Université Hassan 1er
Ecole Nationale des Sciences Appliquées
Khouribga

Tronc Commun

1ere Année du Cycle ingénieur

PROGRAMMATION C++

Compte rendu du projet C ++ : Gestion d'une banque

- Réalisé par :

LAGHLID Ayoub
N° 54

LEMLIH Aymane
N° 59

MANBER Chaïmae
N° 61

ZKIM Youssef
N° 91

- Encadré par :

Pr. OMAR ELBANNAY

Année universitaire : 2019/2020

Dédicace

C'est avec profonde gratitude et sincères mots que nous dédions ce modeste travail à nos chers parents ; qui ont sacrifiés leurs vies pour notre réussite et nous ont éclairés par leurs conseils judicieux. Nous espérons qu'un jour, nous pourrions leurs rendre un peu de ce qu'ils ont fait pour nous que Dieu leurs prêtes bonheur et longue vie.

Nous dédions aussi ce travail à nos frères, nos familles, nos amis, nos condisciples, tous nos professeurs qui nous ont enseigné et à tous ceux qui nous sont chers.

Remerciement

Nous tenons à remercier non seulement comme devoir mais par grand respect et gratitude profonde notre professeur Pr. OMAR ELBANNAY à qui nous adressons nos sentiments de reconnaissance et de respect pour nous avoir guidé dans l'élaboration de ce travail, avec la patience et le dynamisme qui le caractérisent, son soutien, ainsi ses directives précieuses durant le déroulement de ce projet.

Sommaire

Introduction Générale

Dans le cadre de l'élément « Programmation en C++ », nous sommes censés élaborer un projet qui compte sur la gestion d'une organisation.

Pour cela on a choisi d'adopter la gestion des comptes bancaires, notre programme sera donc comme ceci :

- Une base de données afin de stocker tout ce qui concerne les comptes et les informations sur les clients, ainsi que d'autres information qui seront expliquées plus tard dans ce rapport.
- Un menu qui suggère les services que notre programme peut offrir.
- Une interface graphique qui facilite l'usage du programme.

Pour avoir une interface graphique pour l'utilisateur, on va utiliser un outil qui s'appelle « Qt » qui au lieu du console « CMD » adopté par « Code Blocks ».

Qt est une interface de programmation applicative orientée objet et développé en C++, conjointement par The Qt Company et Qt Project. Qt offre des composants d'interface graphique (Widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution… Il permet la portabilité des applications qui n'utilisent que ses composants par simple recompilation du code source. Les environnements supportés sont les Unix (dont GNU/Linux) qui utilisent le système graphique X Window System ou Wayland, Windows, Max OS X, Tizen, et également Genode.

Qt supporte des bindings avec plus d'une dizaine de langages autres que le C++, comme Ada, C#, Java, Python, Ruby, Visual Basic, etc.

Dans ce qui suit, on va passer par les étapes de création du programme en détails, ainsi que l'explication du fonctionnement de chaque étape.

Gestion de base de données

Conception :

Comme tous les autres systèmes de gestion nous avons besoin d'une base de données pour stocker et réutiliser les données des clients, comptes ainsi que les transactions effectuées.

Pour cela on a choisi d'aller avec ***SQLITE MANAGER*** vu aux facilités qu'il présente dans la gestion des bases de données soit au niveau d'implémentation ou manipulation.

Notre base de données à trois tables :

- Clients.
- Comptes.
- Transactions.

Schéma :

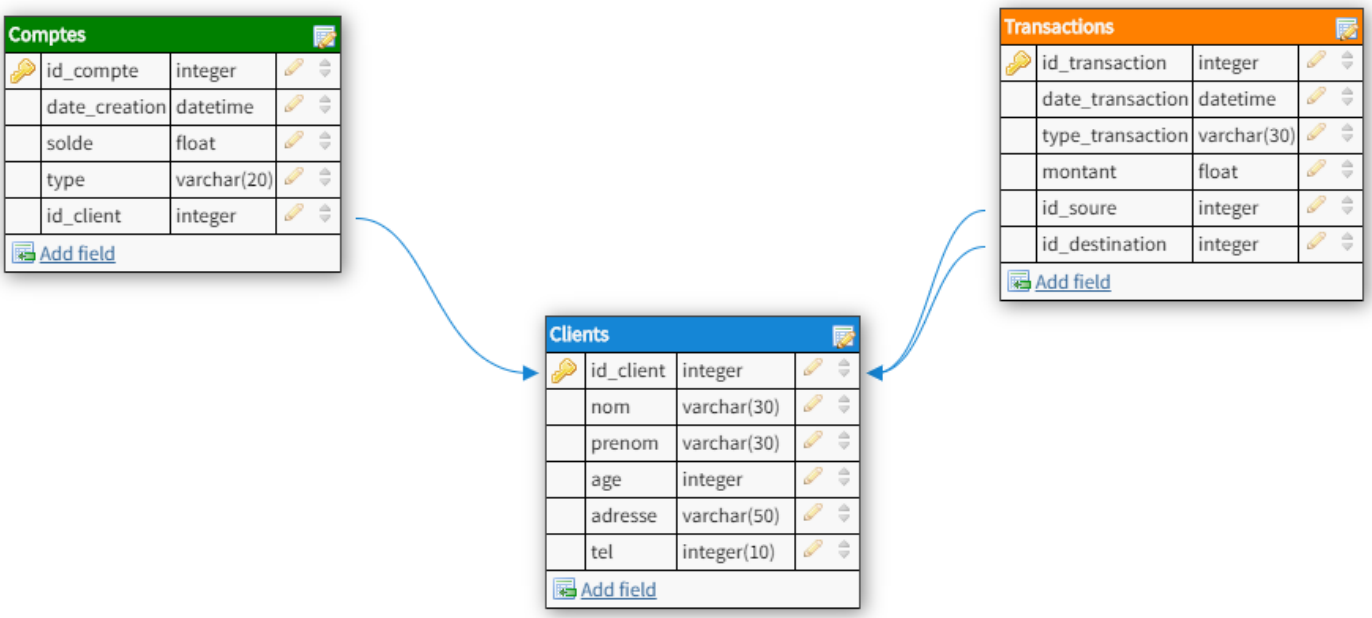


Table clients :

C'est celui qui regroupe tout ce qui concerne les données personnelles des clients ; le nom, prénom, âge, adresse et numéro du téléphone.

Il se peut que deux clients aient le même nom, prénom parfois âge aussi, pour éviter cette redondance on a utilisé un autre attribut comme clé primaire : *id_client* qui s'initialise et s'incrémente automatiquement.

Table comptes :

Contient les données indispensables pour les comptes des clients et qui facilite l'accès, la recherche et les transactions.

Les attributs de cette table sont comme suit : date de création, solde, type et id_client qui est utilisé comme une clé étrangère pour définir le propriétaire du compte.

Id_compte est encore présente pour distinguer chaque et éviter tout type de redondance.

Table transactions :

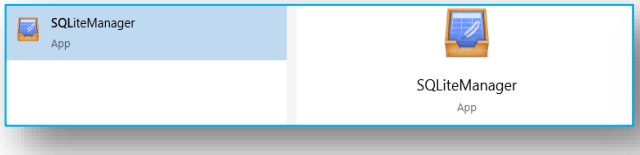
Les transactions ont trois formes ; Déposition, tirage et transfert.

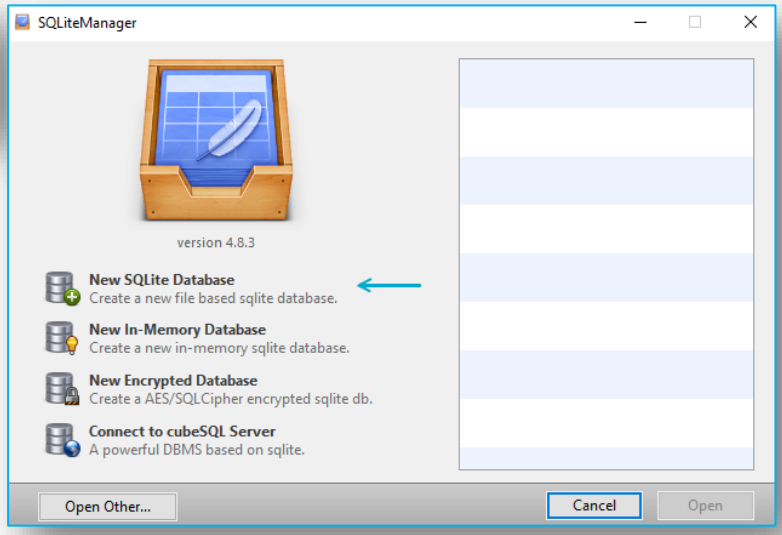
On a besoin donc des attributs qui couvres ces opérations ainsi pour assurer la traçabilité de ces transactions pour des raisons de sécurité de chaque client.

La table regroupe les attributs suivants : date de transaction, type de transaction, montant ainsi que le compte source et compte de destination en cas de transfert. Qui sont des clés étrangères. La clé primaire est : *id_transaction.*

→ Création de la base de données :

Comme déjà cité, on a travaillé avec SQLITEMANAGER :





On choisit nouvelle base de données SQLite.

```sql
CREATE TABLE clients (
    id_client INTEGER PRIMARY KEY AUTOINCREMENT,
    nom VARCHAR(30) NOT NULL,
    prenom VARCHAR(30) NOT NULL,
    age INT NOT NULL,
    adresse VARCHAR(50) NOT NULL,
    tel NUMBER(10) NOT NULL,

    CONSTRAINT const_age CHECK( age > 0 )
);
CREATE TABLE comptes (
    id_compte INTEGER PRIMARY KEY AUTOINCREMENT,
    date_creation DATE NOT NULL,
    solde REAL NOT NULL,
    type VARCHAR(20) NOT NULL,
    id_client INT,

    CONSTRAINT fk_id_client FOREIGN KEY(id_client) REFERENCES clients(id_client),
    CONSTRAINT fk_solde CHECK( SOLDE > 0)
);
CREATE TABLE transactions (
    id_transaction INTEGER PRIMARY KEY AUTOINCREMENT,
    date_transaction DATE NOT NULL,
    type_transaction VARCHAR(30) NOT NULL,
    montant REAL NOT NULL,

    /* numero de compte object de la transaction */
    id_source INTEGER NOT NULL,
    id_destination INTEGER,

    CONSTRAINT fk_id_source FOREIGN KEY(id_source) REFERENCES comptes(id_compte) ON DELETE CASCADE,
    CONSTRAINT fk_id_destination FOREIGN KEY(id_destination) REFERENCES comptes(id_compte) ON DELETE CASCADE
);
```

→ Remplissage de la base de données :

```
INSERT INTO "clients" ( "id_client","nom","prenom","age","adresse","tel" ) VALUES ( '1','Manber','Chaimae','22','Lot ben abdoun Khouribga','607433748');
INSERT INTO "clients" ( "id_client","nom","prenom","age","adresse","tel" ) VALUES ( '2','Laghlid','Ayoub','22','Lot El Firdaous Khouribga','618206924');
INSERT INTO "clients" ( "id_client","nom","prenom","age","adresse","tel" ) VALUES ( '3','Zkim','Youssef','22','Lot Khou abdoun Khouribga','649017161');
INSERT INTO "clients" ( "id_client","nom","prenom","age","adresse","tel" ) VALUES ( '4','Trachi','Hasnae','22','Lot X Khouribga','634080607');
INSERT INTO "clients" ( "id_client","nom","prenom","age","adresse","tel" ) VALUES ( '5','Lemlih','Aymane','70','El Qarya Sale','664551986');
INSERT INTO "clients" ( "id_client","nom","prenom","age","adresse","tel" ) VALUES ( '7','Khartoum','Mohamed','50','Bloc Zoz V Settat','679005368');
INSERT INTO "clients" ( "id_client","nom","prenom","age","adresse","tel" ) VALUES ( '8','Gym','Ali','18','Sebata 04 Casa','601998415');
INSERT INTO "clients" ( "id_client","nom","prenom","age","adresse","tel" ) VALUES ( '9','Rifqaoui','Chaimaa','24','Bloc Zoz Settat','607668716');
INSERT INTO "clients" ( "id_client","nom","prenom","age","adresse","tel" ) VALUES ( '10', 'Lhamidi','Frix','25','Lot 36 Berrechid','629966165');
INSERT INTO "clients" ( "id_client","nom","prenom","age","adresse","tel" ) VALUES ( '11','Faoukhar','Qamama','22','Souk sebt ouled nema','670222405');

INSERT INTO "comptes" ( "id_compte","date_creation","solde","type","id_client" ) VALUES ( '9','2020-03-29 14:18:50','87.28295','Etudiant (e)','8' );
INSERT INTO "comptes" ( "id_compte","date_creation","solde","type","id_client" ) VALUES ( '1','2020-04-18 23:55:16','990000.1629','Etudiant (e)','1' );
INSERT INTO "comptes" ( "id_compte","date_creation","solde","type","id_client" ) VALUES ( '8','2020-04-19 11:10:14','25.786','Etudiant (e)','5' );
INSERT INTO "comptes" ( "id_compte","date_creation","solde","type","id_client" ) VALUES ( '5','2020-04-20 17:48:53','190.95','Sans-Travail','10' );
INSERT INTO "comptes" ( "id_compte","date_creation","solde","type","id_client" ) VALUES ( '6','2020-04-26 00:03:47','28.122','Sans-Travail','9' );
INSERT INTO "comptes" ( "id_compte","date_creation","solde","type","id_client" ) VALUES ( '7','2020-05-20 02:17:38','85.624','Sans-Travail','11' );
INSERT INTO "comptes" ( "id_compte","date_creation","solde","type","id_client" ) VALUES ( '4','2020-05-20 03:36:51','130000.1566','Sans-Travail','4' );
INSERT INTO "comptes" ( "id_compte","date_creation","solde","type","id_client" ) VALUES ( '2','2020-05-20 03:39:05','500000','Fonctionnaire','2' );
INSERT INTO "comptes" ( "id_compte","date_creation","solde","type","id_client" ) VALUES ( '3','2020-05-20 12:29:28','458000.72','Fonctionnaire','3' );
INSERT INTO "comptes" ( "id_compte","date_creation","solde","type","id_client" ) VALUES ( '11','2020-05-21 10:03:01','58.495','Fonctionnaire','7');
```

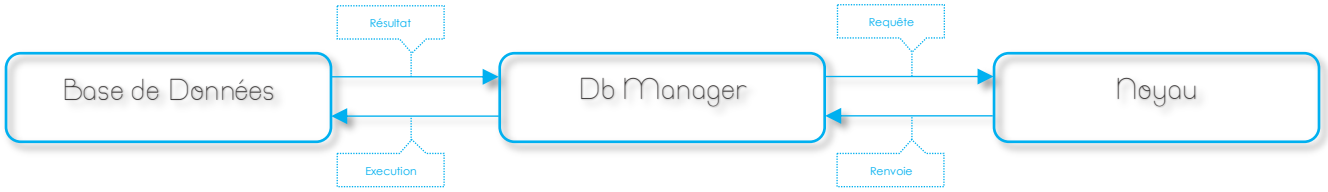
C'est tout simplement la saisie des données dans chaque table afin de les exploiter après.

