

Rapport du Projet Final

Programmation Réseau – Système Bancaire

Client-Serveur Sécurisé

KINYANTA WA KINYANTA MARIA
Université Nouveaux Horizons

10 mars 2025

Table des matières

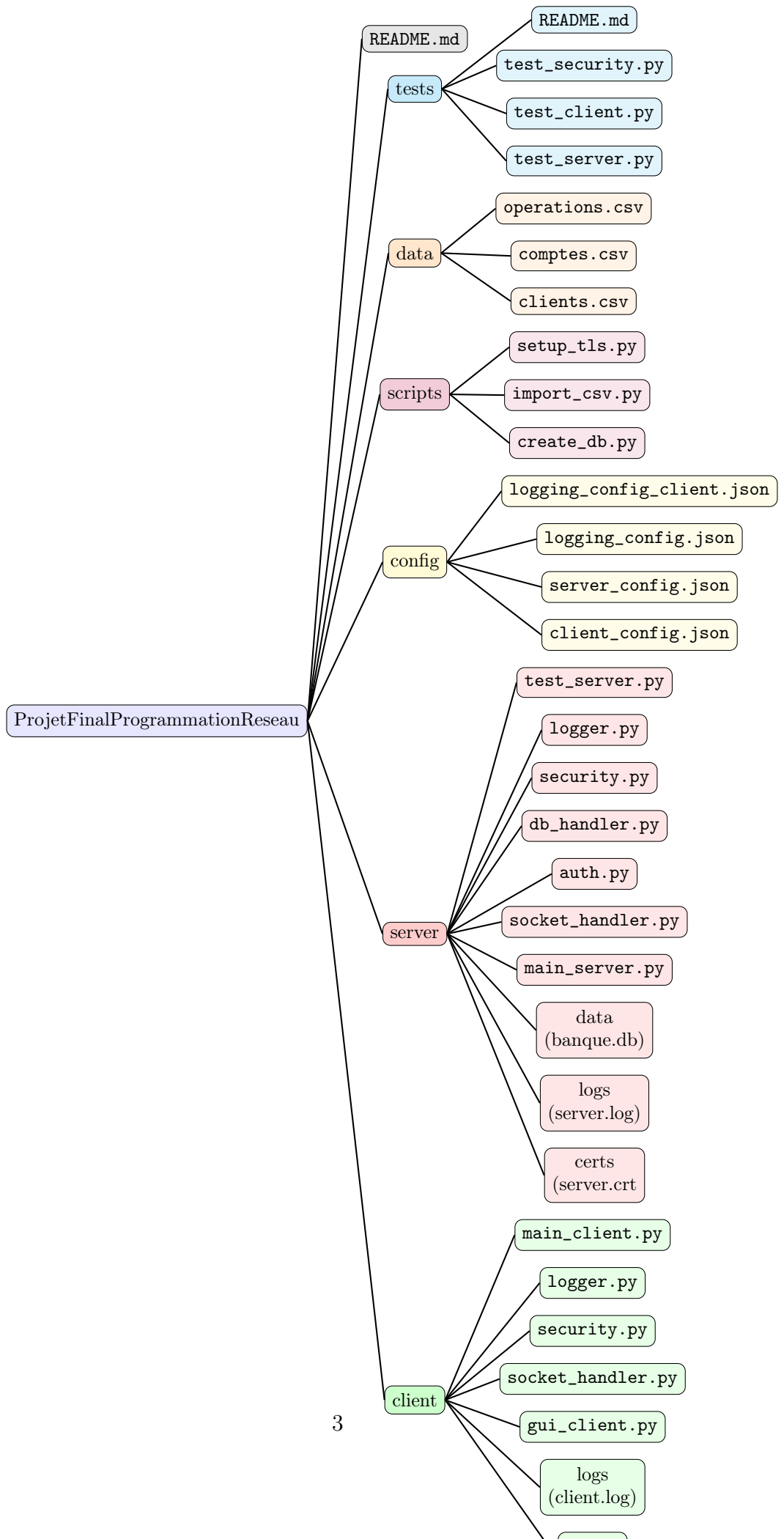
1	Introduction	2
2	Structure du Projet	2
2.1	Détail de la structure du dossier	4
3	Architecture Modulaire et 3 Tiers	6
3.1	Architecture Modulaire	6
3.2	Architecture 3 Tiers	7
4	Captures d'écran des interfaces	8
4.1	Écran de connexion	9
4.2	Création de compte	10
4.3	Accueil	11
4.4	Dépôt	11
4.5	Retrait	12
4.6	Transfert	13
4.7	Historique	15
4.8	Retour à l'accueil	17
5	Analyse Wireshark	17
5.1	Préparation de l'analyse	18
5.2	Captures d'écran et interprétation	18
5.3	Conclusion de l'analyse	18
6	Lien github	20

1 Introduction

Ce rapport présente le projet final réalisé dans le cadre du cours de **Programmation Réseau**. Le but de ce projet était de développer un système bancaire qui simule la communication entre des distributeurs automatiques (clients) et un serveur central. La communication s'effectue via TCP sécurisée par TLS.

