FSBM

FACULTE DES SCIENCES BEN M'SICK
UNIVERSITÉ HASSAN II DE CASABLANCA

MATHÉMATIQUES & INFORMATIQUE

GÉSTION DES CLIENTS ET DES COMPTES BANCAIRES

RAPPORT DU FIN MODULE LANGAGE C 1

Développement Informatique / I3A / S3 MI
Année Universitaire : 2025/2026

Réalisé Par:

- **OUATGAMMI Rayane**
- **BENDAYA Amine**
- **HAMDI Zakariaa**
- **MOUJAHID Adam**

Responsable de Module:

- **Mme EL FILALI Sanaa**

Encadré Par:

- **Mme ABOURIFA Hanane**

Les Jurys:

- **Mme ABOURIFA Hanane**
- **Mr AMZIL Taoufik**
- **Mr KAISSOUNI Oussama**

Remerciement

Au terme de ce projet de fin de module, nous tenons à exprimer notre profonde reconnaissance et nos sincères remerciements à toutes les personnes qui ont contribué, de près ou de loin, à la réussite de ce travail.

Nous tenons tout d'abord à exprimer notre immense gratitude à la **Faculté des Sciences Ben M'Sik**. Nous remercions l'administration ainsi que l'ensemble du corps professoral pour les efforts inlassables déployés afin de nous offrir un environnement académique d'excellence et une formation de haut niveau. La rigueur de cette institution et la qualité des ressources mises à notre disposition ont été des piliers fondamentaux dans notre parcours d'étudiants en **Filière MI S3**.

Un grand merci à notre professeur, **Madame Sanaa EL FILALI**. Nous la remercions pour la qualité de son enseignement, pour sa disponibilité constante et pour les conseils avisés qu'elle nous a prodigués tout au long de l'année universitaire **2025/2026**. Ses directives précieuses ont été déterminantes pour la structuration de notre application de gestion bancaire.

Nous témoignons également notre gratitude particulière à notre encadrante, **Madame ABOURIFA HANANE**. Son suivi rigoureux, ses encouragements et son expertise technique nous ont permis de surmonter les difficultés rencontrées lors de la phase d'implémentation en langage C.

Nos remerciements s'étendent également aux membres du **jury**. C'est un honneur pour nous de présenter notre travail devant vous. Nous vous remercions pour le temps que vous consacrez à l'évaluation de ce projet et pour l'intérêt que vous portez à nos efforts académiques au sein de l'**Université Hassan II de Casablanca**.

Enfin, nous souhaitons remercier l'ensemble des **membres du groupe** de projet. La réussite de ce travail est le fruit d'une collaboration étroite, d'une entraide mutuelle et d'un partage de connaissances constant. Chacun a su apporter sa pierre à l'édifice pour transformer les contraintes théoriques en un système de gestion fonctionnel et structuré.

Résume

Contexte et Problématique Dans le cadre de la transition numérique des services financiers, la gestion rigoureuse des flux de données est devenue un impératif de sécurité et d'efficacité. Ce projet traite de la problématique de l'informatisation des processus bancaires manuels, qui sont souvent sujets à des incohérences de données et à des failles de contrôle lors des transactions financières.

Objectif du Projet L'objectif majeur est de concevoir un système de gestion bancaire robuste et structuré. Ce projet vise à modéliser les interactions entre les clients et leurs actifs financiers en appliquant les concepts avancés du **langage C**, tels que la manipulation des structures de données complexes et la gestion dynamique de collections via des tableaux.

Principales Fonctionnalités L'application s'articule autour de trois modules interconnectés :

- **Module de Gestion des Clients :** Assure l'intégrité du référentiel client à travers l'ajout sécurisé (unicité de l'ID), la modification, la recherche multicritère et la suppression avec confirmation.
- **Module de Gestion des Comptes :** Permet l'ouverture de comptes liés à des clients existants avec l'application d'une règle métier stricte imposant un solde initial minimal de **1000 DH**.
- **Module des Opérations Financières :** Automatise les flux monétaires incluant les retraits (plafonnés à **700 DH**) et les virements de compte à compte avec vérification de la solvabilité en temps réel.

Langage et Approche Technique Le développement a été réalisé exclusivement en **Langage C**. L'architecture logicielle repose sur une programmation modulaire utilisant des fonctions dédiées pour chaque règle métier, garantissant ainsi une maintenance aisée et une exécution optimisée des algorithmes de recherche et de calcul.

Résultats Obtenus Le résultat final est un système fonctionnel, interactif et sécurisé par des protocoles de validation stricts. Le programme réussit à prévenir les erreurs critiques telles que les doublons d'identifiants ou les découverts non autorisés, offrant ainsi une base fiable pour une éventuelle migration vers une gestion sur fichiers ou une interface graphique

Abstract

the rigorous management of data flows has become a security and efficiency imperative. This project addresses the challenge of computerizing manual banking processes, which are often prone to data inconsistencies and control failures during financial transactions.

Project Objective The primary objective is to design a robust and structured banking management system. This project aims to model interactions between clients and their financial assets by applying advanced **C programming** concepts, such as complex data structures and the management of data collections through arrays.

Key Functionalities The application is built around three interconnected modules:

- **Client Management Module:** Ensures the integrity of the client database through secure addition (ID uniqueness), modification, multi-criteria search, and deletion with confirmation.
- **Account Management Module:** Facilitates the opening of accounts linked to existing clients, enforcing a strict business rule requiring a minimum initial balance of **1,000 DH**.
- **Financial Operations Module:** Automates monetary flows, including withdrawals (capped at **700 DH**) and account-to-account transfers with real-time solvency verification.

Technical Approach Development was conducted exclusively in the **C language**. The software architecture relies on modular programming using dedicated functions for each business rule, ensuring easy maintenance and optimized execution of search and calculation algorithms.

Expected Results The final result is a functional, interactive, and secure system governed by strict validation protocols. The program successfully prevents critical errors, such as duplicate identifiers or unauthorized overdrafts, providing a reliable foundation for future migration to file-based data persistence or a graphical user interface

Liste des figures

Table des matières

Introduction

Depuis plusieurs décennies, le monde assiste à une mutation profonde de ses infrastructures financières. Autrefois basées sur des registres physiques et des processus manuels lents, les banques sont devenues aujourd'hui de véritables entreprises technologiques. Au cœur de cette transformation se trouve la capacité à traiter, stocker et sécuriser des millions de données en temps réel.

Dans le cadre de notre formation en **Filière MI (Mathématiques et Informatique) S3** à la **Faculté des Sciences Ben M'Sik**, l'apprentissage du **Langage C** constitue une étape charnière. Ce langage, bien que de bas niveau, est celui qui permet de comprendre le mieux comment la mémoire d'un ordinateur est structurée. C'est précisément cette puissance que nous avons mobilisée pour concevoir notre système de **"Gestion des Clients et des Comptes Bancaires"** pour l'année universitaire 2025/2026.

Pourquoi est-il crucial de développer un tel système ? La gestion manuelle d'une banque est confrontée à trois défis majeurs que notre logiciel vise à résoudre :

1. **L'Intégrité des Données :** Comment s'assurer qu'un compte bancaire est toujours rattaché à un client réel ? Comment éviter que deux clients ne possèdent le même identifiant ?
2. **La Sécurité Transactionnelle :** En l'absence de contrôles automatisés, rien n'empêche un retrait supérieur au solde disponible. Il est donc impératif d'intégrer des "règles métier" au sein même du code.
3. **L'Efficacité Opérationnelle :** Un employé de banque doit pouvoir retrouver, modifier ou supprimer une information en quelques secondes. L'algorithmique est ici la clé de la performance.

Pourquoi est-il crucial de développer un tel système ? La gestion manuelle d'une banque est confrontée à trois défis majeurs que notre logiciel vise à résoudre :

1. **L'Intégrité des Données :** Comment s'assurer qu'un compte bancaire est toujours rattaché à un client réel ? Comment éviter que deux clients ne possèdent le même identifiant ?
2. **La Sécurité Transactionnelle :** En l'absence de contrôles automatisés, rien n'empêche un retrait supérieur au solde disponible. Il est donc impératif d'intégrer des "règles métier" au sein même du code.
3. **L'Efficacité Opérationnelle :** Un employé de banque doit pouvoir retrouver, modifier ou supprimer une information en quelques secondes. L'algorithmique est ici la clé de la performance.

Le choix du **Langage C** pour ce projet n'est pas fortuit. Contrairement à des langages de plus haut niveau, le C impose une rigueur absolue.

- **La modularité :** En découpant notre projet en fonctions spécifiques (ajouterClient, effectuerVirement), nous apprenons à construire des systèmes maintenables.
- **La gestion des structures :** L'utilisation de struct nous permet de modéliser des objets complexes du monde réel (un client avec son nom, son prénom, son métier) à l'intérieur de la mémoire vive de la machine.

L'objectif principal de ce projet est de livrer une application console stable, sécurisée et ergonomique. Pour atteindre cette ambition, nous avons défini un périmètre d'action divisé en trois piliers :

- **Le Pilier Relationnel (Gestion des Clients) :** Créer une base de données volatile capable de stocker les profils complets des clients. Chaque ajout est soumis à une vérification d'unicité de l'Id_client.
- **Le Pilier Comptable (Gestion des Comptes) :** Permettre l'ouverture de comptes avec une contrainte de **solde initial de 1000 DH**, garantissant une réserve de liquidité pour la banque.
- **Le Pilier Transactionnel (Opérations) :** Automatiser les mouvements de fonds avec deux garde-fous : un plafond de retrait de **700 DH** et une interdiction de découvert lors des virements entre comptes.

Ce rapport a été conçu comme un guide technique et fonctionnel retraçant chaque étape de notre réflexion. Il s'articule autour des chapitres suivants :

- **Analyse Conceptuelle :** Où nous exposons la définition de nos structures de données et le choix des variables.
- **Architecture du Système :** Une présentation de l'interface utilisateur et de la logique de navigation par menus circulaires.
- **Algorithmique et Règles Métier :** Un zoom sur les fonctions critiques, expliquant comment le code valide chaque transaction avant de l'exécuter.
- **Tests et Fiabilité :** Une démonstration par l'exemple des scénarios de réussite et de gestion d'erreurs (identifiants inexistants, soldes insuffisants).
- **Bilan et Perspectives :** Une conclusion sur les acquis techniques et une ouverture sur les améliorations possibles, telles que la sauvegarde des données dans des fichiers physiques pour éviter la perte d'informations à la fermeture du programme.

En somme, cette introduction marque le début d'une exploration technique où la rigueur mathématique rencontre la logique informatique. Nous vous invitons à présent à découvrir le cœur du système que nous avons bâti sous la direction de **Mme Sanaa EL FILALI** et avec l'encadrement de **Mme ABOURIFA HANANE**.

