Université Cadi Ayyad
École Supérieure De Technologie-Safi
Département : Informatique
Filière : GI

Rapport Du TP java 1 et 2
GI 2eme année

Gestion des employes et des conges

Réalisé par :

MAHFOUD Rayhana

enseigné par :

M.ELABDLLAOUI Said

encadrée par :

Mme.ELKOURCHI Asmae

Année Universitaire : 2024/2025

Table des matières

Liste des figures

Introduction

Dans ce projet, nous avons développé une application de gestion des ressources humaines destinée à faciliter la gestion des employés et des congés au sein d'une organisation. Cette application, construite selon les principes du modèle MVC (Modèle-Vue-Contrôleur), offre une interface utilisateur intuitive et réactive. En séparant les préoccupations entre les données (Modèle), l'interface utilisateur (Vue) et la logique de contrôle (Contrôleur), nous avons créé une architecture flexible et maintenable.

De plus, nous avons implémenté le modèle DAO (Data Access Object) pour gérer les interactions avec la base de données de manière structurée. Ce modèle permet de séparer la logique de persistance des données de la logique métier, facilitant ainsi les opérations de lecture, d'écriture, de mise à jour et de suppression des données des employés et des congés.

Chapter 1

Section 1: Code du projet

Ce chapitre présentera les codes de l'application et leurs description .

1.1 Controller

1.1.1 EmployeController.java

Le ‘EmployeController‘ gère les interactions entre la vue (‘MainView‘) et le modèle (‘EmployeModel‘) pour les opérations sur les employés. Il initialise les écouteurs d'événements pour les boutons d'interface utilisateur. Les principales fonctions incluent l'affichage des employés, l'ajout, la suppression et la mise à jour des informations d'employés. ‘displayEmploye()‘ met à jour la table avec la liste des employés. ‘addEmploye()‘ collecte les informations saisies et ajoute un nouvel employé. ‘deleteEmploye()‘ supprime l'employé sélectionné. ‘updateEmployebyselect()‘ récupère les informations de l'employé sélectionné pour modification, et ‘updateEmploye()‘ enregistre les modifications. De plus, ‘resetSolde()‘ réinitialise le solde des congés au début de chaque année. L'ensemble de ces méthodes permet une gestion efficace des employés via une interface graphique intuitive.

```
1
2  package Controller;
3
4  import Model.*;
5  import View.*;
6
7  import java.util.Calendar;
8  import java.util.List;
9
10 import javax.swing.table.DefaultTableModel;
11
12
13
14 public class EmployeController {
15
16     private final MainView View;
17     public static EmployeModel model_employe ;
18     public static int id = 0;
19     public static int oldselectedrow = -1;
20     public static boolean test = false;
21     String nom = "";
22     String prenom = "";
23     String email = "";
24     String telephone = "";
25     double salaire = 0;
26     Role role = null;
27     Poste poste = null;
28     int solde = 0;
29     boolean updatereussi = false;
30
31     public EmployeController(MainView view, EmployeModel model) {
32         this.View = view;
33         this.model_employe = model;
```

```
34
35          View.getaddButton_employe().addActionListener(e -> addEmploye());
36          View.getdeleteButton_employe().addActionListener(e -> deleteEmploye());
37          View.getupdateButton_employe().addActionListener(e -> updateEmploye());
38          View.getdisplayButton_employe().addActionListener(e -> displayEmploye()
    );
39          MainView.Tableau.getSelectionModel().addListSelectionListener(e ->
    updateEmployebyselect());
40      }
41
42
43
44      public void displayEmploye() {
45          List<Employe> Employes = model_employe.displayEmploye();
46          if(Employes.isEmpty()){
47              View.afficherMessageErreur("Aucun employe.");
48          }
49          DefaultTableModel tableModel = (DefaultTableModel) MainView.Tableau.
    getModel();
50          tableModel.setRowCount(0);
51          for(Employe e : Employes){
52              tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom()
    , e.getEmail(), e.getTelephone(), e.getSalaire(), e.getRole(), e.getPoste(),
    e.getSolde()});
53          }
54          View.remplaire_les_employes();
55      }
56
57
58      // function of add Employe :
59
60      private void addEmploye() {
61          String nom = View.getNom();
62          String prenom = View.getPrenom();
63          String email = View.getEmail();
64          String telephone = View.getTelephone();
65          double salaire = View.getSalaire();
66          Role role = View.getRole();
67          Poste poste = View.getPoste();
68
69          View.viderChamps_em();
70          boolean addreussi = model_employe.addEmploye(0,nom, prenom, email,
    telephone, salaire, role, poste ,25);
71
72          if(addreussi == true){
73              View.afficherMessageSucces("L'employe a bien ete ajoutee.");
74              displayEmploye();
75          }else{
76              View.afficherMessageErreur("L'employe n'a pas ete ajoutee.");
77          }
78      }
79
80
81
82      // function of delete Employe :
83
84      private void deleteEmploye(){
85          int selectedrow = MainView.Tableau.getSelectedRow();
86          if(selectedrow == -1){
87              View.afficherMessageErreur("Veuillez selectionner une ligne.");
88          }else{
89              int id = (int) MainView.Tableau.getValueAt(selectedrow, 0);
90              if(model_employe.deleteEmploye(id)){
```

```
91                View.afficherMessageSucces("L'employe a bien ete supprimer.");
92                displayEmploye();
93            }else{
94                View.afficherMessageErreur("L'employe n'a pas ete supprimer.");
95            }
96        }
97    }
98
99    // function of Update :
100
101   private void updateEmployebyselect(){
102       int selectedrow = MainView.Tableau.getSelectedRow();
103
104       if (selectedrow == -1) {
105           return;
106       }
107       try{
108           id = (int) MainView.Tableau.getValueAt(selectedrow, 0);
109           nom = (String) MainView.Tableau.getValueAt(selectedrow, 1);
110           prenom = (String) MainView.Tableau.getValueAt(selectedrow, 2);
111           email = (String) MainView.Tableau.getValueAt(selectedrow, 3);
112           telephone = (String) MainView.Tableau.getValueAt(selectedrow, 4);
113           salaire = (double) MainView.Tableau.getValueAt(selectedrow, 5);
114           role = (Role) MainView.Tableau.getValueAt(selectedrow, 6);
115           poste = (Poste) MainView.Tableau.getValueAt(selectedrow, 7);
116           solde = (int) MainView.Tableau.getValueAt(selectedrow, 8);
117           View.remplaireChamps_em(id, nom, prenom, email, telephone, salaire,
    role, poste);
118           test = true;
119       }catch(Exception e){
120            View.afficherMessageErreur("Erreur lors de la recuperation des
    donnees");
121       }
122   }
123
124   private void updateEmploye(){
125       if (!test) {
126           View.afficherMessageErreur("Veuillez d'abord selectionner une ligne
      modifier.");
127           return;
128       }
129       try {
130           nom = View.getNom();
131           prenom = View.getPrenom();
132           email = View.getEmail();
133           telephone = View.getTelephone();
134           salaire = View.getSalaire();
135           role = View.getRole();
136           poste = View.getPoste();
137
138           boolean updateSuccessful = model_employe.updateEmploye(id, nom,
    prenom, email, telephone, salaire, role, poste , solde);
139
140           if (updateSuccessful) {
141               test = false;
142               View.afficherMessageSucces("L'employe a est modifie avec succes
    .");
143               displayEmploye();
144               View.viderChamps_em();
145           } else {
146               View.afficherMessageErreur("Erreur lors de la mise a jour de l'
    employe.");
147           }
```

```
148         } catch (Exception e) {
149
150             View.afficherMessageErreur("Erreur lors de la mise a jour");
151         }
152     }
153
154     public void resetSolde(){
155         Calendar now = Calendar.getInstance();
156         if(now.get(Calendar.DAY_OF_YEAR) == 1){
157             for (Employe employe : model_employe.displayEmploye()) {
158                 updateSolde(employe.getId(), 25);
159             }
160         }
161     }
162
163     public static void updateSolde(int  id , int solde){
164         boolean updateSuccessful = model_employe.updateSolde(id, solde);
165     }
166
167 }
```

Listing 1.1: EmployeController

1.1.2 HolidayController.java

Le ‘HolidayController‘ gère les opérations liées aux congés des employés via l'interface ‘MainView‘. Il initialise les écouteurs pour les boutons d'ajout, suppression, mise à jour et affichage des congés. La méthode ‘addHoliday()‘ vérifie la validité des dates et le solde des congés avant d'ajouter un nouveau congé. ‘displayHoliday()‘ met à jour la table avec la liste des congés. ‘deleteHoliday()‘ supprime le congé sélectionné et ajuste le solde de l'employé. ‘updateHolidaybyselect()‘ récupère les informations du congé sélectionné pour modification, et ‘updateHoliday()‘ enregistre les modifications. Ces fonctionnalités assurent une gestion fluide des congés via une interface intuitive.

```
1  package Controller;
2
3  import DAO.EmployeDAOImpl;
4  import Model.*;
5  import View.*;
6  import java.sql.Date;
7  import java.util.List;
8  import javax.swing.table.DefaultTableModel;
9
10
11 public class HolidayController {
12
13     private final MainView View;
14     public HolidayModel model_holiday;
```

```
15      public static int id = 0;
16      public static int oldselectedrow = -1;
17      public static boolean test = false;
18      int id_employe = 0;
19      String nom_employe = "";
20      public static String OldstartDate = null;
21      public static String OldendDate = null;
22      HolidayType type = null;
23      int oldsolde = 0;
24      int solde = 0;
25      boolean updatereussi = false;
26      Employe targetEmploye = null;
27
28      public HolidayController(MainView view, HolidayModel model) {
29          this.View = view;
30          this.model_holiday= model;
31
32          View.getdeleteButton_holiday().addActionListener(e -> deleteHoliday());
33          View.getupdateButton_holiday().addActionListener(e -> updateHoliday());
34          MainView.Tableau1.getSelectionModel().addListSelectionListener(e ->
    updateHolidaybyselect());
35          View.getaddButton_holiday().addActionListener(e -> addHoliday());
36          View.getdisplayButton_holiday().addActionListener(e -> displayHoliday()
    );
37      }
38
39      private void addHoliday() {
40          int id_employe = View.getId_employe();
41          Date startDate = Date.valueOf(View.getStartDate());
42          Date endDate = Date.valueOf(View.getEndDate());
43          HolidayType type = View.getHolidayType();
44
45          View.viderChamps_ho();
46
47          Employe targetEmploye = null;
48
49          // Rechercher l'employe correspondant
50          for (Employe employe : new EmployeModel(new EmployeDAOImpl()).
    displayEmploye()) {
51              if (employe.getId() == id_employe) {
52                  targetEmploye = employe;
53                  break;
54              }
55          }
56
57          if (targetEmploye == null) {
58              View.afficherMessageErreur("Cet employe n'existe pas.");
59              return;
60          }
61
62          // Calculer la dur e du cong  demand
63          long daysBetween = java.time.temporal.ChronoUnit.DAYS.between(
64              startDate.toLocalDate(),
65              endDate.toLocalDate()
66          );
67
68          if (daysBetween <= 0) {
69              View.afficherMessageErreur("Les dates de d but et de fin sont
    invalides.");
70              return;
71          }
72
73          // V rifier si le solde de cong  est suffisant
```

```
74          if (targetEmploye.getSolde() < daysBetween) {
75              View.afficherMessageErreur("Le solde de cong  de l'employ  est
    insuffisant.");
76              return;
77          }
78
79          // V rification de chevauchement des dates de cong  existantes pour l
    'employ
80          for (Holiday existingHoliday : model_holiday.displayHoliday()) {
81              if (existingHoliday.getId_employe() == id_employe) {
82                  Date existingStartDate = existingHoliday.getStartDate();
83                  Date existingEndDate = existingHoliday.getEndDate();
84
85                  // V rification si les dates se chevauchent
86                  if ((startDate.before(existingEndDate) && endDate.after(
    existingStartDate))) {
87                      View.afficherMessageErreur("Le cong  se chevauche avec une
     autre p riode de cong .");
88                      return;
89                  }
90              }
91          }
92
93          try {
94              // Ajouter la demande de cong
95              boolean addReussi = model_holiday.addHoliday(0, id_employe,
    startDate, endDate, type, targetEmploye);
96
97              if (addReussi) {
98                  // R duire le solde de cong  apr s l'ajout r ussi
99                  targetEmploye.setSolde(targetEmploye.getSolde() - (int)
    daysBetween);
100                 View.afficherMessageSucces("Holiday est ajout e.");
101             }
102         } catch (Exception e) {
103             e.printStackTrace();
104             View.afficherMessageErreur("Erreur lors de l'ajout : " + e.
    getMessage());
105         }
106     }
107
108     private void displayHoliday() {
109         List<Holiday> Holidays = model_holiday.displayHoliday();
110
111         if (Holidays == null || Holidays.isEmpty()) {
112             View.afficherMessageErreur("Aucune holiday.");
113             return; // Retourner pour ne pas continuer l'execution si la liste
    est vide
114         }
115
116         DefaultTableModel tableModel1 = (DefaultTableModel) MainView.Tableau1.
    getModel();
117         tableModel1.setRowCount(0); // Clear existing rows in the table
118
119         for (Holiday e : Holidays) {
120             String nom_employe = null;
121             List<Employe> Employes = new EmployeModel(new EmployeDAOImpl()).
    displayEmploye();
122             for (Employe em : Employes) {
123                 if (em.getId() == e.getId_employe()) {
124                     nom_employe = em.getId() + " - " + em.getNom() + " " + em.
    getPrenom();
125                     break;
```

```
126                 }
127             }
128             // Ajout de la ligne dans le tableau
129             tableModel1.addRow(new Object[]{e.getId_holiday(), nom_employe, e.
    getStartDate(), e.getEndDate(), e.getType()});
130         }
131         View.remplaire_les_employes();
132 
133     }
134 
135 
136     private void deleteHoliday(){
137         int selectedrow = MainView.Tableau1.getSelectedRow();
138         if(selectedrow == -1){
139             View.afficherMessageErreur("Veuillez selectionner une ligne.");
140         }else{
141             int id = (int) MainView.Tableau1.getValueAt(selectedrow, 0);
142             int id_employe = Integer.parseInt((MainView.Tableau1.getValueAt(
    selectedrow, 1)).toString().split(" - ")[0]);
143             int olddaysbetween =  (int) ( (Date.valueOf(OldendDate).toLocalDate
    ().toEpochDay() - Date.valueOf(OldstartDate).toLocalDate().toEpochDay()));
144             for(Employe e : new EmployeModel(new EmployeDAOImpl()).
    displayEmploye()){
145                 if(e.getId() == id_employe){
146                     solde = e.getSolde();
147                     break;
148                 }
149             }
150             EmployeController.updateSolde(id_employe,solde+olddaysbetween);
151             boolean deletereussi = model_holiday.deleteHoliday(id);
152             if(deletereussi){
153                 View.afficherMessageSucces("Holiday est supprimer.");
154                 displayHoliday();
155             }else{
156                 View.afficherMessageErreur("Holiday n'est pas supprimer.");
157             }
158         }
159     }
160 
161     private void updateHolidaybyselect(){
162         int selectedrow = MainView.Tableau1.getSelectedRow();
163 
164         if (selectedrow == -1) {
165             return;
166         }
167         try{
168             id = (int) MainView.Tableau1.getValueAt(selectedrow, 0);
169             nom_employe = (String) MainView.Tableau1.getValueAt(selectedrow, 1)
    ;
170             id_employe = Integer.parseInt(nom_employe.split(" - ")[0]);
171             OldstartDate = String.valueOf(MainView.Tableau1.getValueAt(
    selectedrow, 2));
172             OldendDate = String.valueOf(MainView.Tableau1.getValueAt(
    selectedrow, 3));
173             type = (HolidayType) MainView.Tableau1.getValueAt(selectedrow, 4);
174             View.remplaireChamps_ho(id_employe, OldstartDate, OldendDate, type)
    ;
175             test = true;
176         }catch(Exception e){
177              View.afficherMessageErreur("Erreur lors de la recuperation des
    donnees");
178         }
179     }
```

```
180
181
182     private void updateHoliday(){
183         if (!test) {
184             View.afficherMessageErreur("Veuillez d'abord selectionner une ligne
        a modifier.");
185             return;
186         }
187         try {
188             nom_employe = View.getNom();
189             Date startDate_holiday = Date.valueOf(View.getStartDate());
190             Date endDate_holiday = Date.valueOf(View.getEndDate());
191             type = View.getHolidayType();
192             id_employe = View.getId_employe();
193
194             int olddaysbetween =  (int) ( (Date.valueOf(OldendDate).toLocalDate
    ().toEpochDay() - Date.valueOf(OldstartDate).toLocalDate().toEpochDay()));
195
196
197             for (Employe employe : new EmployeModel(new EmployeDAOImpl()).
    displayEmploye()) {
198                 if (employe.getId() == id_employe) {
199                     targetEmploye = employe;
200                     break;
201                 }
202             }
203
204             boolean updateSuccessful = model_holiday.updateHoliday(id,
    id_employe, startDate_holiday, endDate_holiday, type , targetEmploye ,
    olddaysbetween);
205
206             if (updateSuccessful) {
207                 test = false;
208                 View.afficherMessageSucces("Holiday est modifie avec succes.");
209                 displayHoliday();
210                 View.viderChamps_ho();
211             } else {
212                 View.afficherMessageErreur("Erreur lors de la mise a jour de
    holiday.");
213             }
214         } catch (Exception e) {
215
216             View.afficherMessageErreur("Erreur lors de la mise a jour");
217         }
218     }
219 }
```

