INSTITUT FRANCOPHONE INTERNATIONAL

ĐẠI HỌC QUỐC GIA HÀ NỘI
ĐHQGHN
VNU
Since 1906

RAPPORT TP1 GENIE LOGICIEL

THEME :

SIMPLE APPLICATION DE GESTION BANCAIRE

Rédigé par :

KENGNI Hippolyte et OUSSEINI.H Abdoul-Djalil
Promotion XXII

En vue de l'obtention du Master Recherche en Systèmes Intelligents et Multimédia

SOUS L'ENCADREMENT

M. HO Tuong Vinh, Ph.D
Directeur adjoint IFI

Année académique: 2017/2018

SOMMAIRE

TABLE DES ILLUSTRATIONS

INTRODUCTION

Dans un monde de plus en plus moderne, disposer d'un compte bancaire est nécessaire pour sécuriser son argent, percevoir un salaire, une allocation ou faire des achats, payer son loyer ou ses factures... En ouvrant un compte bancaire, vous pouvez disposer d'une multitude de solutions pour gérer votre argent au quotidien en toute sécurité.

Ainsi donc, Les établissements bancaires et financiers doivent faire face à l'augmentation de la clientèle qui est de plus en plus croissante, pour y répondre il est important de mettre en œuvre un programme quasi infaillible permettant d'administrer les comptes des clients.

Notre travail consistera à élaborer une application de gestion de comptes bancaires avec Java. Le but du projet est d'implémenter l'application avec une démarche orientée objet, il ne suffit pas d'implémenter mais il faut aussi décrire de manière visuelle et graphique les besoins et les solutions fonctionnelles et techniques du projet logiciel, ceci nous amènera donc à modéliser avec UML. Dans l'étude qui suivra, nous présenterons la spécification de notre application, sa conception à travers les différentes classes, son implémentation et le test.

I. LA SPECIFICATION DE L'APPLICATION

Le succès de tout projet dépend étroitement de la capacité de ses acteurs à respecter point par point les clauses définies dans le cahier de charges. Il s'agit là de faire une brève présentation des opérations que peut faire la banque afin de donner une vision globale de notre application, elle permettra éventuellement de :

- Gestion des clients : ici, il s'agira d'enregistrer un client, consulter un client, modifier un client, supprimer un client, faire une recherche sur un client donné ;
- Gestions des comptes : ici nous allons créer les comptes des différents clients, le gestionnaire aura la possibilité de créer plusieurs comptes différents pour un client donné, nous pourra faire une recherche à l'aide de l'identification du client afin de savoir si ce dernier possède un ou plusieurs comptes (compte courant ou épargne). Le numéro de compte et le numéro d'identification du titulaire du compte seront générés automatiquement par le système. Le gestionnaire pourra aussi faire une mise à jour du solde des comptes (le système calculera automatique les intérêts à ajouter à chaque compte et modifiera le solde de tous les comptes par le solde + l'intérêt correspondant).
- Gestions des opérations bancaires : ici, le gestionnaire aura la possibilité d'effectuer un dépôt ou un retrait (lors du retrait, le système vérifiera automatiquement si le montant à retirer n'est pas supérieur au solde en compte ; si dès est le cas, l'opération sera annulé) dans un compte précis. Le gestionnaire pourra aussi visualiser les différentes transactions effectuées, leur date, et les comptes ayant un solde de moins de 100€.

II. LA CONCEPTION

Une bonne méthodologie de réalisation de logiciel suppose une bonne maitrise de l'analyse et de la conception, cette dernière nous offre tous les modèles destinés à assurer le fonctionnement du logiciel. Ces modèles permettent d'expliciter les fonctionnalités. De manière globale, elle offre une vue panoramique sur l'ensemble des éléments et les interactions pris en compte dans la conception. Pour ce faire nous utiliseront UML pour modéliser notre futur système.

UML est un langage pour :

- ❖ Visualiser chaque symbole graphique à une sémantique ;
- ❖ Spécifier de manière précise et complète, sans ambiguïté ;
- ❖ Construire les classes, les relations SQL peuvent être généré automatiquement.

Le langage UML comprend 9 diagrammes principaux à savoir :

- ❖ Le diagramme des cas d'utilisation ;
- ❖ Le diagramme de classe ;
- ❖ Le diagramme d'activité ;
- ❖ Le diagramme d'Etat transition ;
- ❖ Le diagramme de déploiement ;
- ❖ Le diagramme de séquence ;
- ❖ Le diagramme d'objet ;
- ❖ Le diagramme de composant ;
- ❖ Le diagramme de collaboration.

Dans notre cas d'étude, nous nous limiterons à la modélisation de trois (03) diagrammes à savoir :

- Le diagramme des cas d'utilisation : représente la structure des fonctionnalités nécessaires aux utilisateurs du système. Il est normalement utilisé lors des étapes de capture des besoins fonctionnels et techniques ;
- Le diagramme des classes : sûrement l'un des diagrammes les plus importants dans un développement orienté objet. Sur la branche fonctionnelle, ce diagramme est prévu pour développer la structure des entités manipulées par les utilisateurs. En conception, le diagramme de classes représente la structure d'un code orienté objet ;
- Le diagramme de séquence : représente les échanges de messages entre objets, dans le cadre d'un fonctionnement particulier du système.

1. Diagramme de Cas d'utilisation

Pour illustrer les fonctionnalités de notre application. Il permet de ressortir les différents acteurs du système, les fonctionnalités et ainsi que les relations entre eux.

