Faculté des Sciences Agadir
Centre d'excellence IT
Filière : ADIA

جامعة ابن زهر
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵓⵏ ⵣⵓⵀⵔ
UNIVERSITÉ IBN ZOHR

Projet de Distributed System
Sous le thème :

Système de Gestion de Bibliothèque Distribué

Réalisé par :

- BOULAADAM Aimad
- CHARAHIL Adil
- BAIHA Fatima Ezzahrae
- MOUHIB Zakaria

Sous la supervision de :

- Dr. Youssef El Habouz

Année Universitaire : 2025 – 2026

Liste des figures

Chapitre I : CONTEXTE GÉNÉRAL DU PROJET

I. Contexte du projet :

L'informatisation des services est devenue une nécessité incontournable pour les institutions modernes, et les bibliothèques ne font pas exception. Historiquement, la gestion des bibliothèques reposait sur des systèmes manuels ou des applications monolithiques isolées, installées sur un poste unique. Cependant, avec l'augmentation du volume des ouvrages, la diversification des utilisateurs et la nécessité d'un accès à distance en temps réel, ces solutions traditionnelles montrent leurs limites.

Le contexte actuel exige des architectures logicielles plus robustes, capables de séparer l'interface utilisateur de la logique métier et des données. C'est dans ce cadre que s'inscrit l'informatique distribuée. Elle permet de répartir les traitements sur plusieurs machines ou processus, offrant ainsi une meilleure performance, une plus grande tolérance aux pannes et une scalabilité accrue. Ce projet s'insère donc dans une démarche de modernisation des systèmes d'information, en exploitant des protocoles de communication avancés pour répondre aux exigences de rapidité et de fiabilité.

II. Présentation ou description du projet

Ce projet consiste en la conception et le développement d'un **Système de Gestion de Bibliothèque Distribué (DLMS – Distributed Library Management System)**. Il s'agit d'une solution logicielle structurée selon une **architecture Client-Serveur**, dans laquelle les différentes composantes communiquent entre elles via un réseau.

Concrètement, l'application se compose de **deux parties principales** :

Le Serveur (Backend) : Développé en **Python** avec le framework **Django**, le serveur centralise la **logique métier** du système, incluant la gestion des règles d'emprunt, l'authentification des utilisateurs et le suivi des transactions. Il interagit avec une **base de données relationnelle MySQL** afin d'assurer la persistance et la cohérence des informations relatives aux livres, aux utilisateurs et aux historiques d'emprunt.

Le Client (Frontend/Interface utilisateur) : Cette interface permet aux bibliothécaires ainsi qu'aux adhérents d'interagir facilement avec le système, en effectuant des opérations telles que la consultation, l'emprunt ou le retour des ouvrages. L'interface est conçue pour être intuitive et réactive, facilitant la gestion quotidienne de la bibliothèque.

La particularité technique majeure de ce projet réside dans l'utilisation de **gRPC (gRPC Remote Procedure Call)** comme **middleware de communication** entre le client et le serveur. Contrairement aux API REST traditionnelles, gRPC s'appuie sur le format binaire **Protocol Buffers**, permettant des **échanges de données rapides, sécurisés et fortement typés**, ainsi qu'une meilleure performance globale pour la communication inter-processus.

Ainsi, le projet DLMS combine à la fois **robustesse technique, modularité et expérience utilisateur fluide**, tout en illustrant les avantages d'une architecture distribuée moderne pour la gestion d'une bibliothèque.

III. Problématique

La mise en place d'un système de gestion pour une bibliothèque universitaire ou publique soulève plusieurs défis, tant fonctionnels que techniques. Les architectures classiques (monolithiques) souffrent souvent de problèmes de performance lorsqu'elles sont sollicitées par de multiples utilisateurs simultanés.

La problématique centrale de ce projet peut se formuler ainsi :

Comment concevoir une architecture logicielle distribuée capable de gérer efficacement les opérations d'une bibliothèque tout en garantissant la rapidité des échanges, la cohérence des données et l'indépendance entre l'interface utilisateur et le serveur de traitement ?

De cette problématique centrale découlent plusieurs sous-questions :

- Comment assurer une communication performante et à faible latence entre le client et le serveur distant ?
- Comment gérer la concurrence d'accès (par exemple, éviter que deux utilisateurs réservent le même livre au même moment) ?
- Comment structurer les données pour permettre une évolution facile du système (ajout de nouvelles fonctionnalités sans casser l'existant) ?

IV. Objectifs du Projet

Pour répondre à la problématique posée, ce projet vise plusieurs objectifs, classés en objectifs techniques et fonctionnels :

1. Objectifs Principaux (Techniques)

- **Implémentation d'une architecture distribuée :** Mettre en œuvre une séparation stricte entre le client et le serveur.
- **Utilisation de RPC moderne :** Maîtriser et intégrer la technologie **gRPC** et les **Protocol Buffers** pour la communication inter-processus.
- **Persistance des données :** Concevoir une base de données relationnelle (**MySQL**) optimisée pour stocker les catalogues et les historiques d'emprunt.
- **Gestion des erreurs :** Assurer la robustesse du système face aux problèmes réseaux ou aux mauvaises saisies utilisateurs.

2. Objectifs Fonctionnels (Métier)

- **Gestion des Ouvrages (CRUD) :** Permettre l'ajout, la modification, la suppression et la consultation des livres.
- **Gestion des Utilisateurs :** Gérer l'inscription et l'authentification sécurisée (différenciation entre administrateurs et utilisateurs simples).
- **Gestion des Emprunts :** Permettre l'emprunt et le retour de livres, avec vérification automatique de la disponibilité et des stocks.
- **Recherche :** Offrir un moteur de recherche performant pour retrouver des ouvrages par titre, auteur ou ISBN.

Chapitre II : ANALYSE ET CONCEPTION

I. Introduction :

L'étape d'analyse et de conception est fondamentale dans le cycle de vie de développement d'un logiciel, particulièrement pour les systèmes distribués qui impliquent une complexité architecturale accrue. Ce chapitre a pour but de formaliser les besoins des utilisateurs du système de gestion de bibliothèque et de définir l'architecture technique qui y répondra.

Nous commencerons par identifier les besoins fonctionnels et non fonctionnels, avant de présenter la conception détaillée du système à travers des diagrammes UML (Unified Modeling Language) et la structure de la base de données. Cette démarche garantit que la solution développée avec **gRPC** et **Django** est cohérente, performante et adaptée aux attentes.