Listing 1.2: HolidayController

1.2 DAO

1.2.1 DBconnexion.java

La classe 'DBConnexion' gère la connexion à la base de données MySQL pour l'application de gestion des ressources humaines. Elle utilise un modèle Singleton pour garantir qu'une

seule instance de connexion est créée et réutilisée tout au long de l'application. La méthode 'getConnexion()' initialise et retourne une connexion à la base de données en utilisant le pilote JDBC de MySQL. Si le pilote JDBC n'est pas trouvé ou si une erreur de connexion se produit, une exception est levée et un message d'erreur est imprimé dans la console.

```
package DAO;

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

public class DBConnexion {
  private static final String URL="jdbc:mysql://localhost:3306/tp1.2";
  private static final String user="root";
    private static final String password="";
    static Connection conn= null; //possibilit  d'erreur
    public static Connection getConnexion() {


      try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        conn = DriverManager.getConnection(URL, user, password);
      }catch (ClassNotFoundException |SQLException e) {
        e.printStackTrace();
        throw new RuntimeException("ERREUR DE CONN");
      }

    return conn ;
    }
}
```

Listing 1.3: DBconnexion

1.2.2 EmployeDAOI.java

L'interface EmployeDAOI définit les méthodes essentielles pour gérer les opérations CRUD (Create, Read, Update, Delete) sur les objets Employe.

```
package DAO;

import Model.Employe;
import java.util.List;

public interface EmployeDAOI {
    void add(Employe employee); // Ajouter un employ
    void delete(int id); // Supprimer un employ
    List<Employe> listAll(); // Lister tous les employ s
    Employe findById(int id); // Trouver un employ  par ID
```

```
11      void update(Employe employee, int id); // Mettre    jour un employ
12  }
```

Listing 1.4: EmployeDAOI

1.2.3 EmployeDAOImpl.java

La classe EmployeDAOImpl implémente l'interface GenericDAOI<Employe> pour gérer les opérations CRUD sur les objets Employe. Elle utilise des PreparedStatement pour interagir avec la base de données MySQL via la classe DBConnexion.

–add(Employe e) : Ajoute un nouvel employé à la base de données en insérant les données fournies, sauf l'ID qui est auto-incrémenté.

–delete(int id) : Supprime un employé de la base de données en utilisant son ID.

–update(Employe e) : Met à jour les informations d'un employé existant en fonction de son ID.

–display() : Retourne une liste de tous les employés présents dans la base de données.

–updateSolde(int id, int solde) : Met à jour le solde de congé d'un employé spécifique en fonction de son ID.

Chaque méthode gère les exceptions SQL et imprime des messages d'erreur en cas de problème.

```
1  package DAO;
2
3  package DAO;
4  import Model.Employe;
5  import Model.Poste;
6  import Model.Role;
7  import java.sql.PreparedStatement;
8  import java.sql.ResultSet;
9  import java.sql.SQLException;
10 import java.util.ArrayList;
11 import java.util.List;
12
13
14 public class EmployeDAOImpl implements GenericDAOI<Employe> {
15
16     @Override
17     public void add(Employe e) {
18         // Do not include the 'id' column in the INSERT statement because it is
       AUTO_INCREMENT
19         String sql = "INSERT INTO employe (nom, prenom, email, telephone,
       salaire, role, poste, solde) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
20         try (PreparedStatement stmt = DBConnexion.getConnexion().
       prepareStatement(sql)) {
21             stmt.setString(1, e.getNom());
22             stmt.setString(2, e.getPrenom());
```

```
23              stmt.setString(3, e.getEmail());
24              stmt.setString(4, e.getTelephone());
25              stmt.setDouble(5, e.getSalaire());
26              stmt.setString(6, e.getRole().name());  // Assuming Role is an enum
     and needs to be converted to string
27              stmt.setString(7, e.getPoste().name());  // Assuming Post is an
    enum and needs to be converted to string
28              stmt.setInt(8, e.getSolde());
29              stmt.executeUpdate();
30          } catch (SQLException exception) {
31              System.err.println("Failed to add employee: " + exception.
    getMessage());
32              exception.printStackTrace();  // Prints the full stack trace for
    debugging
33          }
34      }
35
36      @Override
37      public void delete(int id) {
38          String sql = "DELETE FROM employe WHERE id = ?";
39          try (PreparedStatement stmt = DBConnexion.getConnexion().
    prepareStatement(sql)) {
40              stmt.setInt(1,id);
41              stmt.executeUpdate();
42          } catch (SQLException exception) {
43              System.err.println("failed of delete employe");
44          }
45      }
46
47      @Override
48      public void update(Employe e) {
49          String sql = "UPDATE employe SET nom = ?, prenom = ?, email = ?,
    telephone = ?, salaire = ?, role = ?, poste = ? WHERE id = ?";
50          try (PreparedStatement stmt = DBConnexion.getConnexion().
    prepareStatement(sql)) {
51              stmt.setString(1, e.getNom());
52              stmt.setString(2, e.getPrenom());
53              stmt.setString(3, e.getEmail());
54              stmt.setString(4, e.getTelephone());
55              stmt.setDouble(5, e.getSalaire());
56              stmt.setString(6, e.getRole().name());
57              stmt.setString(7, e.getPoste().name());
58              stmt.setInt(8,e.getId());
59              stmt.executeUpdate();
60          } catch (SQLException exception) {
61              System.err.println("failed of update employe");
62          }
63      }
64      @Override
65      public List<Employe> display() {
66          String sql = "SELECT * FROM employe";
67          List<Employe> Employes = new ArrayList<>();
68          try (PreparedStatement stmt = DBConnexion.getConnexion().
    prepareStatement(sql)) {
69              ResultSet re = stmt.executeQuery();
70              while (re.next()) {
71                  int id = re.getInt("id");
72                  String nom = re.getString("nom");
73                  String prenom = re.getString("prenom");
74                  String email = re.getString("email");
75                  String telephone = re.getString("telephone");
76                  double salaire = re.getDouble("salaire");
77                  String role = re.getString("role");
```

```
78                  String poste = re.getString("poste");
79                  int solde = re.getInt("solde");
80                  Employe e = new Employe(id,nom, prenom, email, telephone,
    salaire, Role.valueOf(role), Poste.valueOf(poste),solde);
81                  Employes.add(e);
82              }
83              return Employes;
84          } catch (SQLException ex) {
85              System.err.println("failed of display employe");
86              return null;
87          }
88      }
89
90
91      public void updateSolde(int id, int solde) {
92          String sql = "UPDATE employe SET solde = ? WHERE id = ?";
93          try (PreparedStatement stmt = DBConnexion.getConnexion().
    prepareStatement(sql)) {
94              stmt.setInt(1, solde);
95              stmt.setInt(2, id);
96              stmt.executeUpdate();
97          } catch (SQLException exception) {
98              System.err.println("failed of update solde employe");
99          }
100     }
101
102 }
```

Listing 1.5: EmployeDAOImpl

1.2.4 GenericDAOI.java

GenericDAOI<T> est une interface générique définissant les opérations CRUD de base pour tout type T

```
1 package DAO;
2
3 import java.util.List;
4 public interface GenericDAOI <T> {
5     public void add(T e);
6     public void delete(int id);
7     public void update(T e);
8     public List<T> display();
9 }
```

Listing 1.6: GenericDAOI

1.2.5 HolidayDAOImpl.java

HolidayDAOImpl implémente l'interface GenericDAOI<Holiday> pour gérer les opérations CRUD sur les objets Holiday. Elle utilise des PreparedStatement pour interagir avec la base de données via DBConnexion.

–add(Holiday e) : Vérifie le solde de congé et ajoute une nouvelle demande de congé. Met à jour le solde de l'employé si la demande est valide.

–delete(int id) : Supprime une demande de congé par ID.

–update(Holiday e) : Met à jour les informations d'une demande de congé existante.

–display() : Retourne une liste de toutes les demandes de congé.

Les méthodes gèrent les exceptions SQL et affichent des messages d'erreur en cas de problème.

```java
package DAO;
import Model.Holiday;
import Model.HolidayType;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class HolidayDAOImpl implements GenericDAOI<Holiday>{

    @Override
    public void add(Holiday e) {
        String checkSoldeSql = "SELECT solde FROM employe WHERE id = ?";
        String insertHolidaySql = "INSERT INTO holiday (id_employe, startdate, enddate, type) VALUES (?, ?, ?, ?)";

        try (PreparedStatement checkStmt = DBConnexion.getConnexion().prepareStatement(checkSoldeSql)) {
            // Recuperer le solde de conge de l'employe
            checkStmt.setInt(1, e.getId_employe());
            ResultSet rs = checkStmt.executeQuery();

            if (rs.next()) {
                int solde = rs.getInt("solde");

                // Calculer le nombre de jours demandes
                long daysBetween = java.time.temporal.ChronoUnit.DAYS.between(
                    e.getStartDate().toLocalDate(),
                    e.getEndDate().toLocalDate()
                );

                if (daysBetween > solde) {
                    System.err.println("Le solde de conge est insuffisant.");
                    return;
```

```
35                 }
36
37                 // Inserer la demande de conge
38                 try (PreparedStatement insertStmt = DBConnexion.getConnexion().
   prepareStatement(insertHolidaySql)) {
39                     insertStmt.setInt(1, e.getId_employe());
40                     insertStmt.setDate(2, e.getStartDate());
41                     insertStmt.setDate(3, e.getEndDate());
42                     insertStmt.setString(4, e.getType().name());
43
44                     insertStmt.executeUpdate();
45
46                     // Mettre a jour le solde de conge
47                     String updateSoldeSql = "UPDATE employe SET solde= solde -
   ? WHERE id = ?";
48                     try (PreparedStatement updateStmt = DBConnexion.
   getConnexion().prepareStatement(updateSoldeSql)) {
49                         updateStmt.setInt(1, (int) daysBetween);
50                         updateStmt.setInt(2, e.getId_employe());
51                         updateStmt.executeUpdate();
52                     }
53                 }
54             } else {
55                 System.err.println("Employe introuvable.");
56             }
57         } catch (SQLException exception) {
58             exception.printStackTrace();
59         }
60     }
61
62
63     @Override
64     public void delete(int id) {
65         String sql = "DELETE FROM holiday WHERE id = ?";
66         try (PreparedStatement stmt = DBConnexion.getConnexion().
   prepareStatement(sql)) {
67             stmt.setInt(1,id);
68             stmt.executeUpdate();
69         } catch (SQLException exception) {
70             System.err.println("failed of delete holiday");
71         }
72     }
73
74     @Override
75     public void update(Holiday e) {
76         String sql = "UPDATE holiday SET id_employe = ?, startdate = ?, enddate
    = ?, type = ? WHERE id = ?";
77         try (PreparedStatement stmt = DBConnexion.getConnexion().
   prepareStatement(sql)){
78             stmt.setInt(1, e.getId_employe());
79             stmt.setDate(2, e.getStartDate());
80             stmt.setDate(3, e.getEndDate());
81             stmt.setString(4, e.getType().name());
82             stmt.setInt(5,e.getId_holiday());
83             stmt.executeUpdate();
84         } catch (SQLException exception) {
85             System.err.println("failed of update holiday");
86         }
87     }
88
89     @Override
90     public List<Holiday> display() {
91         String sql = "SELECT * FROM holiday";
```

```
92          List<Holiday> Holidays = new ArrayList<>();
93          try (PreparedStatement stmt = DBConnexion.getConnexion().
    prepareStatement(sql)) {
94              ResultSet re = stmt.executeQuery();
95              while (re.next()) {
96                  int id = re.getInt("id");
97                  int id_employe = re.getInt("id_employe");
98                  Date startdate = re.getDate("startdate");
99                  Date enddate = re.getDate("enddate");
100                 String type = re.getString("type");
101                 Holiday e = new Holiday(id, id_employe, startdate, enddate,
    HolidayType.valueOf(type));
102                 Holidays.add(e);
103             }
104         } catch (SQLException ex) {
105             System.err.println("Failed to fetch holidays: " + ex.getMessage());
106         }
107         return Holidays; // Retourne une liste vide si une erreur se produit
108     }
109
110 }
```