CHAPITRE 1

Contexte Général

CHAPITRE 1 : CONTEXTE GÉNÉRAL ET ANALYSE DU PROBLÈME

I. Description du problème réel : Les limites de la gestion manuelle

Dans le fonctionnement traditionnel d'une agence bancaire non informatisée, la gestion des flux d'informations repose sur des registres papier ou des fichiers disparates. Cette méthode artisanale engendre des problématiques critiques :

1. **L'opacité et l'accès lent à l'information :** Retrouver le dossier d'un client parmi des centaines d'autres pour vérifier son adresse ou son métier prend un temps considérable.
2. **L'insécurité des fonds :** Sans un système de calcul automatisé, les erreurs lors des retraits ou des virements sont fréquentes. Un agent peut autoriser par inadvertance un retrait dépassant le solde disponible, mettant ainsi la banque en péril financier.
3. **L'absence de traçabilité :** Il est difficile de garder une trace fiable de la date d'ouverture d'un compte ou de l'historique des modifications des coordonnées d'un client.
4. **La redondance et les doublons :** Sans vérification d'unicité, un même client peut être enregistré plusieurs fois avec des identifiants différents, rendant la base de données incohérente.

II. Identification des Besoins

Pour pallier ces dysfonctionnements, le projet doit répondre à des besoins spécifiques exprimés par les utilisateurs (agents bancaires) :

- **Centralisation :** Regrouper toutes les données (noms, professions, téléphones, soldes) dans un système unique et structuré.
- **Automatisation :** Les calculs de solde après un retrait ou un virement doivent être instantanés et sans erreur.
- **Contrôle :** Le système doit agir comme un filtre, refusant systématiquement les opérations qui ne respectent pas les règles de l'établissement.
- **Interactivité :** Offrir une interface simple où l'utilisateur n'a qu'à choisir une option dans un menu pour exécuter une tâche complexe.

III. Objectifs Fonctionnels

Les objectifs fonctionnels décrivent ce que le logiciel "doit faire". Pour ce projet, ils se déclinent en trois axes majeurs :

Gestion du Référentiel Client :

- Permettre l'ajout de nouveaux clients avec un identifiant unique (Id_client).
- Offrir la possibilité de mettre à jour les informations (téléphone, profession) ou de supprimer un client (avec confirmation).
- Rechercher un client instantanément par son nom ou son ID.

Pilotage des Comptes Bancaires :

- Ouvrir un compte lié à un client existant.
- Appliquer la règle du **solde minimum de 1000 DH** à l'ouverture pour garantir une base financière.
- Consulter l'état des comptes à tout moment.

Sécurisation des Opérations Financières :

- Exécuter des retraits en respectant le **plafond de 700 DH** par transaction.
- Réaliser des virements de compte à compte, en s'assurant que le solde émetteur est suffisant avant de créditer le destinataire.

IV. Limites du Projet

Il est important de préciser ce que le projet ne fait pas, afin de définir clairement son périmètre :

- Persistance des données : Dans cette version actuelle en langage C, les données sont stockées en mémoire vive (RAM). Par conséquent, elles sont effacées dès que l'on quitte le programme.
- Multi-utilisateurs : Le système est conçu pour un usage monoposte ; il ne gère pas les accès simultanés de plusieurs agents via un réseau.
- Interface graphique : L'application utilise une interface "Console" (texte), sans fenêtres ni boutons graphiques.
- Historique exhaustif : Bien que les soldes soient mis à jour, le programme ne conserve pas (dans cette version) la liste détaillée de toutes les transactions passées pour chaque compte.

V. CONCLUSION

Ce premier chapitre nous a permis de poser le diagnostic d'une gestion bancaire manuelle, devenue obsolète face aux exigences modernes de rapidité et de fiabilité. L'analyse du contexte a mis en évidence des risques critiques : la redondance des informations, la vulnérabilité aux erreurs humaines de calcul et l'absence de contrôles de sécurité en temps réel.

Nous avons défini les objectifs de notre projet qui ne se limitent pas à une simple informatisation, mais visent la conception d'un système structuré capable de garantir l'intégrité des données financières. Le cahier des charges est désormais clair : l'application doit gérer le cycle de vie des **clients** et des **comptes** tout en appliquant rigoureusement des règles métier, telles que le solde minimum de **1000 DH** et le plafond de retrait de **700 DH**.

La problématique étant clairement identifiée et les besoins fonctionnels établis, la question centrale devient maintenant d'ordre technique : **comment traduire ces exigences métiers en un programme informatique performant ?**

C'est l'objet du chapitre suivant, où nous détaillerons la phase de **conception technique**. Nous y exposerons les choix architecturaux, notamment la modélisation des données à travers les structures Client et Compte, qui constituent la colonne vertébrale de **notre application en langage C.**

Conception & Modélisation

CHAPITRE 2 : CONCEPTION ET MODÉLISATION

L'étape de conception vise à traduire les besoins fonctionnels en une architecture technique cohérente. Avant d'écrire la moindre ligne de code en langage C, il est impératif de définir la structure des données, les relations entre les entités et la logique algorithmique qui régira le système.

I. Analyse et Spécification des Besoins

Le système doit permettre à un utilisateur (agent bancaire) de gérer les entités via une interface console. Les besoins se décomposent en trois axes principaux :

Gestion administrative des clients :

- Ajouter, modifier, supprimer et rechercher des clients.
- **Contrainte :** Unicité de l'identifiant client (Id_client).

Gestion des avoirs (Comptes) :

- Créer des comptes pour des clients existants et les clôturer si nécessaire.
- **Contrainte :** Le solde à l'ouverture doit être impérativement supérieur ou égal à **1000 DH**.

Opérations financières :

- Effectuer des retraits et des virements.
- **Contrainte :** Le retrait est limité à **700 DH** par opération.
- **Contrainte :** Vérification stricte du solde avant validation d'un virement.

II. Modélisation UML (Unified Modeling Language)

Pour visualiser la structure du projet, nous utilisons des diagrammes UML simplifiés adaptés à la programmation procédurale.

2.2.1 Définition

Le **Diagramme de Cas d'Utilisation (Use Case Diagram)** est un outil de modélisation UML qui permet de représenter les fonctionnalités du système du point de vue des utilisateurs. Il définit **"qui fait quoi"** sans entrer dans les détails techniques du code. Son but est de capturer les besoins fonctionnels et d'identifier les interactions entre les acteurs externes et les différentes missions du système bancaire.

2.2.2 Analyse du Diagramme

Votre diagramme structure l'application autour de trois acteurs principaux et d'un système de persistance.

A. Les Acteurs et leurs Rôles

Le système distingue trois profils d'utilisateurs (Acteurs) :

- **Le Client :** Il interagit avec ses comptes pour effectuer des virements, ouvrir un compte, faire des dépôts ou vérifier son solde.
- **L'Agent Bancaire :** Responsable de la gestion administrative, il peut ajouter des clients, fermer des comptes ou modifier les données existantes.
- **L'Admin :** Possède les droits les plus élevés, notamment pour la suppression de données et la consultation de l'historique complet des transactions.
- **Le Distributeur :** Un système externe qui interagit spécifiquement pour la fonction "Effectuer retrait".

B. Les Relations « Include » (Dépendances)

Une caractéristique majeure de votre conception est la centralisation de la sécurité via l'**Authentification PIN**.

- Presque tous les cas d'utilisation (Virement, Dépôt, Ajout, Modification, etc.) pointent vers "Authentification PIN" avec une relation **« include »**.

C. La Persistance (Base de Données)

Le diagramme montre que le processus d'**enregistrement des données** est directement lié à la **Base de donnée**. C'est ici que la gestion des fichiers (dont nous avons parlé précédemment) intervient : chaque action validée par un acteur est immédiatement archivée pour garantir que les informations ne soient pas perdues.

2.2.3 Synthèse Logique

Ce diagramme prouve que votre application est sécurisée et hiérarchisée. Le choix de séparer les rôles (Client vs Agent vs Admin) permet de respecter les principes de sécurité bancaire, où chaque utilisateur n'accède qu'aux fonctions qui lui sont autorisées, le tout protégé par un code PIN obligatoire.

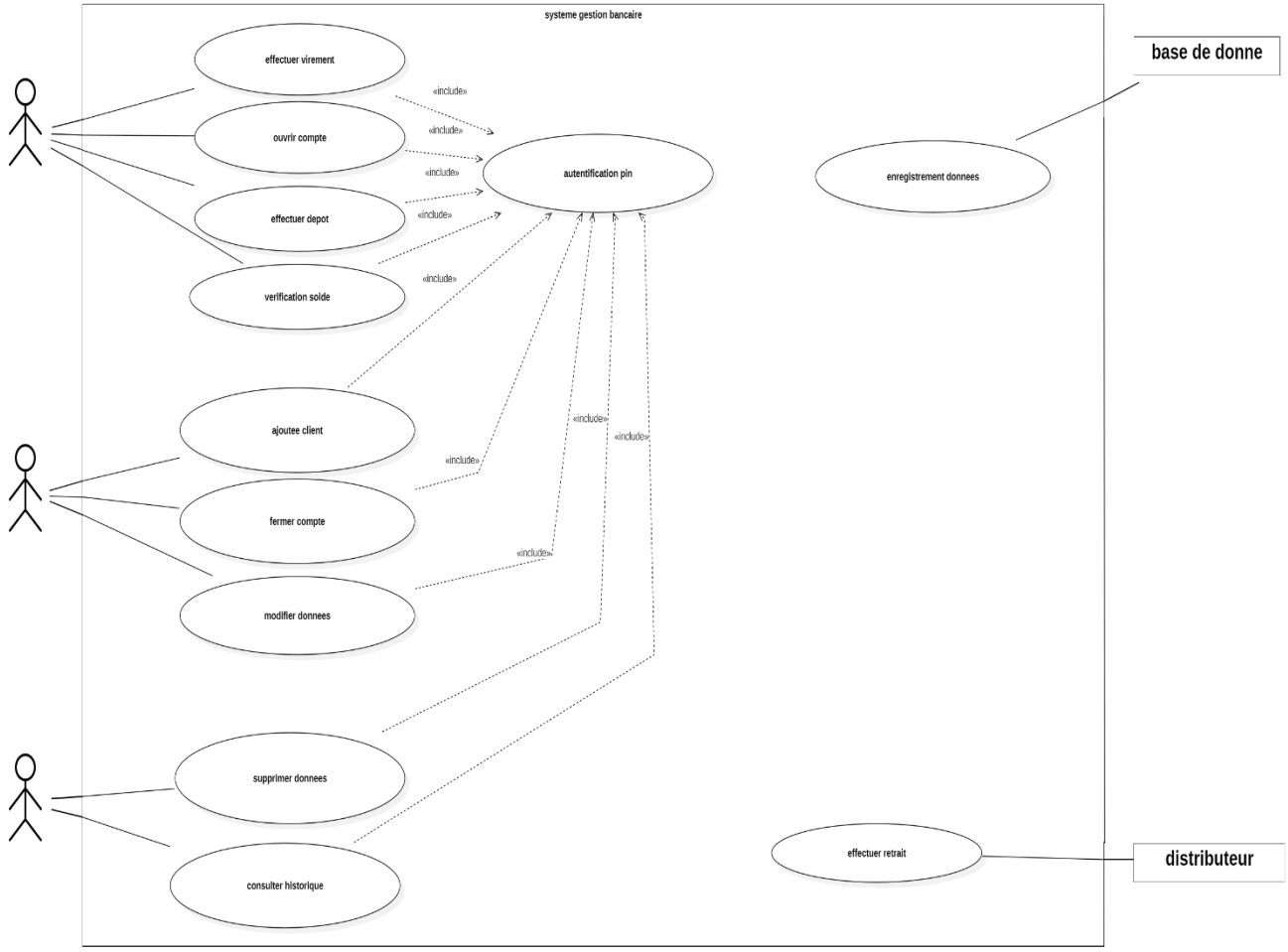

Figure 1 : Diagramme de cas d'utilisation

2.3 Modélisation Statique : Le Diagramme de Classe

2.3.1 Définition et Rôle

Le **Diagramme de Classe** est le pilier de la conception technique. Dans le contexte de notre projet en langage C, il sert à définir les **structures (**struct**)** et les **fonctions** associées. Il illustre la hiérarchie des données et les relations (associations) entre les différentes entités du système bancaire.

