalisation du système de gestion des bibliothèques universitaires, mettant en avant les résultats pratiques obtenus. Nous détaillerons les captures d'écran des tests effectués avec Postman pour valider les fonctionnalités backend, ainsi que l'interface utilisateur développée en Angular pour garantir une expérience fluide et intuitive. Cette section illustre les principales étapes de la mise en œuvre, les résultats obtenus, et leur conformité avec les spécifications définies, tout en démontrant le bon fonctionnement global du système.

## ¶ . Configuration des Clients dans Keycloak



The screenshot shows the Keycloak administration interface. The left sidebar is dark-themed with white text and icons. It includes sections for Manage, Clients, Client scopes, Realm roles, Users, Groups, Sessions, Events, Configure, Realm settings, and Authentication. The 'Clients' section is currently selected, indicated by a blue border. The main content area has a light gray background and displays a table of clients. The columns are Client ID, Name, Type, Description, and Home URL. There are also three vertical ellipsis icons for each row. The table contains the following data:

Client ID	Name	Type	Description	Home URL	⋮
account	client_account	OpenID Connect	–	<a href="http://localhost:8080/realm/bibliotheque/account/">http://localhost:8080/realm/bibliotheque/account/</a>	⋮
account-console	client_account-console	OpenID Connect	–	<a href="http://localhost:8080/realm/bibliotheque/account/">http://localhost:8080/realm/bibliotheque/account/</a>	⋮
admin-cli	client_admin-cli	OpenID Connect	–	–	⋮
angular-client	angular client	OpenID Connect	–	<a href="http://localhost:4200/">http://localhost:4200/</a>	⋮
broker	client_broker	OpenID Connect	–	–	⋮
realm-management	client_realm-management	OpenID Connect	–	–	⋮
security-admin-console	client_security-admin-cons...	OpenID Connect	–	<a href="http://localhost:8080/admin/bibliotheque/console/">http://localhost:8080/admin/bibliotheque/console/</a>	⋮
service-emprunts-client	service emprunts client	OpenID Connect	–	<a href="http://localhost:8083/">http://localhost:8083/</a>	⋮
service-livres-client	service livres-client	OpenID Connect	–	<a href="http://localhost:8081/">http://localhost:8081/</a>	⋮
service-reservations-client	service reservations client	OpenID	–	<a href="http://localhost:8084/">http://localhost:8084/</a>	⋮

Figure 6: Liste des clients configurés dans Keycloak

Cette capture d'écran illustre la configuration des clients dans Keycloak pour le système de gestion des bibliothèques universitaires. Chaque client représente une application ou un microservice enregistré dans le **realm "bibliotheque"**, avec des identifiants uniques comme **service-emprunts-client**, **service-livres-client**, **service-reservations-client** et **angular-client**. Ces clients sont configurés pour utiliser le protocole **OpenID Connect** et permettent une gestion centralisée de l'authentification et de l'autorisation. Chaque client est associé à des rôles et des permissions spécifiques, garantissant une interaction sécurisée entre les utilisateurs, le frontend, et les microservices backend.

## ¶ . Configuration des Microservices dans Eureka

The screenshot shows the Eureka interface at localhost:8761. It includes sections for 'System Status' and 'DS Replicas'. The 'System Status' section displays environment details like 'Environment: test', 'Data center: default', and uptime metrics. The 'DS Replicas' section lists four registered services: API-GATEWAY, SERVICE-EMPRUNTS, SERVICE-LIVRES, and SERVICE-RESERVATIONS, each with its status and port information.

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-3OEOEMA/api-gateway:9090</a>
SERVICE-EMPRUNTS	n/a (1)	(1)	UP (1) - <a href="#">service-emprunts:8083</a>
SERVICE-LIVRES	n/a (1)	(1)	UP (1) - <a href="#">service-livres:8081</a>
SERVICE-RESERVATIONS	n/a (1)	(1)	UP (1) - <a href="#">service-reservations:8084</a>

Figure 7: Services enregistrés dans Eureka

Cette capture d'écran montre l'interface de l'outil Eureka, utilisé pour l'enregistrement et la découverte des microservices du système de gestion des bibliothèques universitaires. Les services enregistrés incluent API-Gateway, SERVICE-EMPRUNTS, SERVICE-LIVRES, et SERVICE-RESERVATIONS, tous en état "UP", indiquant qu'ils sont opérationnels et prêts à recevoir des requêtes. Cela garantit une communication fluide entre les différents composants du système, facilitant leur coordination dans l'architecture microservices.

## ¶ . Validation d'un Token JWT sur jwt.io

Cette capture d'écran montre un token **JWT (JSON Web Token)** validé à l'aide de l'outil **jwt.io**. La section **Encoded** affiche le token complet, tandis que la section **Decoded** présente les informations décodées, comprenant :

- **Header** : Spécifie l'algorithme utilisé pour signer le token (ici **RS256**).
- **Payload** : Contient les données utilisateur telles que :
  - Les rôles attribués (par exemple, **ROLE\_ADMIN**).
  - Les informations d'accès au service (**service-livres-client**).
  - Les détails de l'utilisateur comme l'email, le prénom et le nom.

- **Signature Verified** : Confirme que le token a été signé et validé correctement avec la clé publique.