Listing 1.7: HolidayDAOImpl

1.3 Main

1.3.1 Main.java

La classe ‘Main‘ initialise et lie les composants principaux de l’application de gestion des ressources humaines. Elle crée les instances de la vue (‘MainView‘), des implémentations DAO (‘EmployeDAOImpl‘, ‘HolidayDAOImpl‘), et des modèles (‘EmployeModel‘, ‘HolidayModel‘). Ensuite, elle initialise les contrôleurs (‘EmployeController‘, ‘HolidayController‘) en leur passant les instances de la vue et des modèles. Cette classe sert de point d’entrée pour l’application, orchestrant la configuration initiale et démarrant l’interaction entre les différentes couches (vue, modèle, contrôleur).

```
1  package Main;
2
3  import Controller.*;
4  import DAO.*;
5  import Model.*;
6  import View.*;
7
8  public class Main {
9
10     public static void main(String[] args) {
11         MainView view = new MainView();
```

```
        EmployeDAOImpl dao = new EmployeDAOImpl();
        HolidayDAOImpl dao_holiday = new HolidayDAOImpl();
        EmployeModel model_employe = new EmployeModel(dao);
        HolidayModel model_holiday = new HolidayModel(dao_holiday);
        new EmployeController(view, model_employe);
        new HolidayController(view, model_holiday);
    }
}
```

Listing 1.8: Main

1.4 Model

1.4.1 Employe.java

La classe Employe représente un employé avec plusieurs attributs : id, nom, prenom, email, telephone, salaire, role, poste et solde. Le constructeur initialise ces attributs. Elle fournit des méthodes getter et setter pour chacun des attributs, permettant de récupérer et de modifier les valeurs des attributs d'un employé. Cette classe encapsule toutes les informations nécessaires pour gérer un employé dans l'application de gestion des ressources humaines.

```
package Model;

public class Employe{
    private int id;
    private String nom;
    private String prenom;
    private String email;
    private String telephone;
    private double salaire;
    private Role role;
    private Poste poste ;
    private int solde ;

    public Employe(int id ,String nom, String prenom, String email, String
    telephone, double salaire, Role role, Poste poste ,int solde){
        this.id = id;
        this.nom = nom;
        this.prenom = prenom;
        this.email = email;
        this.telephone = telephone;
        this.salaire = salaire;
        this.role = role;
        this.poste = poste;
        this.solde = solde;
    }

```

```
26
27      public int getId(){
28          return id;
29      }
30
31      public void setId(int id){
32          this.id = id;
33      }
34      public String getNom() {
35          return nom;
36      }
37
38      public void setNom(String nom) {
39          this.nom = nom;
40      }
41
42      public String getPrenom() {
43          return prenom;
44      }
45
46      public void setPrenom(String prenom) {
47          this.prenom = prenom;
48      }
49
50      public String getEmail() {
51          return email;
52      }
53
54      public void setEmail(String email) {
55          this.email = email;
56      }
57
58      public String getTelephone() {
59          return telephone;
60      }
61
62      public void setTelephone(String telephone) {
63          this.telephone = telephone;
64      }
65
66      public double getSalaire() {
67          return salaire;
68      }
69
70      public void setSalaire(double salaire) {
71          this.salaire = salaire;
72      }
73
74      public Role getRole() {
75          return role;
76      }
77
78      public void setRole(Role role) {
79          this.role = role;
80      }
81
82      public Poste getPoste() {
83          return poste;
84      }
85
86      public void setPost(Poste poste) {
87          this.poste = poste;
88      }
```

```
89
90      public void setSolde (int conge){
91          this.solde = conge;
92      }
93
94      public int getSolde(){
95          return solde;
96      }
97  }
```

Listing 1.9: Employe

1.4.2 EmployeModel.java

EmployeModel interagit avec EmployeDAOImpl pour gérer les employés. Elle inclut :

–addEmploye() : Valide les données, crée un nouvel employé et l'ajoute via le DAO.

–deleteEmploye(int id) : Supprime un employé par ID.

–updateEmploye() : Met à jour les informations d'un employé existant.

–updateSolde(int id, int solde) : Met à jour le solde de congé d'un employé.

–displayEmploye() : Retourne une liste de tous les employés.

Cette classe assure la validation des données avant de déléguer les opérations au DAO.

```
1  package Model;
2  import DAO.EmployeDAOImpl;
3  import java.util.List;
4
5  public class EmployeModel {
6      private EmployeDAOImpl dao;
7      public EmployeModel(EmployeDAOImpl dao) {
8          this.dao = dao;
9      }
10
11     // funtion of add Employe :
12
13     public boolean addEmploye(int id ,String nom, String prenom, String email,
       String telephone, double salaire, Role role, Poste poste, int solde) {
14         if(salaire < 0 ){
15             System.out.println("Erreur : le salaire doit etre positif.");
16             return false;
17         }
18         if(id < 0 ){
19             System.out.println("Erreur : l'id doit etre positif.");
20             return false;
21         }
22         if(telephone.length() != 10){
23             System.out.println("Erreur : le telephone doit etre 10 num.");
24             return false;
```

```
25          }
26          if(!email.contains("@")){
27              System.out.println("Erreur : le mail doit contenir le @.");
28              return false;
29          }
30
31          Employe e = new Employe(id,nom, prenom, email, telephone, salaire, role
    , poste ,solde);
32
33          dao.add(e);
34
35          return true;
36      }
37
38      // function of delete Employe :
39
40      public  boolean deleteEmploye(int id){
41          dao.delete(id);
42          return true;
43      }
44
45      // function of update Employe :
46
47      public boolean updateEmploye(int id, String nom, String prenom, String
    email, String telephone, double salaire, Role role, Poste poste , int solde)
     {
48
49          Employe e = new Employe(id,nom, prenom, email, telephone, salaire, role
    , poste,solde);
50          dao.update(e);
51          return true;
52      }
53
54      //function of update solde Employe :
55
56      public boolean updateSolde(int id, int solde) {
57          dao.updateSolde(id, solde);
58          return true;
59      }
60
61      //function of display Employe :
62
63      public List<Employe> displayEmploye() {
64          List<Employe> Employes = dao.display();
65          return Employes;
66      }
67  }
```

Listing 1.10: EmployeModel

1.4.3 Holiday.java

La classe Holiday représente une demande de congé d'un employé. Elle comprend les attributs suivants :

–id$_h$*oliday* : $L'identifiantuniquedelademandedecong$.

–id$_e$*mploye* : $L'identifiantdel'employquidemandelecong$.

–startDate : La date de début du congé.

–endDate : La date de fin du congé.

–type : Le type de congé (par exemple, payé, non payé).

Le constructeur initialise ces attributs, et la classe fournit des méthodes getter pour chacun d'eux, permettant de récupérer les valeurs des attributs d'une demande de congé. La méthode getSolde() n'est pas supportée et lève une exception si elle est appelée.

Cette classe encapsule toutes les informations nécessaires pour gérer une demande de congé dans le système de gestion des ressources humaines.

```
1  package Model;
2
3  import java.sql.Date;
4
5
6  public class Holiday{
7      private int id_holiday;
8      private int id_employe;
9      private Date startDate;
10     private Date endDate;
11     private HolidayType type;
12
13 public Holiday(int id_holiday, int id_employe,Date startDate, Date endDate ,
       HolidayType type){
14     this.id_holiday = id_holiday;
15     this.id_employe = id_employe;
16     this.startDate = startDate;
17     this.endDate = endDate;
18     this.type = type;
19 }
20
21 public int getId_holiday() {
22     return id_holiday;
23 }
24 public Date getStartDate() {
25     return startDate;
26 }
27 public Date getEndDate() {
28     return endDate;
29 }
30 public HolidayType getType() {
31     return type;
32 }
33
34 public int getId_employe() {
35     return id_employe;
36 }
37
38     public Object getSolde() {
39         throw new UnsupportedOperationException("Not supported yet.");
40     }
41
```

```
}
```

Listing 1.11: Holiday

1.4.4 HolidayModel.java

HolidayModel gère les opérations sur les congés via HolidayDAOImpl. Voici les fonctionnalités principales :

–addHoliday(...) : Valide les dates de congé, vérifie le solde de congé de l'employé, met à jour le solde, et ajoute un congé.

–displayHoliday() : Récupère et retourne une liste de toutes les demandes de congé.

–deleteHoliday(int id) : Supprime une demande de congé par ID.

–updateHoliday(...) : Valide et met à jour les informations d'une demande de congé existante, en ajustant le solde de l'employé en conséquence.

Cette classe assure la gestion efficace des congés dans le système de gestion des ressources humaines.

```
package Model;
import Controller.EmployeController;
import java.util.List;

import DAO.HolidayDAOImpl;
import java.sql.Date;
public class HolidayModel {
    private HolidayDAOImpl dao;

    public HolidayModel(HolidayDAOImpl dao) {
        this.dao = dao;
    }

    public boolean addHoliday(int id, int id_employe, Date startdate, Date enddate, HolidayType type , Employe targetEmploye) {

        if(startdate.after(enddate)) return false;
        if(startdate.equals(enddate)) return false;
        if(startdate.before(new Date(System.currentTimeMillis()))) return false;
        if(enddate.before(new Date(System.currentTimeMillis()))) return false;

        long daysBetween = (enddate.toLocalDate().toEpochDay() - startdate.toLocalDate().toEpochDay());
        if(daysBetween > targetEmploye.getSolde()) return false;
        EmployeController.updateSolde(targetEmploye.getId(), targetEmploye.getSolde() - (int) daysBetween);
        Holiday e = new Holiday(id, id_employe, startdate, enddate, type);
```

```
25          dao.add(e);
26          return true;
27      }
28
29
30      public List<Holiday> displayHoliday() {
31          List<Holiday> Holidays = dao.display();
32          return Holidays;
33      }
34
35      public boolean deleteHoliday(int id) {
36          dao.delete(id);
37          return true;
38      }
39
40      public boolean updateHoliday(int id, int id_employe, Date startdate, Date
    enddate, HolidayType type , Employe targetEmploye , int olddaysbetween ) {
41
42          long daysBetween = (enddate.toLocalDate().toEpochDay() - startdate.
    toLocalDate().toEpochDay());
43
44          if(startdate.after(enddate)) return false;
45          if(startdate.equals(enddate)) return false;
46          if(startdate.before(new Date(System.currentTimeMillis()))) return false
    ;
47          if(enddate.before(new Date(System.currentTimeMillis()))) return false;
48
49          if(daysBetween > (targetEmploye.getSolde()+olddaysbetween)) return
    false;
50          EmployeController.updateSolde(targetEmploye.getId(), (targetEmploye.
    getSolde()+olddaysbetween) - (int) daysBetween);
51
52          Holiday e = new Holiday(id, id_employe, startdate, enddate, type);
53          dao.update(e);
54          return true;
55      }
56
57  }
```

Listing 1.12: HolidayModel

1.4.5 HolidayType.java

HolidayType est une énumération définissant les différents types de congés

```
1  package Model;
2
3  public enum HolidayType {
4      CONGE_PAYE,    // Cong  pay
5      CONGE_NON_PAYE, // Cong  non pay
6      CONGE_MALADIE    // Cong  maladie
7  }
```

Listing 1.13: HolidayType

1.4.6 Poste.java

Poste est une énumération définissant différents postes au sein de l'entreprise.

```
package Model;

public enum Poste {
    INGENIEURE_ETUDE_ET_DEVELOPPEMENT,
    TEAM_LEADER,
    PILOTE
}
```

Listing 1.14: Poste

1.4.7 Role.java

Role est une énumération définissant les différents rôles des utilisateurs dans l'application.

```
package Model;

public enum Role{
    ADMIN,
    EMPLOYE
}
```