2.3.2 Analyse des Entités (Classes)

Le diagramme se compose de quatre classes principales qui structurent le code :

1. La Classe client **(Référentiel)**

C'est le point d'entrée du système.

- **Attributs :** Elle stocke l'identité (id_client), le nom, le prenom, la profession et le num_tel.

- **Méthodes (Fonctions) :** Elle regroupe les fonctions de gestion de base : ajouter(), modifier(), supprimer() et rechercher().

2. La Classe compte **(Actifs financiers)**

Elle représente le coffre-fort numérique du client.

- **Attributs :** On y trouve l'id_compte, le lien avec le client (id_client), le solde_base et la date_ouverture.
- **Méthodes :** Elle gère le cycle de vie du compte avec creer(), consulter(), fermer() et la fonction critique verifierSolde().

3. La Classe operation **(Journalisation)**

Cette classe gère les flux et l'historique.

- **Attributs :** Chaque mouvement possède un id_operation, un type (type), un montant et une date.
- **Méthodes :** Elle contient la logique métier pure : effectuerRetrait(), effectuerVirement() et effectuerLimites() (pour le contrôle du plafond de 700 DH).

4. La Classe admin **(Superviseur)**

Elle joue le rôle de gestionnaire global (souvent représentée par le main ou une structure de contrôle).

- **Attributs :** Elle possède des compteurs globaux (nb_client, nb_comptes) et des tableaux (listes) de clients et de comptes.
- **Méthodes :** Elle orchestre les grands modules de l'application.

2.3.3 Analyse des Relations et Cardinalités

C'est ici que l'on comprend comment les structures "parlent" entre elles :

1. **Relation Client - Compte (Possède) :**
    - **Cardinalité :** 1 vers +*.
    - **Explication :** Un client peut posséder plusieurs comptes (compte courant, épargne, etc.), mais un compte appartient à un seul et unique client. La flèche avec le losange plein (composition/agrégation forte) indique qu'un compte ne peut pas exister sans un client.
2. **Relation Compte - Operation (Assurer) :**
    - **Cardinalité :** 1 vers 0..*.
    - **Explication :** Un compte peut n'avoir aucune opération (nouveau compte) ou en avoir une infinité. Chaque opération est rattachée à un compte spécifique.

3. **Relation Admin - Client/Compte (Gérer/Ajouter) :**
    - L'administrateur a une vue d'ensemble et contrôle la création et la maintenance de toutes les entités.

2.3.4 Traduction Technique vers le Langage C

Pour votre rapport, il est bon de préciser que :

- Les **Classes** deviendront des typedef struct.
- Les **Attributs** seront les membres de ces structures.
- Les **Méthodes** seront des fonctions indépendantes prenant ces structures en paramètres.

Note de conception : Ce diagramme respecte parfaitement la logique de séparation des tâches. La structure operation isolée permet de créer facilement le fichier historique.txt dont nous avons discuté précédemment.

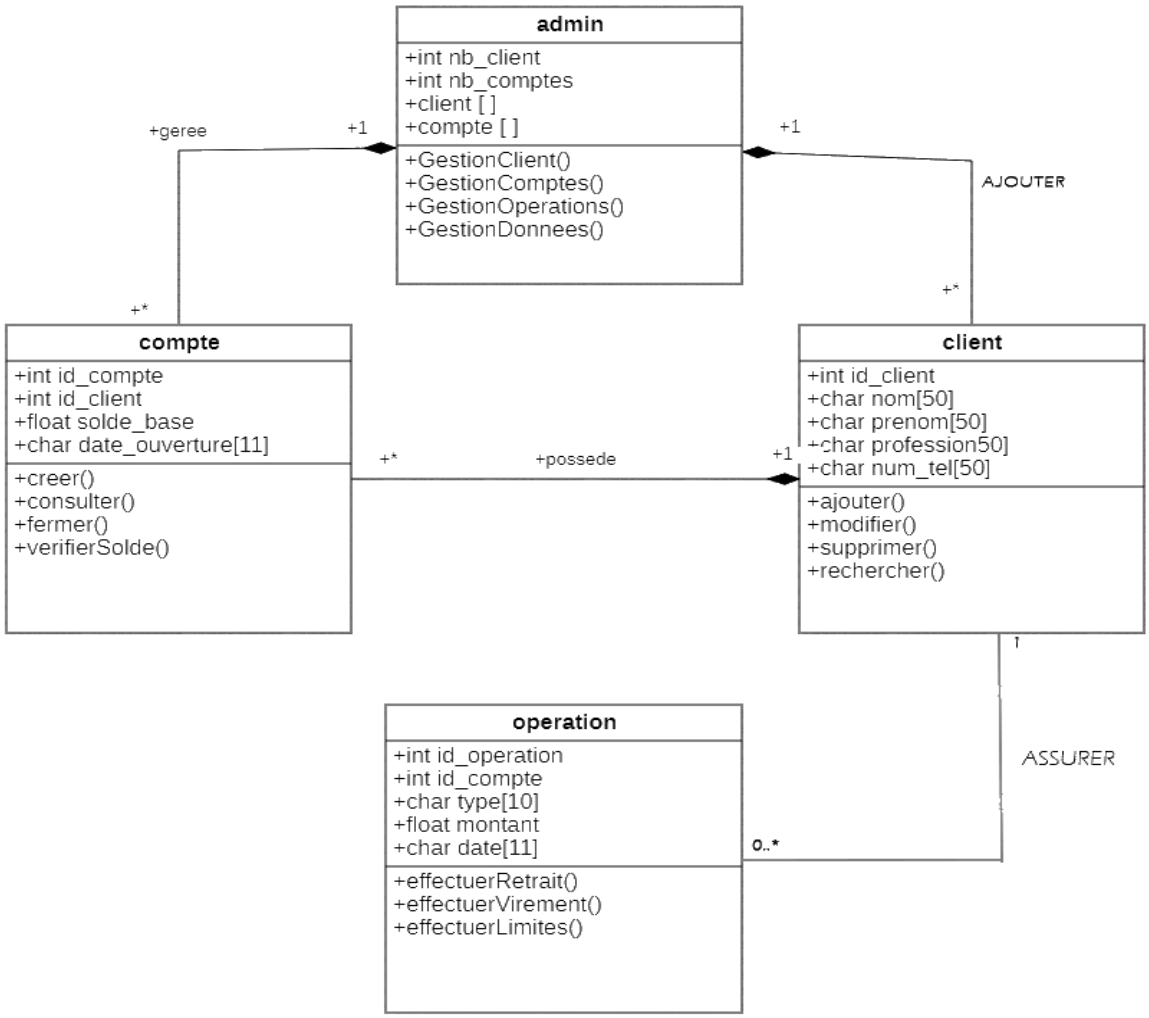

Figure 2: Diagramme des classes

2.4 Modélisation Dynamique : Les Diagrammes de Séquence

2.4.1 Définition

Le **Diagramme de Séquence** est un diagramme UML d'interaction qui montre comment les objets communiquent entre eux dans un ordre chronologique. Il illustre le cheminement des messages entre les différents composants du système (Interface, Validation, Stockage) pour accomplir une tâche spécifique. C'est l'outil idéal pour visualiser la logique de contrôle et le traitement des erreurs (scénarios alternatifs).

2.4.2 Analyse du Diagramme de Séquence : "Ajouter Client"

Ce diagramme détaille le processus de création d'un nouveau profil client dans le système.

- **Acteurs et Objets :** L'interaction implique l'**Agent Bancaire**, l'**Interface** utilisateur, le module de **Validation** et le système de **Stockage** (fichiers).
- **Le Verrou d'Unicité :** Dès que l'ID est saisi, le système interroge le Stockage. Si l'ID existe déjà, le fragment **alt** (alternative) arrête le processus et affiche un message d'erreur.
- **Le Flux Nominal :** Si l'ID est libre, l'interface demande successivement le nom, le prénom, la profession et le téléphone.
- **Finalisation :** Une fois les données saisies, le module de Validation ordonne au Stockage d'enregistrer le nouveau client, puis confirme le succès à l'Agent.

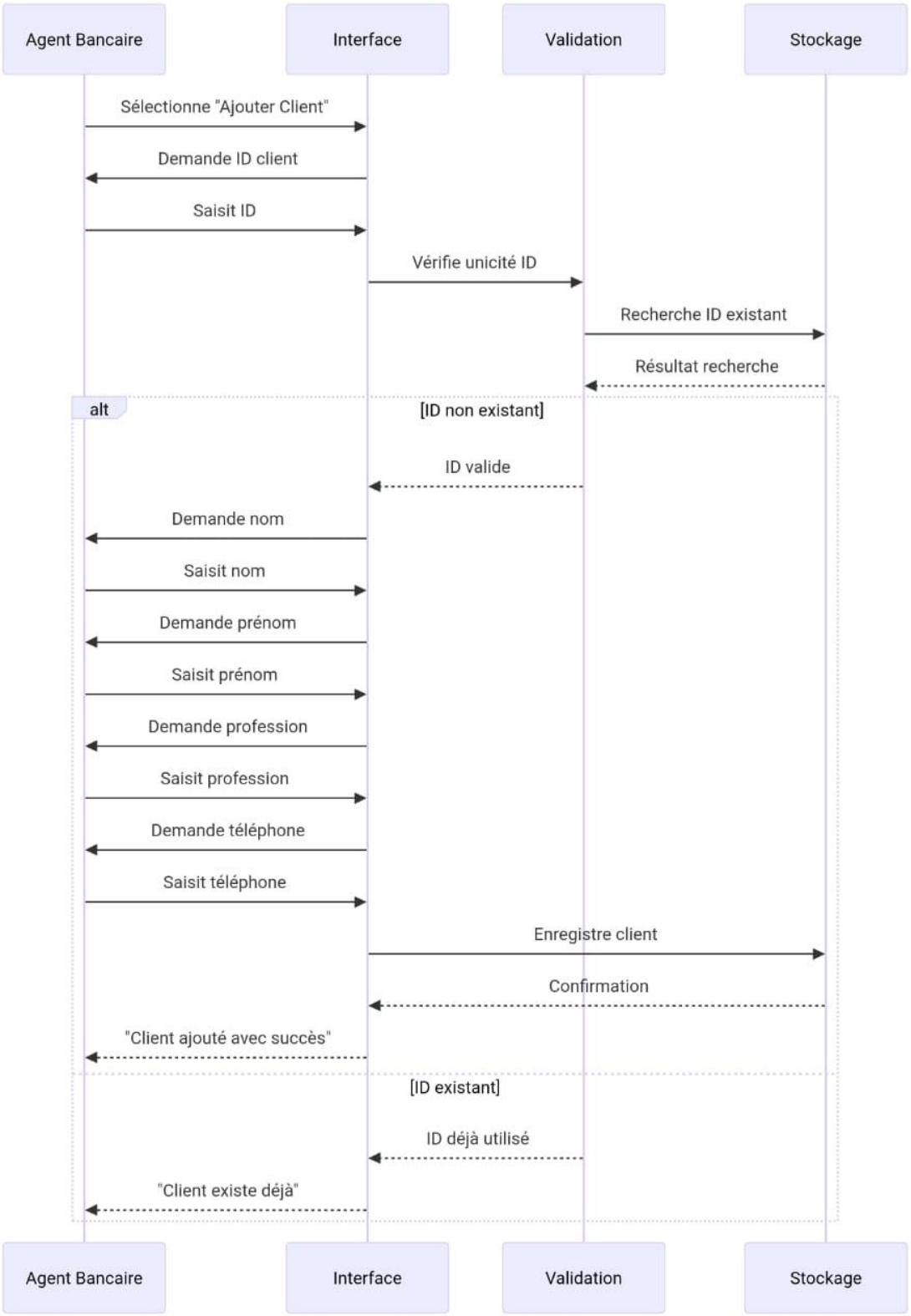

Figure 3: Diagramme de séquence (ajouter Client)

2.4.3 Analyse du Diagramme de Séquence : "Effectuer Retrait"

Ce diagramme illustre la sécurité transactionnelle lors d'un retrait d'argent.