**Encoded** PASTE A TOKEN HERE

```
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIia6ICJcb0VIny1vSTRmYU9qaUFSSVuXOGM2YzZNS050bGZLNFJnV1dKQk1ZV3A4In0eyJleHAI0jE3MzcxMzg1NjIsIm1hdCI6MTczNzEzODI2MiwianRpIjoiMzdiNWQ5NGMtNTN1Ni00M2VhLWJKZDMtN2I1MmM0Nz1MWE3IiwiiaXNzIjoiaHR0CdovL2xyY2FsaG9zdDo4MDgwL3J1YWxty9iaWJsaW90aGVxdWUiLCJhdWQiOjYh2NvdW50IiwiC3ViIjoiMTAwNzFkNWUtZTcyMi00ZmI2LTg20GItMjc5NGEzMGR1YzWiiwidHlwIjoiQmVhcmVyiIwiYXpwIjoiC2VydmljZS1saXZyZXMy2xpZW50IiwiC2lkIjoiYjJiNzdkYmUtMmE1Mi00MWQ3LTg0NTETYzI2YTJjZjk0Y2R1IiwiYWNyIjoiMSIsImFsbG93ZWQtb3JpZ2lucyI6WyIvkIjdLCJyZWFsbV9hY2Nlc3Mi0nsicm9sZXMi0lsizGVmYXVsdC1yb2xlcyliaWJsaW90aGVxdWUiLCJvZmZsaW51X2FjY2VzcyIsIlJPTEVfQURNSU4iLCJ1bWFfYXV0aG9yaXphdG1vbijDfSwicmVzb3VyY2VfYWNjZXNzIjp7ImFjY291bnQi0nsicm9sZXMi0lsibWFuYWd1LWFjY291bnQiLCJtYW5hZ2UtYWNjb3VudC1saW5rcyIsInZpZXcteHJvZm1sZSjdfX0sInNjb3B1IjoiZW1haWwgchJvZm1sZSiisImVtYW1lsX3Zlcm1maWVkJipmYWxzZSwibmFtZSI6ImFkbWluMDIGYWRtaW4wM1IsInByZWZlcnJ1ZF91c2VybmfZSI6ImFkbWluMDIiLCJnaXZl19uYW11IjoiYWRtaW4wMiIsImZhbw1lsev9uYW11IjoiYWRtaW4wMiIsImVtYW1sIjoiYWRtaW4wMkBleGVtcGx1LmNvbSJ9.YQkm34CWVVgRf2dHG1PNCS9m3Q2afE0EhkZIE0uR5Z-YRS0v1HS_tk6P7C9VkkahJX06RSAeyH0dUNo-N0ts4P_N-6EtoSLHLn7I1sU-GEcXnAo-mnvMdZFFvX2kgkdJ4U-JwwmfsQukINpWb19XDbApuG1qm9NV8kjUfoc97bctvsT1hdz3cHcunCE6YudrRzuVxti_4sGdd1kgLr1PYw3sRE8KwX2Dh_rPCaw5riUxc2_sPmBfk-gn5CF2e57e6HCLPn7DDGqL1Zp5tGIBRygN1eLxmJ4cVqvjPDRnd02rdtE7aRsxcmCA_ioYD_WFbiqv2viBkB3gbEYgrmA
```

**Decoded** EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE	
<pre>{   "alg": "RS256",   "typ": "JWT",   "kid": "BoEh7-oI4fa0jiARIU18c6c6MKNtlfK4RgWWJBMYWp8" }</pre>	
PAYLOAD: DATA	
<pre>{   "exp": 1737138562,   "iat": 1737138262,   "jti": "37b5d94c-53e6-43ea-bdd3-7b52c47351a7",   "iss": "http://localhost:8080/realm/bibliothèque",   "aud": "account",   "sub": "10071dse-e722-4fb6-868b-2794a30dec60",   "typ": "Bearer",   "azp": "service-livres-client",   "sid": "b2b77dbe-2a52-41d7-8451-c26a2cf94cde",   "acr": "1",   "allowed_origins": [     "*"   ],   "realm_access": {     "roles": [       "default-roles-bibliothèque",       "offline_access",       "ROLE_ADMIN",       "uma_authorization"     ]   },   "resource_access": {     "account": {       "roles": [         "manage-account",         "manage-account-links",         "view-profile"       ]     }   },   "scope": "email profile",   "email_verified": false,   "name": "admin02 admin02",   "preferred_username": "admin02",   "given_name": "admin02",   "family_name": "admin02",   "email": "admin02@example.com" }</pre>	
VERIFY SIGNATURE	
<p>RSASHA256</p> <pre>base64UrlEncode(header) + "." + base64UrlEncode(payload), {   "e": "AQAB",   "kty": "RSA",   "n": "thT-Ilovyp4FWfuUsFet" }</pre> <p>Private Key in PKCS #8, PKCS #1, or JWK string format. The key never leaves your browser.</p>	

Signature Verified
SHARE JWT

Figure 8: Décodage et validation d'un token JWT

# ¶ . Tests des Microservices

## 4.1 Test du Module : Gestion des Livres

- Récupération de toutes les catégories

The screenshot shows the Postman application interface. At the top, there are tabs for 'POST openid-connect' and 'GET Get All Categories'. The 'GET Get All Categories' tab is active. Below it, the URL 'http://localhost:9090/api/livres/categories' is entered. The 'Headers' tab is selected, showing an 'Authorization' header with the value 'Bearer eyJhbGciOiJSUzI1NiIsInR5cC1gOAIzIdUiwia2lk...'. The 'Body' tab shows a JSON response with three categories: Informatique, Technologie et Innovation, and Histoire. The status bar at the bottom indicates a '200 OK' response with a duration of 4.67 seconds and a size of 564 B.

Key	Value	Description	...	Bulk Edit	Presets
Authorization	Bearer eyJhbGciOiJSUzI1NiIsInR5cC1gOAIzIdUiwia2lk...				
Key	Value	Description			

```
1 [  
2 {  
3   "id": 1,  
4   "nom": "Informatique"  
5 },  
6 {  
7   "id": 2,  
8   "nom": "Technologie et Innovation"  
9 },  
10 {  
11   "id": 3,  
12   "nom": "Histoire"  
13 }  
14 ]
```

Figure 9: Test de l'API Get All Categories dans le module Gestion des Livres

La requête GET a été envoyée à l'endpoint /api/livres/categories via l'API Gateway.

- **Objectif du test :** Récupérer toutes les catégories de livres disponibles dans le système.
- **Résultats :**
  - Statut de la réponse : 200 OK, indiquant que la requête a été traitée avec succès.
  - Corps de la réponse : Une liste en format JSON contenant les catégories des livres, telles que "Informatique", "Technologie et Innovation", et "Histoire".
  - Un token Bearer a été utilisé dans l'en-tête de la requête pour authentifier l'utilisateur, conformément aux règles de sécurité définies avec Keycloak.

- Ajout d'une catégorie

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost:9090/api/livres/categories
- Method:** POST
- Headers:**
  - Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cC1gOiAiSldUliwia2lkI...
- Body:** JSON response showing a new category added:
 

```

1  {
2    "id": 6,
3    "nom": "Physique"
4  }
```
- Status:** 201 Created
- Time:** 371 ms
- Size:** 494 B

Figure 10: Test de l'API Add Category dans le module Gestion des Livres

Cette capture d'écran montre un test de l'API Add Category du module de gestion des livres, effectué avec Postman.

- Objectif du test : Vérifier la création d'une nouvelle catégorie de livres via l'endpoint /api/livres/categories.
- Détails :
  - Méthode : POST
  - Endpoint : http://localhost:9090/api/livres/categories (via l'API Gateway).
  - En-tête : Un token Bearer est utilisé dans l'en-tête Authorization pour authentifier l'utilisateur avec des droits d'administrateur.
- Résultats :
  - Statut de la réponse : 201 Created - La catégorie a été ajoutée avec succès.

