Projet personnel réalisé en cours de formation — Application JAVA

Projet personnel réalisé en cours de formation :

Développement d'un client lourd de gestion de locations de vacances — Neige & Soleil

avec JAVA, MySQL et le pattern MVC

Sommaire

1. Présentation du Projet

1.1 Contexte et Objectifs

Neige et Soleil est une application de bureau développée en Java permettant la gestion complète d'une agence de location de biens immobiliers de vacances (maisons et appartements) par les administrateurs. Le projet intégre la gestion des propriétaires, des clients, des habitations, des réservations et des contrats.

L'application s'appuie sur une architecture MVC classique :

- Couche Présentation (Vue) : Interface graphique Java Swing
- Couche Métier (Contrôleur) : Classes de traitement et logique applicative
- Couche Données (Modèle) : Accès JDBC à une base de données MySQL

1.2 Technologies Utilisées

Technologie	Usage	Version
Java	Langage principal de développement	JDK 11+
Java Swing	Interface graphique utilisateur (GUI)	Intégré au JDK
JDBC	Connexion et requêtes base de données	MySQL Connector/J
MySQL	Système de gestion de base de données	8.x
com.mysql.cj.jdbc.Driver	Pilote JDBC MySQL	Connector/J 8.x

2. Architecture du Projet

2.1 Modèle MVC

Le projet respecte le patron de conception Modèle-Vue-Contrôleur (MVC), organisé en trois packages distincts :

Package	Rôle	Classes principales
Modele	Accès aux données (couche DAO)	Modele.java, Bdd.java
Controleur	Logique métier et entités	Controleur.java, Utilisateur, Client, Proprietaire, Habitation, Maison, Appartement, Reservation, Contrat, Admin, Tableau
Vue	Interface utilisateur Swing	VueConnexion, VueGeneral (non fournies)

2.2 Point d'Entrée de l'Application

La classe NSEvent.java constitue le point d'entrée (méthode main) de l'application. Elle instancie la vue de connexion au démarrage et gère la navigation entre les vues :

```java
package controleur;
import vue.VueConnexion;
import vue.VueGeneral;

public class NSEvent {
    private static VueConnexion uneVueConnexion ;
    private static VueGeneral uneVueGeneral;

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        uneVueConnexion  = new VueConnexion();
    }

    public static void rendreVisibleConnexion(boolean action) {
        uneVueConnexion.setVisible(action);
    }

    public static void creerVueGenerale (boolean action) {
        if(action == true) {
            uneVueGeneral = new VueGeneral();
        }else {
            uneVueGeneral.dispose();
        }
    }

}
```

La classe expose également des méthodes statiques de navigation :

- rendreVisibleConnexion(boolean) : Affiche ou masque la fenêtre de connexion
- creerVueGenerale(boolean) : Crée ou détruit la vue principale de l'application

3. Modèle de Données

3.1 Schéma de la Base de Données

La base de données neigeetsoleil est composée des tables suivantes :

Table	Description	Clé Primaire
utilisateur	Table centrale des comptes (client, propriétaire, admin)	id_user (AUTO_INCREMENT)
client	Données complémentaires du client (hérite de utilisateur)	id_c → utilisateur
proprietaire	Données complémentaires du propriétaire	id_p → utilisateur
admin	Compte administrateur	id_a → utilisateur
habitation	Table parent pour toutes les habitations	ref_hab (AUTO_INCREMENT)
maison	Maisons (sous-type habitation)	ref_hab (AUTO_INCREMENT)
appartement	Appartements (sous-type habitation)	ref_hab (AUTO_INCREMENT)
reservation	Réservations des clients	ref_res (AUTO_INCREMENT)
contrat	Contrats entre propriétaires et agence	ref_c (AUTO_INCREMENT)
photos	Photos des habitations	id_photo (AUTO_INCREMENT)
station	Stations de vacances	num_sta (AUTO_INCREMENT)
activite	Activités proposées par station	(num_sta, num_acti)
reset_mdp	Codes de réinitialisation de mot de passe	Email
archiveReservation	Historique des réservations modifiées	ref_res
archiveContrat	Historique des contrats modifiés	ref_c

```sql
drop database if exists neigeetsoleil;
create database neigeetsoleil;
use neigeetsoleil;

/*** Creation de la table utilisateur pour gérer la connexion ***/
drop table if exists utilisateur;
CREATE TABLE utilisateur (
    id_user INT AUTO_INCREMENT,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    mdp VARCHAR(255) NOT NULL,
    tel VARCHAR(15) NOT NULL,
    role ENUM('client', 'proprietaire', 'admin') NOT NULL DEFAULT 'client',
    PRIMARY KEY (id_user)
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

/*** maj table proprietaire ***/
drop table if exists proprietaire;
create table proprietaire(
    id_p int not null,
    adresse varchar(100),
    cp varchar(10),
    ville varchar(50),
    RIB varchar(50),
    nb_contrat int default 0,

    primary key(id_p),

    constraint fk_proprietaire_user foreign key(id_p) references utilisateur(id_user) on delete cascade
)ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

create table habitation(
    ref_hab int(5) not null auto_increment,
    type_hab varchar(20) not null,
    adr_hab varchar(120) not null,
    cp_hab int(5) not null,
    ville_hab varchar(50) not null,
    tarif_hab_bas float(5) not null,
    tarif_hab_moy float(5) not null,
    tarif_hab_hau float(5) not null,
    surface varchar(10) not null,
    id_p int(5) not null,
    description_hab varchar(200) not null,
    titre_hab varchar(60) not null,
    capacite_hab int(2) not null,
    primary key (ref_hab),
    foreign key (id_p) references proprietaire(id_p)
);
/*
    Faire : "ALTER TABLE habitation ENGINE=InnoDB;" pour que la table photos supporte
    ref_hab en clés etrangere
 */

create table contrat(
    ref_c int(20) not null auto_increment,
    status_c enum("En validation","En cours","Annule","Resilie"),
    annee_signature date,
    annee_fin date,
    id_p int(5) not null,
    ref_hab int(5) not null,
    primary key (ref_c),
    foreign key (id_p) references proprietaire(id_p),
    foreign key (ref_hab) references habitation(ref_hab)
);

/*** maj table client ***/
drop table if exists client;
create table client(
    id_c int not null,
    adresse varchar(100),
    cp varchar(10),
    ville varchar(50),
    RIB varchar(50),

    primary key(id_c),

    constraint fk_client_user foreign key(id_c) references utilisateur(id_user) on delete cascade
)ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

create table reservation(
    ref_res int(5) not null auto_increment,
    date_res date not null,
    nb_perso int(2) not null,
    date_debut date not null,
    date_fin date not null,
    etat_res enum("Validee","En attente","Annulee"),
    id_c int(5) not null,
    ref_hab int(5) not null,
    primary key (ref_res),
    foreign key (id_c) references client(id_c),
    foreign key (ref_hab) references habitation(ref_hab)
);
```

```sql
create table appartement(
    ref_hab int(5) not null auto_increment,
    type_hab varchar(20) not null,
    adr_hab varchar(120) not null,
    cp_hab int(5) not null,
    ville_hab varchar(50) not null,
    tarif_hab_bas float(5) not null,
    tarif_hab_moy float(5) not null,
    tarif_hab_hau float(5) not null,
    surface varchar(10) not null,
    id_p int(5) not null,
    description_hab varchar(200) not null,
    titre_hab varchar(60) not null,
    capacite_hab int(2) not null,
    etage_ap int(2) not null,
    type_ap varchar(3),
    primary key (ref_hab)
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

create table maison(
    ref_hab int(5) not null auto_increment,
    type_hab varchar(20) not null,
    adr_hab varchar(120) not null,
    cp_hab int(5) not null,
    ville_hab varchar(50) not null,
    tarif_hab_bas float(5) not null,
    tarif_hab_moy float(5) not null,
    tarif_hab_hau float(5) not null,
    surface varchar(10) not null,
    id_p int(5) not null,
    description_hab varchar(200) not null,
    titre_hab varchar(60) not null,
    capacite_hab int(2) not null,
    carac_m varchar(50) not null,
    primary key (ref_hab)
) ENGINE = InnoDB CHARSET = utf8mb4;
create table image(
    ref_image int(5) not null auto_increment,
    url_image varchar(200) not null,
    ref_hab int(5) not null,
    primary key (ref_image),
    foreign key (ref_hab) references habitation(ref_hab)
);

create table station(
    num_sta int(5) not null auto_increment,
    nom_sta varchar(50) not null,
    code_reg int(5) not null,
    primary key (num_sta)
);

create table activite(
    num_sta int(5) not null,
    num_acti int(5) not null,
    nom_acti varchar(50) not null,
    tarif_acti float(5) not null,
    primary key (num_sta,num_acti),
    foreign key (num_sta) references station(num_sta)
);

/*** maj table admin ***/
drop table if exists admin;
create table admin (
    id_a int not null,
    primary key(id_a),
    constraint fk_admin_user foreign key (id_a) references utilisateur(id_user) on delete cascade
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;



create table archiveReservation like reservation;
alter table archivereservation add column datehisto date;
```

```sql
create table archiveContrat like contrat;
alter table archiveContrat add column datehisto date;

create table if not exists photos(
     id_photo int not null auto_increment,
     ref_hab int not null,
     url_photo varchar(255) not null,
     is_principal boolean default false,
     primary key(id_photo),
     foreign key(ref_hab) references habitation(ref_hab) on delete cascade on update cascade
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
```