Manipulation :

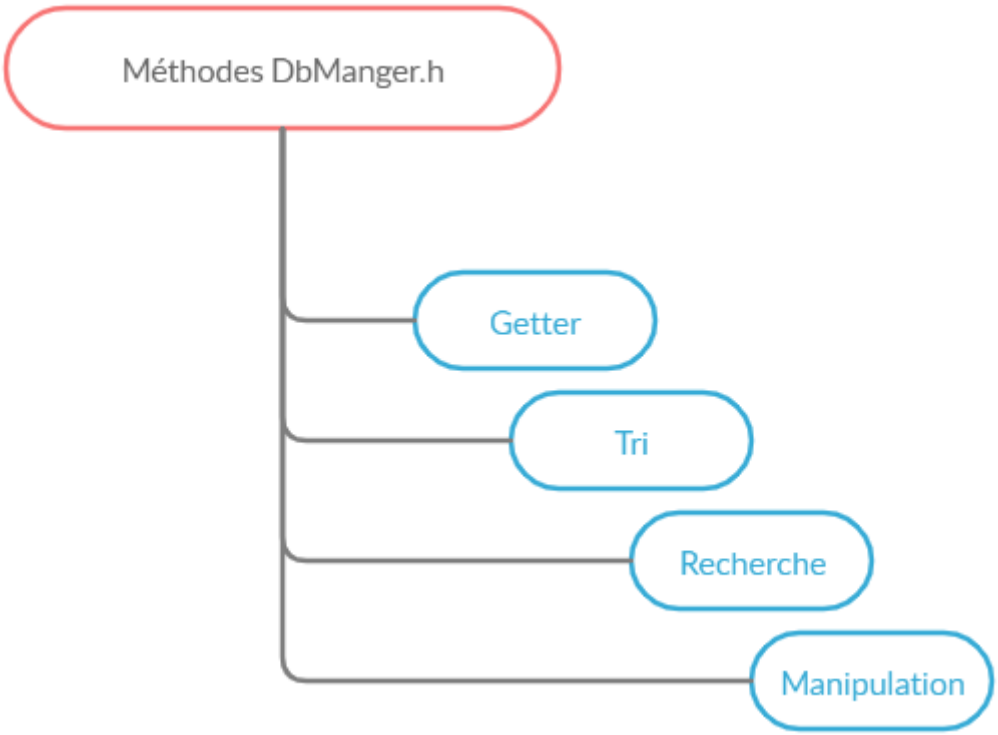
Afin de manipuler cette base de données nous devons créer une connexion entre la base de données et le noyau du programme.

Nous nous sommes trouvés devant l'obligation de créer une classe *DbManager.h* , qui s'exécute par le noyau au niveau de la base de données et renvoie le résultat.

Le constructeur de cette classe s'occupe seulement de se connecter avec la base de données.



La classe DbManager.h contient 4 types de méthodes regroupé de cette façon :



Getter :

Ces fonctions renvoient un champ spécifié.

Exemple : Renvoyer le montant total des comptes bancaire.

Tri :

Ce type des méthodes renvoie les résultats triés selon le champ spécifié.

Exemple : Trié les comptes par date de création.

Recherche :

Ce type effectue la recherche dans la base de données et renvoie le résultat en type *QSQlQuery* (les Champs de ce résultat peuvent être retirer avec la fonction *.value(int)*).

Exemple : Renvoyer les clients ayant plus d'un compte.

Manipulation :

Ce type de fonctions renvoie des booléens. C-à-d ;

TRUE : si l'opération a été bien effectué

FALSE : sinon

Outils QT utilisés :

- Signal et slot

Avant de se lancer dans l'explication des classes, nous avons vu qu'il serait préférable de détailler un peu le fonctionnement général de la plupart des classes du projet, et plus particulièrement celui des fonctions Slot que nous ayons utilisées. En effet, chaque fonction doit avoir une connexion avec le Widget qui lui correspond. Chose qui va rendre la fenêtre dynamique. L'exemple le plus significatif nous est fourni par l'appui sur un bouton. C'est à dire, ce bouton va émettre un « signal » et par la suite la fonction membre de la classe, que l'on va nommer dans ce cas « Slot », va être appelée quand ce signal va lui être « connecté ». En général, quand un signal est émis, les slots qui y sont connectés sont habituellement exécutés immédiatement, tout comme un appel normal à une fonction. Donc si on veut faire plus simple on pourrait dire qu'un slot est une fonction qui va être appelée en réponse à un signal particulier.

Le principe de cet exemple sera appliqué sur tous les boutons et Widgets du programme tout entier.

Maintenant, si on veut comprendre un peu comment cela fonctionne on peut prendre les deux boutons « A propos » et « Quitter » figurant sur la fenêtre menu (MainWindow) comme un bel exemple. Le détail du fonctionnement nous apparait sur la deuxième figure (MainWindow.cpp), cela montre comment nous avons pu connecter les « Signals » et « Slots » de ces deux boutons.

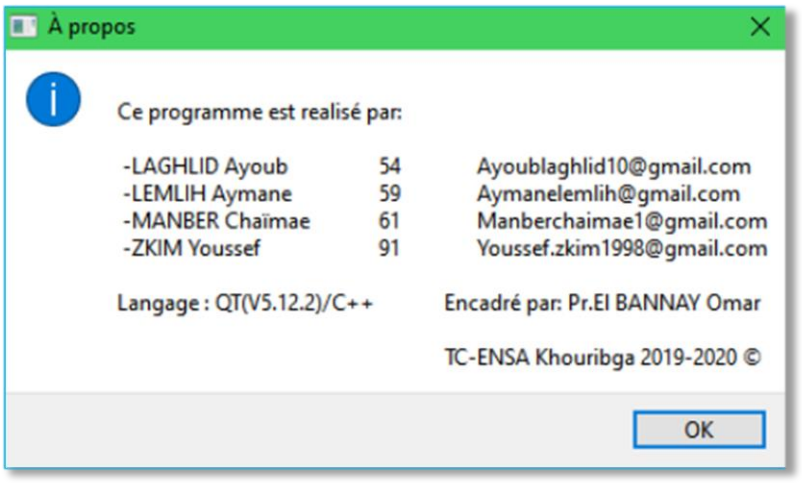

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    // Initialize database
    m_db = new DbManager();

    // connect slot of about and exit button
    connect(ui->exitButton, SIGNAL(clicked()), qApp, SLOT(quit()));
    connect(ui->aboutButton, SIGNAL(clicked()), this, SLOT(aboutDialogSlot()));
```

```
void MainWindow::aboutDialogSlot(){
    this->hide();
    QMessageBox::information(this, "À propos", "\nCe programme est realisé par:\n\n"
                                               " -LAGHLID Ayoub          \t54\t  Ayoublaghlid10@gmail.com\n"
                                               " -LEMLIH Aymane         \t59\t  Aymanelemlih@gmail.com\n"
                                               " -MANBER Chaïmae      \t61\t  Manberchaimae1@gmail.com\n"
                                               " -ZKIM Youssef              \t91\t  Youssef.zkim1998@gmail.com\n\n"
                                               "Langage : QT(V5.12.2)/C++\t           Encadré par: Pr.El BANNAY Omar\n\n"
                                               "                                  \t           TC-ENSA Khouribga 2019-2020 ©");
    this->show();
}
```

- QmessageBox :



Loin des signaux et slots, nous avons découvert un nouveau concept moins exigeant, que nous avons utilisé pour afficher des messages sous forme de boîtes de dialogue. En effet, nous allons nous servir du même exemple (plus précisément « A propos ») de la partie Slot et Signals, vu que nous l'avons trouvé parfait pour bien expliquer le principe. Les boîtes de dialogues « afficher un message » sont contrôlées par la classe « QMessageBox », qui vont généralement avec des méthodes statiques, ces dernières se comportent comme de simples fonctions. Dans cet exemple, nous avons utilisé la méthode statique information (), qui va permettre d'ouvrir une boite de dialogue constituée d'une icône information, d'un message personnalisé et d'un bouton Ok (si par exemple on veut avoir une icône d'avertissement on peut utiliser une méthode statique warning (), et ainsi de suite). Le résultat doit s'afficher comme ci-dessus.

- Librairies QT :

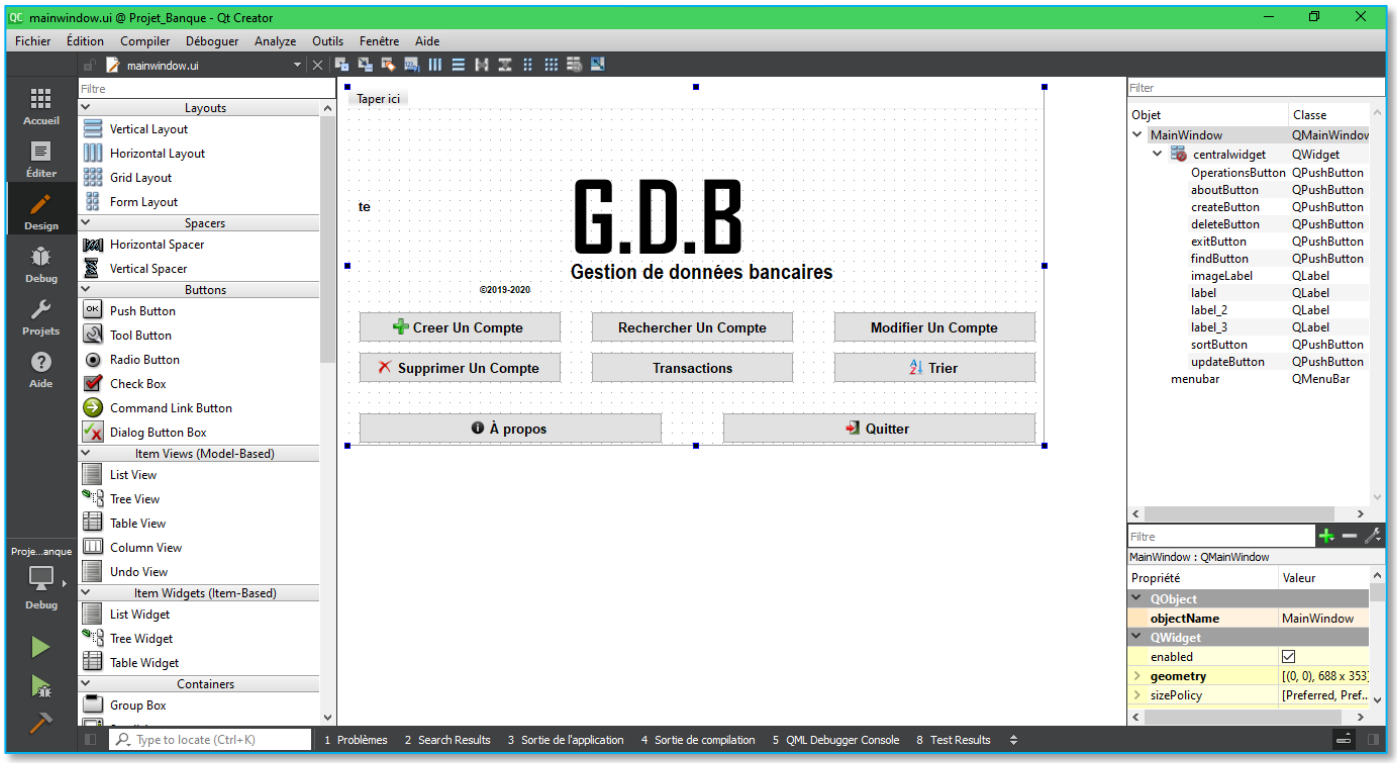
Qt offre un large choix de bibliothèques, nous en avons choisi quelques-unes qui vont nous servir comme par exemple dans tout ce qui est en rapport avec l'environnement SQL, les boites de dialogues, les boutons, les champs d'entrée de texte (LineEditInput), etc…

