Présentation de projet

Réaliser par:

Aabirrouche bouchra

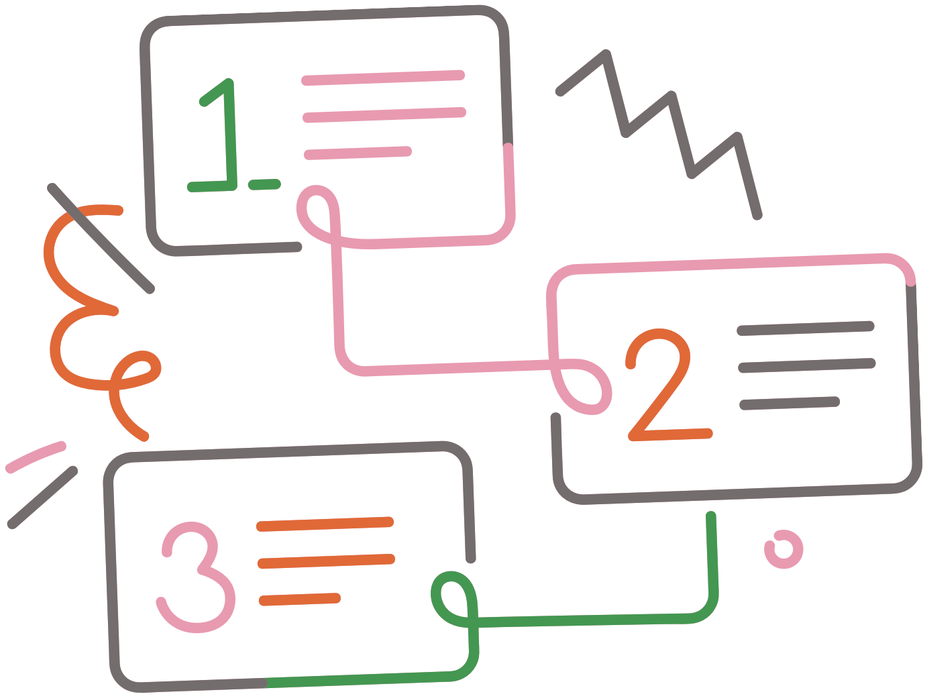
Ait m'bark houda

Boumahra ahmed

MARFAK OUSSAMA

Rouchdi Asmaa

Sommaire



01. Introduction
02. Conception
03. Modélisation
04. Stratégie
05. Algorithms
06. Résultats en C

Introduction

Les applications bancaires mobiles sont devenues un outil essentiel pour les clients des banques, leur permettant de gérer leurs comptes et d'effectuer des transactions en ligne

L'objectif de ce projet est de développer un algorithme de gestion de comptes bancaires pour une application bancaire mobile, qui permettra de :

- Gérer les comptes bancaires des clients de manière simple.
- Effectuer des transactions en ligne de manière rapide.
- Fournir des informations en temps réel sur les comptes et les transactions

