

juin 2024

# RAPPORT

## Application de Gestion de Bibliothèque



### Réalisé par :

- marouane boughalmi
- abdelhamid abouri
- youssef elaamari

### Encadrée par :

- Mr. FOUAD MOUFARREH

# SOMMAIRE

## 01 Introduction

- 1.1 Présentation du projet
- 1.2 Objectifs du projet

## 02 Description du Projet

- 2.1 Description de l'application
- 2.2 Fonctionnalités principales de l'application
- 2.3 Technologies utilisées

## 03 Architecture de l'Application

- 3.1 Vue d'ensemble de l'architecture (frontend et backend)
- 3.2 Technologies utilisées pour chaque partie de l'application
- 3.3 Schéma de la base de données

## 04 Réalisation Technique

- 4.1 Détails sur l'implémentation de l'application
- 4.2 Explication des choix techniques
- 4.3 Principaux défis rencontrés et solutions apportées

## 05 Tests et Validation

- 5.1 Stratégie de test utilisée
- 5.2 Résultats des tests
- 5.3 Démonstration de l'application (captures d'écran)

## 06 Conclusion

# INTRODUCTION

## 1.1 Présentation du projet

L'application de gestion de bibliothèque est une solution logicielle complète destinée à simplifier et optimiser les opérations quotidiennes d'une bibliothèque. Elle permet aux bibliothécaires de gérer efficacement les collections de livres, les utilisateurs, et les transactions de prêts et de retours. L'application comprend des fonctionnalités robustes telles que la création et la gestion des comptes utilisateurs, l'ajout et la mise à jour des informations sur les livres, et le suivi précis des prêts et des retours. Grâce à une interface utilisateur intuitive et à une architecture backend sécurisée et performante, l'application assure une expérience utilisateur fluide et sécurisée. Elle intègre des mécanismes d'authentification et de sécurité pour protéger les données sensibles et garantit que toutes les opérations sont effectuées de manière fiable et efficace. En facilitant les tâches administratives et en offrant une visibilité complète sur les opérations de la bibliothèque, cette application aide à améliorer la productivité et à offrir un meilleur service aux utilisateurs de la bibliothèque.

# **1.2 Objectifs du projet**

## **1.Faciliter la Gestion des Livres**

- Permettre l'ajout, la mise à jour, la suppression et la recherche de livres de manière efficace.
- Fournir une vue d'ensemble des livres disponibles, empruntés et en attente de retour.

## **2.Améliorer l'Expérience Utilisateur**

- Offrir une interface utilisateur intuitive et conviviale pour une navigation facile.
- Permettre aux utilisateurs de gérer leurs comptes, de consulter l'historique de leurs emprunts et de réserver des livres en ligne.

## **3.Optimiser la Gestion des Prêts et Retours**

- Automatiser le suivi des prêts et des retours des livres.
- Envoyer des notifications pour les dates de retour et les rappels pour les retards.

## **4.Assurer la Responsivité de l'Application**

- Adapter l'application à différents types d'appareils (ordinateurs, tablettes, smartphones) pour une accessibilité optimale.

## **5.Renforcer la Sécurité et la Confidentialité**

- Implémenter l'authentification JWT pour sécuriser les routes et protéger les données utilisateur.
- Assurer la validation des données côté serveur pour prévenir les erreurs et les accès non autorisés.

## **6.Améliorer l'Administration de la Bibliothèque**

- Fournir des outils d'administration pour gérer les utilisateurs, les livres et les prêts.
- Générer des rapports et des statistiques sur les opérations de la bibliothèque.

## **7.Optimiser la Gestion des Prêts et Retours**

- Utiliser une architecture backend robuste basée sur Node.js et Express.js pour gérer un grand volume de données et de transactions.
- Utiliser une base de données relationnelle (MySQL) pour une gestion efficace et fiable des données.

# DESCRIPTION DU PROJET

## 1. Description de l'application

L'application de gestion de bibliothèque est une solution complète conçue pour faciliter la gestion des utilisateurs, des livres, et des prêts dans une bibliothèque. Elle offre une interface utilisateur conviviale et un backend robuste, garantissant une expérience utilisateur fluide et sécurisée.

Voici une description détaillée des principales fonctionnalités de l'application.

## 2. Principales Fonctionnalités de l'App

### 1. Gestion des Utilisateur

- Création de Comptes Utilisateur** : Les utilisateurs peuvent s'inscrire via la page d'inscription. Chaque compte utilisateur est associé à un identifiant unique, un nom, une adresse email, un mot de passe hashé, et un rôle (utilisateur ou administrateur).
- Authentification et Sécurité** : L'application utilise JSON Web Tokens (JWT) pour l'authentification. Les utilisateurs peuvent se connecter via la page de connexion, recevant un token JWT pour sécuriser leurs sessions.

- **Gestion des Comptes Utilisateur** : Les administrateurs peuvent visualiser, modifier et supprimer les comptes utilisateur via une interface dédiée. Les utilisateurs peuvent mettre à jour leurs informations personnelles.