2 Structure du Projet

La structure du projet est organisée de la manière suivante :



2.1 Détail de la structure du dossier

Le projet `ProjetFinalProgrammationReseau` est organisé de manière à assurer une séparation des responsabilités. La structure se répartit en plusieurs dossiers principaux, chacun a un rôle spécifique dans l'application.

client/

Ce dossier contient tous les fichiers de la partie client, c'est-à-dire le simulateur de distributeur automatique :

- `certs/` : Contient les certificats TLS du client (`ca.crt`, `client.crt`, `client.key`).
- `logs/` : Fichiers de logs générés par le client (`client.log`).
- `gui_client.py` : Interface graphique développée avec Tkinter, permettant l'interaction utilisateur.
- `socket_handler.py` : Module de communication avec le serveur, gérant l'envoi et la réception des requêtes.
- `security.py` : Configure le contexte TLS pour sécuriser les connexions du client.
- `logger.py` : Module de configuration des logs côté client.
- `main_client.py` : Fichier de test unitaire pour vérifier le bon fonctionnement des requêtes.

server/

Ce dossier regroupe le code du serveur qui gère les transactions bancaires :

- `certs/` : Contient les certificats TLS du serveur (`server.crt`, `server.key`).
- `logs/` : Fichiers de logs générés par le serveur (`server.log`).
- `data/` : Contient la base de données SQLite3 (`banque.db`) utilisée pour stocker les informations.
- `main_server.py` : Point d'entrée du serveur, qui gère l'acceptation des connexions.
- `socket_handler.py` : Module qui traite les requêtes des clients et lui renvoie les réponses.
- `auth.py` : Gère l'authentification, le suivi des tentatives de connexion et la gestion des sessions.
- `db_handler.py` : Interagit avec la base de données pour récupérer et mettre à jour les informations des comptes et des opérations.
- `security.py` : Assure la sécurisation des communications côté serveur avec TLS.
- `logger.py` : Module de configuration des logs côté serveur.
- `test_server.py` : Tests unitaires pour vérifier les fonctionnalités du serveur.

config/

Ce dossier contient l'ensemble des fichiers de configuration du projet :

- `client_config.json` : Paramètres de connexion pour le client.
- `server_config.json` : Paramètres de connexion pour le serveur.
- `logging_config.json` : Configuration des logs côté serveur.
- `logging_config_client.json` : Configuration des logs côté client.

scripts/

Ce dossier regroupe les scripts permettant de préparer l'environnement du projet :

- `create_db.py` : Crée la base de données et initialise les tables.
- `import_csv.py` : Importe les données des fichiers CSV dans la base de données.
- `setup_tls.py` : Génère les certificats TLS pour sécuriser les communications.

data/

Ce dossier contient les données utilisées par le projet :

- `clients.csv` : Liste des clients.
- `comptes.csv` : Informations sur les comptes bancaires.
- `operations.csv` : Historique des transactions.

tests/

Ce dossier contient les tests unitaires et d'intégration du projet :

- `test_server.py` : Tests unitaires pour le côté serveur.
- `test_client.py` : Tests unitaires pour le côté client.
- `test_security.py` : Tests pour la configuration TLS.
- `README.md` : Documentation spécifique aux tests.

README.md

Le fichier principal de documentation du projet.

3 Architecture Modulaire et 3 Tiers

3.1 Architecture Modulaire

Dans notre projet, chaque module a un rôle, c'est pour faciliter la maintenance et l'extension du code en séparant les responsabilités. La figure ?? ci-dessous montre les principaux blocs (Client, Serveur, Configuration/Scripts et test) ainsi que les modules qui les composent.

— **Client :**

- `gui_client.py` gère l'interface graphique.
- `socket_handler.py` s'occupe de la communication réseau avec le serveur.
- `security.py` assure la sécurisation TLS côté client.
- `logger.py` configure et gère les logs du client.
- `main_client.py` permet de tester les requêtes envoyées au serveur.

— **Serveur :**

- `main_server.py` lance le serveur et gère l'écoute des connexions.
- `socket_handler.py` reçoit les requêtes des clients et y répond.
- `auth.py` gère l'authentification et les sessions.
- `db_handler.py` interagit avec la base de données (SQLite).
- `security.py` s'occupe de la sécurisation TLS côté serveur.
- `logger.py` configure les logs côté serveur.
- `test_server.py` contient les tests unitaires du serveur.

— **Configuration & Scripts :**

- Les fichiers `client_config.json`, `server_config.json` et `logging_config*.json` définissent les paramètres de connexion et de logs.
- `create_db.py` et `import_csv.py` permettent de créer la base de données et d'importer les données (clients, comptes, opérations).
- `setup_tls.py` génère les certificats TLS pour sécuriser la communication.

La figure ci-dessous montre comment ces modules interagissent :