- **Mise à jour d'une catégorie**

The screenshot shows a Postman interface with the following details:

- Request Method:** PUT
- URL:** http://localhost:9090/api/livres/categories/6
- Headers:**
  - Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCigOAISSldUliwia2lkli...
  - Key: Value
- Body:** JSON (id: 6, nom: "Physique-chimie")
- Response Status:** 200 OK
- Response Time:** 319 ms
- Response Size:** 496 B

Figure 11: Test de l'API Update Category dans le module Gestion des Livres

Cette capture d'écran montre un test de l'API **Update Category** du module de gestion des livres, effectué avec **Postman**.

- **Objectif du test :** Vérifier la mise à jour des informations d'une catégorie existante via l'endpoint /api/livres/categories/{id}.
- **Détails :**
  - Méthode : **PUT**
  - Endpoint : http://localhost:9090/api/livres/categories/6 (via l'API Gateway).
  - En-tête : Un token **Bearer** est utilisé dans l'en-tête **Authorization** pour authentifier l'utilisateur avec des droits d'administrateur.
- **Résultats :**
  - Statut de la réponse : **200 OK** - La mise à jour a été effectuée avec succès.

- **Suppression d'une catégorie**

The screenshot shows a Postman interface with the following details:

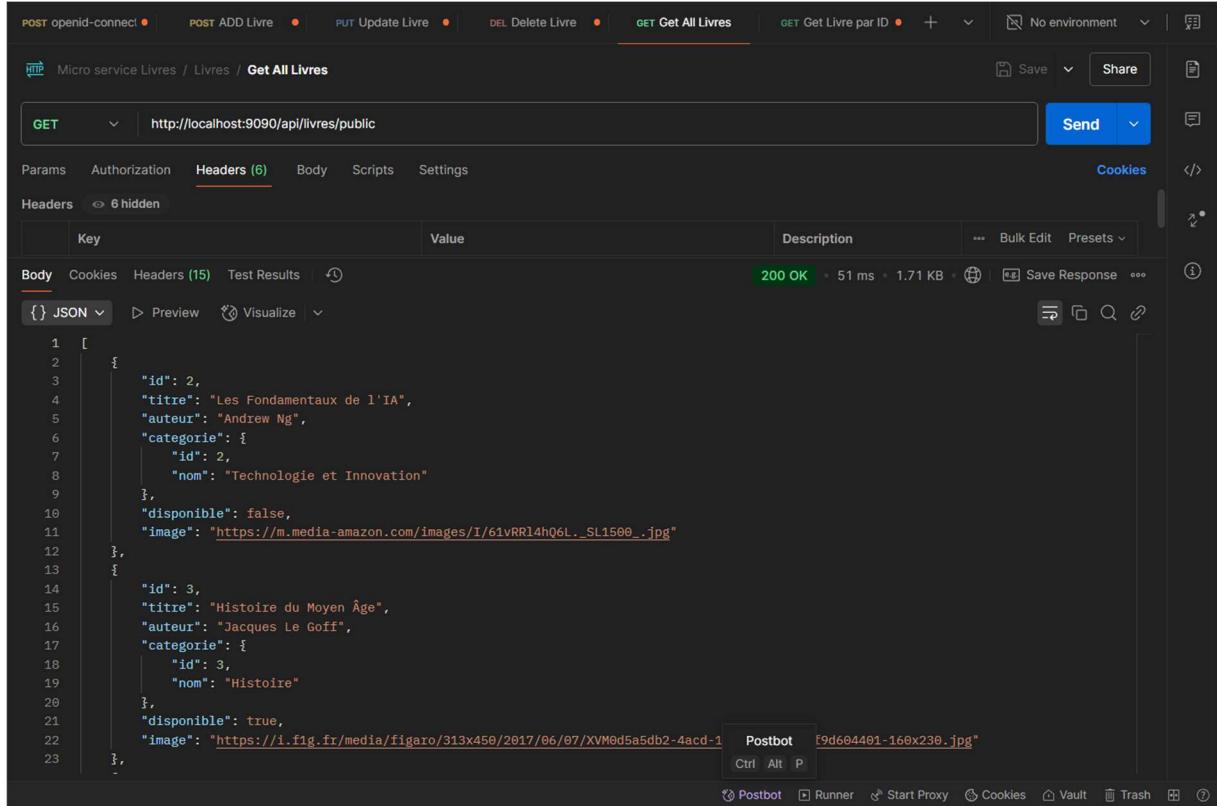
- URL:** `http://localhost:9090/api/livres/categories/6`
- Method:** `DELETE`
- Headers:**
  - `Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cIgOAISSldUlivia2lkli...`
  - `Key`: Value
- Body:** Raw, Preview, Visualize
- Response:** Status: `204 No Content`, Time: `1.17 s`, Size: `412 B`
- Postbot:** Ctrl Alt P

Figure 12: Test de l'API Delete Category dans le module Gestion des Livres

Cette capture d'écran montre un test de l'API **Delete Category** du module de gestion des livres, effectué avec **Postman**.

- **Objectif du test :** Vérifier la suppression d'une catégorie existante via l'endpoint `/api/livres/categories/{id}`.
- **Détails :**
  - Méthode : **DELETE**
  - Endpoint : `http://localhost:9090/api/livres/categories/6` (via l'API Gateway).
  - En-tête : Un token **Bearer** est utilisé dans l'en-tête **Authorization** pour authentifier l'utilisateur avec des droits d'administrateur.
- **Résultats :**
  - Statut de la réponse : **204 No Content** - La catégorie a été supprimée avec succès.

- **Récupération de tous les livres**



The screenshot shows a Postman interface with the following details:

- Request URL:** `http://localhost:9090/api/livres/public`
- Response Status:** `200 OK`
- Response Time:** `51 ms`
- Response Size:** `1.71 KB`
- Response Content:**

```

1 [ 
2   {
3     "id": 2,
4     "titre": "Les Fondamentaux de l'IA",
5     "auteur": "Andrew Ng",
6     "categorie": {
7       "id": 2,
8       "nom": "Technologie et Innovation"
9     },
10    "disponible": false,
11    "image": "https://m.media-amazon.com/images/I/61vRRl4hQ6L._SL1500_.jpg"
12  },
13  {
14    "id": 3,
15    "titre": "Histoire du Moyen Âge",
16    "auteur": "Jacques Le Goff",
17    "categorie": {
18      "id": 3,
19      "nom": "Histoire"
20    },
21    "disponible": true,
22    "image": "https://i.fig.fr/media/figaro/313x450/2017/06/07/XVM0d5a5db2-4acd-1"
23  }
]

```

Figure 13: Test de l'API Get All Livres dans le module Gestion des Livres

Cette capture d'écran montre un test de l'API **Get All Livres** du module de gestion des livres, effectué avec **Postman**.