## 2. Gestion des Livres

- **Ajout de Livres** : Les administrateurs peuvent ajouter de nouveaux livres à la bibliothèque via un formulaire dédié. Chaque livre est associé à un identifiant unique, un titre, un auteur, une année de publication, un genre, un résumé et un statut de disponibilité.
- **Modification et Suppression de Livres** : Les administrateurs peuvent mettre à jour les informations des livres ou supprimer des livres via l'interface d'administration.
- **Recherche et Consultation de Livres** : Les utilisateurs peuvent rechercher des livres par titre, auteur, genre ou année de publication. Ils peuvent consulter les détails de chaque livre, y compris le résumé et la disponibilité.

## 3. Gestion des Prêts

- **Historique des Prêts** : Les utilisateurs peuvent consulter l'historique de leurs prêts passés. Les administrateurs peuvent accéder à l'historique complet des prêts de la bibliothèque.

- **Création de Prêts** : Les utilisateurs peuvent emprunter des livres disponibles via une interface conviviale. Chaque prêt est associé à un identifiant unique, à l'utilisateur qui a emprunté le livre, au livre emprunté, à la date d'emprunt et à la date de retour prévue.
- **Suivi des Prêts** : Les utilisateurs peuvent consulter la liste de leurs prêts en cours, voir les dates de retour prévues et recevoir des notifications pour les retours imminents ou les retards.
- **Retour de Livres** : Les administrateurs peuvent enregistrer le retour des livres empruntés, mettant à jour le statut de disponibilité des livres dans la bibliothèque.

## 4. Interface Utilisateur Conviviale

- **Responsivité** : L'application est conçue pour être utilisable sur divers appareils (ordinateurs, tablettes, smartphones), garantissant une expérience utilisateur optimale quel que soit le dispositif utilisé.
- **Interactivité** : Utilisation de formulaires interactifs pour l'ajout et la modification des livres, des prêts et des utilisateurs. Affichage des données sous forme de tables et de listes pour une visualisation claire et organisée.

- **Navigation Facile** : Une barre de navigation intuitive permet aux utilisateurs d'accéder facilement aux différentes sections de l'application (inscription, connexion, liste des livres, administration).

## 5. Sécurité et Validation

- **Sécurisation des Routes** : Les routes critiques de l'application sont sécurisées via l'authentification JWT, empêchant les accès non autorisés.
- **Validation des Données** : Les données saisies par les utilisateurs sont validées côté serveur pour éviter les erreurs de saisie et garantir l'intégrité des informations stockées dans la base de données.

En résumé, l'application de gestion de bibliothèque offre une solution complète pour la gestion des utilisateurs, des livres et des prêts, tout en assurant une expérience utilisateur fluide, sécurisée et intuitive.

## 2. Technologies utilisées

### 1. Frontend : React.js

- **React** : Une bibliothèque JavaScript pour construire des interfaces utilisateur interactives et dynamiques.
- **React Router** : Utilisé pour gérer la navigation entre les différentes pages de l'application.

- **Axios** : Utilisé pour effectuer des requêtes HTTP vers le backend.
- **CSS** : Pour le styling des composants et l'interface utilisateur.