- **Vérification d'Existence :** Le processus débute par la vérification de l'ID du compte. Si le compte n'est pas trouvé dans le Stockage, l'opération s'arrête immédiatement.
- **Contrôle des Règles Métier (Fragment alt) :** C'est ici que sont appliquées les deux contraintes majeures du projet :
    1. **Plafond :** Si le montant est supérieur à **700 DH**, le système renvoie une erreur "Limite dépassée".
    2. **Solvabilité :** Le module de Validation compare le montant au solde actuel récupéré dans le Stockage. Si le solde est insuffisant, l'opération est bloquée.
- **Confirmation et Mise à Jour :** Si toutes les conditions sont validées, le système demande une confirmation finale à l'Agent ("O/N"). Après saisie de "O", le solde est mis à jour dans le Stockage et un message de succès est affiché

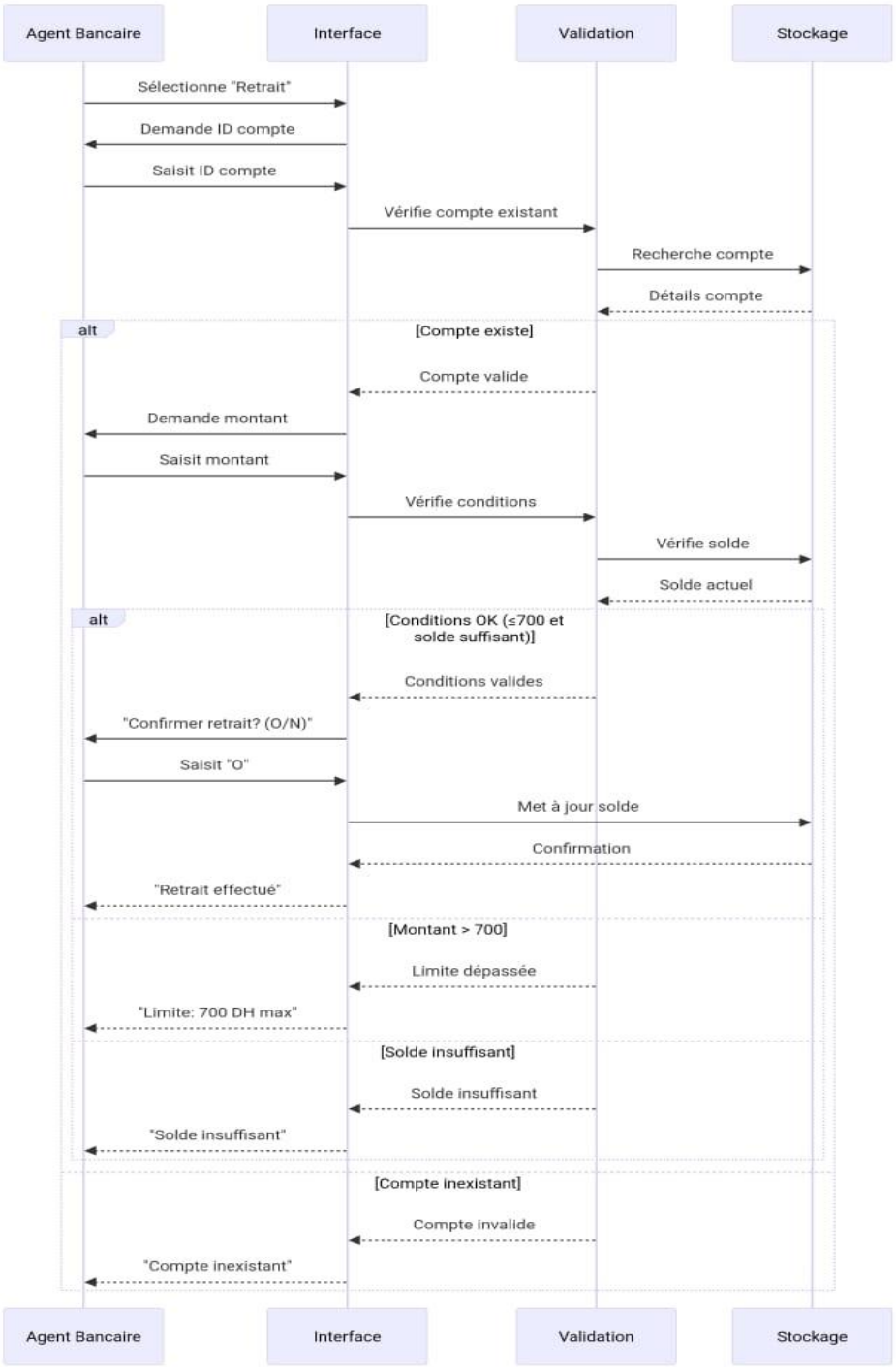

Figure 4: Diagramme se séquence (Retrait)

2.4.4 Conclusion de la Conception

Ces diagrammes montrent que l'architecture de votre programme est **robuste** : aucune donnée n'est écrite dans le stockage sans avoir franchi les étapes de validation et de vérification des règles métiers.

III. Structures de Données (Implémentation Logique)

Le stockage des informations en mémoire vive repose sur deux structures principales (struct) définies comme suit :

Structure "Client"

Cette structure regroupe les informations identitaires.

Champ	Type	Rôle
id_client	int	Clé primaire unique pour identifier le client.
nom	char[50]	Nom de famille.
prenom	char[50]	Prénom.
profession	char[50]	Métier du client.
num_tel	char[15]	Coordonnées téléphoniques.

Figure 5 : Tableau de structure client

2. Structure "Compte"

Cette structure contient les données financières.

Champ	Type	Rôle
id_compte	int	Clé unique du compte.
id_client	int	Clé de liaison vers la structure Client.
solde	float	Montant disponible (Doit être ≥ 0).
date_ouverture	char[11]	Date au format JJ/MM/AAAA.

Figure 6 : Tableau de structure de compte

Les données seront stockées dans des **tableaux de structures** (ex: Client clients[100]), gérés par des compteurs (ex: nb_clients) pour suivre le nombre d'éléments en temps réel.

IV. Algorithmes Principaux (Pseudo-code)

Avant le codage, nous avons défini la logique des fonctions critiques pour respecter les règles de gestion.

Algorithme 1 : Ajout d'un Client (Vérification d'unicité)

Description de la logique

L'ajout d'un client est une opération critique car elle modifie la base de données. La contrainte majeure ici est de garantir l'**unicité de l'identifiant**. Deux clients ne peuvent absolument pas avoir le même ID, sinon le système ne saurait plus à qui appartient un compte bancaire.

L'algorithme suit une approche de "Vérification préalable" :

1. L'utilisateur saisit l'ID souhaité.
2. Le système parcourt tout le tableau des clients existants pour voir si cet ID est déjà utilisé.
3. Si l'ID est trouvé (**Doublon**), l'ajout est bloqué immédiatement.
4. Si l'ID est libre, le système demande le reste des informations (Nom, Prénom, etc.) et enregistre le client à la suite des autres.

```
Début
Fonction AjouterClient(Tableau Clients, Compteur N)
    Ecrire("Entrez l'ID du nouveau client :")
    Lire(id_saisi)

    // Étape 1 : Vérification d'existence (Boucle de recherche)
    Pour i <-- 0 à N-1 pas 1 faire
        Si Clients[i].id == id_saisi ALORS
            Ecrire("Erreur : Cet ID existe déjà !")
            Retourner (Arrêt de la fonction)
        Fin Si
    Fin Pour

    // Étape 2 : Saisie des données (Si l'ID est valide)
    Clients[N].id = id_saisi
    Ecrire("Entrez le Nom :")
    Lire(Clients[N].nom)
    Ecrire("Entrez le Prénom :")
    Lire(Clients[N].prenom)
    // ... (autres champs)

    // Étape 3 : Mise à jour du compteur
    N = N + 1
    Ecrire("Client ajouté avec succès.")
Fin
```

Figure 7 : algorithme (ajouter Client)

Algorithme 2 : Opération de Retrait (Contrôle des plafonds)

Description de la logique

L'algorithme de retrait est conçu comme un "Entonnoir de sécurité". Pour qu'une transaction soit validée, elle doit traverser deux filtres successifs. Si l'un des filtres bloque, l'opération est annulée.

- **Filtre 1 (Règle Métier) :** Le montant ne doit pas dépasser le plafond réglementaire fixé à **700 DH**.
- **Filtre 2 (Règle Comptable) :** Le compte doit disposer d'un solde suffisant (Solvabilité).

C'est seulement si ces deux conditions sont fausses (donc respectées) que le solde est décrémenté.

```
Début
Fonction Retrait(Tableau Comptes, ID_Compte, Montant)
// Étape 1 : Recherche du compte
    TROUVER l'index 'pos' du compte correspondant ID_Compte
    Si Compte non trouvé ALORS
        Ecrire("Compte inexistant")
        Retourner
    Fin Si
     // Étape 2 : Le Double Contrôle
    Si Montant > 700 ALORS
        Ecrire("Erreur : Plafond de 700 DH dépassé.")

    Sinon Si Comptes[pos].solde < Montant alors
        Ecrire("Erreur : Solde insuffisant pour ce retrait.")&

    Sinon
        // Étape 3 : Exécution de la transaction
        Comptes[pos].solde = Comptes[pos].solde - Montant
        Ecrire("Retrait effectué. Nouveau solde : ", Comptes[pos].solde)
    Fin Sinon
    Fin Si
Fin Fonction
```

Figure 8 : algorithme (Retrait)

V. Architecture Modulaire du Programme

Pour assurer la maintenabilité, le programme est découpé en fonctions distinctes, orchestrées par un menu principal.

Organisation des Fonctions

- **Menu Principal :** Aiguille vers les sous-menus A, B, C ou Quitter .
- **Module Clients :** Contient les fonctions ajouterClient, modifierClient, supprimerClient, rechercherClient.
- **Module Comptes :** Contient nouveauCompte, consultationCompte, fermetureCompte.
- **Module Opérations :** Contient retrait et virement.