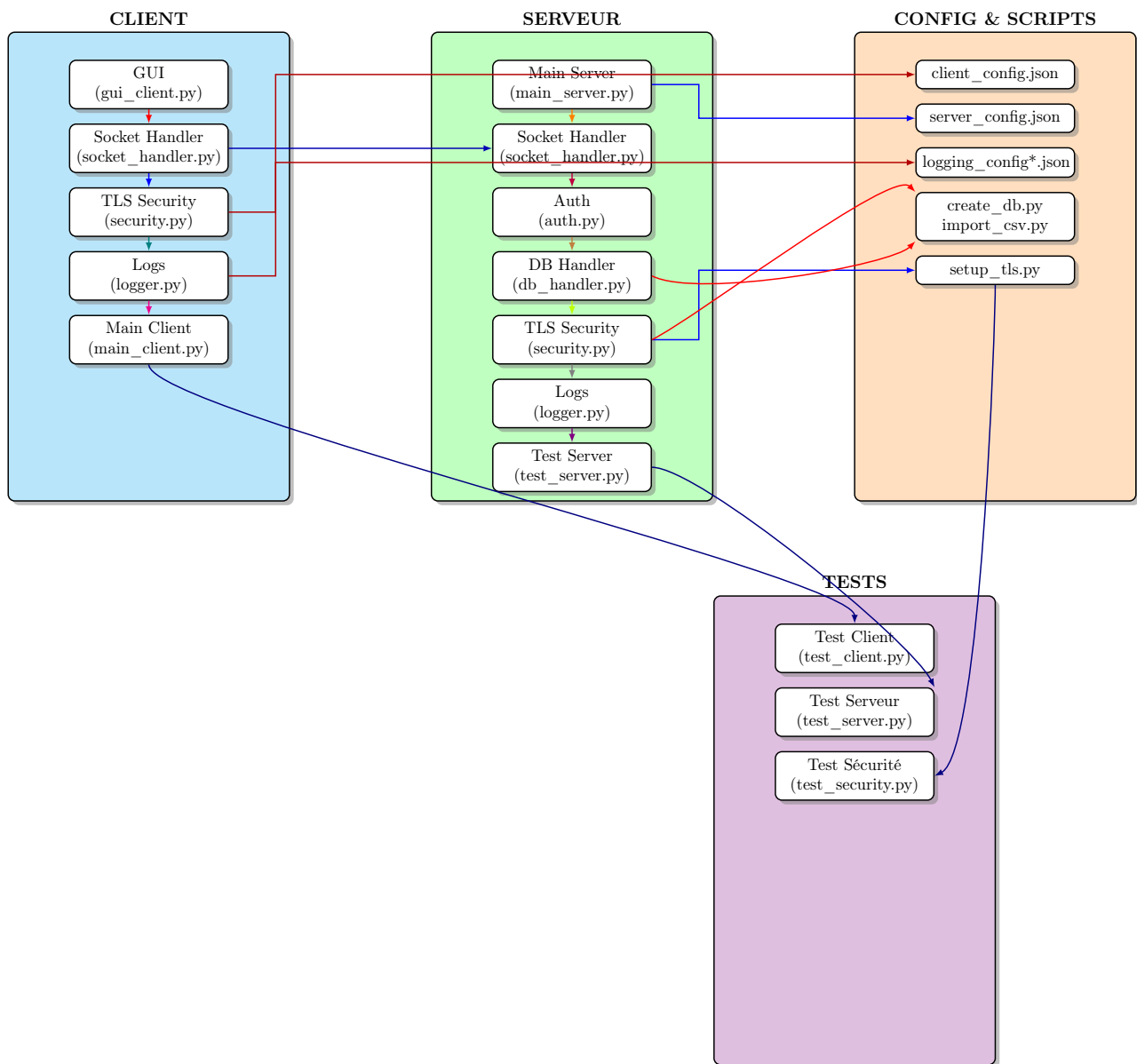


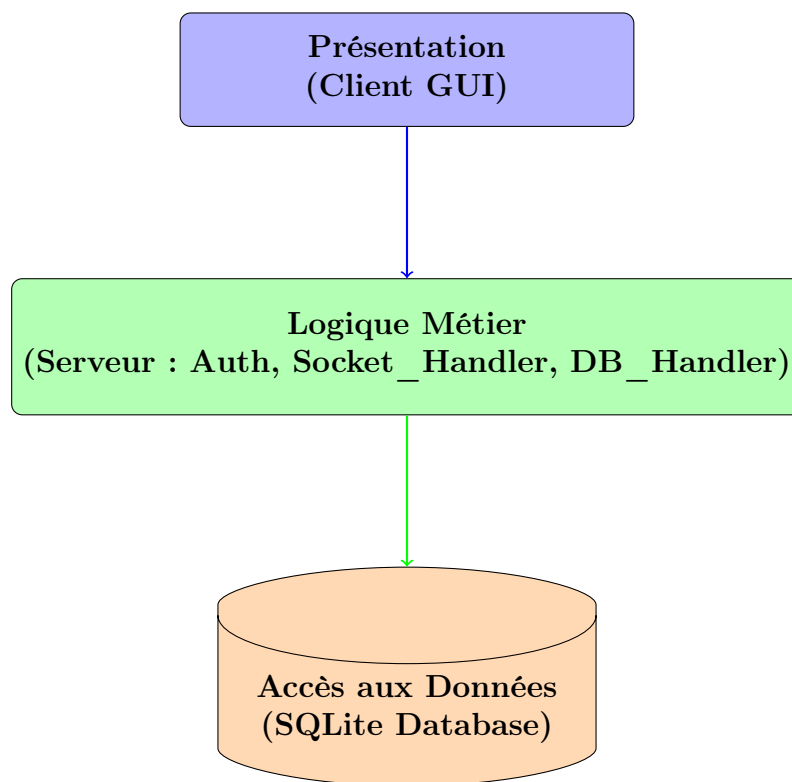
FIGURE 2 – Diagramme de l'architecture modulaire du système

3.2 Architecture 3 Tiers

Le système est structuré selon une architecture **3 tiers**, permettant une séparation claire des responsabilités entre la présentation, la logique métier et l'accès aux données. Cette organisation améliore la modularité, la maintenabilité et la scalabilité du projet. La figure ?? illustre cette architecture.