3.2 Héritage et Sous-Types

Le modèle implémente un héritage de table (Table Inheritance Pattern) pour les habitations et les utilisateurs :

Utilisateurs :

- utilisateur → client (id_c = id_user, FK avec CASCADE)
- utilisateur → proprietaire (id_p = id_user, FK avec CASCADE)
- utilisateur → admin (id_a = id_user, FK avec CASCADE)

Habitations :

- habitation → maison (synchronisation via TRIGGERS before insert/update/delete)
- habitation → appartement (synchronisation via TRIGGERS before insert/update/delete)

3.3 Triggers et Automatismes

La base de données comporte 13 triggers assurant l'intégrité et l'automatisation des données :

Trigger	Événement	Action
insert_maison	BEFORE INSERT ON maison	Synchronise la table habitation
insert_appart	BEFORE INSERT ON appartement	Synchronise la table habitation
update_maison	BEFORE UPDATE ON maison	Met à jour habitation
update_appart	BEFORE UPDATE ON appartement	Met à jour habitation
delete_maison	BEFORE DELETE ON maison	Supprime dans habitation
delete_appart	BEFORE DELETE ON appartement	Supprime dans habitation
histoRes	BEFORE UPDATE ON reservation	Archive l'ancienne réservation
histoCon	AFTER UPDATE ON contrat	Archive l'ancien contrat
insert_contrat	AFTER INSERT ON habitation	Crée automatiquement un contrat 'En validation'
updateNbContratInsert	AFTER INSERT ON contrat	Incrémente nb_contrat du propriétaire
updateNbContratDelete	AFTER DELETE ON contrat	Décrémente nb_contrat du propriétaire
formeNomsPrenomsUtilisateurInsert	BEFORE INSERT ON utilisateur	Capitalise nom/prénom
formeNomsPrenomsUtilisateurUpdate	BEFORE INSERT ON utilisateur	Capitalise nom/prénom (update)

```
drop trigger if exists insert_maison;
delimiter //
create trigger insert_maison
before insert on maison for each row BEGIN
    if new.ref_hab is null or new.ref_hab in (select ref_hab from habitation) or new.ref_hab = 0
    then
   set new.ref_hab = ifnull((select ref_hab from habitation where ref_hab >= all
    (select ref_hab from habitation)), 0) +1 ;
end if;
insert into habitation values(new.ref_hab,new.type_hab,new.adr_hab,new.cp_hab,new.ville_hab,new.tarif_hab_bas,new.tarif_hab_moy,new.tarif_hab_hau,
                              new.surface,new.id_p,new.description_hab,new.titre_hab,new.capacite_hab);
end //
delimiter ;

drop trigger if exists insert_appart;
delimiter //
create trigger insert_appart
before insert on appartement for each row BEGIN
    if new.ref_hab is null or new.ref_hab in (select ref_hab from habitation) or new.ref_hab = 0
    then
   set new.ref_hab = ifnull((select ref_hab from habitation where ref_hab >= all
    (select ref_hab from habitation)), 0) +1 ;
end if;
insert into habitation values(new.ref_hab,new.type_hab,new.adr_hab,new.cp_hab,new.ville_hab,new.tarif_hab_bas,new.tarif_hab_moy,new.tarif_hab_hau,
                              new.surface,new.id_p,new.description_hab,new.titre_hab,new.capacite_hab);
end //
delimiter ;
```

```
drop trigger if exists update_maison;
delimiter //
create trigger update_maison
before update on maison for each row BEGIN
    update habitation set ref_hab=new.ref_hab,type_hab=new.type_hab,adr_hab=new.adr_hab,cp_hab=new.cp_hab,ville_hab=new.ville_hab,
                          tarif_hab_bas=new.tarif_hab_bas,tarif_hab_moy=new.tarif_hab_moy,tarif_hab_hau=new.tarif_hab_hau,surface=new.surface,
                          id_p=new.id_p,description_hab=new.description_hab,titre_hab=new.titre_hab,capacite_hab=new.capacite_hab
    where habitation.ref_hab=new.ref_hab;
end //
delimiter ;

drop trigger if exists update_appart;
delimiter //
create trigger update_appart
before update on appartement for each row BEGIN
    update habitation set ref_hab=new.ref_hab,type_hab=new.type_hab,adr_hab=new.adr_hab,cp_hab=new.cp_hab,ville_hab=new.ville_hab,
                          tarif_hab_bas=new.tarif_hab_bas,tarif_hab_moy=new.tarif_hab_moy,tarif_hab_hau=new.tarif_hab_hau,surface=new.surface,
                          id_p=new.id_p,description_hab=new.description_hab,titre_hab=new.titre_hab,capacite_hab=new.capacite_hab
    where habitation.ref_hab=new.ref_hab;
end //
delimiter ;


drop trigger if exists delete_maison;
delimiter //
create trigger delete_maison
before delete on maison for each row BEGIN
    delete from habitation where habitation.ref_hab=old.ref_hab;
end //
delimiter ;
```

```
drop trigger if exists delete_appart;
delimiter //
create trigger delete_appart
before delete on appartement for each row BEGIN
    delete from habitation where habitation.ref_hab=old.ref_hab;
end //
delimiter ;

drop trigger if exists histoRes;
delimiter //
create trigger histoRes
before update on reservation
for each row
begin
    INSERT INTO archiveReservation
        values(old.ref_res,old.date_res,old.nb_perso,old.date_debut,old.date_fin,'Validee',old.id_c,old.ref_hab,curdate());
end //
delimiter ;

drop trigger if exists histoCon;
delimiter //
create trigger histoCon
after update on contrat
for each row
begin
    insert into archiveContrat
        values(old.ref_c,old.status_c,old.annee_signature,old.annee_fin,old.id_p,old.ref_hab);
end //
delimiter ;
```

```
drop trigger if exists insert_contrat;
delimiter //
create trigger insert_contrat
after insert on habitation
for each row
begin
insert into contrat values (null,'En validation',null,null,new.id_p,new.ref_hab);
end //
delimiter ;
```

```
drop trigger if exists updateNbContratInsert;
delimiter //
create trigger updateNbContratInsert
after insert on contrat
for each row
begin
update proprietaire set nb_contrat = nb_contrat + 1
where id_p = new.id_p;
end //
delimiter ;

drop trigger if exists updateNbContratDelete;
delimiter //
create trigger updateNbContratDelete
after delete on contrat
for each row
begin
update proprietaire set nb_contrat = nb_contrat - 1
where id_p = old.id_p;
end //
delimiter ;
```

4. Classes du Contrôleur

4.1 Hiérarchie des Classes

Le package contrôleur regroupe les entités métier et la classe de coordination :