- **Objectif du test :** Vérifier la récupération de la liste complète des livres disponibles dans le système via l'endpoint `/api/livres/public`.
- **Détails :**
  - Méthode : **GET**
  - Endpoint : `http://localhost:9090/api/livres/public` (via l'API Gateway).
  - En-tête : Le test a été effectué sans autorisation spécifique, car l'endpoint est public.
- **Résultats :**
  - Statut de la réponse : **200 OK** - La requête a été traitée avec succès.

- **Récupération d'un livre par ID**

The screenshot shows a Postman interface with the following details:

- Request URL:** GET http://localhost:9090/api/livres/public/3
- Headers:** (6) - includes Content-Type: application/json
- Body:** JSON response (15 items) showing a single book record:
 

```

1 {
2   "id": 3,
3   "titre": "Histoire du Moyen Âge",
4   "auteur": "Jacques Le Goff",
5   "categorie": {
6     "id": 3,
7     "nom": "Histoire"
8   },
9   "disponible": true,
10  "image": "https://i.f1g.fr/media/figaro/313x450/2017/06/07/XVMed5a5db2-4acd-11e7-9fe8-035f9d604401-160x230.jpg"
11 }
      
```
- Test Results:** 200 OK, 23 ms, 699 B
- Buttons:** Save Response, Postbot, Runner, Start Proxy, Cookies, Vault, Trash

Figure 14: Test de l'API Get Livre par ID dans le module Gestion des Livres

Cette capture d'écran montre un test de l'API **Get Livre par ID** du module de gestion des livres, effectué avec **Postman**.

- **Objectif du test :** Vérifier la récupération des informations d'un livre spécifique via son ID en utilisant l'endpoint /api/livres/public/{id}.
- **Détails :**
  - Méthode : **GET**
  - Endpoint : http://localhost:9090/api/livres/public/3 (via l'API Gateway).
  - En-tête : Le test a été effectué sans autorisation spécifique, car l'endpoint est public.
- **Résultats :**
  - Statut de la réponse : **200 OK** - La requête a été traitée avec succès.

- **Ajout d'un livre**

The screenshot shows a Postman interface with the following details:

- Header:** Authorization: Bearer eyJhbGciOiJSUzI1Nl... (a long token)
- Body (JSON):**

```

1 {
2     "id": 10,
3     "titre": "Programmation en Java",
4     "auteur": "Les Fondamentaux de l'IA",
5     "categorie": {
6         "id": 2,
7         "nom": "Technologie et Innovation"
8     },
9     "disponible": true,
10    "image": "amazon.com/images/I/61vRRl4hQ6L._SL1500_.jpg"
11 }

```

- Response:** 201 Created, 550 ms, 674 B

Figure 15: Test de l'API Add Livre dans le module Gestion des Livres

Cette capture d'écran montre un test de l'API **Add Livre** du module de gestion des livres, effectué avec **Postman**.

- **Objectif du test :** Vérifier la création d'un nouveau livre dans le système via l'endpoint /api/livres.
- **Détails :**
  - Méthode : **POST**
  - Endpoint : <http://localhost:9090/api/livres> (via l'API Gateway).
  - En-tête : Un token **Bearer** est utilisé dans l'en-tête **Authorization** pour authentifier l'utilisateur avec des droits d'administrateur.
- **Résultats :**
  - Statut de la réponse : **201 Created** - Le livre a été ajouté avec succès.

- **Mise à jour d'un livre**

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost:9090/api/livres/10
- Method:** PUT
- Headers:**
  - Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCigOiAiSldUiwi2IkI... (A long token)
  - Key: Value (Empty)
- Body:** JSON (Request Body)

```

1 {
2     "id": 10,
3     "titre": "Titre Mis à Jour",
4     "auteur": "Auteur Mis à Jour",
5     "categorie": {
6         "id": 2,
7         "nom": "Technologie et Innovation"
8     },
9     "disponible": false,
10    "image": "url/image/mise-a-jour.jpg"
11 }

```
- Response:** 200 OK (40 ms, 641 B)

```

{
    "id": 10,
    "titre": "Titre Mis à Jour",
    "auteur": "Auteur Mis à Jour",
    "categorie": {
        "id": 2,
        "nom": "Technologie et Innovation"
    },
    "disponible": false,
    "image": "url/image/mise-a-jour.jpg"
}

```

Figure 16: Test de l'API Update Livre dans le module Gestion des Livres

Cette capture d'écran montre un test de l'API **Update Livre** du module de gestion des livres, effectué avec **Postman**.

- **Objectif du test :** Vérifier la mise à jour des informations d'un livre existant dans le système via l'endpoint /api/livres/{id}.
- **Détails :**
  - Méthode : **PUT**
  - Endpoint : http://localhost:9090/api/livres/10 (via l'API Gateway).
  - En-tête : Un token **Bearer** est utilisé dans l'en-tête **Authorization** pour authentifier l'utilisateur avec des droits d'administrateur.
- **Résultats :**
  - Statut de la réponse : **200 OK** - Les informations du livre ont été mises à jour avec succès.

- Suppression d'un livre

The screenshot shows a Postman interface with the following details:

- Request Method:** DELETE
- URL:** http://localhost:9090/api/livres/10
- Headers:**

Key	Value	Description
Authorization	Bearer eyJhbGciOiJSUzI1NiIsInR5cC1gOiAiSldUIiwia2lkIi...	
Key	Value	Description
- Body:** Raw (empty)
- Response Status:** 204 No Content
- Response Headers:** 33 ms, 412 B

Figure 17: Test de l'API Delete Livre dans le module Gestion des Livres

Cette capture d'écran montre un test de l'API **Delete Livre** du module de gestion des livres, effectué avec **Postman**.

- **Objectif du test :** Vérifier la suppression d'un livre existant dans le système via l'endpoint /api/livres/{id}.
- **Détails :**
  - Méthode : **DELETE**
  - Endpoint : http://localhost:9090/api/livres/10 (via l'API Gateway).
  - En-tête : Un token **Bearer** est utilisé dans l'en-tête **Authorization** pour authentifier l'utilisateur avec des droits d'administrateur.
- **Résultats :**
  - Statut de la réponse : **204 No Content** - Le livre a été supprimé avec succès.

## 4.2 Test du Module : Gestion des Emprunts

- **Création d'un emprunt**

The screenshot shows a Postman interface with the following details:

- Request URL:** POST http://localhost:9090/api/emprunts/3/creer
- Body Content:**

```
1 {
2   "dateRetour": "2025-01-15"
3 }
4 }
```
- Response Status:** 200 OK
- Response Body (JSON):**

```
1 {
2   "id": 11,
3   "utilisateurId": "59c96beb-7b71-4e35-9865-38028d626ffc",
4   "livreId": "3",
5   "dateEmprunt": "2025-01-17",
6   "dateRetour": "2025-01-15",
7   "penalite": 0.0
8 }
```

Figure 18: Test de l'API Créer un Emprunt dans le module Gestion des Emprunts

Cette capture d'écran montre un test de l'API Crée un emprunt du module de gestion des emprunts, effectué avec Postman.