Listing 1.15: Role

1.5 View

1.5.1 MainView.java

MainView est la classe principale pour l'interface graphique de l'application de gestion des employés et des congés, créée avec Java Swing. Elle comporte deux onglets principaux : un pour la gestion des employés et un pour la gestion des congés. Chaque onglet contient un tableau (JTable) pour afficher les données et des formulaires (JPanel) pour saisir des informations. Des boutons d'action permettent d'ajouter, mettre à jour, supprimer et afficher les employés et les congés. La classe inclut également des méthodes pour afficher des messages d'erreur ou de succès, vider et remplir les champs des formulaires, et vérifier si les champs sont vides. Les interactions utilisateur sont gérées par des écouteurs d'événements associés aux composants de l'interface.

```
1  package Model;
2
3  package View;
4
5  import DAO.EmployeDAOImpl;
6  import DAO.EmployeDAOImpl;
7  import Model.Employe;
8  import Model.EmployeModel;
9  import Model.Poste;
10 import Model.Role;
11 import Model.EmployeModel;
12 import Model.HolidayType;
13 import java.awt.*;
14 import javax.swing.*;
15 import javax.swing.table.DefaultTableModel;
16 import java.util.List;
17
18 public class MainView extends JFrame {
19
20     // le tableau de employe et conge
21     private JTabbedPane tabbedPane = new JTabbedPane();
22
23     // les tabs
24     private JPanel employeTab = new JPanel();
25     private JPanel holidayTab = new JPanel();
26
27     // les panels
28     private JPanel Employepan = new JPanel();
29     private JPanel Holidaypan = new JPanel();
30     private JPanel Display_Table_employe = new JPanel();
31     private JPanel Display_Table_holiday = new JPanel();
32     private final JPanel Forme_employe = new JPanel();
33     private final JPanel Forme_holiday = new JPanel();
34     private JPanel panButton_employe = new JPanel();
35     private JPanel panButton_holiday = new JPanel();
36
37     // les labels du l'employe
38     private JLabel label_nom = new JLabel("Nom");
39     private JLabel label_prenom = new JLabel("Prenom");
40     private JLabel label_email = new JLabel("Email");
41     private JLabel label_tele = new JLabel("Telephone");
42     private JLabel label_salaire = new JLabel("Salaire");
43     private JLabel label_role = new JLabel("Role");
44     private JLabel label_poste = new JLabel("Poste");
45
46     // les labels du conge
47     private JLabel label_employe = new JLabel("Nom de l'employe");
48     private JLabel label_startDate = new JLabel("Date de debut (YYYY-MM-DD)");
49     private JLabel label_endDate = new JLabel("Date de fin (YYYY-MM-DD)");
50     private JLabel label_type = new JLabel("Type");
51     private JComboBox<HolidayType> TypeComboBox = new JComboBox<>(HolidayType.
       values());
52
53     // les textfield du l'employe
54     private JTextField text_nom = new JTextField();
55     private JTextField text_prenom = new JTextField();
56     private JTextField text_email = new JTextField();
57     private JTextField text_tele = new JTextField();
58     private JTextField text_salaire = new JTextField();
59
```

```
60      private JComboBox<Role> roleComboBox = new JComboBox<>(Role.values());
61      private JComboBox<Poste> posteComboBox = new JComboBox<>(Poste.values());
62
63      // les textfield du conge
64      private JComboBox<String> text_employe = new JComboBox<>();
65      private JTextField text_startDate = new JTextField("");
66      private JTextField text_endDate = new JTextField("");
67
68      // les boutons du l'employe
69      private JButton addButton_employe = new JButton("Ajouter");
70      private JButton updateButton_employe = new JButton("Modifier");
71      private JButton deleteButton_employe = new JButton("Supprimer");
72      private JButton displayButton_employe = new JButton("Afficher");
73
74      // les boutons du conge
75      private JButton addButton_holiday = new JButton("Ajouter");
76      private JButton updateButton_holiday = new JButton("Modifier");
77      private JButton deleteButton_holiday = new JButton("Supprimer");
78      private JButton displayButton_holiday = new JButton("Afficher");
79
80
81      // le tableau de l'employe
82      JPanel pan0 = new JPanel(new BorderLayout());
83      public static String[] columnNames_employe = {"ID", "Nom", "Prenom", "Email
        ", "Telephone", "Salaire", "Role", "Poste","Solde"};
84      public static DefaultTableModel tableModel = new DefaultTableModel(
        columnNames_employe, 0);
85      public static JTable Tableau = new JTable(tableModel);
86
87      // le tableau du conge
88      JPanel pan1 = new JPanel(new BorderLayout());
89      public static String[] columnNames_holiday = {"ID", "nom_employe","
        date_debut","date_fin","type"};
90      public static DefaultTableModel tableModel1 = new DefaultTableModel(
        columnNames_holiday, 0);
91      public static JTable Tableau1 = new JTable(tableModel1);
92
93      public MainView() {
94
95          setTitle("Gestion des employes et des conges");
96          setSize(1000, 600);
97          setDefaultCloseOperation(EXIT_ON_CLOSE);
98          setLocationRelativeTo(null);
99
100         add(tabbedPane);
101
102     // Employe Tab
103         employeTab.setLayout(new BorderLayout());
104         employeTab.add(Employepan, BorderLayout.CENTER);
105
106         Employepan.setLayout(new BorderLayout());
107         Employepan.add(Display_Table_employe, BorderLayout.CENTER);
108         Tableau.setFillsViewportHeight(true);
109         Dimension preferredSize = new Dimension(900, 500);
110         Tableau.setPreferredScrollableViewportSize(preferredSize);
111         pan0.add(new JScrollPane(Tableau), BorderLayout.CENTER);
112         Display_Table_employe.add(pan0);
113
114         Employepan.add(panButton_employe, BorderLayout.SOUTH);
115         panButton_employe.add(addButton_employe);
116         panButton_employe.add(updateButton_employe);
117         panButton_employe.add(deleteButton_employe);
118         panButton_employe.add(displayButton_employe);
```

```
119
120         Employepan.add(Forme_employe, BorderLayout.NORTH);
121         Forme_employe.setLayout(new GridLayout(7, 2, 10, 10));
122         Forme_employe.add(label_nom);
123         Forme_employe.add(text_nom);
124         Forme_employe.add(label_prenom);
125         Forme_employe.add(text_prenom);
126         Forme_employe.add(label_email);
127         Forme_employe.add(text_email);
128         Forme_employe.add(label_tele);
129         Forme_employe.add(text_tele);
130         Forme_employe.add(label_salaire);
131         Forme_employe.add(text_salaire);
132         Forme_employe.add(label_role);
133         Forme_employe.add(roleComboBox);
134         Forme_employe.add(label_poste);
135         Forme_employe.add(posteComboBox);
136
137     // Holiday Tab
138         holidayTab.setLayout(new BorderLayout());
139         holidayTab.add(Holidaypan, BorderLayout.CENTER);
140         Holidaypan.setLayout(new BorderLayout());
141         Holidaypan.add(Display_Table_holiday, BorderLayout.CENTER);
142
143         Tableau1.setFillsViewportHeight(true);
144         Tableau1.setPreferredScrollableViewportSize(preferredSize);
145         pan1.add(new JScrollPane(Tableau1), BorderLayout.CENTER);
146         Display_Table_holiday.add(pan1);
147
148         Holidaypan.add(Forme_holiday, BorderLayout.NORTH);
149         Forme_holiday.setLayout(new GridLayout(4, 2, 10, 10));
150         Forme_holiday.add(label_employe);
151         Forme_holiday.add(text_employe);
152         Forme_holiday.add(label_startDate);
153         Forme_holiday.add(text_startDate);
154         Forme_holiday.add(label_endDate);
155         Forme_holiday.add(text_endDate);
156         Forme_holiday.add(label_type);
157         Forme_holiday.add(TypeComboBox);
158
159         Holidaypan.add(panButton_holiday, BorderLayout.SOUTH);
160         panButton_holiday.add(addButton_holiday);
161         panButton_holiday.add(updateButton_holiday);
162         panButton_holiday.add(deleteButton_holiday);
163         panButton_holiday.add(displayButton_holiday);
164
165     // TabbedPane
166         tabbedPane.addTab("Employe", employeTab);
167         tabbedPane.addTab("Holiday", holidayTab);
168
169         remplaire_les_employes();
170         setVisible(true);
171     }
172
173     public void remplaire_les_employes () {
174         List<Employe> Employes = new EmployeModel(new EmployeDAOImpl()).
    displayEmploye();
175         text_employe.removeAllItems();
176         for (Employe elem : Employes) {
177             text_employe.addItem(elem.getId() + " - " + elem.getNom()+" "+elem.
    getPrenom());
178         }
179     }
```

```
180
181
182     // getters
183
184
185         public int getId_employe() {
186             return Integer.parseInt(text_employe.getSelectedItem().toString().
    split(" - ")[0]);
187         }
188         public String getNom() {
189             return text_nom.getText();
190         }
191
192         public JTable getTable() {
193             return (JTable) Display_Table_employe.getComponent(0);
194         }
195
196         public String getPrenom() {
197             return text_prenom.getText();
198         }
199
200         public String getEmail() {
201             return text_email.getText();
202         }
203
204         public String getTelephone() {
205             return text_tele.getText();
206         }
207
208         public double getSalaire() {
209             return Double.parseDouble(text_salaire.getText());
210         }
211
212         public Role getRole() {
213             return (Role) roleComboBox.getSelectedItem();
214         }
215
216         public Poste getPoste() {
217             return (Poste) posteComboBox.getSelectedItem();
218         }
219
220         public JButton getaddButton_employe () {
221             return addButton_employe;
222         }
223
224         public JButton getupdateButton_employe () {
225             return updateButton_employe;
226         }
227
228         public JButton getdeleteButton_employe () {
229             return deleteButton_employe;
230         }
231
232         public JButton getdisplayButton_employe () {
233             return displayButton_employe;
234         }
235
236         public JButton getaddButton_holiday () {
237             return addButton_holiday;
238         }
239
240         public JButton getupdateButton_holiday () {
241             return updateButton_holiday;
```

```
242         }
243         public JButton getdeleteButton_holiday () {
244             return deleteButton_holiday;
245         }
246 
247         public JButton getdisplayButton_holiday () {
248             return displayButton_holiday;
249         }
250         public String getStartDate () {
251             return text_startDate.getText();
252         }
253 
254         public String getEndDate() {
255             return text_endDate.getText();
256         }
257 
258         public HolidayType getHolidayType(){
259             return (HolidayType) TypeComboBox.getSelectedItem();
260         }
261 
262     // methods d'affichage des messages
263         public void afficherMessageErreur(String message) {
264             JOptionPane.showMessageDialog(this, message, "Erreur", JOptionPane.
    ERROR_MESSAGE);
265         }
266 
267         public void afficherMessageSucces(String message) {
268             JOptionPane.showMessageDialog(this, message, "Succes", JOptionPane.
    INFORMATION_MESSAGE);
269         }
270 
271     // methodes de vider les champs
272         public void viderChamps_em() {
273             text_nom.setText("");
274             text_prenom.setText("");
275             text_email.setText("");
276             text_tele.setText("");
277             text_salaire.setText("");
278             roleComboBox.setSelectedIndex(0);
279             posteComboBox.setSelectedIndex(0);
280         }
281 
282         public void viderChamps_ho() {
283             text_startDate.setText("");
284             text_endDate.setText("");
285             TypeComboBox.setSelectedIndex(0);
286         }
287 
288     // methodes de remplir les champs
289         public void remplaireChamps_em (int id, String nom, String prenom,
    String email, String telephone, double salaire, Role role, Poste poste) {
290             text_nom.setText(nom);
291             text_prenom.setText(prenom);
292             text_email.setText(email);
293             text_tele.setText(telephone);
294             text_salaire.setText(String.valueOf(salaire));
295             roleComboBox.setSelectedItem(role);
296             posteComboBox.setSelectedItem(poste);
297         }
298 
299         public void remplaireChamps_ho(int id_employe, String date_debut,
    String date_fin, HolidayType type) {
300             List<Employe> Employes = new EmployeModel(new EmployeDAOImpl()).
```

```
    displayEmploye();
301             text_employe.removeAllItems();
302             for (Employe elem : Employes) {
303                 if (elem.getId() == id_employe) {
304                     text_employe.addItem(elem.getId() + " - " + elem.getNom()+"
     "+elem.getPrenom());
305                     text_employe.setSelectedItem(elem.getId() + " - " + elem.
    getNom()+" "+elem.getPrenom());
306                 }
307             }
308             text_startDate.setText(date_debut);
309             text_endDate.setText(date_fin);
310             TypeComboBox.setSelectedItem(type);
311         }
312 
313     // methodes de test des champs
314         public boolean testChampsVide_em (){
315             return text_nom.getText().equals("") || text_prenom.getText().
    equals("") || text_email.getText().equals("") || text_tele.getText().equals(
    "") || text_salaire.getText().equals("");
316         }
317 
318         public boolean testChampsVide_ho () {
319             return text_employe.getSelectedItem().equals("") || text_startDate.
    getText().equals("") || text_endDate.getText().equals("") || TypeComboBox.
    getSelectedItem().equals("");
320         }
321 
322 }
```

Listing 1.16: MainView

Chapter 2

Section 2: Présentation des interfaces

Dans cette section, nous allons explorer les interfaces de notre application de gestion des ressources humaines ,nous fournirons une brève explication de chaque interface. Cela permettra d'avoir un aperçu visuel des fonctionnalités et de l'interface utilisateur.

2.1 Gestion des employes

2.1.1 Ajouter un employe

Dans la section d'ajout, j'ai entré les données d'un nouvel employé. Après avoir soumis le formulaire, un message de succès s'est affiché pour confirmer l'ajout réussi de l'employé. Ensuite, en affichant la liste des employés, l'employé nouvellement ajouté apparaît avec les autres employés existants.