VI. Gestion de la Persistance des Données (Fichiers)

La mémoire vive (RAM) étant volatile, toutes les données saisies (nouveaux clients, transactions) disparaissent à l'arrêt du programme. Pour pallier ce problème et assurer la continuité du service, nous avons mis en place un système de **persistance sur fichiers textes (.txt)**.

A. Stratégie de Stockage

Nous utilisons trois fichiers distincts pour séparer les responsabilités :

1. **clients.txt** : Pour sauvegarder la liste des clients.
2. **comptes.txt** : Pour sauvegarder l'état des comptes et leurs soldes.
3. **historique.txt** : Pour tracer toutes les opérations effectuées (audit).

Le format choisi est le **CSV (Comma Separated Values)** simplifié, utilisant le point-virgule ; comme séparateur. Ce format est léger et facile à lire par un humain ou par Excel.

VII. Conclusion

Cette phase de conception a permis de figer les structures de données et de valider la logique des algorithmes critiques comme le contrôle du plafond de retrait. Nous disposons désormais d'une feuille de route claire pour passer à l'étape suivante : l'implémentation technique en langage C.

Réalisation

CHAPITRE 3 : Réalisation

3.1. Implémentation en C

3.1.1 Description globale du code

Le code source constitue le moteur logique et interactif de l'application bancaire. Il est structuré de manière modulaire en langage C, utilisant des structures de données relationnelles pour simuler une base de données en mémoire vive (RAM). L'application se distingue par une séparation claire entre la gestion des données, l'interface utilisateur (UX) et la persistance des fichiers.

3.1.2 Organisation des fichiers et structure des données

Bien que le code soit présenté dans un fichier principal pour la compilation, il s'articule autour de deux structures fondamentales liées par une « clé étrangère »:

- **Structure Client** : Gère l'identité civile et la sécurité (ID unique, Nom, Prénom, Profession, Téléphone, et code PIN à 4 chiffres).
- **Structure Compte** : Gère le produit financier (ID compte, lien vers le client via id_client, solde et date d'ouverture).

3.1.3 Explication des fonctions importantes

Le code est divisé en modules fonctionnels spécialisés:

- **Module Interface & Système** :
    - Color() : Pilote l'API Windows pour une interface "Style BIOS" (Bleu/Blanc) et une charte graphique intuitive (Vert pour le succès, Rouge pour l'erreur).
    - menuInteractif() : Utilise getch() (de <conio.h>) pour une navigation fluide via les flèches du clavier, éliminant les erreurs de saisie numérique.
    - clean_stdin() : Fonction de sécurité "pare-feu" qui vide le tampon du clavier pour éviter les bugs lors des saisies successives.
- **Module de Persistance** :
    - sauvegarderDonnees() et chargerDonnees() : Assurent la sérialisation des données dans des fichiers .txt au format CSV (séparateur ;), permettant de retrouver les informations après redémarrage.
    - ajouterHistorique() : Module "Boîte Noire" qui enregistre chaque action critique avec un horodatage précis grâce à la bibliothèque <time.h>.

- **Module Métier (Transactions)** :
    - retrait() : Simule un GAB avec une double vérification systématique du solde suffisant et du code PIN correct.
    - virement() : Gère une transaction atomique sécurisée entre deux comptes avec débit/crédit simultané.

3.1.4 Extraits de code commentés

Ces extraits mettent en lumière les mécanismes de sécurité, d'interface et de gestion de données qui font la particularité de notre solution.

A. **Sécurité et Logique Métier (Le Retrait)** Cet extrait illustre la "Double Vérification" (PIN + Solde) et l'application du plafond de retrait.

```
// Extrait de la fonction retrait()
// Vérification simultanée des trois conditions de sécurité
if(pin == clients[cl].pin && comptes[pos].solde >= mt && mt <= 700) {
    comptes[pos].solde -= mt; // Débit effectif du compte

    // Journalisation automatique dans la "Boîte Noire"
    char msg[100];
    sprintf(msg, "Retrait -%.2f DH sur Compte %d", mt, id);
    ajouterHistorique(msg); // [cite: 83]

    C_SUCCES(); printf("\n  SUCCES. Nouveau solde : %.2f DH", comptes[pos].solde);
} else {
    C_ERREUR(); printf("\n  ECHEC : PIN faux, Solde insuffisant ou plafond > 700 DH.");
}
```

Figure 9 : extrait 1 Sécurité et Logique Métier

B. **Le Moteur de Navigation (UX Interactive)** Cet extrait montre comment nous avons remplacé la saisie de chiffres par une navigation visuelle avec les flèches du clavier.

```
// Extrait du menuInteractif() utilisant <conio.h>
touche = getch(); // Capture instantanée de la touche sans "Entrée" [cite: 61]

if (touche == -32 || touche == 224) { // Détection des touches spéciales (flèches)
    touche = getch();
    if (touche == 72) { // Code ASCII pour la flèche HAUT
        choix--;
        if (choix < 0) choix = nb_opt - 1;
    }
    if (touche == 80) { // Code ASCII pour la flèche BAS
        choix++;
        if (choix >= nb_opt) choix = 0;
    }
} else if (touche == 13) return choix; // Touche ENTREE pour valider
```

Figure 10: extrait 2 Le Moteur de Navigation

C. **Persistance des Données (Sauvegarde CSV)** Nous utilisons un format structuré avec des délimiteurs pour garantir que les données sont conservées après la fermeture du programme.

```c
// Extrait de sauvegarderDonnees()
FILE *fc = fopen("clients.txt", "w"); // Mode "w" pour écraser et mettre à jour
if (fc) {
    for (int i = 0; i < nb_clients; i++) {
        // Sauvegarde au format CSV avec point-virgule (;)
        fprintf(fc, "%d;%s;%s;%s;%s;%d\n",
                clients[i].id_client,
                clients[i].nom,
                clients[i].prenom,
                clients[i].profession,
                clients[i].num_tel,
                clients[i].pin);
    }
    fclose(fc);
}
```

Figure 11: extrait 3 Persistance des Données

D. Sécurité du Tampon (Le Pare-feu) Cette fonction courte est essentielle pour éviter que le programme ne "saute" des étapes à cause de résidus en mémoire.

```c
void clean_stdin() {
    int c;
    // Lit et ignore les caractères restants dans le tampon jusqu'au saut de ligne
    while ((c = getchar()) != '\n' && c != EOF);
}
```

Figure 12 : extrait 4 Sécurité du Tampon

3.2. Environnement de travail

Le choix de l'environnement de développement a été dicté par la nécessité de concilier la portabilité du langage C avec des fonctionnalités d'interface utilisateur spécifiques au système Windows.

3.2.1. Système d'Exploitation et Langage

- **Système d'exploitation : Windows** Ce choix est stratégique car l'application utilise l'API Windows pour la gestion avancée de la console (couleurs, dimensions, titres). Contrairement à Linux ou macOS, Windows offre des fonctions natives comme SetConsoleTextAttribute indispensables à notre charte graphique.

Figure 12 : Windows 10

- **Langage : C (Standard C99/C11)** Le langage C a été retenu pour sa gestion rigoureuse de la mémoire et sa proximité avec le matériel, permettant une exécution rapide des transactions financières et une manipulation précise des structures de données.

Figure 13 : Langage C

3.2.2. Chaîne de Développement (Toolchain)

- **IDE : Code::Blocks ou Dev-C++** Ces environnements de développement intégrés ont été choisis pour leur légèreté et leur excellente intégration du compilateur GCC. Ils facilitent le débogage et la gestion des fichiers sources du projet.

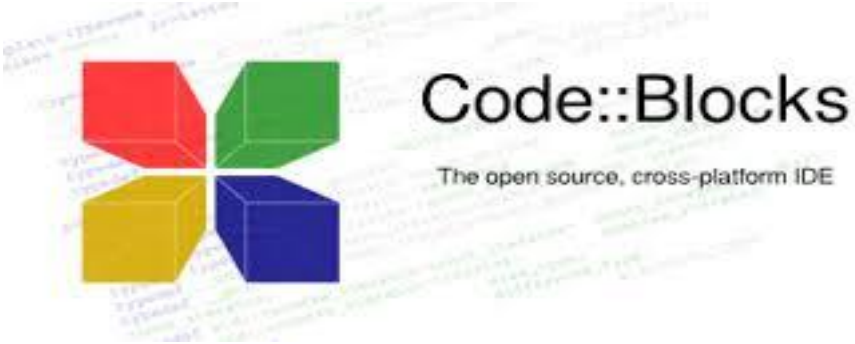

Figure 14 : Code::Blocks

- **Compilateur : GCC (GNU Compiler Collection)** Réputé pour sa fiabilité, GCC assure que le code respecte les standards du langage tout en optimisant la gestion des pointeurs et des accès aux fichiers .txt de notre base de données.

Figure 15: le compilateur GCC

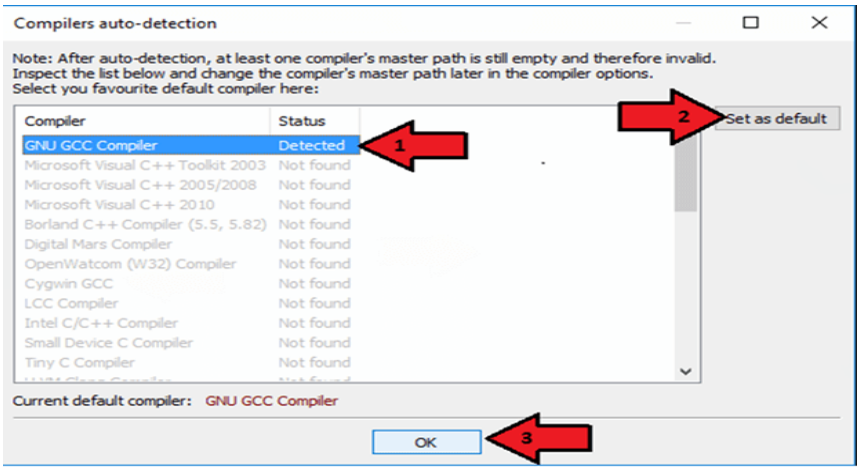

Figure 16 : choisissant le compilateur GCC

3.2.3. Analyse des Bibliothèques Exploitées

Le projet repose sur une approche hybride utilisant le socle standard pour la logique et des bibliothèques systèmes pour l'expérience utilisateur.

- ***<stdio.h> (Standard Input/Output)*** : Constitue la colonne vertébrale du système. Elle gère la persistance des données via les fonctions de manipulation de fichiers (fopen, fprintf, fscanf) pour stocker les comptes et les clients.
- ***<stdlib.h> (Standard Library)*** *:* Utilisée pour les fonctions utilitaires comme system("cls"), permettant de rafraîchir dynamiquement l'affichage pour simuler une interface à pages multiples.
- ***<string.h> (String Handling)*** *:* Cruciale pour sécuriser le traitement des noms, prénoms et professions, évitant ainsi les erreurs de débordement de mémoire lors de la saisie.
- ***<windows.h> (Windows API)*** *:* Cette bibliothèque permet d'aller au-delà de la console classique. Elle pilote le moteur de couleurs (thème Bleu Profond / Blanc Brillant) et modifie l'apparence de la fenêtre pour lui donner un aspect professionnel.
- ***<conio.h> (Console Input/Output)*** *:* C'est le moteur de notre menu interactif. Grâce à la fonction getch(), nous capturons les codes ASCII des touches directionnelles instantanément, sans attendre que l'utilisateur appuie sur "Entrée", ce qui fluidifie la navigation.
- ***<time.h> (Time and Date)*** *:* Intégrée pour automatiser la traçabilité. Elle permet de générer des horodatages précis (Timestamps) pour chaque action enregistrée dans le module de "Boîte Noire" (historique.txt).