- Objectif du test : Vérifier la création d'un nouvel emprunt pour un utilisateur et un livre spécifique via l'endpoint /api/emprunts/{livreId}/creer.
- Détails :
  - Méthode : POST
  - Endpoint : http://localhost:9090/api/emprunts/3/creer (via l'API Gateway), où 3 correspond à l'ID du livre emprunté.
  - En-tête : Un token Bearer est utilisé dans l'en-tête Authorization pour authentifier l'utilisateur.
- Résultats :
  - Statut de la réponse : 200 OK - L'emprunt a été créé avec succès.

- **Consulter Mes Emprunts**

The screenshot shows a Postman collection named "POST openid-". The "Consulter Mes Emprunts" endpoint is selected. The request method is GET, and the URL is http://localhost:8083/api/emprunts/mes-emprunts. The "Headers" tab is active, showing an Authorization header with a Bearer token. The "Body" tab shows a JSON response with one item in an array, representing a loan record. The response status is 200 OK.

```

1 [ 
2   {
3     "id": 11,
4     "utilisateurId": "59c96beb-7b71-4e35-9865-38028d626ffc",
5     "livreId": "3",
6     "dateEmprunt": "2025-01-17",
7     "dateRetour": "2025-01-15",
8     "penalite": 0.0
9   }
10 ]

```

Figure 19: Test de l'API Consulter Mes Emprunts dans le module Gestion des Emprunts

Cette capture illustre un test de l'API **Consulter Mes Emprunts** effectué avec **Postman** pour vérifier la récupération des emprunts effectués par un utilisateur authentifié.

- **Objectif du test :** Récupérer la liste des emprunts associés à l'utilisateur connecté grâce à l'endpoint /api/emprunts/mes-emprunts.
- **Détails :**
  - Méthode : **GET**
  - Endpoint : <http://localhost:8083/api/emprunts/mes-emprunts>, correspondant au microservice **Emprunts**.
  - En-tête : Utilisation d'un token **Bearer** dans l'en-tête **Authorization** pour authentifier la requête.
- **Résultats :**
  - Statut de la réponse : **200 OK** - La requête a été traitée avec succès.

- **Consulter Tous les Emprunts**

The screenshot shows a Postman collection named "openid-". The "Tous Les Emprunts" endpoint is selected. The "Headers" tab is active, showing an "Authorization" header with the value "Bearer eyJhbGciOiJSUzI1NiIsInR5cC1gOAIISldUliwia2IkI...". The "Body" tab shows a JSON response containing two loan records:

```

1 [
2   {
3     "id": 2,
4     "utilisateurId": "55e270a7-aaaa-483a-a03e-6c87bddb4394",
5     "livreId": "6",
6     "dateEmprunt": "2025-01-13",
7     "dateRetour": "2025-01-15",
8     "penalite": 20.0
9   },
10  {
11    "id": 9,
12    "utilisateurId": "55e270a7-aaaa-483a-a03e-6c87bddb4394",
13    "livreId": "2",
14    "dateEmprunt": "2025-01-13",
15    "dateRetour": "2025-01-15",
16    "penalite": 0.0
17  }
]

```

Figure 20: Test de l'API Consulter Tous les Emprunts dans le module Gestion des Emprunts

Cette capture montre un test de l'API **Consulter Tous les Emprunts** effectué avec **Postman**. Cette fonctionnalité est destinée à l'administrateur pour visualiser tous les emprunts enregistrés dans le système.

- **Objectif du test :** Vérifier que l'API renvoie une liste complète de tous les emprunts, incluant les détails de chaque transaction.
- **Détails :**
  - Méthode : **GET**
  - Endpoint : <http://localhost:9090/api/emprunts/admin/tous-les-emprunts>, qui appartient au microservice **Emprunts**.
  - En-tête : Utilisation d'un token **Bearer** dans l'en-tête **Authorization** pour authentifier la requête en tant qu'administrateur.
- **Résultats :**
  - Statut de la réponse : **200 OK** - La requête a été traitée avec succès.

- **Mise à Jour de la Pénalité**

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost:9090/api/emprunts/11/mise-a-jour-penalite?penalite=10.5
- Method:** PUT
- Headers:**
  - Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cIgOIAiSldUiwia2lkli...
- Body:** JSON response showing the updated loan details:

```

1 {
2   "id": 11,
3   "utilisateurId": "59c96beb-7b71-4e35-9865-38028d626ffc",
4   "livreId": "3",
5   "dateEmprunt": "2025-01-17",
6   "dateRetour": "2025-01-15",
7   "penalite": 10.5
8 }
```
- Status:** 200 OK
- Test Results:** 180 ms, 522 B

Figure 21: Test de l'API Mise à Jour de la Pénalité dans le module Gestion des Emprunts

Cette capture illustre un test de l'API **Mise à Jour de la Pénalité** pour un emprunt spécifique, effectué avec **Postman**.

- **Objectif du test :** Vérifier que l'API permet de mettre à jour correctement la pénalité associée à un emprunt particulier.
- **Détails :**
  - Méthode : **PUT**
  - Endpoint : http://localhost:9090/api/emprunts/11/mise-a-jour-penalite?penalite=10.5, où 11 est l'ID de l'emprunt et 10.5 est le montant de la nouvelle pénalité.
  - En-tête : Utilisation d'un token **Bearer** dans l'en-tête **Authorization** pour authentifier la requête.
- **Résultats :**
  - **Statut de la réponse :** **200 OK** - La requête a été exécutée avec succès.

- **Consultation des Emprunts en Retard**

The screenshot shows a Postman interface with the following details:

- Request URL:** GET http://localhost:9090/api/emprunts/en-retard
- Headers:**
  - Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cC1gOAIISldUlwia2Ik... (with a long token)
  - Key: Value (empty)
- Body:** JSON response (19 lines of code)
- Test Results:** 200 OK (55 ms, 963 B)

```

1 [ 
2   { 
3     "id": 2,
4     "utilisateurId": "55e270a7-aaaa-483a-a03e-6c87bddb4394",
5     "livreId": "6",
6     "dateEmprunt": "2025-01-13",
7     "dateRetour": "2025-01-15",
8     "penalite": 0.0
9   },
10  { 
11    "id": 9,
12    "utilisateurId": "55e270a7-aaaa-483a-a03e-6c87bddb4394",
13    "livreId": "2",
14    "dateEmprunt": "2025-01-13",
15    "dateRetour": "2025-01-15",
16    "penalite": 0.0
17  },
18  { 
19    "id": 10,
...

