

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

---

<b>I. Présentation du projet</b>	3
1.1 Contexte et objectifs	3
1.2 Fonctionnalités attendues	3
<b>II. Architecture du projet</b>	4
2.1 Modèle MVC	4
2.2 Point d'entrée de l'application	4
<b>III. Développement</b>	5
3.1 Schéma de la base de données	5
3.2 Héritage et sous-types	8
3.3 Triggers et automatismes	8
<b>IV. Classes du contrôleur</b>	11
4.1 Hiérarchie des classes	11
4.2 Classe contrôleur	15
4.3 Classe tableau	17
<b>V. Couche données (modèle)</b>	19
5.1 Classe BDD – connexion pilote JDBC	19
5.2 Classe modèle (opérations CRUD)	20
<b>VI. Gestion des rôles et des droits</b>	23
<b>VII. Fonctionnalités métiers</b>	24
7.1 Gestion des habitations	24
7.2 Gestion des réservations	26
7.3 Gestion des clients	27
7.4 Gestion des propriétaires	27
<b>VIII. Points d'amélioration identifiés</b>	28
<b>IX. Conclusion</b>	29

# 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 intègre 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 :

```
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

```

1 drop database if exists neigeetsoleil;
2 create database neigeetsoleil;
3 use neigeetsoleil;
4
5 /**/ Creation de la table utilisateur pour gérer la connexion ***/
6 drop table if exists utilisateur;
7 CREATE TABLE utilisateur (
8     id_user INT AUTO_INCREMENT,
9     nom VARCHAR(50) NOT NULL,
10    prenom VARCHAR(50) NOT NULL,
11    email VARCHAR(100) NOT NULL UNIQUE,
12    mdp VARCHAR(255) NOT NULL,
13    tel VARCHAR(15) NOT NULL,
14    role ENUM('client', 'proprietaire', 'admin') NOT NULL DEFAULT 'client',
15    PRIMARY KEY (id_user)
16 ) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
17
18 /**/ maj table proprietaire ***/
19 drop table if exists proprietaire;
20 create table proprietaire(
21     id_p int not null,
22     adresse varchar(100),
23     cp varchar(10),
24     ville varchar(50),
25     RIB varchar(50),
26     nb_contrat int default 0,
27
28     primary key(id_p),
29
30     constraint fk_proprietaire_user foreign key(id_p) references utilisateur(id_user) on delete cascade
31 )ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
32
33 create table habitation(
34     ref_hab int(5) not null auto_increment,
35     type_hab varchar(20) not null,
36     adr_hab varchar(120) not null,
37     cp_hab int(5) not null,
38     ville_hab varchar(50) not null,
39     tarif_hab_bas float(5) not null,
40     tarif_hab_moy float(5) not null,
41     tarif_hab_hau float(5) not null,
42     surface varchar(10) not null,
43     id_p int(5) not null,
44     description_hab varchar(200) not null,
45     titre_hab varchar(60) not null,
46     capacite_hab int(2) not null,
47     primary key (ref_hab),
48     foreign key (id_p) references proprietaire(id_p)
49 );
50 /*
51 Faire : "ALTER TABLE habitation ENGINE=InnoDB;" pour que la table photos supporte
52 ref_hab en clés étrangere
53 */
54
55 create table contrat(
56     ref_c int(20) not null auto_increment,
57     status_c enum("En validation","En cours","Annule","Resilie"),
58     annee_signature date,
59     annee_fin date,
60     id_p int(5) not null,
61     ref_hab int(5) not null,
62     primary key (ref_c),
63     foreign key (id_p) references proprietaire(id_p),
64     foreign key (ref_hab) references habitation(ref_hab)
65 );
66
67 /**/ maj table client ***/
68 drop table if exists client;
69 create table client(
70     id_c int not null,
71     adresse varchar(100),
72     cp varchar(10),
73     ville varchar(50),
74     RIB varchar(50),
75
76     primary key(id_c),
77
78     constraint fk_client_user foreign key(id_c) references utilisateur(id_user) on delete cascade
79 )ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
80
81 create table reservation(
82     ref_res int(5) not null auto_increment,
83     date_res date not null,
84     nb_perso int(2) not null,
85     date_debut date not null,
86     date_fin date not null,
87     etat_res enum("Validee", "En attente", "Annulee"),
88     id_c int(5) not null,
89     ref_hab int(5) not null,
90     primary key (ref_res),
91     foreign key (id_c) references client(id_c),
92     foreign key (ref_hab) references habitation(ref_hab)
93 );
94

```