```
#include <QWidget>
#include <QMessageBox>
#include <QSqlQueryModel>
#include <QItemSelection>
#include <QtSql/QSqlDatabase>
#include <QtSql/QSqlDriver>
#include <QtSql/QSqlError>
#include <QtSql/QSqlQuery>
#include <QDebug>
#include <QDate>
```

Ces librairies sont utilisées dans les headers(.h) selon le besoin.

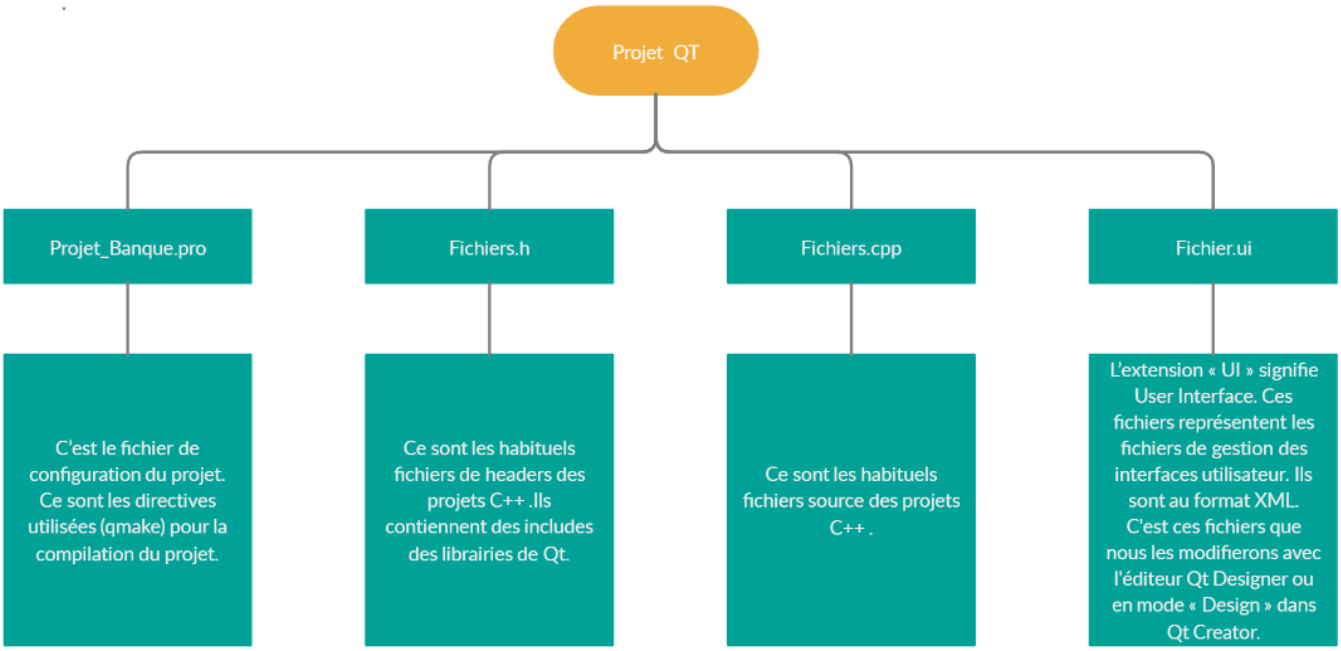
- Design « Dans QTCreator » :

Ce mode est utilisé pour modéliser et concevoir les fenêtres visuellement, il permet aussi d'ajouter, modifier et personnaliser les Widgets. Il suffit de choisir le widget désiré et le mettre dans la fenêtre, puis on utilise le principe déjà mentionné pour établir la connexion entre le signal de ce widget et son Slot.



- Fichiers du projet :

Notre projet englobe 4 types de fichiers :



N.B : *Puisque « Qt » est un langage conçu en anglais, et pour des raisons de cohérence, nous avons vu qu'il serait préférable d'écrire quelques commentaires en anglais, ainsi que de choisir des noms anglais pour les fichiers et fonctions.*

Classes principales

Le constructeur de chaque classe figurant ci-dessous, prend en argument lors de la création de chaque objet : un pointeur sur la base de données et un pointeur sur la fenêtre principale du Menu.

Exemple : Création d'un Compte

Lorsque l'utilisateur clique sur le bouton « Créer Un Compte » la fonction responsable est :

```
void MainWindow::createAccountSlot(){
    // hide the main menu
    this->hide();

    // Show the child menu
    CreateAccountWidget *createwidget = new CreateAccountWidget(m_db, this);
    createwidget->show();
}
```

1- AccountClientsWidget :

Après avoir insérer les données, il se peut qu'on ait entré les mauvaises informations. Pour cela, nous avons conçu une classe responsable de modifier les données des clients. Cette dernière contient des fonctions de recherche (par numéro de compte, nom ou prénom), ainsi que d'autres fonctions de modification et d'affichage. Lorsqu'on appuie sur le bouton « Rechercher » le résultat de la recherche s'affiche sous forme d'un tableau qui peut contenir un ou plusieurs comptes. Par la suite, la main est donnée à l'utilisateur pour choisir le compte sur lequel il va effectuer les modifications. Une fois que c'est fait, le formulaire se remplit automatiquement tout en laissant à l'utilisateur la liberté de modifier dans les champs. Quand le bouton « sauvegarder » est appuyé, les modifications s'enregistrent sur la base de données. Le bouton « annuler » permet de retourner au menu principal.

```
namespace Ui {
class AccountClientsWidget;
}

class AccountClientsWidget : public QDialog
{
    Q_OBJECT

public:
    explicit AccountClientsWidget(DbManager *db, QWidget *parent = nullptr);
    ~AccountClientsWidget();
private slots:

    void searchByIDSlot(bool checked);                  // Recherche par Numéro de compte
    void searchByNomSlot(bool checked);                 // Recherche par Nom
    void searchByPrenomSlot(bool checked);              // Recherche par Prénom

    void searchButtonSlot();                            // Lorsqu'on clique sur le bouton "Rechercher"
    void searchDBSlot();                                // Affichage du résultat sous forme de tableau
    void cancelButtonSlot();                            // Lorsqu'on clique sur le bouton "Annuler"
    void on_tableView_clicked(const QModelIndex &index);// Auto-remplissage du formulaire à partir du compte choisi sur le tableau
    void on_pushButton_clicked();                       // Lorsqu'on clique sur le bouton "Sauvegarder"

signals:
    void searchSignal();                                // Emettre un signal de "Rechercher"

private:
    void closeEvent(QCloseEvent *bar);                  // Fermeture de la fenetre
    Ui::AccountClientsWidget *ui;                       // Pointeur sur Interface Utilisateur liée à la fenetre
    QString nom, prenom, accountid;
    DbManager *m_db;                                    // Pointeur sur la Base de Données
    QSqlQueryModel *m_model;                            // Résultat sous forme de tableau
    QSqlQuery *m_query;                                 // Résultat des requetes SQL
    QString *m_clientID, *m_accountID;                  // Pointeurs sur le Numéro de Compte et Le Numéro du Client à modifier
};
```

2- CreateAccountWidget :

Le but de cette classe est la création d'un nouveau compte. En effet, elle permet de récupérer les données entrées par l'utilisateur en remplissant tous les champs requis, et d'ajouter ces informations à la même base de données comme nouveau compte, après avoir appuyé sur le bouton « Valider ».

```
namespace Ui {
class CreateAccountWidget;
}

class CreateAccountWidget : public QDialog
{
    Q_OBJECT

public:
    explicit CreateAccountWidget(DbManager *db, QWidget *parent = nullptr);
    ~CreateAccountWidget();

private slots:
    void on_ValiderButton_clicked();            // Lorsqu'on clique sur le bouton "Valider"
    void on_AnnulerButton_clicked();            // Lorsqu'on clique sur le bouton "Annuler"

private:
    void closeEvent(QCloseEvent *bar);          // Fermeture de la fenetre
    Ui::CreateAccountWidget *ui;                // Pointeur sur Interface Utilisateur liée à la fenetre
    DbManager *m_db;                            // Gestionnaire de la base de données
};
```

3- DbManager :

Cette classe est le pilier sur lequel se base la totalité du projet. Elle regroupe toutes les fonctions qui vont servir dans les autres classes, par exemple : l'ajout d'un compte, la modification, le tri, les opérations bancaires etc. L'objectif de cette classe est de mettre à jour la base de données en exécutant les requêtes SQL.

```
class DbManager
{
public:
    DbManager();

    bool addClient(QString nom, QString prenom, int age, QString adresse, int tel);                    // Ajout d'un Client dans BD
    bool addAccount(double solde, QString type, int id_client);                                        // Ajout d'un Compte
    bool updateClient(int client_id, QString nom, QString prenom, int age, QString adresse, QString tel);  // Mettre à jour les données du Client
    bool updateAccountType(int account_id, QString account_type);                                      // Mettre à jour le Type du Compte
    bool deposit(int accountid, double amount);                                                        // Créditer un Compte
    int draw(int accountid, double amount);                                                            // Débiter un Compte
    int transfer(int source, int destination, double amount);                                          // Echange entre deux Comptes
    bool deleteAccount(int AccountId);                                                                 // Supprimer un Compte de la BD
    bool doesClientExists(int id);                                                                     // Verification de l'existence du Client
    bool hasMultipleAccounts(int id_account, bool search_by_account=true);                             // Tester si un Client a plusieurs Comptes
    bool isClientRedoundant(QString nom, QString prenom, int age, QString adresse, int tel);          // Tester si un Client se répète
    bool doesAccountExists(int account_id);                                                            // Verification de l'existence du Compte
    // Récupérer le Numéro du Client
    int getClientID(QString nom, QString prenom, int age, QString adresse, int tel);
    int getClientID(int id_account);
    // execution des requetes SQL en retournant le résultat
    QSqlQuery query_nom(QString nom);
    QSqlQuery query_prenom(QString prenom);
    QSqlQuery query_accountid(QString accountid);
    QSqlQuery getAccount(int accountid);
    // Recherche par des requetes SQL en retournant le résultat
    QSqlQuery searchByAccountID(int accountid);
    QSqlQuery searchByNom(QString nom);
    QSqlQuery searchByPrenom(QString prenom);
    // Trier par des requetes SQL en retournant le résultat
    QSqlQuery sortByAccountId();
    QSqlQuery sortByDate();
    QSqlQuery sortByCredit();
    QSqlQuery sortByNom();
    QSqlQuery sortByPrenom();

    QSqlDatabase getDB() const;                                                                        // Récupérer la Base de Données
    QString getAccountName(int accountid) const;                                                       // Récupérer le nom du Client
    QString getAccountAmount(int accountid) const;                                                     // Récupérer le Solde du Client

private:
    QSqlDatabase *m_db;                                                                                // Gestionnaire de la Base de Donnée
};
```

4- DeleteWidget :

Pour la suppression d'un compte de la base de données, il suffit d'entrer le numéro de compte dans la case qui lui a été consacrée. Ceci permet d'afficher immédiatement le résultat, c'est-à-dire les données liées au numéro de compte, sans avoir besoin d'un bouton pour valider la recherche. Après s'être assuré que c'est le compte qu'on veuille supprimer, il reste que de cliquer sur le bouton « supprimer » afin d'achever l'opération.

```
namespace Ui {
class DeleteWidget;
}

class DeleteWidget : public QDialog
{
    Q_OBJECT

public:
    explicit DeleteWidget(DbManager *db, QWidget *parent = nullptr);
    ~DeleteWidget();

private slots:
    void on_cancelButton_clicked();                         // Lorsqu'on clique sur le bouton "Annuler"
    void on_deleteButton_clicked();                         // Lorsqu'on clique sur le bouton "Supprimer"
    void on_lineEdit_textChanged(const QString &arg1);      // Afficher les Informations du Compte immédiatement

private:
    void closeEvent(QCloseEvent *bar);                      // Fermeture de la fenetre
    Ui::DeleteWidget *ui;                                   // Pointeur sur Interface Utilisateur liée à la fenetre
    DbManager *m_db;                                        // Gestionnaire de la base de données
};
```

5- SearchWidget :

La classe de recherche est simple. Elle permet d'afficher le résultat de recherche sous forme d'un tableau contenant tous les comptes correspondants aux données entrées par l'utilisateur. Le mode de recherche peut s'effectuer à l'aide du numéro de compte, nom, ou prénom.

```
namespace Ui {
class SearchWidget;
}

class SearchWidget : public QDialog
{
    Q_OBJECT

public:
    explicit SearchWidget(DbManager *db, QWidget *parent = nullptr);
    ~SearchWidget();

private slots:
    void on_searshButton_clicked();                         // Lorsqu'on clique sur le bouton "Valider"
    void on_pushButton_clicked();                           // Lorsqu'on clique sur le bouton "Annuler"
    void on_comboBox_currentIndexChanged(int index);        // Choix du mode de Recherche

private:
    void closeEvent(QCloseEvent *bar);                      // Fermeture de la fenetre
    Ui::SearchWidget *ui;                                   // Pointeur sur Interface Utilisateur liée à la fenetre
    DbManager *m_db;                                        // Gestionnaire de la base de données
    QSqlQueryModel *m_model;                                // Résultat sous forme de tableau
    TableViewWidget *tableviewwidget;                       // Modifier la taille des colonnes du tableau affiché
};
```

6- SortWidget :

```
namespace Ui {
class SortWidget;
}

class SortWidget : public QDialog
{
    Q_OBJECT

public:
    explicit SortWidget(DbManager *db, QWidget *parent = nullptr);
    ~SortWidget();

private slots:
    void on_applyButton_clicked();                          // Lorsqu'on clique sur le bouton "Valider"
    void on_cancelButton_clicked();                         // Lorsqu'on clique sur le bouton "Annuler"

private:
    void closeEvent(QCloseEvent *bar);                      // Fermeture de la fenetre
    Ui::SortWidget *ui;                                     // Pointeur sur Interface Utilisateur liée à la fenetre
    DbManager *m_db;                                        // Gestionnaire de la base de données
    QSqlQueryModel *m_model;                                // Résultat sous forme de tableau
};
```

La classe de tri permet de classer les comptes dans un tableau par ordre croissant.

Le mode de tri peut s'effectuer à l'aide du numéro de compte, nom, prénom, date de création ou solde.

7- TableViewWidget :

Cette classe n'est pas principale néanmoins, elle joue un rôle important quand il s'agit de redimensionner les colonnes des tableaux d'affichage des comptes dans les opérations nécessitant une recherche à part celle de la suppression.

