ACADÉMIE
DE TOULOUSE
Liberté
Égalité
Fraternité

iut Midi-Pyrénées BLAGNAC

Documentation technique V1

GROUPE 3A3
PHAKEOVILAY Andrew
ZEJNULLAHI Egxon
ZOULI-BARRERE Karim

Table des matières

ACADÉMIE
DE TOULOUSE
Liberté
Égalité
Fraternité

BLAGNAC
iut
Midi-Pyrénées

Présentation de l'application

L'application permet de gérer des comptes bancaires de dépôt pour des clients. L'application possède actuellement quelques fonctionnalités tels que modifier les informations clients, créer un nouveau client, consulter un compte, créer un compte, créditer/débiter un compte, effectuer un virement, clôturer un compte, gérer les employés et rendre inactif un client à l'aide des boutons interactifs et des champs à remplir.

Architecture

Le projet de l'application est travaillé sous Eclipse où nous pouvons catégoriser les classes sous des packages afin d'avoir une meilleure visibilité et un meilleur repère pour travailler.

Architecture

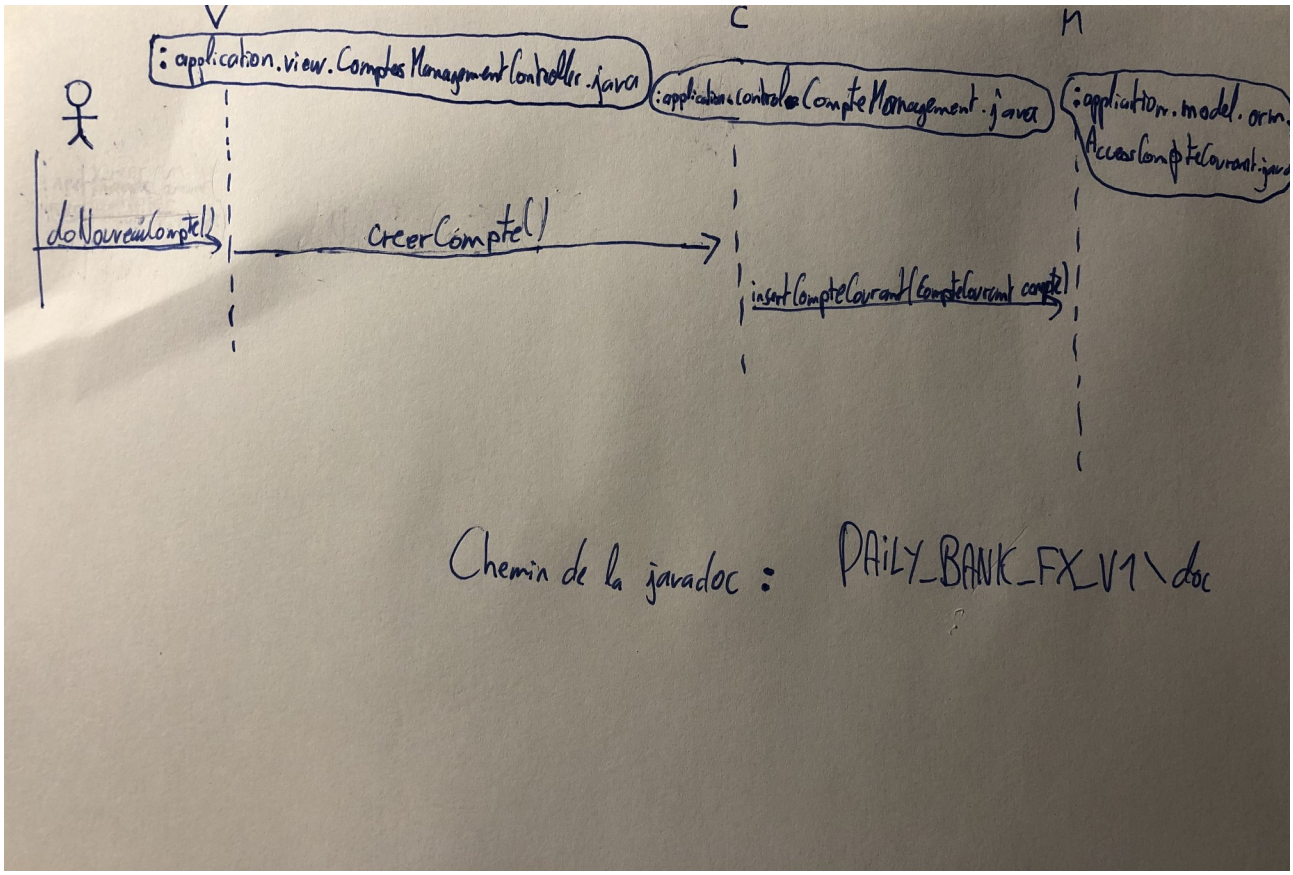
Une partie java avec des packages java et une partie avec des packages pour la base de donnée.

Nous avons un fichier .jar nommé ojdbc6.jar qui est une librairie.