## 2. Backend : Node.js / Express.js

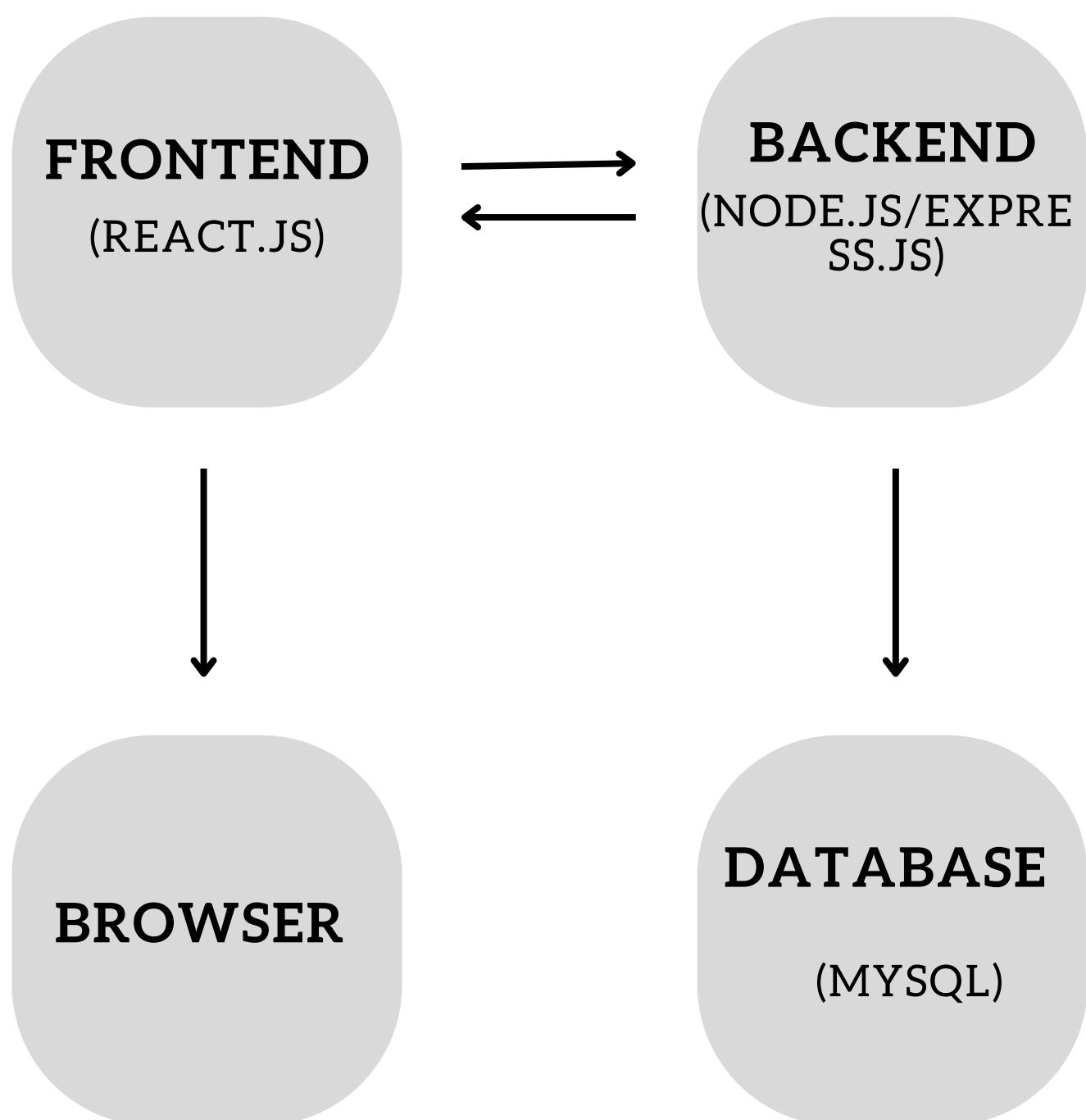
- **Node.js** : Un environnement d'exécution JavaScript côté serveur.
- **Express.js** : Un framework web pour Node.js utilisé pour construire des API RESTful.
- **Sequelize** : Un ORM (Object-Relational Mapping) pour interagir avec la base de données MySQL.
- **JSON Web Tokens (JWT)** : Utilisé pour l'authentification sécurisée des utilisateurs.
- **dotenv** : Pour gérer les variables d'environnement.

## 3. Base de données : MySQL

- **MySQL** : Un système de gestion de base de données relationnelle utilisé pour stocker les données des utilisateurs, des livres et des prêts.

# ARCHITECTURE DE L'APPLICATION

## 1. Diagramme de l'architecture de l'app



## 2.Schéma de la base de données

- tableau : users

USERS
ID (PK)
NAME
EMAIL (UNIQUE)
PASSWORD
ROLE

- Table des Utilisateurs (Users)

Cette table contient les informations des utilisateurs de l'application. Chaque utilisateur a un identifiant unique (id), un nom (nom), une adresse email unique (email), un mot de passe (motDePasse) et un rôle (role) qui peut être soit 'utilisateur' soit 'administrateur'.

- **tableau : loans**

LOANS
ID (PK)
USERID (FK)
BOOKID (FK)
LOANDATE
RETURNDATE

- **Table des Prêts (Loans)**

Cette table enregistre les informations sur les prêts de livres. Chaque prêt a un identifiant unique (id), un identifiant d'utilisateur (idUtilisateur) qui fait référence à un utilisateur dans la table Users, un identifiant de livre (idLivre) qui fait référence à un livre dans la table Books, une date d'emprunt (dateEmprunt) et une date de retour prévue (dateRetour).

- **tableau : books**

BOOKS
ID (PK)
TITLE
AUTHOR
PUBLICATIONYEAR
GENRE
SUMMARY
AVAILABLE

- **Table des Livres (Books)**

Cette table stocke les informations des livres disponibles dans la bibliothèque. Chaque livre a un identifiant unique (id), un titre (titre), un auteur (auteur), une année de publication (anneePublication), un genre (genre), un résumé (resume) et un statut de disponibilité (disponible).

# RÉALISATION TECHNIQUE

## 1.Détails sur l'implémentation de l'app

- **Gestion des utilisateurs**

### 1.inscription :

Pour l'inscription, un utilisateur envoie une requête POST à **/api/users/register** avec son nom, email et mot de passe. Le mot de passe est haché avant d'être stocké dans la base de données.

```
// controllers/user.controller.js
const bcrypt = require('bcrypt');
const User = require('../models/user.model.js');

exports.register = async (req, res) => {
  try {
    const { nom, email, motDePasse } = req.body;
    const hashedPassword = await bcrypt.hash(motDePasse, 10);
    const user = new User({ nom, email, motDePasse: hashedPassword, role: 'utilisateur' });
    await user.save();
    res.status(201).send({ message: "Utilisateur enregistré avec succès." });
  } catch (error) {
    res.status(500).send({ message: error.message });
  }
};
```

## 2.Connexion

Pour la connexion, un utilisateur envoie une requête POST à /api/users/login avec son email et son mot de passe. Si les informations sont correctes, un token JWT est généré et renvoyé.