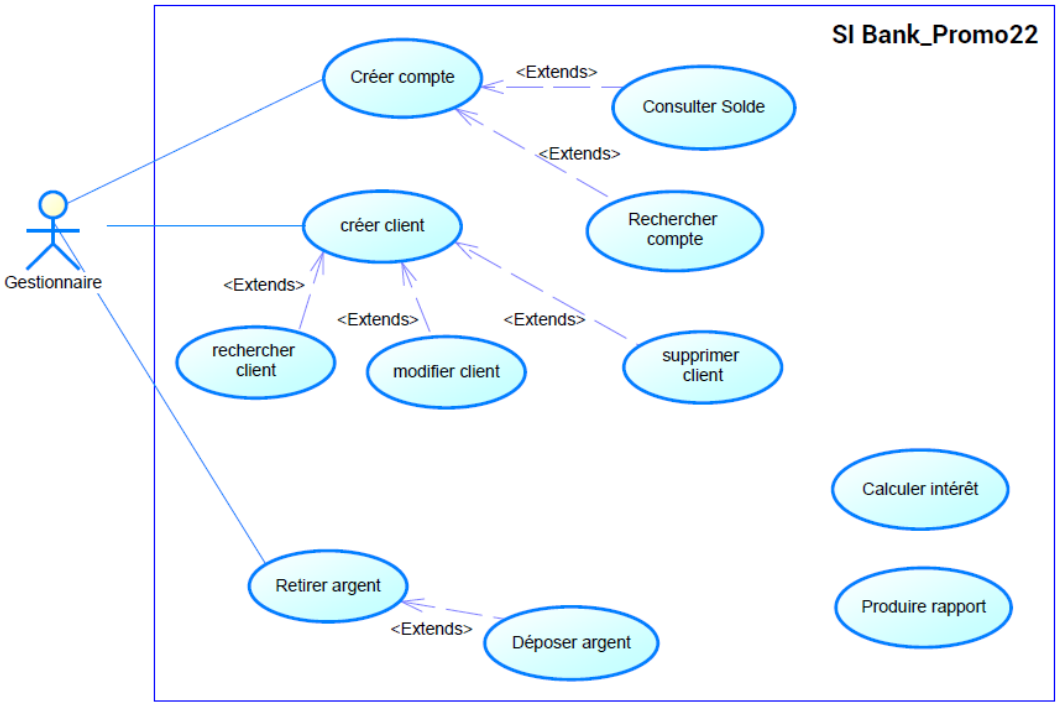

Figure 1: Diagramme de Cas d'utilisation

2. Diagramme des classes

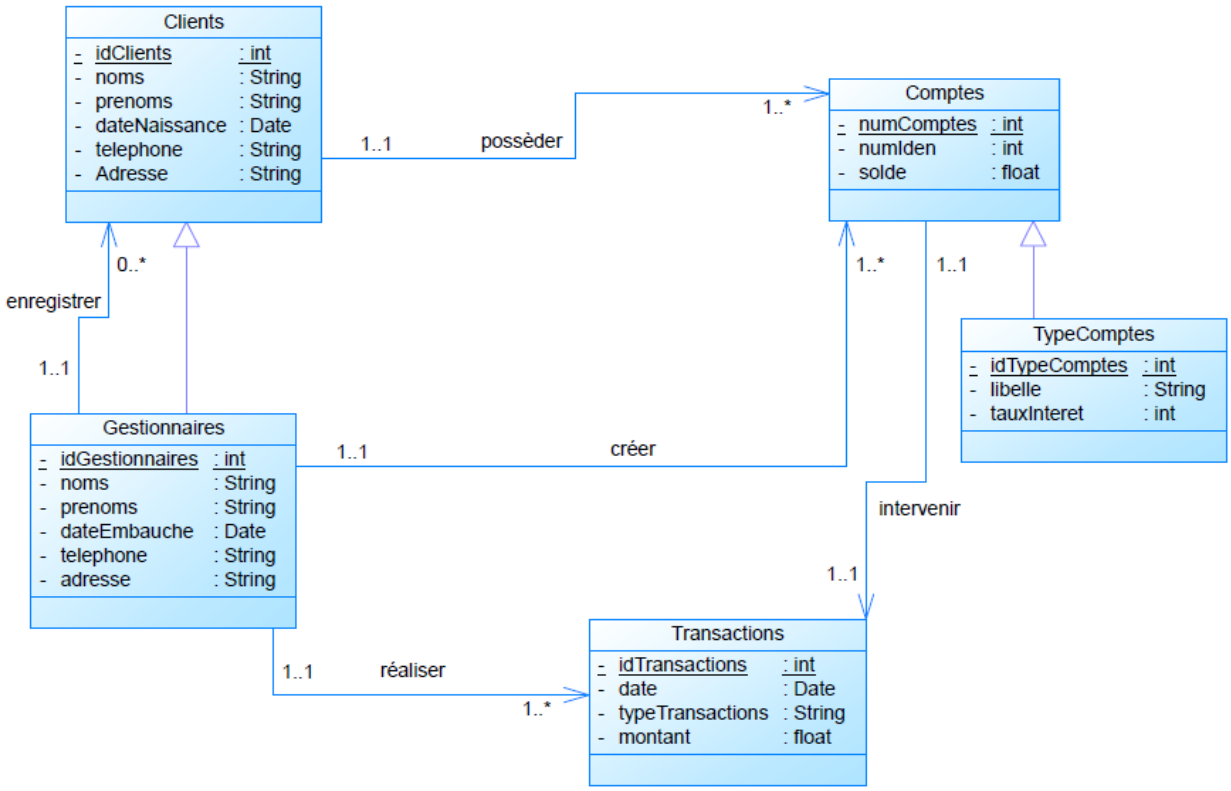

Figure 2: Diagramme des classes

3. Diagramme de séquence

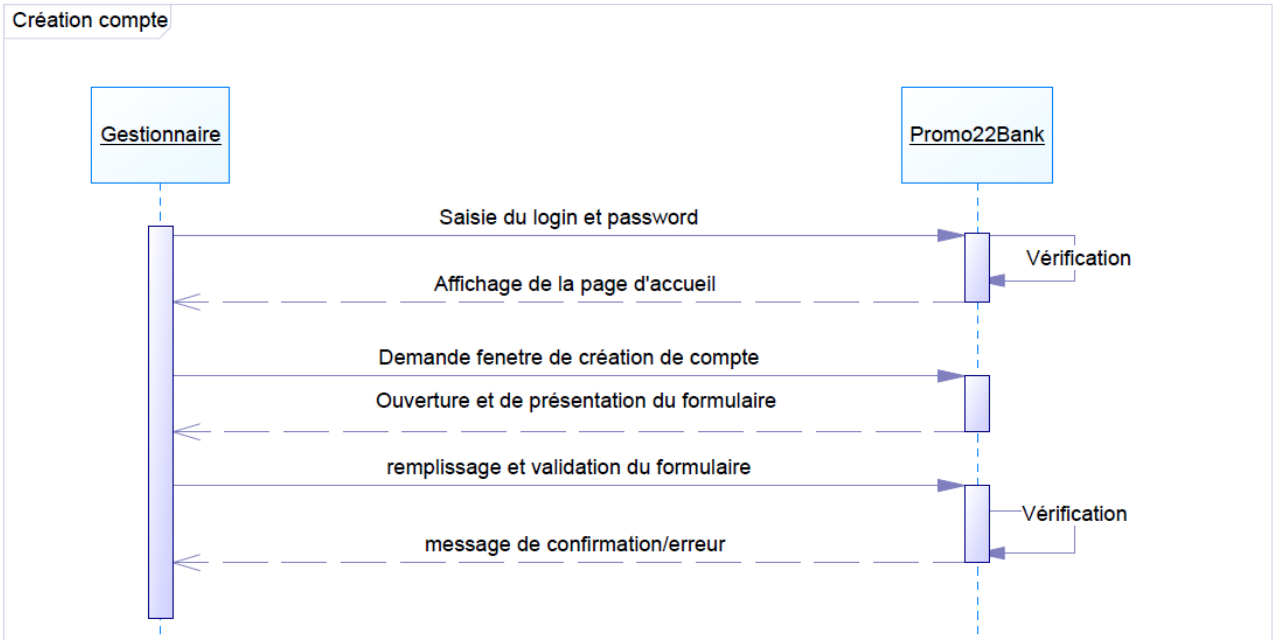

Figure 3: Diagramme de séquence de création d'un compte

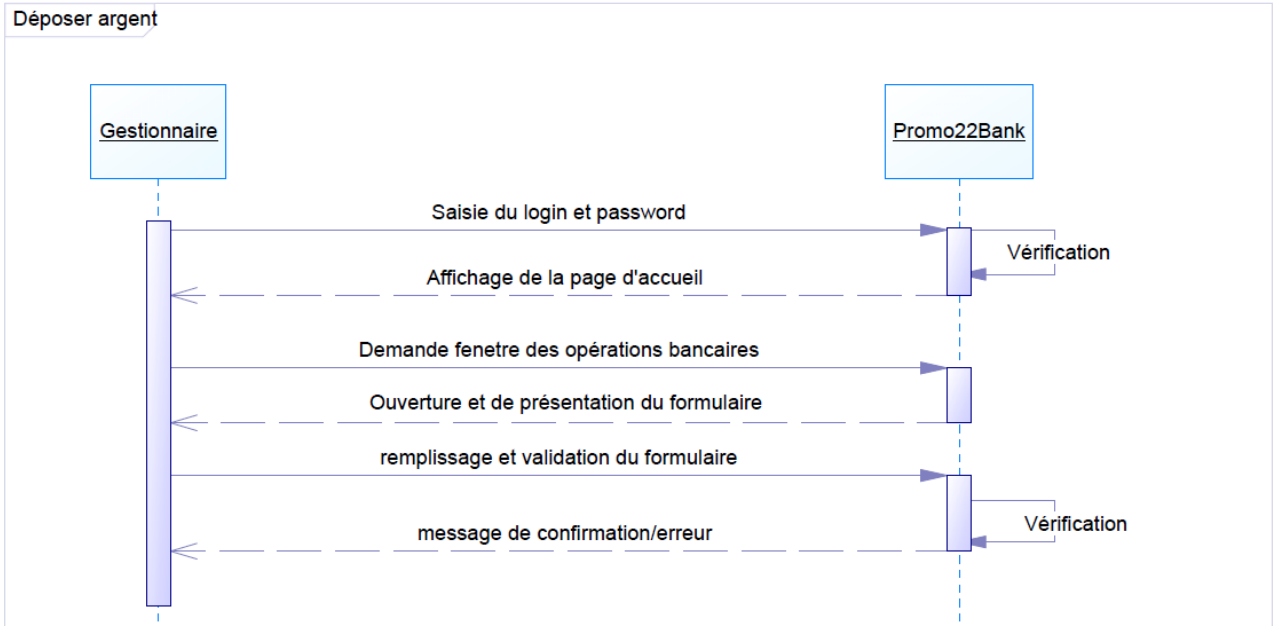

Figure 4: Diagramme de séquence Déposer argent

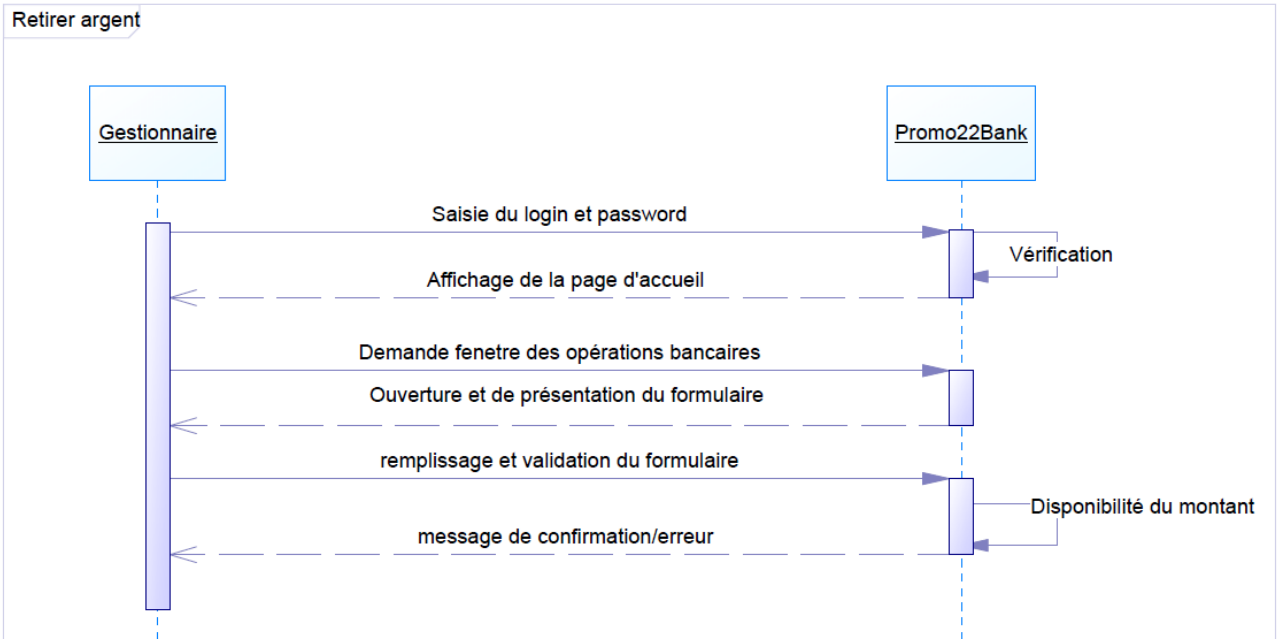

Figure 5: Diagramme de séquence Retirer argent

A partir de ces diagrammes nous avons pu générer le schéma de la base de données suivant :

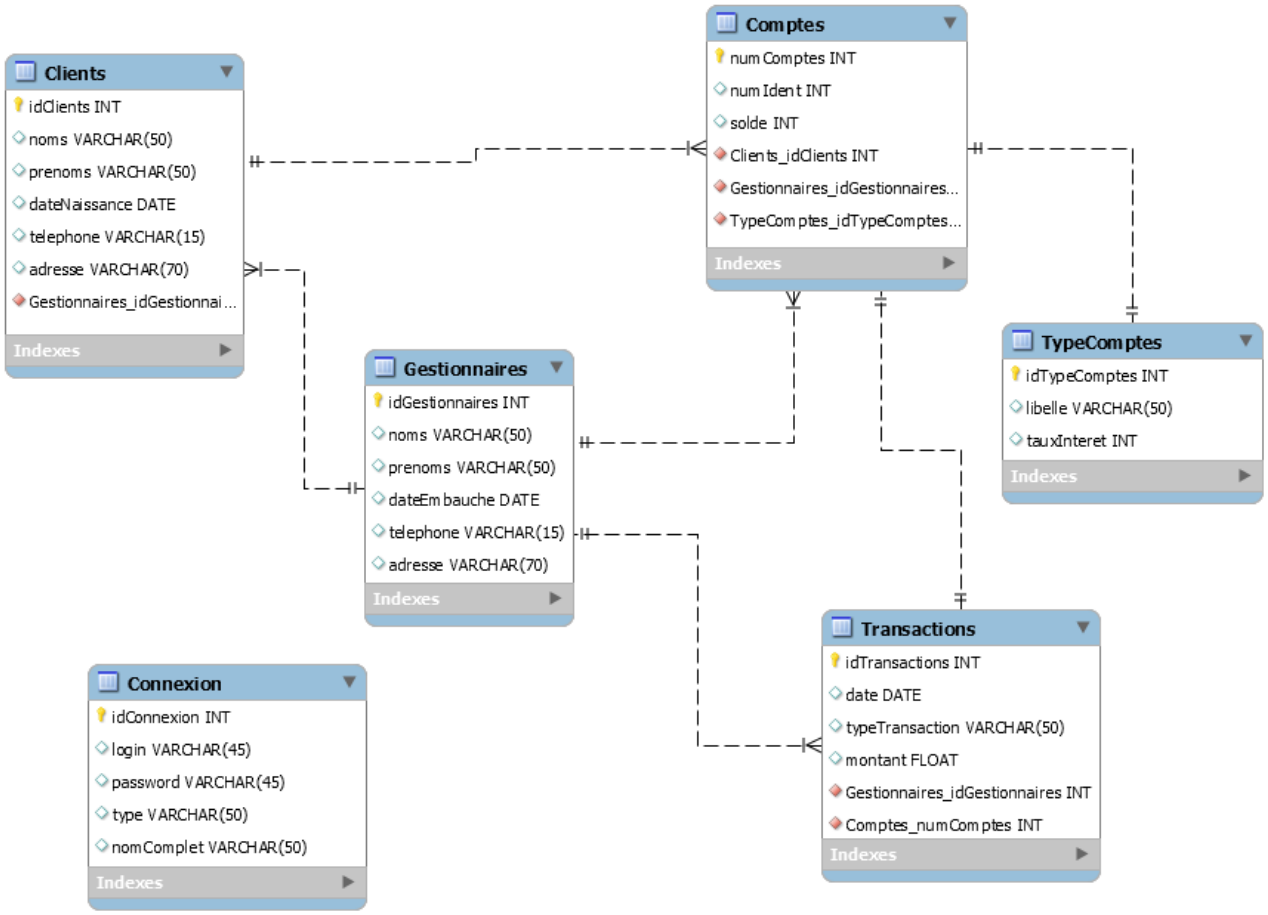

Figure 6: Architecture de la base de données de notre application

III. IMPLEMENTATION ET TESTS

1. Implémentation

Afin d'implémenter la solution de notre système, nous avons utilisé :

- Eclipse : Environnement de développement libre ;
- mysql-connector-java-5.1.28-bin : librairie pour la connexion à la base de données ;
- jcalendar-1.4 : librairie java pour la gestion des dates ;
- SQLyog-11.5.0-1_x86-x64 : comme gestionnaire de la base de données.

L'architecture suivante nous présente l'organisation ou la structure d'un projet java. La couche service contient tous les traitements ou méthodes métiers, la couche Dao contient toutes les méthodes reliées à la base de données, la couche présentation contient les interfaces du système. Nous pouvons ainsi différencier le Front (gestion des interfaces) avec le Back (gestion de la partie métier).

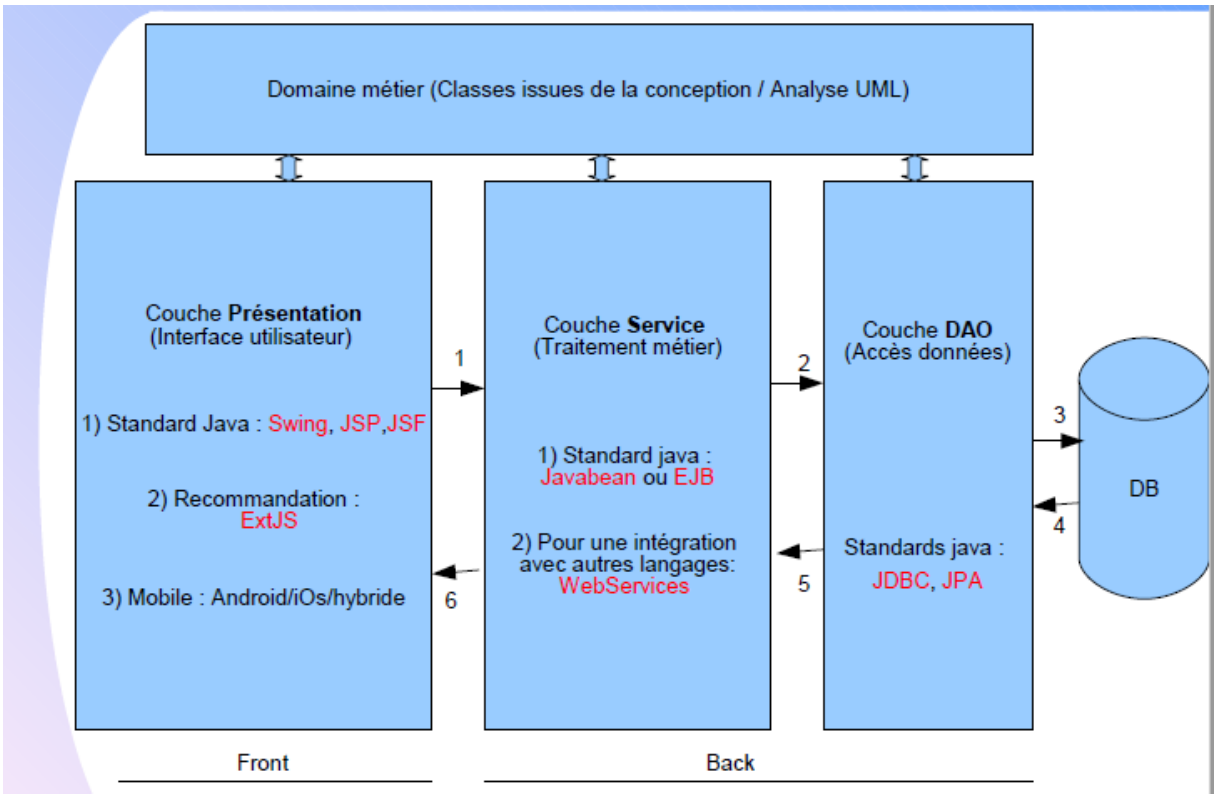

Figure 7: Architecture logicielle cible

2. Tests

Nous avons effectué des tests unitaires sur quelques méthodes métiers de notre application. Les captures suivantes nous permettent d'apprécier les différents résultats.

