Rapport détaillé sur le projet "Budget Manager"

Realise par :

Abdelwadoud Ait ouali

Mehdi Ramach

Introduction

Ce rapport présente une analyse complète de l'application CLI "Budget Manager". Cette application a pour objectif de permettre aux utilisateurs de gérer leurs budgets et leurs transactions de manière simple et efficace grâce à une interface en ligne de commande.

Structure du code

Modules utilisés

- **Clap** : Permet de définir et de gérer les commandes de ligne de commande.
- **Dialoguer** : Fournit des fonctionnalités interactives pour l'utilisateur.
- **Rusqlite** : Utilisé pour la gestion de la base de données SQLite.

Structure des données

- **Budgets** : Chaque budget a un identifiant unique, un nom et un montant total.
- **Transactions** : Chaque transaction est associée à un budget via un identifiant et représente un montant.

Fonctionnalités principales

Commandes disponibles

1. **AddBudget** : Ajouter un nouveau budget.
2. **RemoveBudget** : Supprimer un budget existant.
3. **EditBudget** : Modifier un budget.

4. **AddTransaction** : Ajouter une transaction à un budget.
5. **RemoveTransaction** : Supprimer une transaction.
6. **EditTransaction** : Modifier une transaction existante.
7. **ShowBudgets** : Afficher les budgets et leur solde restant.

Gestion de la base de données SQLite

Tables définies

1.

budgets

2.

- **id** : Identifiant unique.
- **name** : Nom du budget.
- **amount** : Montant total du budget.

3.

transactions

4.

- **id** : Identifiant unique.
- **budget_id** : Référence à un budget.
- **amount** : Montant de la transaction.

Relations

- La table transactions est liée à la table budgets via budget_id (clé étrangère).

Exemple d'utilisation

Ajouter un budget

Commande : budget_manager AddBudget --name "Voyage" --amount 2000

Afficher les budgets

Commande : budget_manager ShowBudgets Sortie :

Budgets :
ID: 1, Name: Voyage, Amount: 2000, Remaining: 2000

Code source commenté

Fonction `calculate_remaining`

Permet de calculer le solde restant d'un budget :

```
fn calculate_remaining(conn: &Connection, budget_id: i32) -> Result<f64> {
   let mut stmt = conn.prepare("SELECT SUM(amount) FROM transactions WHERE budget_id = ?1")?;
   let total: f64 = stmt.query_row(params![budget_id], |row| row.get(0)).unwrap_or(0.0);
   let budget_amount: f64 = conn.query_row("SELECT amount FROM budgets WHERE id = ?1",
params![budget_id], |row| row.get(0))?;
   Ok(budget_amount - total)
}
```

Explication : Cette fonction interroge la base de données pour récupérer la somme des transactions et le montant total du budget, puis retourne la différence.

Améliorations possibles

1. Ajouter des catégories aux budgets pour une meilleure organisation.
2. Permettre l'import/export des données en formats CSV ou JSON.
3. Ajouter une fonctionnalité de génération de rapports financiers.
4. Améliorer la gestion des erreurs pour une expérience utilisateur optimale.

Conclusion

Ce projet "Budget Manager" est une solution pratique pour gérer les budgets et transactions à travers une interface CLI. Avec quelques améliorations, il pourrait devenir un outil incontournable pour les particuliers ou les petites entreprises.