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const User = require('../models/user.model.js');

exports.login = async (req, res) => {
    try {
        const { email, motDePasse } = req.body;
        const user = await User.findOne({ where: { email } });
        if (!user || !(await bcrypt.compare(motDePasse, user.motDePasse))) {
            return res.status(401).send({ message: "Email ou mot de passe incorrect." });
        }
        const token = jwt.sign({ id: user.id, role: user.role }, process.env.ACCESS_TOKEN);
        res.send({ token });
    } catch (error) {
        res.status(500).send({ message: error.message });
    }
};
```

## 3.Mise à jour des informations

Un utilisateur peut mettre à jour ses informations via une requête PUT à /api/users/:id.

```
exports.update = async (req, res) => {
    try {
        const { id } = req.params;
        const { nom, email } = req.body;
        await User.update({ nom, email }, { where: { id } });
        res.send({ message: "Utilisateur mis à jour avec succès." });
    } catch (error) {
        res.status(500).send({ message: error.message });
    }
};
```

# • Gestion des livres

## 1. Ajout de livres

Les administrateurs peuvent ajouter des livres via une requête POST à /api/books.

```
const Book = require('../models/book.model.js');

exports.create = async (req, res) => {
    try {
        const { titre, auteur, anneePublication, genre, resume, disponible } = req.body;
        const book = new Book({ titre, auteur, anneePublication, genre, resume, disponible });
        await book.save();
        res.status(201).send(book);
    } catch (error) {
        res.status(500).send({ message: error.message });
    }
};
```

## 2. Mise à jour et suppression de livres

Les administrateurs peuvent mettre à jour et supprimer des livres via les requêtes PUT et DELETE à /api/books/:id.

```
exports.update = async (req, res) => {
    try {
        const { id } = req.params;
        const { titre, auteur, anneePublication, genre, resume, disponible } = req.body;
        await Book.update({ titre, auteur, anneePublication, genre, resume, disponible },
            res.send({ message: "Livre mis à jour avec succès." });
    } catch (error) {
        res.status(500).send({ message: error.message });
    }
};

exports.delete = async (req, res) => {
    try {
        const { id } = req.params;
        await Book.destroy({ where: { id } });
        res.send({ message: "Livre supprimé avec succès." });
    } catch (error) {
        res.status(500).send({ message: error.message });
    }
};
```

## • Gestion des prêts

### 1. Crédation de prêts

Les utilisateurs peuvent emprunter des livres via une requête POST à /api/loans.

```
const Loan = require('../models/loan.model.js');

exports.create = async (req, res) => {
    try {
        const { idUtilisateur, idLivre, dateEmprunt, dateRetour } = req.body;
        const loan = new Loan({ idUtilisateur, idLivre, dateEmprunt, dateRetour });
        await loan.save();
        res.status(201).send(loan);
    } catch (error) {
        res.status(500).send({ message: error.message });
    }
};
```

### 2. Suivi et retour de prêts

Les administrateurs peuvent mettre à jour et supprimer des livres via les requêtes PUT et DELETE à /api/books/:id.