```

97 create table appartement(
98     ref_hab int(5) not null auto_increment,
99     type_hab varchar(20) not null,
100    adr_hab varchar(120) not null,
101    cp_hab int(5) not null,
102    ville_hab varchar(50) not null,
103    tarif_hab_bas float(5) not null,
104    tarif_hab_moy float(5) not null,
105    tarif_hab_hau float(5) not null,
106    surface varchar(10) not null,
107    id_p int(5) not null,
108    description_hab varchar(200) not null,
109    titre_hab varchar(60) not null,
110    capacite_hab int(2) not null,
111    etage_ap int(2) not null,
112    type_ap varchar(3),
113    primary key (ref_hab)
114 ) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
115
116 create table maison(
117     ref_hab int(5) not null auto_increment,
118     type_hab varchar(20) not null,
119     adr_hab varchar(120) not null,
120     cp_hab int(5) not null,
121     ville_hab varchar(50) not null,
122     tarif_hab_bas float(5) not null,
123     tarif_hab_moy float(5) not null,
124     tarif_hab_hau float(5) not null,
125     surface varchar(10) not null,
126     id_p int(5) not null,
127     description_hab varchar(200) not null,
128     titre_hab varchar(60) not null,
129     capacite_hab int(2) not null,
130     carac_m varchar(50) not null,
131     primary key (ref_hab)
132 ) ENGINE = InnoDB CHARSET = utf8mb4;
133
134 create table image(
135     ref_image int(5) not null auto_increment,
136     url_image varchar(200) not null,
137     ref_hab int(5) not null,
138     primary key (ref_image),
139     foreign key (ref_hab) references habitation(ref_hab)
140 );
141
142 create table station(
143     num_sta int(5) not null auto_increment,
144     nom_sta varchar(50) not null,
145     code_reg int(5) not null,
146     primary key (num_sta)
147 );
148
149 create table activite(
150     num_sta int(5) not null,
151     num_acti int(5) not null,
152     nom_acti varchar(50) not null,
153     tarif_acti float(5) not null,
154     primary key (num_sta,num_acti),
155     foreign key (num_sta) references station(num_sta)
156 );
157
158 /** maj table admin */
159 drop table if exists admin;
160 create table admin (
161     id_a int not null,
162     primary key(id_a),
163     constraint fk_admin_user foreign key (id_a) references utilisateur(id_user) on delete cascade
164 ) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
165
166
167
168 create table archiveReservation like reservation;
169 alter table archivereservation add column datehisto date;
170
171 create table archiveContrat like contrat;
172 alter table archiveContrat add column datehisto date;
173
174 create table if not exists photos(
175     id_photo int not null auto_increment,
176     ref_hab int not null,
177     url_photo varchar(255) not null,
178     is_principal boolean default false,
179     primary key(id_photo),
180     foreign key(ref_hab) references habitation(ref_hab) on delete cascade on update cascade
181 ) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;
182

```

### 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 → propriétaire (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	Incréménte nb_contrat du propriétaire
updateNbContratDelete	AFTER DELETE ON contrat	Décréménte nb_contrat du propriétaire
formeNomsPrenomsUtilisateurInsert	BEFORE INSERT ON utilisateur	Capitalise nom/prénom utilisateur
formeNomsPrenomsUtilisateurUpdate	BEFORE INSERT ON utilisateur	Capitalise nom/prénom (update)

```

183 drop trigger if exists insert_maison;
184 delimiter //
185 create trigger insert_maison
186 before insert on maison for each row BEGIN
187     if new.ref_hab is null or new.ref_hab in (select ref_hab from habitation) or new.ref_hab = 0
188     then
189         set new.ref_hab = ifnull((select ref_hab from habitation where ref_hab >= all
190             (select ref_hab from habitation)), 0) +1 ;
191     end if;
192 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,
193     new.surface,new.id_p,new.description_hab,new.titre_hab,new.capacite_hab);
194 end //
195 delimiter ;
196
197 drop trigger if exists insert_appart;
198 delimiter //
199 create trigger insert_appart
200 before insert on appartement for each row BEGIN
201     if new.ref_hab is null or new.ref_hab in (select ref_hab from habitation) or new.ref_hab = 0
202     then
203         set new.ref_hab = ifnull((select ref_hab from habitation where ref_hab >= all
204             (select ref_hab from habitation)), 0) +1 ;
205     end if;
206 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,
207     new.surface,new.id_p,new.description_hab,new.titre_hab,new.capacite_hab);
208 end //
209 delimiter ;
---

211 drop trigger if exists update_maison;
212 delimiter //
213 create trigger update_maison
214 before update on maison for each row BEGIN
215     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,
216         tarif_hab_bas=new.tarif_hab_bas,tarif_hab_moy=new.tarif_hab_moy,tarif_hab_hau=new.tarif_hab_hau,surface=new.surface,
217         id_p=new.id_p,description_hab=new.description_hab,titre_hab=new.titre_hab,capacite_hab=new.capacite_hab
218     where habitation.ref_hab=new.ref_hab;
219 end //
220 delimiter ;
221
222 drop trigger if exists update_appart;
223 delimiter //
224 create trigger update_appart
225 before update on appartement for each row BEGIN
226     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,
227         tarif_hab_bas=new.tarif_hab_bas,tarif_hab_moy=new.tarif_hab_moy,tarif_hab_hau=new.tarif_hab_hau,surface=new.surface,
228         id_p=new.id_p,description_hab=new.description_hab,titre_hab=new.titre_hab,capacite_hab=new.capacite_hab
229     where habitation.ref_hab=new.ref_hab;
230 end //
231 delimiter ;
232
233
234 drop trigger if exists delete_maison;
235 delimiter //
236 create trigger delete_maison
237 before delete on maison for each row BEGIN
238     delete from habitation where habitation.ref_hab=old.ref_hab;
239 end //
240 delimiter ;

