Rapport de Projet : Système de Gestion de Bibliothèque en Rust

1. Introduction

Le but de ce projet consiste à réaliser une console de gestion de bibliothèque développée en Rust. L'objectif est de permettre l'ajout, l'emprunt, le retour et la consultation de livres à travers une interface en ligne de commande.

2. Architecture du Programme

2.1 Les Structures de Données

Book : Représente un livre avec ses attributs (titre, auteur, année de publication, genre et disponibilité).

Bookshelf : Contient une collection de livres sous forme de vecteur (Vec<Book>).

2.2 Les Traits (Interfaces)

Library : Correspond aux opérations effectuées sur la bibliothèque (ajout, affichage de tous les livres).

OneBook : Correspond aux opérations effectuées sur un livre individuel (emprunter, retourner).

3. Fonctionnalités Implémentées

- **Ajout d'un livre :** Permet d'ajouter un nouveau livre avec vérification des doublons basée sur le titre.
- **Emprunt d'un livre :** Marque un livre comme indisponible s'il est actuellement disponible.
- **Retour d'un livre :** Remet un livre emprunté comme disponible dans la bibliothèque.
- **Affichage de tous les livres :** Liste tous les livres avec leur statut de disponibilité.
- **Affichage des livres disponibles :** Filtre et affiche uniquement les livres disponibles à l'emprunt.

4. Concepts Rust Utilisés

- **Traits :** J'ai utilisé les traits car je trouvais que ça correspondait le mieux pour la création d'une bibliothèque et aussi par rapport au situation de polymorphisme.
- **Pattern Matching :** J'ai utilisé des match pour gérer les choix du menu et la validation des entrées.
- **Gestion des erreurs :** Traitement des erreurs de parsing pour convertir en nombre et unwrap() pour gérer mieux les erreurs.
- **Vecteurs et itérateurs :** Grâce aux méthodes comme iter() et any() j'ai pu manipuler les vecteurs pour pouvoir avoir les valeurs que je voulais.

4.1 Définitions

- **Situation de polymorphisme :** Cas où le même code peut travailler avec différents types, à condition qu'ils implémentent tous le même trait (comportement commun).
- **Erreur de parsing :** Erreur qui se produit lors de la conversion d'une chaîne de caractères vers un type quand le format de la donnée est invalide.
- **unwrap() :** Méthode sur Option et Result qui récupère directement la valeur contenue, mais provoque une erreur fatale si c'est None ou Err.
- **iter() :** Méthode qui crée un itérateur sur les éléments d'une collection sans en prendre possession, permettant de les parcourir facilement.
- **any() :** Méthode d'itérateur qui renvoie true si au moins un élément satisfait une condition donnée, sinon false.

5. Flux du Programme

Le programme fonctionne en boucle infinie (loop) affichant un menu avec 6 options. L'utilisateur saisit son choix via l'entrée standard, puis le programme exécute l'action correspondante. La boucle continue jusqu'à ce que l'utilisateur choisisse de quitter.

6. Conclusion

Ce projet nous a permis de mettre en pratique nos connaissances et de renforcer nos compétences en programmation avec Rust. Nous avons découvert et appliqué plusieurs concepts clés tels que la gestion de la mémoire, l'utilisation des traits, le pattern matching et la manipulation de structures de données. Grâce à cet exercice, nous disposons désormais d'une base solide pour des applications futures.