Rapport Projet LOA:
Application Qt pour Gestion de budget

AZIZE Salma et RAJAT Chorouk

Semestre 3, 2025–2026

Table des matières

Chapitre 1

Présentation générale du projet

Ce projet consiste à développer une application de gestion de budget personnel en langage **C++** à l'aide du framework **Qt** et d'une base de données **SQLite**.

L'objectif principal de l'application est de permettre à un utilisateur de :

— gérer ses comptes bancaires (comptes courants et comptes d'épargne),

— enregistrer ses revenus et ses dépenses,

— organiser ses dépenses par catégories et sous-catégories,

— définir et suivre des budgets mensuels,

— analyser sa situation financière à l'aide de graphiques et statistiques.

L'application vise à fournir une interface intuitive, interactive et cohérente, tout en respectant les principes de la programmation orientée objet et de la conception logicielle étudiés dans le module LOA.

Chapitre 2

Architecture logicielle

L'application repose sur une architecture **MVC (Modèle – Vue – Contrôleur)**, permettant une séparation claire des responsabilités.

2.1 Modèle

Le modèle regroupe :
— les classes métier : `Utilisateur`, `Compte`, `Operation`, `Categorie`, `Budget`, `Transfert`,
— les classes `Repository` chargées de l'accès à la base de données SQLite.
Le modèle encapsule la logique métier, notamment :
— le calcul dynamique des soldes,
— le calcul des dépenses par catégorie,
— le suivi des budgets mensuels.

2.2 Vue

La vue est constituée de l'interface graphique développée avec **Qt Designer**. Elle comprend plusieurs onglets correspondant aux fonctionnalités principales : comptes, catégories, opérations, historique et tableau de bord.

2.3 Contrôleur

Le contrôleur est assuré principalement par la classe `MainWindow`. Il gère :
— les interactions utilisateur (clics, sélections, modifications),
— la communication entre la vue et le modèle,
— la mise à jour dynamique de l'interface.

2.4 Description des classes principales

2.4.1 Classe Utilisateur

La classe `Utilisateur` représente le propriétaire de l'application.
Responsabilités :
— Posséder un ou plusieurs comptes bancaires
— Gérer l'ajout et la suppression de comptes

Cette classe est au sommet de la hiérarchie fonctionnelle de l'application.

2.4.2 Classe Compte

La classe `Compte` est une classe abstraite représentant un compte bancaire générique.
Attributs principaux :
— identifiant du compte
— nom du compte
— solde
— liste des opérations associées
— transferts entrants et sortants
Elle définit les comportements communs à tous les comptes.

2.4.3 Classes CompteCourant et CompteEpargne

Les classes `CompteCourant` et `CompteEpargne` héritent de la classe `Compte`.
— **CompteCourant** : utilisé pour les dépenses quotidiennes et les revenus.
— **CompteEpargne** : destiné à l'épargne.
Le solde est recalculé dynamiquement à partir des opérations et transferts.

2.4.4 Classe Operation

La classe `Operation` représente une transaction financière.
Types d'opérations :
— Revenu
— Dépense
Chaque opération est associée à un compte et à une catégorie.

2.4.5 Classe Transfert

La classe `Transfert` représente un mouvement d'argent entre deux comptes.
Types de transferts :
— Transfert manuel
— Transfert automatique (déclenché en cas de solde négatif)

2.4.6 Classe Categorie

La classe `Categorie` permet de classer les opérations.
Les catégories peuvent être hiérarchiques, ce qui permet l'héritage des budgets.

2.4.7 Classe Budget

La classe `Budget` définit un plafond de dépenses mensuel pour une catégorie donnée.
Elle est utilisée pour calculer les montants dépensés et restants.

2.5 Justification des choix de conception

— L'utilisation de l'héritage permet de différencier les types de comptes.
— Les relations de composition assurent une bonne cohérence des données.
— L'architecture MVC garantit une séparation claire des responsabilités.

Chapitre 3

Modèle de données

3.1 Diagramme de classes

La Figure 3.1 présente le diagramme de classes UML de l'application. Ce diagramme met en évidence :

— les relations d'héritage entre les types de comptes,
— les relations de composition entre comptes, opérations et transferts,
— la hiérarchie des catégories.

3.2 Classe Utilisateur

La classe `Utilisateur` représente le propriétaire de l'application. Elle regroupe l'ensemble des comptes bancaires associés à un utilisateur donné.

Ses responsabilités principales sont :

— l'ajout et la suppression de comptes,
— l'accès centralisé aux données financières.

3.3 Classe Compte

La classe `Compte` est une classe abstraite représentant un compte bancaire. Elle contient :

— un identifiant,
— un nom,
— un solde,
— des opérations et des transferts associés.

Le solde n'est pas stocké de manière statique, mais recalculé dynamiquement à partir des opérations et des transferts enregistrés.

3.4 Classes CompteCourant et CompteEpargne

Les classes `CompteCourant` et `CompteEpargne` héritent de la classe `Compte` :

— le compte courant est utilisé pour les revenus et les dépenses,
— le compte épargne est destiné à la mise de côté.