3.2.4. Synthèse de l'Environnement

L'utilisation combinée de ces outils a permis de transformer un programme académique en une simulation réaliste d'un terminal bancaire (GAB). Cette infrastructure logicielle garantit une robustesse face aux erreurs de saisie (approche "Zero-Fail") et une intégrité des données digne d'un logiciel professionnel.

3. Résultats :

- ✓ Interface **texte (console)** de gestion bancaire avec un **menu principal** :
  Gestion clients, gestion comptes, opérations bancaires, quitter.
  Navigation au **clavier** (haut/bas, entrée).
  Fond bleu, texte blanc, option sélectionnée en jaune.

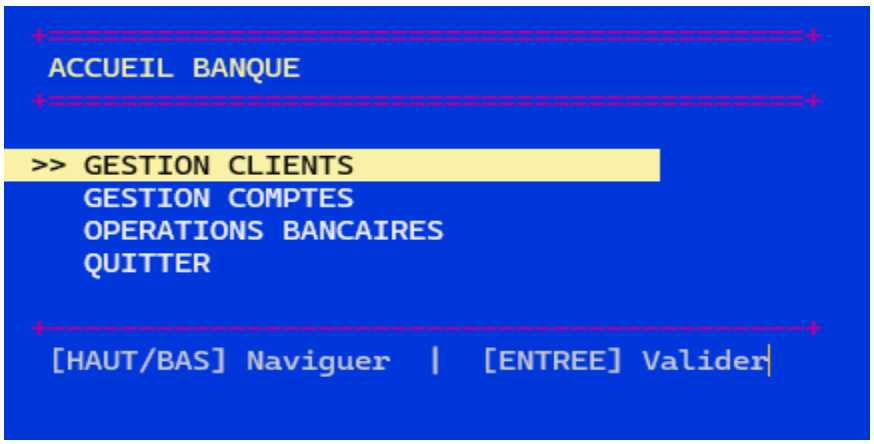

Figure 17 : Interface Menus

- ✓ Interface **texte (console)** de l'**espace clients** permettant :
  ajouter, modifier, supprimer et lister des clients, avec option retour.
  Navigation au clavier, option sélectionnée en surbrillance.

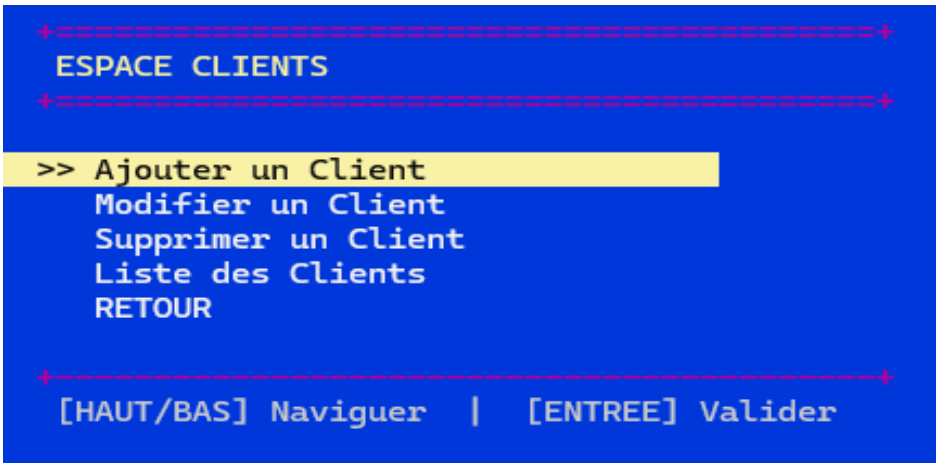

Figure 18 : Espace Clients

✓ Écran **texte (console)** d'**ajout d'un nouveau client** :
saisie de l'ID, nom, prénom, profession, téléphone et code PIN,
avec message de **succès** confirmant l'enregistrement du client.

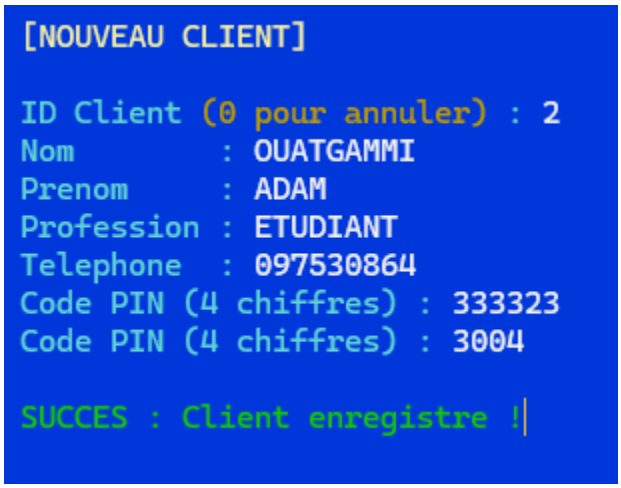

Figure 19 : Ajouter Client

✓ Cette **interface Suppression Client** permettre de faire la suppression d'un client déjà existé dans la base de données est stocké dans le fichier historique

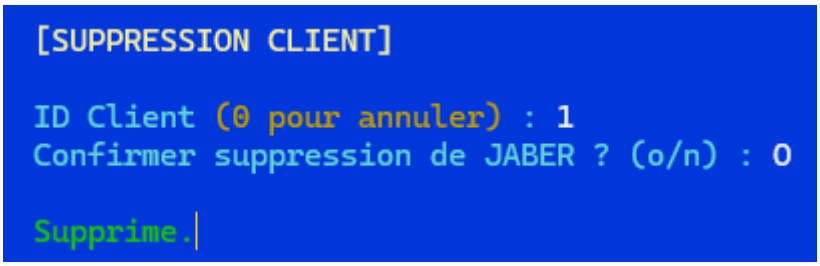

Figure 20 : suppression client

✓ une **interface de menu en ligne de commande (CLI)** au style rétro, avec un fond bleu et du texte en jaune/blanc.

- **Titre :** "ESPACE COMPTES".
- **Options :** Ouvrir un compte (sélectionné), Fermer un compte, Liste des comptes, et RETOUR.
- **Contrôles :** Utilisation des flèches [HAUT/BAS] pour naviguer et [ENTREE] pour valider.

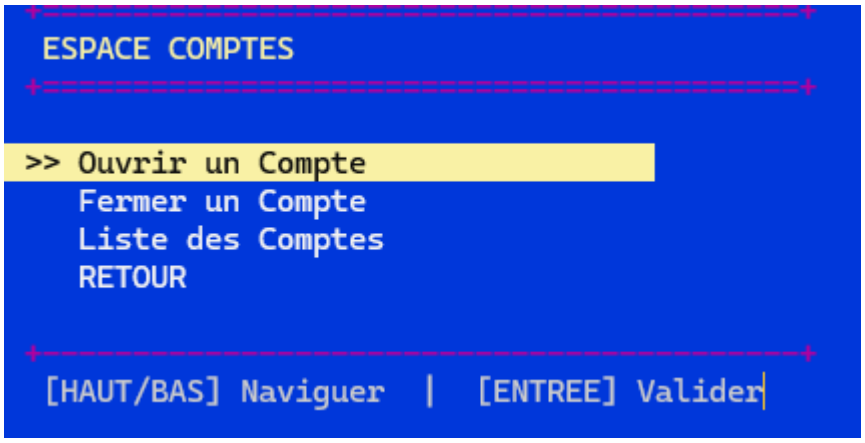

Figure 21 : Espace Comptes

- ✓ interface permet l'**ouverture d'un nouveau compte** bancaire via un formulaire de saisie séquentiel. On y retrouve l'identifiant du compte, celui du propriétaire, ainsi que le montant du solde initial qui exige un minimum de **1000**. Après avoir renseigné la date au format JJ/MM/AAAA, le système valide l'opération par un message de succès en vert indiquant que le **compte est ouvert**.

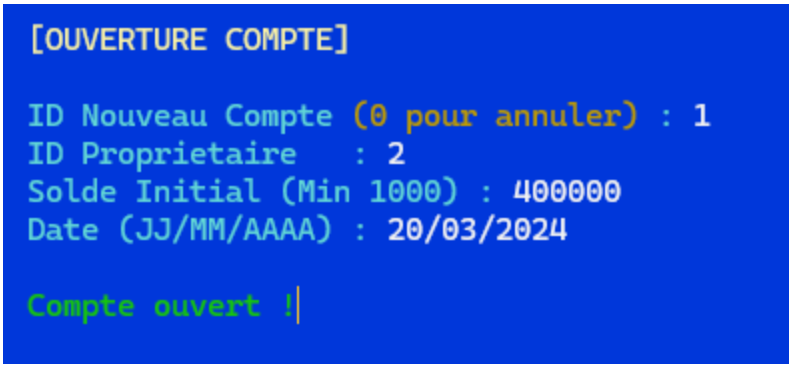

Figure 22 : ouverture Compte

- ✓ Cette interface permet de gérer le cycle de vie des comptes bancaires à travers deux formulaires distincts. Le premier, dédié à l'**ouverture de compte**, recueille l'identifiant du nouveau compte, celui du propriétaire, un solde initial (minimum **1000**) et la date, avant de confirmer l'opération par le message "**Compte ouvert !**". Le second formulaire concerne la **fermeture de compte**, où l'utilisateur saisit l'ID du compte pour afficher son solde actuel, puis confirme la clôture par une commande "**o/n**", ce qui déclenche le message final "**Ferme.**".

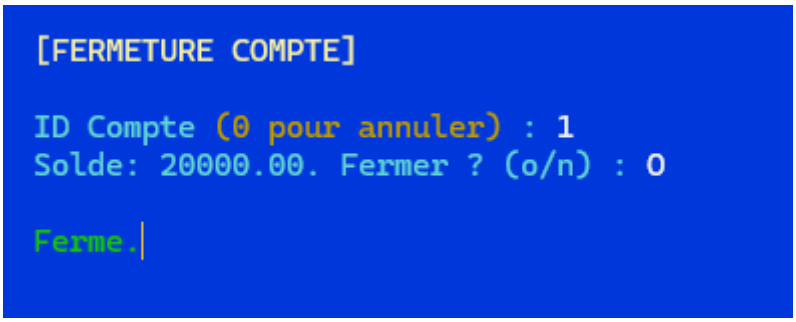

Figure 23 : Fermeture Compte

✓ **l'espace DES TRANSACTIONS :**

- **Options disponibles** : Effectuer un retrait, effectuer un dépôt, faire un virement, consulter l'historique des opérations ou retourner au menu précédent.
- **Commandes** : La navigation se fait avec les touches **[HAUT/BAS]** et la sélection avec la touche **[ENTREE]**.