```

Figure 22: Test de l'API Consultation des Emprunts en Retard dans le module Gestion des Emprunts

Cette capture met en évidence un test réalisé pour la fonctionnalité de consultation des emprunts en retard, effectué avec Postman.

- Objectif du test : Vérifier que l'API permet de récupérer correctement la liste des emprunts dont la date de retour est dépassée.
- Détails :
  - Méthode : GET
  - Endpoint : http://localhost:9090/api/emprunts/en-retard
  - En-tête : Un token Bearer est utilisé pour authentifier la requête via le champ Authorization.
- Résultats :
  - Statut de la réponse : 200 OK - La requête a été traitée avec succès.

- **Modification de la Date de Retour**

The screenshot shows a Postman test environment for a 'Micro Service Emprunts' API. The request URL is `http://localhost:9090/api/emprunts/11/modifier-date-retour`. The Headers tab is selected, showing an Authorization header with a Bearer token. The Body tab displays a JSON response with fields: id (11), utilisateurId (59c96beb-7b71-4e35-9865-38028d626ffc), livreId (3), dateEmprunt (2025-01-17), dateRetour (2025-01-20), and penalite (10.5). The status bar at the bottom indicates a 200 OK response.

Figure 23: Test de l'API Modification de la Date de Retour dans le module Gestion des Emprunts

Cette capture illustre un test réalisé pour la fonctionnalité de **modification de la date de retour d'un emprunt**, exécuté avec **Postman**.

- **Objectif du test :** Vérifier que l'API permet de mettre à jour la date de retour d'un emprunt spécifique pour prolonger ou ajuster la période d'emprunt.
- **Détails :**
  - **Méthode : PUT**
  - **Endpoint :** `http://localhost:9090/api/emprunts/11/modifier-date-retour`
  - **En-tête :** Un token **Bearer** est inclus pour authentifier la requête.
  - **Corps de la requête :** Spécifie la nouvelle **date de retour**.
- **Résultats :**
  - **Statut de la réponse : 200 OK** - La modification a été effectuée avec succès.

- **Retourner un Livre**

The screenshot shows a Postman collection named "POST openid-". The selected request is "PUT Retourner Livre" with the URL `http://localhost:9090/api/emprunts/10/retourner-livre`. The "Headers" tab is active, showing an "Authorization" header with the value `Bearer eyJhbGciOiJSUzI1NiIsInR5cIgOAISSldUlivia2Ikli...`. The "Body" tab shows a single JSON object: `{ "livre": "Livre retourné avec succès et marqué comme disponible." }`. The "Test Results" tab shows a successful response: `200 OK`, `193 ms`, `432 B`. The response body is: `1 Livre retourné avec succès et marqué comme disponible.`.

Figure 24: Test de l'API Retourner un Livre dans le module Gestion des Emprunts

Cette capture illustre un test de la fonctionnalité **Retourner un Livre**, réalisé à l'aide de **Postman**.

- **Objectif du test :** Vérifier que l'API permet de marquer un livre comme retourné et le rend disponible pour de futurs emprunts.
- **Détails :**
  - **Méthode : PUT**
  - **Endpoint :** `http://localhost:9090/api/emprunts/10/retourner-livre`
  - **En-tête :** Inclut un token **Bearer** pour l'authentification de l'utilisateur.
  - **Corps de la réponse :** Retourne un message de confirmation indiquant que le livre a été retourné avec succès et marqué comme disponible.
- **Résultats :**
  - **Statut de la réponse : 200 OK** - L'action a été exécutée avec succès.

## 1.1. Test du Module : Gestion des réservations

- Réserver un Livre

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost:8084/api/reservations/2/reserver
- Method:** POST
- Headers:** Authorization (Value: Bearer eyJhbGciOiJSUzI1NiIsInR5cC1gOAIzIdUlwiaw2Ikli...)
- Body:** JSON response (id: 19, utilisateurId: "59c96beb-7b71-4e35-9865-38028d626ff", livreId: "2", dateReservation: "2025-01-17", disponible: false)
- Status:** 200 OK
- Time:** 45 ms
- Size:** 551 B

Figure 25: Test de l'API Réserver un Livre dans le module Gestion des Réservations

Cette capture illustre un test de la fonctionnalité **Réserver un Livre**, effectué à l'aide de **Postman**.

- **Objectif du test :** Vérifier que l'API permet à un utilisateur de réserver un livre et de marquer ce dernier comme non disponible pour d'autres emprunts ou réservations.
- **Détails :**
  - **Méthode :** POST
  - **Endpoint :** http://localhost:8084/api/reservations/2/reserver
  - **En-tête :** Inclut un token **Bearer** pour l'authentification de l'utilisateur.
- **Résultats :**
  - **Statut de la réponse :** 200 OK - La réservation a été effectuée avec succès.

- **Afficher Mes Réservations**

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost:9090/api/reservations/mes-reservations
- Method:** GET
- Headers:**
  - Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cC1gOiAiSldUliwia2lk...
- Body:** JSON response (200 OK) containing reservation and livre details.

```

1 [ 
2   {
3     "reservation": {
4       "id": 19,
5       "utilisateurId": "59c96beb-7b71-4e35-9865-38028d626ff",
6       "livreId": "2",
7       "dateReservation": "2025-01-17",
8       "disponible": false
9     },
10    "livre": {
11      "id": 2,
12      "titre": "Les Fondamentaux de l'IA",
13      "auteur": "Andrew Ng",
14      "categorie": {
15        "id": 2,
16        "nom": "Technologie et Innovation"
17      },
18      "disponible": false,
19      "image": "https://m.media-amazon.com/images/I/61vRRl4hQ6L._SL1500_.jpg"
-- ]
  
```

Figure 26: Test de l'API Afficher Mes Réservations dans le module Gestion des Réservations

Cette capture montre un test de la fonctionnalité **Afficher Mes Réservations**, réalisé via **Postman**.

- **Objectif du test :** Vérifier que l'API permet à un utilisateur authentifié d'afficher toutes ses réservations en cours.
- **Détails :**
  - **Méthode :** GET
  - **Endpoint :** http://localhost:9090/api/reservations/mes-reservations
  - **En-tête :** Inclus un token **Bearer** pour l'authentification de l'utilisateur.
- **Résultats :**
  - **Statut de la réponse :** 200 OK - Les données des réservations de l'utilisateur ont été récupérées avec succès.

- **Annuler une Réservation**

The screenshot shows a Postman interface with the following details:

- Request Method:** DELETE
- Request URL:** http://localhost:9090/api/reservations/18/annuler
- Headers:**

Key	Value	Description	... Bulk Edit	Presets
Authorization	Bearer eyJhbGciOiJSUzI1NiIsInR5cCigOiAiSldUiwi2Ikli...			
Key	Value	Description		
- Body:** Raw (Response body shown below)
- Response Headers:** 200 OK, 66 ms, 499 B
- Response Body:**

```
1 Réserve annulée avec succès.
```

Figure 27: Test de l'API Annuler une Réservation dans le module Gestion des Réservations

Cette capture montre un test de la fonctionnalité **Annuler une Réservation**, réalisé via **Postman**.