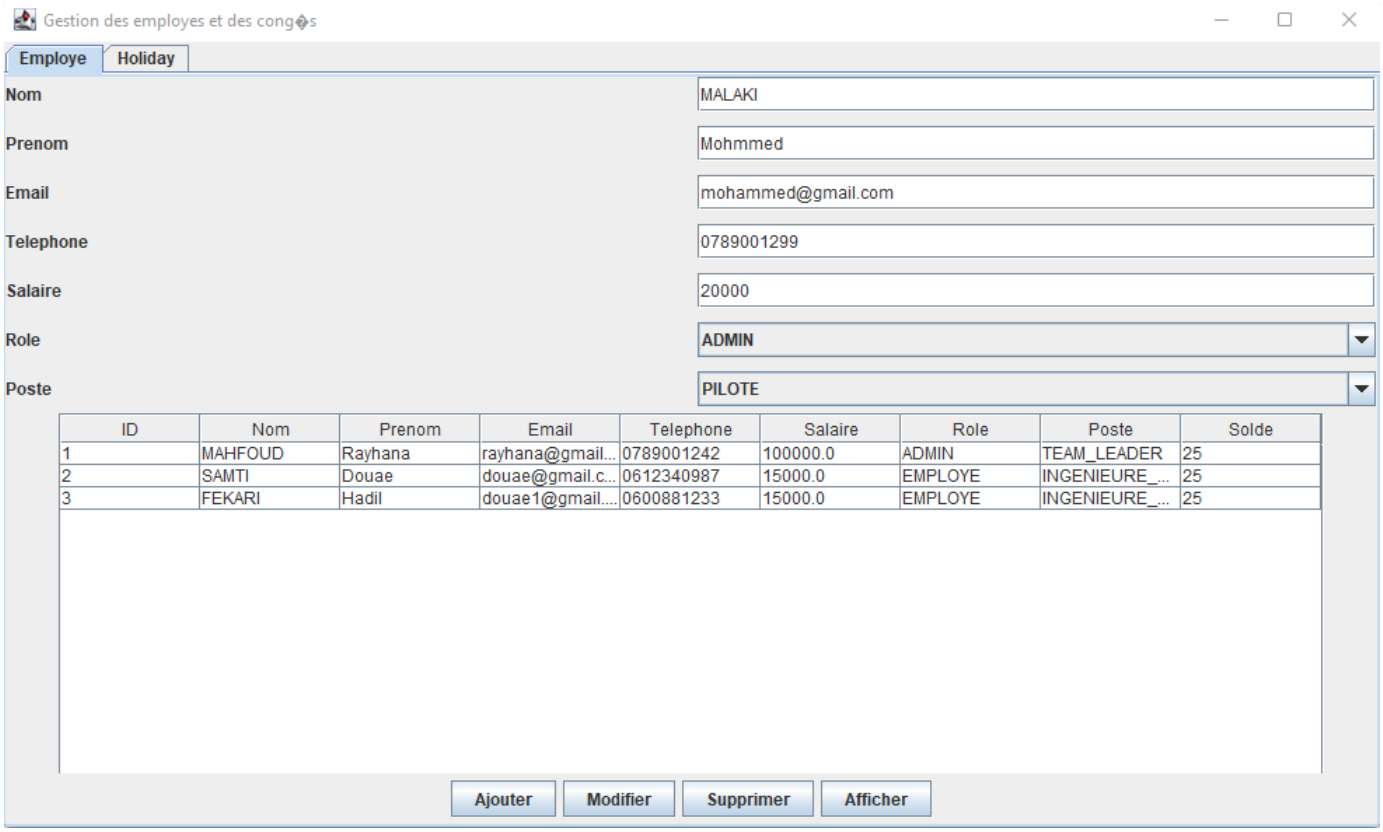

Figure 2.1: entrer les donnees du l'employe a ajouter

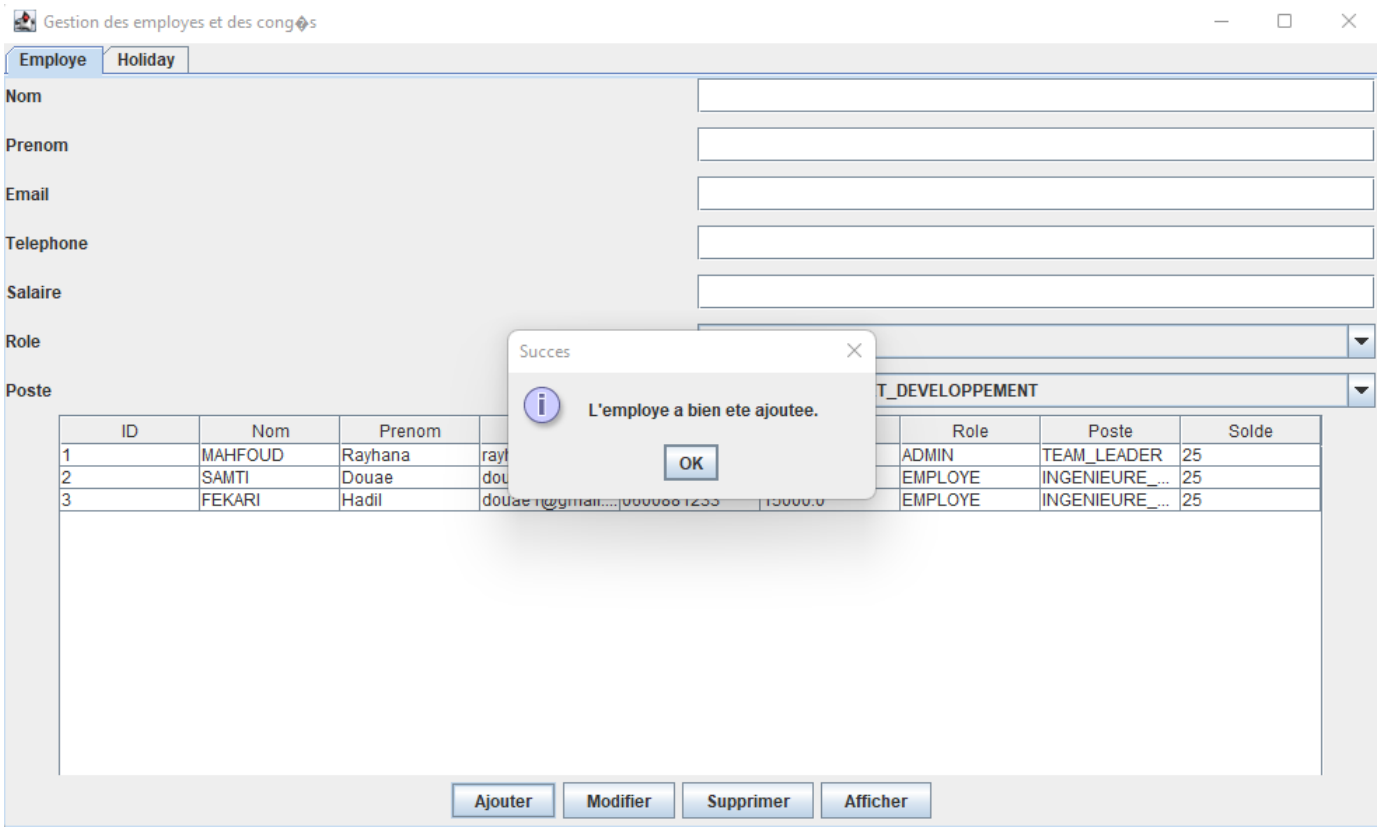

Figure 2.2: succes d'ajout

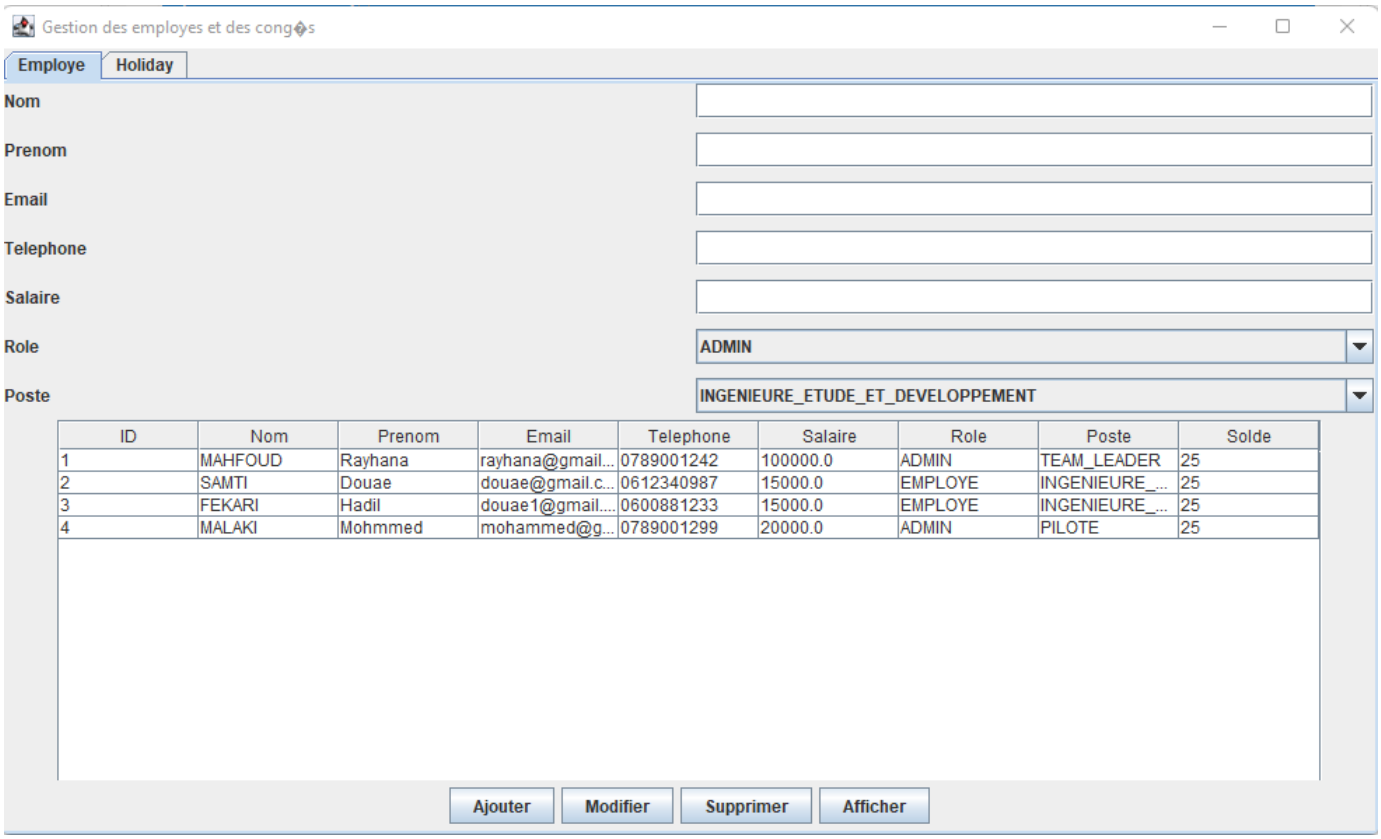

Figure 2.3: affichage apres l'ajout

2.1.2 Modification d'un employe

Dans la section de modification, je commence par sélectionner l'employé que je souhaite modifier. Après avoir apporté les changements nécessaires aux informations de l'employé, je soumets les modifications. Un message de succès s'affiche pour confirmer que la modification a été effectuée avec succès. Ensuite, la liste des employés est mise à jour pour refléter les changements apportés à l'employé sélectionné.