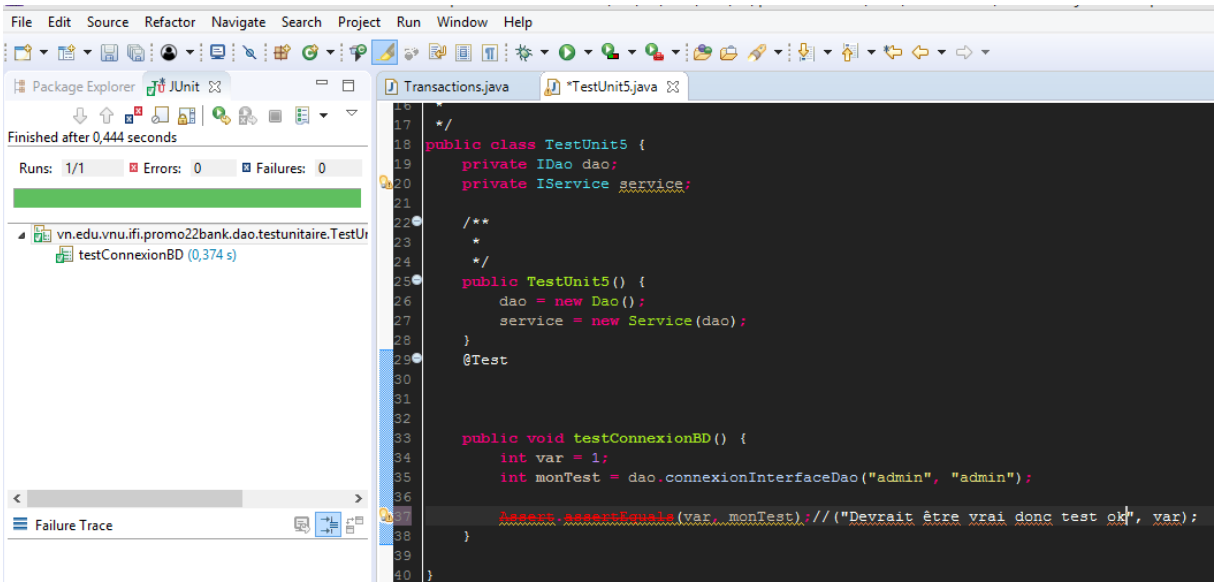

Figure 8: Tests unitaire sur la méthode connectionInterfaceDao(String a, String b)

Cette méthode est censée retourner un entier (1) lorsque la connexion à la BD est réussite. Dans ce test, nous pouvons voir que nous le test est réussi car nous avons pu comparer la valeur entière var avec la valeur de monTest.

Modifions à présent la valeur de var par 3 et comparons-la avec la valeur de monTest.

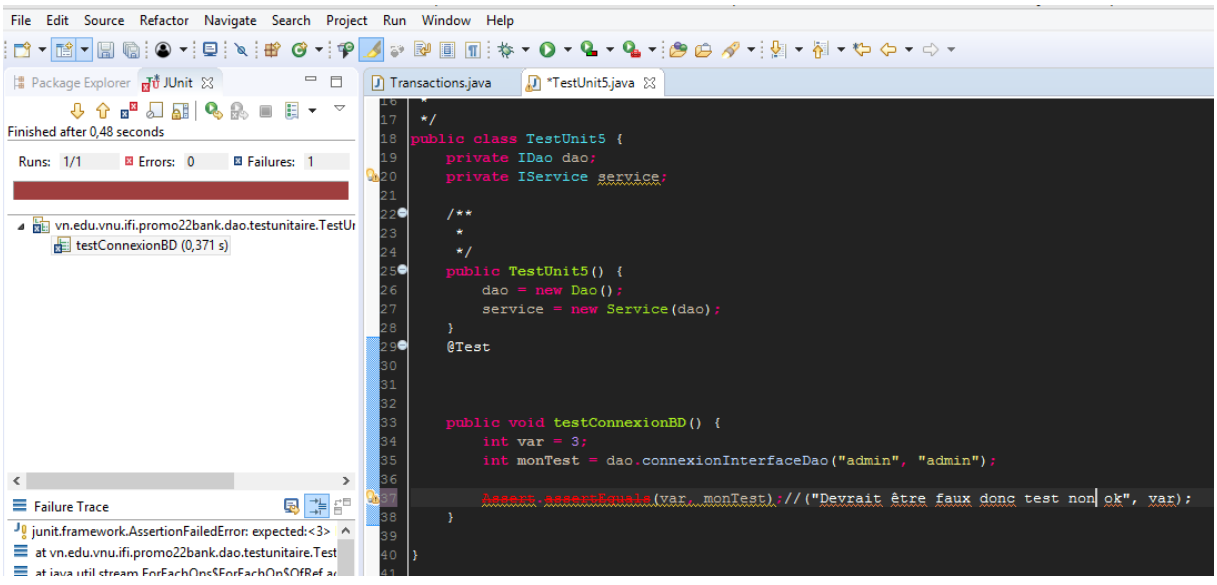

Figure 9: Test unitaire sur la méthode connectionInterfaceDao(String a, String b)

Testons présentement la méthode métier d'enregistrement d'un compte et apprécions les résultats. Cette méthode est censée retourner une valeur entière (1) lorsque le l'enregistrement est un succès et une valeur (0) si échec lors de l'enregistrement.

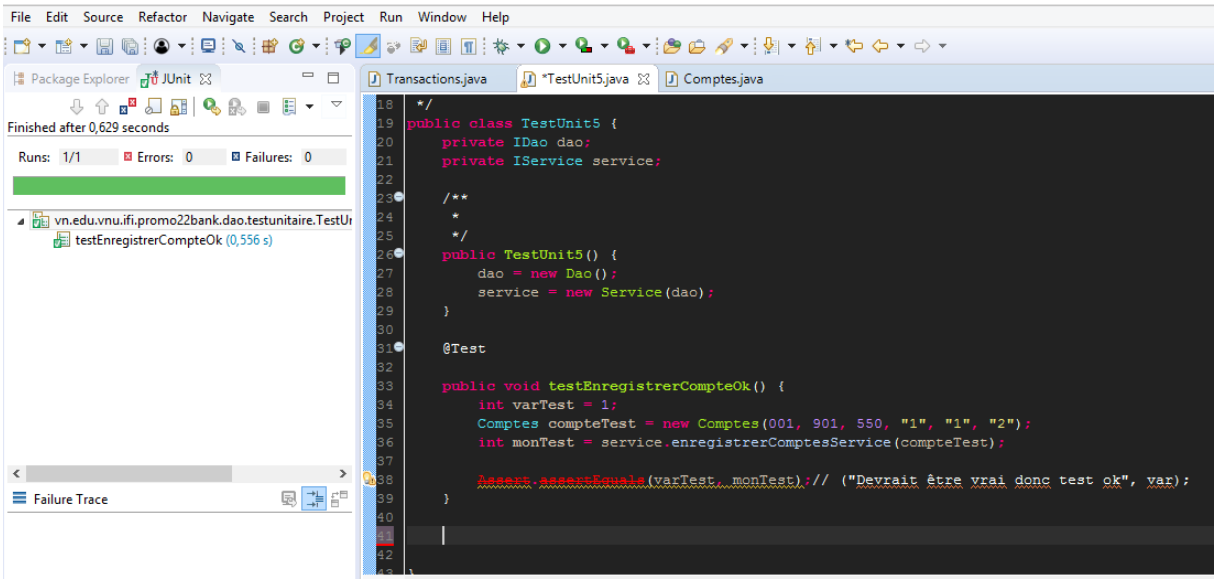

Figure 10: Test unitaire sur la méthode enregistrerComptesService(Comptes compte)

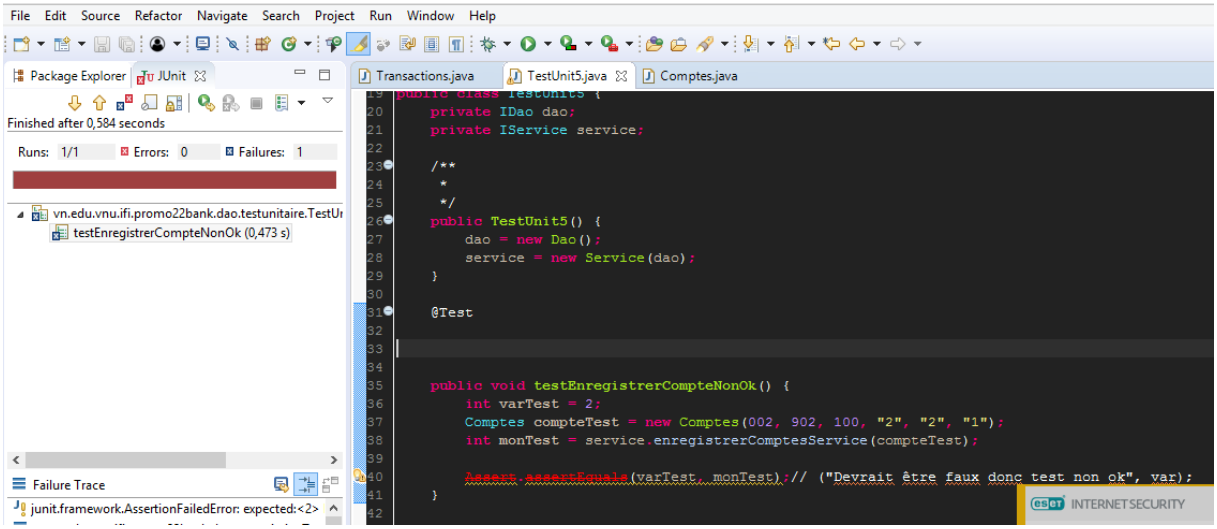

Figure 11: Test unitaire sur la méthode enregistrerComptesService(Comptes compte)

Dans la figure 10 on peut noter que le test est une réussite car la valeur de var = 1 et la méthode retourne 1 lorsqu'il y a succès d'enregistrement ; mais dans la figure 11 on peut s'apercevoir que comme nous nous attendions le test est un échec.

Les différentes captures d'écran suivant permet d'avoir une vue globale de notre application. Nous pouvons y voir la fenêtre d'accueil, la gestion des clients, la gestion des comptes et les transactions bank.