Classe	Héritage	Description
Utilisateur	–	Classe de base : id, nom, prenom, email, mdp, tel, role
Client	extends Utilisateur	Ajoute adresse, cp, ville, rib
Proprietaire	extends Utilisateur	Ajoute adresse, cp, ville, rib
Admin	–	Entité admin indépendante (Id_a, nom, prenom, email, mdp, role)
Habitation	–	Entité habitation : ref_hab, type, adresse, tarifs, surface, capacite
Maison	–	Étend Habitation avec caracteristique
Appartement	–	Étend Habitation avec etage et typeap
Reservation	–	ref_res, dates, nb_perso, etat_res, id_c, ref_hab
Contrat	–	ref_c, status, annee_sign, annee_fin, id_p, ref_hab
Controleur	–	Classe statique de délégation vers Modele
Tableau	extends AbstractTableModel	Modèle de tableau Swing réutilisable
NSEvent	–	Point d'entrée, gestion des vues principales

```java
package controleur;

public class Admin {
    private int Id_a;
    private String nom_a,prenom_a,email_a,mdp_a,role_a;

    public Admin(int Id_a, String nom_a, String prenom_a, String email_a, String mdp_a, String role_a) {
        super();
        this.Id_a = Id_a;
        this.nom_a = nom_a;
        this.prenom_a = prenom_a;
        this.email_a = email_a;
        this.mdp_a = mdp_a;
        this.role_a = role_a;
    }

    public Admin(String nom_a, String prenom_a, String email_a, String mdp_a, String role_a) {
        super();
        this.Id_a = 0;
        this.nom_a = nom_a;
        this.prenom_a = prenom_a;
        this.email_a = email_a;
        this.mdp_a = mdp_a;
        this.role_a = role_a;
    }
```

```java
package controleur;

public class Appartement {

    private int ref_hab,idProprietaire,capacite,etage;
    String type,adresse,cp,ville,tarifBas,tarifMoy,tarifHaut,surface,description,titre,typeap;

    public Appartement(int ref_hab,String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
            String surface, int idProprietaire, String description, String titre, int capacite, int etage, String typeap) {
        super();
        this.ref_hab = ref_hab;
        this.type = type;
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.tarifBas = tarifBas;
        this.tarifMoy = tarifMoy;
        this.tarifHaut = tarifHaut;
        this.surface = surface;
        this.idProprietaire = idProprietaire;
        this.description = description;
        this.titre = titre;
        this.capacite = capacite;
        this.etage = etage;
        this.typeap = typeap;

    }
    public Appartement(String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
            String surface, int idProprietaire, String description, String titre, int capacite, int etage, String typeap) {
        super();
        this.ref_hab = 0;
        this.type = type;
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.tarifBas = tarifBas;
        this.tarifMoy = tarifMoy;
        this.tarifHaut = tarifHaut;
        this.surface = surface;
        this.idProprietaire = idProprietaire;
        this.description = description;
        this.titre = titre;
        this.capacite = capacite;
        this.etage = etage;
        this.typeap = typeap;
    }
```

```java
package controleur;

import java.util.Set;

public class Client extends Utilisateur {

    private String adresse,cp,ville,rib;

    public Client(int id_user,String nom,String prenom,String email,String mdp,String tel, String role,String adresse,String cp,
                  String ville,String rib){
        super(id_user,nom,prenom,email,mdp,tel,role);
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.rib = rib;
    }
    public Client(int id_user,String nom,String prenom,String email,String mdp,String tel,String adresse,String cp,
            String ville,String rib){
        super(id_user,nom,prenom,email,mdp,tel);
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.rib = rib;
    }
    public Client(String nom,String prenom,String email,String mdp,String tel, String role, String adresse,String cp,
            String ville,String rib){
        super(nom,prenom,email,mdp,tel,role);
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.rib = rib;
    }

    public Client(String nom,String prenom,String email,String mdp,String tel, String adresse,String cp,
            String ville,String rib){
        super(nom,prenom,email,mdp,tel);
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.rib = rib;
    }
```

```java
package controleur;

public class Contrat {

    private int ref_c,idProprietaire,ref_hab;
    private String status,annee_sign,annee_fin;

    public Contrat(int ref_c, int idProprietaire, int ref_hab, String status, String annee_sign,
            String annee_fin) {
        super();
        this.ref_c = ref_c;
        this.idProprietaire = idProprietaire;
        this.ref_hab = ref_hab;
        this.status = status;
        this.annee_sign = annee_sign;
        this.annee_fin = annee_fin;
    }

    public Contrat(int idProprietaire, int ref_hab, String status, String annee_sign,
            String annee_fin) {
        super();
        this.ref_c = 0;
        this.idProprietaire = idProprietaire;
        this.ref_hab = ref_hab;
        this.status = status;
        this.annee_sign = annee_sign;
        this.annee_fin = annee_fin;
    }
```

```java
package controleur;

public class Habitation {

    private int ref_hab,idProprietaire,capacite;
    String type,adresse,cp,ville,tarifBas,tarifMoy,tarifHaut,surface,description,titre;

    public Habitation(int ref_hab,String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
            String surface, int idProprietaire, String description, String titre, int capacite) {
        super();
        this.ref_hab = ref_hab;
        this.type = type;
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.tarifBas = tarifBas;
        this.tarifMoy = tarifMoy;
        this.tarifHaut = tarifHaut;
        this.surface = surface;
        this.idProprietaire = idProprietaire;
        this.description = description;
        this.titre = titre;
        this.capacite = capacite;

    }
    public Habitation(String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
            String surface, int idProprietaire, String description, String titre, int capacite) {
        super();
        this.ref_hab = 0;
        this.type = type;
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.tarifBas = tarifBas;
        this.tarifMoy = tarifMoy;
        this.tarifHaut = tarifHaut;
        this.surface = surface;
        this.idProprietaire = idProprietaire;
        this.description = description;
        this.titre = titre;
        this.capacite = capacite;
    }
```

```java
package controleur;

public class Maison {

    private int ref_hab,idProprietaire,capacite;
    String type,adresse,cp,ville,tarifBas,tarifMoy,tarifHaut,surface,description,titre,caracteristique;

    public Maison(int ref_hab,String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
            String surface, int idProprietaire, String description, String titre, int capacite, String caracteristique) {
        super();
        this.ref_hab = ref_hab;
        this.type = type;
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.tarifBas = tarifBas;
        this.tarifMoy = tarifMoy;
        this.tarifHaut = tarifHaut;
        this.surface = surface;
        this.idProprietaire = idProprietaire;
        this.description = description;
        this.titre = titre;
        this.capacite = capacite;
        this.caracteristique = caracteristique;

    }
    public Maison(String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
            String surface, int idProprietaire, String description, String titre, int capacite, String caracteristique) {
        super();
        this.ref_hab = 0;
        this.type = type;
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.tarifBas = tarifBas;
        this.tarifMoy = tarifMoy;
        this.tarifHaut = tarifHaut;
        this.surface = surface;
        this.idProprietaire = idProprietaire;
        this.description = description;
        this.titre = titre;
        this.capacite = capacite;
        this.caracteristique = caracteristique;
    }
```

```java

package controleur;
import vue.VueConnexion;
import vue.VueGeneral;

public class NSEvent {
    private static VueConnexion uneVueConnexion ;
    private static VueGeneral uneVueGeneral;

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        uneVueConnexion  = new VueConnexion();
    }

    public static void rendreVisibleConnexion(boolean action) {
        uneVueConnexion.setVisible(action);
    }

    public static void creerVueGenerale (boolean action) {
        if(action == true) {
            uneVueGeneral = new VueGeneral();
        }else {
            uneVueGeneral.dispose();
        }
    }

}

```

```java
package controleur;

public class Proprietaire extends Utilisateur {

    private String adresse,cp,ville,rib;

    public Proprietaire(int id_user,String nom,String prenom,String email,String mdp,String tel,String role,String adresse,String cp,
                String ville,String rib){
        super(id_user,nom,prenom,email,mdp,tel,role);
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.rib = rib;
    }
    public Proprietaire(int id_user,String nom,String prenom,String email,String mdp,String tel,String adresse,String cp,
                String ville,String rib){
    super(id_user,nom,prenom,email,mdp,tel);
    this.adresse = adresse;
    this.cp = cp;
    this.ville = ville;
    this.rib = rib;
}
    public Proprietaire(String nom,String prenom,String email,String mdp,String tel,String role,String adresse,String cp,
                String ville,String rib){
        super(nom,prenom,email,mdp,tel,role);
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.rib = rib;
    }
    public Proprietaire(String nom,String prenom,String email,String mdp,String tel,String adresse,String cp,
                String ville,String rib){
        super(nom,prenom,email,mdp,tel);
        this.adresse = adresse;
        this.cp = cp;
        this.ville = ville;
        this.rib = rib;
    }
```

```java
package controleur;

public class Reservation {

    private int ref_res,nb_perso,id_c,ref_hab;
    private String date_res,date_debut,date_fin,etat_res;


    public Reservation(int ref_res,String date_res,int nb_perso, String date_debut,String date_fin,String etat_res,
                       int id_c, int ref_hab) {
        this.ref_res = ref_res;
        this.date_res = date_res;
        this.nb_perso = nb_perso;
        this.date_debut = date_debut;
        this.date_fin = date_fin;
        this.etat_res = etat_res;
        this.id_c = id_c;
        this.ref_hab = ref_hab;
    }
    public Reservation(String date_res,int nb_perso, String date_debut,String date_fin,String etat_res,
                       int id_c, int ref_hab) {
        this.ref_res = 0;
        this.date_res = date_res;
        this.nb_perso = nb_perso;
        this.date_debut = date_debut;
        this.date_fin = date_fin;
        this.etat_res = etat_res;
        this.id_c = id_c;
        this.ref_hab = ref_hab;
    }
```

```java
package controleur;

import javax.swing.table.AbstractTableModel;

public class Tableau extends AbstractTableModel {

    private String enTetes[];
    private Object donnees[][];

    public Tableau(Object [][] donnees, String enTetes[]) {
        this.donnees = donnees;
        this.enTetes = enTetes;
    }

    @Override
    public int getRowCount() {
        // TODO Auto-generated method stub
        return this.donnees.length;
    }

    @Override
    public int getColumnCount() {
        // TODO Auto-generated method stub
        return this.enTetes.length;
    }

    @Override
    public Object getValueAt(int i, int j) {
        // TODO Auto-generated method stub
        return this.donnees[i][j];
    }
```

```java
public void ajoutLigne(Object ligne[]) {
    Object matrice [] [] = new Object[this.donnees.length + 1][this.enTetes.length];
    //faire la copie de la matrice
    for (int i = 0; i<this.donnees.length; i++) {
        matrice[i] = this.donnees[i];
    }
    //ajout ligne
    matrice[this.donnees.length] = ligne;
    //recopie la matrice dans les données
    this.donnees = matrice;
    //actualisation changement
    this.fireTableDataChanged();
}

public void setDonnes (Object [][]matrice) {
    this.donnees = matrice;
    this.fireTableDataChanged();//actualise affichage donnees
}

@Override
public String getColumnName(int i) {
    // TODO Auto-generated method stub
    return this.enTetes[i];
}
```