```
namespace Ui {
class TableViewWidget;
}

class TableViewWidget : public QDialog
{
    Q_OBJECT

public:
    explicit TableViewWidget(QSqlQuery query, QWidget *parent = nullptr);
    ~TableViewWidget();
    void resize_column_width(int col, int width);                    // Modifier la taille des colonnes du tableau

private slots:
    void on_pushButton_clicked();                                    // Lorsqu'on clique sur le bouton "Annuler"

private:
    void closeEvent(QCloseEvent *bar);                               // Fermeture de la fenetre
    Ui::TableViewWidget *ui;                                         // Pointeur sur Interface Utilisateur liée à la fenetre
    QSqlQueryModel *m_model;                                         // Résultat sous forme de tableau
};
```

8- TransactionWidget :

Les fonctions de cette classe jouent un rôle primordial. C'est-elle qui consistent le but du projet bancaire. La transaction bancaire met en jeux les deux facteurs suivants : un client émetteur et un client récepteur. Les opérations utilisées sont les suivantes :

```
namespace Ui {
class TransactionWidget;
}

class TransactionWidget : public QDialog
{
    Q_OBJECT

public:
    explicit TransactionWidget(DbManager *db, QWidget *parent = nullptr);
    ~TransactionWidget();

private slots:
    // Lorsqu'on clique sur le bouton "Annuler"
    void on_cancelButton_clicked();
    void on_cancelButton2_clicked();
    void on_cancelButton_2_clicked();

    void on_depositButton_clicked();                                 // Lorsqu'on clique sur le bouton "Deposer"
    void on_depositButton_2_clicked();                               // Lorsqu'on clique sur le bouton "Tirer"
    void on_exchangeButton_clicked();                                // Lorsqu'on clique sur le bouton "Echanger"

private:
    void closeEvent(QCloseEvent *bar);                               // Fermeture de la fenetre
    Ui::TransactionWidget *ui;                                       // Pointeur sur Interface Utilisateur liée à la fenetre
    DbManager *m_db;                                                 // Gestionnaire de la base de données
};
```

Déposer (créditer un compte) :

L'inscription d'une somme d'argent au crédit d'un client.

Tirer (Débiter un compte) : *Ôter de l'argent d'un compte client.*

Transférer (échanger) : *Envoyer et recevoir l'argent entre deux comptes bancaires.*

Ces opérations nécessitent l'exécution des requêtes SQL, utilisées dans les fonctions de la classe DbManager.

9- MainWindow :

La fenêtre principale résulte de cette classe.

Cette dernière comporte toutes les fonctions slot qui permettent d'accéder aux autres fenêtres. Il suffit de choisir l'un des boutons pour effectuer l'une des opérations vues précédemment (Créer Un Compte, Rechercher Un Compte, Supprimer Un Compte, Trier ou Transactions).

```
namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    DbManager *m_db;                            // Gestionnaire de la base de données

private slots:
    void aboutDialogSlot();                     // Slot de A propos
    void deleteAccountSlot();                   // Slot de Supprimer Un Compte
    void findAccountSlot();                     // Slot de Rechercher Un Compte
    void updateAccountSlot();                   // Slot de Modifier Un Compte
    void createAccountSlot();                   // Slot de Creer Un Compte
    void sortAccountSlot();                     // Slot de Trier
    void transactionsSlot();                    // Slot de Transactions