```
exports.findAll = async (req, res) => {
    try {
        const loans = await Loan.findAll();
        res.send(loans);
    } catch (error) {
        res.status(500).send({ message: error.message });
    }
};
```

## 2. Explication des choix techniques

### Explication des choix techniques:

- **React.js pour le frontend :** React offre une manière efficace de créer des interfaces utilisateur interactives. Son approche basée sur les composants permet une réutilisabilité et une maintenance du code plus faciles.
- **Node.js/Express.js pour le backend :** Node.js permet de gérer un grand nombre de requêtes simultanées avec une faible latence, tandis qu'Express.js simplifie la création d'API RESTful.
- **MySQL pour la base de données :** MySQL est une base de données relationnelle robuste, idéale pour gérer les relations complexes entre les utilisateurs, les livres et les prêts.
- **JWT pour l'authentification :** Les tokens JWT (JSON Web Tokens) offrent un moyen sécurisé et stateless d'authentifier les utilisateurs, réduisant la charge du serveur et améliorant la sécurité.

## 2. Explication des choix techniques

### 1. Liaison entre backend et frontend

Lors du développement de l'application, le backend et le frontend ont été développés séparément. Cette séparation a entraîné des défis lors de leur intégration.

#### Défis rencontrés :

- **Incompatibilité des API :** Les endpoints du backend n'étaient pas toujours alignés avec les requêtes du frontend, entraînant des erreurs de communication et des données incorrectes.

- **CORS (Cross-Origin Resource Sharing)** : Les requêtes du frontend vers le backend étaient souvent bloquées par les politiques de CORS, empêchant la communication entre les deux.
- **Gestion des états** : Synchroniser les états entre le frontend et le backend était compliqué, particulièrement pour les actions asynchrones comme les emprunts de livres.

## 2. Gestion des autorisations et des rôles utilisateurs

Un autre défi majeur a été de gérer les autorisations et les rôles des utilisateurs (administrateurs vs utilisateurs réguliers).

### Défis rencontrés :

- **Sécurité des endpoints** : Assurer que seuls les administrateurs puissent accéder à certaines routes sensibles comme l'ajout ou la suppression de livres.