1. **Présentation (Client) :** Cette couche représente l'interface utilisateur et la gestion des interactions avec l'utilisateur. Elle inclut :
 - L'**interface graphique** via `gui_client.py`, permettant à l'utilisateur d'effectuer des actions comme l'authentification ou l'envoi de requêtes.
 - La **communication réseau** via `socket_handler.py`, qui établit la connexion avec le serveur.
 - La **sécurisation des échanges** via `security.py`, qui utilise TLS pour chiffrer les données transmises.

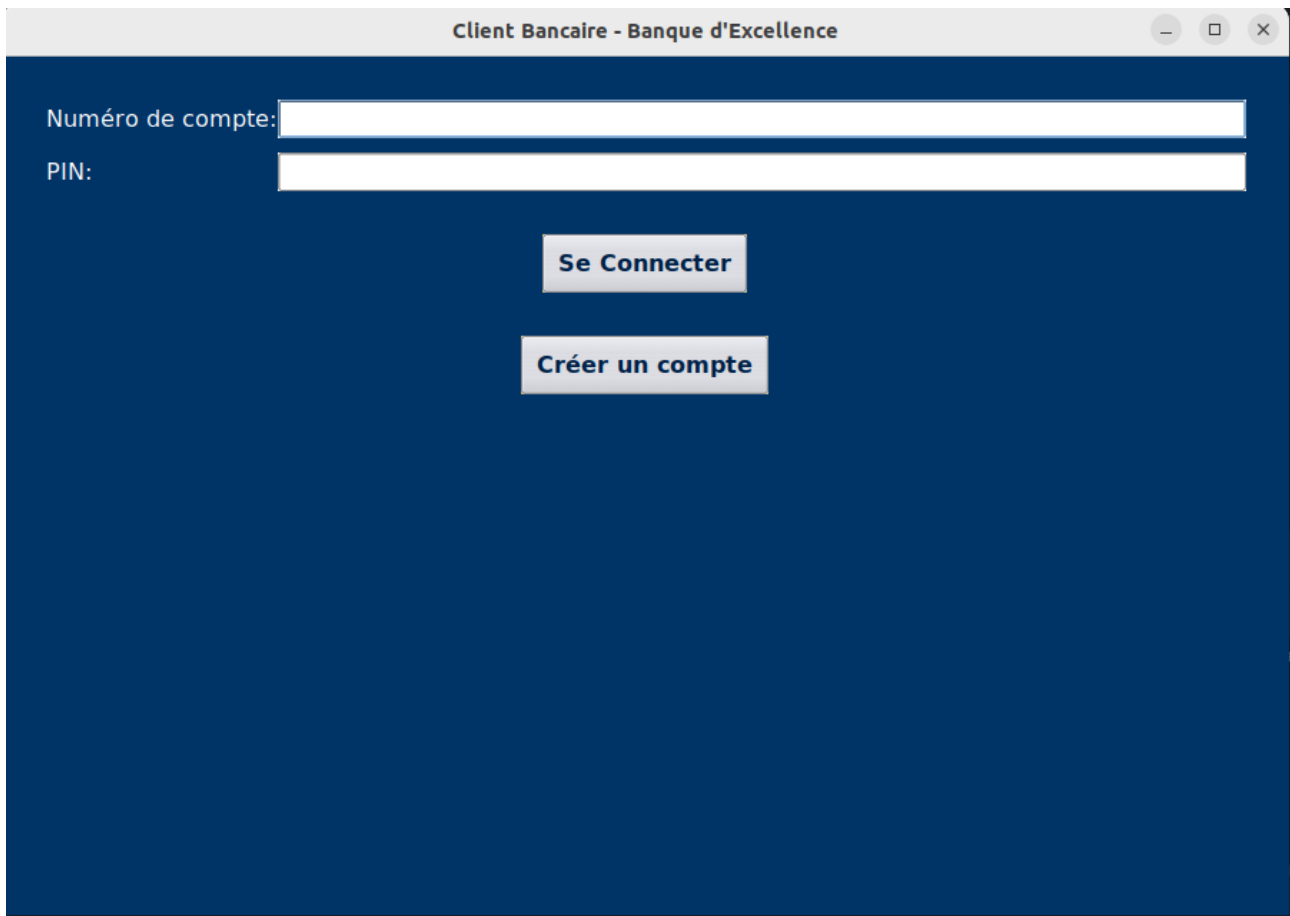
2. **Logique Métier (Serveur)** : Cette couche gère le traitement des requêtes et applique les règles métier du projet.
 - **Gestion des requêtes** et interactions avec le client avec `socket_handler.py`.
 - **Authentification et gestion des sessions** avec `auth.py`, permettant de vérifier l'identité des utilisateurs.
 - **Traitement des transactions et interactions avec la base de données**, fait par `db_handler.py`.
3. **Accès aux Données (Base de Données)** : Cette couche permet de stocker et gérer les informations nécessaires au fonctionnement du système.
 - La base de données SQLite3 (`banque.db`) est utilisée pour enregistrer les utilisateurs, les transactions et les logs.
 - `db_handler.py` assure l'interaction entre la logique métier et la base de données, en effectuant des requêtes SQL pour lire et écrire des informations.



4 Captures d'écran des interfaces

Dans cette section, nous présentons quelques écrans de l'interface graphique du client, avec une brève description de chaque image.

4.1 Écran de connexion



Client Bancaire - Banque d'Excellence

Numéro de compte:

PIN:

Se Connecter

Créer un compte

FIGURE 3 – Écran de connexion

Écran de connexion , ici l'utilisateur saisit son numéro de compte et son PIN. Lorsque l'utilisateur n'a pas de compte, il peut cliquer sur le bouton pour en créer un compte qui ramene l'image suivante.