private:
    Ui::MainWindow *ui;                         // Pointeur sur Interface Utilisateur liée à la fenetre du menu
    CreateAccountWidget *createwidget;          // Attribut responsable de "Creer un Compte"
    AccountClientsWidget *updatewidget;         // Attribut responsable de "Modifier un Compte"
    SortWidget *sortwidget;                     // Attribut responsable de "Trier"
    DeleteWidget *deletewidget;                 // Attribut responsable de "Supprimer un Compte"
    TransactionWidget *transactionswidget;      // Attribut responsable de "Transactions"
    SearchWidget *searchwidget;                 // Attribut responsable de "Rechercher un Compte"
};
```

Interfaces et fonctionnement

- Menu :

C'est la première interface que l'utilisateur voie, dès que le programme s'exécute elle s'ouvre.

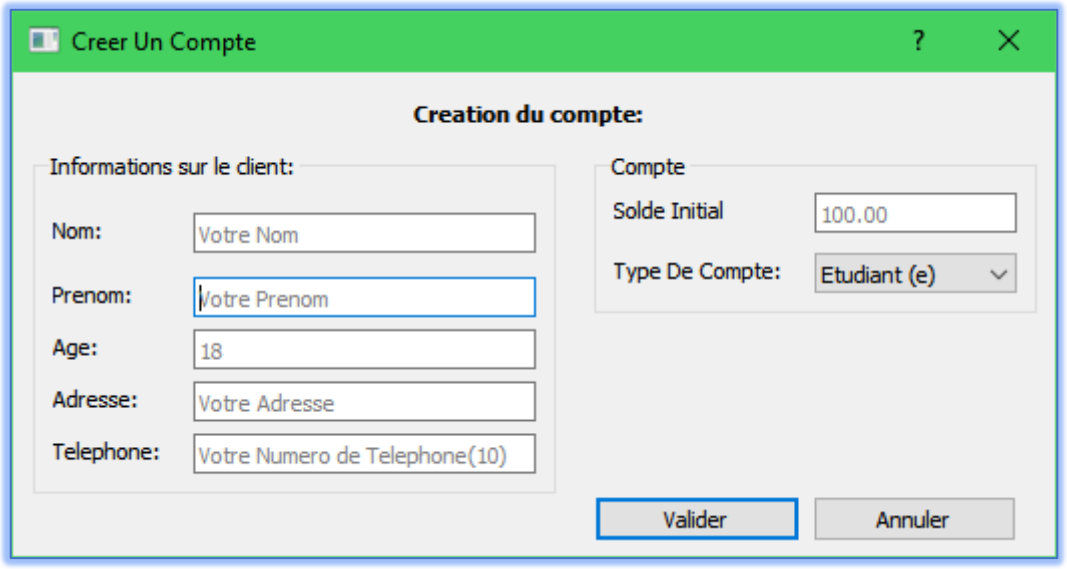
Dans cette fenêtre l'utilisateur choisi un parmi tous les choix possibles, profitant de ce que les fonctions du programme puissent faire.

Le menu regroupe toutes les fonctionnalités dont notre programme est capable de faire, et sont comme-suit :

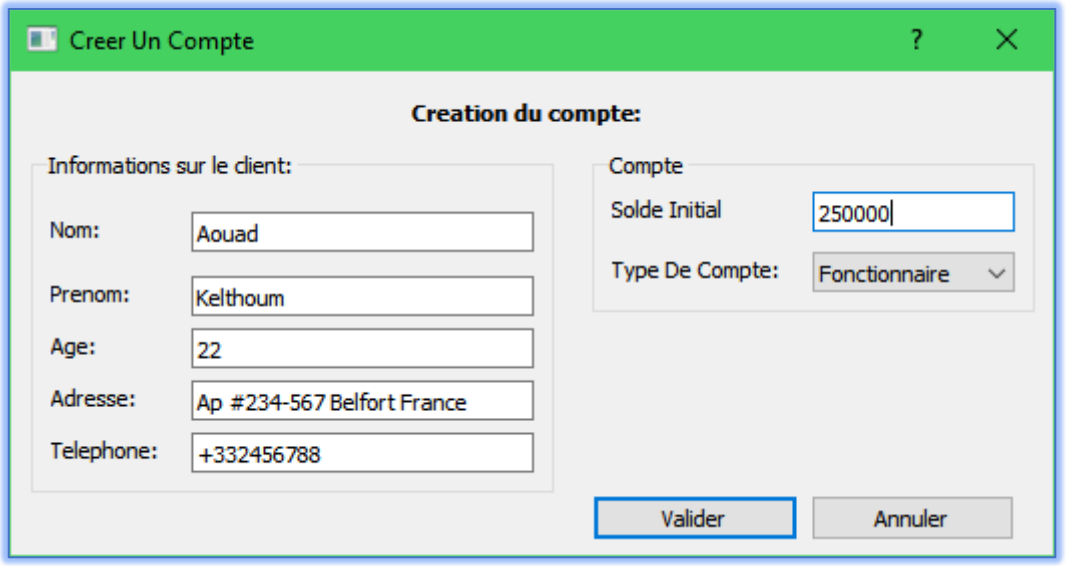
Créer un compte	: Création et enregistrement des comptes dans la base de données.
Rechercher un compte	: Affichage des informations d'un client recherché.
Modifier un compte	: Modification des informations d'un client et les remplacer dans la base de données.
Supprimer un compte	: Suppression des comptes existants dans la base de données.
Transactions	: Dépôt, retrait ou transfert de fonds d'un compte à un autre avec la traçabilité de ces transactions.
Trier	: Tri des résultats obtenues selon un caractère choisi.
A propos	: Renvoie les informations sur la construction du programme.
Quitter	: Fermeture du programme.

- Créer un compte :

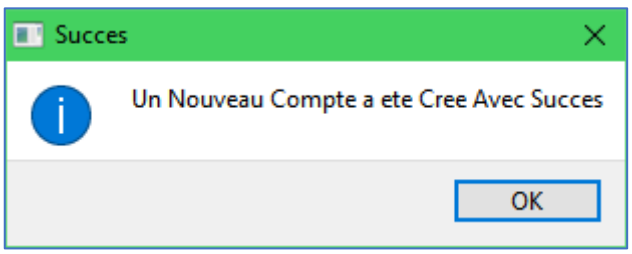
→ Interface :

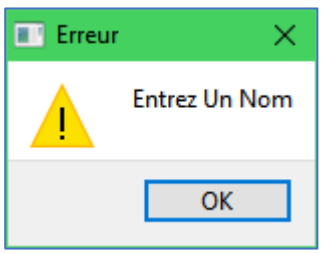


→ Fonctionnement :

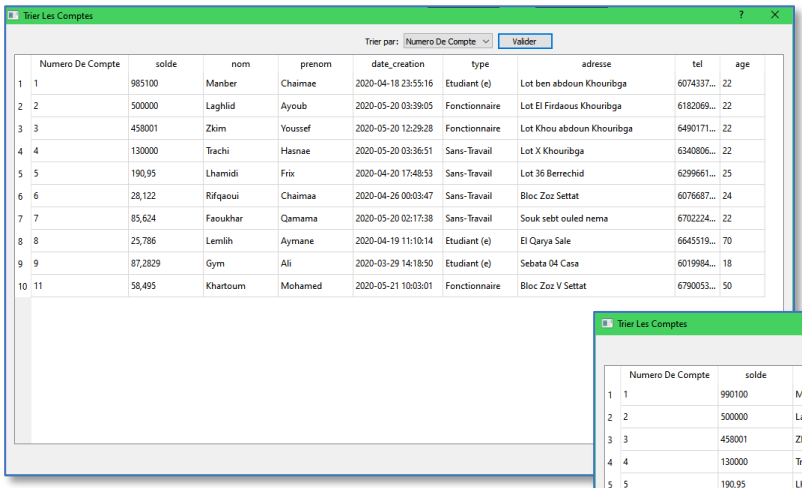


En cas de succès le programme renvoie : et en cas d'échec :





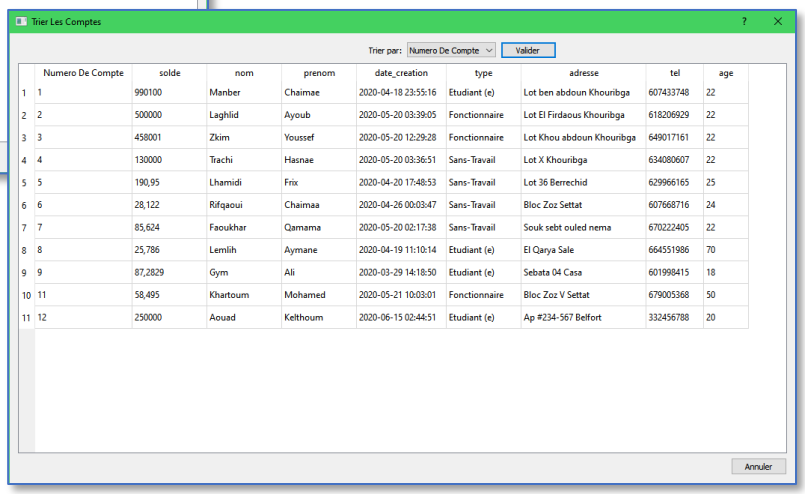
Vérification de l'ajout des comptes dans la base de données :



	Numero De Compte	solde	nom	prenom	date_creation	type	adresse	tel	age
1	1	985100	Manber	Chaimae	2020-04-18 23:55:16	Etudiant (e)	Lot ben abdoun Khouribga	6074337...	22
2	2	500000	Laghlid	Ayoub	2020-05-20 03:39:05	Fonctionnaire	Lot El Firdaous Khouribga	6182069...	22
3	3	458001	Zkim	Youssef	2020-05-20 12:29:28	Fonctionnaire	Lot Khou abdoun Khouribga	6490171...	22
4	4	130000	Trachi	Hasnae	2020-05-20 03:36:51	Sans-Travail	Lot X Khouribga	6340806...	22
5	5	190,95	Lhamidi	Frix	2020-04-20 17:48:53	Sans-Travail	Lot 36 Berrechid	6299661...	25
6	6	28,122	Rifqaoui	Chaimaa	2020-04-26 00:03:47	Sans-Travail	Bloc Zoz Settat	6076687...	24