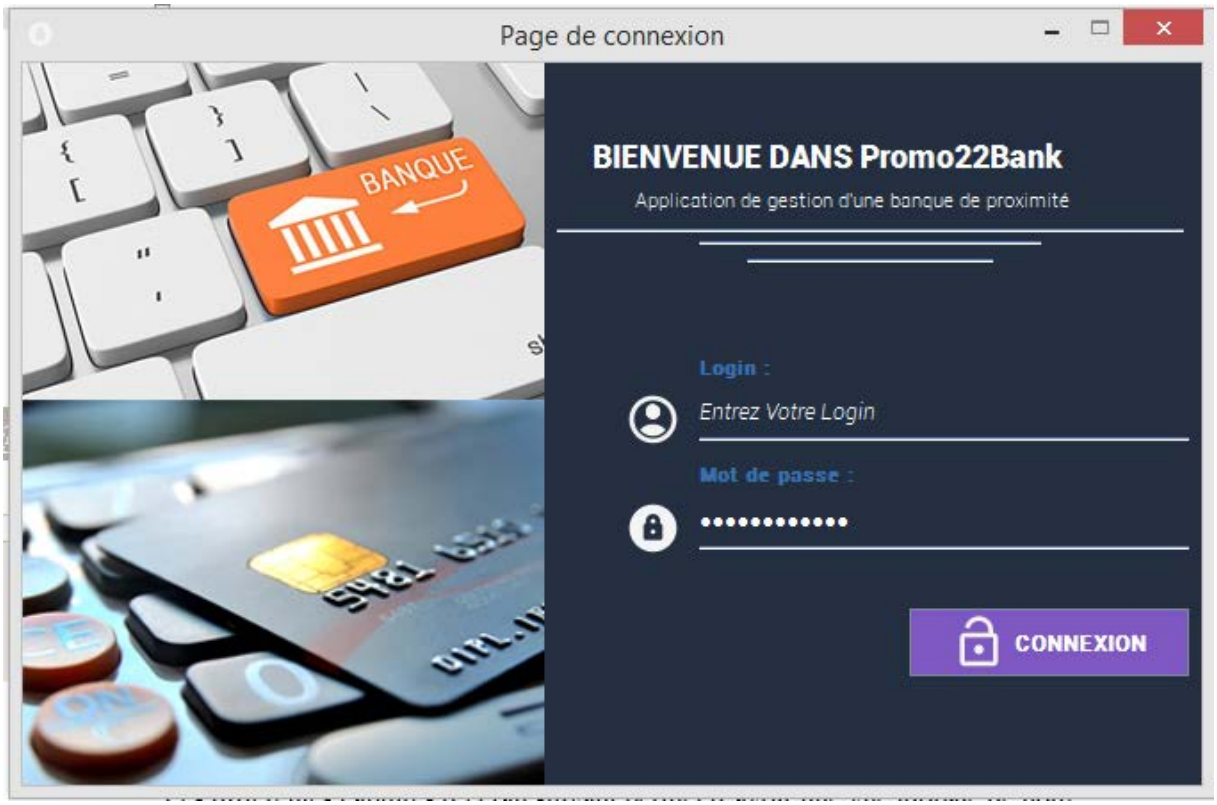

Figure 12: Capture d'écran pour la connexion à l'application

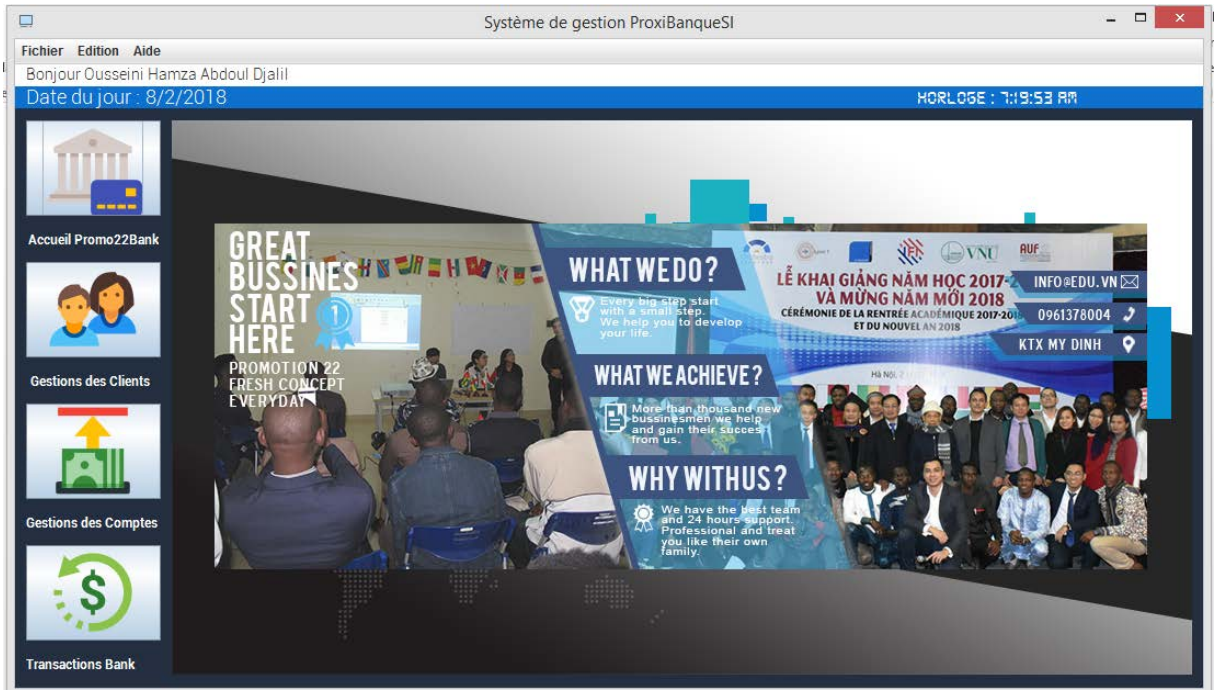

Figure 13: Capture de la page d'accueil de l'application

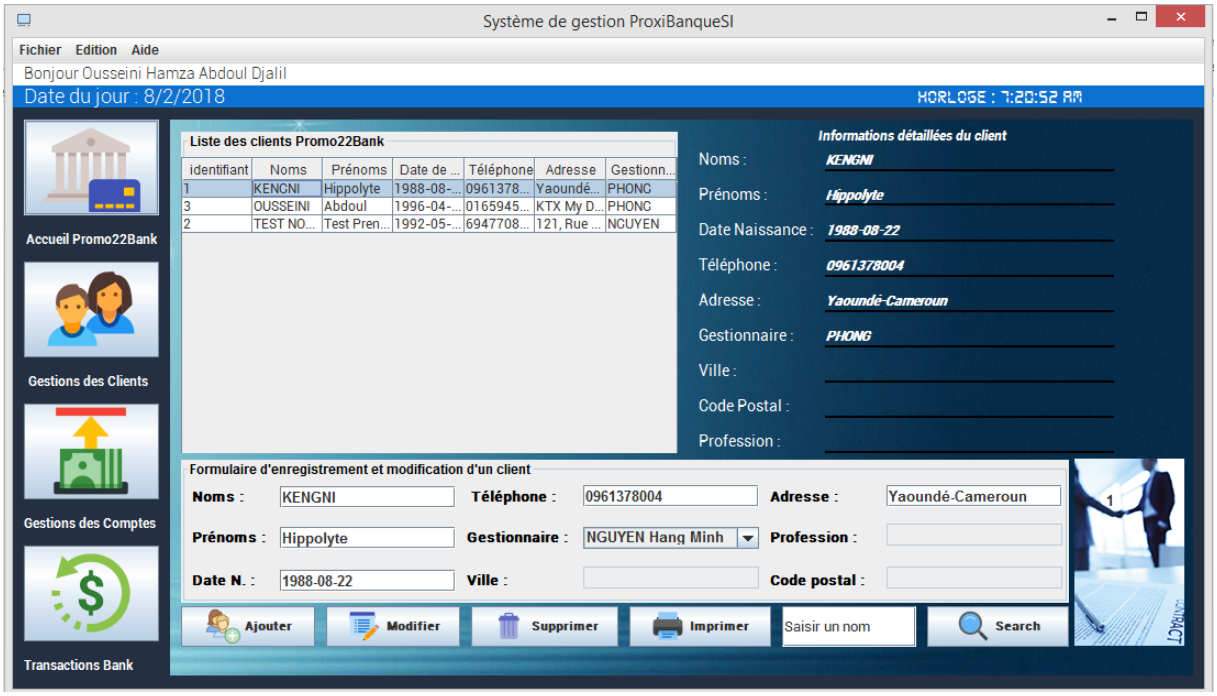

Figure 14: Capture de la fenêtre Gestion des clients

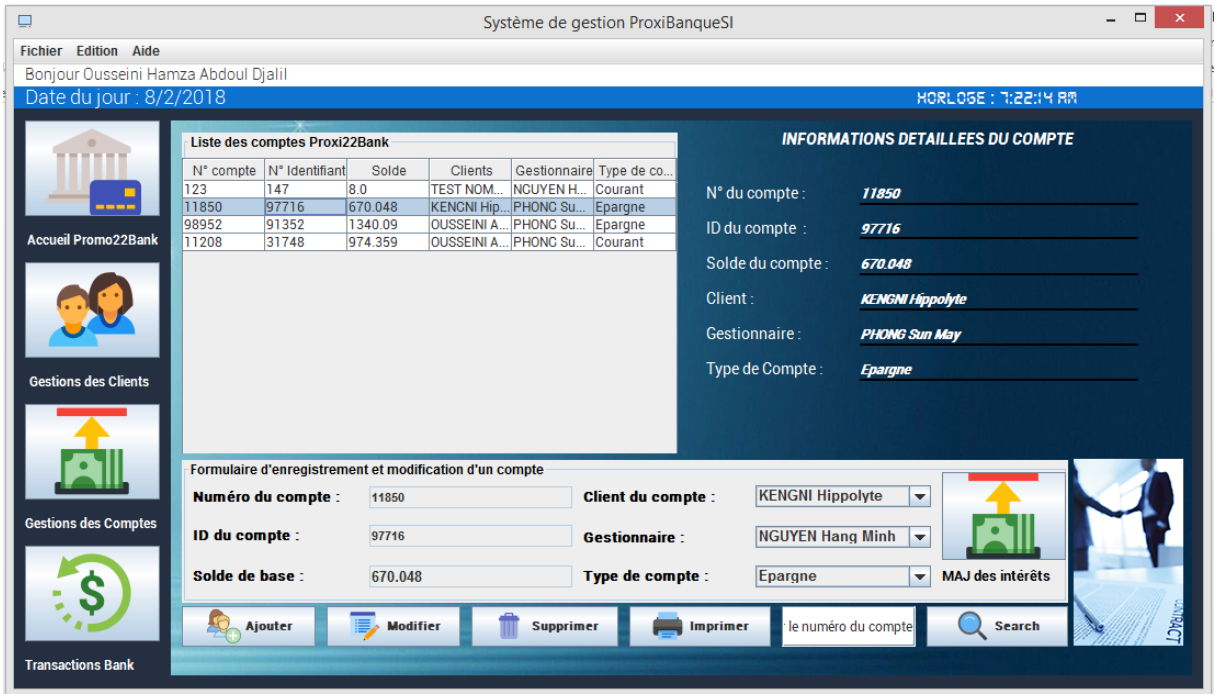

Figure 15: Capture de la fenêtre Gestion des comptes

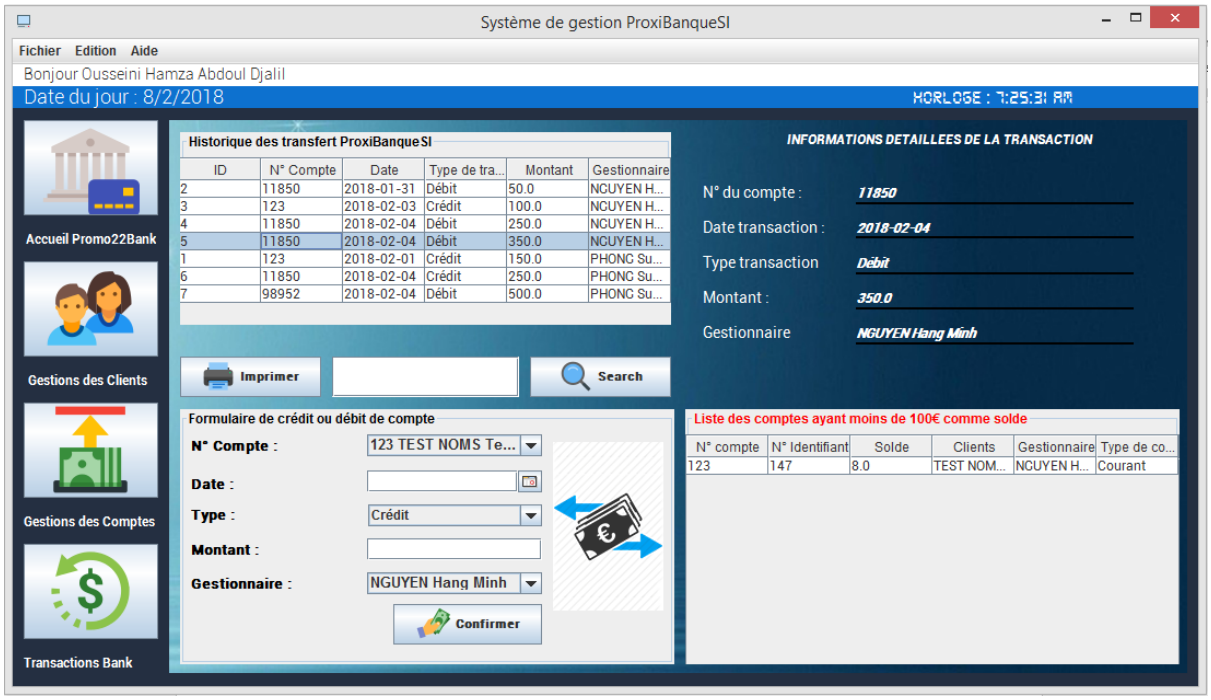

Figure 16: Capture de la fenêtre Transaction Bank (Gestion des transactions)

CONCLUSION

Au terme de notre travail, nous avons pu réaliser une application de gestionnaire de compte bancaire, nous estimons avoir atteint l'objectif qui a été demandé au cours de ce TP1. Nous avons jugé utile de concevoir ce TP 1 avec une interface graphique (à l'aide de SWING) qui interagit avec une base de données. Le travail que nous avons réalisé nous a rappelé les concepts de la programmation orientée-objet avec Java et la modélisation avec UML. Par ailleurs les spécifications énumérées plus haut ont toutes été respectées et notre application est fonctionnelle. Toutefois des efforts restent à fournir dans le sens de la réflexion orientée objet et de la maîtrise du langage java mais aussi de la mise en application des bonnes pratiques de programmation. Nous pouvons donc retenir que ce TP a été un excellent exercice pour l'assimilation des concepts de base du cours de génie logiciel.

ANNEXE

Package : vn.edu.vnu.ifi.promo22bank.domaine

- **Class Clients**

```
/**
 *
 */
package vn.edu.vnu.ifi.promo22bank.domaine;

/**
 * Cette classe contient toutes les informations / attribut d'un client
 * Elle permettra de créer un objet de type Client
 * @version 0.1
 */
public class Clients {

	private String idClients;
	private String noms;
	private String prenoms;
	private String dateNaissance;
	private String telephone;
	private String adresse;
	private String Gestionnaires_idGestionnaires;

	/**
	 * Constructeur par défaut de la classe Clients
	 */
	public Clients() {
		super();
	}


	/**
	 * Constructeur de la classe Clients qui prend en paramètre tous les attributs de celle-ci
	 * @param idClients
	 * @param noms
	 * @param prenoms
	 * @param dateNaissance
	 * @param telephone
	 * @param adresse
	 */
	public Clients(String idClients, String noms, String prenoms, String dateNaissance,
String telephone, String adresse) {
		super();
```

```
		this.idClients = idClients;
		this.noms = noms;
		this.prenoms = prenoms;
		this.dateNaissance = dateNaissance;
		this.telephone = telephone;
		this.adresse = adresse;
	}

	/**
	 * Constructeur de la classe Clients qui prend en paramètre 5 attributs de Clients
	 * @param idClients
	 * @param noms
	 * @param prenoms
	 * @param telephone
	 * @param adresse
	 */
	public Clients(String idClients, String noms, String prenoms, String telephone, String
adresse) {
		super();
		this.idClients = idClients;
		this.noms = noms;
		this.prenoms = prenoms;
		this.telephone = telephone;
		this.adresse = adresse;
	}


	//GETTER AND SETTER


	/**
	 * Cette méthode permet d'obtenir l'id d'un Clients
	 * @return the idClients
	 */
	public String getIdClients() {
		return idClients;
	}


	/**
	 * Cette méthode permet de modifier l'id d'un Clients
	 * @param idClients the idClients to set
	 */
	public void setIdClients(String idClients) {
		this.idClients = idClients;
	}


	/**
	 * Cette métode permet d'obtenir le noms d'un Clients
```

```
 * @return the noms
 */
public String getNoms() {
	return noms;
}


/**
 * Cette méthode permet de modifier le noms d'un Clients
 * @param noms the noms to set
 */
public void setNoms(String noms) {
	this.noms = noms;
}


/**
 * Cette m''thode permet d'obtenir le prenoms d'un Clients
 * @return the prenoms
 */
public String getPrenoms() {
	return prenoms;
}


/**
 * Cette méthode permet de modifier le prenoms d'un Clients
 * @param prenoms the prenoms to set
 */
public void setPrenoms(String prenoms) {
	this.prenoms = prenoms;
}


/**
 * Cette méthode permet d'obtenir la date de naissance d'un Clients
 * @return the dateNaissance
 */
public String getDateNaissance() {
	return dateNaissance;
}


/**
 * Cette méthode permet de modifier la date de naissance d'un Clients
 * @param dateNaissance the dateNaissance to set
 */
public void setDateNaissance(String dateNaissance) {
	this.dateNaissance = dateNaissance;
}
```

```
/**
 * Cette méthode permet d'obtenir le téléphone d'un Clients
 * @return the telephone
 */
public String getTelephone() {
	return telephone;
}


/**
 * Cette méthode permet modifier le téléphone d'un Clients
 * @param telephone the telephone to set
 */
public void setTelephone(String telephone) {
	this.telephone = telephone;
}


/**
 * Cette méthode permet d'obtenir l'adresse d'un Clients
 * @return the adresse
 */
public String getAdresse() {
	return adresse;
}


/**
 * Cette méthode permet de modifier l'adresse d'un Clients
 * @param adresse the adresse to set
 */
public void setAdresse(String adresse) {
	this.adresse = adresse;
}

/**
 * Cette méthode permet d'obtenir l'identifiant d'un Gestionnaire
 * @return the gestionnaires_idGestionnaires
 */
public String getGestionnaires_idGestionnaires() {
	return Gestionnaires_idGestionnaires;
}


/**
 * Cette méthode permet de modifier l'identifiant d'un Gestionnaires
 * @param gestionnaires_idGestionnaires the gestionnaires_idGestionnaires to set
 */
```

```
	public void setGestionnaires_idGestionnaires(String gestionnaires_idGestionnaires) {
		Gestionnaires_idGestionnaires = gestionnaires_idGestionnaires;
	}


	//METHODE toString

	/**
	 * Cette méthode permet de connaitre l'état de l'objet Clients
	 */
	@Override
	public String toString() {
		return  noms + " " + prenoms ;
	}

}
```