- **Gestion des rôles utilisateurs** : Différencier les actions disponibles pour les utilisateurs en fonction de leur rôle.

## 3. Gestion des contraintes de clé étrangère dans la table des prêts (loans)

La gestion des relations entre les livres et les prêts a présenté des difficultés, surtout lorsque des livres devaient être supprimés.

### Défis rencontrés :

- **Contrainte de clé étrangère** : Lorsqu'un livre lié à un prêt devait être supprimé, la contrainte de clé étrangère dans la table des prêts empêchait la suppression du livre.

- **Maintien de l'intégrité des données** : Assurer que la suppression d'un livre ne laisse pas d'enregistrements orphelins dans la table des prêts, tout en maintenant l'intégrité des données et en évitant les pertes de données importantes.

## 4. Collaboration en groupe

Travailler en groupe a présenté son lot de défis, notamment lors de l'intégration de nouvelles fonctionnalités ou de la résolution de problèmes complexes.

### Défis rencontrés :

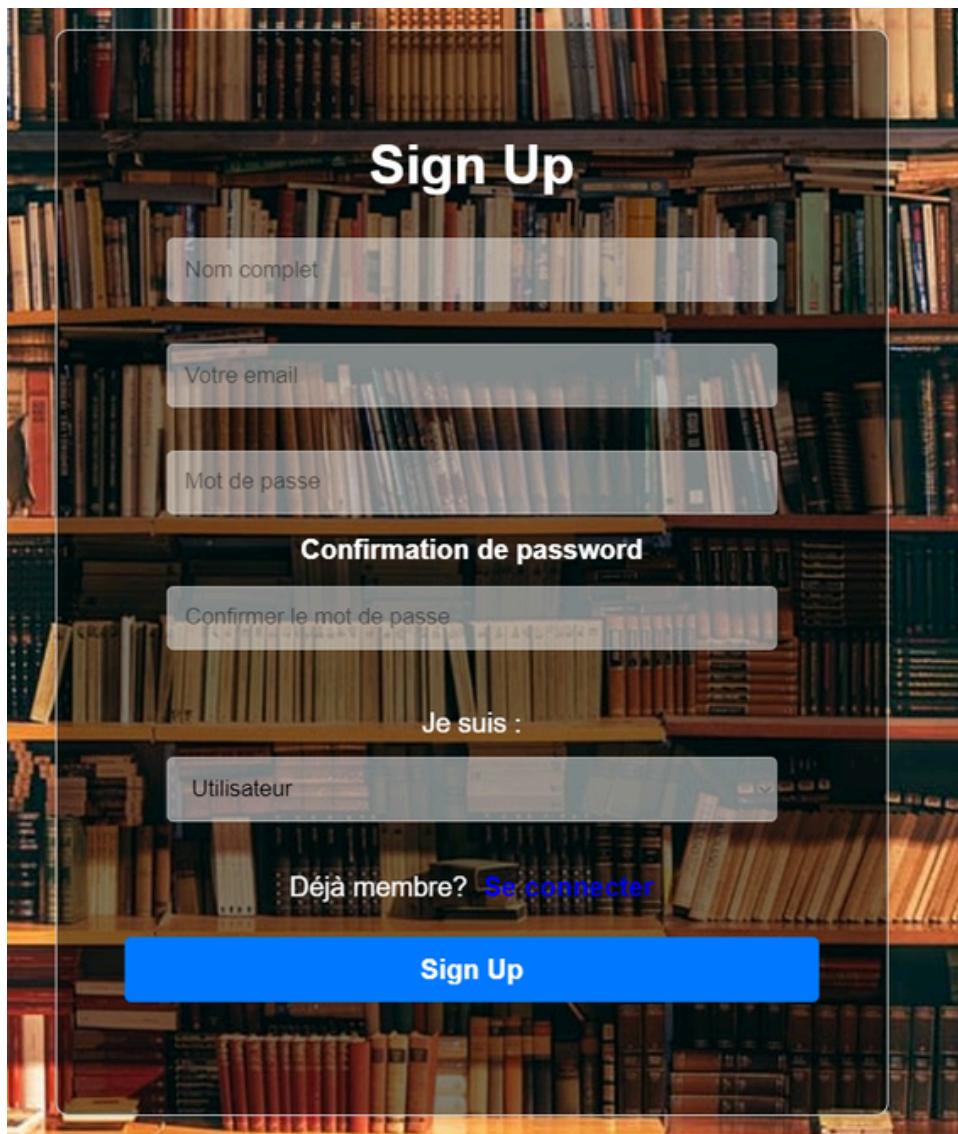
- **Coordination des tâches** : Assurer une répartition équitable des tâches entre les membres du groupe tout en maintenant une cohérence dans le code et une progression régulière du projet.
- **Gestion des conflits** : Les divergences d'opinions ou les approches différentes pour résoudre un problème ont parfois mené à des conflits. Il était nécessaire de trouver des compromis et des solutions qui convenaient à tous.
- **Communication efficace** : Maintenir une communication claire et régulière entre les membres du groupe était essentiel pour éviter les malentendus et les retards dans le développement.

## 5. Liaison entre la page de connexion (login) et la page d'inscription (signup)

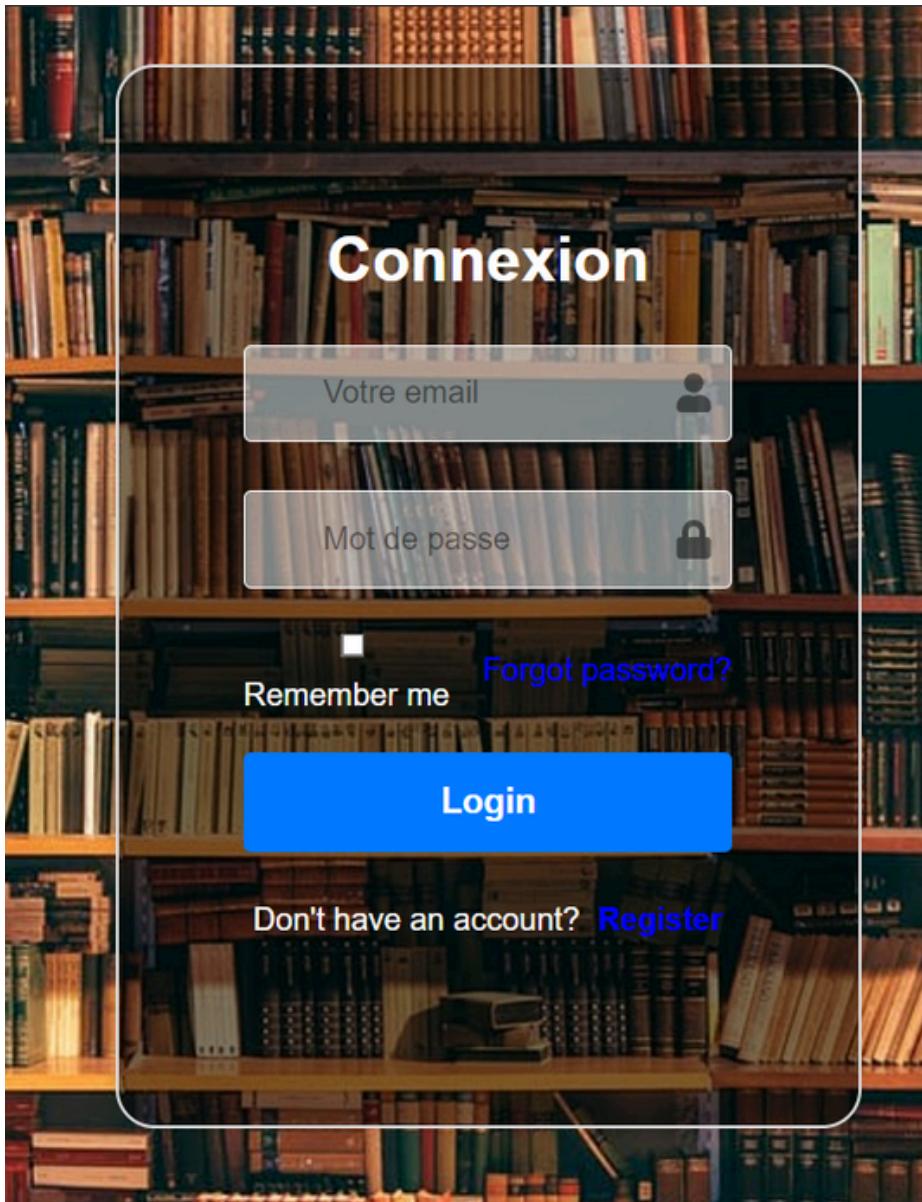
Lors du développement de l'application, nous avons rencontré des difficultés pour assurer une transition fluide entre la page de connexion et la page d'inscription. Il était essentiel de permettre aux utilisateurs de passer facilement de l'une à l'autre en fonction de leurs besoins.

# DÉMONSTRATION DE L'APPLICATION

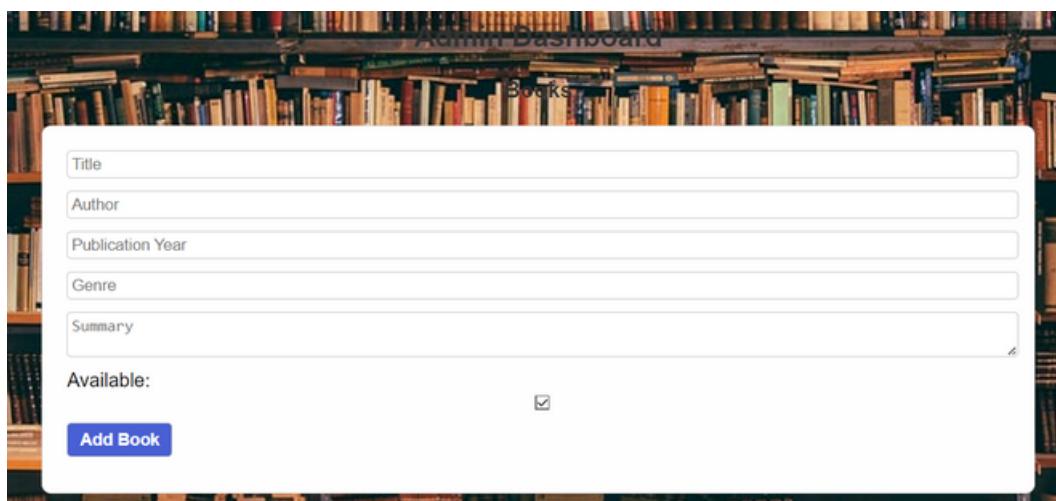
## 1.page de sign-up:



## 2.page de login:



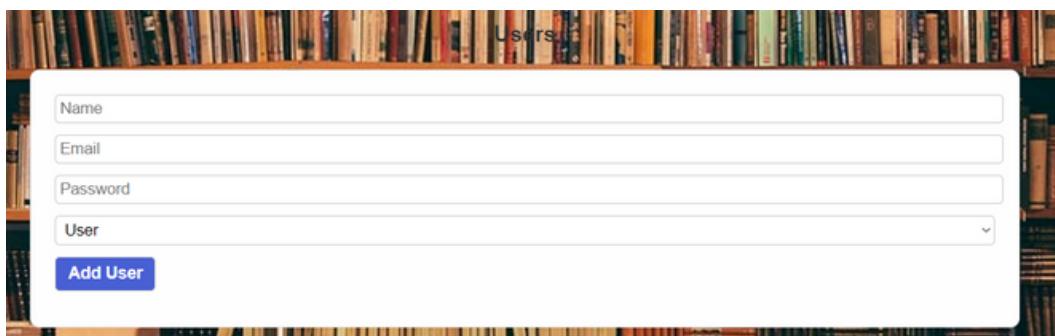