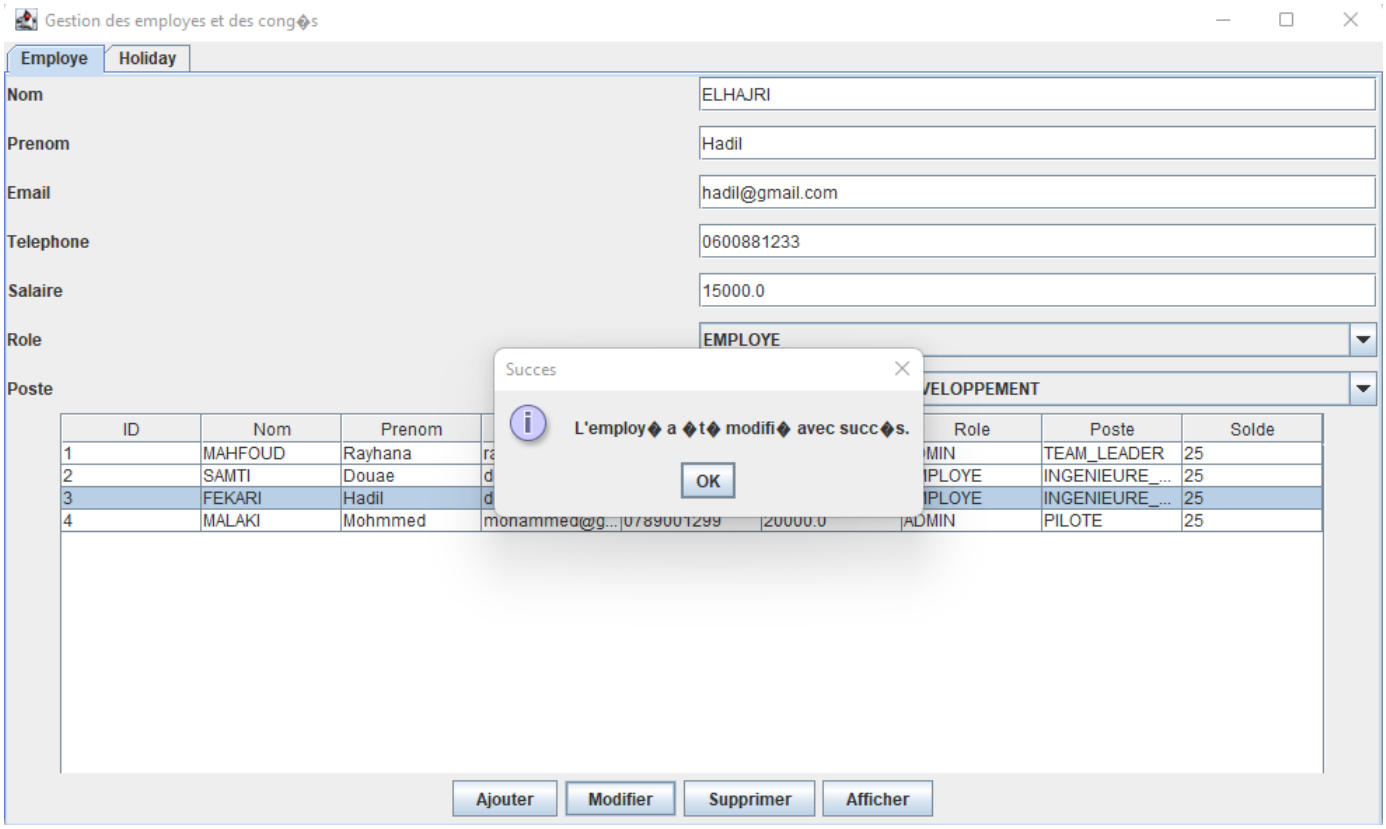

Figure 2.4: Modification d'employe

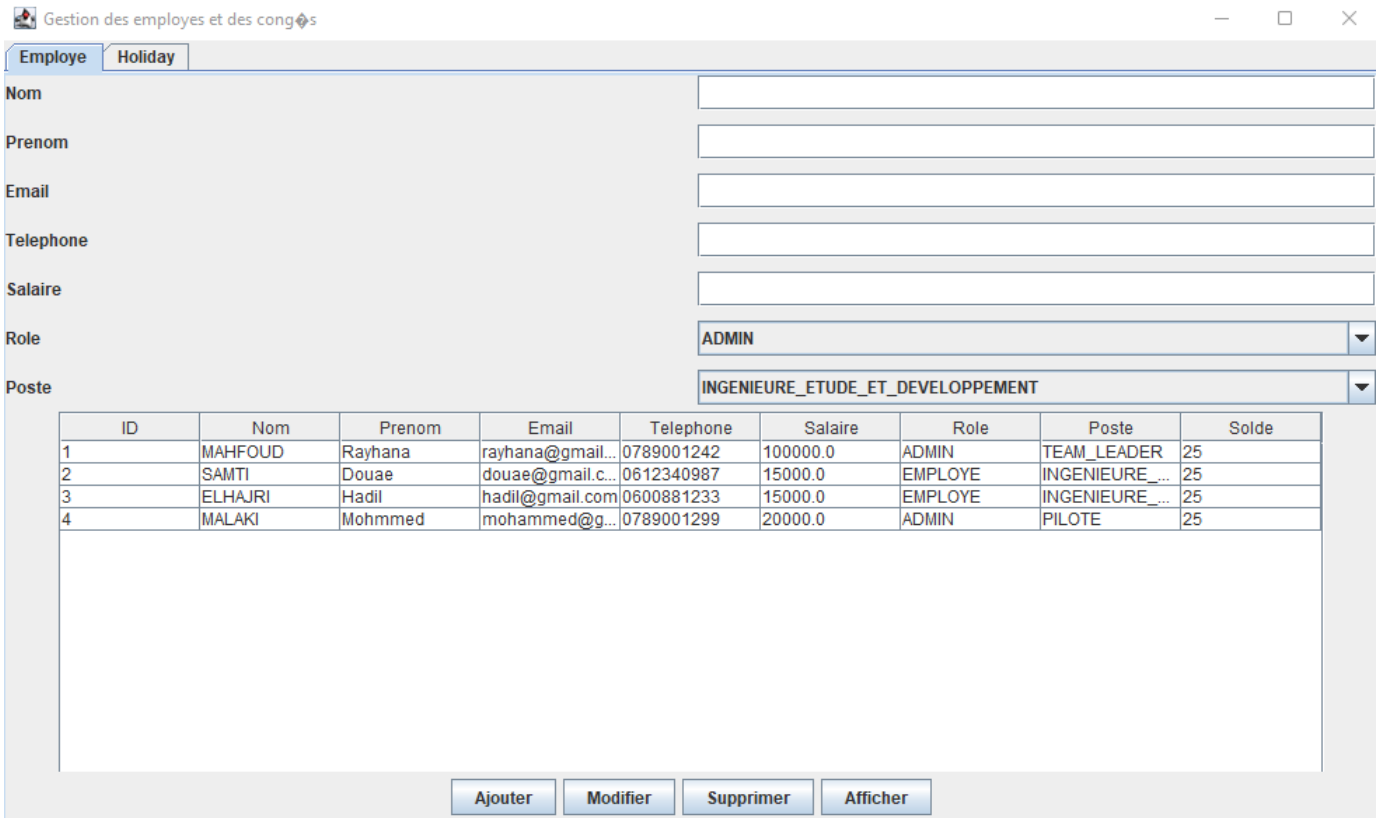

Figure 2.5: affichage apres la modification

2.1.3 Suppression d'un emolye

Dans la section de suppression, je commence par sélectionner l'employé que je souhaite supprimer. Après un message de succès s'affiche pour indiquer que l'opération a été effectuée avec succès. Ensuite l'employé supprimé ne figure plus dans la liste.

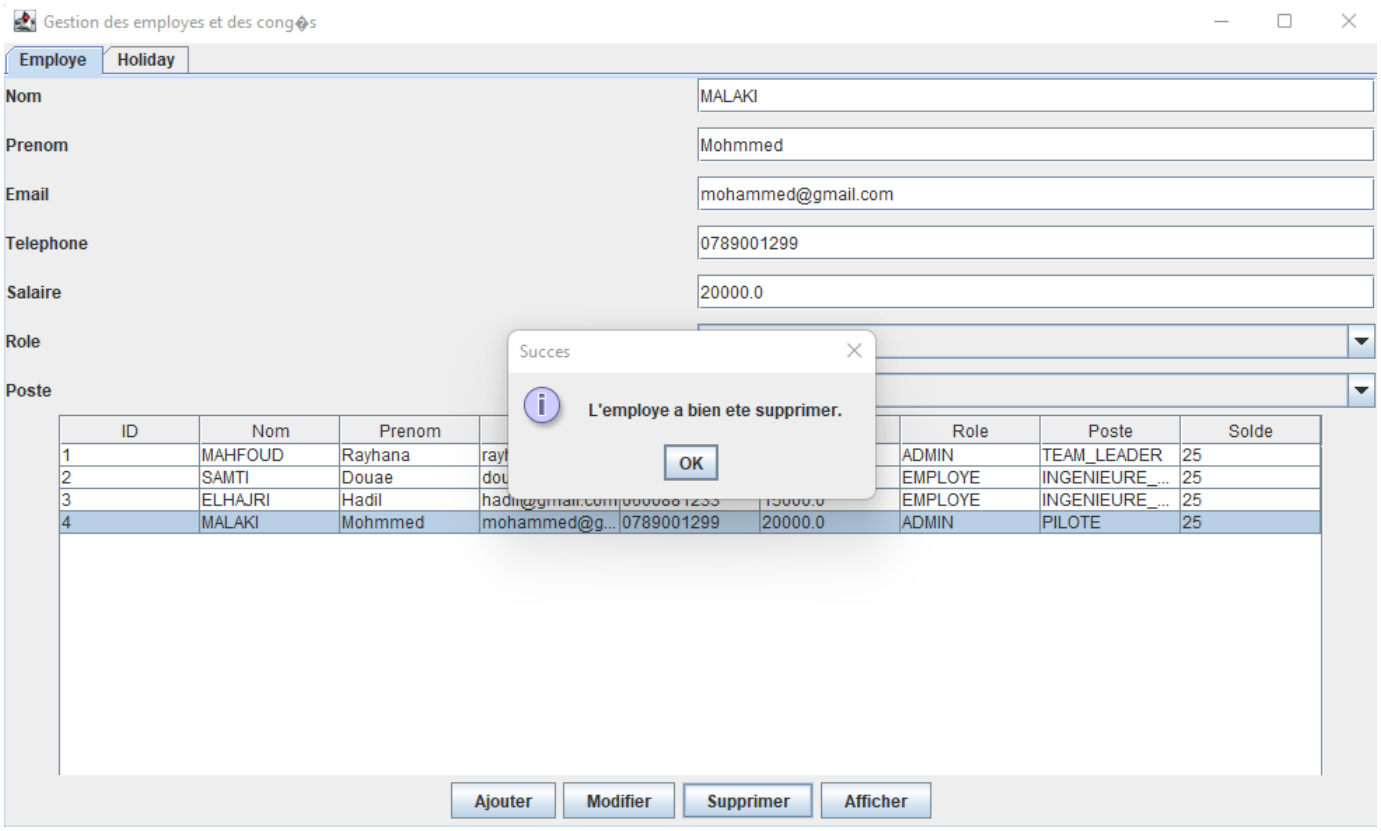

Figure 2.6: supprimer un employe

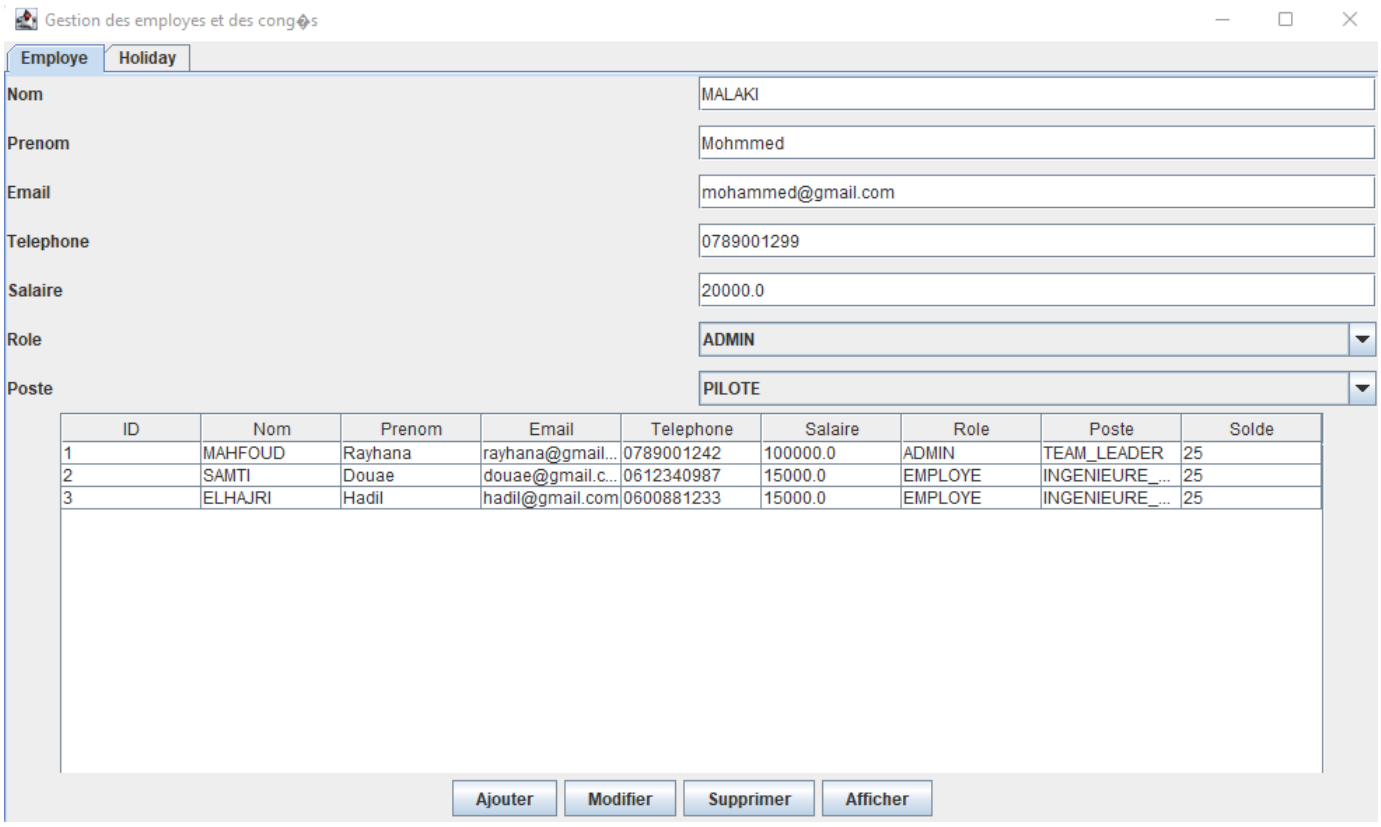

Figure 2.7: affichage apres la suppression

2.2 Gestion des conges

2.2.1 Ajout d'un conge

LDans la section d'ajout de congé, nous entrons les données nécessaires (nom de l'employé, dates, type de congé). Un message de succès confirme l'ajout. Le nouveau congé apparaît ensuite dans la liste des demandes de congé.