- **Class Comptes**

```
/**
 *
 */
package vn.edu.vnu.ifi.promo22bank.domaine;

/**
 * Cette classe contient toutes les informations / attribut d'un comptes
 * Elle permettra de créer un objet de type Comptes
* @version 0.1
 */
public class Comptes {

	private int numComptes;
	private int numIdent;
	private float solde;
	private String TypesComptes_idTypesComptes;
	private String Clients_idClients;
	private String Gestionnaires_idGestionnaires;

	/**
	 * Constructeur par défaut de la classe Comptes
	 */
	public Comptes() {
		super();
	}


	/**
	 * Constructeur de la classe qui prend en paramètre tous les attributs de la classe
	 * @param numComptes
	 * @param numIdent
	 * @param solde
```

```
     * @param typesComptes_idTypesComptes
     * @param clients_idClients
     * @param gestionnaires_idGestionnaires
     */
    public Comptes(int numComptes, int numIdent, float solde, String
typesComptes_idTypesComptes,
                    String clients_idClients, String gestionnaires_idGestionnaires) {
        super();
        this.numComptes = numComptes;
        this.numIdent = numIdent;
        this.solde = solde;
        this.TypesComptes_idTypesComptes = typesComptes_idTypesComptes;
        this.Clients_idClients = clients_idClients;
        this.Gestionnaires_idGestionnaires = gestionnaires_idGestionnaires;
    }


    //GETTER AND SETTERS

    /**
     * Cette méthode permet d'obtenir le numéro de compte
     * @return the numComptes
     */
    public int getNumComptes() {
        return numComptes;
    }

    /**
     * Cette méthode permet de modifier le numéro d'un compte
     * @param numComptes the numComptes to set
     */
    public void setNumComptes(int numComptes) {
        this.numComptes = numComptes;
    }

    /**
     * Cette permet d'obtenir l'identifiant d'un compte
     * @return the numIdent
     */
    public int getNumIdent() {
        return numIdent;
    }

    /**
     * Cette méthode permet de modifier l'identifiant d'un compte
     * @param numIdent the numIdent to set
     */
    public void setNumIdent(int numIdent) {
        this.numIdent = numIdent;
    }
```

```
	/**
	 * Cette méthode permet d'obtenir le solde d'un compte
	 * @return the solde
	 */
	public float getSolde() {
		return solde;
	}

	/**
	 * Cette méthode permet de modifier le solde d'un compte
	 * @param solde the solde to set
	 */
	public void setSolde(float solde) {
		this.solde = solde;
	}

	/**
	 * Cette méthode permet d'obtenir le type d'un compte
	 * @return the typesComptes_idTypesComptes
	 */
	public String getTypesComptes_idTypesComptes() {
		return TypesComptes_idTypesComptes;
	}

	/**
	 * Cette méthode permet de modifier le type d'un compte
	 * @param typesComptes_idTypesComptes the typesComptes_idTypesComptes to set
	 */
	public void setTypesComptes_idTypesComptes(String
typesComptes_idTypesComptes) {
		TypesComptes_idTypesComptes = typesComptes_idTypesComptes;
	}

	/**
	 * Cette méthode permet d'obtenir l'id d'un Clients
	 * @return the clients_idClients
	 */
	public String getClients_idClients() {
		return Clients_idClients;
	}

	/**
	 * Cette méthode permet de modifier l'id d'un Clients
	 * @param clients_idClients the clients_idClients to set
	 */
	public void setClients_idClients(String clients_idClients) {
		Clients_idClients = clients_idClients;
	}
```

```
	/**
	 * Cette méthode permet d'obtenir l'id d'un Gestionnaires
	 * @return the gestionnaires_idGestionnaires
	 */
	public String getGestionnaires_idGestionnaires() {
		return Gestionnaires_idGestionnaires;
	}

	/**
	 * Cette méthode permet de modifier l'id d'un Gestionnaire
	 * @param gestionnaires_idGestionnaires the gestionnaires_idGestionnaires to set
	 */
	public void setGestionnaires_idGestionnaires(String gestionnaires_idGestionnaires) {
		Gestionnaires_idGestionnaires = gestionnaires_idGestionnaires;
	}

	//METHODE toString()

	/**
	 * Cette méthode permet de connaitre l'état d'un objet de type Comptes
	 * @see java.lang.Object#toString()
	 */
	@Override
	public String toString() {
		return numComptes + " " + Clients_idClients ;
	}

}
```

- **Class Transactions**

```
/**
 *
 */
package vn.edu.vnu.ifi.promo22bank.domaine;

/**
 * Cette classe contient toutes les informations / attribut d'une Transactions
 * Elle permettra de créer un objet de type Transactions
 * @version 0.1
 */
public class Transactions {

	private String idTransaction;
	private int compte;
	private String date;
	private String typeTransaction;
	private float montant;
	private String Gestionnaires_idGestionnaires;
```

```
	/**
	 * Constructeur par défaut de la classe Transaction
	 */
	public Transactions() {
		super();
	}

	/**
	 * Constructeur de la classe Transactions prend en paramètre tous les attibuts de la
classe
	 * @param idTransaction
	 * @param date
	 * @param typeTransaction
	 * @param montant
	 * @param gestionnaires_idGestionnaires
	 */
	public Transactions(String idTransaction, String date, String typeTransaction, float
montant,
			String gestionnaires_idGestionnaires) {
		super();
		this.idTransaction = idTransaction;
		this.date = date;
		this.typeTransaction = typeTransaction;
		this.montant = montant;
		this.Gestionnaires_idGestionnaires = gestionnaires_idGestionnaires;
	}

	//GETTER AND SETTERS

	/**
	 * Cette méthode permet d'obtenir l'id d'une transaction
	 * @return the idTransaction
	 */
	public String getIdTransaction() {
		return idTransaction;
	}

	/**
	 * Cette méthode permet de modifier l'id d'une transaction
	 * @param idTransaction the idTransaction to set
	 */
	public void setIdTransaction(String idTransaction) {
		this.idTransaction = idTransaction;
	}

	/**
	 * Cette méthode permet d'obtenir le compte de la transaction
	 * @return the compte
	 */
	public int getCompte() {
```

```
        return compte;
}

/**
 * Cette méthode permet de modifier le compte de la transaction
 * @param compte the compte to set
 */
public void setCompte(int compte) {
        this.compte = compte;
}

/**
 * Cette méthode permet d'obtenir la date d'une transaction
 * @return the date
 */
public String getDate() {
        return date;
}

/**
 * Cette méthode permet de modifier la date d'une transaction
 * @param date the date to set
 */
public void setDate(String date) {
        this.date = date;
}

/**
 * Cette méthode permet d'obtenir la type d'une transaction
 * @return the typeTransaction
 */
public String getTypeTransaction() {
        return typeTransaction;
}

/**
 * Cette méthode permet de modifier le type d'une transaction
 * @param typeTransaction the typeTransaction to set
 */
public void setTypeTransaction(String typeTransaction) {
        this.typeTransaction = typeTransaction;
}

/**
 * Cette méthode permet d'obtenir le montant d'une transaction
 * @return the montant
 */
public float getMontant() {
        return montant;
}
```

```
	/**
	 * Cette méthode permet de modifier le montant d'une transaction
	 * @param montant the montant to set
	 */
	public void setMontant(float montant) {
		this.montant = montant;
	}

	/**
	 * Cette méthode permet d'obtenir l'identifiant d'un gestionnaire
	 * @return the gestionnaires_idGestionnaires
	 */
	public String getGestionnaires_idGestionnaires() {
		return Gestionnaires_idGestionnaires;
	}

	/**
	 * Cette méthode permet de modifier l'identifiant d'un gestionnaire
	 * @param gestionnaires_idGestionnaires the gestionnaires_idGestionnaires to set
	 */
	public void setGestionnaires_idGestionnaires(String gestionnaires_idGestionnaires) {
		Gestionnaires_idGestionnaires = gestionnaires_idGestionnaires;
	}

	//METHODE toString()
	/**
	 * @see java.lang.Object#toString()
	 */
	@Override
	public String toString() {
		return "Transactions [idTransaction=" + idTransaction + ", date=" + date + ",
typeTransaction="
				+ typeTransaction + ", montant=" + montant + ",
Gestionnaires_idGestionnaires="
				+ Gestionnaires_idGestionnaires + "]";
	}

}
```

- **Class Gestionnaires**

```
/**
 * Cette classe contient toutes les informations / attribut d'un gestionnaires
 * Elle permettra de créer un objet de type Gestionnaires
 *@version 0.1
 */
public class Gestionnaires extends Clients {

	private String idGestionnaires;
	private String dateEmbauche;
```

```
	/**
	 * Constructeur par défaut de la classe Gestionnaires
	 */
	public Gestionnaires() {
		super();
	}

	/**
	 * Constructeur de la classe Gestionnaires qui prend en paramètre tous les attributs de la classe
	 * @param idClients
	 * @param noms
	 * @param prenoms
	 * @param telephone
	 * @param adresse
	 * @param idGestionnaires
	 * @param dateEmbauche
	 */
	public Gestionnaires(String idClients, String noms, String prenoms, String telephone, String adresse, String idGestionnaires, String dateEmbauche) {
		super(idClients, noms, prenoms, telephone, adresse);
		this.idGestionnaires = idGestionnaires;
		this.dateEmbauche = dateEmbauche;
	}

	//GETTERS AND SETTERS

	/**
	 * Cette méthode permet d'obtenir l'id d'un Gestionnaire
	 * @return the idGestionnaires
	 */
	public String getIdGestionnaires() {
		return idGestionnaires;
	}

	/**
	 * Cette méthode permet de modifier l'id d'un Gestionnaires
	 * @param idGestionnaires the idGestionnaires to set
	 */
	public void setIdGestionnaires(String idGestionnaires) {
		this.idGestionnaires = idGestionnaires;
	}

	/**
	 * Cette méthode permet d'obtenir la date d'embauche d'un Gestionnaire
	 * @return the dateEmbauche
	 */
	public String getDateEmbauche() {
		return dateEmbauche;
```

```
	}

	/**
	 * Cette méthode permet de modifier la date d'embauche d'un Gestionnaire
	 * @param dateEmbauche the dateEmbauche to set
	 */
	public void setDateEmbauche(String dateEmbauche) {
		this.dateEmbauche = dateEmbauche;
	}

	/**
	 * Cette méthode permet de connaitre l'état de l'objet Gestionnaires
	 * @see java.lang.Object#toString()
	 */
	@Override
	public String toString() {
		return getNoms() + " " + getPrenoms();
	}
```

- **Class TypeComptes**

```
/**
 * Cette classe contient toutes les informations / attribut d'un type de comptes
 * Elle permettra de créer un objet de type TypeComptes
 *@version 0.1
 */
public class TypeComptes {
	private String idTypeComptes;
	private String libelle;
	private int tauxInteret;

	/**
	 * Constructeur par défaut de la class TypeComptes
	 */
	public TypeComptes() {
		super();
	}

	//GETTEURS AND SETTEURS DE LA CLASS

	/**
	 * Cette méthode permet d'obtenir l'idendifiant d'un compte
	 * @return the idTypeComptes
	 */
	public String getIdTypeComptes() {
		return idTypeComptes;
	}

	/**
	 * Cette méthode permet de modifier l'identifiant d'un compte
```

```
 * @param idTypeComptes the idTypeComptes to set
 */
public void setIdTypeComptes(String idTypeComptes) {
	this.idTypeComptes = idTypeComptes;
}

/**
 * Cette méthode permet d'obtenir le libellé d'un compte
 * @return the libelle
 */
public String getLibelle() {
	return libelle;
}

/**
 * Cette méthode permet de modifier le libellé d'un compte
 * @param libelleComptes the libelleComptes to set
 */
public void setLibelle(String libelle) {
	this.libelle = libelle;
}

/**
 * Cette méthode permet d'obtenir un taux d'interet d'un compte
 * @return the tauxInteret
 */
public int getTauxInteret() {
	return tauxInteret;
}

/**
 * Cette méthode permet de modifier le taux d'intérêt d'un compte
 * @param tauxInteret the tauxInteret to set
 */
public void setTauxInteret(int tauxInteret) {
	this.tauxInteret = tauxInteret;
}

/**
 * Cette méthode permet de connaitre l'état de l'objet
 * @see java.lang.Object#toString()
 */
public String toString() {
	return libelle;
}
```