II. SPECIFICATION DES BESOINS

La spécification des besoins permet de définir le périmètre du projet. Nous distinguons ici ce que le système doit *faire* (fonctionnel) de *comment* il doit le faire (non-fonctionnel).

a) Identification des acteurs

Le système interagit avec deux types d'acteurs principaux :

- **L'Administrateur / Bibliothécaire :** Il gère le fond documentaire et les utilisateurs. Il possède des droits d'accès étendus.
- **L'Utilisateur (Adhérent/Étudiant) :** Il consulte le catalogue et effectue des demandes d'emprunt.

b) Besoins fonctionnelles

Le système doit fournir les fonctionnalités suivantes :

- **Gestion des Utilisateurs (Authentification) :**
  - Inscription sécurisée des nouveaux membres.
  - Authentification via identifiant et mot de passe.
  - Gestion des rôles (Admin vs Utilisateur standard).
- **Gestion du Catalogue (CRUD Livres) :**
  - Ajout de nouveaux ouvrages (Titre, Auteur, ISBN, Quantité).

    - Mise à jour des informations d'un livre.
    - Suppression d'un ouvrage obsolète.
- **Opérations de Bibliothèque :**
    - Recherche d'ouvrages par critères (Mots-clés, Auteur).
    - Consultation de la disponibilité en temps réel.
    - Enregistrement d'un emprunt.
    - Retour d'un livre et mise à jour du stock.

c) Besoins non fonctionnelles :

Ces besoins justifient le choix de l'architecture distribuée et de gRPC :

- **Performance et Latence :** Les échanges entre le client et le serveur doivent être rapides, d'où l'utilisation du format binaire **Protocol Buffers** (plus léger que JSON).
- **Interopérabilité :** Le système doit pouvoir faire communiquer des composants potentiellement écrits dans des langages différents (ex: Client Python, Serveur Python/Django).
- **Scalabilité :** L'architecture doit permettre de gérer une montée en charge du nombre de requêtes simultanées.
- **Fiabilité :** Le système doit gérer les erreurs de connexion réseau sans planter (gestion des exceptions gRPC).

III. CONCEPTION :

Cette section traduit les besoins en une solution technique concrète.

a) Architecture Globale du Système

Nous avons opté pour une **architecture distribuée Client-Serveur** basée sur le framework RPC (Remote Procedure Call).

- **Communication :** Assurée par **gRPC**. Les méthodes sont définies dans des fichiers .proto qui servent de contrat d'interface.
- **Sérialisation :** Les données sont sérialisées en **Protobuf**, garantissant une taille de message minimale sur le réseau.
- **Backend :** Le serveur gRPC est couplé au framework **Django**. Django fournit l'ORM (Object-Relational Mapping) pour manipuler la base de données de manière abstraite et sécurisée.

b) Conception UML (Unified Modeling Language)

1. Diagramme de Cas d'Utilisation (Use Case Diagram)

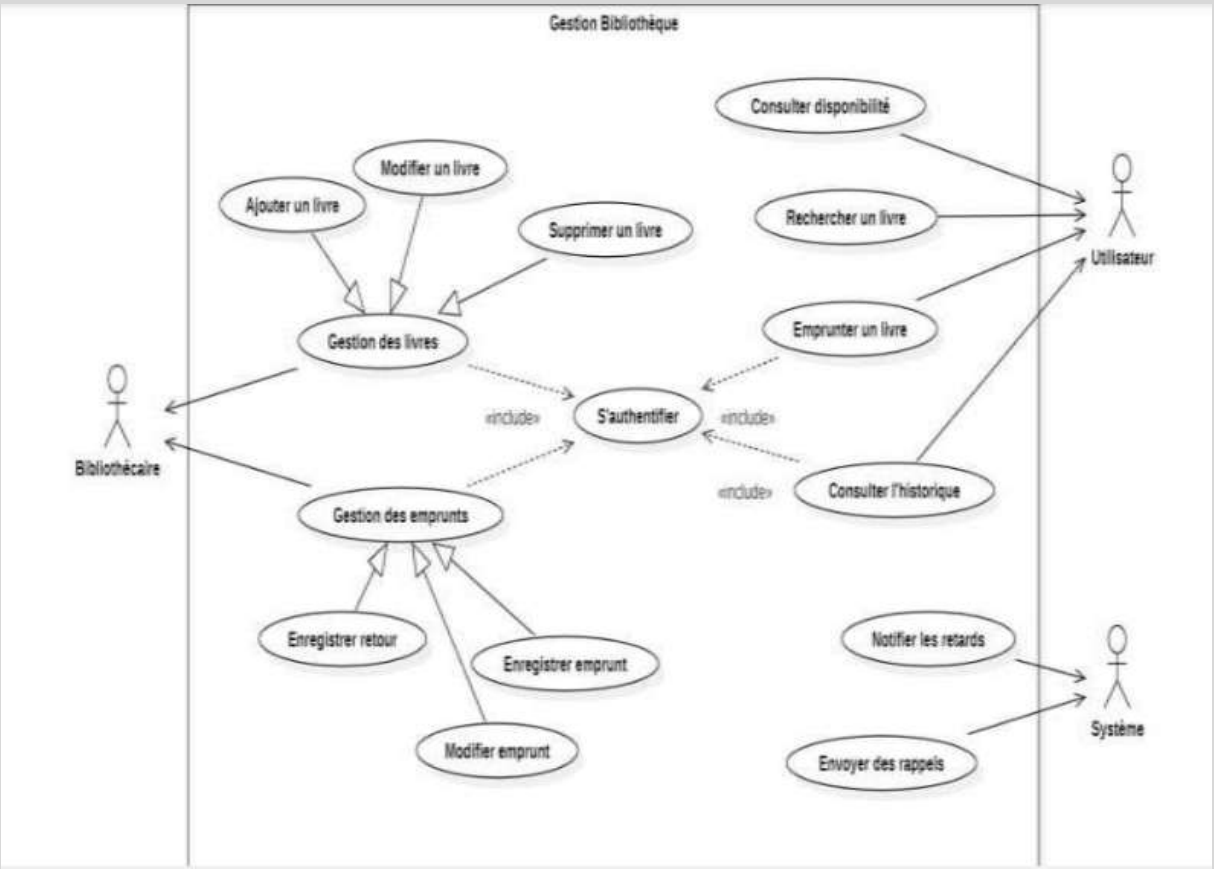

Figure 1 : Diagramme de Cas d'Utilisation

2. Diagramme de Classes (Class Diagram)

Ce diagramme représente la structure statique des données. Dans notre système Django/MySQL, les classes principales sont :

- **Book (Livre) :** isbn, title, author, total_copies, available_copies.
- **User (Utilisateur) :** username, password, email, role.
- **Transaction (Emprunt) :** user_id, book_id, borrow_date, return_date, status.