4.2 Classe Contrôleur

La classe Contrôleur joue le rôle de façade statique entre la vue et le modèle. Elle délègue systématiquement les appels à Modele.java et expose les opérations CRUD pour chaque entité :

Entité	Méthodes exposées
Utilisateur	selectWhereUtilisateur(email, mdp), selectWhereAdmin(email, mdp)
Client	insertClient, selectWhereClient, selectAllClients, deleteClient, updateClient
Proprietaire	insertProprietaire, selectWhereProprietaire, selectAllProprietaires, deleteProprietaire, updateProprietaire
Habitation	insertHabitation, selectWhereHabitation, selectAllHabitations, deleteHabitation, updateHabitation
Maison	insertMaison, selectWhereMaison, selectAllMaisons, deleteMaison, updateMaison
Appartement	insertAppartement, selectWhereAppartement, selectAllAppartements, deleteAppartement, updateAppartement
Reservation	insertReservation, selectWhereReservation, selectAllReservations, deleteReservation, updateReservation
Utilitaires	selectCountUtilisateur(role), selectCount(table)

```java
package controleur;

import java.util.ArrayList;

import modele.Modele;

public class Controleur {

    public static Utilisateur selectWhereAdmin(String email, String mdp) {
        return Modele.selectWhereAdmin(email,mdp);
    }


    //Utilisateur
    public static Utilisateur selectWhereUtilisateur(String email, String mdp) {
        return Modele.selectWhereUtilisateur(email,mdp);
    }


    //Client
    public static void insertClient(Client unClient) {
        Modele.insertClient(unClient);
    }
    public static Client selectWhereClient(String email) {
        return Modele.selectWhereClient(email);
    }
    public static ArrayList<Client> selectAllClients(String filtre) {
        return Modele.selectAllClients(filtre);
    }
    public static void deleteClient(int idClient) {
        Modele.deleteClient(idClient);
    }
    public static void updateClient(Client unClient) {
        Modele.updateClient(unClient);
    }
```

```java
    //Proprio
    public static void insertProprietaire(Proprietaire unProprietaire) {
        Modele.insertProprietaire(unProprietaire);
    }
    public static Proprietaire selectWhereProprietaire(String email) {
        return Modele.selectWhereProprietaire(email);
    }
    public static ArrayList<Proprietaire> selectAllProprietaires(String filtre) {
        return Modele.selectAllProprietaires(filtre);
    }
    public static void deleteProprietaire(int idProprietaire) {
        Modele.deleteProprietaire(idProprietaire);
    }
    public static void updateProprietaire(Proprietaire unProprietaire) {
        Modele.updateProprietaire(unProprietaire);
    }


    //habitations
    public static int insertHabitation(Habitation uneHabitation) {
        return Modele.insertHabitation(uneHabitation);
    }
    public static Habitation selectWhereHabitation(int ref_hab) {
        return Modele.selectWhereHabitation(ref_hab);
    }
    public static ArrayList<Habitation> selectAllHabitations(String filtre) {
        return Modele.selectAllHabitations(filtre);
    }
    public static void deleteHabitation(int refHab) {
        Modele.deleteHabitation(refHab);
    }
    public static void updateHabitation (Habitation uneHabitation){
        Modele.updateHabitation(uneHabitation);
    }
```

```java
    //reservations
    public static int insertReservation(Reservation uneReservation) {
        return Modele.insertReservation(uneReservation);
    }
    public static Reservation selectWhereReservation(int ref_res) {
        return Modele.selectWhereReservation(ref_res);
    }
    public static ArrayList<Reservation> selectAllReservations(String filtre) {
        return Modele.selectAllReservations(filtre);
    }
    public static void deleteReservation (int refRes) {
        Modele.deleteReservation(refRes);
    }
    public static void updateReservation(Reservation uneReservation) {
        Modele.updateReservation(uneReservation);
    }

    //maisons
    public static int insertMaison(Maison uneMaison) {
        return Modele.insertMaison(uneMaison);
    }
    public static Maison selectWhereMaison(int ref_hab) {
        return Modele.selectWhereMaison(ref_hab);
    }
    public static ArrayList<Maison> selectAllMaisons(String filtre) {
        return Modele.selectAllMaisons(filtre);
    }
    public static void deleteMaison(int refHab) {
        Modele.deleteMaison(refHab);
    }
    public static void updateMaison (Maison uneMaison){
        Modele.updateMaison(uneMaison);
    }
```

```
109     //appartement
110     public static int insertAppartement(Appartement unAppartement) {
111         return Modele.insertAppartement(unAppartement);
112     }
113     public static Appartement selectWhereAppartement(int ref_hab) {
114         return Modele.selectWhereAppartement(ref_hab);
115     }
116     public static ArrayList<Appartement> selectAllAppartements(String filtre) {
117         return Modele.selectAllAppartements(filtre);
118     }
119     public static void deleteAppartement(int refHab) {
120         Modele.deleteAppartement(refHab);
121     }
122     public static void updateAppartement (Appartement unAppartement){
123         Modele.updateAppartement(unAppartement);
124     }
125
126     /********** Autres méthodes **********/
127     public static int selectCountUtilisateur(String role) {
128         return Modele.selectCountUtilisateur(role);
129     }
130
131     public static int selectCount(String table) {
132         return Modele.selectCount(table);
133     }
```

4.3 Classe Tableau

Tableau étend « AbstractTableModel » de Swing pour fournir un modèle de données réutilisable pour les JTable. Elle implémente :

- getRowCount() / getColumnCount() / getValueAt() : Méthodes obligatoires de AbstractTableModel
- getColumnName() : Retourne les en-têtes de colonnes
- ajoutLigne() : Ajoute dynamiquement une ligne au tableau
- setDonnes() : Remplace toutes les données et déclenche fireTableDataChanged()

```
1  package controleur;
2
3  import javax.swing.table.AbstractTableModel;
4
5  public class Tableau extends AbstractTableModel {
6
7      private String enTetes[];
8      private Object donnees[][];
9
10     public Tableau(Object [][] donnees, String enTetes[]) {
11         this.donnees = donnees;
12         this.enTetes = enTetes;
13     }
14
15     @Override
16     public int getRowCount() {
17         // TODO Auto-generated method stub
18         return this.donnees.length;
19     }
20
21     @Override
22     public int getColumnCount() {
23         // TODO Auto-generated method stub
24         return this.enTetes.length;
25     }
26
27     @Override
28     public Object getValueAt(int i, int j) {
29         // TODO Auto-generated method stub
30         return this.donnees[i][j];
31     }
```

```
33      public void ajoutLigne(Object ligne[]) {
34          Object matrice [] [] = new Object[this.donnees.length + 1][this.enTetes.length];
35          //faire la copie de la matrice
36          for (int i = 0; i<this.donnees.length; i++) {
37              matrice[i] = this.donnees[i];
38          }
39          //ajout ligne
40          matrice[this.donnees.length] = ligne;
41          //recopie la matrice dans les données
42          this.donnees = matrice;
43          //actualisation changement
44          this.fireTableDataChanged();
45      }
46
47      public void setDonnes (Object [][]matrice) {
48          this.donnees = matrice;
49          this.fireTableDataChanged();//actualise affichage donnees
50      }
51
52      @Override
53      public String getColumnName(int i) {
54          // TODO Auto-generated method stub
55          return this.enTetes[i];
56      }
```