Package: vn.edu.vnu.ifi.promo22bank.service

- **Class Services**

```
/**
 * Cette Class contient toutes les méthodes dont fera appel l'utilisateur
 * @version 0.1 Promo22Bank
 */
/**
 * La class Service implémente toutes les methods de l'interface IService
 */
public class Service implements IService {

	private IDao dao;

	/**
	 * Constructeur par défaut de la classe Service
	 */
	public Service() {

	}

	public Service(IDao dao) {
		super();
		this.dao = dao;
	}

	/**
	 * Cette méthode permet de se connecter à la base de données via la couche dao
	 *
	 * @param login
	 * @param pwd
	 * @return 1 si la connexion est ok et 0 sinon
	 */
	public int connexionService(String login, String pwd) {
		return dao.connexionInterfaceDao(login, pwd);
	}

	/**
	 * Cette méthode permet d'afficher le nom de l'utilisateur connecté via la
	 * couche dao
	 *
	 * @param login
	 * @param pwd
	 * @return le nom et le prénom de l'utilisateur
	 */
	public String afficherMessageService(String recLogin) {
		return dao.afficheMessageDao(recLogin);
	}
```

```
/**
 * Cette méthode permet d'afficher la liste des clients de la base de données
 *
 * @return une liste de clients
 */
public List<Clients> getAllClientsService() {
	return dao.getAllDaoClients();
}

/**
 * Cette méthode permet d'enregistrer un client dans la BD
 *
 * @param client
 * @return 1 si l'enregistrement est ok et 0 si echec lors de l'enregistrement
 */
public int enregistrerClientsService(Clients client) {
	return dao.enregistrerDaoClients(client);
}

/**
 * Cette méthode permet de modifier un client sélectionné en BD
 *
 * @param client
 */
public int modifierClientsService(Clients client) {
	return dao.modifierDaoClients(client);
}

/**
 * Cette méthode permet de rechercher un client en BD
 *
 * @param search
 * @return une liste de client
 */
public List<Clients> rechercherClientsService(String search) {
	return dao.rechercherDaoClients(search);
}

/**
 * Cette méthode permet de supprimer un client sélectionné en BD
 *
 * @param client
 */
public int supprimerClientsService(Clients client) {
	return dao.supprimerDaoClients(client);
}

/**
 * METHODES LIEES AUX GESTIONAIRES
```

```
	 */

	/**
	 * Cette méthode permet de récupérer la liste des Gestionnaires en BD
	 */
	public List<Gestionnaires> getAllGestionnairesService() {
		return dao.getAllGestionnairesDao();
	}

	/**
	 * ICI LES METHODES LIEES AUX COMPTES
	 */

	/**
	 * Méthode qui permet de générer un ensemble de 05 chiffres pour le numéro de
	 * compte d'un client
	 *
	 * @return numero les chiffres générés
	 */
	public int genererNumeroCompte() {
		int valeurGeneree = 0;
		String CARACTERES = "1234567890";
		StringBuilder salt = new StringBuilder();
		Random rnd = new Random();
		while (salt.length() < 5) { // taille de la valeur à retourner
			int index = (int) (rnd.nextFloat() * CARACTERES.length());
			salt.append(CARACTERES.charAt(index));
		}
		String saltStr = salt.toString();
		try {
			valeurGeneree = Integer.parseInt(saltStr);
		} catch (NumberFormatException e) {
			JOptionPane.showMessageDialog(null, saltStr + " n'est pas un int",
"Attention!",
					JOptionPane.WARNING_MESSAGE);
		}
		return valeurGeneree;

	}

	/**
	 * Méthode qui permet de générer un ensemble de 05 chiffres pour le numéro de
	 * compte d'un client
	 *
	 * @return numero les chiffres générés
	 */
	public int genererIdentCompte() {
		int valeurIdent = 0;
		String CARACTERES = "1234567890";
		StringBuilder salt = new StringBuilder();
```

```
		Random rnd = new Random();
		while (salt.length() < 5) { // taille de la valeur à retourner
			int index = (int) (rnd.nextFloat() * CARACTERES.length());
			salt.append(CARACTERES.charAt(index));
		}
		String saltStr = salt.toString();

		try {
			valeurIdent = Integer.parseInt(saltStr);
		} catch (NumberFormatException e) {
			JOptionPane.showMessageDialog(null, saltStr + " n'est pas un int",
"Attention!",
					JOptionPane.WARNING_MESSAGE);
		}

		return valeurIdent;

	}

	/**
	 * Cette méthode permet d'afficher la liste des type de comptes en BD
	 *
	 * @return une liste de type de compte
	 */
	public List<TypeComptes> getAllTypeComptesService() {
		return dao.getAllDaoTypeComptes();
	}

	/**
	 * Cette méthode permet d'afficher la liste des comptes en BD
	 *
	 * @return une liste de compte
	 */
	public List<Comptes> getAllComptesService() {
		return dao.getAllDaoComptes();
	}

	/**
	 * Cette méthode permet d'enregistrer un compte en BD
	 *
	 * @param compte
	 * @return 1 si l'enregistre est un succès 0 si échec
	 */
	public int enregistrerComptesService(Comptes compte) {
		return dao.enregistrerDaoComptes(compte);
	}

	/**
	 * Cette méthode permet de modifier un compte en BD
	 *
```

```
	 * @param compte
	 * @return 1 si modification succès 0 si échec
	 */
	public int modifierComptesServices(Comptes compte) {
		return dao.modifierDaoCompte(compte);
	}

	/**
	 * Cette méthode permet de supprimer le compte d'un client
	 *
	 * @param compte
	 * @return 0 si échec lors de la suppression 1 si succès
	 */
	public int supprimerComptesService(Comptes compte) {
		return dao.supprimerDaoCompte(compte);
	}

	/**
	 * Cette méthode permet de rechercher un compte en BD
	 *
	 * @param search
	 * @return une liste de compte
	 */
	public List<Comptes> rechercherComptesService(String search) {
		return dao.rechercherDaoComptes(search);
	}

	/**
	 * Cette méthode permet de calculer et mettre à jour le solde de tous les comptes enn
BD
	 * @see
vn.edu.vnu.ifi.promo22bank.service.IService#calculInteretComptesService(int, float)
	 */
	public int calculInteretComptesService(int numCompte, float interet) {
		return dao.calculInteretDaoComptes(numCompte, interet);
	}

	/**
	 * ICI LES METHODES LIEES AUX TRANSACTIONS
	 */

	/**
	 * Cette méthode permet d'afficher la liste des comptes en seuil en BD
	 *
	 * @return une liste de compte seuil
	 */
	public List<Comptes> getAllComptesSeuilService() {
		return dao.getAllDaoComptesSeuil();
	}
```

```
/**
 * Cette méthode permet de récupérer le solde d'un compte en BD
 *
 * @return le solde du compte donné en paramètre.
 */
public float getSoldeComptesService(int compte) {
	return dao.afficherSoldeDao(compte);
}

/**
 * Cette méthode permet d'afficher la liste des transactions
 *
 * @return une liste de transactions
 */
public List<Transactions> getAllTransactionsService() {
	return dao.getAllDaoTransactions();
}

/**
 * Cette méthode permet d'enregistrer une transaction en BD
 *
 * @param transaction
 * @return 1 si l'enregistre est un succès 0 si échec
 */
public int enregistrerTransactionsService(Transactions transaction) {
	return dao.enregistrerDaoTransactions(transaction);
}

/**
 * Cette méthode permet de faire le débit du solde d'un compte dans la BD
 *
 * @param transaction
 * @return 1 si la mise à jour est un succès 0 si échec
 */
public int faireCreditTransactionsService(float montantCredit, int numCompte) {
	return dao.faireCreditDaoTransactions(montantCredit, numCompte);
}

/**
 * Cette méthode permet de faire le débit du solde d'un compte dans la BD
 *
 * @param transaction
 * @return 1 si la mise à jour est un succès 0 si échec
 */
public int faireDebitTransactionsService(float montantCredit, int numCompte) {
	return dao.faireDebitDaoTransactions(montantCredit, numCompte);
}

/**
 * Cette méthode permet de rechercher un compte seuil en BD
```

```
	 *
	 * @param search
	 * @return une liste de compte seuil
	 */
	public List<Transactions> rechercherTransactionsService(String search) {
		return dao.rechercherDaoTransactions(search);
	}
```