UML Use Case Diagram

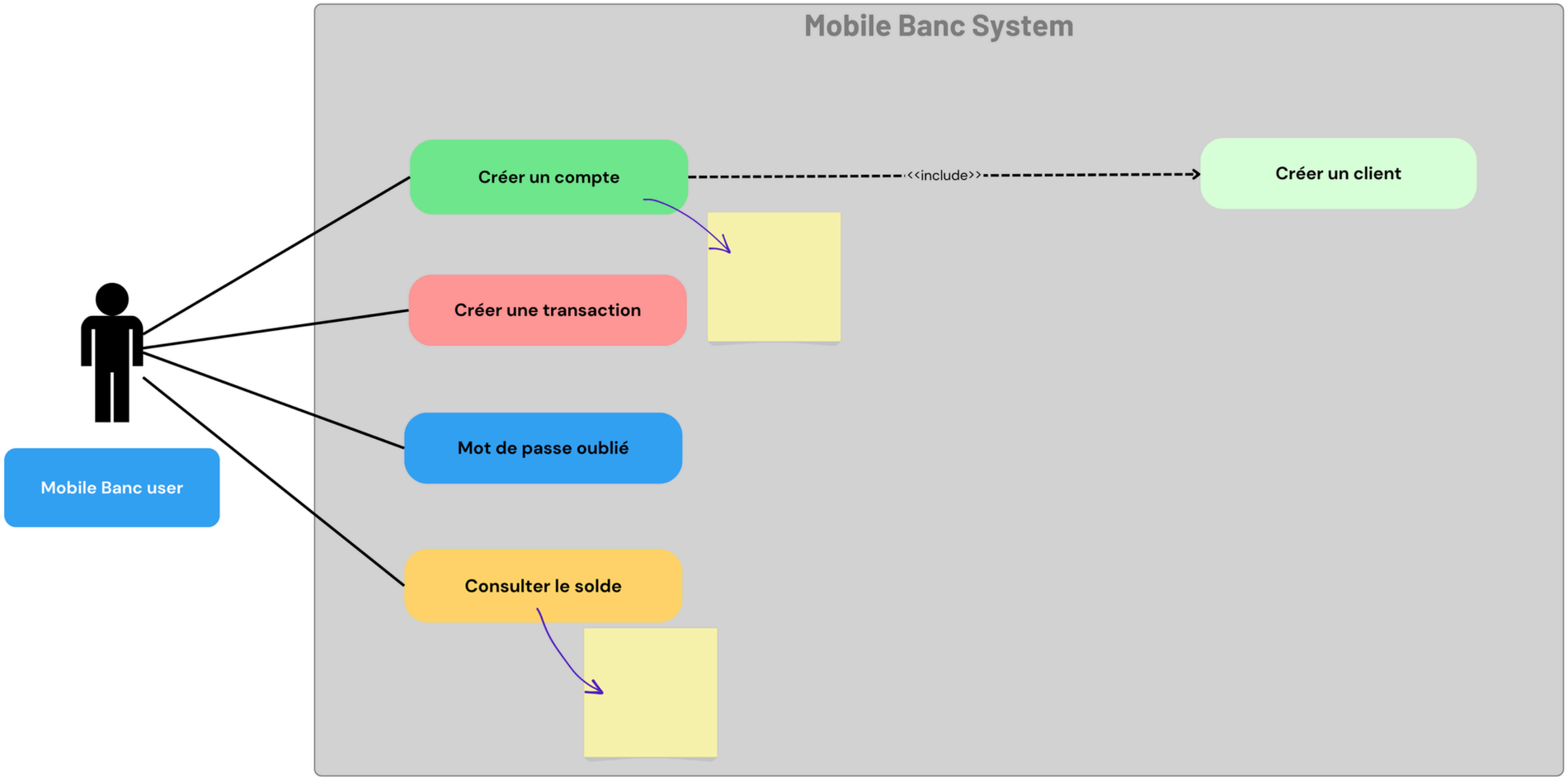
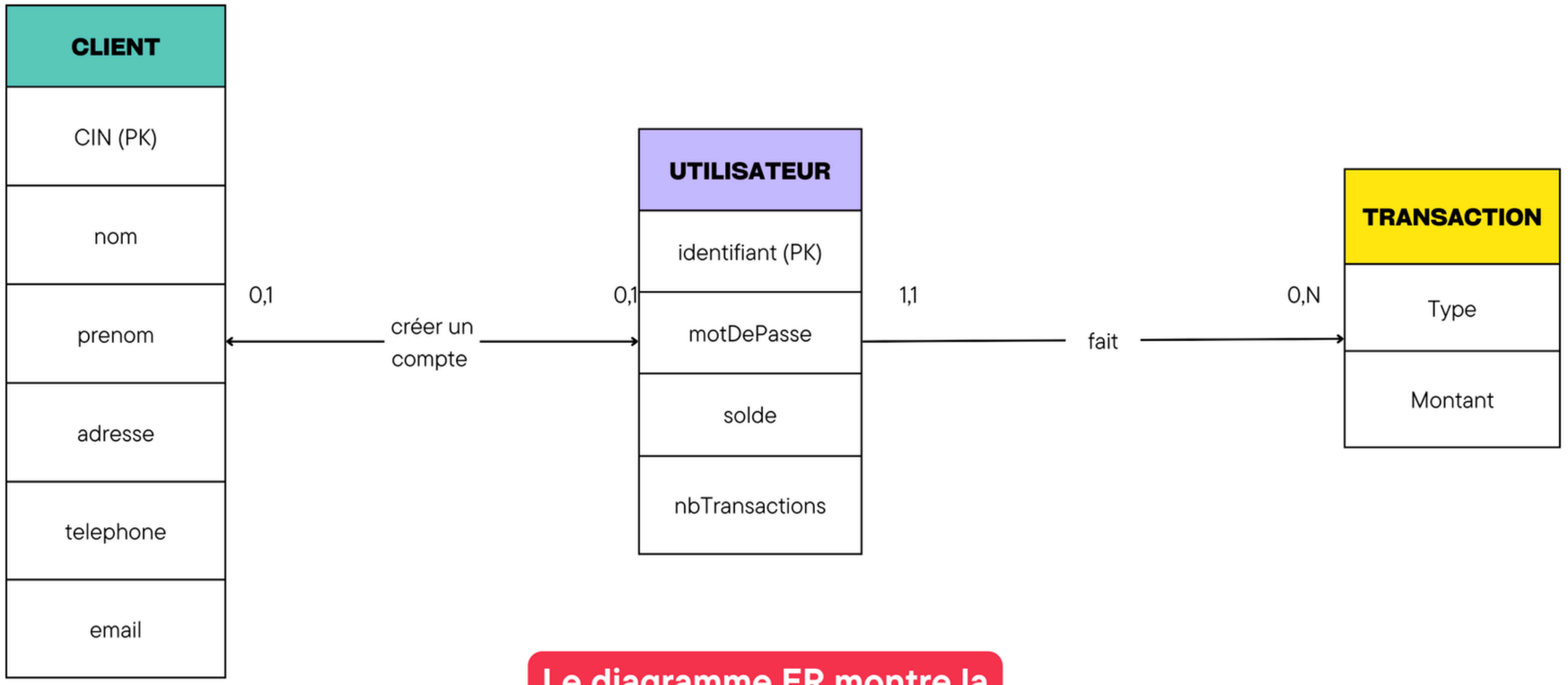