4.2 Création de compte

Client Bancaire - Banque d'Excellence

Inscription

PIN:

Nom:

Prénom:

Adresse:

Code Postal:

Ville:

Téléphone Fixe:

Téléphone Portable:

Type de Compte:

[Créer le compte](#)

[Retour au Login](#)

FIGURE 4 – Formulaire d'inscription

Ici, l'utilisateur renseigne ses informations (nom, numéro de compte, etc.) pour créer un nouveau compte.

4.3 Accueil



FIGURE 5 – Accueil

L'écran d'accueil affiche plusieurs onglets pour les différentes opérations bancaires : dépôt, retrait, transfert, etc.

4.4 Dépôt

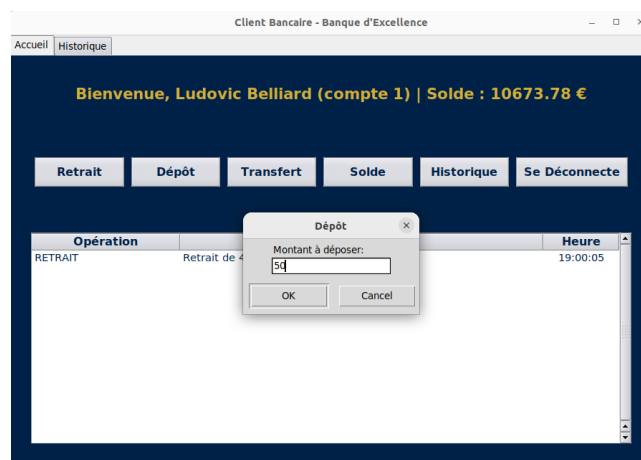


FIGURE 6 – Saisie du montant à déposer

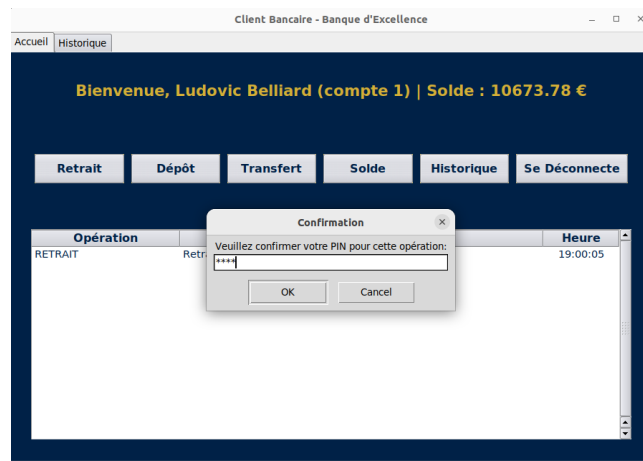


FIGURE 7 – Confirmation du PIN



FIGURE 8 – Validation du dépôt

On indique d'abord le montant, puis on confirme avec le PIN, et enfin un message confirme le succès du dépôt.

4.5 Retrait

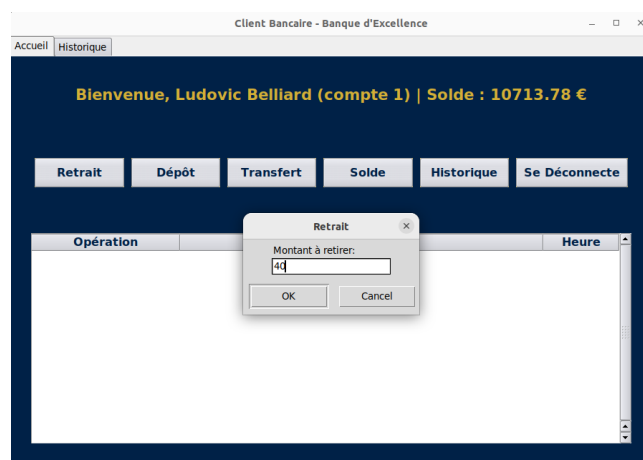


FIGURE 9 – Saisie du montant à retirer



FIGURE 10 – Confirmation du PIN pour le retrait

Le fonctionnement est similaire au dépôt : on saisit le montant, puis on valide avec le PIN.