242 drop trigger if exists delete_appart;
243 delimiter //
244 create trigger delete_appart
245 before delete on appartement for each row BEGIN
246     delete from habitation where habitation.ref_hab=old.ref_hab;
247 end //
248 delimiter ;
249
250 drop trigger if exists histoRes;
251 delimiter //
252 create trigger histoRes
253 before update on reservation
254 for each row
255 begin
256     INSERT INTO archiveReservation
257     values(old.ref_res,old.date_res,old.nb_perso,old.date_debut,old.date_fin,'Validee',old.id_c,old.ref_hab,curdate());
258 end //
259 delimiter ;
260
261 drop trigger if exists histoCon;
262 delimiter //
263 create trigger histoCon
264 after update on contrat
265 for each row
266 begin
267     insert into archiveContrat
268     values(old.ref_c,old.status_c,old.annee_signature,old.annee_fin,old.id_p,old.ref_hab);
269 end //
270 delimiter ;

272 drop trigger if exists insert_contrat;
273 delimiter //
274 create trigger insert_contrat
275 after insert on habitation
276 for each row
277 begin
278     insert into contrat values (null,'En validation',null,null,new.id_p,new.ref_hab);
279 end //
280 delimiter ;

```

```
282 drop trigger if exists updateNbContratInsert;
283 delimiter //
284 create trigger updateNbContratInsert
285 after insert on contrat
286 for each row
287 begin
288 update propriétaire set nb_contrat = nb_contrat + 1
289 where id_p = new.id_p;
290 end //
291 delimiter ;
292
293 drop trigger if exists updateNbContratDelete;
294 delimiter //
295 create trigger updateNbContratDelete
296 after delete on contrat
297 for each row
298 begin
299 update propriétaire set nb_contrat = nb_contrat - 1
300 where id_p = old.id_p;
301 end //
302 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

```

1 package controleur;
2
3 public class Admin {
4     private int Id_a;
5     private String nom_a, prenom_a, email_a, mdp_a, role_a;
6
7     public Admin(int Id_a, String nom_a, String prenom_a, String email_a, String mdp_a, String role_a) {
8         super();
9         this.Id_a = Id_a;
10        this.nom_a = nom_a;
11        this.prenom_a = prenom_a;
12        this.email_a = email_a;
13        this.mdp_a = mdp_a;
14        this.role_a = role_a;
15    }
16
17    public Admin(String nom_a, String prenom_a, String email_a, String mdp_a, String role_a) {
18        super();
19        this.Id_a = 0;
20        this.nom_a = nom_a;
21        this.prenom_a = prenom_a;
22        this.email_a = email_a;
23        this.mdp_a = mdp_a;
24        this.role_a = role_a;
25    }

```

```

1 package controleur;
2
3 public class Appartement {
4
5     private int ref_hab,idProprietaire,capacite,etage;
6     String type,adresse,cp,ville,tarifBas,tarifMoy,tarifHaut,surface,description,titre,typeap;
7
8     public Appartement(int ref_hab,String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
9         String surface, int idProprietaire, String description, String titre, int capacite, int etage, String typeap) {
10         super();
11         this.ref_hab = ref_hab;
12         this.type = type;
13         this.adresse = adresse;
14         this.cp = cp;
15         this.ville = ville;
16         this.tarifBas = tarifBas;
17         this.tarifMoy = tarifMoy;
18         this.tarifHaut = tarifHaut;
19         this.surface = surface;
20         this.idProprietaire = idProprietaire;
21         this.description = description;
22         this.titre = titre;
23         this.capacite = capacite;
24         this.etage = etage;
25         this.typeap = typeap;
26     }
27
28     public Appartement(String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
29         String surface, int idProprietaire, String description, String titre, int capacite, int etage, String typeap) {
30         super();
31         this.ref_hab = 0;
32         this.type = type;
33         this.adresse = adresse;
34         this.cp = cp;
35         this.ville = ville;
36         this.tarifBas = tarifBas;
37         this.tarifMoy = tarifMoy;
38         this.tarifHaut = tarifHaut;
39         this.surface = surface;
40         this.idProprietaire = idProprietaire;
41         this.description = description;
42         this.titre = titre;
43         this.capacite = capacite;
44         this.etage = etage;
45         this.typeap = typeap;
46     }

```

```

1 package controleur;
2
3 import java.util.Set;
4
5 public class Client extends Utilisateur {
6
7     private String adresse,cp,ville,rib;
8
9     public Client(int id_user,String nom,String prenom,String email,String mdp,String tel, String role,String adresse,String cp,
10         String ville,String rib){
11         super(id_user,nom,prenom,email,mdp,tel,role);
12         this.adresse = adresse;
13         this.cp = cp;
14         this.ville = ville;
15         this.rib = rib;
16     }
17     public Client(int id_user,String nom,String prenom,String email,String mdp,String tel,String adresse,String cp,
18         String ville,String rib){
19         super(id_user,nom,prenom,email,mdp,tel);
20         this.adresse = adresse;
21         this.cp = cp;
22         this.ville = ville;
23         this.rib = rib;
24     }
25     public Client(String nom,String prenom,String email,String mdp,String tel, String role, String adresse,String cp,
26         String ville,String rib){
27         super(nom,prenom,email,mdp,tel,role);
28         this.adresse = adresse;
29         this.cp = cp;
30         this.ville = ville;
31         this.rib = rib;
32     }
33     public Client(String nom,String prenom,String email,String mdp,String tel, String adresse,String cp,
34         String ville,String rib){
35         super(nom,prenom,email,mdp,tel);
36         this.adresse = adresse;
37         this.cp = cp;
38         this.ville = ville;
39         this.rib = rib;
40     }
41 }