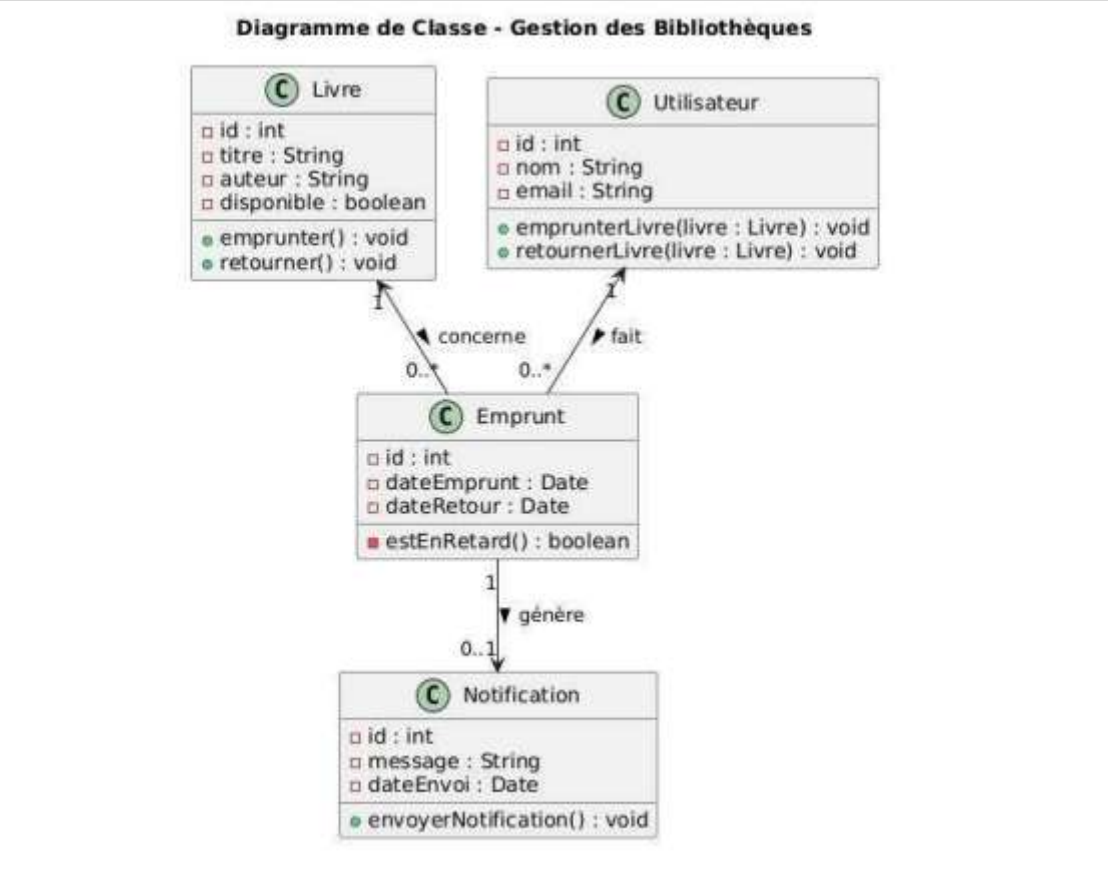

Figure 2 : Diagramme de Classes

3. Diagramme de Séquence (Sequence Diagram) - Focus sur l'Emprunt

Ce diagramme détaille la logique d'un appel distant gRPC pour un emprunt :

1. Le **Client** appelle la méthode BorrowBook(stub).
2. Le **Stub** sérialise la requête en Protobuf.
3. Envoi via le réseau.
4. Le **Serveur** reçoit, désérialise et interroge la **BDD (MySQL)**.
5. Le Serveur renvoie TransactionStatus.