7	7	85,624	Faoukhar	Qamama	2020-05-20 02:17:38	Sans-Travail	Souk sebt ouled nema	6702224...	22
8	8	25,786	Lemlih	Aymane	2020-04-19 11:10:14	Etudiant (e)	El Qarya Sale	6645519...	70
9	9	87,2829	Gym	Ali	2020-03-29 14:18:50	Etudiant (e)	Sebata 04 Casa	6019984...	18
10	11	58,495	Khartoum	Mohamed	2020-05-21 10:03:01	Fonctionnaire	Bloc Zoz V Settat	6790053...	50

Avant l'ajout.

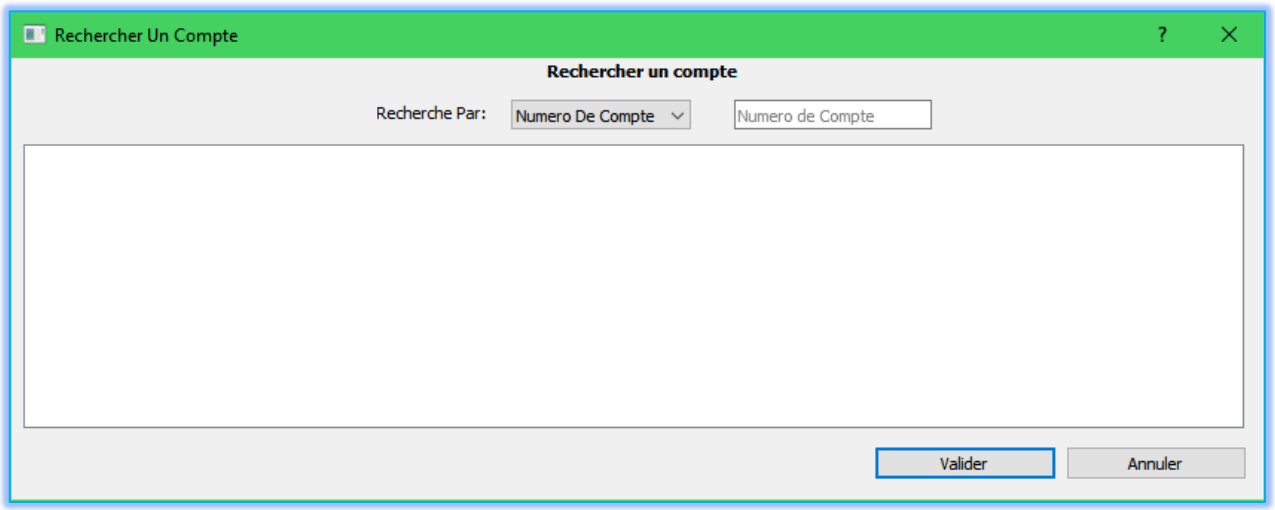
Après l'ajout.



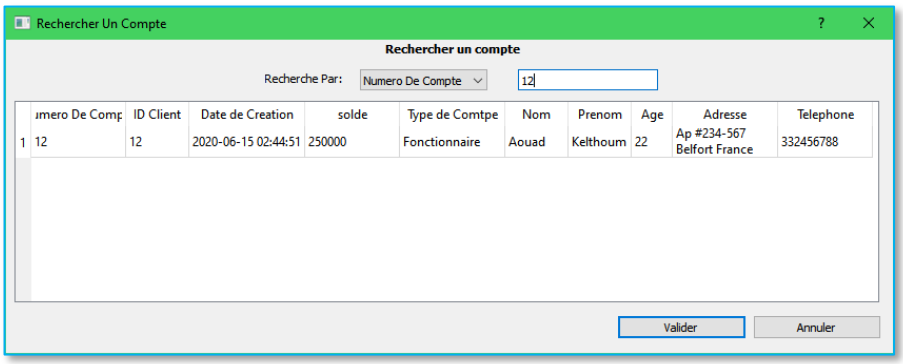
	Numero De Compte	solde	nom	prenom	date_creation	type	adresse	tel	age
1	1	990100	Manber	Chaimae	2020-04-18 23:55:16	Etudiant (e)	Lot ben abdoun Khouribga	607433748	22
2	2	500000	Laghlid	Ayoub	2020-05-20 03:39:05	Fonctionnaire	Lot El Firdaous Khouribga	618206929	22
3	3	458001	Zkim	Youssef	2020-05-20 12:29:28	Fonctionnaire	Lot Khou abdoun Khouribga	649017161	22
4	4	130000	Trachi	Hasnae	2020-05-20 03:36:51	Sans-Travail	Lot X Khouribga	634080607	22
5	5	190,95	Lhamidi	Frix	2020-04-20 17:48:53	Sans-Travail	Lot 36 Berrechid	629966165	25
6	6	28,122	Rifqaoui	Chaimaa	2020-04-26 00:03:47	Sans-Travail	Bloc Zoz Settat	607668716	24
7	7	85,624	Faoukhar	Qamama	2020-05-20 02:17:38	Sans-Travail	Souk sebt ouled nema	670222405	22
8	8	25,786	Lemlih	Aymane	2020-04-19 11:10:14	Etudiant (e)	El Qarya Sale	664551986	70
9	9	87,2829	Gym	Ali	2020-03-29 14:18:50	Etudiant (e)	Sebata 04 Casa	601998415	18
10	11	58,495	Khartoum	Mohamed	2020-05-21 10:03:01	Fonctionnaire	Bloc Zoz V Settat	679005368	50
11	12	250000	Aouad	Kelthoum	2020-06-15 02:44:51	Etudiant (e)	Ap #234-567 Belfort	332456788	20

- Rechercher un compte :

→ Interface :

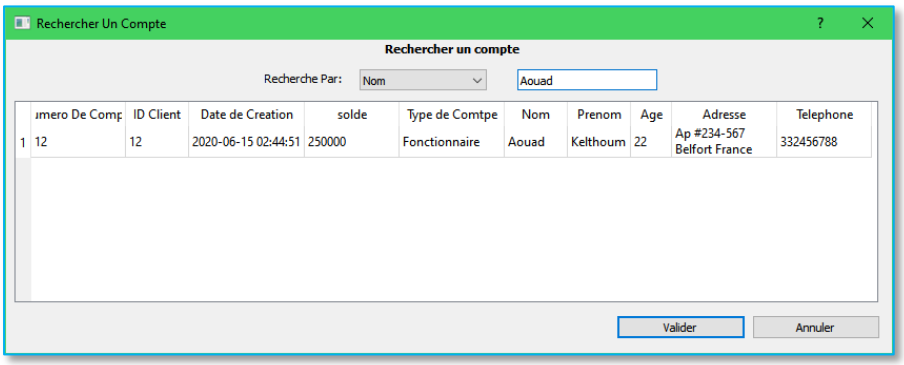


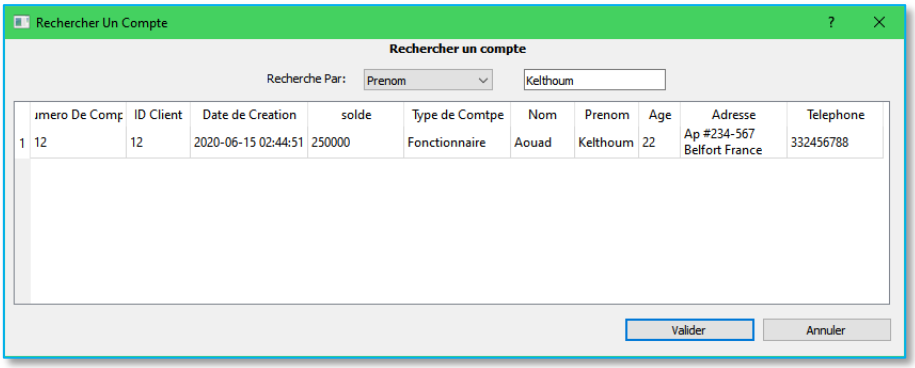
→ Fonctionnement :



Par numéro.

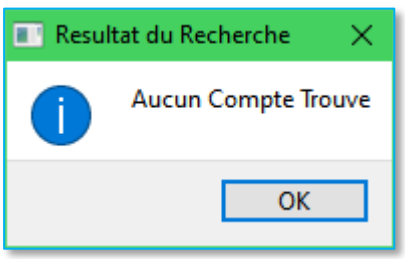
Par prénom.

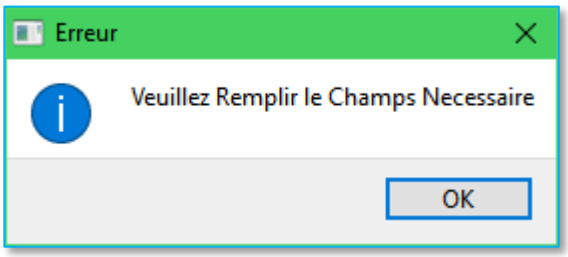




Par nom.

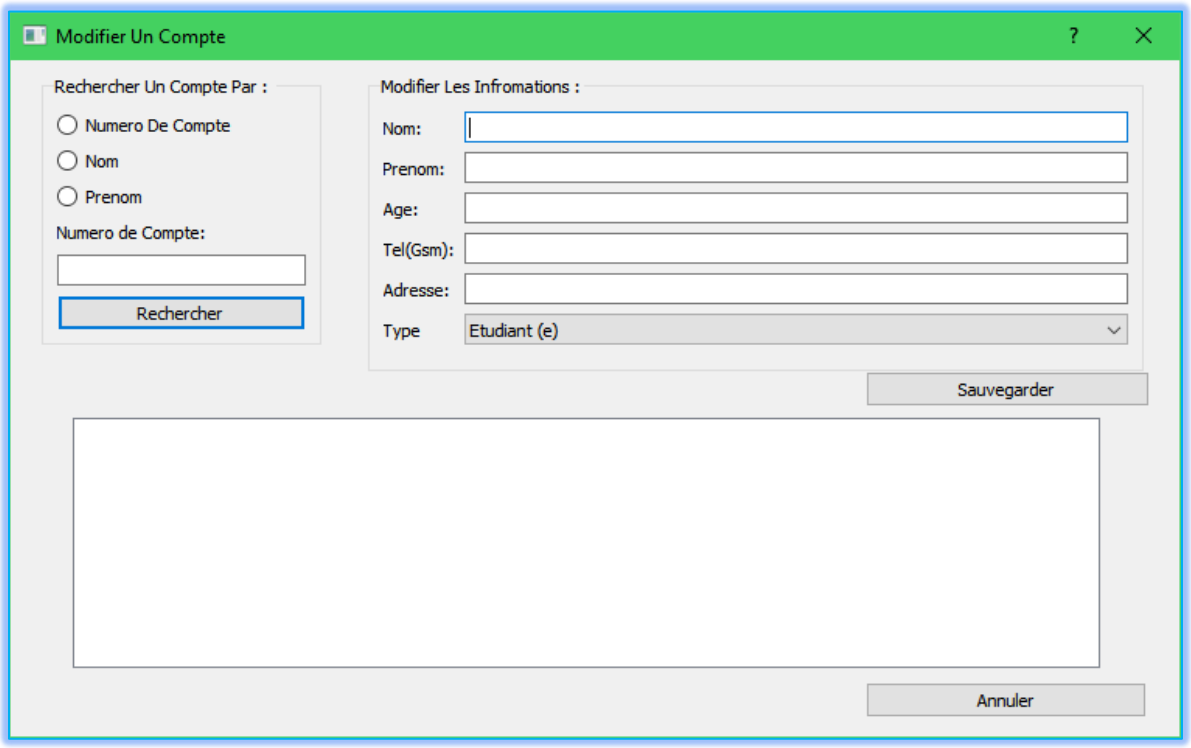
En cas d'échec (aucun résultat trouvé ou le champ de recherche n'est pas rempli), un de ces messages d'erreur suivants s'affiche :



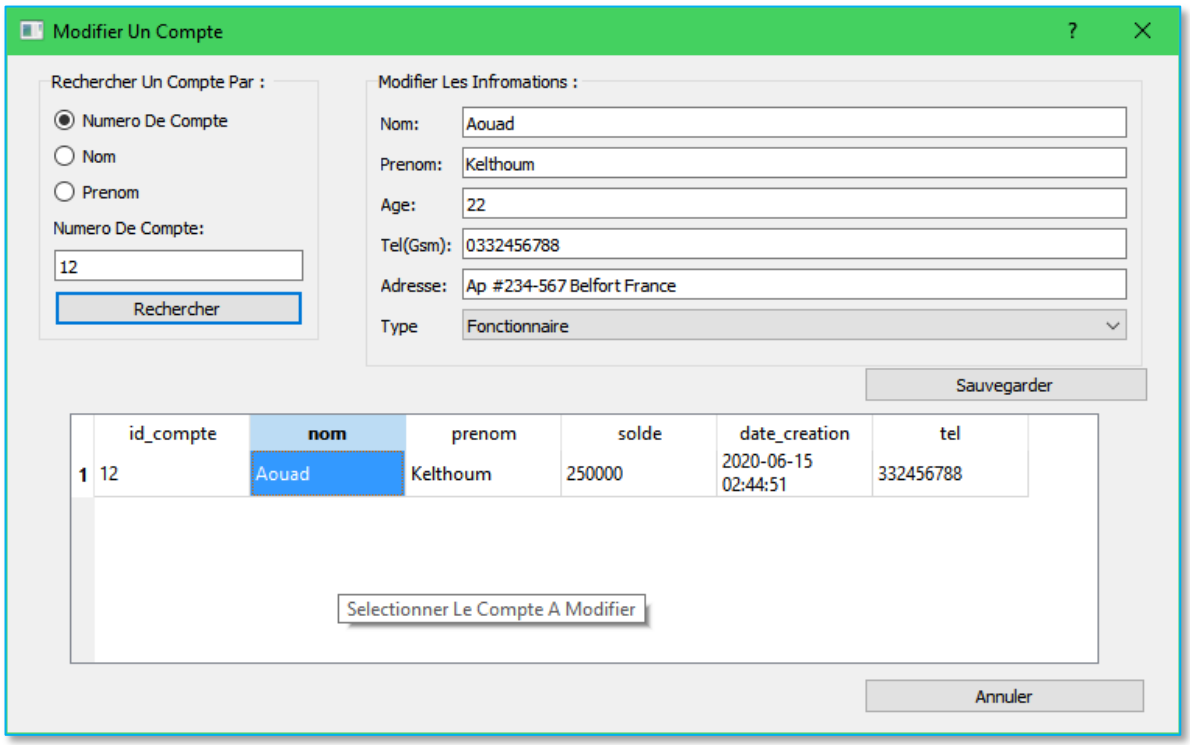


- Modifier un compte :

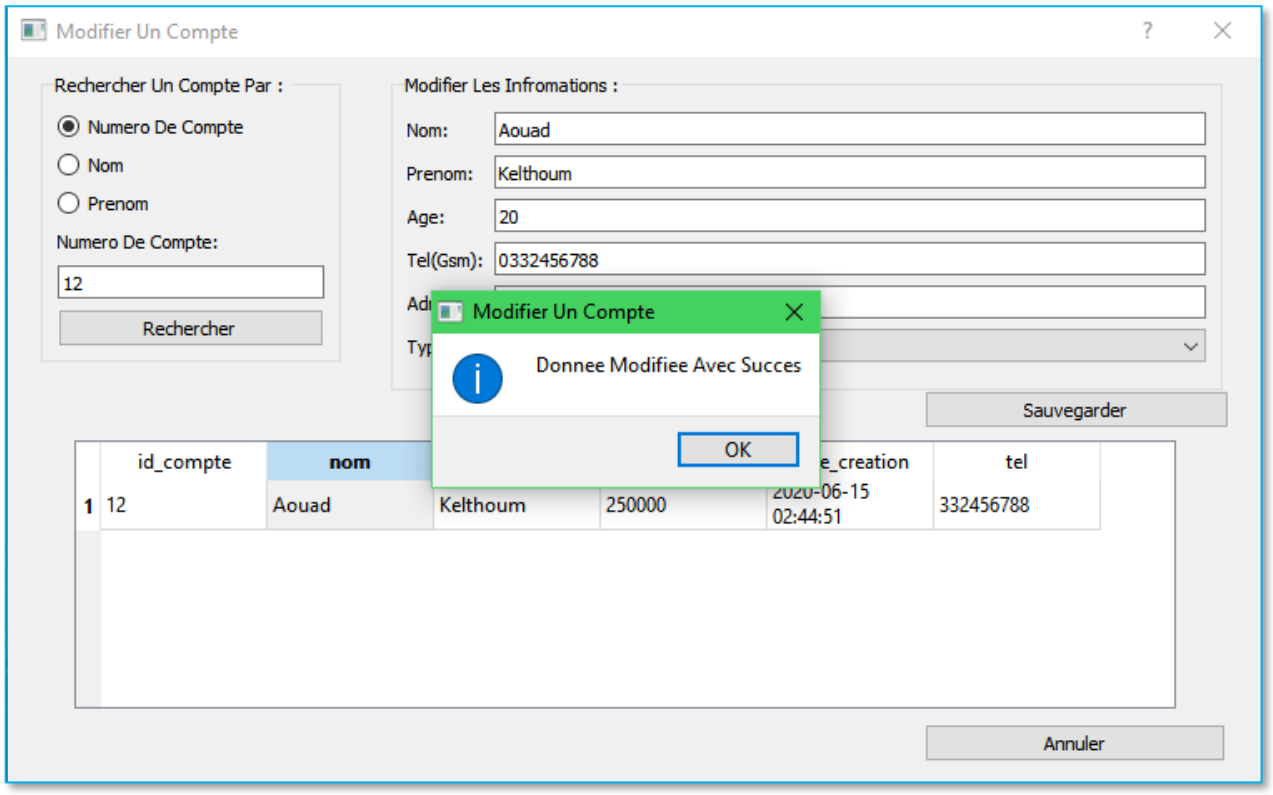
→ Interface :



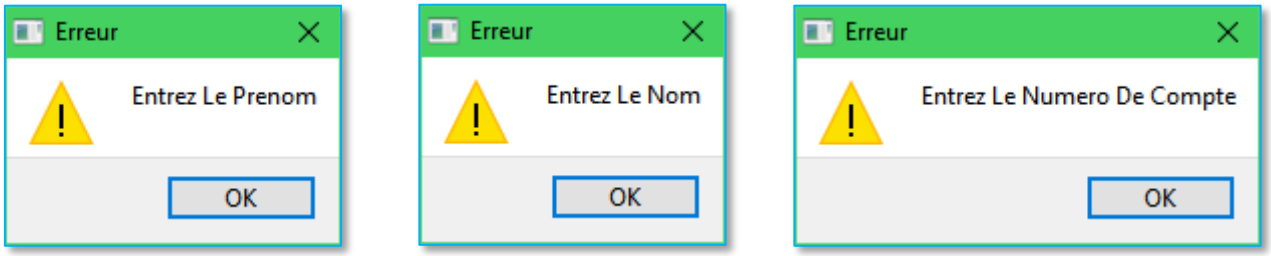
→ Fonctionnement :



NB : Auto-remplissage après le choix du compte a modifier.



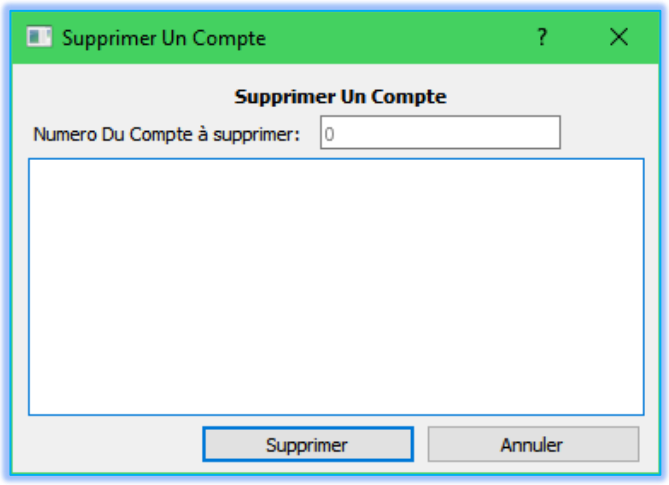
En cas d'échec (exemples des erreurs) :



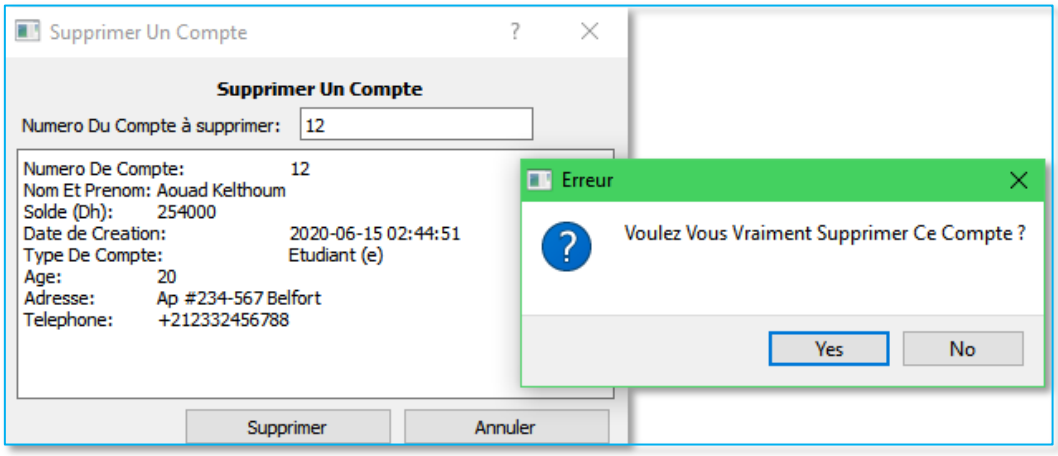
- Supprimer un compte :

→ Interface :

La suppression se fait par numéro.

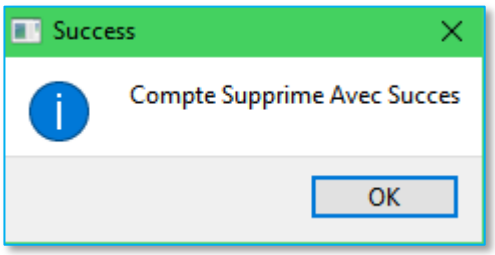


→ Fonctionnement :

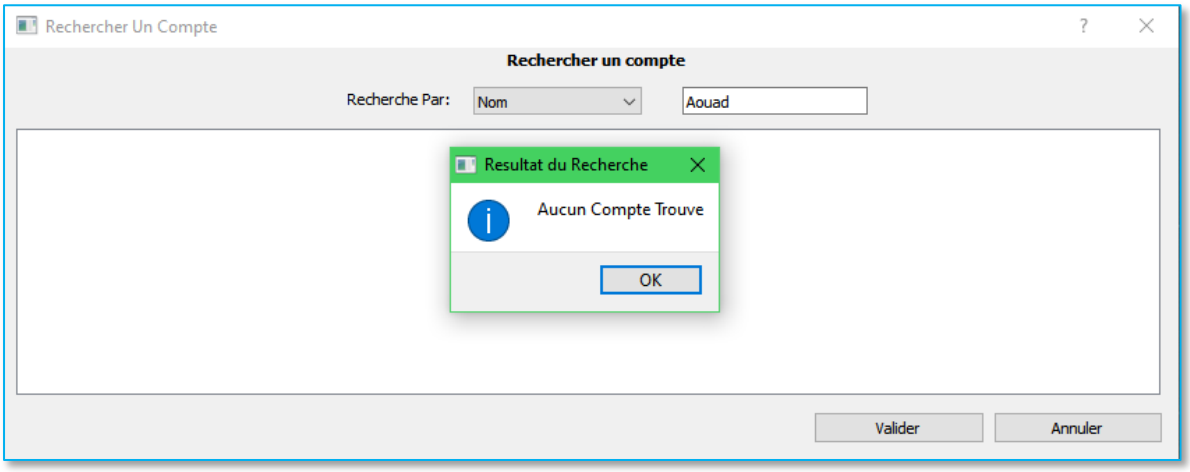


La suppression passe aussi par confirmation pour éviter les fautes de frappes

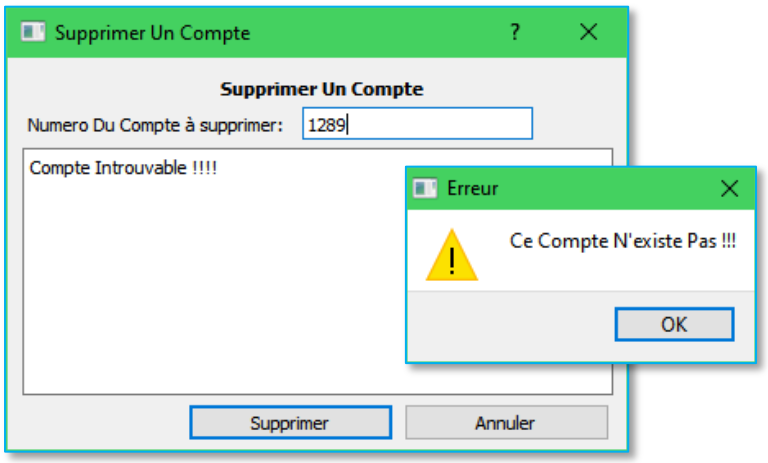
Si le compte existe et la suppression a bien été exécuté le programme renvoie le message de succès suivant :