5. Couche Données (Modèle)

5.1 Classe Bdd – Connexion JDBC

La classe Bdd (package modele) encapsule la connexion JDBC à MySQL :

- Paramètres : serveur, nom de la base, utilisateur, mot de passe
- chargerPilote() : Charge com.mysql.cj.jdbc.Driver via Class.forName()
- seConnecter() : Établit la connexion via DriverManager.getConnection()
- seDeconnecter() : Ferme proprement la connexion
- getMaConnexion() : Accesseur vers l'objet Connection actif

```java
package modele;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Bdd {

    /* Bdd est une classe pour la connexion au serveur sql avec le pilote jbdc*/

    private String serveur, bdd, user, mdp;

    private Connection maConnexion;

    public Bdd(String serveur, String bdd, String user, String mdp) {
        super();
        this.serveur = serveur;
        this.bdd = bdd;
        this.user = user;
        this.mdp = mdp;
    }

public void chargerPilote() {

    try {
        Class.forName("com.mysql.cj.jdbc.Driver");

    }catch(ClassNotFoundException e){
        System.out.println("Abscence du pilote JDBC");
    }

}
```

```java
public void seConnecter() {
    String url = "jdbc:mysql://" + this.serveur + "/" + this.bdd;
    this.chargerPilote();

    try {
        this.maConnexion = DriverManager.getConnection(url,user,mdp);
    }catch(SQLException e) {
        System.out.println("impossible de se connecter à : " + url);
    }
}

public void seDeconnecter() {
    try {
        if(this.maConnexion != null) {
            this.maConnexion.close();
        }
    }catch(SQLException e) {
        System.out.println("impossible de fermer la connexion");
}

    }
public Connection getMaConnexion() {
    return this.maConnexion;
}
```

5.2 Classe Modele – Opérations CRUD

La classe Modele (package modele) centralise toutes les requêtes SQL. Elle utilise l'instance de Bdd pour exécuter des PreparedStatements et retourner les entités métier correspondantes.

Les opérations disponibles couvrent :

- Authentification : Vérification email/mdp pour utilisateurs et administrateurs
- CRUD Clients : Insertion, lecture, liste filtrée, mise à jour, suppression
- CRUD Propriétaires : Mêmes opérations que Client
- CRUD Habitations : Gestion générique + spécialisée Maison/Appartement
- CRUD Réservations : Avec gestion de l'état (Validée, En attente, Annulée)
- CRUD Contrats : Avec gestion du statut (En validation, En cours, Annulé, Résilié)
- Comptages : selectCountUtilisateur(role), selectCount(table) pour tableaux de bord

```
26  /*** Utilisateur ***/
27  public static Utilisateur selectWhereUtilisateur(String email, String mdp) {
28      Utilisateur unUtilisateur = null;
29      String requete = "select * from utilisateur where email ='"+ email +"' and mdp ='"+ mdp +"';";
30
31      try {
32          uneBdd.seConnecter();
33          //comme le prepare en php
34          Statement unStat = uneBdd.getMaConnexion().createStatement();
35          //comme execute en phph
36          ResultSet unResultat = unStat.executeQuery(requete);
37          //comme le fetch
38          if(unResultat.next()) {
39              unUtilisateur = new Utilisateur(
40                                      unResultat.getInt("Id_user"),
41                                      unResultat.getString("nom"),
42                                      unResultat.getString("prenom"),
43                                      unResultat.getString("email"),
44                                      unResultat.getString("mdp"),
45                                      unResultat.getString("tel"),
46                                      unResultat.getString("role")
47                                  );
48              }
49              uneBdd.seDeconnecter();
50          }catch(SQLException e) {
51              System.out.println("erreur requete : " + requete);
52          }
53          return unUtilisateur;
54      }
```

```
95      /*** Clients ***/
96      public static void insertClient(Client unClient) {
97          String reqUser = "insert into utilisateur values (null, '"
98                  + unClient.getNom() + "', '" + unClient.getPrenom() + "', '"
99                  + unClient.getEmail() + "', '" + unClient.getMdp() + "', '"
100                 + unClient.getTel() + "', 'client');";
101
102         try {
103             uneBdd.seConnecter();
104             Statement unStat = uneBdd.getMaConnexion().createStatement();
105
106             unStat.executeUpdate(reqUser, Statement.RETURN_GENERATED_KEYS);
107
108             int lastId = 0;
109             ResultSet rs = unStat.getGeneratedKeys();
110             if (rs.next()) {
111                 lastId = rs.getInt(1);
112             }
113
114             String reqClient = "insert into client values ("
115                     + lastId + ", '" + unClient.getAdresse() + "', '"
116                     + unClient.getCp() + "', '" + unClient.getVille() + "', '"
117                     + unClient.getRib() + "');";
118
119             unStat.executeUpdate(reqClient);
120
121             unStat.close();
122             uneBdd.seDeconnecter();
123
124             System.out.println("Insertion réussie pour l'ID : " + lastId);
125
126         } catch (SQLException e) {
127             System.out.println("Erreur SQL : " + e.getMessage());
128         }
129     }
```

```java
public static void updateProprietaire(Proprietaire unProprietaire) {
    // 1. Mise à jour de la table parente (Utilisateur)
    String reqUser = "update utilisateur set nom = '" + unProprietaire.getNom() + "', "
            + "prenom = '" + unProprietaire.getPrenom() + "', "
            + "email = '" + unProprietaire.getEmail() + "', "
            + "mdp = '" + unProprietaire.getMdp() + "', "
            + "tel = '" + unProprietaire.getTel() + "' "
            + "where id_user = " + unProprietaire.getId_user() + ";";

    // 2. Mise à jour de la table fille (Client)
    String reqProprietaire = "update proprietaire set adresse = '" + unProprietaire.getAdresse() + "', "
            + "cp = '" + unProprietaire.getCp() + "', "
            + "ville = '" + unProprietaire.getVille() + "', "
            + "RIB = '" + unProprietaire.getRib() + "' "
            + "where id_p = " + unProprietaire.getId_user() + ";";

    try {
        uneBdd.seConnecter();
        Statement unStat = uneBdd.getMaConnexion().createStatement();

        // On exécute les deux mises à jour
        unStat.executeUpdate(reqUser);
        unStat.executeUpdate(reqProprietaire);

        unStat.close();
        uneBdd.seDeconnecter();

        System.out.println("Mise à jour réussie pour l'ID : " + unProprietaire.getId_user());

    } catch (SQLException e) {
        System.out.println("Erreur SQL lors de l'update : " + e.getMessage());
    }
}
```

```java
public static Habitation selectWhereHabitation(int ref_hab) {
    Habitation uneHabitation = null;
    String requete = "SELECT * FROM habitation WHERE ref_hab =" + ref_hab + ";";

    try {
        uneBdd.seConnecter();
        Statement unStat = uneBdd.getMaConnexion().createStatement();
        ResultSet rs = unStat.executeQuery(requete);

        if (rs.next()) {
            uneHabitation = new Habitation(
                rs.getInt("ref_hab"),
                rs.getString("type_hab"),
                rs.getString("adr_hab"),
                rs.getString("cp_hab"),
                rs.getString("ville_hab"),
                rs.getString("tarif_hab_bas"),
                rs.getString("tarif_hab_moy"),
                rs.getString("tarif_hab_hau"),
                rs.getString("surface"),
                rs.getInt("id_p"),
                rs.getString("description"),
                rs.getString("titre"),
                rs.getInt("capacite")
            );
        }

    } catch (SQLException e) {
        System.out.println("Erreur SELECT WHERE : " + e.getMessage());
    }

    return uneHabitation;
}
```

```java
public static void deleteReservation(int refRes) {
    String requete = "delete from reservation where ref_res ='"+refRes+"';";
    executerRequete(requete);
}
```

```
677     public static ArrayList<Maison> selectAllMaisons(String filtre){
678         ArrayList<Maison> lesMaisons = new ArrayList<Maison>();
679         String requete;
680
681         if(filtre.equals("")) {
682             requete = "select * from maison";
683         }else {
684             requete = "select * from maison where ville_hab like '%"+filtre+"%' or cp_hab like '%"+filtre+"%' or type_hab like '%"+filtre+"%';";
685         }
686
687         try {
688             uneBdd.seConnecter();
689             Statement unStat = uneBdd.getMaConnexion().createStatement();
690             ResultSet desResultats = unStat.executeQuery(requete);
691
692             while (desResultats.next()) {
693                 /*Instantiation class habitation*/
694                 Maison uneMaison = new Maison(desResultats.getInt("ref_hab"),
695                         desResultats.getString("type_hab"),desResultats.getString("adr_hab"),desResultats.getString("cp_hab"),
696                         desResultats.getString("ville_hab"),desResultats.getString("tarif_hab_bas"),desResultats.getString("tarif_hab_moy"),
697                         desResultats.getString("tarif_hab_hau"),desResultats.getString("surface"),desResultats.getInt("id_p"),
698                         desResultats.getString("description_hab"),desResultats.getString("titre_hab"),desResultats.getInt("capacite_hab"),
699                         desResultats.getString("carac_m"));
700                 lesMaisons.add(uneMaison);
701             }
702
703         }catch(SQLException e) {
704         System.out.println("Erreur SELECT * : "+ e.getMessage());
705         }
706         return lesMaisons;
707     }
```

```
791     //appartements
792
793     public static void updateAppartement(Appartement unAppartement) {
794         String requete = "update Appartement set type_hab ='"+unAppartement.getType()+"', adr_hab ='"
795                          +unAppartement.getAdresse()+"', cp_hab ='"+unAppartement.getCp()+"', ville_hab ='"
796                          +unAppartement.getVille()+"', tarif_hab_bas ='"+unAppartement.getTarifBas()+"', tarif_hab_moy ='"
797                          +unAppartement.getTarifMoy()+"', tarif_hab_hau ='"+unAppartement.getTarifHaut()+"', surface ='"
798                          +unAppartement.getSurface()+"', id_p='"+unAppartement.getIdProprietaire()+"', description_hab = '"
799                          +unAppartement.getDescription()+"', titre_hab = '"+unAppartement.getTitre()+"', capacite_hab = '"
800                          +unAppartement.getCapacite()+"', etage_ap = '"+unAppartement.getEtage()+"', type_ap = '"
801                          +unAppartement.getTypeap()+"' where ref_hab = '"+unAppartement.getRef_hab()+"';";
802
803         executerRequete(requete);
804     }
```

```
927     /********************************* Methodes communes ***********************************************/
928     public static void executerRequete(String requete) {
929         try {
930             uneBdd.seConnecter();
931             Statement unStat = uneBdd.getMaConnexion().createStatement();
932             unStat.execute(requete);
933             unStat.close();
934             uneBdd.seDeconnecter();
935
936         }catch(SQLException e) {
937             System.out.println("erreur requete : " + requete);
938             e.printStackTrace();
939         }
940     }
941
942     public static int selectCountUtilisateur(String role) {
943         int nb=0;
944         String requete = "select count(*) as nb from utilisateur where role = '"+ role +"';";
945         try {
946             uneBdd.seConnecter();
947             Statement unStat = uneBdd.getMaConnexion().createStatement();
948             ResultSet unRes = unStat.executeQuery(requete);
949             if(unRes.next()) {
950                 nb = unRes.getInt("nb");
951             }
952             unStat.close();
953             uneBdd.seDeconnecter();
954
955         }catch(SQLException e) {
956             System.out.println("erreur requete : " + requete);
957         }
958         return nb;
959     }
```

```
961     public static int selectCount(String table) {
962         int nb=0;
963         String requete = "select count(*) as nb from "+ table +";";
964         try {
965             uneBdd.seConnecter();
966             Statement unStat = uneBdd.getMaConnexion().createStatement();
967             ResultSet unRes = unStat.executeQuery(requete);
968             if(unRes.next()) {
969                 nb = unRes.getInt("nb");
970             }
971             unStat.close();
972             uneBdd.seDeconnecter();
973
974         }catch(SQLException e) {
975             System.out.println("erreur requete : " + requete);
976         }
977         return nb;
978     }
```