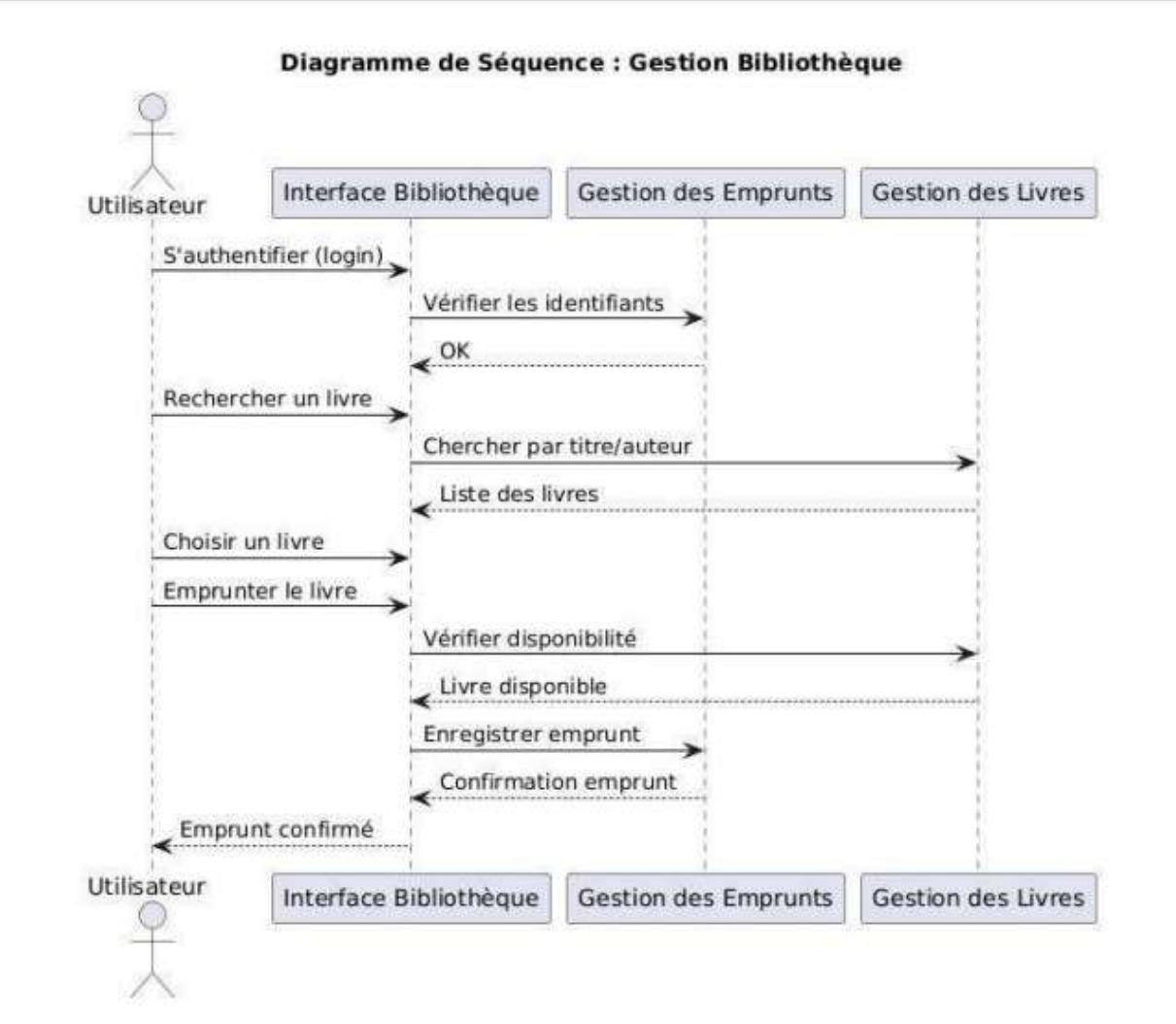

Figure 3 : Diagramme de Séquence (Scénario d'Emprunt)

c) Conception de la Base de Données (Modèle Relationnel) :

La persistance des données est assurée par le SGBD **MySQL**. Le schéma relationnel découlant du diagramme de classes est le suivant :

- **Table library_book :** Clé primaire sur ISBN.
- **Table auth_user :** Gérée nativement par le système d'authentification de Django.
- **Table library_transaction :** Contient des clés étrangères vers library_book et auth_user pour lier un emprunt à un livre et une personne.

Chapitre III : ENVIRONNEMENT DE TRAVAIL

I. Introduction :

La réalisation d'un système distribué complexe nécessite un ensemble d'outils performants, allant du langage de programmation aux logiciels de gestion de base de données, en passant par les environnements de développement intégrés (IDE). Ce chapitre décrit l'environnement matériel et logiciel mis en place pour le développement du projet **"Distributed Library Management System"**, ainsi que les technologies et bibliothèques spécifiques utilisées pour implémenter l'architecture gRPC avec Django..

II. Environnement de travail :

Pour mener à bien ce projet, nous avons sélectionné des logiciels standards de l'industrie, favorisant la productivité et la collaboration.

a) Système d'Exploitation

Le projet est multiplateforme. Il a été développé et testé sous **Windows 10/11** garantissant la portabilité du code Python.

b) IDE (Environnement de Développement Intégré)

Nous avons utilisé **Visual Studio Code (VS Code)** comme éditeur principal.

- **Pourquoi ce choix ?** Sa légèreté, son extensibilité et son excellente intégration avec Python et le terminal intégré ont facilité l'écriture des scripts et l'exécution des commandes de génération de code gRPC.

c) Gestionnaire de Base de Données

- **SGBD :** MySQL Server.
- **Interface d'administration : MySQL Workbench** (ou phpMyAdmin) a été utilisé pour visualiser le schéma de la base de données, vérifier la création des tables par l'ORM de Django et contrôler les transactions enregistrées.

d) Outils de Conception

- **StarUML / Draw.io :** Utilisés pour la réalisation des diagrammes UML (Cas d'utilisation, Classes, Séquence) et du schéma d'architecture présentés au chapitre précédent.

III. Technologies et Bibliothèques Utilisées:

a) Langage de Programmation : Python

Python a été choisi pour sa syntaxe claire et sa richesse en bibliothèques pour le développement web et réseau.

- **Version :** Python 3.x

b) Framework Web : Django

Django est utilisé pour structurer le backend.

- **Rôle :** Il gère l'authentification, les modèles de données (ORM) et la logique métier. Bien que Django soit traditionnellement un framework Web HTTP, nous l'avons adapté pour fonctionner avec un service gRPC.

c) Middleware de Communication : gRPC & Protobuf

C'est le cœur technologique du projet.

- **gRPC (grpcio) :** Framework RPC haute performance développé par Google. Il permet au client d'appeler des méthodes sur le serveur comme si elles étaient locales.
- **Protocol Buffers (protobuf) :** Langage de description d'interface (IDL). Nous avons défini nos services et messages dans des fichiers .proto, qui sont ensuite compilés pour générer le code Python (Stubs et Servicers).
- **Outil de compilation :** grpcio-tools a été utilisé pour compiler les fichiers .proto en code Python exécutable.

d) Connecteur de Base de Données

- **mysqlclient :** Pilote permettant à Django de communiquer avec le serveur MySQL.

IV. Technologies et Bibliothèques Utilisées:

Conformément aux exigences du projet, le travail en équipe a été géré via **GitHub**.

a) Dépôt Centralisé

Un dépôt (repository) GitHub a été créé pour héberger le code source. Il a permis de centraliser les fichiers du projet et de garder une trace de toutes les modifications.

GitHub.

b) Workflow Git

Nous avons adopté une méthodologie collaborative :

- **Branches :** Création de branches spécifiques pour chaque fonctionnalité (ex: feature/auth, feature/book-crud, feature/grpc-setup) afin de ne pas perturber la branche principale (main).
- **Commits :** Des commits réguliers avec des messages descriptifs pour suivre l'avancement.
- **Merge :** Fusion des branches une fois les fonctionnalités testées et validées.

V. Structure du Projet:

L'organisation des fichiers du projet respecte l'architecture Django standard, augmentée des fichiers nécessaires à gRPC.

VI . Conclusion:

L'environnement de travail mis en place, combinant la puissance de Python/Django, la rapidité de gRPC et la rigueur de gestion de version avec GitHub, a permis de développer le projet dans des conditions professionnelles. Cette organisation a facilité l'intégration des différents modules développés par les membres de l'équipe.