4.6 Transfert



FIGURE 11 – Saisie du compte destinataire

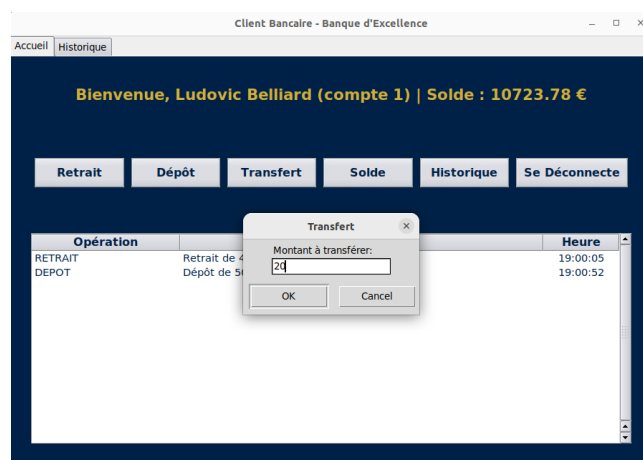


FIGURE 12 – Indication du montant à transférer

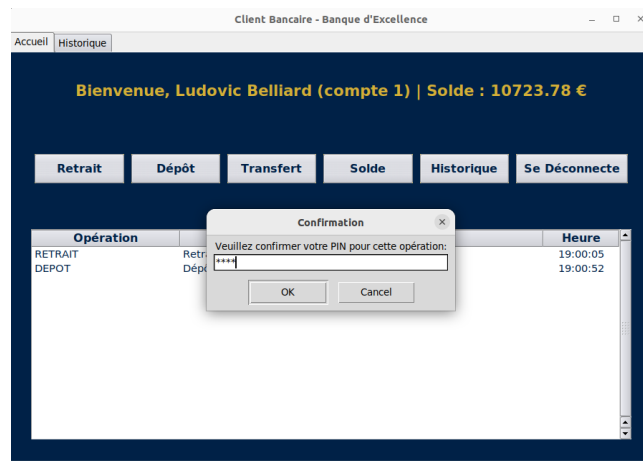


FIGURE 13 – Confirmation du PIN pour le transfert



FIGURE 14 – Validation ou annulation du transfert

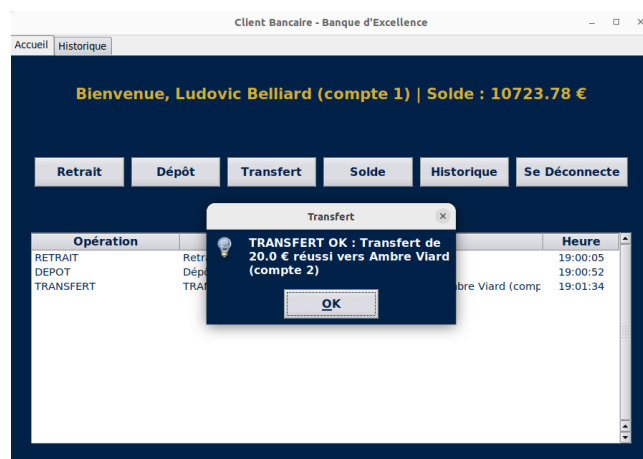


FIGURE 15 – Transfert validé

Pour un transfert, on saisit d'abord le compte destinataire, puis le montant, et on valide avec le PIN. L'utilisateur peut encore annuler à la fin, sinon un message indique le succès de l'opération.

4.7 Historique

Client Bancaire - Banque d'Excellence

Accueil

Historique

Date	Libellé	Montant
2025-03-08 19:01:34	TRANSFERT SORTANT	-20.0
2025-03-08 19:00:52	DEPOT	50.0
2025-03-08 19:00:04	RETRAIT	-40.0
2025-03-08 18:55:24	TRANSFERT SORTANT	-20.0
2025-03-08 18:34:17	DEPOT	20.0
2025-03-08 18:33:44	RETRAIT	-20.0
2025-03-08 14:03:56	RETRAIT	-20.0
2017-02-10	Facture carte	-170.18
2017-02-10	Virement	43.4
2017-02-09	Prélèvement automatique	-83.06