6. Gestion des Rôles et des Droits

L'application distingue trois rôles définis dans la colonne ENUM rôle de la table utilisateur :

Rôle	Description	Accès principaux
client	Locataire effectuant des réservations	-
proprietaire	Bailleur proposant des biens	-
admin	Administrateur de la plateforme	Accès complet : tous clients, propriétaires, habitations, contrats, réservations

```sql
/*** Creation de la table utilisateur pour gérer la connexion ***/
drop table if exists utilisateur;
CREATE TABLE utilisateur (
    id_user INT AUTO_INCREMENT,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    mdp VARCHAR(255) NOT NULL,
    tel VARCHAR(15) NOT NULL,
    role ENUM('client', 'proprietaire', 'admin') NOT NULL DEFAULT 'client',
    PRIMARY KEY (id_user)
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
```

L'authentification est réalisée en interrogeant la base avec l'email et le mot de passe..

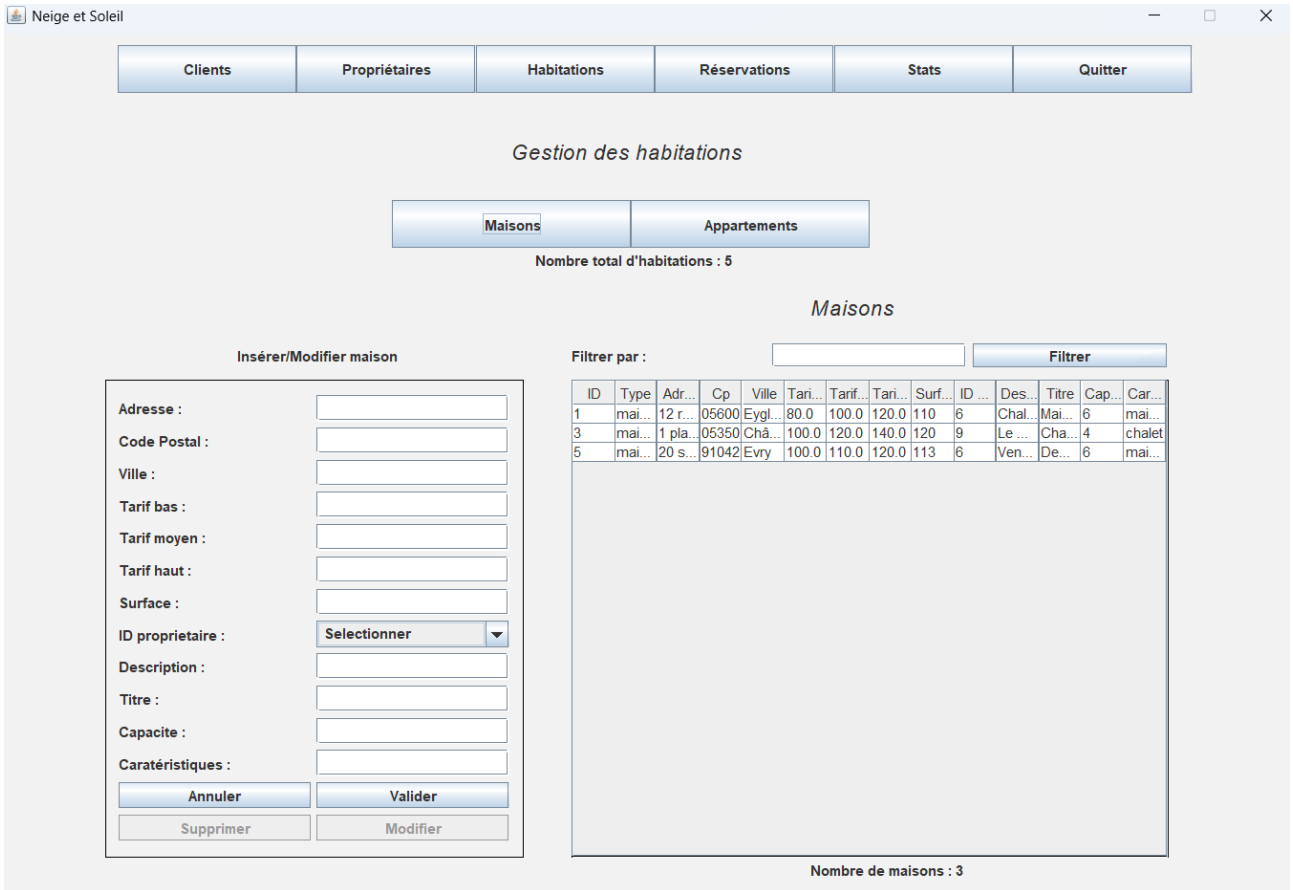
```java
    /*** Admin ***/
    public static Utilisateur selectWhereAdmin(String email, String mdp) {
        Utilisateur unAdmin = null;
        String requete = "select * from utilisateur where email ='"+ email +"' and mdp ='"+ mdp +"' and role = 'admin';";

        try {
            uneBdd.seConnecter();
            //comme le prepare en php
            Statement unStat = uneBdd.getMaConnexion().createStatement();
            //comme execute en phph
            ResultSet unResultat = unStat.executeQuery(requete);
            //comme le fetch
            if(unResultat.next()) {
                unAdmin = new Utilisateur(
                                unResultat.getInt("Id_user"),
                                unResultat.getString("nom"),
                                unResultat.getString("prenom"),
                                unResultat.getString("email"),
                                unResultat.getString("mdp"),
                                unResultat.getString("tel"),
                                unResultat.getString("role")
                                );
            }
            uneBdd.seDeconnecter();
        }catch(SQLException e) {
            System.out.println("erreur requete : " + requete);
        }
        return unAdmin;
    }
```