Nous avons des packages :

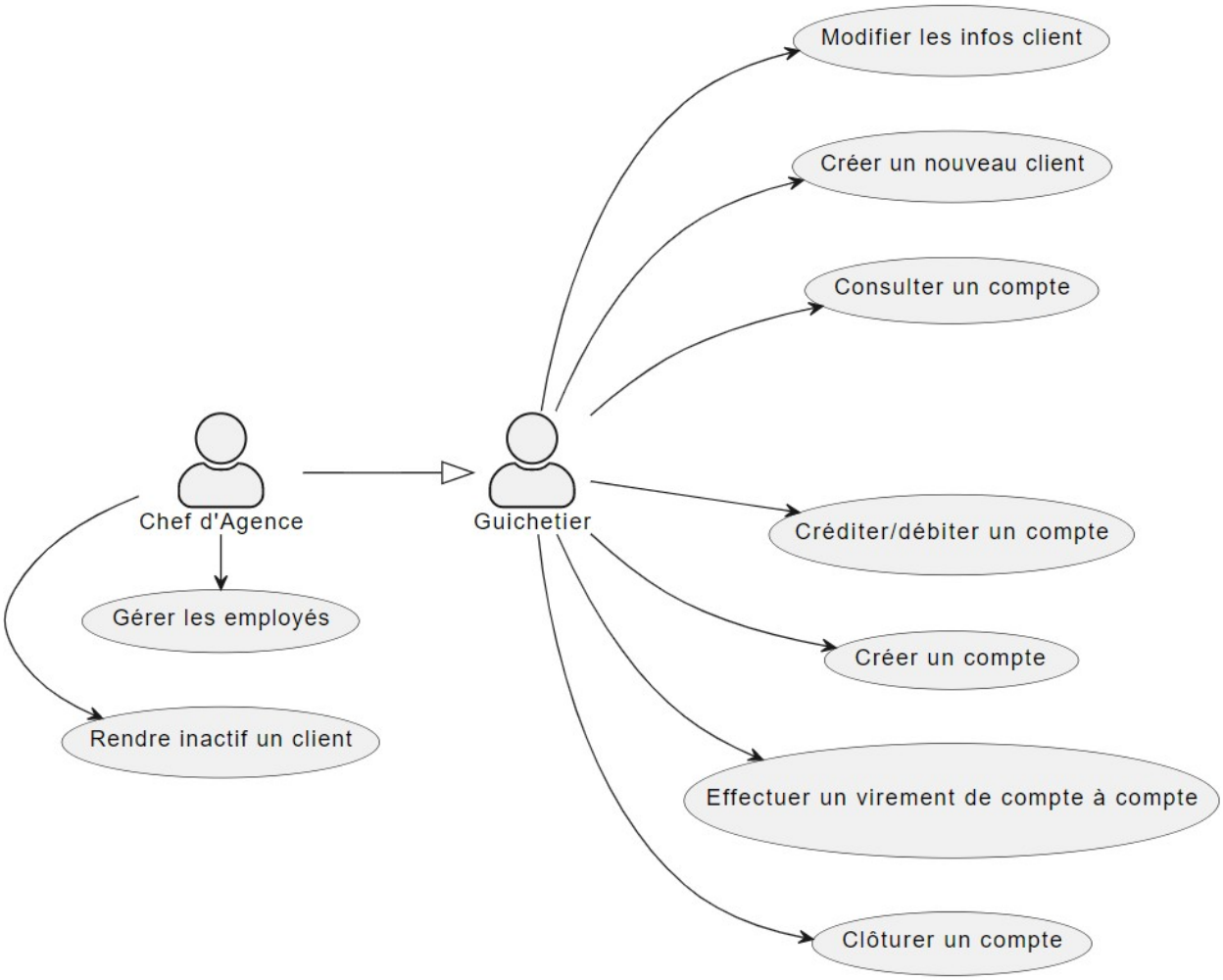
- application qui contient tous les packages dans lesquelles on retrouve les .java liés à l'application.
- application.control qui contient des contrôleurs liés aux fichiers FXML permettant aussi d'accéder aux données
- application.tools qui contient les fichiers .java essentiel au fonctionnement d'application.view et application.control, car dans application.tools permet le fonctionnement des contrôleurs pour afficher l'interface.
- application.view où il y a les fichiers fxml permettant de faire des schémas à l'aide de Scene Builder et il y a les classes contrôleurs permettant de faire fonctionner des contrôles de l'interface à l'aide des fonctions
- model.data qui contient les différents constructeurs utiles pour intéragir avec la base de donnée. Chaque constructeur correspond à une table sur la base de donnée.
- model.orm qui contient les classes java connectés à la base de donnée, les classes effectuent des requêtes SQL et communique avec la classe model.data.
- model.exception gère les exceptions lorsque un problème intervient avec la base de donnée quand une erreur survient lors de l'exécution des classes du package model.orm

Nous avons le modèle MVC (Modèle Vue Contrôleur) illustré par ce schéma :



Présentation et explication de tous les diagrammes demandés

Use Case de l'application



Nous avons deux types d'utilisateurs :

Le guichetier peut :