Chapitre IV : Réalisation du projet

I. Introduction

Ce chapitre présente les interfaces graphiques finales du **Système de Gestion de Bibliothèque Distribué**. Il illustre le parcours utilisateur à travers les différentes fonctionnalités implémentées, allant de l'authentification à la gestion administrative des livres et des membres. Chaque interface présentée ci-dessous communique avec le serveur backend via le protocole **gRPC**.

II. Module d'Authentification et d'Accès

L'accès au système est sécurisé. Les utilisateurs doivent s'identifier pour accéder aux fonctionnalités, et les nouveaux membres peuvent créer un compte.

1. Inscription (Create New Account)

Cette page permet à un nouvel utilisateur de s'enregistrer dans la base de données. Le formulaire recueille les informations personnelles et les transmet au serveur pour la création du profil.

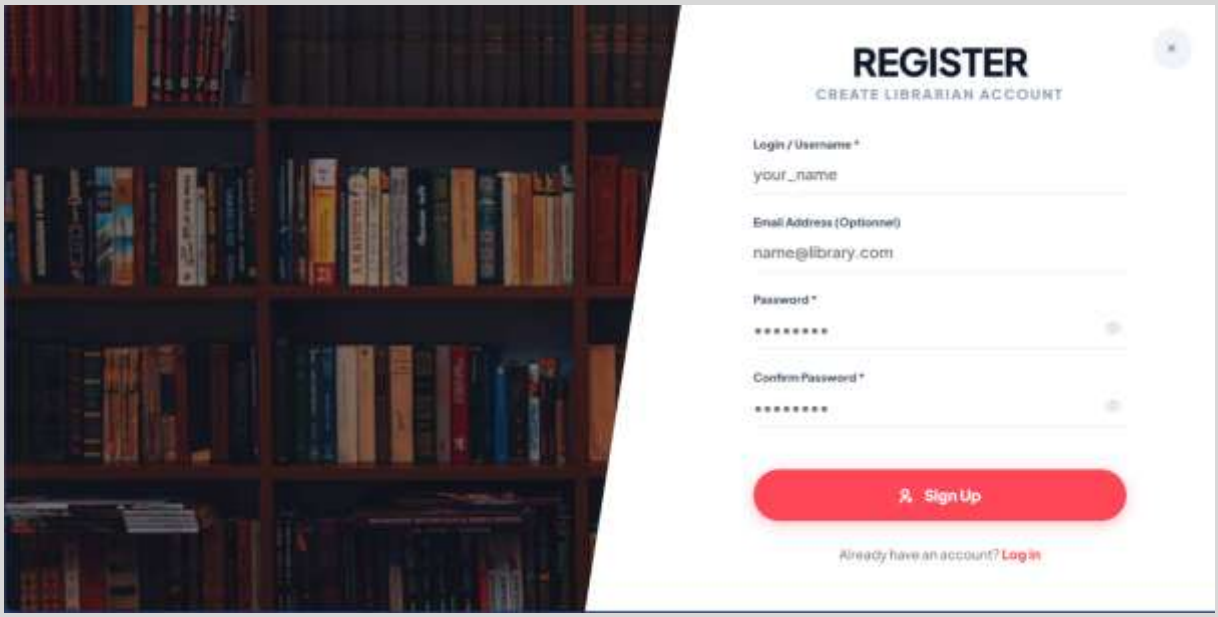

Figure 4 : Page d'inscription (Register)

2. Connexion (Login)

L'interface de connexion sécurise l'entrée au système. Elle vérifie les identifiants (nom d'utilisateur et mot de passe) via une requête d'authentification envoyée au backend Django.

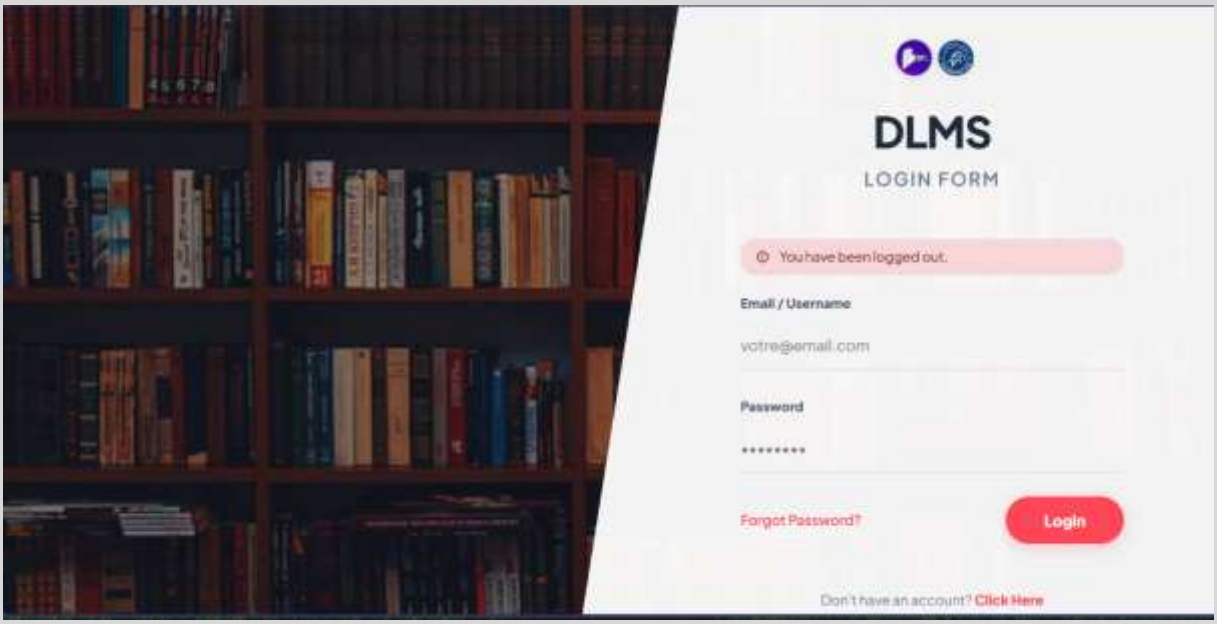

Figure 5: Page de connexion (Login)

III. Tableau de Bord et Navigation

Une fois authentifié, l'utilisateur est redirigé vers le tableau de bord principal.