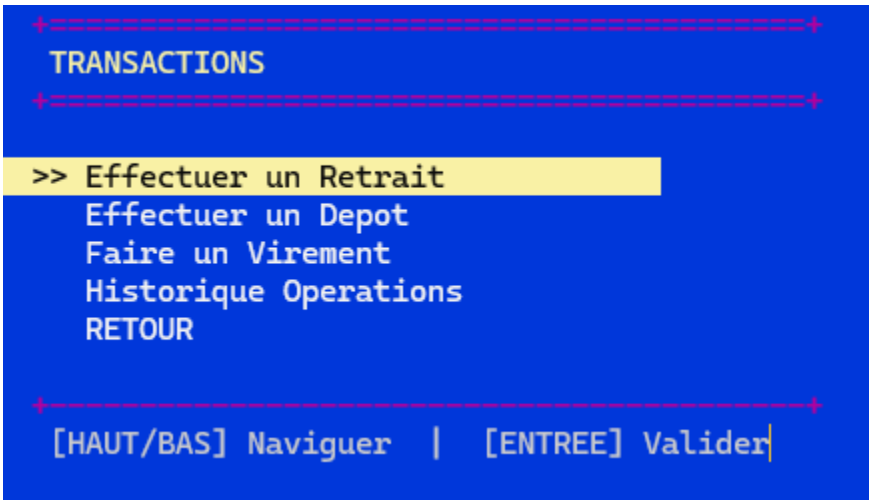

Figure 24 : Espace Des Transactions & opérations

✓ **Erreurs** : Le système affiche des alertes comme "Compte introuvable" en cas de mauvaise saisie lors d'un virement.

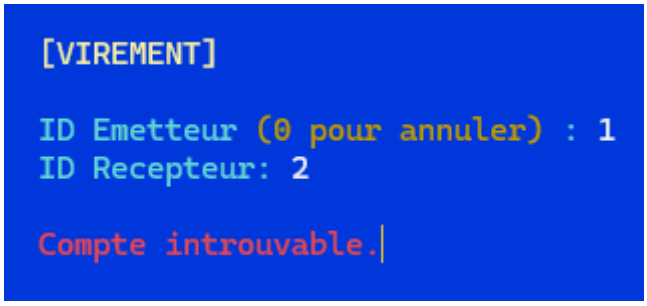

Figure 25 : compte introuvable

✓ **Erreurs** : Affiche des alertes rouges en cas de "**Compte introuvable**" ou d'échec dû au "**PIN ou Solde**".

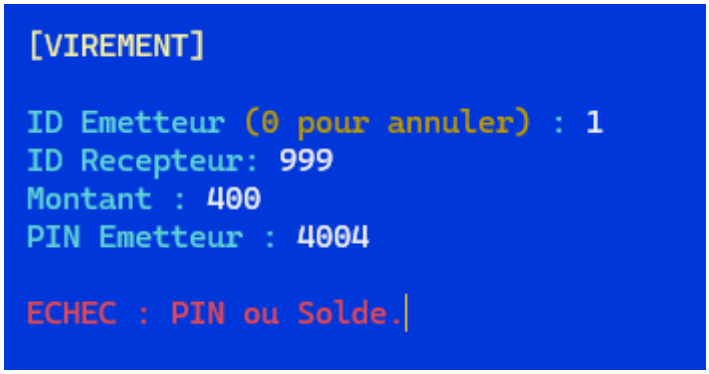

Figure 26 : Erreurs de solide ou code Pin

Opérations de Virement

- **Saisie des données** : L'utilisateur doit renseigner l'ID de l'émetteur, celui du récepteur, le montant et le code PIN de l'émetteur.
- **Sécurité et Validation** :
  - En cas d'erreur sur les identifiants, le message rouge "**Compte introuvable.**" s'affiche.
  - Si le code PIN est incorrect ou le solde insuffisant, l'écran affiche "**ECHEC : PIN ou Solde.**".
  - Une transaction réussie est confirmée par le message vert "**VIREMENT EFFECTUE.**".

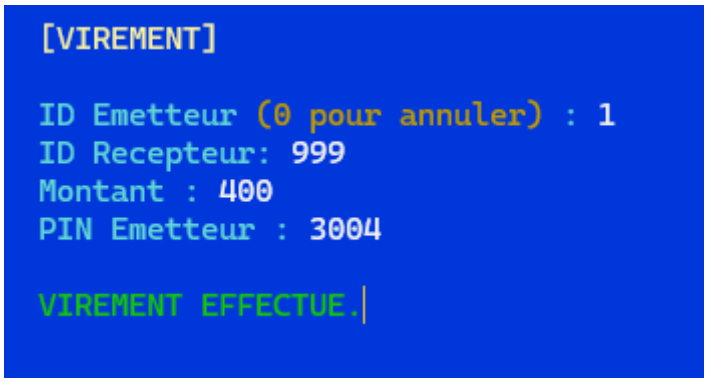

Figure 27 : Virement Effectue

✓ **Opérations de Virement et GAB**

- **Virement** : Demande l'ID de l'émetteur, celui du récepteur, le montant et le code PIN.
- **Retrait GAB** : Permet d'initier un retrait en saisissant l'ID du compte.
- **Sécurité** : Le système affiche des alertes rouges comme "**Compte introuvable**" ou "**ECHEC : PIN ou Solde**" en cas d'erreur ou de fonds insuffisants.

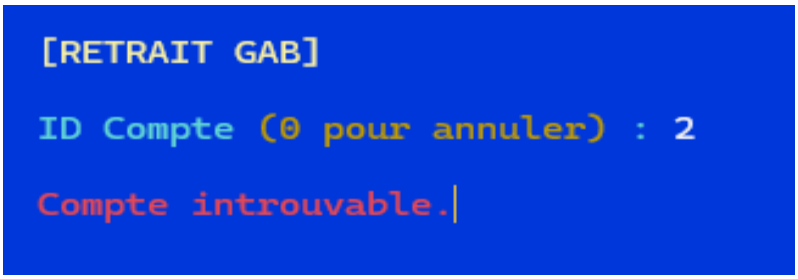

Figure 28 : Compte introuvable GAB

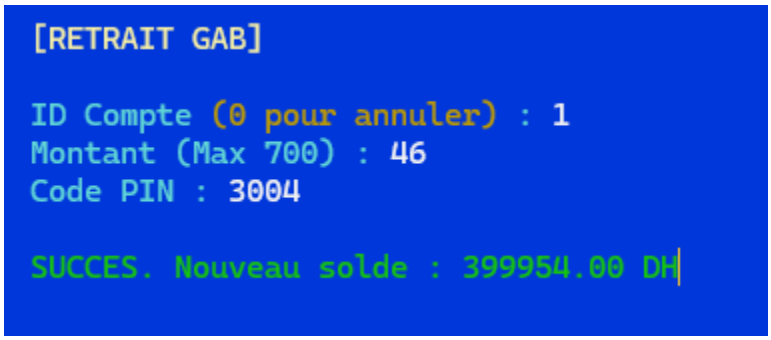

Figure 29 : Retrait GAB SUCCES

✓ Ce formulaire permet d'ajouter des fonds sur un compte existant en saisissant son **ID Compte** et le **montant à déposer**. Une fois l'opération validée, le système affiche un message de succès en vert indiquant le **nouveau solde** mis à jour en DH. Comme pour les autres écrans, il est possible de taper **0** à tout moment pour annuler la saisie.

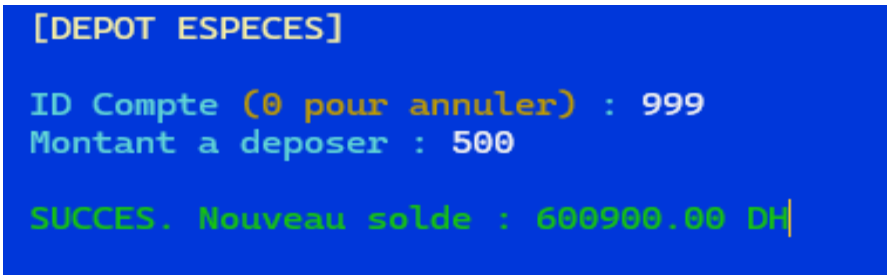

Figure 30 : Depot Especes

✓ **Modifier Client** Cette interface permet de mettre à jour les informations personnelles d'un client enregistré. L'utilisateur navigue dans un menu dédié pour choisir l'élément spécifique à changer :

- **Options de modification** : Il est possible de modifier le **NOM**, le **PRENOM**, le **TELEPHONE** ou le **PIN**.
- **Processus** : Une fois l'élément choisi (par exemple le nom), l'utilisateur saisit la nouvelle valeur.
- **Validation** : Le système confirme le succès de l'opération par le message vert "**Modification enregistree !**".

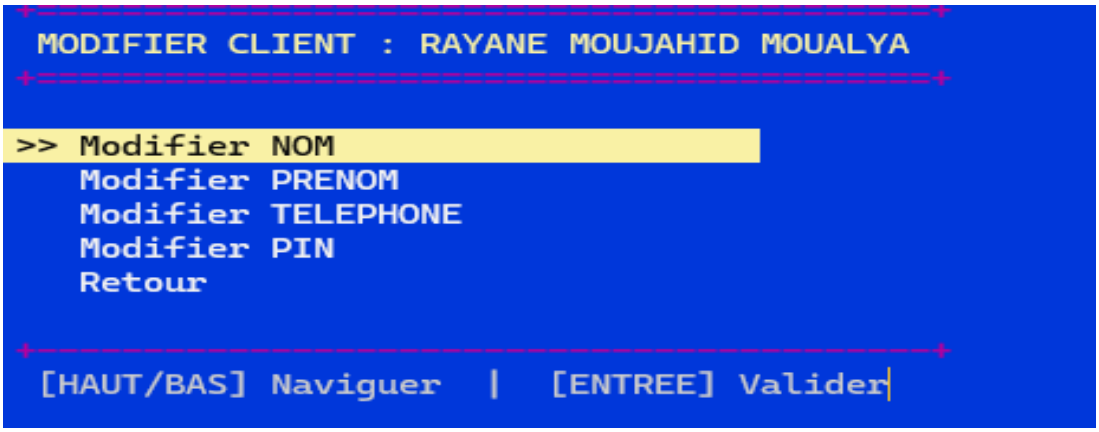

Figure 31 : Modifier Client

✓ **Par exemple :**

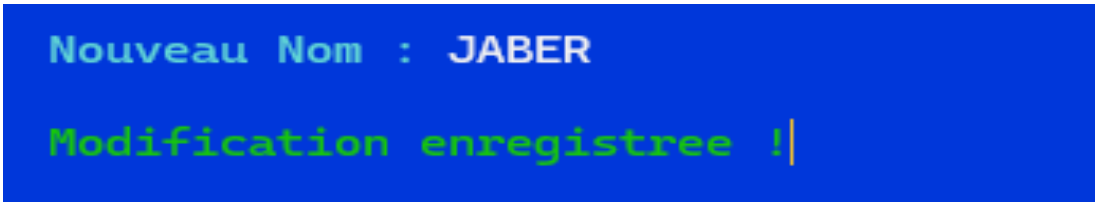

Figure 32 : exemple de Modification

✓ **Liste des Historiques des transactions et Clients/Comptes :**

```
HISTORIQUE DES OPERATIONS
+-----------------------------------------------------------------------+
| DATE & HEURE       | DETAILS DE L'OPERATION                           |
+-----------------------------------------------------------------------+
| [28/01/2026 20:29] Nouveau Client ID: 1 (RAYANE MOUJAHID MOUALYA)     |
| [28/01/2026 20:31] Nouveau Client ID: 2 (OUATGAMMI ADAM)              |
| [28/01/2026 20:34] Ouverture Compte ID: 1 pour Client 2               |
| [28/01/2026 20:35] Fermeture Compte ID: 1                             |
| [28/01/2026 20:35] Ouverture Compte ID: 1 pour Client 2               |
| [28/01/2026 20:36] Ouverture Compte ID: 999 pour Client 1             |
| [28/01/2026 20:43] Retrait -46.00 DH sur Compte 1                     |
| [28/01/2026 20:46] Virement 400.00 DH de 1 vers 999                   |
| [28/01/2026 20:47] Depot +500.00 DH sur Compte 999                    |
+-----------------------------------------------------------------------+

Appuyez sur une touche pour revenir...
```

