# CAHIER DES CHARGES DE PROJET : BUDGET -MANAGER

Bouziane Nouha & EL Idrissi Aicha

23 DECEMBRE 2024

ENSA FES
ISEIA

# I. Introduction :

Dans le cadre de la fin du premier semestre en tant qu'étudiantes en Système embarqués et intelligence artificielle à ENSAF, nous avons l'opportunité de mettre en œuvre les connaissances acquises durant les séances de RUST en un projet. Nous avons choisi de réaliser le projet qui a pour but : gestion de budgets

Nous avons décidé de choisir ce projet afin de développer une application pratique qui nous aide à mieux gérer nos finances tout en approfondissant nos connaissances en programmation Rust. Ce projet nous permettra également de nous familiariser avec les concepts de bases de données et leur utilisation dans un contexte réel, tout en abordant des notions telles que l'organisation des données, l'interaction avec une interface utilisateur, et l'optimisation de la gestion des transactions financières.

Ce rapport apparaît dans le cadre de ce projet afin de rendre compte au terme d'un mois alloué du travail accompli. Ainsi, plusieurs points seront abordés dans ce rapport, comme le cahier des charges, le travail effectué, les difficultés rencontrées et les améliorations possibles.

# II. Contexte :

Dans un monde où la gestion financière est devenue essentielle pour atteindre des objectifs personnels et professionnels, ce projet permet de développer une application permettant de suivre les revenus, les dépenses et le solde restant. Il implique également l'utilisation de bases de données pour stocker et organiser les transactions et les budgets, ce qui constitue un bon moyen d'apprendre à gérer des informations de manière structurée.

La gestion des finances personnelles est une préoccupation majeure dans la vie quotidienne, mais elle peut souvent sembler complexe sans outils adaptés. Ce projet de gestion de budget répond à ce besoin en proposant une solution numérique simple et efficace. En parallèle, il constitue une opportunité d'apprentissage pour renforcer les compétences en programmation avec Rust, tout en découvrant les bases de données et leur rôle clé dans la structuration et le stockage des informations.

# III. Objectifs :

L'objectif final de notre projet est donc de créer un gestionnaire de budget simple mais fonctionnel qui permettra à l'utilisateur de suivre ses finances. Ainsi ce projet vise à fournir une expérience pratique dans l'intégration d'une base de données avec une application en Rust, et à apprendre à manipuler et organiser des données efficacement. Pour cela, nous avons décidé de découper le projet en plusieurs sous-objectifs :

- Ajouter, modifier et supprimer des budgets.
- Ajouter, modifier et supprimer des transactions.
- Afficher le solde restant dans un budget.
- Afficher la liste des budgets.

# IV. Besoins Fonctionnels :

Ce document présente un tableau détaillé des fonctions utilisées dans le projet Rust, incluant leurs descriptions, détails et utilisations dans le projet.

| Fonctionnalité | Description | Détails |
|---|---|---|
| **Gestion des Budgets** | Permet à l'utilisateur de gérer ses budgets. | |
| Ajouter un budget | Ajouter un nouveau budget avec un nom et un montant total. | Le montant restant est initialisé à égalité avec le montant total. Vérification des montants (positifs, raisonnables). |
| Supprimer un budget | Supprimer un budget existant. | Supprimer un budget en spécifiant son nom. Vérification de l'existence du budget. |
| Modifier un budget | Modifier un budget existant (nom, montant total). | Modification du nom et du montant total d'un budget existant. |
| Afficher les budgets | Afficher tous les budgets existants avec les informations (ID, nom, montant total, montant restant). | Liste des budgets affichée sous forme de tableau. |
| **Gestion des Transactions** | Permet à l'utilisateur de gérer les transactions associées aux budgets. | |
| Ajouter une transaction | Ajouter une transaction à un budget. | Spécifier le nom de la transaction et le montant. Vérification de l'existence du budget avant ajout. |
| Supprimer une transaction | Supprimer une transaction d'un budget. | Suppression d'une transaction en spécifiant le nom de la transaction et le budget associé. |
| Modifier une transaction | Modifier une transaction (nom, montant). | Modification du nom et du montant d'une transaction. Vérification de l'existence de la transaction avant modification. |
| **Calcul du Solde Restant** | Calculer le montant restant d'un budget après dépenses. | Le solde restant est calculé comme étant la différence entre le montant total et les transactions effectuées. Alerte si le solde restant est inférieur à 10 %. |
| **Interactivité** | Rendre l'application interactive via la console. | Utilisation de couleurs pour le texte (rouge pour les erreurs, vert pour les succès) et affichage sous forme de tableau pour une meilleure lisibilité. |
| **Sauvegarde et Persistance** | Stockage des données de budgets et transactions dans une base de données SQLite. | Les budgets et transactions sont sauvegardés dans la base de données SQLite, garantissant la persistance des données après fermeture de l'application. |