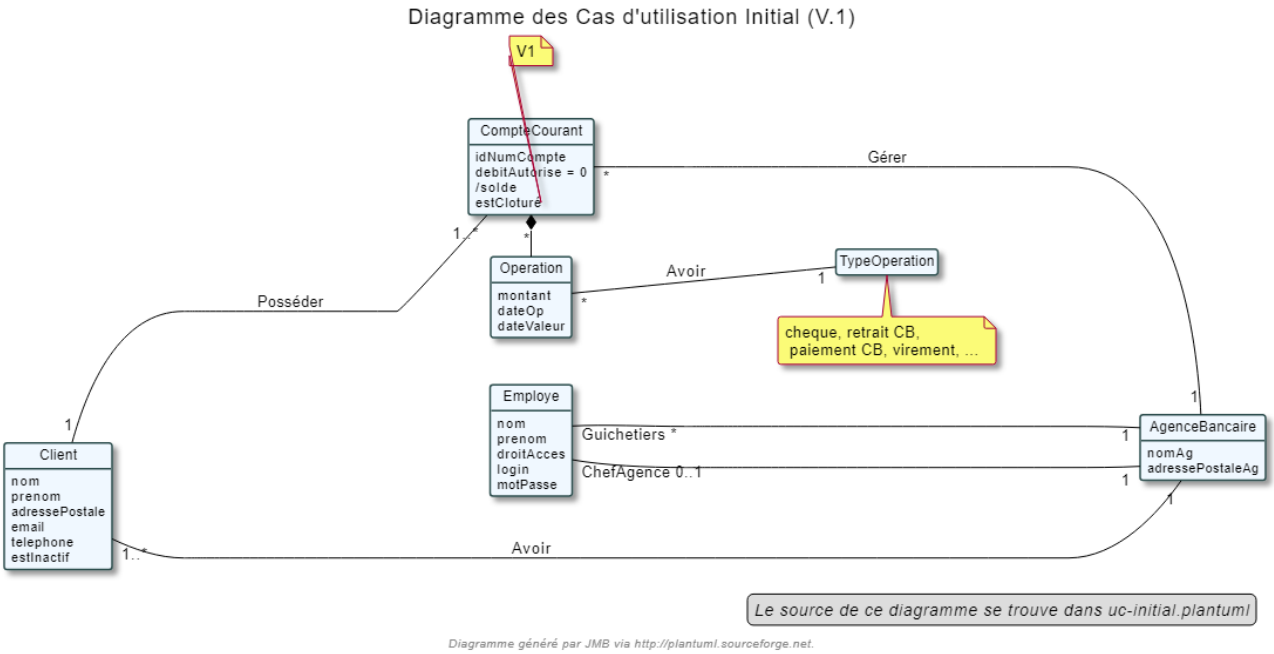
- Modifier les infos client
- Créer un nouveau client
- Consulter un compte
- Créditer/débiter un compte
- Créer un compte
- Effectuer un virement de compte à compte

- Clôturer un compte

Le chef d'agence en plus des fonctionnalités du guichetier, il peut :

- Gérer les employés
- Rendre inactif un client

Diagramme des Cas d'utilisation initial (V.1)



Une agence bancaire dispose de plusieurs employés

Une agence bancaire a un ou plusieurs clients

Une agence bancaire gère plusieurs comptes courants

Un employé peut être un chef d'agence ou un guichetier selon ses droits d'accès

Un client a une agence bancaire

Un client possède un ou plusieurs comptes courants

Un compte courant est géré par un agence bancaire

Un compte courant est composé d'opérations

Une opération a un type d'opération

Dans la V1, le compte courant a un été estCloturé et les ils y a plusieurs TypeOperation (cheque, retrait CB, paiement CB, virement)

Éléments de codes significatifs commentés

```java
public void supprimerCompteCourant(CompteCourant compte)
        throws RowNotFoundOrTooManyRowsException, DataAccessException, DatabaseConnexionException {

    try {
        Connection con = LogToDatabase.getConnexion();

        String query = "UPDATE COMPTECOURANT SET solde = 0, estCloture = 'O'  WHERE idNumCompte = " + "?";

        PreparedStatement pst = con.prepareStatement(query);

        pst.setInt(1, compte.idNumCompte);

        System.err.println(query);

        int result = pst.executeUpdate();
        pst.close();

        if (result != 1) {
            con.rollback();
            throw new RowNotFoundOrTooManyRowsException(Table.CompteCourant, Order.UPDATE,
                    "Suppression anormal", null, result);
        }

        con.commit();

    } catch (SQLException e) {
        throw new DataAccessException(Table.CompteCourant, Order.UPDATE, "Erreur accès", e);
    }
}
```

Cette fonction permet de clôturer un compte courant de l'agence bancaire.

La variable query de type String correspond à la commande qui sera utilisé pour une précompilation SQL.

pst.setINt(1, compte.idNumCompte) permet de changer le « ? » en la valeur saisie.

Ensuite, la variable result de type int permet de voir si la commande a bien marché lors du test avec le if juste en dessous.