Diagramme Entité-Relation (DER)



Le diagramme ER montre la relation entre entité client et utilisateur et les transactions faites par chaque utilisateur

Stratégie

L'utilisateur peut se connecter, créer un compte ou récupérer un mot de passe.

Une fois connecté, il accède à un menu de services bancaires (solde, virements, factures...).

Les données sont stockées dans des tableaux structurés (utilisateurs, transactions).

Le système gère la sécurité avec un mot de passe et un code de vérification (exemple statique).

Trois comptes sont prédéfinis pour simuler et tester facilement l'application.

Simplifier pour un prototype: Données fixes, code fixe, pas de fichier

Algorithmes

Fonction verifier_identifiants(user : Chaine de caractères , password : Chaine de caractères) : BOOLEEN

```
Variables
i : entier
Début
Pour i ← 0 à nb_utilisateurs - 1 Faire
Si utilisateurs[i].identifiant = user ET utilisateurs[i].motDePasse = password Alors
        Retourne VRAI
FinSi
 FinPour
Retourne FAUX
Fin
```

Fonction verifier_code(code_saisi : Chaine de caractères, code_attendu : Chaine de caractères) : BOOLEEN

```
Début
Si code_saisi = code_attendu Alors
Retourne VRAI
Sinon
Retourne FAUX
 FinSi
Fin
```

Algorithmes

Procédure changer_mot_de_passe(identifiant, nouveau_mdp : Chaine de caractères)

```
Variables i : entier
Début
Pour i ← 0 à nb_utilisateurs - 1 Faire
Si utilisateurs[i].identifiant = identifiant Alors
        utilisateurs[i].motDePasse ← nouveau_mdp
FinSi
 FinPour
Fin
```

Procédure envoyer_code_verification(cin, email : Chaine de caractères)

```
Variables
i : entier
client_trouve : BOOLEEN
code_utilisateur, code_envoye, nouveau_mdp : Chaine de caractères
```

Algorithmes

```
Début
client_trouve ← FAUX
 Pour i ← 0 à nb_clients - 1 Faire
 Si base_clients[i][0] = cin ET base_clients[i][1] = email Alors
        client_trouve ← VRAI
FinSi
FinPour
Si client_trouve = Vrai Alors
Ecrire("Code envoyé à l'email.")
code_envoye ← "123456"/* Simulation de code envoyé*/
Ecrire("Saisir le code envoyé :")
Lire(code_utilisateur)
Si verifier_code(code_utilisateur, code_envoye) Alors
        Ecrire("Saisir le nouveau mot de passe :")
        Lire(nouveau_mdp)
        changer_mot_de_passe(cin, nouveau_mdp)
        Ecrire("Mot de passe mis à jour")
Sinon
        Ecrire("Code incorrect")
    FinSi
Sinon
Ecrire("CIN ou email incorrect")
 FinSi
Fin
```