- **Objectif du test :** Vérifier que l'API permet à un utilisateur authentifié d'annuler une réservation existante.
- **Détails :**
  - **Méthode : DELETE**
  - **Endpoint :** http://localhost:9090/api/reservations/18/annuler
  - **En-tête :** Inclus un token **Bearer** pour l'authentification de l'utilisateur.
  - **Corps de la réponse :**
    - Message de confirmation indiquant que la réservation a été annulée avec succès.
- **Résultats :**
  - **Statut de la réponse : 200 OK** - La réservation spécifiée a été supprimée avec succès du système.

- **Afficher Toutes les Réservations**

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost:9090/api/reservations/admin/toutes-les-reservations
- Method:** GET
- Headers:**
  - Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCigOIAiSldUiwia2lkii...
- Body:** A JSON array with one element representing a reservation:
 

```

1 [ 
2   {
3     "id": 19,
4     "utilisateurId": "59c96beb-7b71-4e35-9865-38028d626ffc",
5     "livreId": "2",
6     "dateReservation": "2025-01-17",
7     "disponible": false
8   }
9 ]
      
```
- Status:** 200 OK
- Time:** 29 ms
- Size:** 594 B

Figure 28: Test de l'API Afficher Toutes les Réservations dans le module Gestion des Réservations

Cette capture montre un test de la fonctionnalité **Afficher Toutes les Réservations**, réalisé via **Postman**.

- **Objectif du test :** Vérifier que l'API permet à un administrateur authentifié de consulter toutes les réservations enregistrées dans le système.
- **Détails :**
  - **Méthode :** GET
  - **Endpoint :** http://localhost:9090/api/reservations/admin/toutes-les-reservations
  - **En-tête :** Inclus un token **Bearer** pour l'authentification de l'administrateur.
- **Résultats :**
  - **Statut de la réponse :** 200 OK - Toutes les réservations ont été récupérées avec succès.

- **Afficher les Livres Réservés**

The screenshot shows a Postman interface with the following details:

- Request URL:** GET http://localhost:9090/api/reservations/admin/livres-reserves
- Headers:**
  - Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCigOjAiSldUliwia2ikli...
- Body:** JSON response (15 items) showing a single book reservation record.
- Test Results:** Status 200 OK, 42 ms, 675 B.

```

1 [ 
2   { 
3     "id": 2,
4     "titre": "Les Fondamentaux de l'IA",
5     "auteur": "Andrew Ng",
6     "categorie": {
7       "id": 2,
8       "nom": "Technologie et Innovation"
9     },
10    "disponible": false,
11    "image": "https://m.media-amazon.com/images/I/61vRRl4hQ6L.._SL1500_.jpg"
12  }
13 ]
  
```

Figure 29: Test de l'API Afficher les Livres Réservés dans le module Gestion des Réservations

Cette capture montre un test de la fonctionnalité **Afficher les Livres Réservés**, réalisé via **Postman**.

- **Objectif du test :** Vérifier que l'API permet à un administrateur authentifié de consulter tous les livres actuellement réservés dans le système.
- **Détails :**
  - **Méthode :** GET
  - **Endpoint :** http://localhost:9090/api/reservations/admin/livres-reserves
  - **En-tête :** Inclus un token **Bearer** pour l'authentification de l'administrateur.
- **Résultats :**
  - **Statut de la réponse :** 200 OK - Tous les livres réservés ont été récupérés avec succès.

## Interface Utilisateur

### Page de Connexion

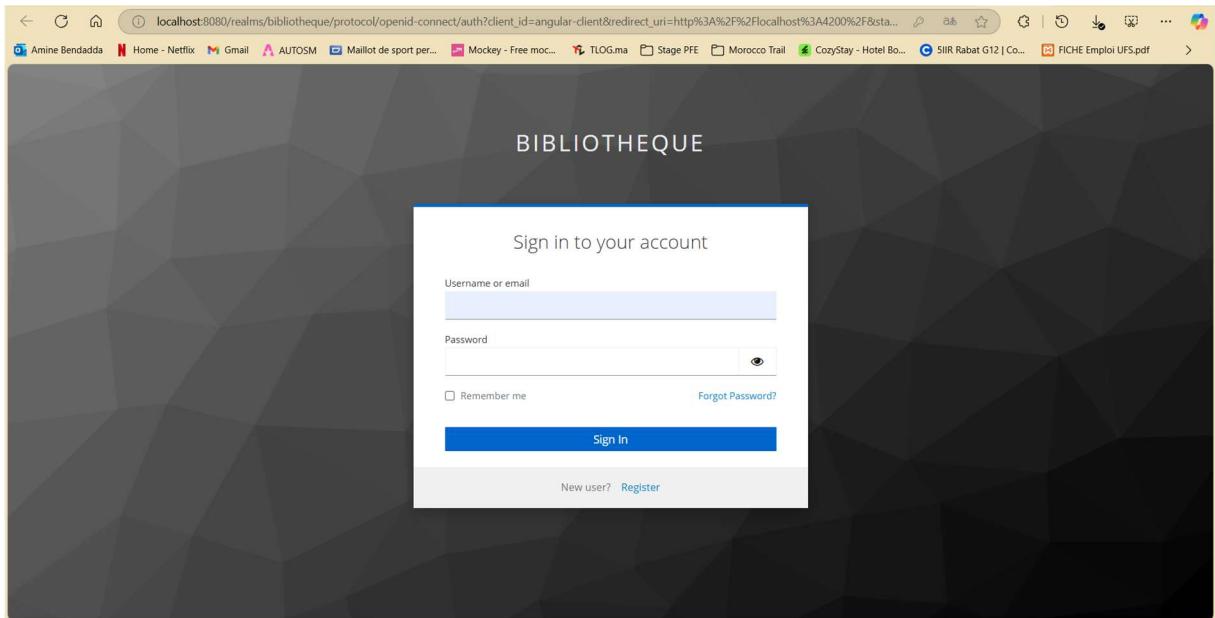


Figure 30: Interface Utilisateur - Page de Connexion

L'interface de connexion permet aux utilisateurs de s'authentifier et d'accéder à leurs comptes dans l'application de gestion de bibliothèque.