```

```

1 package controleur;
2
3 public class Contrat {
4
5     private int ref_c,idProprietaire,ref_hab;
6     private String status,annee_sign,annee_fin;
7
8     public Contrat(int ref_c, int idProprietaire, int ref_hab, String status, String annee_sign,
9         String annee_fin) {
10         super();
11         this.ref_c = ref_c;
12         this.idProprietaire = idProprietaire;
13         this.ref_hab = ref_hab;
14         this.status = status;
15         this.annee_sign = annee_sign;
16         this.annee_fin = annee_fin;
17     }
18
19     public Contrat(int idProprietaire, int ref_hab, String status, String annee_sign,
20         String annee_fin) {
21         super();
22         this.ref_c = 0;
23         this.idProprietaire = idProprietaire;
24         this.ref_hab = ref_hab;
25         this.status = status;
26         this.annee_sign = annee_sign;
27         this.annee_fin = annee_fin;
28     }

```

```

1 package controleur;
2
3 public class Habitation {
4
5     private int ref_hab,idProprietaire,capacite;
6     String type,adresse,cp,ville,tarifBas,tarifMoy,tarifHaut,surface,description,titre;
7
8     public Habitation(int ref_hab,String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
9         String surface, int idProprietaire, String description, String titre, int capacite) {
10         super();
11         this.ref_hab = ref_hab;
12         this.type = type;
13         this.adresse = adresse;
14         this.cp = cp;
15         this.ville = ville;
16         this.tarifBas = tarifBas;
17         this.tarifMoy = tarifMoy;
18         this.tarifHaut = tarifHaut;
19         this.surface = surface;
20         this.idProprietaire = idProprietaire;
21         this.description = description;
22         this.titre = titre;
23         this.capacite = capacite;
24     }
25 }
26 public Habitation(String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
27     String surface, int idProprietaire, String description, String titre, int capacite) {
28     super();
29     this.ref_hab = 0;
30     this.type = type;
31     this.adresse = adresse;
32     this.cp = cp;
33     this.ville = ville;
34     this.tarifBas = tarifBas;
35     this.tarifMoy = tarifMoy;
36     this.tarifHaut = tarifHaut;
37     this.surface = surface;
38     this.idProprietaire = idProprietaire;
39     this.description = description;
40     this.titre = titre;
41     this.capacite = capacite;
42 }

```

```

1 package controleur;
2
3 public class Maison {
4
5     private int ref_hab,idProprietaire,capacite;
6     String type,adresse,cp,ville,tarifBas,tarifMoy,tarifHaut,surface,description,titre,caracteristique;
7
8     public Maison(int ref_hab,String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
9         String surface, int idProprietaire, String description, String titre, int capacite, String caracteristique) {
10         super();
11         this.ref_hab = ref_hab;
12         this.type = type;
13         this.adresse = adresse;
14         this.cp = cp;
15         this.ville = ville;
16         this.tarifBas = tarifBas;
17         this.tarifMoy = tarifMoy;
18         this.tarifHaut = tarifHaut;
19         this.surface = surface;
20         this.idProprietaire = idProprietaire;
21         this.description = description;
22         this.titre = titre;
23         this.capacite = capacite;
24         this.caracteristique = caracteristique;
25     }
26 }
27 public Maison(String type,String adresse, String cp, String ville,String tarifBas,String tarifMoy, String tarifHaut,
28     String surface, int idProprietaire, String description, String titre, int capacite, String caracteristique) {
29     super();
30     this.ref_hab = 0;
31     this.type = type;
32     this.adresse = adresse;
33     this.cp = cp;
34     this.ville = ville;
35     this.tarifBas = tarifBas;
36     this.tarifMoy = tarifMoy;
37     this.tarifHaut = tarifHaut;
38     this.surface = surface;
39     this.idProprietaire = idProprietaire;
40     this.description = description;
41     this.titre = titre;
42     this.capacite = capacite;
43     this.caracteristique = caracteristique;
44 }

```

```

1 package controleur;
2 import vue.VueConnexion;
3 import vue.VueGeneral;
4
5 public class NSEvent {
6     private static VueConnexion uneVueConnexion ;
7     private static VueGeneral uneVueGeneral;
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         uneVueConnexion = new VueConnexion();
12     }
13
14     public static void rendreVisibleConnexion(boolean action) {
15         uneVueConnexion.setVisible(action);
16     }
17
18     public static void creerVueGenerale (boolean action) {
19         if(action == true) {
20             uneVueGeneral = new VueGeneral();
21         }else {
22             uneVueGeneral.dispose();
23         }
24     }
25 }
26
27
28
29

