BANK APP – PROJET OOP

Description du Sujet et Définition du Besoin

Le projet consiste en la conception et l'implémentation d'un système bancaire. L'objectif principal est de créer une structure modulaire qui permet la gestion des clients, des employés, des comptes bancaires, et des prêts au sein d'une banque fictive. Le besoin réside dans la création d'un système robuste, extensible, et facile à comprendre, offrant des fonctionnalités bancaires de base.

Cahier des Charges

Gestion des clients

- Ajout manuel de clients. (Si besoin)
- Suppression de clients.
- Affichage des informations des clients.
- Demande et gestion des prêts par les clients.

Gestion des employés

- Ajout manuel des employés. (Si besoin)
- Suppression des employés
- Affichage des informations des employés.

Gestion des Comptes Bancaires

- Création automatique de comptes avec des numéros uniques.
- Dépôts et retraits d'argent.
- Transferts entre comptes.
- Changement de la devise des comptes.

Gestion des Prêts

- Demande de prêts par les clients.
- Approbation/Rejet des prêts par les employés
- Modification du montant des prêts par les employés.
- Affichage de l'état des prêts.

Formatted: Font: Bold, Font color: Auto

Formatted: Font: Bold, French (France)

Formatted: Font: 12 pt

Formatted: Font: 12 pt

Formatted: Heading 3 Char, Font: 12 pt, Bold, Italic, English (United States)

Formatted: French (France)

Formatted: No bullets or numbering

Formatted: Font: Bold, Italic

Formatted: Font: Bold, Italic

Fonctions Utilitaires

- Modification des informations personnelles (numéro de téléphone, email, etc).
- Envoyer message au support clients

Fonctionnalités Souhaitées (pas encore implémentées)

Système de Messagerie
Système de Connexion/Authentification
Historique des Transactions
Chat BOT
Modèle(s) ML pour aider les conseiller de prendre une décision informé et basé sur les données pour l'approbation ou non d'un prêt
Frontend et Backend pour l'application
Etc

Diagrammes UML

Diagramme de cas d'utilisations

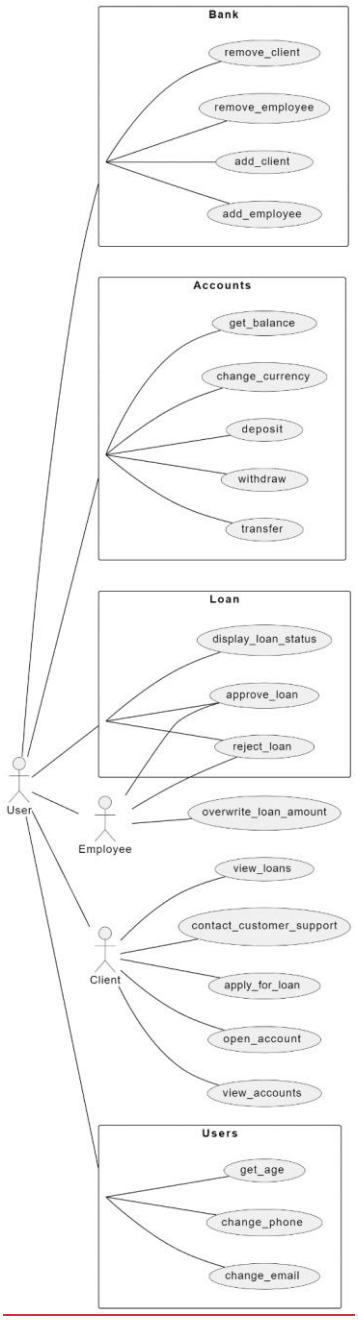
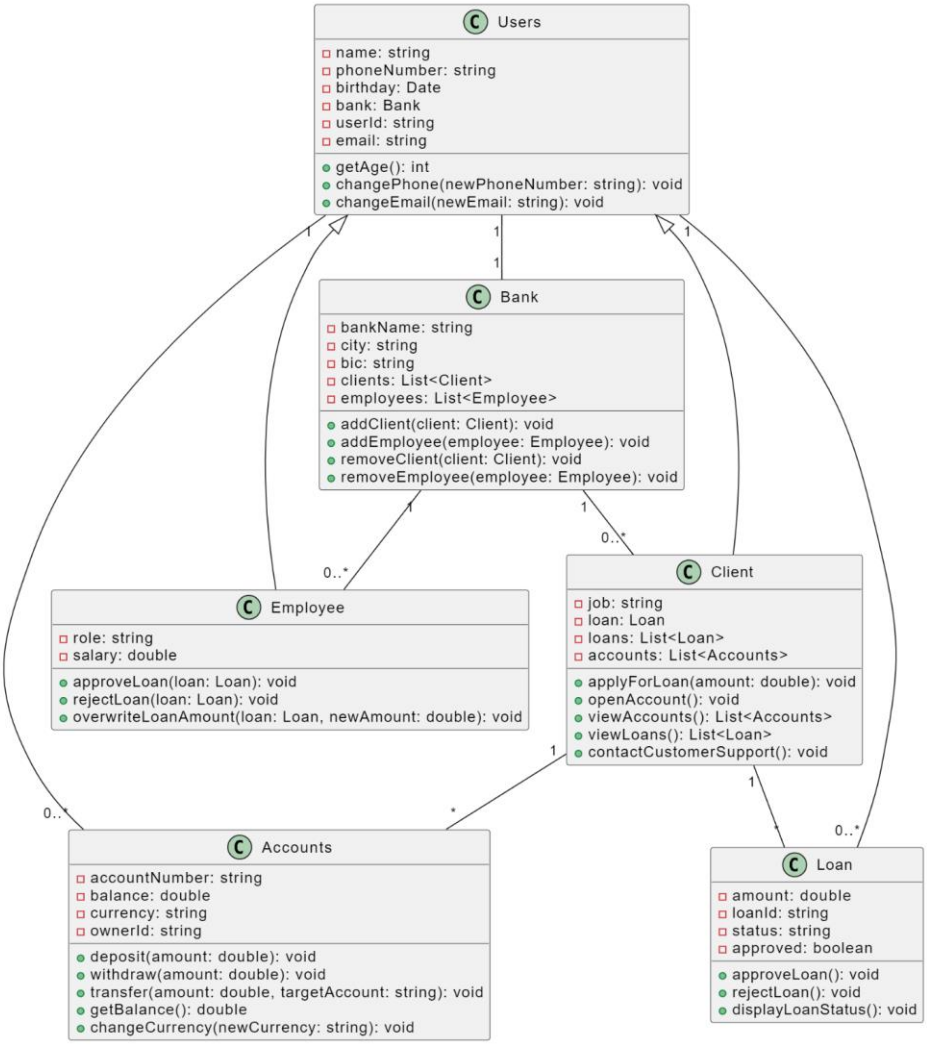