Vérification de la suppression par recherche sur le compte supprimé :

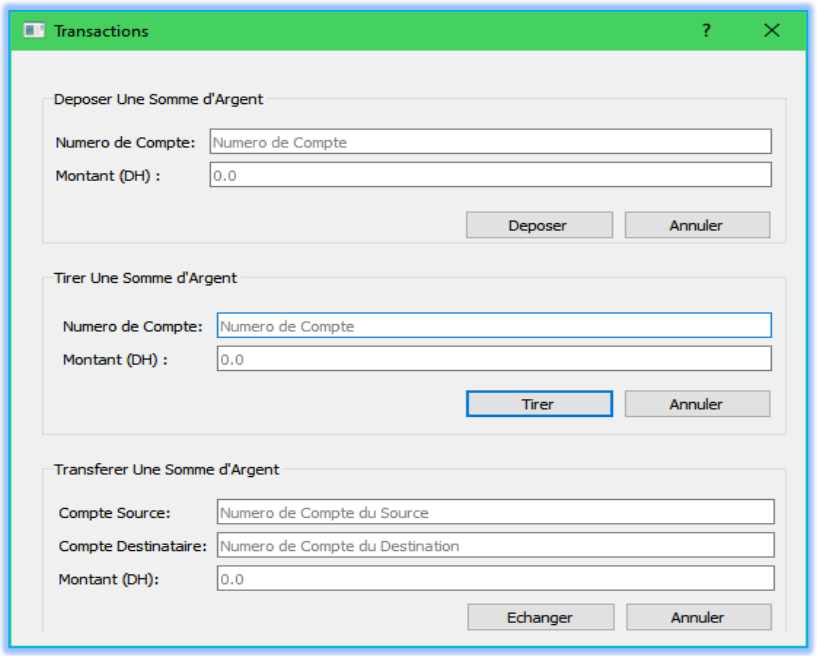


En cas d'échec :



- Transactions :

→ Interface :



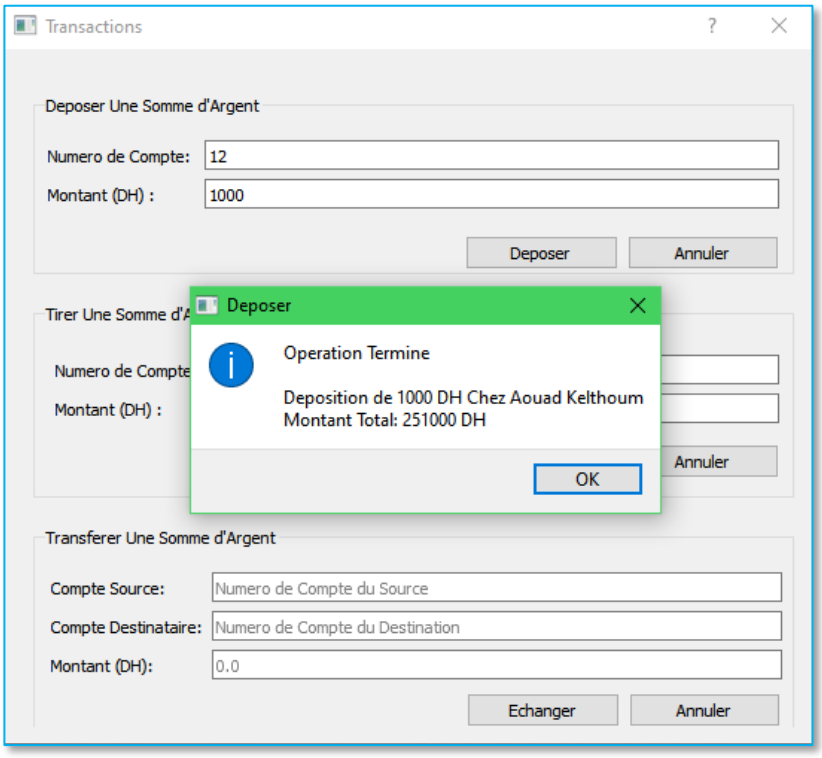
Les transactions se diffèrent entre trois types : dépôt, retrait out transfert de fonds, c'est pour cela on voit ces 3 options dans la fenêtre des transactions.

L'utilisateur doit remplir les champs de son choix

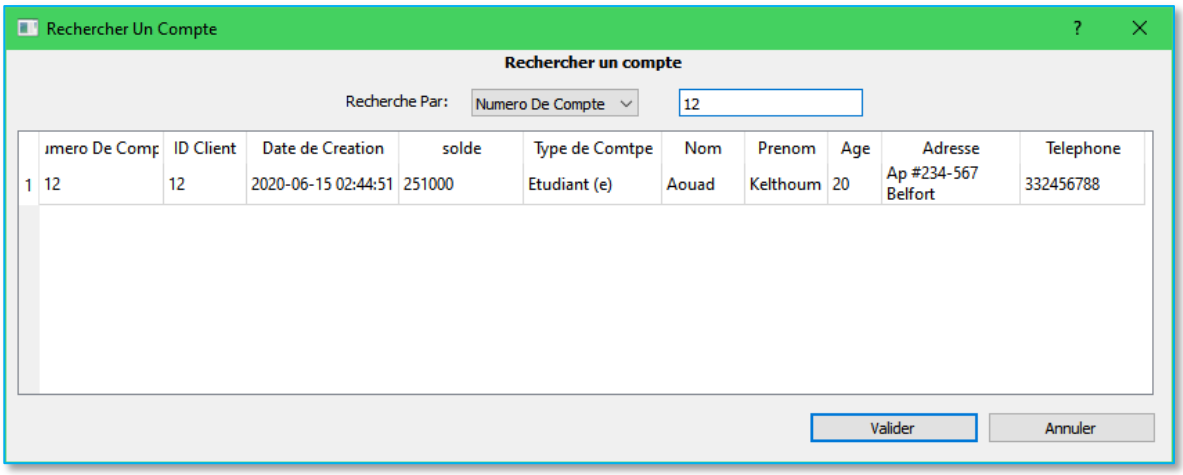
→ Fonctionnement :

- Dépôt :

Une fois l'opération est terminée le programme renvoie le message de vérification du compte crédité avec le montant déposé.



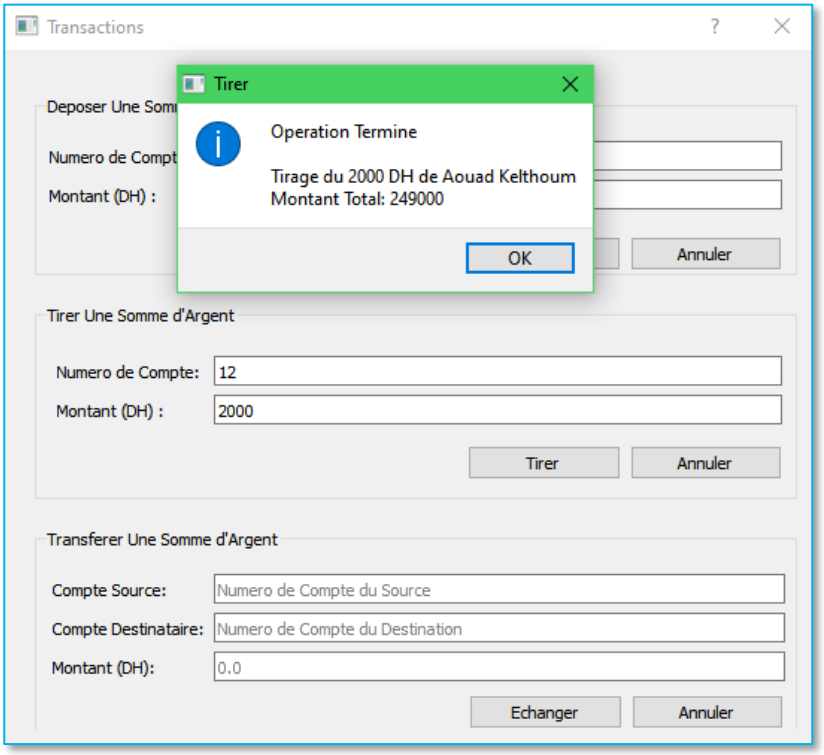
Vérification :



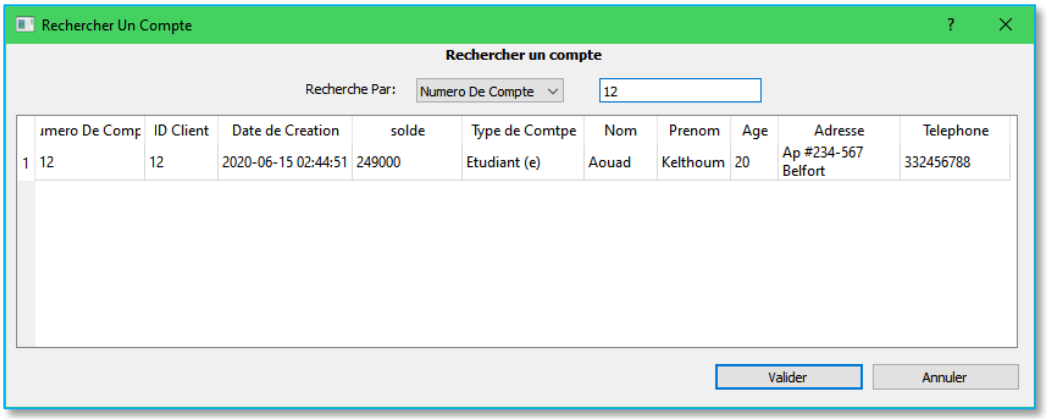
NB : *la vérification se fait par la recherche des informations du compte concerné.*

- Retrait :

Une fois l'opération est terminée le programme renvoie le message de vérification du compte débité avec le montant tiré.

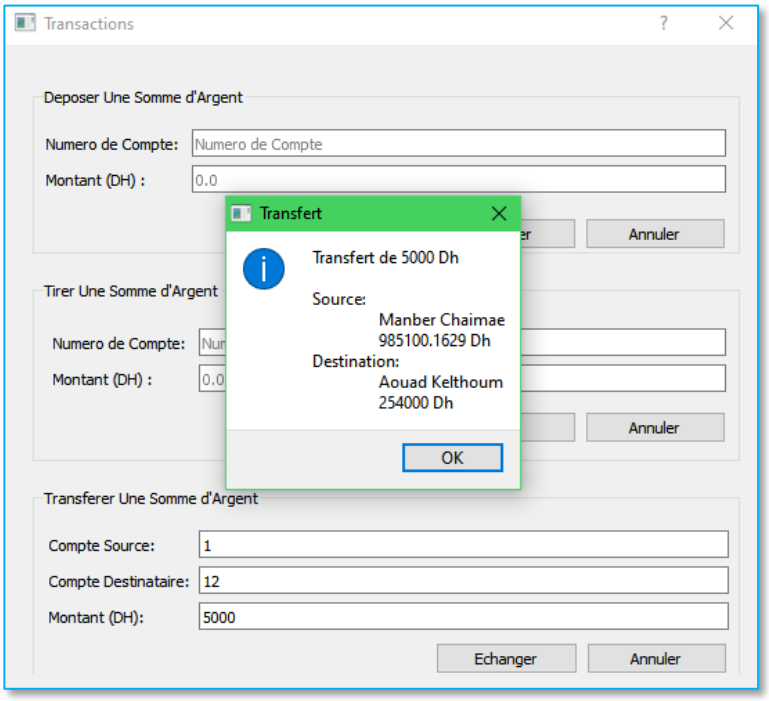


Vérification :



NB : *la vérification se fait par la recherche des informations du compte concerné.*

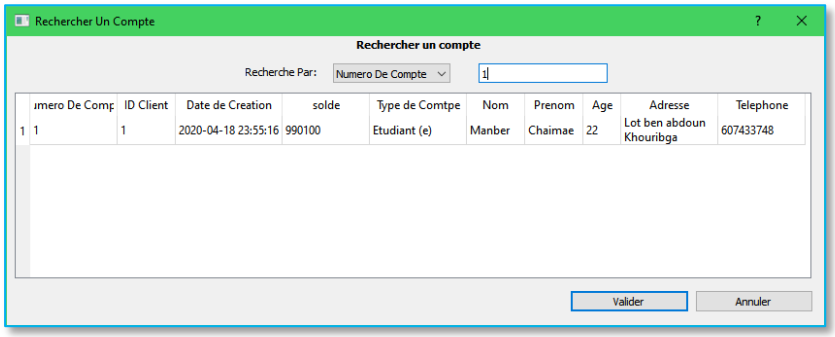
- Transfert :



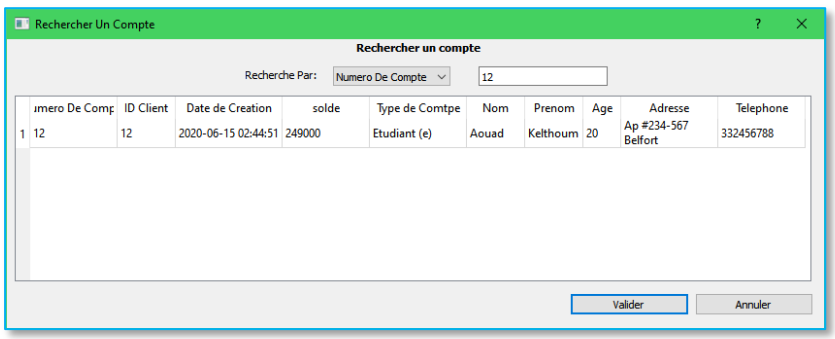
Une fois l'opération est terminée le programme renvoie le message de vérification des comptes avec le montant transféré.

Vérification : Encore une fois la vérification se fera avec l'option de recherche des deux comptes avant et après le transfert de fonds.

AVANT

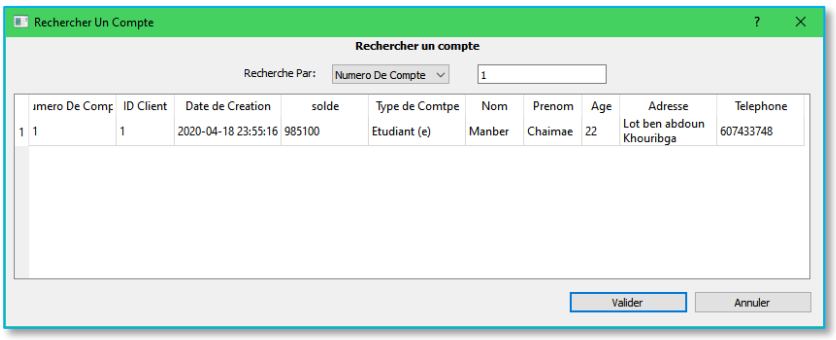


SOURCE

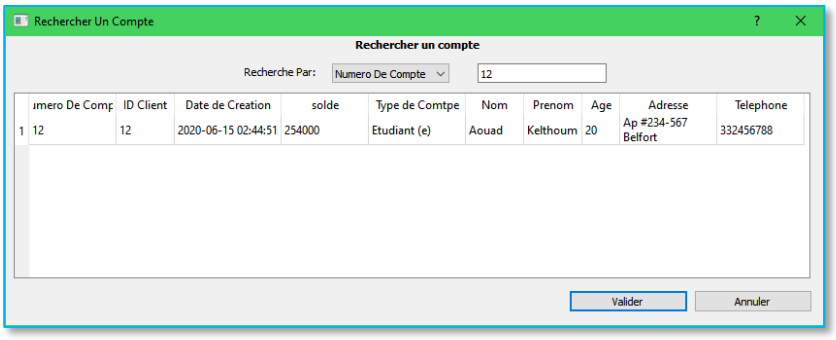


DESTINATION

APRES



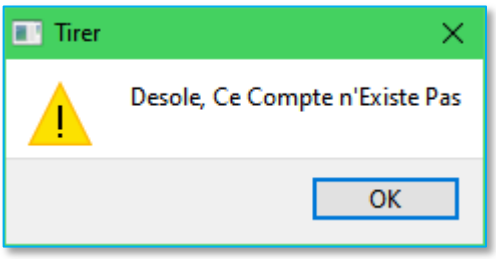
SOURCE



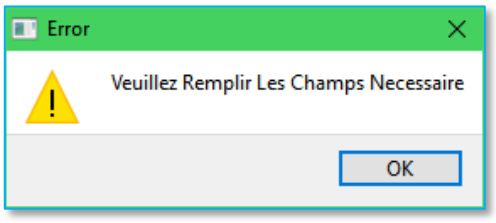
DESTINATION

Les erreurs possibles sont :

- Un ou les deux comptes n'existent pas.

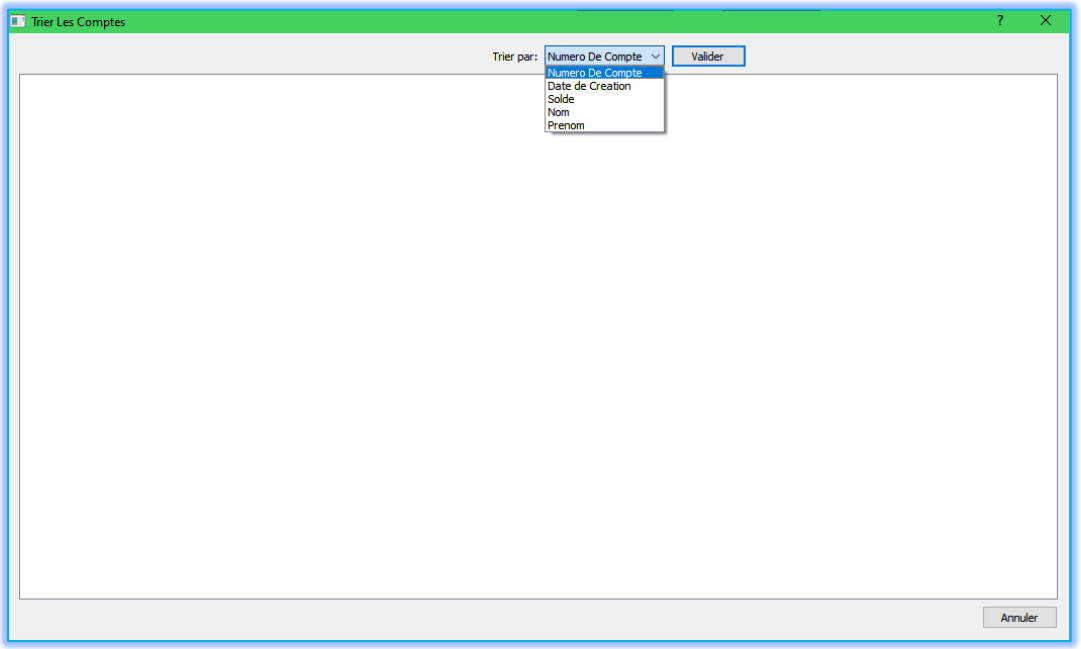


- Le champ du montant est vide.



- Trier :

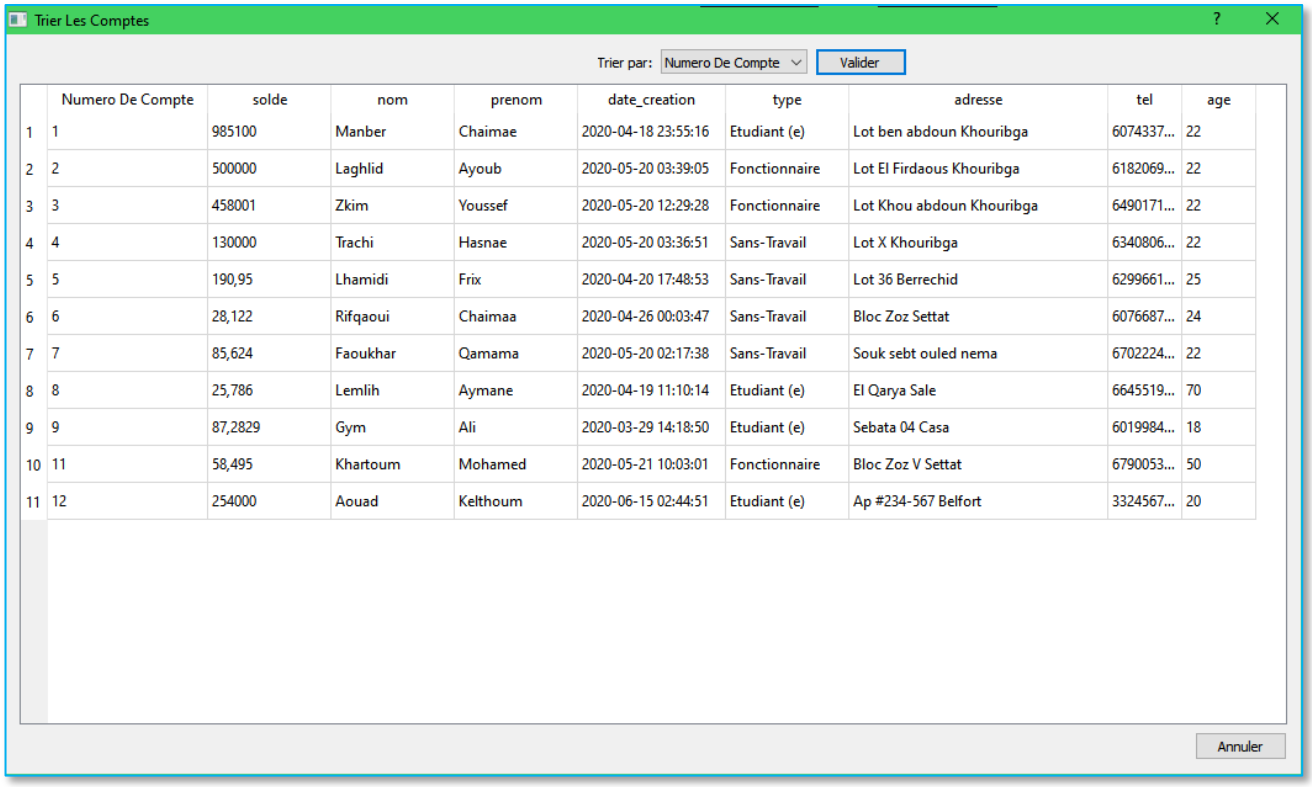
→ Interface :



Pour ajouter des fonctions qui peuvent faciliter l'utilisation de notre programme on a choisi la fonction de Tri selon différents critères ; ID du client, son nom, son prénom, le solde, ou bien la date de création du compte.

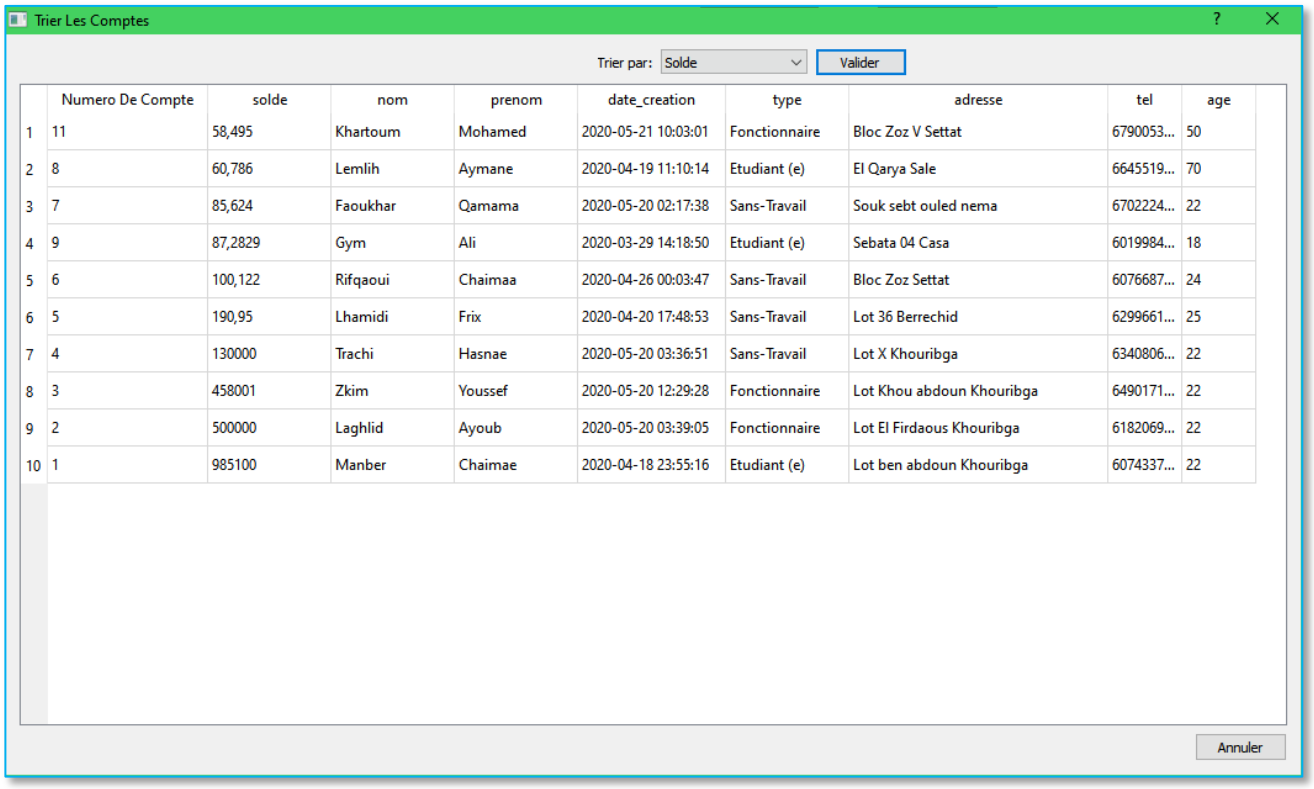
→ Fonctionnement :

Tri selon numéro de compte.



	Numero De Compte	solde	nom	prenom	date_creation	type	adresse	tel	age
1	1	985100	Manber	Chaimae	2020-04-18 23:55:16	Etudiant (e)	Lot ben abdoun Khouribga	6074337...	22
2	2	500000	Laghlid	Ayoub	2020-05-20 03:39:05	Fonctionnaire	Lot El Firdaous Khouribga	6182069...	22
3	3	458001	Zkim	Youssef	2020-05-20 12:29:28	Fonctionnaire	Lot Khou abdoun Khouribga	6490171...	22
4	4	130000	Trachi	Hasnae	2020-05-20 03:36:51	Sans-Travail	Lot X Khouribga	6340806...	22
5	5	190,95	Lhamidi	Frix	2020-04-20 17:48:53	Sans-Travail	Lot 36 Berrechid	6299661...	25