# V. Besoins techniques :

Ce document présente un tableau détaillé des dépendances utilisées dans le projet Rust, incluant leurs descriptions, fonctionnalités clés, utilisations dans le projet et détails techniques.

| Dépendance | Version | Description | Utilisation dans le Projet | Fonctionnalités Clés |
|---|---|---|---|---|
| rusqlite | 0.27 | Wrapper ergonomique pour SQLite dans Rust | Gestion des budgets et transactions | CRUD, requêtes préparées, support de SQLite intégré |
| crossterm | 0.26 | Bibliothèque multiplateforme pour manipuler les terminaux | Créer des interfaces textuelles interactives | Gestion des couleurs, curseurs, et événements clavier |
| dialoguer | 0.10 | Créer des menus interactifs en ligne de commande | Menus pour ajouter des budgets et afficher les transactions | Invites de confirmation, sélection et saisie utilisateur |
| console | 0.15 | Bibliothèque pour styliser les textes et animations | Styliser les messages d'erreur et succès | Gestion des styles (gras, italique, couleurs) |
| serde | 1.0 | Sérialisation et désérialisation des données | Conversion des données Rust en JSON ou CSV | Support des formats JSON et CSV avec #[derive] |
| std | Intégré | Bibliothèque standard de Rust | Manipulation des fichiers et gestion des erreurs | I/O, gestion des chaînes, types Result et Option |

# VI. Améliorations :

## 1. Gestion Multi-Utilisateur :

Cette amélioration permet à plusieurs utilisateurs de gérer leurs propres budgets et transactions.

## 2. Importation Automatique des Transactions :

L'importation automatique des transactions peut rendre la gestion des budgets plus efficace, en particulier pour les utilisateurs qui suivent leurs dépenses sur des plateformes ban-

caires en ligne. En effet, elle automatise l'ajout des transactions pour éviter la saisie manuelle. Ainsi, elle supporter différents formats d'importation (par exemple CSV) des transactions.

### 3. Alertes et Notifications :

Améliorer le système d'alertes pour informer les utilisateurs des actions importantes sur leur budget (voir les alertes lorsque le budget est moins de 10% de son montant restant).

### 4. Ajout d'un graphique dans l'application :

L'amélioration par l'ajout d'un graphique permet de visualiser les données financières de manière intuitive, facilitant ainsi la compréhension et l'analyse des budgets et des transactions.

## VII. Résultats attendus :

- **Un programme fonctionnel** de gestion de budgets et de transactions, où l'utilisateur peut facilement ajouter, supprimer, modifier et consulter ses budgets et transactions.
- **Des alertes appropriées** en cas de solde restant faible.
- **Un stockage persistant** des données, permettant à l'utilisateur de revenir aux données même après la fermeture du programme.
- **Une interface utilisateur claire**, avec des messages d'erreur et des informations utiles concernant l'état des budgets et transactions.
- **Gestion multi-utilisateur** avec possibilité d’ajouter et gérer des utilisateurs distincts.
- **Importation automatique des transactions** depuis un fichier CSV pour un traitement rapide.

## VIII. Contrainte et solution :

- Nous n'avons pas pu réaliser la dernière amélioration (Ajout d'un graphique dans l'application) en raison de sa complexité et du manque de temps.
- Il est nécessaire de travailler collectivement sur un seul programme et en plus à distance mais on a trouvé la solution :

➔ La solution est : **GitHub** qui est une plateforme de gestion de version en ligne qui repose sur Git, un système de contrôle de version décentralisé. Elle nous a permet de collaborer sur un même projet en suivant les modifications du code, en partageant les contributions, et en facilitant la gestion des versions du projet de manière efficace et organisée. Grâce à ses fonctionnalités de collaboration (comme les branches, les pull requests, et les revues de code), GitHub simplifie notre travail collectif, même à distance.

# IX. Conclusion :

Pour conclure, nous sommes satisfaites du rendu final de notre projet, car certes il n'est pas parfait, mais reste néanmoins totalement opérationnel. Avec quelques améliorations supplémentaires comme une Gestion Multi-Utilisateur, Importation Automatique des Transactions et Alertes et Notifications

D'un point de vue technique, ce projet assez complet, nous a beaucoup apporté, notamment grâce à l'approfondissement, l'apprentissage et la maîtrise du langage de

D'un point de vue gestion de projet, celui-ci nous a permis de travailler en binôme, en apprenant à devoir s'adapter à la façon de travailler de chacun, mais aussi de devoir respecter un certain délai, nous obligeant à travailler en parallèle sur différents domaines du projet. Cela reste une expérience humaine très enrichissante qui nous aidera certainement lors de notre stage ou encore en milieu professionnel une fois diplômées.