```

```

1 package controleur;
2
3 public class Proprietaire extends Utilisateur {
4     private String adresse,cp,ville,rib;
5
6     public Proprietaire(int id_user,String nom,String prenom,String email,String mdp,String tel,String role,String adresse,String cp,
7         String ville,String rib){
8         super(id_user,nom,prenom,email,mdp,tel,role);
9         this.adresse = adresse;
10        this.cp = cp;
11        this.ville = ville;
12        this.rib = rib;
13    }
14
15    public Proprietaire(int id_user,String nom,String prenom,String email,String mdp,String tel,String adresse,String cp,
16        String ville,String rib){
17        super(id_user,nom,prenom,email,mdp,tel);
18        this.adresse = adresse;
19        this.cp = cp;
20        this.ville = ville;
21        this.rib = rib;
22    }
23
24    public Proprietaire(String nom,String prenom,String email,String mdp,String tel,String role,String adresse,String cp,
25        String ville,String rib){
26        super(nom,prenom,email,mdp,tel,role);
27        this.adresse = adresse;
28        this.cp = cp;
29        this.ville = ville;
30        this.rib = rib;
31    }
32
33    public Proprietaire(String nom,String prenom,String email,String mdp,String tel,String adresse,String cp,
34        String ville,String rib){
35        super(nom,prenom,email,mdp,tel);
36        this.adresse = adresse;
37        this.cp = cp;
38        this.ville = ville;
39        this.rib = rib;
40    }
41 }

```

```

1 package controleur;
2
3 public class Reservation {
4     private int ref_res,nb_perso,id_c,ref_hab;
5     private String date_res,date_debut,date_fin,etat_res;
6
7
8
9     public Reservation(int ref_res,String date_res,int nb_perso, String date_debut,String date_fin,String etat_res,
10        int id_c, int ref_hab) {
11        this.ref_res = ref_res;
12        this.date_res = date_res;
13        this.nb_perso = nb_perso;
14        this.date_debut = date_debut;
15        this.date_fin = date_fin;
16        this.etat_res = etat_res;
17        this.id_c = id_c;
18        this.ref_hab = ref_hab;
19    }
20
21    public Reservation(String date_res,int nb_perso, String date_debut,String date_fin,String etat_res,
22        int id_c, int ref_hab) {
23        this.ref_res = 0;
24        this.date_res = date_res;
25        this.nb_perso = nb_perso;
26        this.date_debut = date_debut;
27        this.date_fin = date_fin;
28        this.etat_res = etat_res;
29        this.id_c = id_c;
30        this.ref_hab = ref_hab;
31    }
32 }

```

```

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    }
32 }

```

```

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 donnees
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 }

```

## 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)

```

1 package controleur;
2
3 import java.util.ArrayList;
4
5 import modele.Modele;
6
7 public class Controleur {
8
9     public static Utilisateur selectWhereAdmin(String email, String mdp) {
10         return Modele.selectWhereAdmin(email,mdp);
11     }
12
13
14     //Utilisateur
15     public static Utilisateur selectWhereUtilisateur(String email, String mdp) {
16         return Modele.selectWhereUtilisateur(email,mdp);
17     }
18
19
20     //Client
21     public static void insertClient(Client unClient) {
22         Modele.insertClient(unClient);
23     }
24     public static Client selectWhereClient(String email) {
25         return Modele.selectWhereClient(email);
26     }
27     public static ArrayList<Client> selectAllClients(String filtre) {
28         return Modele.selectAllClients(filtre);
29     }
30     public static void deleteClient(int idClient) {
31         Modele.deleteClient(idClient);
32     }
33     public static void updateClient(Client unClient) {
34         Modele.updateClient(unClient);
35     }
36
37
38     //Proprio
39     public static void insertProprietaire(Proprietaire unProprietaire) {
40         Modele.insertProprietaire(unProprietaire);
41     }
42     public static Proprietaire selectWhereProprietaire(String email) {
43         return Modele.selectWhereProprietaire(email);
44     }
45     public static ArrayList<Proprietaire> selectAllProprietaires(String filtre) {
46         return Modele.selectAllProprietaires(filtre);
47     }
48     public static void deleteProprietaire(int idProprietaire) {
49         Modele.deleteProprietaire(idProprietaire);
50     }
51     public static void updateProprietaire(Proprietaire unProprietaire) {
52         Modele.updateProprietaire(unProprietaire);
53     }
54
55
56     //habitations
57     public static int insertHabitation(Habitation uneHabitation) {
58         return Modele.insertHabitation(uneHabitation);
59     }
60     public static Habitation selectWhereHabitation(int ref_hab) {
61         return Modele.selectWhereHabitation(ref_hab);
62     }
63     public static ArrayList<Habitation> selectAllHabitations(String filtre) {
64         return Modele.selectAllHabitations(filtre);
65     }
66     public static void deleteHabitation(int refHab) {
67         Modele.deleteHabitation(refHab);
68     }
69     public static void updateHabitation (Habitation uneHabitation){
70         Modele.updateHabitation(uneHabitation);
71     }
72
73
74     //reservations
75     public static int insertReservation(Reservation uneReservation) {
76         return Modele.insertReservation(uneReservation);
77     }
78     public static Reservation selectWhereReservation(int ref_res) {
79         return Modele.selectWhereReservation(ref_res);
80     }
81     public static ArrayList<Reservation> selectAllReservations(String filtre) {
82         return Modele.selectAllReservations(filtre);
83     }
84     public static void deleteReservation (int refRes) {
85         Modele.deleteReservation(refRes);
86     }
87     public static void updateReservation(Reservation uneReservation) {
88         Modele.updateReservation(uneReservation);
89     }
90
91     //maisons
92     public static int insertMaison(Maison uneMaison) {
93         return Modele.insertMaison(uneMaison);
94     }
95     public static Maison selectWhereMaison(int ref_hab) {
96         return Modele.selectWhereMaison(ref_hab);
97     }
98     public static ArrayList<Maison> selectAllMaisons(String filtre) {
99         return Modele.selectAllMaisons(filtre);
100    }
101    public static void deleteMaison(int refHab) {
102        Modele.deleteMaison(refHab);
103    }
104    public static void updateMaison (Maison uneMaison){
105        Modele.updateMaison(uneMaison);
106    }
107