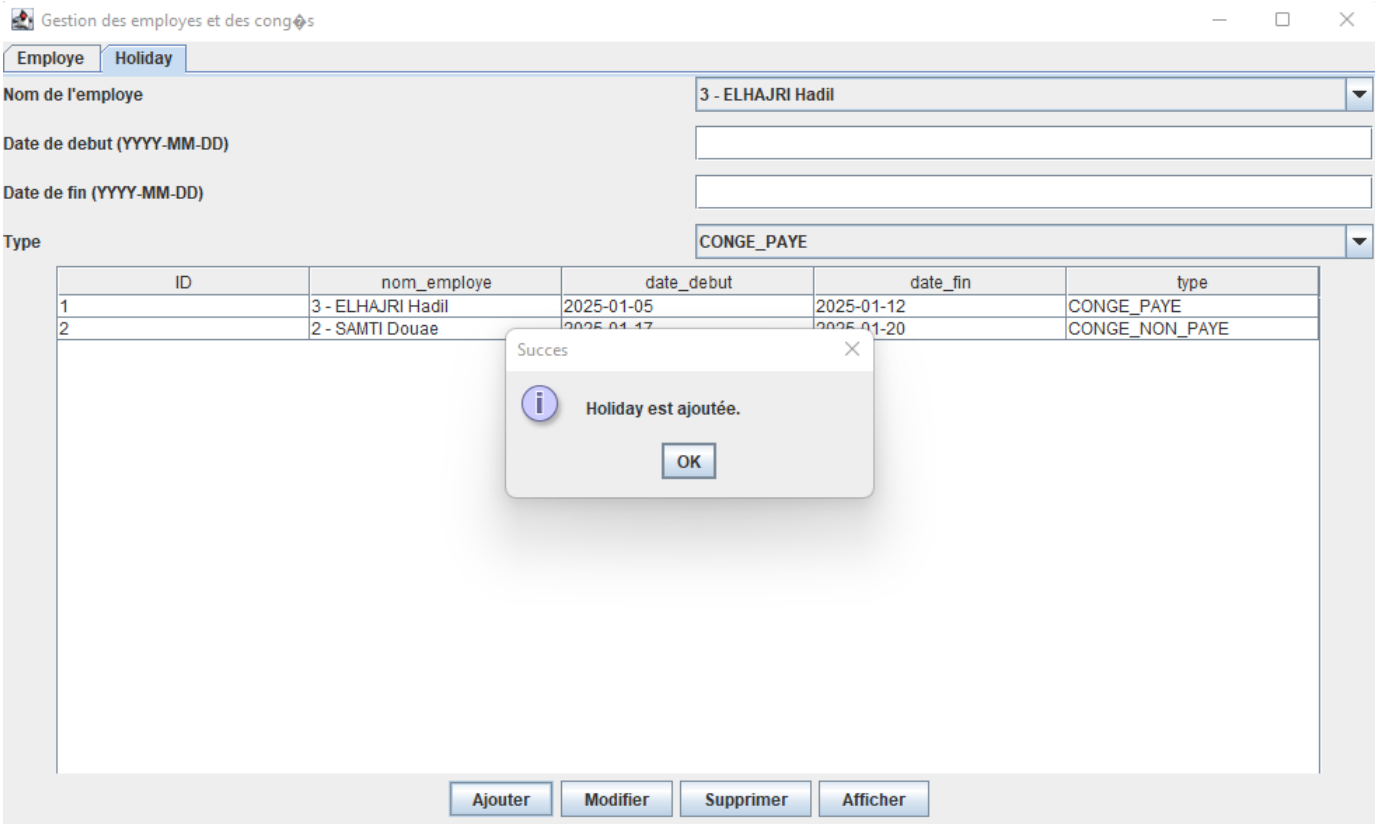

Figure 2.8: Ajouter un conge

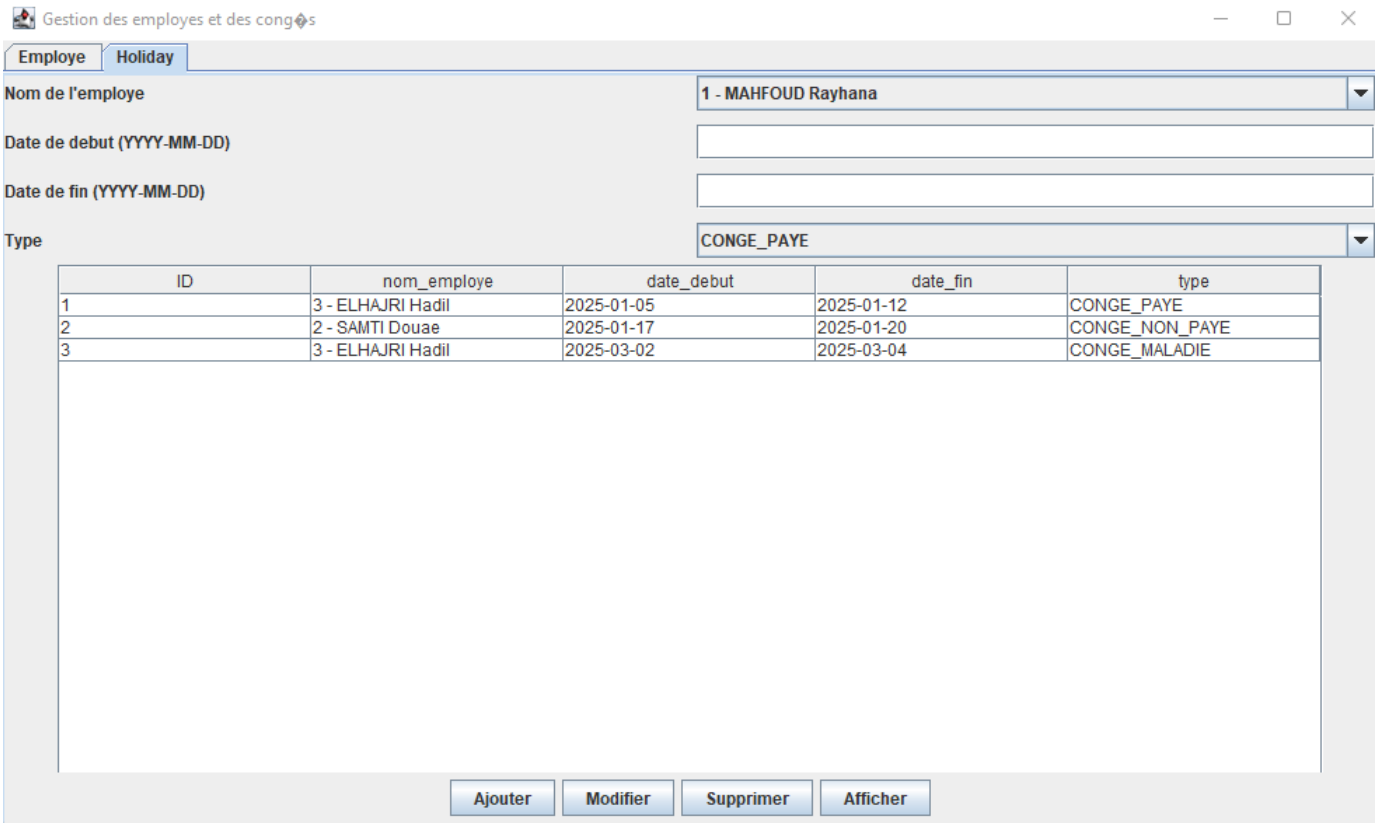

Figure 2.9: Affichage apres l'ajout

2.2.2 Modification d'un conge

Dans la section de modification de congé, nous sélectionnons le congé à modifier, apportons les changements nécessaires aux informations (comme les dates ou le type de congé), et soumettons les modifications. Un message de succès confirme que les changements ont été appliqués. La liste des congés est alors mise à jour pour refléter les modifications effectuées.

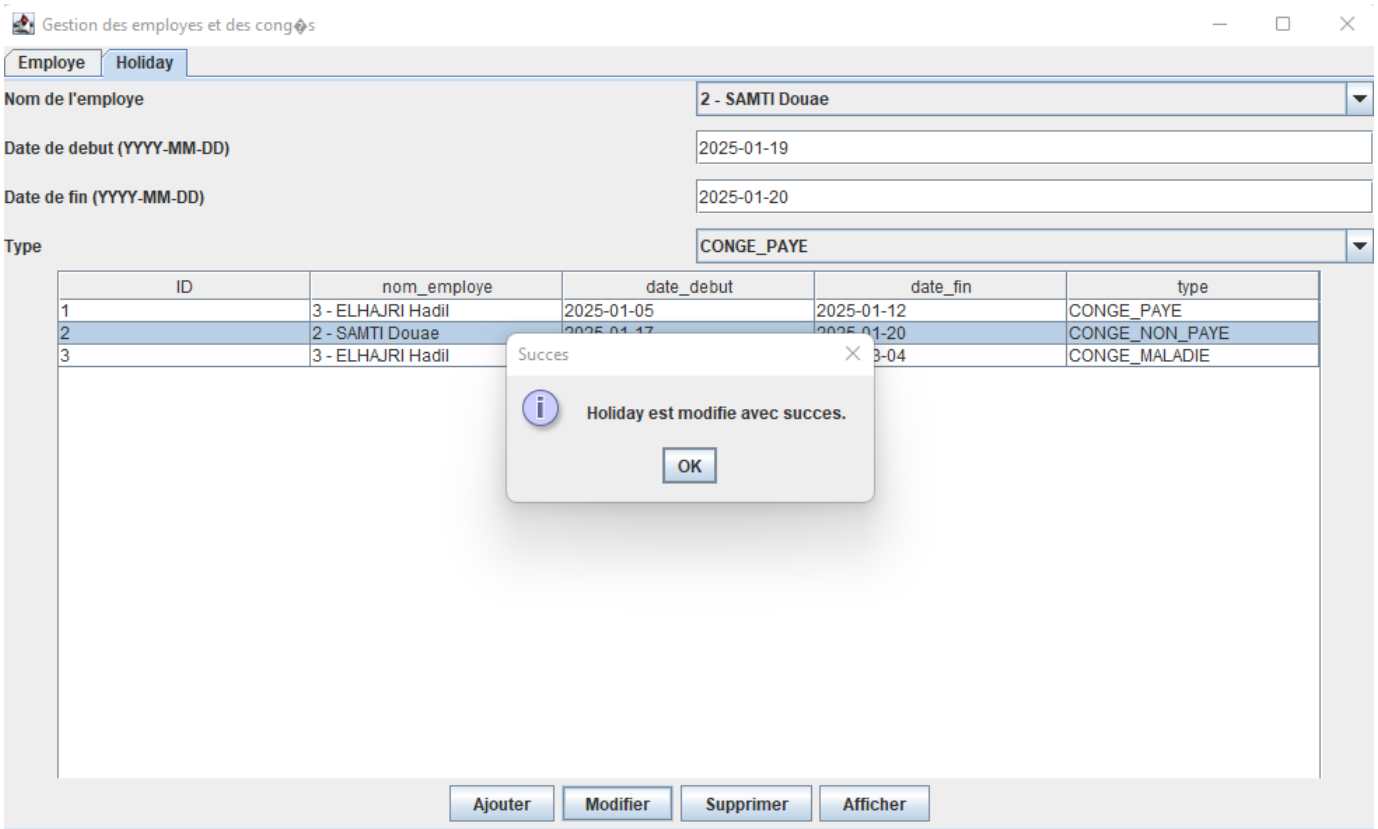

Figure 2.10: Modification d'un conge

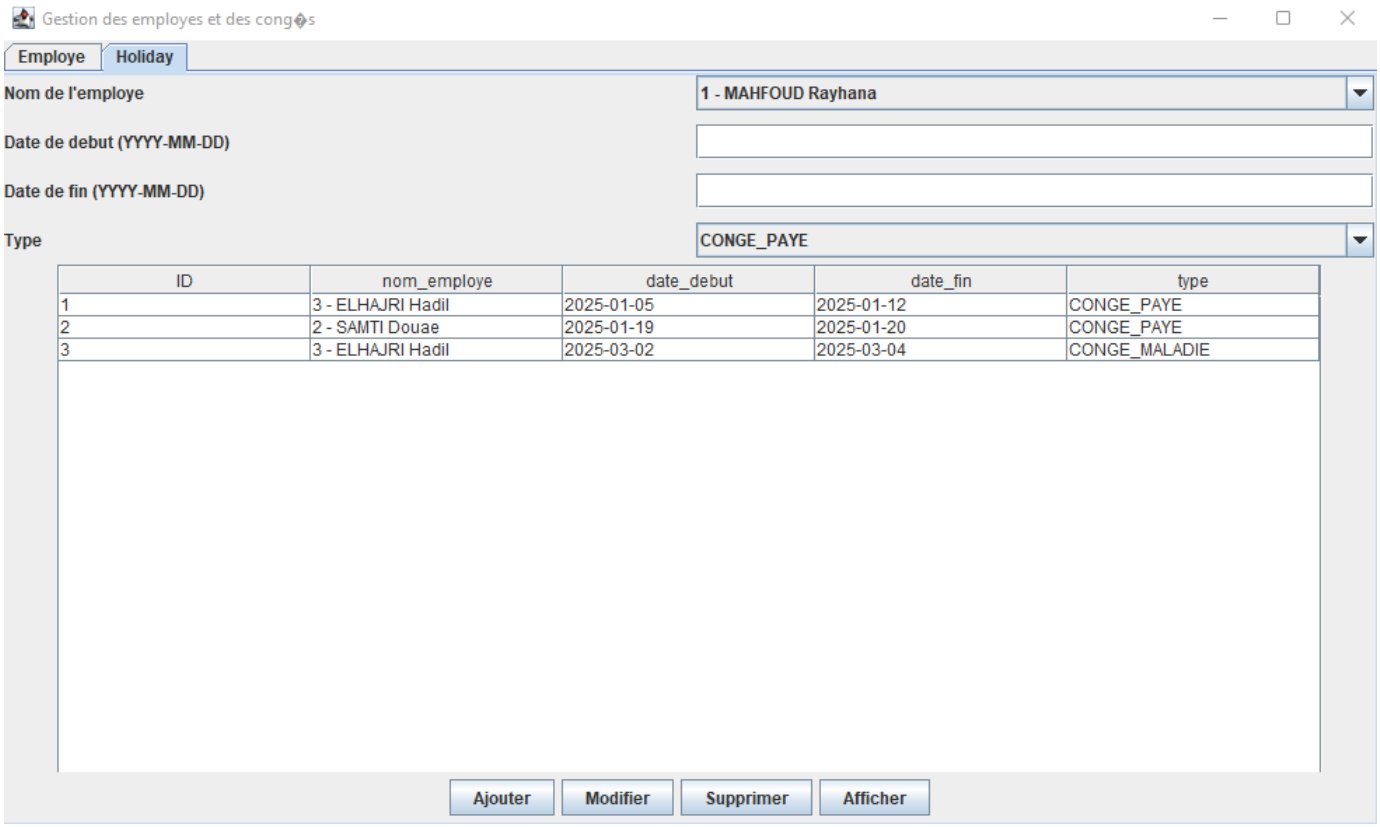

Figure 2.11: Afficher apres la modification

2.2.3 Suppression d'un conge

Dans la section de suppression de congé, nous sélectionnons le congé à supprimer. Un message de succès s'affiche pour indiquer que l'opération a été effectuée. Le congé supprimé ne figure plus dans la liste des demandes de congé.