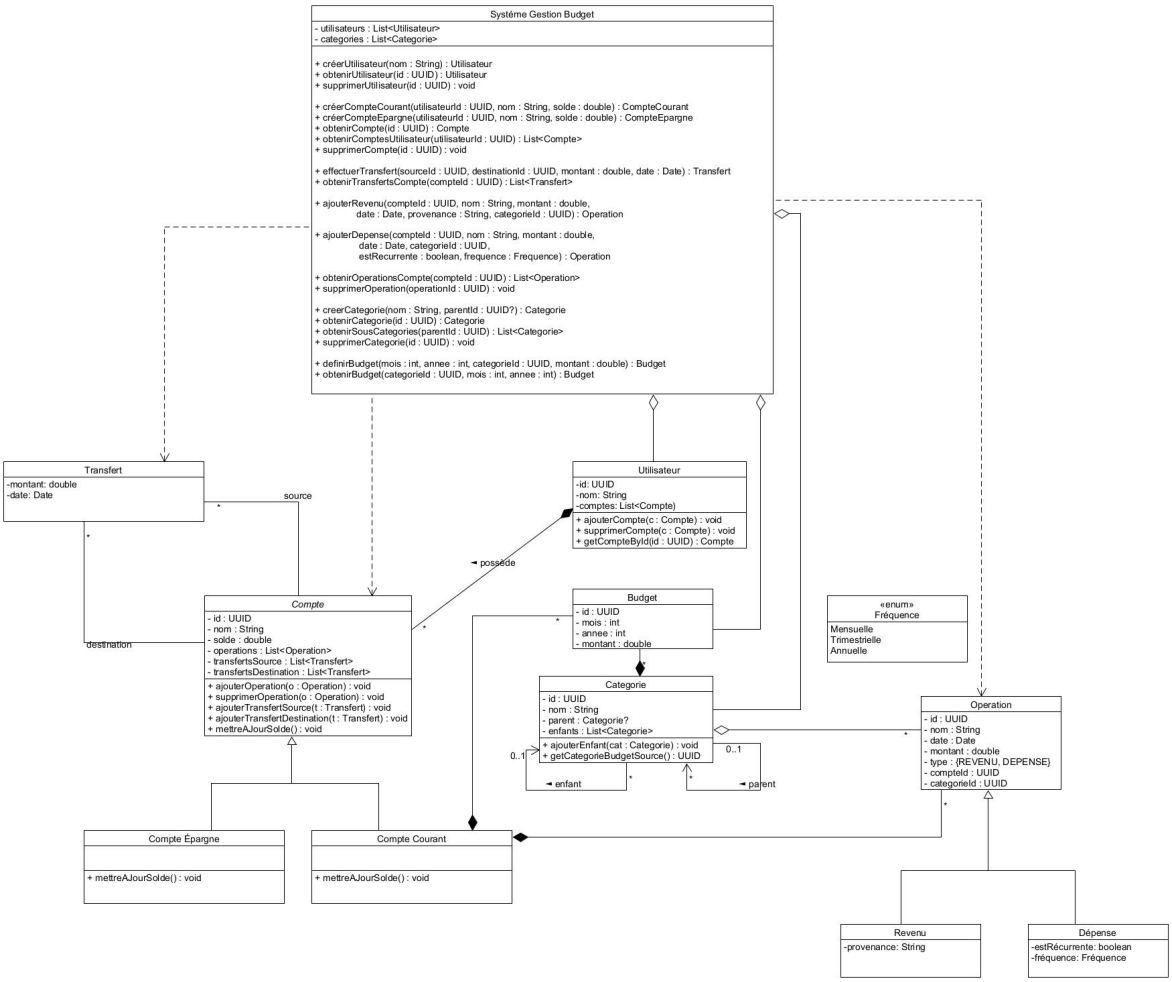

FIGURE 3.1 – Diagramme de classes du système

3.5 Classe Operation

La classe `Operation` représente une transaction financière. Une opération peut être de type :
— revenu,
— dépense.
Chaque opération est associée à :
— un compte,
— une catégorie,
— une date et un montant.

3.6 Classe Transfert

La classe `Transfert` représente un mouvement d'argent entre deux comptes. Elle permet :
— les transferts manuels,
— les transferts automatiques en cas de solde négatif.

3.7 Classe Categorie

La classe `Categorie` permet de structurer les opérations financières. Les catégories peuvent être hiérarchiques (catégories et sous-catégories).

3.8 Classe Budget

La classe `Budget` définit un plafond de dépenses mensuel pour une catégorie. Les dépenses des sous-catégories sont automatiquement prises en compte dans le calcul du budget de la catégorie parente.

Chapitre 4

Interface utilisateur

L'interface graphique est organisée en plusieurs onglets accessibles via un `QTabWidget`.

Chaque onglet correspond à une fonctionnalité principale :

— Comptes,
— Catégories,
— Opérations,
— Historique,
— Dashboard.

Les interactions se font à l'aide de boutons, listes, tableaux, menus déroulants et champs de saisie.

Chapitre 5

Fonctionnalités

5.1 Gestion des comptes

— création de comptes courants et d'épargne,
— suppression d'un compte avec suppression automatique des opérations associées,
— transferts entre comptes.

5.2 Gestion des catégories

— ajout et suppression de catégories,
— définition de budgets mensuels,
— affichage du budget, du montant dépensé et du restant.

5.3 Gestion des opérations

— ajout de revenus et dépenses,
— modification directe dans le tableau,
— suppression d'une opération,
— filtrage par compte et catégorie.

5.4 Dashboard

— graphiques de dépenses par catégorie,
— comparaison revenus / dépenses,
— statistiques mensuelles,
— recommandations automatiques.

Chapitre 6

Scénarios d'utilisation

6.1 Suivi quotidien

L'utilisateur enregistre ses dépenses quotidiennes et visualise l'impact sur son budget en temps réel.

6.2 Analyse mensuelle

En fin de mois, l'utilisateur consulte le tableau de bord afin d'analyser ses dépenses et ajuster ses budgets.

Chapitre 7

Conclusion

Ce projet a permis de développer une application complète de gestion de budget, mettant en œuvre les concepts de programmation orientée objet, d'architecture MVC et de conception d'interfaces graphiques.

L'application répond aux exigences fonctionnelles demandées et constitue une base solide pouvant être enrichie par de futures fonctionnalités.