6	6	28,122	Rifqaoui	Chaimaa	2020-04-26 00:03:47	Sans-Travail	Bloc Zoz Settat	6076687...	24
7	7	85,624	Faoukhar	Qamama	2020-05-20 02:17:38	Sans-Travail	Souk sebt ouled nema	6702224...	22
8	8	25,786	Lemlih	Aymane	2020-04-19 11:10:14	Etudiant (e)	El Qarya Sale	6645519...	70
9	9	87,2829	Gym	Ali	2020-03-29 14:18:50	Etudiant (e)	Sebata 04 Casa	6019984...	18
10	11	58,495	Khartoum	Mohamed	2020-05-21 10:03:01	Fonctionnaire	Bloc Zoz V Settat	6790053...	50
11	12	254000	Aouad	Kelthoum	2020-06-15 02:44:51	Etudiant (e)	Ap #234-567 Belfort	3324567...	20

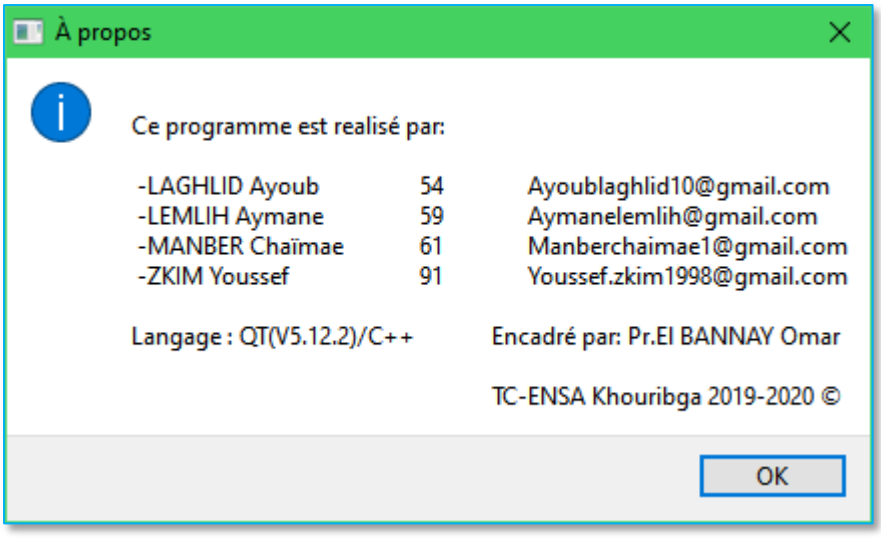
Tri selon le solde



	Numero De Compte	solde	nom	prenom	date_creation	type	adresse	tel	age
1	11	58,495	Khartoum	Mohamed	2020-05-21 10:03:01	Fonctionnaire	Bloc Zoz V Settat	6790053...	50
2	8	60,786	Lemlih	Aymane	2020-04-19 11:10:14	Etudiant (e)	El Qarya Sale	6645519...	70
3	7	85,624	Faoukhar	Qamama	2020-05-20 02:17:38	Sans-Travail	Souk sebt ouled nema	6702224...	22
4	9	87,2829	Gym	Ali	2020-03-29 14:18:50	Etudiant (e)	Sebata 04 Casa	6019984...	18
5	6	100,122	Rifqaoui	Chaimaa	2020-04-26 00:03:47	Sans-Travail	Bloc Zoz Settat	6076687...	24
6	5	190,95	Lhamidi	Frix	2020-04-20 17:48:53	Sans-Travail	Lot 36 Berrechid	6299661...	25
7	4	130000	Trachi	Hasnae	2020-05-20 03:36:51	Sans-Travail	Lot X Khouribga	6340806...	22
8	3	458001	Zkim	Youssef	2020-05-20 12:29:28	Fonctionnaire	Lot Khou abdoun Khouribga	6490171...	22
9	2	500000	Laghlid	Ayoub	2020-05-20 03:39:05	Fonctionnaire	Lot El Firdaous Khouribga	6182069...	22
10	1	985100	Manber	Chaimae	2020-04-18 23:55:16	Etudiant (e)	Lot ben abdoun Khouribga	6074337...	22

- A propos :

→ Interface :



Conclusion

Le projet que nous avons concrétisé tend à répondre au mieux aux besoins bancaires, et d'élaborer des idées, qui peuvent être de plus en plus perfectionnées mais que nous avons malheureusement pas eu la chance de les réaliser toutes faute de temps, regroupant un large éventail de fonctions qui aident à la gestion financière courante notamment en automatisant un certain nombre d'opérations et d'améliorations (saisie rapide des opérations bancaires, recherches, statistiques, historique des transactions, remplacer l'ID par CIN, offrir des réductions et promotions aux clients, etc.), mais aussi à s'adapter à un environnement en constante mutation.

Nous avons travaillé en groupe en organisant des réunions virtuelles (Microsoft Teams). En effet, le bon choix du groupe a eu un effet positif sur la qualité du travail (développer une motivation et esprit d'équipe).

Plutôt que subir cet état de fait comme une fatalité (les contraintes de temps, le confinement et l'obligation de poursuivre les études à distance), nous avons pu à la fin nous en surpasser grâce à une bonne gestion de temps, et organisation du travail.