Télécharger l'historique

FIGURE 16 – Tableau de l'historique

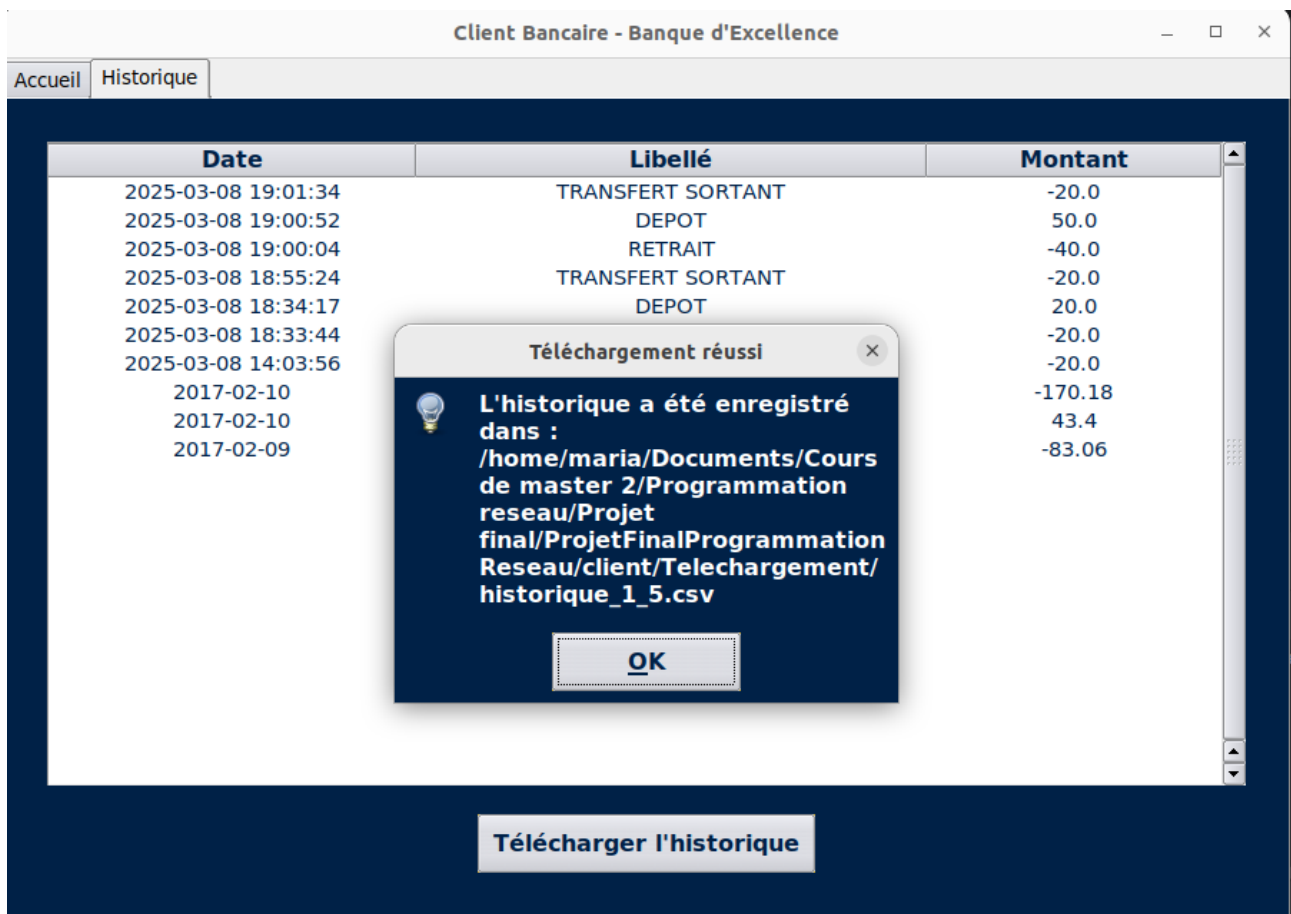


FIGURE 17 – Téléchargement de l'historique

Dans l'onglet historique, on voit la liste des 10 dernières transactions. Un bouton permet de télécharger l'historique au format CSV.

4.8 Retour à l'accueil



FIGURE 18 – Accueil après plusieurs opérations (qcceuil.png).

Après avoir réalisé plusieurs opérations, on revient à l'écran d'accueil pour visualiser la synthèse de la session en cours.

5 Analyse Wireshark

Après que le système client-serveur soit fini, nous réaliserons une analyse réseau avec **Wireshark** pour vérifier plusieurs points essentiels liés à la sécurité et au bon déroulement des échanges :

1. **Vérification du TLS** : S'assurer que la négociation TLS se déroule correctement et que le client et le serveur établissent une connexion chiffrée.
2. **Observation des échanges chiffrés** : Confirmer que les données transmises (notamment les informations sensibles comme le PIN) ne circulent pas en clair sur le réseau.
3. **Séquence des opérations** : Visualiser la chronologie des messages (authentification, retrait, dépôt, transfert, consultation d'historique) pour valider la logique d'enchaînement côté client et côté serveur.

5.1 Préparation de l'analyse

Pour capturer le trafic lié à notre projet, nous utilisons le **port 5001** (ou celui défini dans la configuration) :

```
tcp.port == 5001
```

Cela permet de se concentrer sur les paquets échangés entre le client et le serveur lors des différentes opérations.

5.2 Captures d'écran et interprétation

Nous prendrons des captures d'écran illustrant :

- **Le handshake TLS** : Les premiers paquets (Client Hello, Server Hello, etc.) montrant la mise en place du chiffrement.
- **Les messages chiffrés** : Les paquets ultérieurs devraient être marqués comme *Application Data*, indiquant qu'ils sont chiffrés.
- **Le déroulement des opérations** : Par exemple, un enchaînement de paquets correspondant à un *TESTPIN* ou à un *RETRAIT*, afin de vérifier que le contenu des requêtes n'est pas lisible en clair.

Chaque capture sera accompagnée d'une brève explication :

- **Identification des paquets** : Numéro du paquet, protocole (*TLSv1.2* ou *TLSv1.3*), port source et destination.
- **Nature de l'opération** : À quel moment se fait l'authentification, le retrait, la consultation du solde, etc.
- **Vérification du chiffrement** : Confirmation que les informations sensibles (PIN, numéro de compte) n'apparaissent pas en clair dans les champs *Info* ou *Data* de Wireshark.

5.3 Conclusion de l'analyse

L'objectif final est de **prouver que toutes les données sensibles sont correctement chiffrées** et que le **handshake TLS** se déroule sans erreur. Grâce à Wireshark, nous pouvons également confirmer que le **PIN** ou toute autre information critique n'est pas transmise en clair. Si des informations sensibles apparaissent dans les paquets, cela indiquerait un problème de configuration TLS ou une fuite potentielle à corriger.