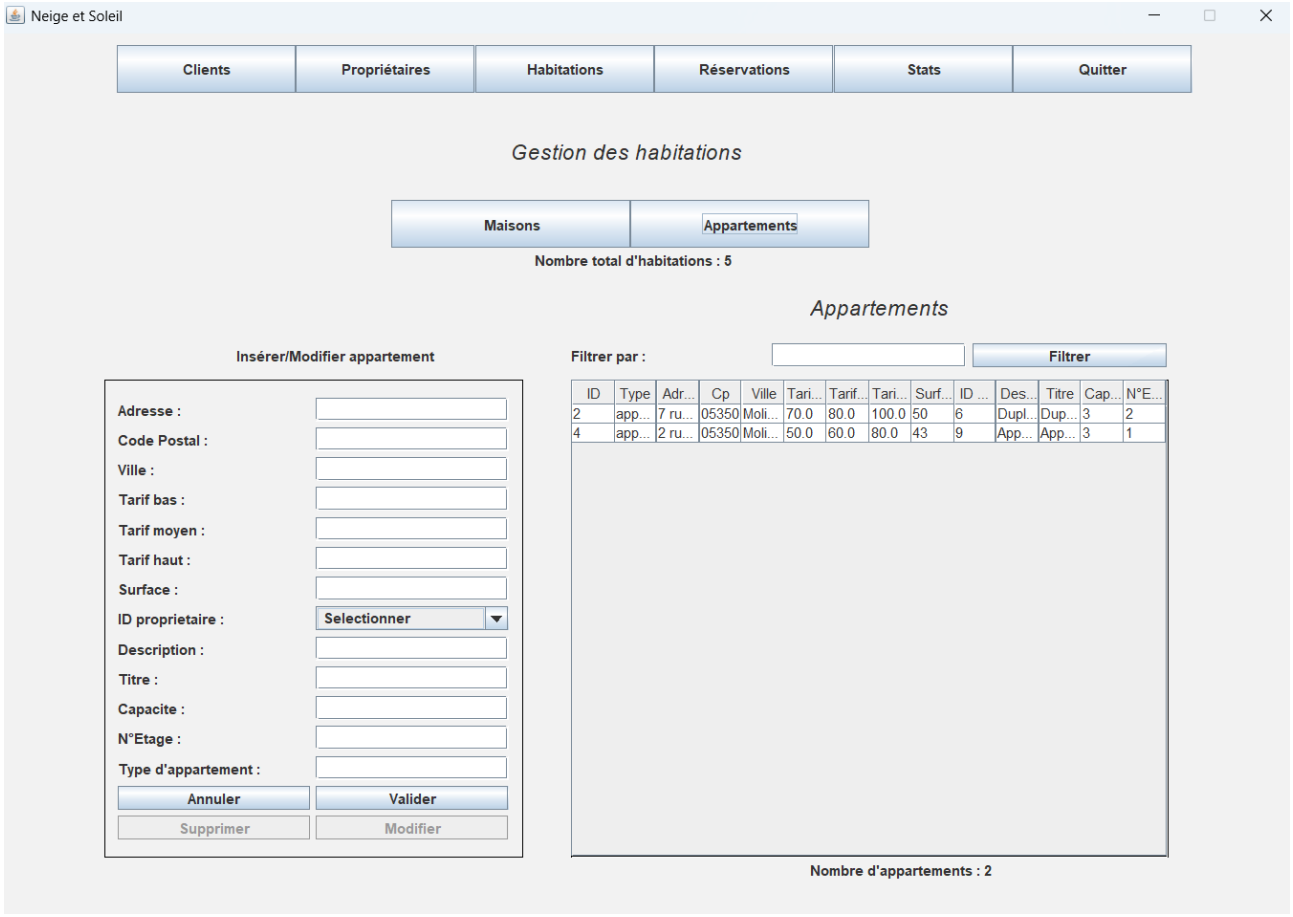
7. Fonctionnalités Métier

7.1 Gestion des Habitations

Les habitations sont de deux types (Maison et Appartement) et partagent une structure commune via la table habitation. Chaque habitation possède :

- Informations géographiques : adresse, code postal, ville
- Tarification à trois niveaux : tarif bas, moyen, haut (selon la saison)
- Caractéristiques : surface, capacité, description, titre, type
- Spécificités Maison : caracteristique (jardin, piscine, etc.)
- Spécificités Appartement : etage, typeap (T1, T2, T3, etc.)
- Photos : gestion des images avec photo principale (is_principal)



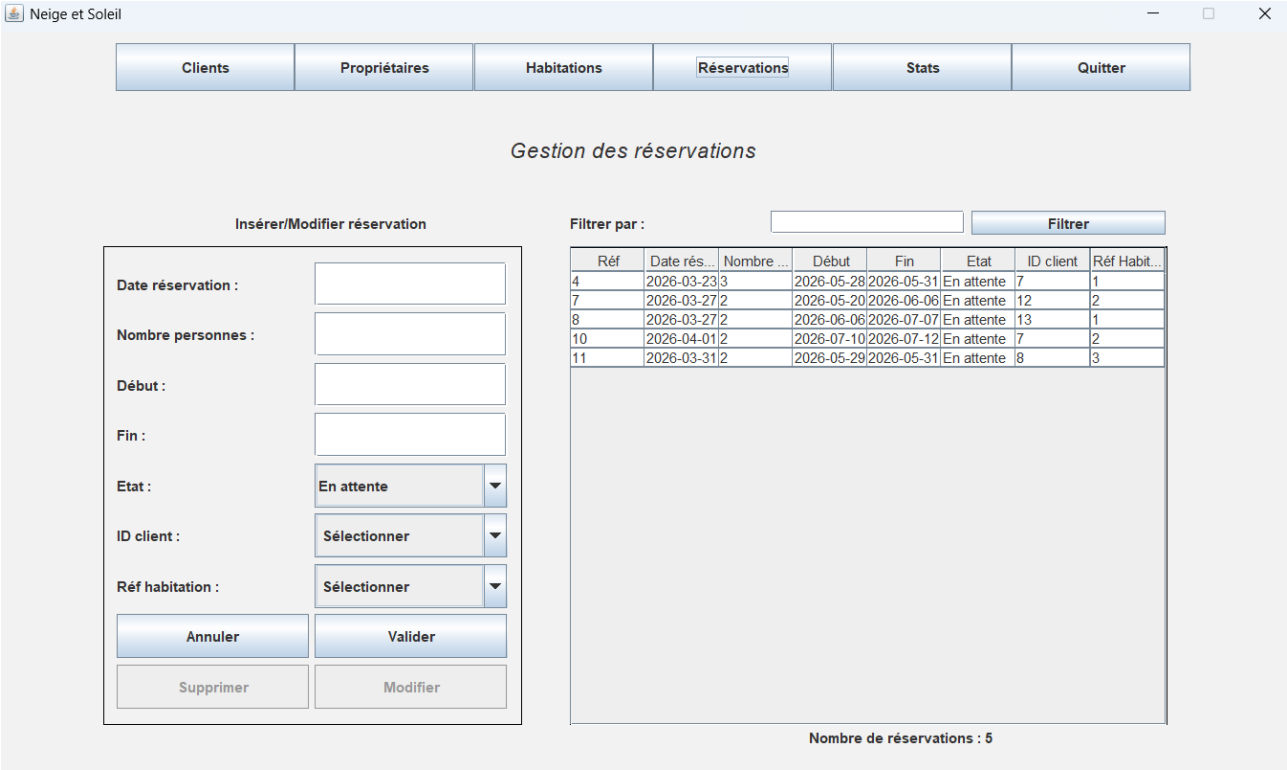

Neige et Soleil
Clients
Propriétaires
Habitations
Réservations
Stats
Quitter
Gestion des habitations
Maisons
Appartements
Nombre total d'habitations : 5
Appartements
Insérer/Modifier appartement
Adresse :
Code Postal :
Ville :
Tarif bas :
Tarif moyen :
Tarif haut :
Surface :
ID proprietaire :
Selectionner
Description :
Titre :
Capacite :
N°Etage :
Type d'appartement :
Annuler
Valider
Supprimer
Modifier
Filtrer par :
Filtrer
ID Type Adr... Cp Ville Tari... Tarif... Tari... Surf... ID ... Des... Titre Cap... N°E...
2 app... 7 ru... 05350 Moli... 70.0 80.0 100.0 50 6 Dupl... Dup... 3 2
4 app... 2 ru... 05350 Moli... 50.0 60.0 80.0 43 9 App... App... 3 1
Nombre d'appartements : 2