Package: vn.edu.vnu.ifi.promo22bank.dao

- **Class Dao**

```
/**
 * Cette Class implémente toutes les signatures des méthodes de l'interface IDao
 *
 * @version 0.1 Promo22Bank
 */
public class Dao implements IDao {
	// Déclaration des variables que nous allons manipuler

	/**
	 * Constructeur de ma classe ne prend aucun paramétre
	 */
	public Dao() {
		super();
	}

	/**
	 * Cette méthode permet de se connecter à la base de données afin d'avoir accès
	 * à l'application
	 *
	 * @param login
	 * @param pwd
	 * @return 1 si la connexion est ok et 0 sinon
	 */
	public int connexionInterfaceDao(String login, String pwd) {

		try {

			Connection connection = ConnectionMYSQL.getInstance();

			Statement st = connection.createStatement();

			String requete = "SELECT * FROM connexion WHERE ( (login = '" +
login + "') AND (password = '" + pwd
					+ "') )";

			ResultSet rs = st.executeQuery(requete);

			if (rs.next()) {
```

```
                return 1;

            } else {

                return 0;
            }

        } catch (Exception e) {
            System.out.println("--> Exception : " + e);
        }
        return 1;
    }

    /**
     * Cette méthode permet d'afficher un message à la connection de l'utilisateur
     * sur la page d'accueil
     *
     * @return message
     */
    public String afficheMessageDao(String recLogin) {

        // Cette varaible contiendra le résultat de la requete de sélection
        String message = null;

        try {

            Connection connection = ConnectionMYSQL.getInstance();

            Statement st = connection.createStatement();

            String requete = "SELECT * FROM connexion WHERE login = '" +
recLogin + "'";

            ResultSet rs = st.executeQuery(requete);

            if (rs.next()) {

                String t1 = rs.getString("nomComplet");

                message = t1;
                // lnom.setText(t1 + " " + t2);

                // type.setText(t3);
            }

        } catch (Exception e) {
            System.out.println("--> Exception : " + e);
        }
```

```
		return message;
	}

	/**
	 * ICI TOUTES LES METHODES LIEES AUX CLIENTS
	 */

	/**
	 * Cette méthode permet d'afficher la liste de tous les client en BD
	 *
	 * @return List<Clients> maListeClients
	 */
	public List<Clients> getAllDaoClients() {
		// création d'une collection de Clients
		List<Clients> maListeClients = new ArrayList<Clients>();

		try {

			// Etape 1 : récupération de la connexion
			Connection cn = ConnectionMYSQL.getInstance();

			// Etape 2 : préparation requête
			Statement st = cn.createStatement();

			String sql = "SELECT clients.idClients, clients.noms, clients.prenoms,
clients.dateNaissance, clients.telephone, clients.adresse, gestionnaires.noms AS
Gestionnaires_idGestionnaires FROM clients INNER JOIN gestionnaires ON
clients.Gestionnaires_idGestionnaires = gestionnaires.idGestionnaires ORDER BY
clients.noms";

			// Etape 3 : exécution requéte
			ResultSet rs = st.executeQuery(sql);

			// Etape 4 :parcours Resultset (optionnel)
			while (rs.next()) {
				Clients client = new Clients();
				client.setIdClients(rs.getString("idClients"));
				client.setNoms(rs.getString("noms"));
				client.setPrenoms(rs.getString("prenoms"));
				client.setDateNaissance(rs.getString("dateNaissance"));
				client.setTelephone(rs.getString("telephone"));
				client.setAdresse(rs.getString("adresse"));

	client.setGestionnaires_idGestionnaires(rs.getString("Gestionnaires_idGestionnaires")
);
				maListeClients.add(client);
			}

			// Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
```

```
		} catch (SQLException e) {
			e.printStackTrace();

		}

		return maListeClients;
	}

	/**
	 * Cette méthode permet d'enregistrer un client en BD
	 *
	 * @return 0 si échec et 1 si succès
	 * @param clients
	 */
	public int enregistrerDaoClients(Clients client) {
		try {

			// Etape 1 : récupération de la connexion
			Connection cn = ConnectionMYSQL.getInstance();

			// Etape 2 : Prépare requête
			String sql = "INSERT INTO `clients`
(`noms`,`prenoms`,`dateNaissance`,`telephone`,`adresse`,`Gestionnaires_idGestionnaires`) "
					+ "VALUES(?,?,?,?,?,?)";

			// Etape 3 : Création d'un preparedStatement
			PreparedStatement ps = cn.prepareStatement(sql);

			ps.setString(1, client.getNoms());
			ps.setString(2, client.getPrenoms());
			ps.setString(3, client.getDateNaissance());
			ps.setString(4, client.getTelephone());
			ps.setString(5, client.getAdresse());
			ps.setString(6, client.getGestionnaires_idGestionnaires());

			// Etape 4 : exécution requête
			ps.executeUpdate();

			// Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
		} catch (SQLException e) {
			e.printStackTrace();
			return 0; // retourne "0" lorsqu'il ya une ereur
		}
		return 1; // retourne "1" lorsque l'enregistrement en BD est réussi
	}

	/**
	 * Cette méthode permet de supprimer un client sélectionné en BD
	 *
```

```
	 * @return 0 si échec et 1 si succès
	 * @param clients
	 */
	public int supprimerDaoClients(Clients client) {
		try {

			// Etape 1 : récupération de la connexion
			Connection cn = ConnectionMYSQL.getInstance();

			// Etape 2 : Prépare requête
			String reqDel = "DELETE from clients WHERE idClients = ?";

			// Etape 3 : Création d'un preparedStatement
			PreparedStatement ps = cn.prepareStatement(reqDel);

			ps.setString(1, client.getIdClients());

			// Etape 4 : exécution requête
			ps.executeUpdate();

			// Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
		} catch (SQLException e) {
			e.printStackTrace();
			return 0; // retourne "0" lorsqu'il ya une ereur
		}
		return 1; // retourne "1" lorsque la suppression en BD est réussi
	}

	/**
	 * Cette méthode permet de modifier un patient sélectionné en BD
	 *
	 * @param clients
	 * @return 0 si échec et 1 si succès
	 */
	public int modifierDaoClients(Clients client) {

		try {

			// Etape 1 : récupération de la connexion
			Connection cn = ConnectionMYSQL.getInstance();

			// Etape 2 : Prépare requête
			String reqMod = "UPDATE `clients` SET
`noms`= ?,`prenoms`= ?,`dateNaissance`= ?,`telephone`= ?,`adresse`= ?,`Gestionnaires_idGes
tionnaires`= ? WHERE idClients ='"
					+ client.getIdClients() + "'";

			// Etape 3 : Création d'un preparedStatement
			PreparedStatement ps = cn.prepareStatement(reqMod);
```

```
                        ps.setString(1, client.getNoms());
                        ps.setString(2, client.getPrenoms());
                        ps.setString(3, client.getDateNaissance());
                        ps.setString(4, client.getTelephone());
                        ps.setString(5, client.getAdresse());
                        ps.setString(6, client.getGestionnaires_idGestionnaires());

                        // Etape 4 : exécution requête
                        ps.executeUpdate();

                        // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
                } catch (SQLException e) {
                        e.printStackTrace();
                        return 0; // retourne "0" lorsqu'il ya une ereur
                }
                return 1; // retourne "1" lorsque la modification en BD est réussi
        }

        /**
         * Cette méthode permet de rechercher un client en BD
         *
         * @param search
         * @return List<Clients> maListeClients
         */
        public List<Clients> rechercherDaoClients(String search) {
                // création d'une collection de patient
                List<Clients> maListeClientsTrouver = new ArrayList<Clients>();

                try {

                        // Etape 1 : récupération de la connexion
                        Connection cn = ConnectionMYSQL.getInstance();

                        // Etape 2 : préparation requête
                        Statement st = cn.createStatement();

                        String reqSearch = "SELECT clients.idClients, clients.noms,
clients.prenoms, clients.dateNaissance, clients.telephone, clients.adresse, gestionnaires.noms
AS Gestionnaires_idGestionnaires FROM clients INNER JOIN gestionnaires ON
clients.Gestionnaires_idGestionnaires = gestionnaires.idGestionnaires WHERE clients.noms
= '"
                                        + search + "' ORDER BY clients.noms";

                        // Etape 3 : exécution requéte
                        ResultSet rs = st.executeQuery(reqSearch);

                        // Etape 4 :parcours Resultset (optionnel)
                        while (rs.next()) {
```

```
                                Clients client = new Clients();
                                client.setIdClients(rs.getString("idClients"));
                                client.setNoms(rs.getString("noms"));
                                client.setPrenoms(rs.getString("prenoms"));
                                client.setDateNaissance(rs.getString("dateNaissance"));
                                client.setTelephone(rs.getString("telephone"));
                                client.setAdresse(rs.getString("adresse"));

        client.setGestionnaires_idGestionnaires(rs.getString("Gestionnaires_idGestionnaires")
);
                                maListeClientsTrouver.add(client);

                        }

                        // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
                } catch (SQLException e) {
                        e.printStackTrace();

                }
                return maListeClientsTrouver; // retourne une liste lorsque la recherche en BD
a réussi
        }

        /**
         * METHODES LIEES AUX GESTIONNAIRES
         */

        /**
         * @return List<Gestionnaires> maListeGestionnaires
         */
        @Override
        public List<Gestionnaires> getAllGestionnairesDao() {
                // création d'une collection de Gestionnaires
                List<Gestionnaires> maListeGestionnaires = new ArrayList<Gestionnaires>();

                try {

                        // Etape 1 : récupération de la connexion
                        Connection cn = ConnectionMYSQL.getInstance();

                        // Etape 2 : préparation requête
                        Statement st = cn.createStatement();

                        String sql = "SELECT * FROM gestionnaires ";

                        // Etape 3 : exécution requéte
                        ResultSet rs = st.executeQuery(sql);

                        // Etape 4 :parcours Resultset (optionnel)
```

```
                while (rs.next()) {
                        Gestionnaires gestionnaire = new Gestionnaires();
                        gestionnaire.setIdGestionnaires(rs.getString("idGestionnaires"));
                        gestionnaire.setNoms(rs.getString("noms"));
                        gestionnaire.setPrenoms(rs.getString("prenoms"));
                        gestionnaire.setDateEmbauche(rs.getString("dateEmbauche"));
                        gestionnaire.setTelephone(rs.getString("telephone"));
                        gestionnaire.setAdresse(rs.getString("adresse"));
                        maListeGestionnaires.add(gestionnaire);
                }

                // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
        } catch (SQLException e) {
                e.printStackTrace();

        }

        return maListeGestionnaires;
    }

    /**
     * ICI TOUTES LES METHODES LIEES AUX TRANSACTIONS
     */

    /**
     * Cette méthode permet de récupérer tous les types de compte en BD
     *
     * @return maListeTypeComptes
     */
    public List<TypeComptes> getAllDaoTypeComptes() {
        // création d'une collection de Clients
        List<TypeComptes> maListeTypeComptes = new ArrayList<TypeComptes>();
        try {

            // Etape 1 : récupération de la connexion
            Connection cn = ConnectionMYSQL.getInstance();

            // Etape 2 : préparation requête
            Statement st = cn.createStatement();

            String sql = "SELECT * FROM typecomptes ";

            // Etape 3 : exécution requéte
            ResultSet rs = st.executeQuery(sql);

            // Etape 4 :parcours Resultset (optionnel)
            while (rs.next()) {
                TypeComptes typeComptes = new TypeComptes();
```

```
typeComptes.setIdTypeComptes(rs.getString("idTypeComptes"));
                        typeComptes.setLibelle(rs.getString("libelle"));
                        typeComptes.setTauxInteret(rs.getInt("tauxInteret"));
                        maListeTypeComptes.add(typeComptes);
                }

                // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
        } catch (SQLException e) {
                e.printStackTrace();

        }
        return maListeTypeComptes;
}

/**
 * Cette méthode permet de récupérer tous les comptes de la BD dont le solde est
 * inférieur à 500€
 *
 * @return List<Comptes> maListeComptesSeuil
 */
public List<Comptes> getAllDaoComptesSeuil() {
        // création d'une collection de Clients
        List<Comptes> maListeComptesSeuil = new ArrayList<Comptes>();

        try {

                // Etape 1 : récupération de la connexion
                Connection cn = ConnectionMYSQL.getInstance();

                // Etape 2 : préparation requête
                Statement st = cn.createStatement();
                String espace = " ";

                String sql = "SELECT numComptes, numIdent, solde,
CONCAT(clients.noms,'"+ espace +"',clients.prenoms) AS Clients_idClients,
CONCAT(gestionnaires.noms,'" + espace + "',gestionnaires.prenoms) AS
Gestionnaires_idGestionnaires, typeComptes.libelle AS TypeComptes_idTypeComptes
FROM comptes, clients, gestionnaires, typeComptes WHERE comptes.Clients_idClients =
clients.idClients AND clients.Gestionnaires_idGestionnaires = gestionnaires.idGestionnaires
AND comptes.TypeComptes_idTypeComptes = typeComptes.idTypeComptes AND solde <
500";

                // Etape 3 : exécution requéte
                ResultSet rs = st.executeQuery(sql);

                // Etape 4 :parcours Resultset (optionnel)
                while (rs.next()) {
                        Comptes compte = new Comptes();
```

```
                                compte.setNumComptes(rs.getInt("numComptes"));
                                compte.setNumIdent(rs.getInt("numIdent"));
                                compte.setSolde(rs.getFloat("solde"));
                                compte.setClients_idClients(rs.getString("Clients_idClients"));

        compte.setGestionnaires_idGestionnaires(rs.getString("Gestionnaires_idGestionnaires
"));

        compte.setTypesComptes_idTypesComptes(rs.getString("TypeComptes_idTypeCompt
es"));
                                maListeComptesSeuil.add(compte);
                        }

                        // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
                } catch (SQLException e) {
                        e.printStackTrace();

                }

                return maListeComptesSeuil;
        }

        /**
         * Cette méthode permet de récupérer toutes les transactions de la BD
         *
         * @return List<Transactions> maListeTransactions
         */
        public List<Transactions> getAllDaoTransactions() {
                // création d'une collection de transactions
                List<Transactions> maListeTransactions = new ArrayList<Transactions>();
                String espace = " ";

                try {

                        // Etape 1 : récupération de la connexion
                        Connection cn = ConnectionMYSQL.getInstance();

                        // Etape 2 : préparation requête
                        Statement st = cn.createStatement();

                        String sql = "SELECT transactions.`idTransactions`,
transactions.`numComptes`, transactions.`date`, transactions.`typeTransaction`,
transactions.`montant`, CONCAT(gestionnaires.`noms`,'"+ espace
+"',gestionnaires.`prenoms`) AS Gestionnaires_idGestionnaires FROM transactions,
gestionnaires WHERE gestionnaires.`idGestionnaires` = Gestionnaires_idGestionnaires ";

                        // Etape 3 : exécution requéte
                        ResultSet rs = st.executeQuery(sql);
```

```
                        // Etape 4 :parcours Resultset (optionnel)
                        while (rs.next()) {
                                Transactions transaction = new Transactions();
                                transaction.setIdTransaction(rs.getString("idTransactions"));
                                transaction.setCompte(rs.getInt("numComptes"));
                                transaction.setDate(rs.getString("date"));
                                transaction.setTypeTransaction(rs.getString("typeTransaction"));
                                transaction.setMontant(rs.getFloat("montant"));

        transaction.setGestionnaires_idGestionnaires(rs.getString("Gestionnaires_idGestionnai
res"));
                                maListeTransactions.add(transaction );
                        }

                        // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
                } catch (SQLException e) {
                        e.printStackTrace();

                }