```

```

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

```

1 package modele;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class Bdd {
8
9     /* Bdd est une classe pour la connexion au serveur sql avec le pilote jdbc*/
10
11     private String serveur, bdd, user, mdp;
12
13     private Connection maConnexion;
14
15     public Bdd(String serveur, String bdd, String user, String mdp) {
16         super();
17         this.serveur = serveur;
18         this.bdd = bdd;
19         this.user = user;
20         this.mdp = mdp;
21     }
22
23     public void chargerPilote() {
24
25         try {
26             Class.forName("com.mysql.cj.jdbc.Driver");
27         } catch (ClassNotFoundException e) {
28             System.out.println("Absence du pilote JDBC");
29         }
30     }
31
32 }
33
34 public void seConnecter() {
35     String url = "jdbc:mysql://" + this.serveur + "/" + this.bdd;
36     this.chargerPilote();
37
38     try {
39         this.maConnexion = DriverManager.getConnection(url,user,mdp);
40     } catch (SQLException e) {
41         System.out.println("impossible de se connecter à : " + url);
42     }
43 }
44
45 public void seDeconnecter() {
46     try {
47         if(this.maConnexion != null) {
48             this.maConnexion.close();
49         }
50     } catch (SQLException e) {
51         System.out.println("impossible de fermer la connexion");
52     }
53 }
54
55 public Connection getMaConnexion() {
56     return this.maConnexion;
57 }

```

## 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 php
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  }

```

```

288 public static void updateProprietaire(Proprietaire unProprietaire) {
289     // 1. Mise à jour de la table parente (Utilisateur)
290     String reqUser = "update utilisateur set nom = '" + unProprietaire.getNom() + "', "
291         + "prenom = '" + unProprietaire.getPrenom() + "', "
292         + "email = '" + unProprietaire.getEmail() + "', "
293         + "mdp = '" + unProprietaire.getMdp() + "', "
294         + "tel = '" + unProprietaire.getTel() + "' "
295         + "where id_user = " + unProprietaire.getId_user() + ";";
296
297     // 2. Mise à jour de la table fille (Client)
298     String reqProprietaire = "update proprietaire set adresse = '" + unProprietaire.getAdresse() + "', "
299         + "cp = '" + unProprietaire.getCp() + "', "
300         + "ville = '" + unProprietaire.getVille() + "', "
301         + "RIB = '" + unProprietaire.getRib() + "' "
302         + "where id_p = " + unProprietaire.getId_user() + ";";
303
304     try {
305         uneBdd.seConnecter();
306         Statement unStat = uneBdd.getMaConnexion().createStatement();
307
308         // On exécute les deux mises à jour
309         unStat.executeUpdate(reqUser);
310         unStat.executeUpdate(reqProprietaire);
311
312         unStat.close();
313         uneBdd.seDeconnecter();
314
315         System.out.println("Mise à jour réussie pour l'ID : " + unProprietaire.getId_user());
316
317     } catch (SQLException e) {
318         System.out.println("Erreur SQL lors de l'update : " + e.getMessage());
319     }
320 }

```

```

456 public static Habitation selectWhereHabitation(int ref_hab) {
457     Habitation uneHabitation = null;
458     String requete = "SELECT * FROM habitation WHERE ref_hab = " + ref_hab + ";";
459
460     try {
461         uneBdd.seConnecter();
462         Statement unStat = uneBdd.getMaConnexion().createStatement();
463         ResultSet rs = unStat.executeQuery(requete);
464
465         if (rs.next()) {
466             uneHabitation = new Habitation(
467                 rs.getInt("ref_hab"),
468                 rs.getString("type_hab"),
469                 rs.getString("adr_hab"),
470                 rs.getString("cp_hab"),
471                 rs.getString("ville_hab"),
472                 rs.getString("tarif_hab_bas"),
473                 rs.getString("tarif_hab_moy"),
474                 rs.getString("tarif_hab_hau"),
475                 rs.getString("surface"),
476                 rs.getInt("id_p"),
477                 rs.getString("description"),
478                 rs.getString("titre"),
479                 rs.getInt("capacite")
480             );
481         }
482
483     } catch (SQLException e) {
484         System.out.println("Erreur SELECT WHERE : " + e.getMessage());
485     }
486
487     return uneHabitation;
488 }

