Fiche technique de mon projet Rust

Ce que fait le programme

Le programme fonctionne avec un menu qui tourne tant que l'utilisateur ne quitte pas. Il propose plusieurs options numérotées (de 1 à 6). L'utilisateur tape un chiffre pour faire une action.

On peut :

- Ajouter un livre (en entrant le titre, l'auteur, et l'année)
- Emprunter un livre (ça le passe en indisponible)
- Retourner un livre (il redevient disponible)
- Voir tous les livres (titre, auteur, année et dispo ou pas)
- Voir seulement les livres qui sont disponibles
- Quitter

Les livres sont stockés dans un tableau dynamique (en fait un `Vec`), et chaque livre est représenté par une structure avec le titre, l'auteur, l'année et un booléen pour savoir s'il est dispo ou pas.

Les parties un peu techniques

J'ai utilisé :

- La **struct** `Livre` pour regrouper les infos sur chaque livre
- Le **Vec<Livre>** pour stocker tous les livres ajoutés
- `std::io` pour lire les entrées de l'utilisateur (avec `read_line`)
- `.trim()` pour enlever les sauts de ligne, et `.parse()` pour convertir l'année en `u32`
- Des boucles `for` pour chercher un livre et modifier son état (dispo ou non)
- Des conditions (`if/else`) un peu partout pour gérer les cas d'erreurs

---

Ce que j'ai appris / les difficultés

Franchement, ce projet m'a appris pas mal de choses :

- J'ai compris comment on déclare une struct et comment on la remplit
- J'ai appris à manipuler les vecteurs, à ajouter dedans, et à les parcourir
- J'ai vu comment Rust est super strict avec les références (parfois ça bloque quand on veut modifier un élément dans une boucle… il faut penser à `&mut`)
- Et aussi que Rust ne te pardonne rien niveau types ! Si tu veux convertir une chaîne en nombre et que ça ne marche pas, il faut tout bien gérer avec `match`

Améliorations possibles (si j’avais plus de temps)

- Ajouter une vraie sauvegarde dans un fichier `.txt` ou `.json` pour ne pas perdre les livres quand on quitte
- Gérer les doublons plus finement (peut-être en mettant un ID unique à chaque livre)
- Peut-être même relier ça à une base de données ou faire une interface graphique (mais là ce serait un autre niveau)

Conclusion

Ce n’est pas un projet très compliqué, mais pour moi c’est une bonne base. Ça m’a permis de mieux comprendre Rust, surtout les types, les structures, les vecteurs, et les entrées utilisateur. J’ai codé ça de manière simple, étape par étape, sans chercher à faire compliqué.