Algorithmes

```
Fonction authentifier(identifiant, motDePasse : Chaine de caractères) : Entier
Variables i : entier
Début
Pour i ← 0 à nb_utilisateurs - 1 Faire
Si utilisateurs[i].identifiant = identifiant ET utilisateurs[i].motDePasse = motDePasse Alors
        Retourne i
    FinSi
 FinPour
Retourne -1
Fin
Procédure ajouter_transaction(u : REF Utilisateur, desc : Chaine de caractères, montant : Réel)
Début
Si u.nbTransactions < MAX_TRANSACTIONS Alors
u.historique[u.nbTransactions].description ← desc
u.historique[u.nbTransactions].montant ← montant
u.nbTransactions ← u.nbTransactions + 1
FinSi
Fin
```

Algorithmes

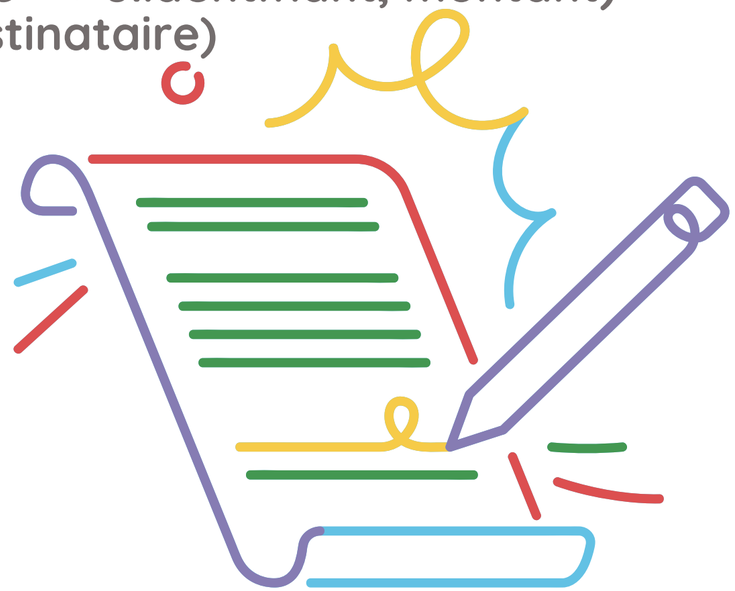
```
        Procédure consulter_solde(u : REF Utilisateur)
Début
Ecrire("Votre Solde est : ", u.solde)
Fin


Procédure voir_transactions(u : REF Utilisateur)
Variables i : entier
Début
Si u.nbTransactions = 0 Alors
Ecrire("Aucune transaction.")
Sinon
Pour i ← 0 à u.nbTransactions - 1 Faire
        Ecrire(u.historique[i].montant, " - ", u.historique[i].description)
    FinPour
  FinSi
Fin
```

Algorithmes

```
Procédure faire_virement(u : REF Utilisateur)
Variables identifiant_destinataire : Chaine de caractères, montant : Réel, destinataireIndex, i : entier
Début
Ecrire("Saisir l'identifiant du destinataire :")
Lire(identifiant_destinataire)

Ecrire("Saisir le montant que vous voulez envoyer :")
Lire(montant)

Si montant ≤ 0 OU montant > u.solde Alors
Ecrire("Montant invalide ou solde insuffisant.")
Retour
  FinSi

destinataireIndex ← -1
Pour i ← 0 à nb_utilisateurs - 1 Faire
Si utilisateurs[i].identifiant = identifiant_destinataire Alors
        destinataireIndex ← i
    FinSi
FinPour
```

Algorithmes

```
Si destinataireIndex = -1 Alors
Ecrire("Destinataire introuvable.")
Retour
   FinSi

u.solde ← u.solde - montant
utilisateurs[destinataireIndex].solde ← utilisateurs[destinataireIndex].solde + montant
ajouter_transaction(u, "Virement vers " + identifiant_destinataire, -montant)
ajouter_transaction(utilisateurs[destinataireIndex], "Virement reçu de " + u.identifiant, montant)
Ecrire("Virement de ", montant, " MAD effectué vers ", identifiant_destinataire)
Fin
```



Résultats en C

Les résultats obtenus après compliation en language C ont va les afficher à l’aide du programme dev C++ .

Merci pour votre attention !