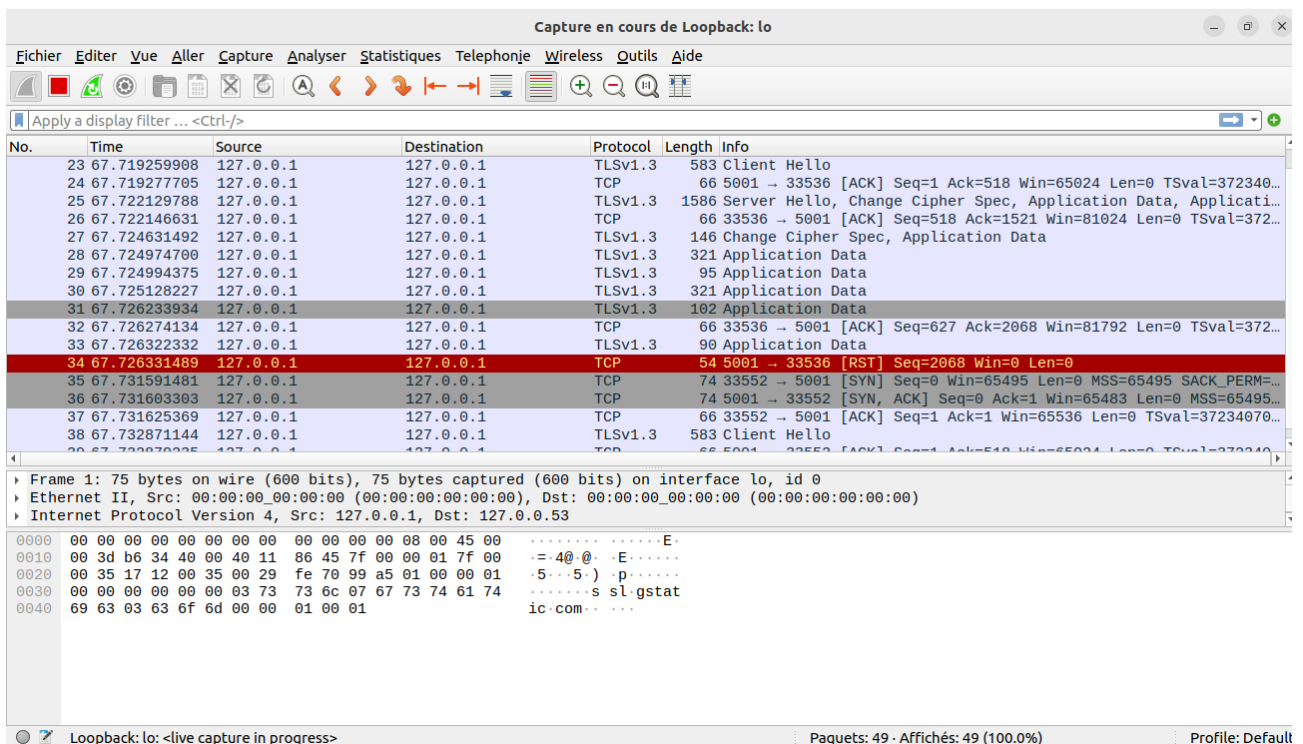


FIGURE 19 – Wireshark trafic

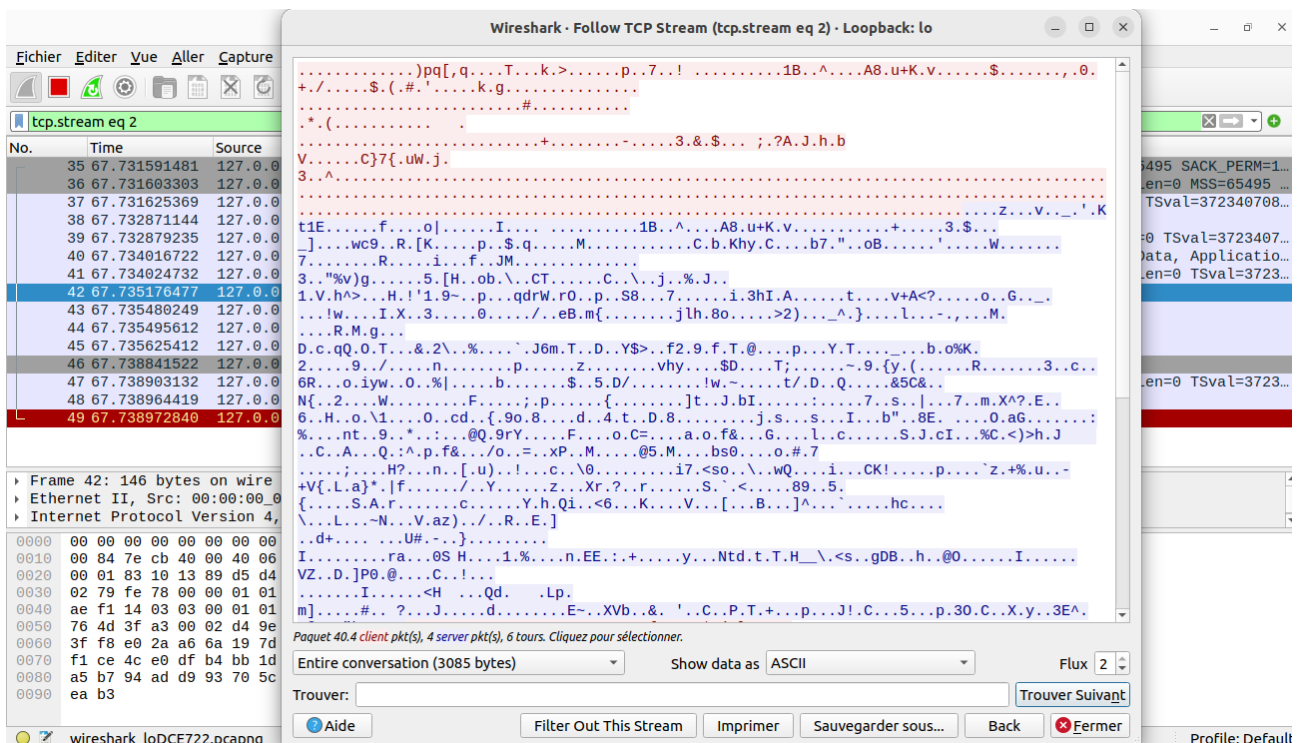


FIGURE 20 – Communication cryptée client - serveur

6 Lien github

Voici le lien du dépôt github : <https://github.com/MariaKinyanta/ProjetProgrammationReseau>

Auteur :

KINYANTA WA KINYANTA MARIA

Université Nouveaux Horizons