                return maListeTransactions;
        }

        /**
         * Cette méthode permet d'enregistre une opération dans la BD
         *
         * @return 0 si échec et 1 si succès lors de l'enregistrement
         * @param transaction
         */
        public int enregistrerDaoTransactions(Transactions transaction) {
                try {
                        // Etape 1 : récupération de la connexion
                        Connection cn = ConnectionMYSQL.getInstance();

                        // Etape 2 : Prépare requête
                        String sql = "INSERT INTO `transactions`
(`numComptes`,`date`,`typeTransaction`,`montant`,`Gestionnaires_idGestionnaires`)
VALUES (?, ?, ?, ?, ?)";

                        // Etape 3 : Création d'un preparedStatement
                        PreparedStatement ps = cn.prepareStatement(sql);

                        ps.setInt(1, transaction.getCompte());
                        ps.setString(2, transaction.getDate());
                        ps.setString(3, transaction.getTypeTransaction());
                        ps.setFloat(4, transaction.getMontant());
                        ps.setString(5, transaction.getGestionnaires_idGestionnaires());

                        // Etape 4 : exécution requête
```

```
                        ps.executeUpdate();

                        // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
                } catch (SQLException e) {
                        e.printStackTrace();
                        return 0; // retourne "0" lorsqu'il ya une ereur
                }
                return 1; // retourne "1" lorsque l'enregistrement en BD est réussi
        }

        /**
         * Cette méthode permet de faire le débit du solde d'un compte dans la BD
         *
         * @return 0 si échec et 1 si succès lors de la modification
         * @param montantCredit, numCompte
         */
        public int faireDebitDaoTransactions(float montantCredit, int numCompte) {
                try {
                        // Etape 1 : récupération de la connexion
                        Connection cn = ConnectionMYSQL.getInstance();

                        // Etape 2 : Prépare requête
                        String sql = "UPDATE `comptes` SET `solde` = ('" + montantCredit +
"') WHERE numComptes = '" + numCompte + "'";

                        // Etape 3 : Création d'un preparedStatement
                        PreparedStatement ps = cn.prepareStatement(sql);

//                      ps.setInt(1, transaction.getCompte());
//                      ps.setString(2, transaction.getDate());
//                      ps.setString(3, transaction.getTypeTransaction());
//                      ps.setFloat(4, transaction.getMontant());
//                      ps.setString(5, transaction.getGestionnaires_idGestionnaires());

                        // Etape 4 : exécution requête
                        ps.executeUpdate();

                        // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
                } catch (SQLException e) {
                        e.printStackTrace();
                        return 0; // retourne "0" lorsqu'il ya une ereur
                }
                return 1; // retourne "1" lorsque l'opération en BD est réussi
        }

        /**
         * Cette méthode permet de faire le crédit du solde d'un compte dans la BD
         *
```

```
	 * @return 0 si échec et 1 si succès lors de la maj
	 * @param montantCredit, numCompte
	 */
	public int faireCreditDaoTransactions(float montantCredit, int numCompte) {
		try {
			// Etape 1 : récupération de la connexion
			Connection cn = ConnectionMYSQL.getInstance();
			//String reqMod = "UPDATE comptes SET solde = ?, dateOpen = ?,
Clients_idClients = ?, typeComptes_idTypeComptes = ?  WHERE  numeroCpte ='" +
compte.getNumeroCpte() + "'";
			// Etape 2 : Prépare requête
			String sql = "UPDATE `comptes` SET `solde` = ('" + montantCredit +
"') WHERE numComptes = '" + numCompte + "'";

			// Etape 3 : Création d'un preparedStatement
			PreparedStatement ps = cn.prepareStatement(sql);
//			Comptes cptes = new Comptes();
//			ps.setInt(1, cptes.getNumComptes());
//			ps.setInt(2, cptes.getNumIdent());
//			ps.setFloat(3, cptes.getSolde());
//			ps.setString(4, cptes.getClients_idClients());
//			ps.setString(5, cptes.getGestionnaires_idGestionnaires());
//			ps.setString(6, cptes.getTypesComptes_idTypesComptes());
//			//ps.setString(2, transaction.getDate());
//			//ps.setString(3, transaction.getTypeTransaction());
//			//ps.setFloat(4, transaction.getMontant());
//			//ps.setString(5, transaction.getGestionnaires_idGestionnaires());

			// Etape 4 : exécution requête
			ps.executeUpdate();

			// Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
		} catch (SQLException e) {
			e.printStackTrace();
			return 0; // retourne "0" lorsqu'il ya une ereur
		}
		return 1; // retourne "1" lorsque l'operation en BD est réussi
	}

	/**
	 * Cette méthode permet d'afficher le solde d'un compte en BD
	 *
	 * @param compte
	 * @return soldeCpte
	 */
	public float afficherSoldeDao(int compte) {

		// Cette varaible contiendra le résultat de la requete de sélection
		float soldeCpte = 0;
```

```
		try {

			Connection connection = ConnectionMYSQL.getInstance();

			Statement st = connection.createStatement();

			String requete = "SELECT solde FROM comptes WHERE
numComptes = '" + compte + "'";

			ResultSet rs = st.executeQuery(requete);

			if (rs.next()) {

				float t1 = rs.getFloat("solde");

				soldeCpte = t1;

			}

		} catch (Exception e) {
			System.out.println("--> Exception : " + e);
		}

		return soldeCpte;
	}


	/**
	 * ICI TOUTES LES METHODES LIEES AUX COMPTES
	 */

	/**
	 * Cette méthode permet de récupérer tous les comptes de la BD
	 *
	 * @return List<Comptes> maListeCompte
	 */
	public List<Comptes> getAllDaoComptes() {
		// création d'une collection de Clients
		List<Comptes> maListeComptes = new ArrayList<Comptes>();

		try {

			// Etape 1 : récupération de la connexion
			Connection cn = ConnectionMYSQL.getInstance();

			// Etape 2 : préparation requête
			Statement st = cn.createStatement();
			String espace = " ";
```

```
                String sql = "SELECT numComptes, numIdent, solde,
CONCAT(clients.noms,'"+ espace +"',clients.prenoms) AS Clients_idClients,
CONCAT(gestionnaires.noms,'" + espace + "',gestionnaires.prenoms) AS
Gestionnaires_idGestionnaires, typeComptes.libelle AS TypeComptes_idTypeComptes
FROM comptes, clients, gestionnaires, typeComptes WHERE comptes.Clients_idClients =
clients.idClients AND clients.Gestionnaires_idGestionnaires = gestionnaires.idGestionnaires
AND comptes.TypeComptes_idTypeComptes = typeComptes.idTypeComptes";

                // Etape 3 : exécution requéte
                ResultSet rs = st.executeQuery(sql);

                // Etape 4 :parcours Resultset (optionnel)
                while (rs.next()) {
                        Comptes compte = new Comptes();
                        compte.setNumComptes(rs.getInt("numComptes"));
                        compte.setNumIdent(rs.getInt("numIdent"));
                        compte.setSolde(rs.getFloat("solde"));
                        compte.setClients_idClients(rs.getString("Clients_idClients"));
                        compte.setClients_idClients(rs.getString("Clients_idClients"));

        compte.setGestionnaires_idGestionnaires(rs.getString("Gestionnaires_idGestionnaires
"));

        compte.setTypesComptes_idTypesComptes(rs.getString("TypeComptes_idTypeCompt
es"));
                        maListeComptes.add(compte);
                }

                // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
        } catch (SQLException e) {
                e.printStackTrace();

        }

        return maListeComptes;
    }

    /**
     * Cette méthode permet d'enregistre un compte dans la BD
     *
     * @return 0 si échec et 1 si succès lors de l'enregistrement
     * @param compte
     */
    public int enregistrerDaoComptes(Comptes compte) {
        try {

                // Etape 1 : récupération de la connexion
                Connection cn = ConnectionMYSQL.getInstance();
```

```
                        // Etape 2 : Prépare requête
                        String sql = "INSERT INTO `comptes`
(`numComptes`,`numIdent`,`solde`,`Clients_idClients`,`Gestionnaires_idGestionnaires`,`Type
Comptes_idTypeComptes`) VALUES (?,?,?,?,?,?)";

                        // Etape 3 : Création d'un preparedStatement
                        PreparedStatement ps = cn.prepareStatement(sql);

                        ps.setInt(1, compte.getNumComptes());
                        ps.setInt(2, compte.getNumIdent());
                        ps.setFloat(3, compte.getSolde());
                        ps.setString(4, compte.getClients_idClients());
                        ps.setString(5, compte.getGestionnaires_idGestionnaires());
                        ps.setString(6, compte.getTypesComptes_idTypesComptes());

                        // Etape 4 : exécution requête
                        ps.executeUpdate();

                        // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
                } catch (SQLException e) {
                        e.printStackTrace();
                        return 0; // retourne "0" lorsqu'il ya une ereur
                }
                return 1; // retourne "1" lorsque l'enregistrement en BD est réussi
        }

        /**
         * Cette méthode permet de supprimer un compte sélectionné en BD
         *
         * @return 1 si succès et 0 si echec de suppression
         * @param compte
         */
        public int supprimerDaoCompte(Comptes compte) {
                try {

                        // Etape 1 : récupération de la connexion
                        Connection cn = ConnectionMYSQL.getInstance();

                        // Etape 2 : Prépare requête
                        String reqDel = "DELETE from comptes WHERE numComptes = ?";

                        // Etape 3 : Création d'un preparedStatement
                        PreparedStatement ps = cn.prepareStatement(reqDel);

                        ps.setInt(1, compte.getNumComptes());

                        // Etape 4 : exécution requête
                        ps.executeUpdate();
```

```
                // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
            } catch (SQLException e) {
                e.printStackTrace();
                return 0; // retourne "0" lorsqu'il ya une ereur
            }
            return 1; // retourne "1" lorsque la suppression en BD est réussi
        }

        /**
         * Cette méthode permet modfier un compte sélectioné en BD
         *
         * @return 0 si échec et 1 si succès de modification
         * @param compte
         */
        public int modifierDaoCompte(Comptes compte) {
            try {

                // Etape 1 : récupération de la connexion
                Connection cn = ConnectionMYSQL.getInstance();

                // Etape 2 : Prépare requête
                String reqMod = "UPDATE comptes SET numComptes = ?, numIdent
= ?, solde = ?, Clients_idClients = ?, Gestionnaires_idGestionnaires = ?,
TypeComptes_idTypeComptes = ?  WHERE  numComptes = '" + compte.getNumComptes()
+ "'";

                // Etape 3 : Création d'un preparedStatement
                PreparedStatement ps = cn.prepareStatement(reqMod);

                ps.setInt(1, compte.getNumComptes());
                ps.setInt(2, compte.getNumIdent());
                ps.setFloat(3, compte.getSolde());
                ps.setString(4, compte.getClients_idClients());
                ps.setString(5, compte.getGestionnaires_idGestionnaires());
                ps.setString(5, compte.getTypesComptes_idTypesComptes());

                // Etape 4 : exécution requête
                ps.executeUpdate();

                // Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
            } catch (SQLException e) {
                e.printStackTrace();
                return 0; // retourne "0" lorsqu'il ya une ereur
            }
            return 1; // retourne "1" lorsque la modification en BD est réussi
        }

        /**
```

```
	 * Cette méthode permet de recherche un compte seuil en BD
	 *
	 * @return List<Comptes> maListeComptesSeuilTrouver
	 * @param search
	 */
	public List<Transactions> rechercherDaoTransactions(String search) {
		// création d'une collection de patient
		List<Transactions> maListeTransactionsTrouver = new
ArrayList<Transactions>();

		try {

			// Etape 1 : récupération de la connexion
			Connection cn = ConnectionMYSQL.getInstance();

			// Etape 2 : préparation requête
			Statement st = cn.createStatement();

			String espace = "  ";

			String reqSearch = "SELECT transactions.`idTransactions`,
transactions.`numComptes`, transactions.`date`, transactions.`typeTransaction`,
transactions.`montant`, CONCAT(gestionnaires.`noms`,'" + espace +
"',gestionnaires.`prenoms`) AS Gestionnaires_idGestionnaires FROM transactions,
gestionnaires WHERE gestionnaires.`idGestionnaires` = Gestionnaires_idGestionnaires AND
`numComptes` = '" + search + "' ";

			// Etape 3 : exécution requéte
			ResultSet rs = st.executeQuery(reqSearch);

			// Etape 4 :parcours Resultset (optionnel)
			while (rs.next()) {

				Transactions transaction = new Transactions();
				transaction.setIdTransaction(rs.getString("idTransactions"));

	transaction.setCompte(Integer.parseInt(rs.getString("numComptes")));
				transaction.setDate(rs.getString("date"));
				transaction.setTypeTransaction(rs.getString("typeTransaction"));
				transaction.setMontant(rs.getFloat("montant"));

	transaction.setGestionnaires_idGestionnaires(rs.getString("Gestionnaires_idGestionnai
res"));
				maListeTransactionsTrouver.add(transaction);
			}

			// Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
		} catch (SQLException e) {
			e.printStackTrace();
```

```
		}
		return maListeTransactionsTrouver; // retourne une liste lorsque la recherche en
BD a réussi
	}

	/**
	 * Cette méthode permet de recherche un compte en BD
	 *
	 * @return List<Comptes> maListeCompte
	 * @param search
	 */
	public List<Comptes> rechercherDaoComptes(String search) {
		// création d'une collection de patient
		List<Comptes> maListeComptesTrouver = new ArrayList<Comptes>();

		try {

			// Etape 1 : récupération de la connexion
			Connection cn = ConnectionMYSQL.getInstance();

			// Etape 2 : préparation requête
			Statement st = cn.createStatement();

			String espace = "  ";

			String reqSearch = "SELECT numComptes, numIdent, solde,
CONCAT(clients.noms,'"+ espace +"',clients.prenoms) AS Clients_idClients,
CONCAT(gestionnaires.noms,'" + espace + "',gestionnaires.prenoms) AS
Gestionnaires_idGestionnaires, typeComptes.libelle AS TypeComptes_idTypeComptes
FROM comptes, clients, gestionnaires, typeComptes WHERE comptes.Clients_idClients =
clients.idClients AND clients.Gestionnaires_idGestionnaires = gestionnaires.idGestionnaires
AND comptes.TypeComptes_idTypeComptes = typeComptes.idTypeComptes AND
numComptes = '" + search + "'";

			// Etape 3 : exécution requéte
			ResultSet rs = st.executeQuery(reqSearch);

			// Etape 4 :parcours Resultset (optionnel)
			while (rs.next()) {

				Comptes compte = new Comptes();
				compte.setNumComptes(rs.getInt("numComptes"));
				compte.setNumIdent(rs.getInt("numIdent"));
				compte.setSolde(rs.getFloat("solde"));
				compte.setClients_idClients(rs.getString("Clients_idClients"));

	compte.setGestionnaires_idGestionnaires(rs.getString("Gestionnaires_idGestionnaires
"));
```

```
		compte.setTypesComptes_idTypesComptes(rs.getString("TypeComptes_idTypeCompt
es"));
					maListeComptesTrouver.add(compte);
				}

				// Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
			} catch (SQLException e) {
				e.printStackTrace();

			}
			return maListeComptesTrouver; // retourne une liste lorsque la recherche en
BD a réussi
		}

		/**
		 * Cette méthode permet de mettre à jour le solde d'un compte dans la BD avec les
intérêts
		 *
		 * @return 0 si échec et 1 si succès lors de la maj
		 * @param numCompte, interet
		 */
		public int calculInteretDaoComptes(int numCompte, float interet) {
			try {
				// Etape 1 : récupération de la connexion
				Connection cn = ConnectionMYSQL.getInstance();

				// Etape 2 : Prépare requête
				String sql = "UPDATE `comptes` SET `solde` = ('" + interet + "')
WHERE numComptes = '" + numCompte + "'";

				// Etape 3 : Création d'un preparedStatement
				PreparedStatement ps = cn.prepareStatement(sql);

//				ps.setInt(1, transaction.getCompte());
//				ps.setString(2, transaction.getDate());
//				ps.setString(3, transaction.getTypeTransaction());
//				ps.setFloat(4, transaction.getMontant());
//				ps.setString(5, transaction.getGestionnaires_idGestionnaires());

				// Etape 4 : exécution requête
				ps.executeUpdate();

				// Etape 5 : gestion des exceptions et libération 'automatique' des
ressources
			} catch (SQLException e) {
				e.printStackTrace();
				return 0; // retourne "0" lorsqu'il ya une ereur
			}
```

```
        return 1; // retourne "1" lorsque la maj en BD est réussi
}
```