```

```

551 public static void deleteReservation(int refRes) {
552     String requete = "delete from reservation where ref_res ='" + refRes + "'";
553     executerRequete(requete);
554 }

```

```

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

```

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

```

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

```

57  /*** Admin ***/
58  public static Utilisateur selectWhereAdmin(String email, String mdp) {
59      Utilisateur unAdmin = null;
60      String requete = "select * from utilisateur where email ='" + email + "' and mdp ='" + mdp + "' and role = 'admin'";
61
62      try {
63          uneBdd.seConnecter();
64          //comme le prepare en php
65          Statement unStat = uneBdd.getMaConnexion().createStatement();
66          //comme execute en php
67          ResultSet unResultat = unStat.executeQuery(requete);
68          //comme le fetch
69          if(unResultat.next()) {
70              unAdmin = new Utilisateur(
71                  unResultat.getInt("Id_user"),
72                  unResultat.getString("nom"),
73                  unResultat.getString("prenom"),
74                  unResultat.getString("email"),
75                  unResultat.getString("mdp"),
76                  unResultat.getString("tel"),
77                  unResultat.getString("role")
78              );
79          }
80          uneBdd.seDeconnecter();
81      } catch (SQLException e) {
82          System.out.println("erreur requete : " + requete);
83      }
84      return unAdmin;
85  }