1. Dashboard

Le tableau de bord offre une vue d'ensemble des activités de la bibliothèque. Il affiche des statistiques clés (nombre de livres, membres actifs, emprunts en cours) et permet une navigation rapide vers les autres modules.

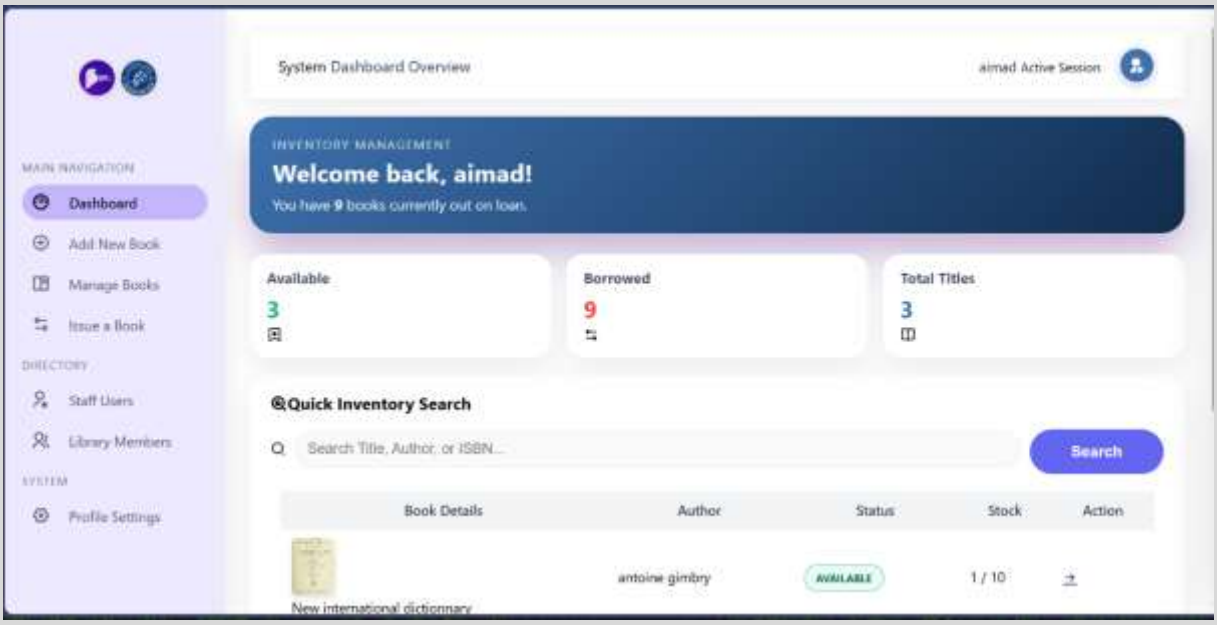

Figure 6 : Tableau de Bord Principal (Dashboard)

IV. Module de Gestion des Ouvrages (Book Management)

Ce module est réservé au personnel (Staff) pour gérer le catalogue de la bibliothèque.

1. Gestion des Livres (Manage Books)

Cette interface liste tous les ouvrages disponibles dans la bibliothèque avec leurs détails (ISBN, Titre, Auteur, Disponibilité). Elle permet de modifier ou supprimer des livres existants.

Figure 7 : Interface de gestion des livres

2. Ajout d'un Nouveau Livre (Add New Book)

Ce formulaire permet aux bibliothécaires d'ajouter une nouvelle référence dans la base de données. Les données saisies sont validées et envoyées au serveur pour mise à jour du stock.

Figure 8 : Formulaire d'ajout de livre

V. Module de Gestion des Utilisateurs

Ce module permet à l'administrateur de gérer les différents acteurs du système.

1. Liste des Utilisateurs Staff (Staff User / List User)

Cette vue présente la liste des administrateurs et employés de la bibliothèque ayant des droits de gestion.

Figure 9 : Liste des utilisateurs du personnel (Staff)

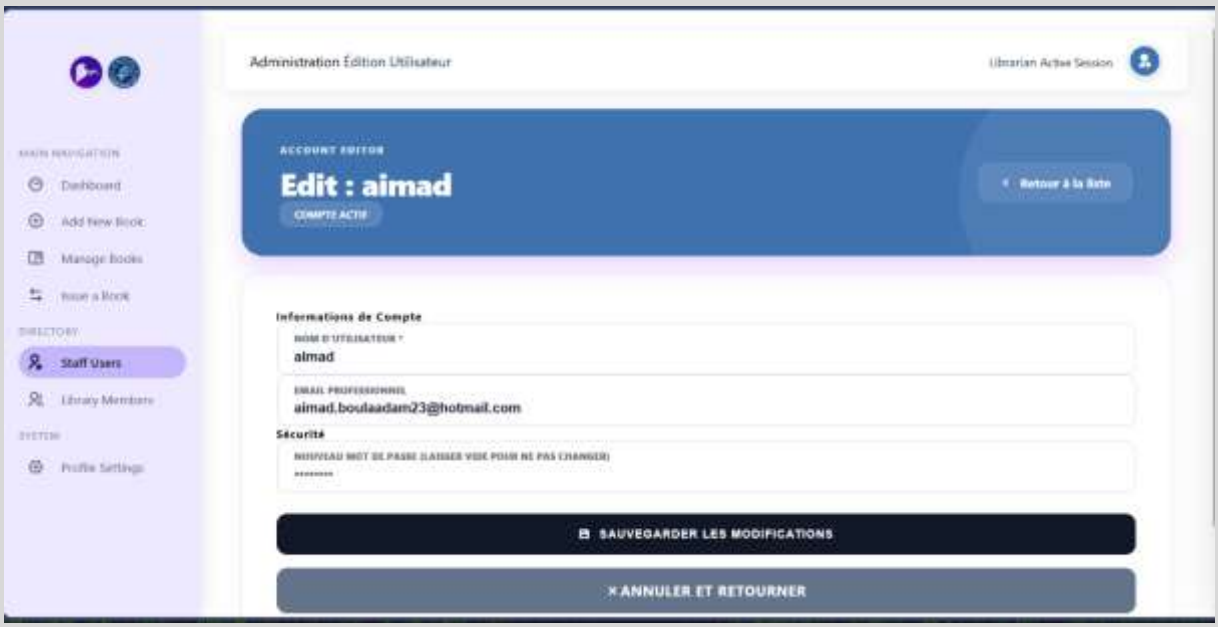

Figure 10 : Edition utilisateurs

2. Gestion des Membres (Library Member / List Member)

Cette interface permet de visualiser et gérer la liste des adhérents (étudiants ou lecteurs) inscrits à la bibliothèque qui ont le droit d'emprunter des livres.