## 3.ajouter un livre :

A screenshot of an admin dashboard titled "Admin Dashboard" showing a "Books" section. The interface has a dark header and a light-colored body. It features a form for adding a new book. The fields are: "Title", "Author", "Publication Year", "Genre", "Summary", and "Available:" with a checked checkbox. A blue "Add Book" button is at the bottom left of the form.

## 4. la liste des livres :

Title	Author	Publication Year	Genre	Summary	Actions	
Book Title	Author Name	2021	Fiction	A brief summary of the book	<button>Edit</button>	<button>Delete</button>
the hobbit	hhh	1021	comedy	ddd	<button>Edit</button>	<button>Delete</button>
arsenal	artita	2022	comedy	h	<button>Edit</button>	<button>Delete</button>
Westham United	David Moise	2002	Football	Bowen	<button>Edit</button>	<button>Delete</button>
the hobbit	hhh	2020	comedy	hzhz	<button>Edit</button>	<button>Delete</button>
the hobbit	med	2022	comedy	hah	<button>Edit</button>	<button>Delete</button>
Sopranos	pep	5858	Fiction	jh	<button>Edit</button>	<button>Delete</button>
everton	chondaych	2003	Football	haha	<button>Edit</button>	<button>Delete</button>
the borrowers	dfghj	2000	dsfd	csgb	<button>Edit</button>	<button>Delete</button>

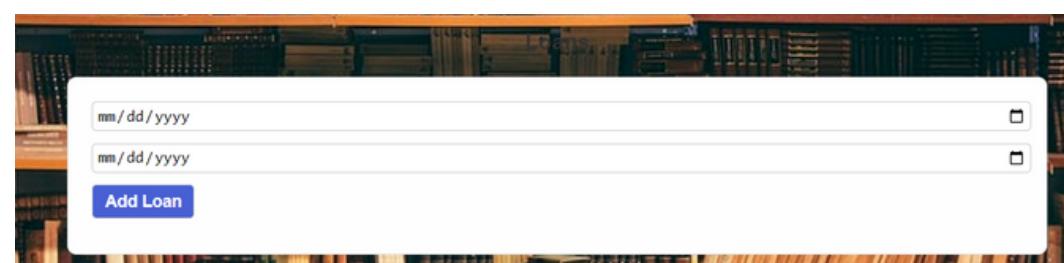
## 5. ajouter un utilisateur :



A form titled "Users" for adding a new user. It contains four input fields: "Name", "Email", "Password", and "User". Below the "User" field is a dropdown menu. At the bottom is a blue "Add User" button.

Name  
Email  
Password  
User

## 6. ajouter un prete :



A form for adding a loan. It has two date input fields labeled "mm/dd/yyyy" and "mm/dd/yyyy" with small calendar icons to their right. At the bottom is a blue "Add Loan" button.

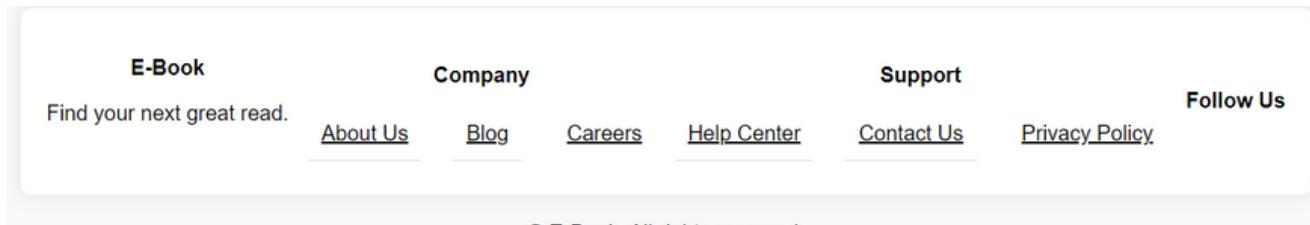
## 7. la liste des utilisateurs :

Name	Email	Password	Role	Actions	
mohamed	med@gmail.com	med	administrateur	<button>Edit</button>	<button>Delete</button>
anass	anass@gmail.com	1515	utilisateur	<button>Edit</button>	<button>Delete</button>
khalid	abourihamid93@gmail.com	555	utilisateur	<button>Edit</button>	<button>Delete</button>
John Doe	john.doe@example.com	password123	utilisateur	<button>Edit</button>	<button>Delete</button>
khalid	aminemouhdade07@gmail.com	mm	utilisateur	<button>Edit</button>	<button>Delete</button>
smail	smail@gmail.com	smail	utilisateur	<button>Edit</button>	<button>Delete</button>
mohamed	hamid93@gmail.com	123456	utilisateur	<button>Edit</button>	<button>Delete</button>