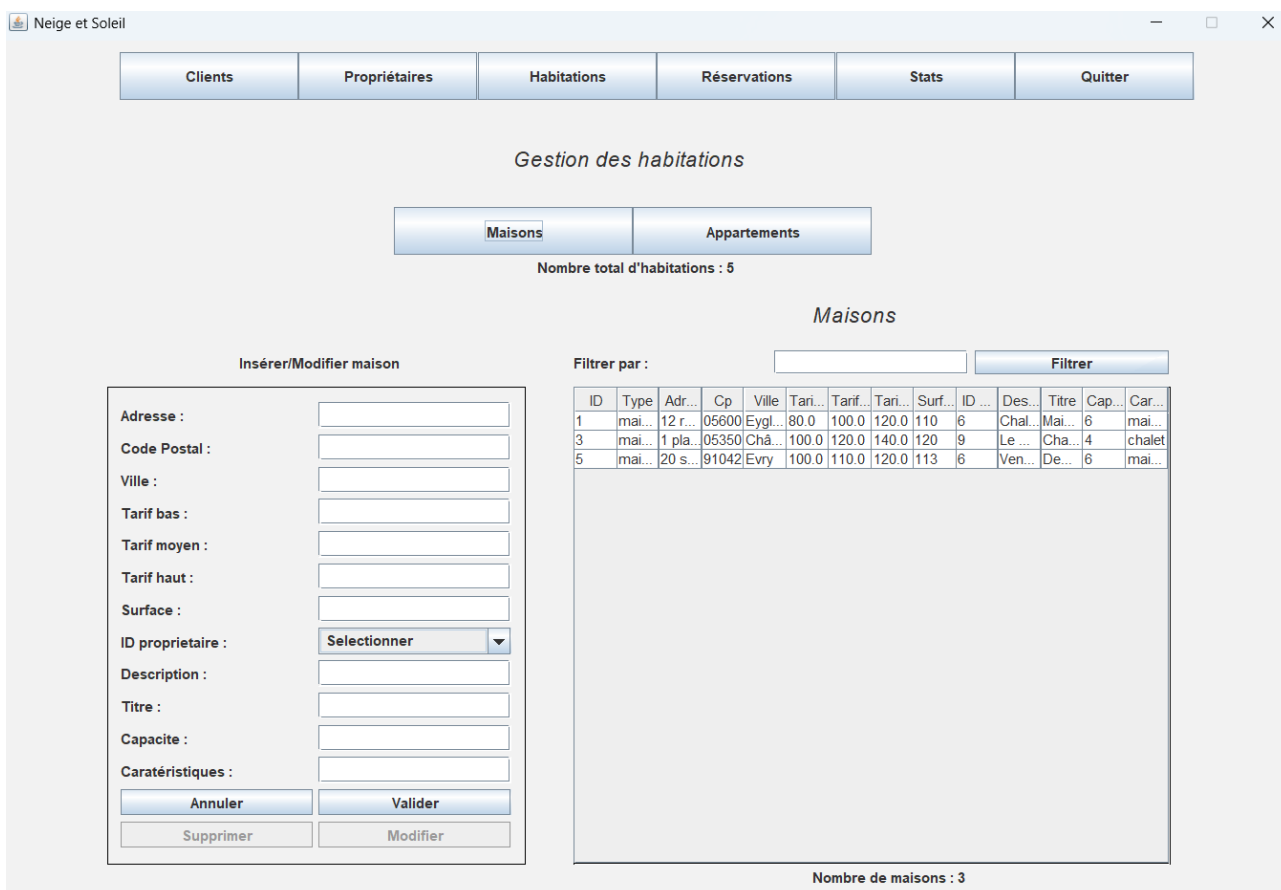
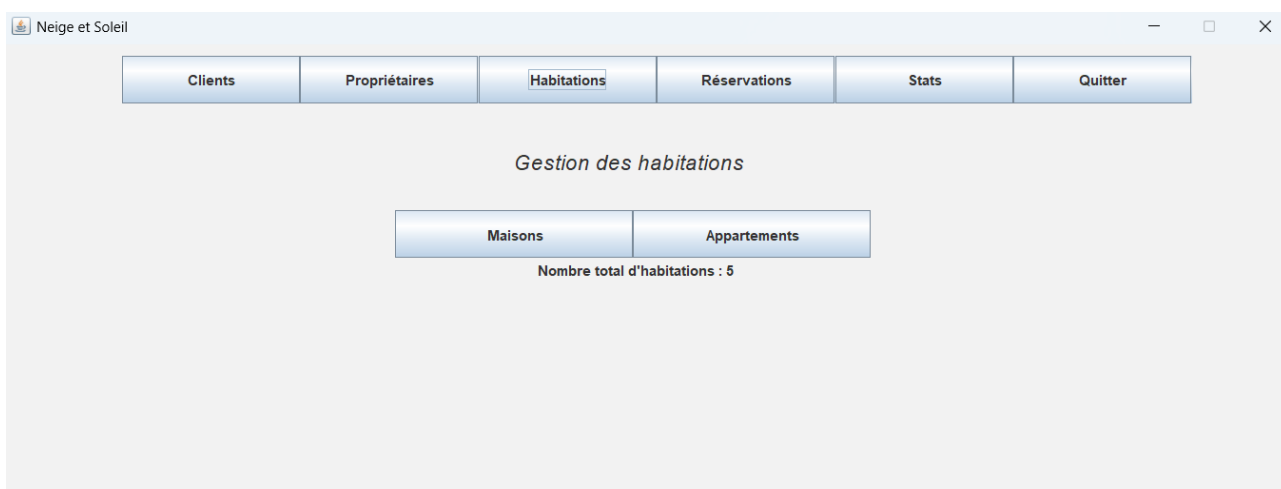
```

## 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 : caractéristique (jardin, piscine, etc.)
- Spécificités Appartement : etage, typeap (T1, T2, T3, etc.)
- Photos : gestion des images avec photo principale (is\_principal)



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 propriétaire : Selectionner ▼

Description :

Titre :

Capacité :

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.

**Insérer/Modifier réservation**

Date réservation :

Nombre personnes :

Début :

Fin :

Etat :

ID client :

Réf habitation :

**Filtrer par :**

Réf	Date rés...	Nombre ...	Début	Fin	Etat	ID client	Réf Habit...
4	2026-03-23	3	2026-05-28	2026-05-31	En attente	7	1
7	2026-03-27	2	2026-05-20	2026-06-06	En attente	12	2
8	2026-03-27	2	2026-06-06	2026-07-07	En attente	13	1
10	2026-04-01	2	2026-07-10	2026-07-12	En attente	7	2
11	2026-03-31	2	2026-05-29	2026-05-31	En attente	8	3

**Nombre de réservations : 5**

## 7.3 Gestion des Clients

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

**Insérer/Modifier client**

Nom :

Prenom :

Email :

Mdp :

Adresse :

Code postal :

Ville :

Tel :

RIB :

Filter par :

ID client	Nom	Prenom	Email	mdp	Adresse	Code p...	Ville	Tel	RIB
7	ZERR...	Salim	s@gm...	client	10 plac...	95100	Argent...	072298...	FR763...
8	MARX	Karl	mk@g...	client	1 place...	95100	Argent...	068891...	FR763...
10	JUAN	Carlos	jc@gm...	client	2 boule...	75018	Paris	068122...	FR763...
12	TIR	Farid	vugreg...	client	1 place...	95300	Pontoise	078289...	FR763...
13	LOPEZ	Maxime	max@...	client	1 clos...	95300	Pontoise	076612...	FR769...
15	NIAKA...	Tino	nt@gm...	client	10 plac...	93200	Saint...	078192...	FR763...
16	DIOP	Alpha	diopa...	client	19 rue...	60100	Creil	068199...	FR768...
22	NYIMA	Tachi	tachn...	client2	42 rue...	93200	Saint...	075517...	FR764...

Nombre de clients : 8

## 7.4 Gestion des Propriétaires

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

**Insérer/Modifier propriétaire**

Nom :

Prenom :

Email :

Mdp :

Adresse :

Code postal :

Ville :

Tel :

RIB :

Filter par :

ID	Nom	Prenom	Email	Mdp	Adresse	CP	Ville	Tel	RIB
6	BENZ...	Karim	kb9@g...	propriet...	2 rue d...	95000	Cergy	062278...	FR763...
9	DIALLO	Amine	da@g...	propriet...	1 rue r...	95200	Sarcell...	078890...	FR763...
11	LARIBI	Mehdi	jc95p...	propriet...	1 ru sai...	93600	aulnay...	076619...	FR763...
18	BAZIZ	Amine	mma...	propriet...	23 ave...	93200	Saint...	067129...	FR768...
26	DAVID	Desclos	iddexk...	propriet...	10 boul...	93100	saint o...	076152...	FR768...
27	MUST...	Moham...	mslo@...	propriet...	10 rue...	92100	colomb...	067781...	FR768...

Nombre de propriétaires : 6

## 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