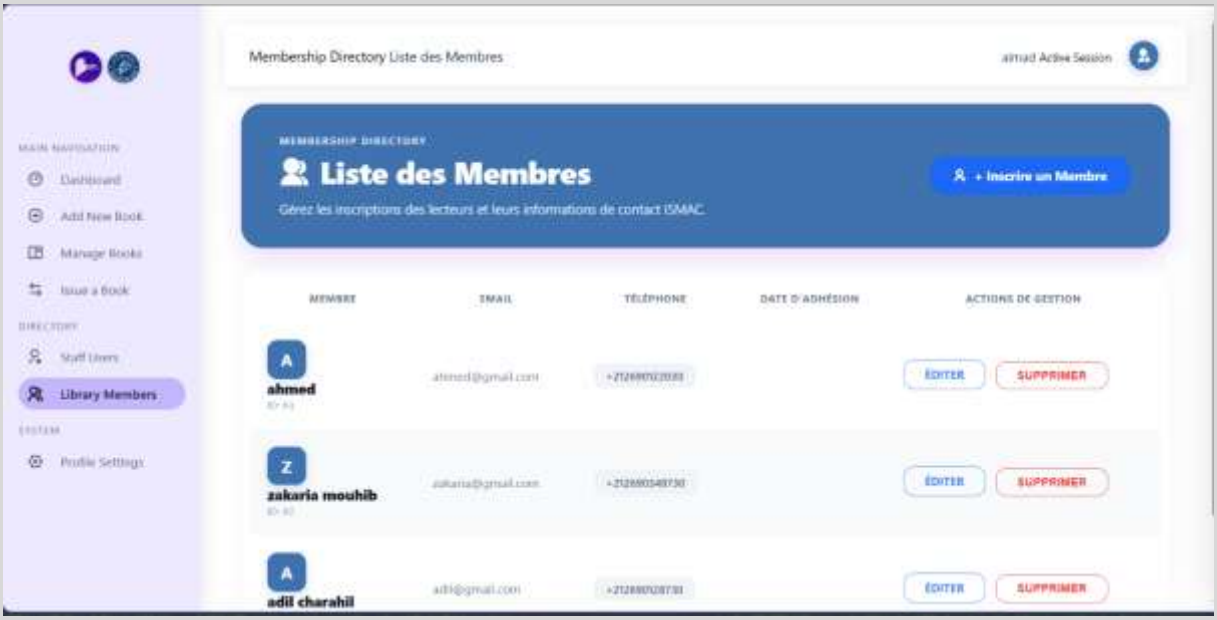

Figure 11 : Liste des membres de la bibliothèque

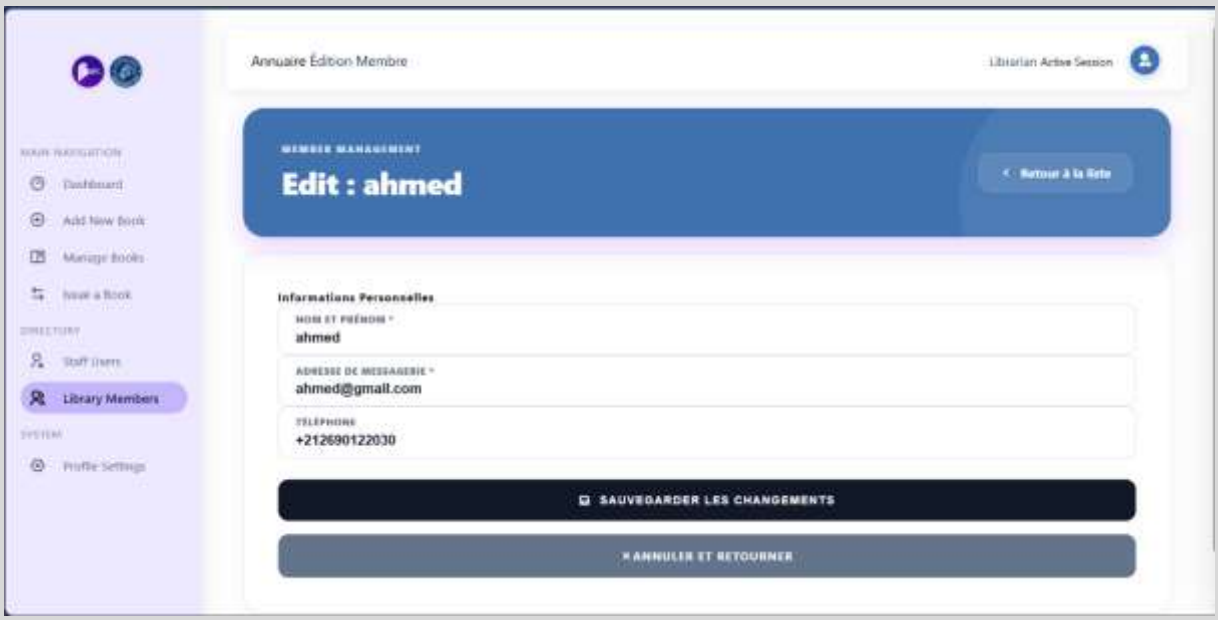

Figure 12 : Edition membres de la bibliothèque

VI. Gestion des Transactions (Emprunts)

C'est le cœur fonctionnel du système distribué.

1. Emprunt de Livre (Issue Book)

Cette interface permet d'enregistrer un nouvel emprunt. L'administrateur sélectionne un membre et un livre. Le système vérifie la disponibilité via gRPC et enregistre la transaction si le stock est suffisant.