### Tableau de Bord Étudiant

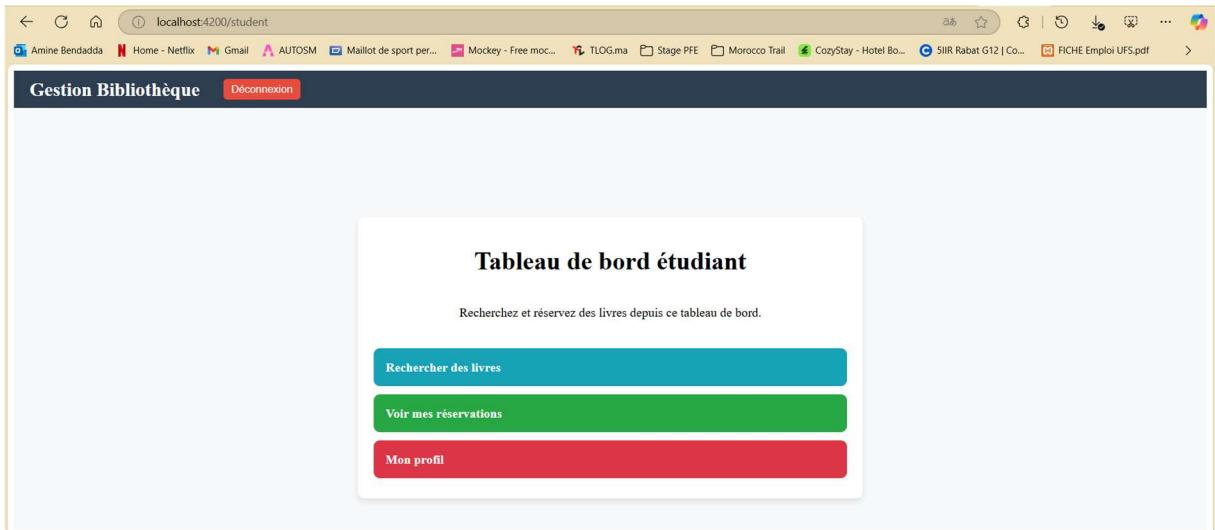


Figure 31: Interface Utilisateur - Tableau de Bord Étudiant

Le tableau de bord étudiant permet de rechercher des livres, consulter les réservations et accéder au profil utilisateur.

## Page de Recherche des Livres

Gestion Bibliothèque   Déconnexion

Liste des livres

Rechercher un livre par titre ou auteur   Toutes les catégories   Tous les statuts

**Intelligence artificielle et enseignement supérieur**  
Auteur : Andrew Ng  
Catégorie : Technologie et Innovation  
Status : Indisponible  
[Voir les détails](#)

**HISTOIRE DU MOYEN ÂGE**  
Auteur : Jacques Le Goff  
Catégorie : Histoire  
Status : Indisponible  
[Voir les détails](#)

**NARUTO**  
Auteur : Auteur Mis à Jour  
Catégorie : Technologie et Innovation  
Status : Indisponible  
[Voir les détails](#)

**JANE DOE**  
Auteur : John Doe  
Catégorie : Informatique  
Status : Disponible  
[Voir les détails](#)

Figure 32: Interface Utilisateur - Page de Recherche des Livres

Cette interface permet aux étudiants de consulter la liste des livres disponibles dans la bibliothèque. Les utilisateurs peuvent filtrer les livres par titre, catégorie et statut. Chaque livre est présenté avec ses détails, notamment le titre, l'auteur, la catégorie et la disponibilité. Un bouton "Voir les détails" est disponible pour accéder à plus d'informations sur chaque livre.

## Page des Détails d'un Livre

Gestion Bibliothèque   Déconnexion

Détails du livre

**HISTOIRE DU MOYEN ÂGE**  
Auteur : Jacques Le Goff  
Catégorie : Histoire  
Status : Indisponible  
[Réserver](#)

Figure 33: Interface Utilisateur - Page des Détails d'un Livre

Cette interface affiche les informations détaillées d'un livre sélectionné. Les données présentées incluent :

- **Titre du livre** : Affiché en haut de la section des détails.
- **Auteur** : Nom de l'auteur.
- **Catégorie** : Catégorie à laquelle le livre appartient.
- **Statut** : Disponibilité du livre (ex. : Disponible, Indisponible).

Un bouton "Réserver" est disponible pour permettre aux utilisateurs de réserver le livre s'il est disponible. Cette action est bloquée si le statut est "Indisponible".

## Mes Réservations

Figure 34: Interface Utilisateur - Mes Réservations

Cette interface permet aux utilisateurs de visualiser leurs réservations en cours. Les informations suivantes sont affichées pour chaque réservation :

- **Titre du livre réservé** : Nom du livre.
- **Auteur** : Auteur du livre.
- **Date de réservation** : Date à laquelle le livre a été réservé.
- **Statut** : Disponibilité du livre (par exemple, "Non disponible" si réservé).

Un bouton "**Annuler la réservation**" est disponible pour chaque réservation, permettant à l'utilisateur de libérer le livre réservé.

## Tableau de Bord Administrateur



Figure 35: Interface Utilisateur - Tableau de Bord Administrateur

Le tableau de bord administrateur permet de gérer les utilisateurs, les livres, et les réservations à partir des options disponibles telles que la recherche de livres, l'ajout de nouveaux livres, la gestion des réservations, et l'accès au profil.

### Ajouter un nouveau livre

A screenshot of a "Ajouter un nouveau livre" (Add new book) form. It has a dark header bar with "Gestion Bibliothèque" and "Déconnexion". The main form area has fields for "Titre" (Title), "Auteur" (Author), "URL de l'image" (Image URL), and "Catégorie" (Category) with a dropdown menu showing "Choisissez une catégorie". There's also a "Disponible" (Available) checkbox which is checked. At the bottom is a blue "Ajouter" (Add) button.

Figure 36: Interface Utilisateur - Ajouter un nouveau livre

La page "Ajouter un nouveau livre" permet à l'administrateur de remplir un formulaire avec les informations du livre (titre, auteur, URL de l'image, catégorie et disponibilité) pour ajouter un nouveau livre à la bibliothèque.

## Conclusion

Ce chapitre a présenté les différentes interfaces et fonctionnalités essentielles du système de gestion de bibliothèque universitaire. Nous avons exploré les interfaces dédiées aux utilisateurs et administrateurs, incluant la connexion sécurisée, la navigation à travers des tableaux de bord intuitifs, et des fonctionnalités clés telles que la recherche, la réservation, et la gestion des livres et des emprunts. Les pages sont conçues pour offrir une expérience utilisateur fluide et adaptée aux besoins des étudiants et administrateurs, avec une mise en œuvre claire des rôles et permissions. Cette structure permet une gestion efficace des opérations bibliothécaires tout en garantissant une navigation conviviale et sécurisée.

## **Annexes**

**Code Source :**

<https://github.com/amine-bendadda/bibliotheque-universitaire>