simo	simo@gmail.com	simo	utilisateur	<button>Edit</button>	<button>Delete</button>
anass obi	anassobi@gmail.com	123	utilisateur	<button>Edit</button>	<button>Delete</button>
hamid	hamid@gmail.com	123	utilisateur	<button>Edit</button>	<button>Delete</button>
moussaoui	moussaoui@gmail.com	moussaoui	utilisateur	<button>Edit</button>	<button>Delete</button>

## 8. liste pretes :

Your Loans		
Book	Due On	Actions
arsenal	2024-06-20T00:00:00.000Z	<button>Return</button>
the hobbit	2024-06-21T00:00:00.000Z	<button>Return</button>

## 9. Footer:



# 10.page home:

E-Book      Home      Featured      Discount      New Books      Log Out

Browse & select E-Books

Find the best e-books from your favorite writers.

Explore Now

DIANA & JACK  
CHRISTMAS HAS COME  
Safe the Date  
TERE & TONY  
CELTIC TALES  
"The adventures of the Warrior".

# 11.page livres disponible:

User Dashboard

Available Books

Book Title	Author Name	Genre	Action
	JRR TOLKIEN	comedy	Loan
	JRR TOLKIEN	comedy	Loan
	Sopranos	Fiction	Loan
	L'Horloger d'Everton		Loan
	the hobbit	comedy	Loan
	the hobbit	comedy	Loan
	the borrowers		Loan
	everton	Football	Loan

# CONCLUSION

En conclusion, le développement de l'application de gestion de bibliothèque a été une expérience enrichissante qui nous a permis d'appliquer nos connaissances en programmation web et de relever divers défis techniques. Nous tenons à remercier chaleureusement notre **professeur Fouad** pour son soutien et ses conseils tout au long de ce projet.

Grâce à une approche méthodique et à une collaboration efficace, nous avons réussi à créer une application fonctionnelle et conviviale pour la gestion des bibliothèques.

L'utilisation de technologies telles que React.js pour le frontend, Node.js/Express.js pour le backend, et MySQL pour la base de données s'est avérée être un choix judicieux, offrant une combinaison robuste et évolutive pour notre application.

Les tests rigoureux que nous avons effectués ont permis de garantir la qualité de l'application et d'identifier et de corriger les problèmes potentiels.

En fin de compte, nous sommes fiers du travail accompli et convaincus que notre application répondra aux besoins de gestion des bibliothèques de manière efficace et intuitive. Nous espérons que cette application servira de base solide pour d'autres projets similaires à l'avenir.

**FIN.**