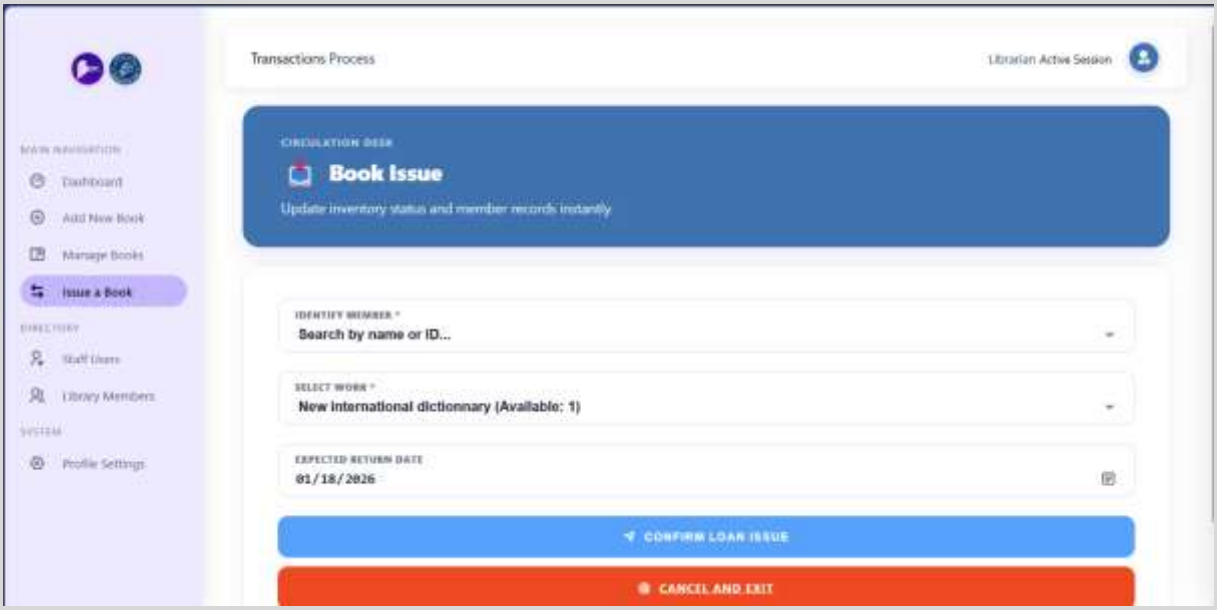

Figure 13 : Interface d'emprunt de livre (Issue Book)

VII. Paramètres du Compte

Chaque utilisateur dispose d'un espace pour gérer ses informations personnelles.

1. Paramètres de Profil (Profile Settings)

Cette page permet à l'utilisateur connecté de modifier ses informations (email, mot de passe, photo de profil).

Figure 14 : Paramètres de profil

VIII. Conclusion

Les interfaces présentées démontrent la couverture fonctionnelle complète du système. L'intégration entre le frontend et le backend distribué est transparente pour l'utilisateur final, offrant une expérience fluide pour la gestion quotidienne de la bibliothèque.

Chapitre VI :CONCLUSION GÉNÉRALE PERSPECTIVES

I. Bilan du Projet

Ce projet avait pour objectif principal la conception et la réalisation d'un **Système de Gestion de Bibliothèque Distribué**, répondant aux exigences modernes de performance et de scalabilité. En nous appuyant sur une architecture distribuée, nous avons cherché à dépasser les limites des applications monolithiques classiques.

Au terme de ce travail, nous avons réussi à implémenter une solution fonctionnelle complète qui sépare distinctement la logique métier (Backend) de l'interface utilisateur (Frontend). L'utilisation du couple technologique **Django** (pour la robustesse de la gestion de données) et **gRPC** (pour la rapidité des échanges réseaux) s'est avérée être un choix architectural pertinent.

Le système final permet de gérer efficacement l'ensemble du cycle de vie d'une bibliothèque :

1. **Gestion administrative :** Ajout et modification des ouvrages et des utilisateurs.
2. **Gestion des transactions :** Suivi fluide des emprunts et des retours avec mise à jour des stocks en temps réel.
3. **Sécurité :** Authentification des membres et contrôle d'accès aux interfaces sensibles.

II. Compétences Acquises

La réalisation de ce projet a été une opportunité d'apprentissage riche pour l'ensemble de l'équipe :

- **Maîtrise des Architectures Distribuées :** Nous avons compris concrètement les mécanismes de communication inter-processus (RPC), la sérialisation des données avec **Protocol Buffers** et les enjeux de la séparation Client-Serveur.
- **Approfondissement Technique :** Nous avons renforcé nos compétences en **Python** et découvert la puissance du framework **gRPC**, qui est aujourd'hui un standard dans l'industrie pour les microservices.
- **Travail Collaboratif :** L'utilisation de **GitHub** nous a permis d'organiser le travail en équipe, de gérer les conflits de code et d'adopter une méthodologie de développement rigoureuse.

III. Limites et Difficultés Rencontrées

Malgré le succès global du projet, nous avons fait face à plusieurs défis :

- **Courbe d'apprentissage de gRPC :** La prise en main de la définition des fichiers .proto et la compréhension du code généré (Stubs/Servicers) ont demandé un temps d'adaptation important par rapport aux API REST classiques.

- **Intégration Django/gRPC :** Faire cohabiter l'ORM de Django (conçu pour le Web synchrone) avec un serveur gRPC a nécessité une configuration spécifique de l'environnement.
- **Interface Graphique :** Bien que fonctionnelle, l'interface actuelle reste basique. L'accent a été mis prioritairement sur la solidité du backend distribué plutôt que sur le design UX/UI avancé.

IV. Perspectives d'Avenir

Ce projet constitue une base solide qui pourrait être étendue et améliorée. Voici quelques pistes d'évolution pour une version future :

1. **Sécurisation des Échanges (SSL/TLS) :** Actuellement, les échanges gRPC sont en clair. Pour une mise en production, il serait impératif de chiffrer la communication entre le client et le serveur.
2. **Déploiement Conteneurisé (Docker) :** Pour faciliter l'installation sur différents serveurs, l'application pourrait être "dockerisée", avec un conteneur pour le serveur Django, un pour la base MySQL et un pour le client.
3. **Version Mobile :** Grâce à l'architecture distribuée, il serait facile de développer une application mobile (Android/iOS) qui se connecte au même serveur gRPC sans avoir à modifier le code du backend.
4. **Système de Notification :** Ajouter un module pour envoyer des emails automatiques aux utilisateurs en cas de retard de retour de livre.

V. Mot de la fin

En conclusion, ce projet de **Distributed Library Management System** nous a permis de mettre en pratique les concepts théoriques des systèmes distribués. Il démontre qu'il est possible de créer des applications performantes et modulaires en utilisant les bons outils de communication à distance. Nous sommes fiers du résultat obtenu, qui répond aux objectifs fixés par le cahier des charges initial.