7.2 Gestion des Réservations

Une réservation lie un client à une habitation pour une période donnée. Elle comporte :

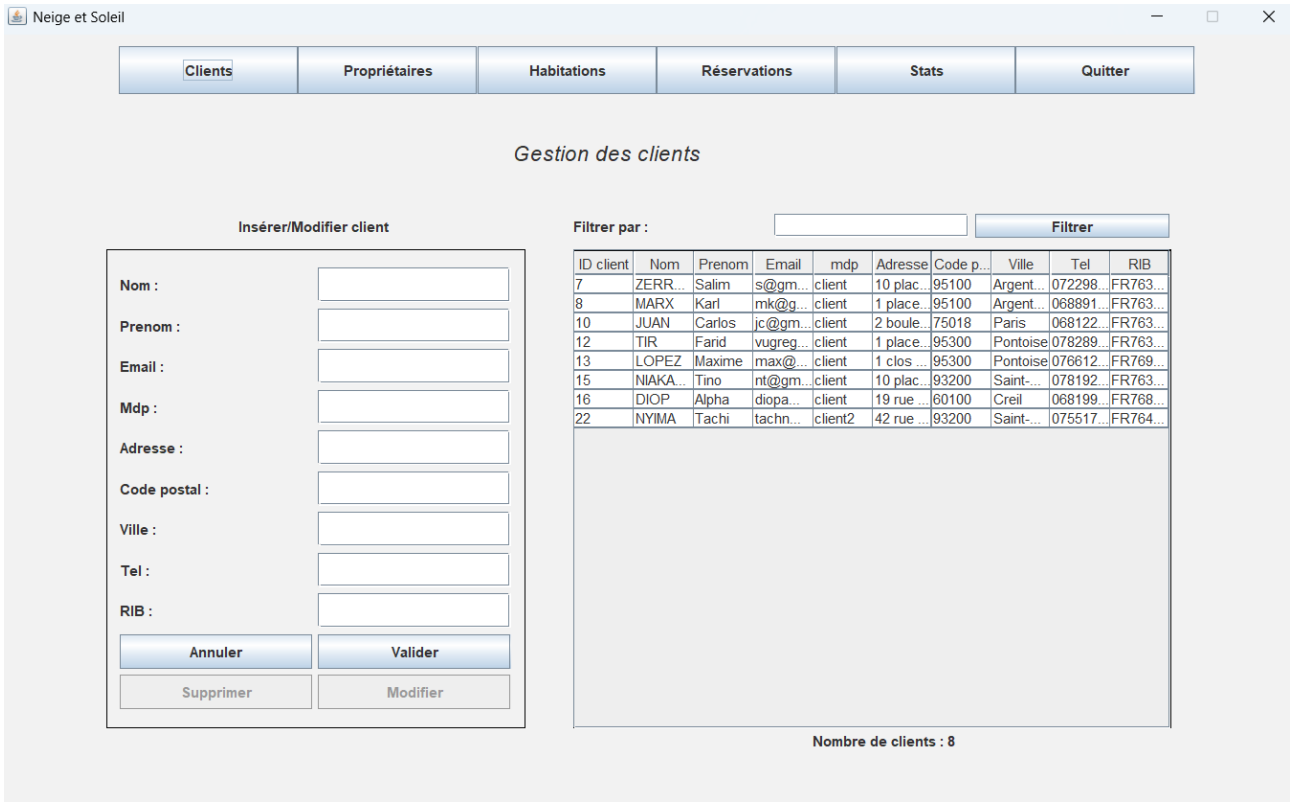
- Date de réservation (date_res)
- Période de séjour (date_debut, date_fin)
- Nombre de personnes (nb_perso)
- État : Validée | En attente | Annulée

À chaque modification d'une réservation, le trigger histoRes archive automatiquement l'état précédent dans archiveReservation.



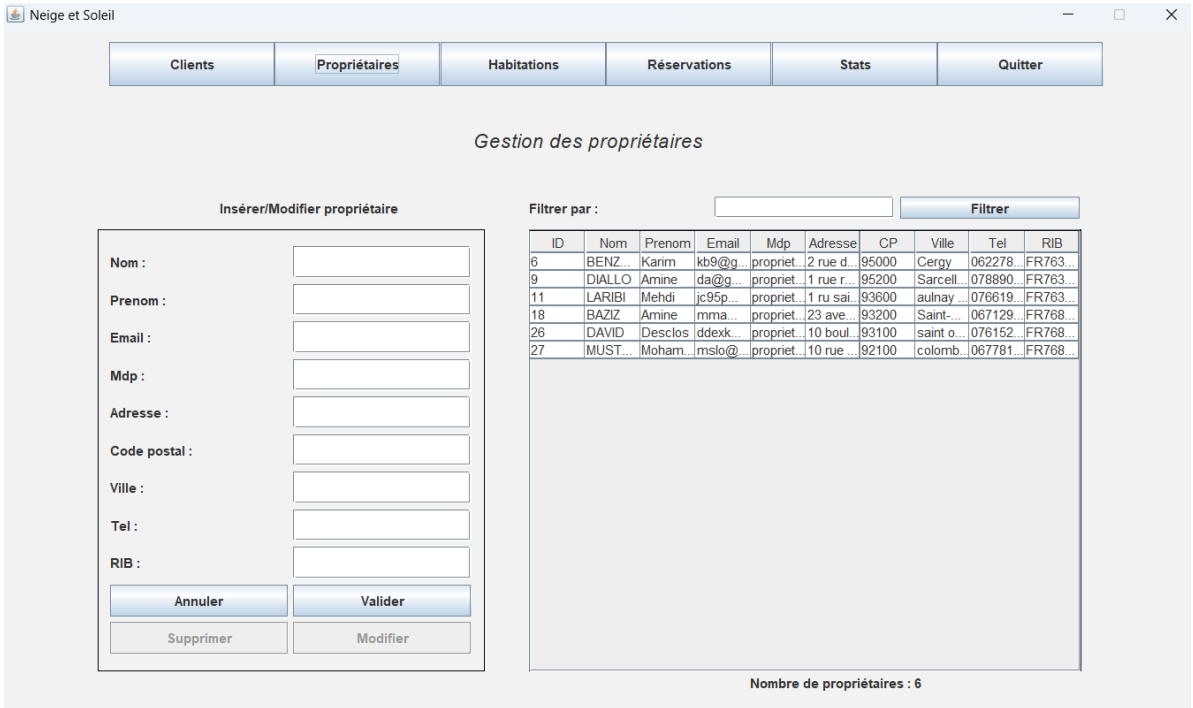
7.3 Gestion des Clients

CRUD complet avec lecture, ajout, modification et suppression de tous les clients.



7.4 Gestion des Propriétaires

CRUD complet avec lecture, ajout, modification et suppression de tous les propriétaires.



8. Points d'Amélioration Identifiés

Point	Description	Priorité
Sécurité – Mots de passe	Les mots de passe stockés en clair. Implémenter BCrypt ou SHA-256 avec sel.	Haute
Héritage Admin	La classe Admin n'hérite pas de Utilisateur contrairement à Client et Propriétaire.	Moyenne
Trigger histoCon	Le trigger histoCon insère dans archiveContrat sans la colonne datehisto alors qu'elle existe.	Haute
Trigger update Utilisateur	Le trigger formeNomsPrenomsUtilisateurUpdate utilise BEFORE INSERT au lieu de BEFORE UPDATE.	Haute
Gestion des erreurs	Les exceptions SQL sont affichées en console. Prévoir une remontée vers l'IHM.	Moyenne
Séparation DAO	Modele.java centralise tout. Envisager une interface DAO par entité.	Basse
Fermeture des ressources	Utiliser try-with-resources pour les PreparedStatement et ResultSet.	Moyenne

9. Conclusion

Le projet Neige et Soleil constitue une application Java Swing complète illustrant les bonnes pratiques de l'architecture MVC appliquée à un contexte de gestion métier réel.

Les points forts du projet sont :

- Architecture MVC claire avec séparation nette des responsabilités
- Modèle de données normalisé avec contraintes d'intégrité référentielle
- Automatisation via triggers MySQL (historisation, synchronisation des sous-types)
- Composant Tableau réutilisable pour l'affichage Swing