Bank
remove_client
remove_employee
add_client
add_employee
Accounts
get_balance
change_currency
deposit
withdraw
transfer
Loan
display_loan_status
approve_loan
reject_loan
overwrite_loan_amount
User
Employee
view_loans
contact_customer_support
apply_for_loan
open_account
view_accounts
Client
Users
get_age
change_phone
change_email

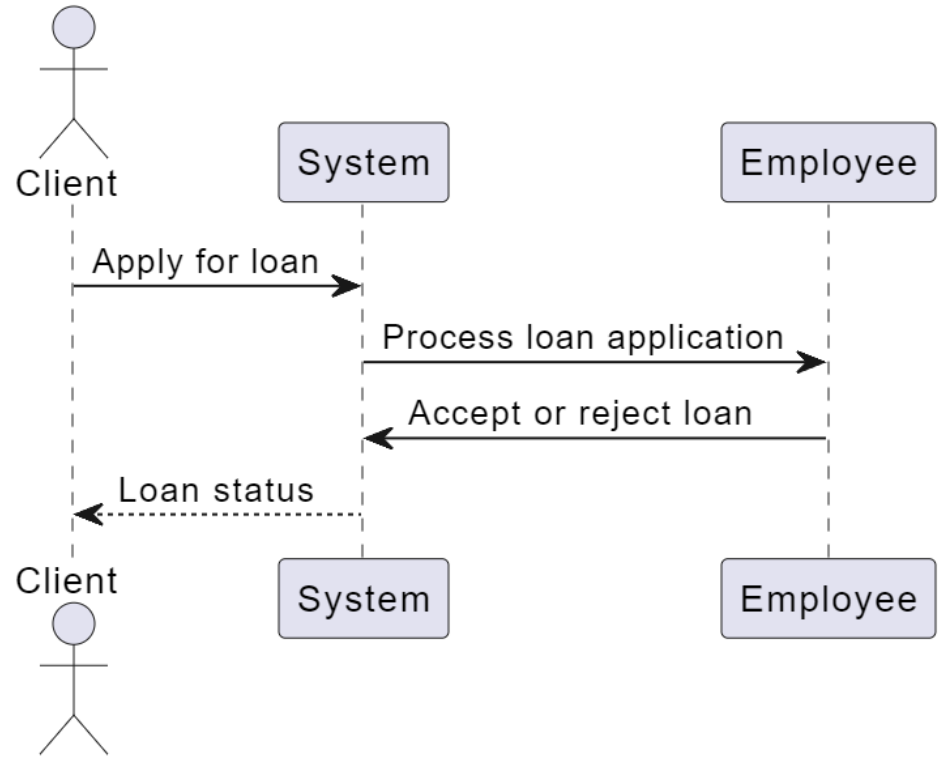
Diagramme de classes



Diagrammes de séquences décrivant différentes interactions

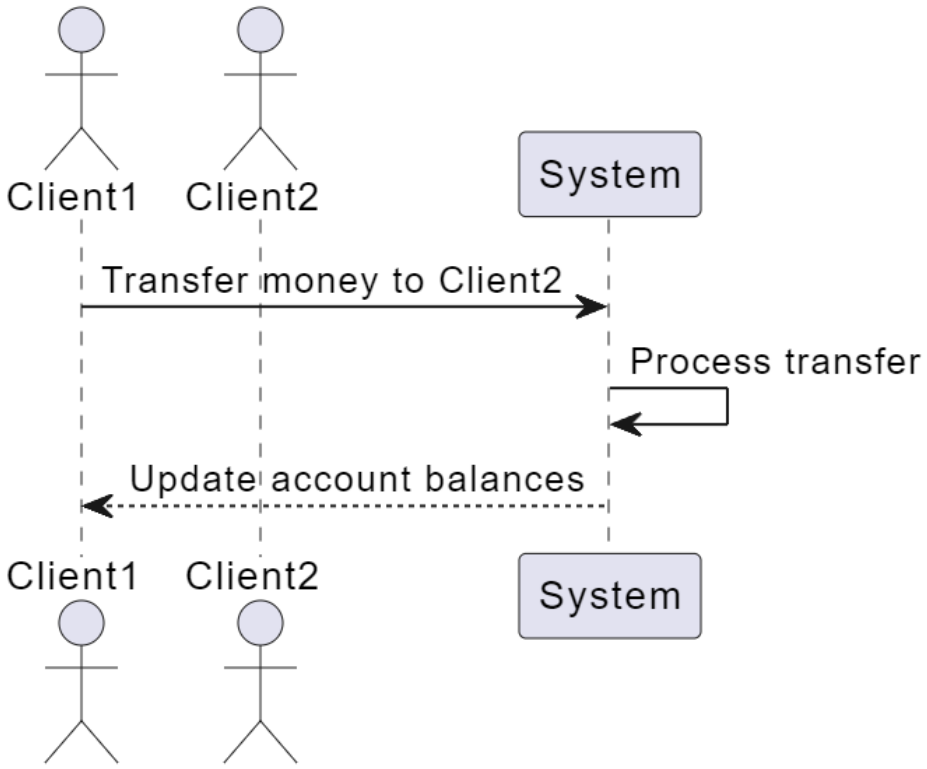
Séquence 1

Le client demande un prêt



Séquence 2

Transfert de compte entre clients



Séquence 3

Le client ouvre un compte

Difficultés rencontrées et idée d'implémentation

Idée d'Implémentation - Algorithme ML pour Approuver/Refuser les Demandes de prêt - L'idée serait d'intégrer un modèle d'apprentissage automatique capable de prendre des décisions sur l'approbation ou le rejet des demandes de prêt. Cela pourrait se faire en utilisant des caractéristiques telles que le montant du prêt, l'historique du client, le revenu, etc.

Idée d'Implémentation – les comptes, les clients, les prêts devront se retrouver par l'index et pas par l'index dans la liste (ce qui a été implémenté par moi car je ne trouvais pas une solution fonctionnelle sans avoir des problèmes secondaires)

Les difficultés rencontrées étaient plutôt liées à des interactions secondaires ou des problèmes non attendues comme par exemple : J'avais ce problème où en appelant le **client.loans** affichait la liste d'un tas d'appels **__repr__** des classes Bank, Client, User, Account, toutes combinées en une seule liste. Même en modifiant le code, en surchargeant les fonctions **__repr__** et **__str__**

J'ai eu aussi un problème similaire sur une importation circulaire sur la classe Bank et le programme qui s'appelle bank.py (J'ai eu un problème similaire pour une tâche de travail ou j'avais appelé mon script xgboost.py. Je me suis rendu compte assez vite dans les 2 cas d'où venait le problème. On apprend tous les jours ^^)