Figure 33 : Historique des operations

```
LISTE DES CLIENTS (2)
+------+-----------------------+--------------------+-------------+
| ID   | NOM & PRENOM          | PROFESSION         | TELEPHONE   |
+------+-----------------------+--------------------+-------------+
| 1    | RAYANE MOUJAHID MOUALYA | ETUDIANT         | 0708097643  |
| 2    | OUATGAMMI ADAM        | ETUDIANT           | 097530864   |
+------+-----------------------+--------------------+-------------+

Appuyez sur une touche...
```

Figure 34 : Liste des clients

```
ETAT DES COMPTES (1)
+--------+-------------+--------------+-----------------------+
| COMPTE | DATE OUV.   | SOLDE (DH)   | PROPRIETAIRE          |
+--------+-------------+--------------+-----------------------+
| 1      | 20/03/2024  | 399554.00    | OUATGAMMI ADAM        |
+--------+-------------+--------------+-----------------------+

Appuyez sur une touche...
```

Figure 35 : état des comptes

✓ **Ecran qui s'affiche Au revoir après la sortie de votre système banque**

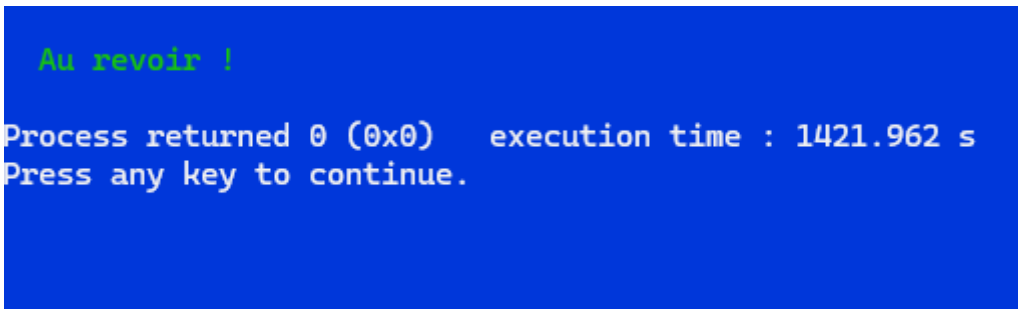

Figure 36 : fermer l'application

L'application assure la persistance des données en enregistrant chaque action dans des fichiers texte, ce qui permet de conserver une trace permanente de l'activité du système.

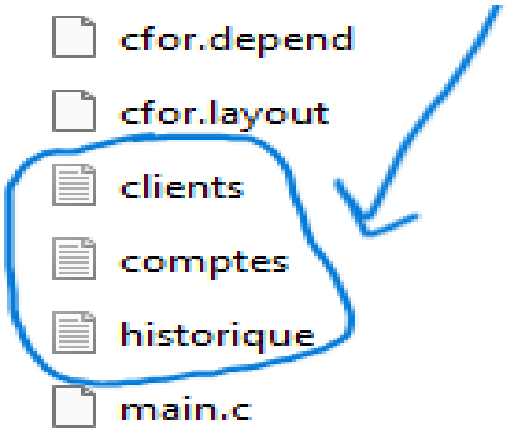

Figure 37 : Les fichiers

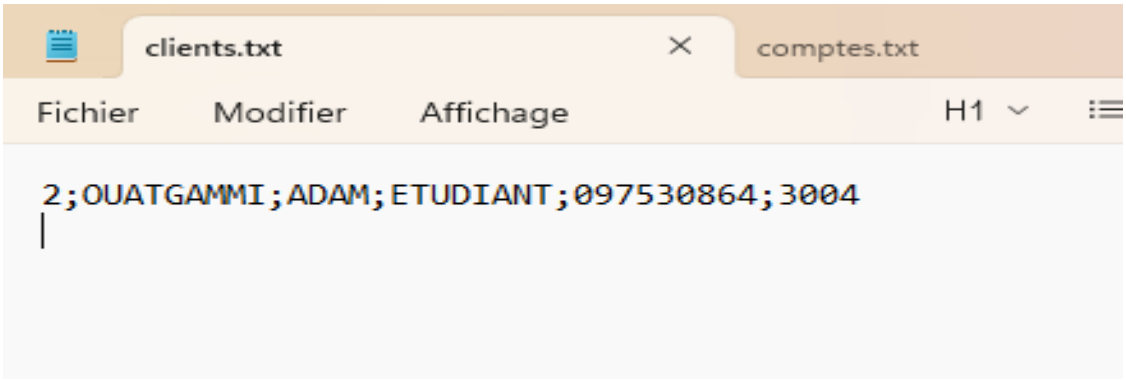

Figure 38 : fichier Clients

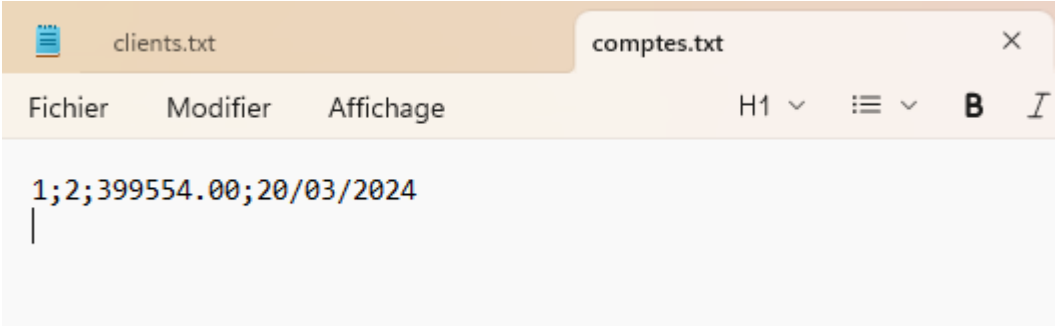

Figure 39 : fichier comptes

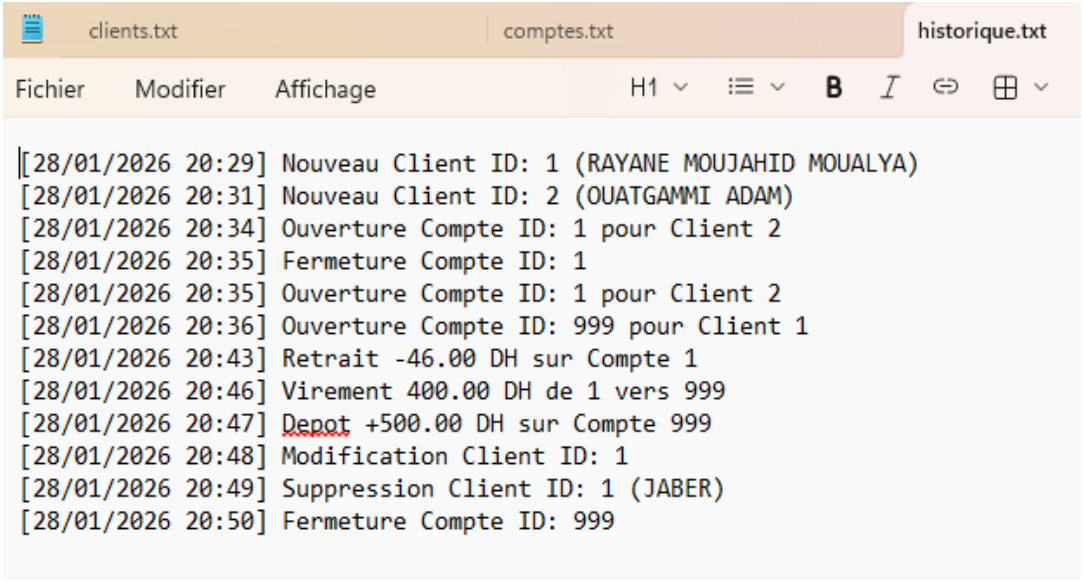

Figure 40 : fichier historique

CONCLUSION GÉNÉRALE

La réalisation de ce projet de Système de Gestion Bancaire a marqué une étape déterminante dans notre apprentissage de la programmation en langage C. Ce travail nous a permis de transformer des concepts théoriques abstraits en une solution logicielle concrète, fonctionnelle et sécurisée. Le bilan de cette expérience est extrêmement positif : nous avons réussi à concevoir une application qui dépasse le cadre de l'exercice académique pour se rapprocher d'un outil professionnel, capable de gérer intégralement le cycle de vie d'un client et de ses comptes, tout en assurant une traçabilité parfaite des opérations financières grâce à un journal d'historique automatisé.

Tous les objectifs initiaux ont été atteints avec succès. Sur le plan ergonomique, nous avons développé une interface innovante en abandonnant la console classique au profit d'un menu interactif et coloré, de style BIOS, piloté par les flèches du clavier. La sécurité des données a été placée au cœur du système par l'implémentation de codes PIN et de plafonds de retrait stricts. De plus, la robustesse du programme est assurée par une gestion rigoureuse du tampon mémoire, protégeant l'application contre les saisies incorrectes, tandis que l'utilisation de fichiers texte garantit la persistance des données à long terme.

Le développement a toutefois présenté des défis techniques notables, notamment la gestion des flux d'entrée et de sortie. La synchronisation entre la mémoire vive et le stockage sur disque a nécessité une attention particulière pour éviter toute perte d'information. De même, l'optimisation de l'ergonomie en console via les bibliothèques spécialisées a demandé de nombreux tests pour capturer avec précision les codes ASCII des touches directionnelles. Ces obstacles ont été des opportunités d'approfondir notre compréhension de l'architecture logicielle et de la manipulation des périphériques.

Pour l'avenir, plusieurs pistes d'évolution sont envisageables pour une version 2.0. L'ajout d'un algorithme de hachage permettrait de crypter les codes PIN, renforçant ainsi la confidentialité des fichiers de stockage. Le passage à l'allocation dynamique via des listes chaînées permettrait également de supprimer les limites de capacité et d'optimiser l'usage de la mémoire. Enfin, la migration vers une interface graphique fenêtrée (GUI) offrirait une expérience utilisateur encore plus moderne. Ce projet témoigne de notre capacité à concevoir une architecture cohérente et à placer l'utilisateur au centre de nos préoccupations techniques.