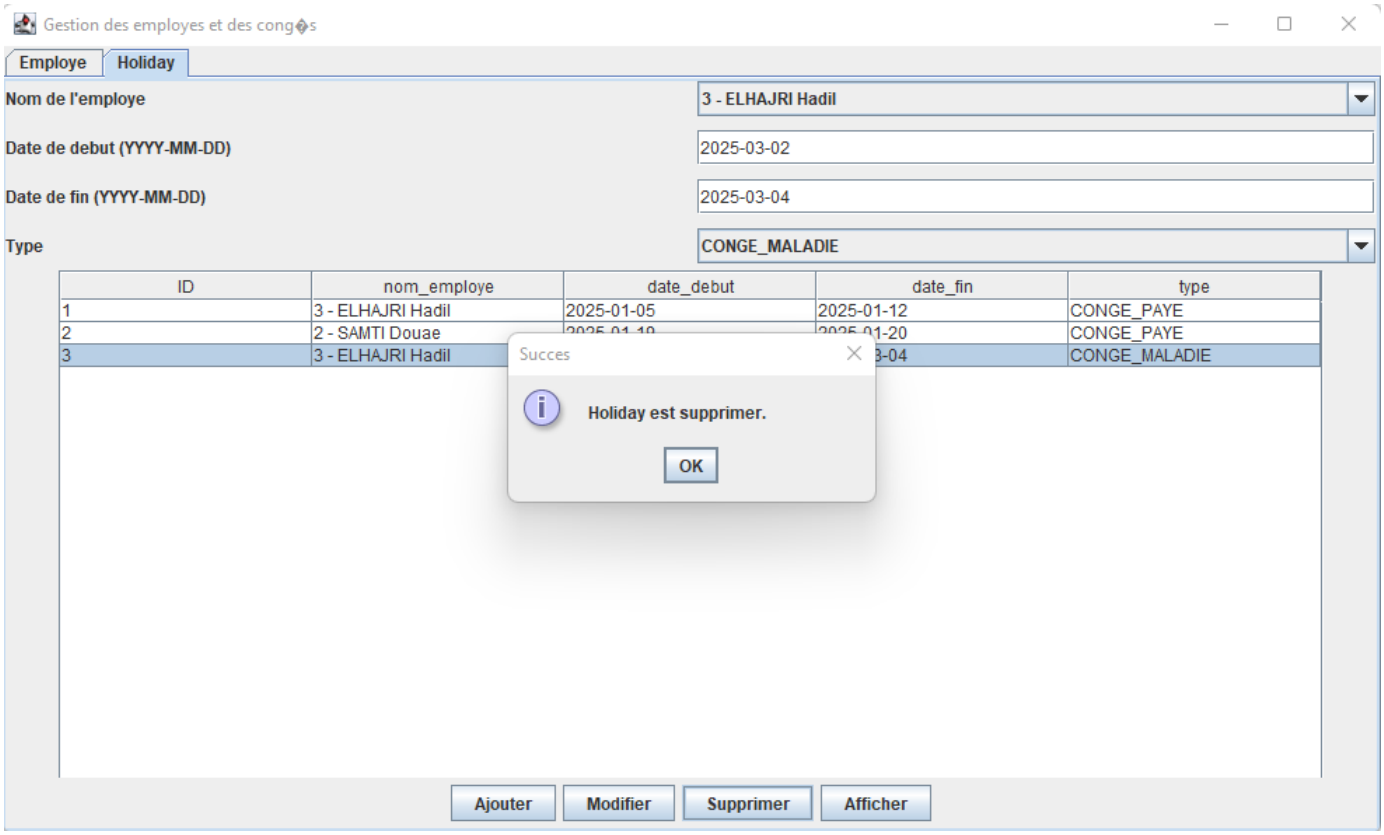

Figure 2.12: Supprimer conge

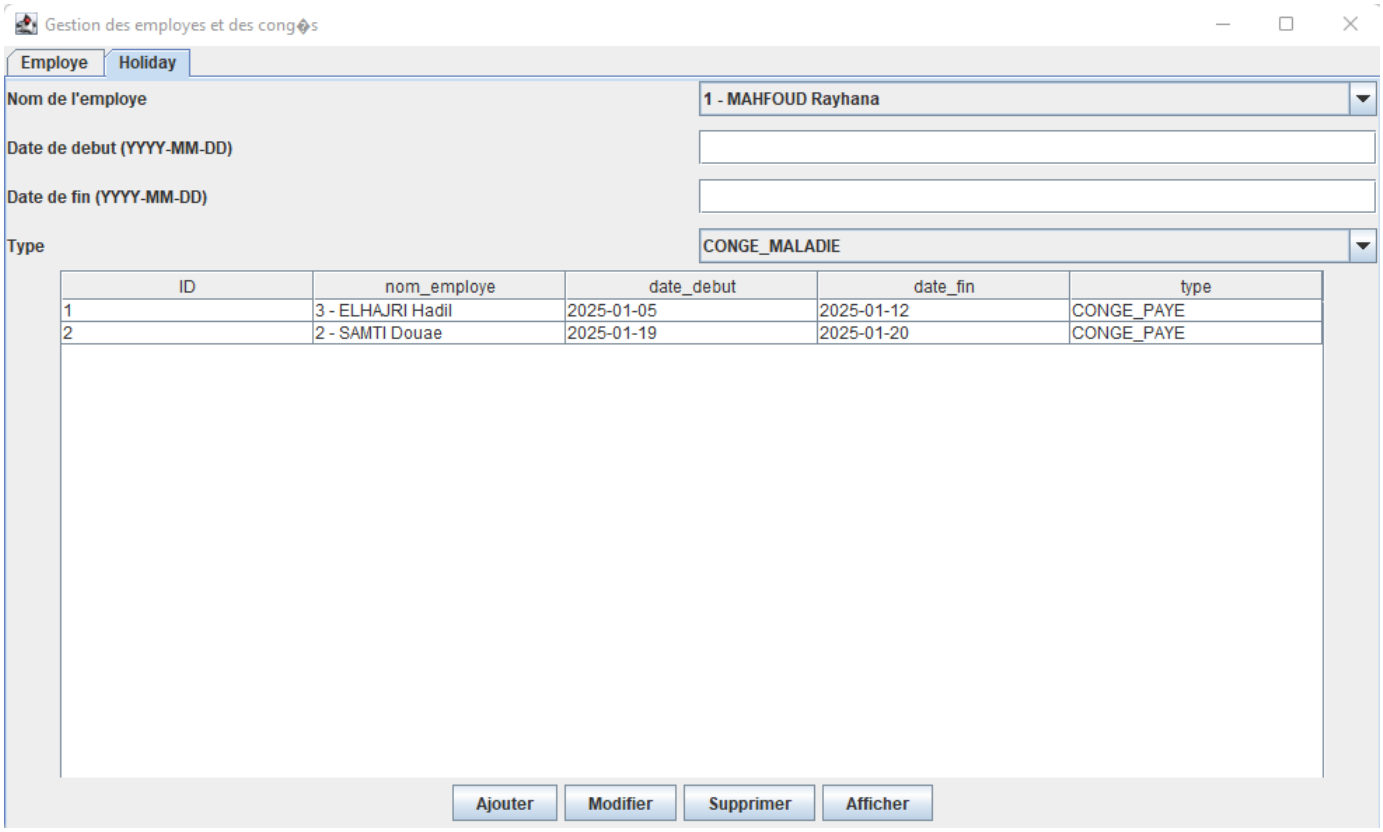

Figure 2.13: Affichage apres la suppression

2.2.4 Essayer d'ajouter un conge qui se cheveuche avec un autre

Lorsque vous essayez d'ajouter un congé dont les dates se chevauchent avec un autre congé existant, un message d'erreur s'affiche pour indiquer que les dates sélectionnées sont invalides en raison du chevauchement. Cette vérification permet d'éviter la superposition de congés pour un même employé, assurant ainsi une gestion cohérente et précise des congés.

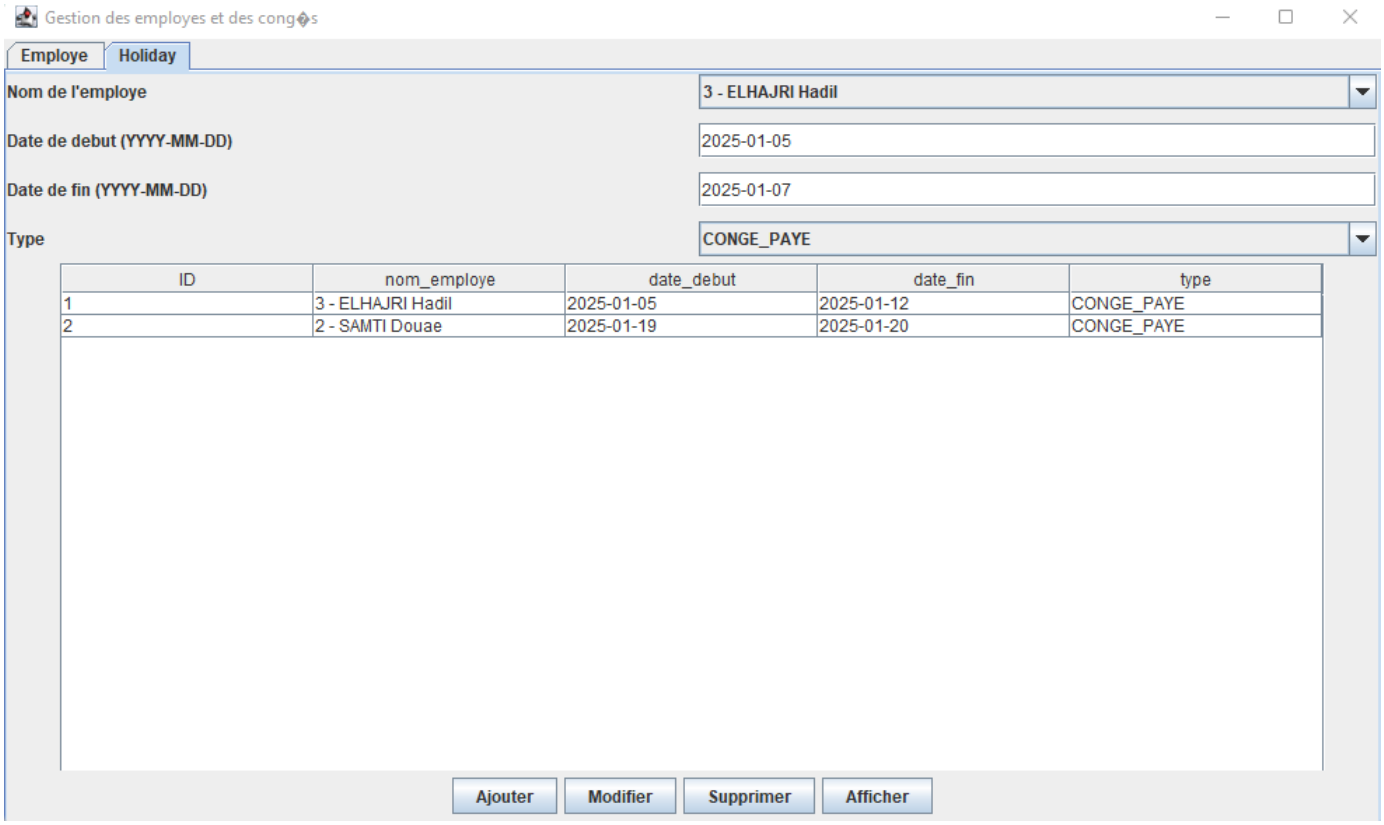

Figure 2.14: entrer une date qui se chevauche avec un autre

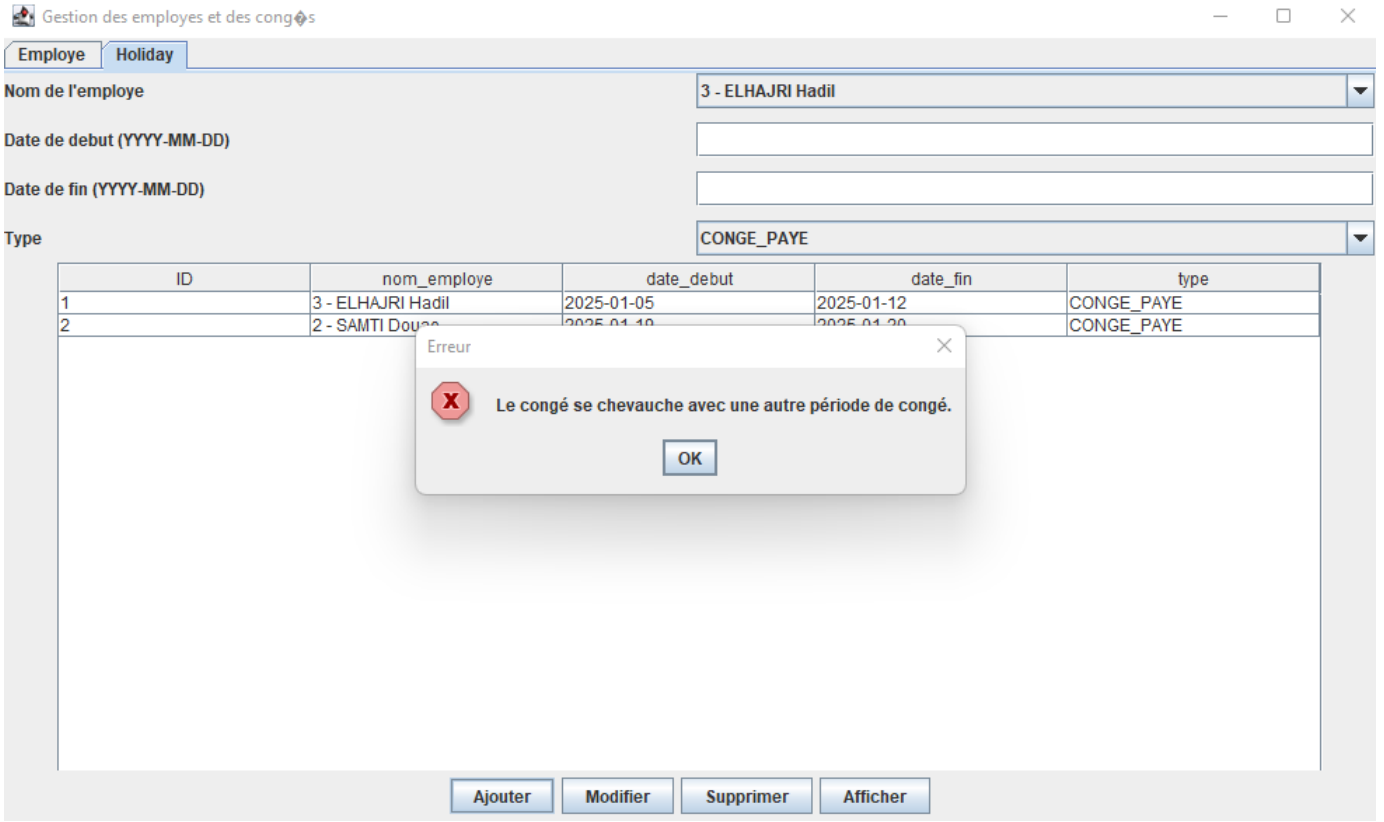

Figure 2.15: Impossible d'ajouter

2.2.5 Decrementation du solde dans la liste des employes

Le solde de congés, initialement de 25 jours, se décrémente automatiquement lorsqu'un nouveau congé est ajouté.

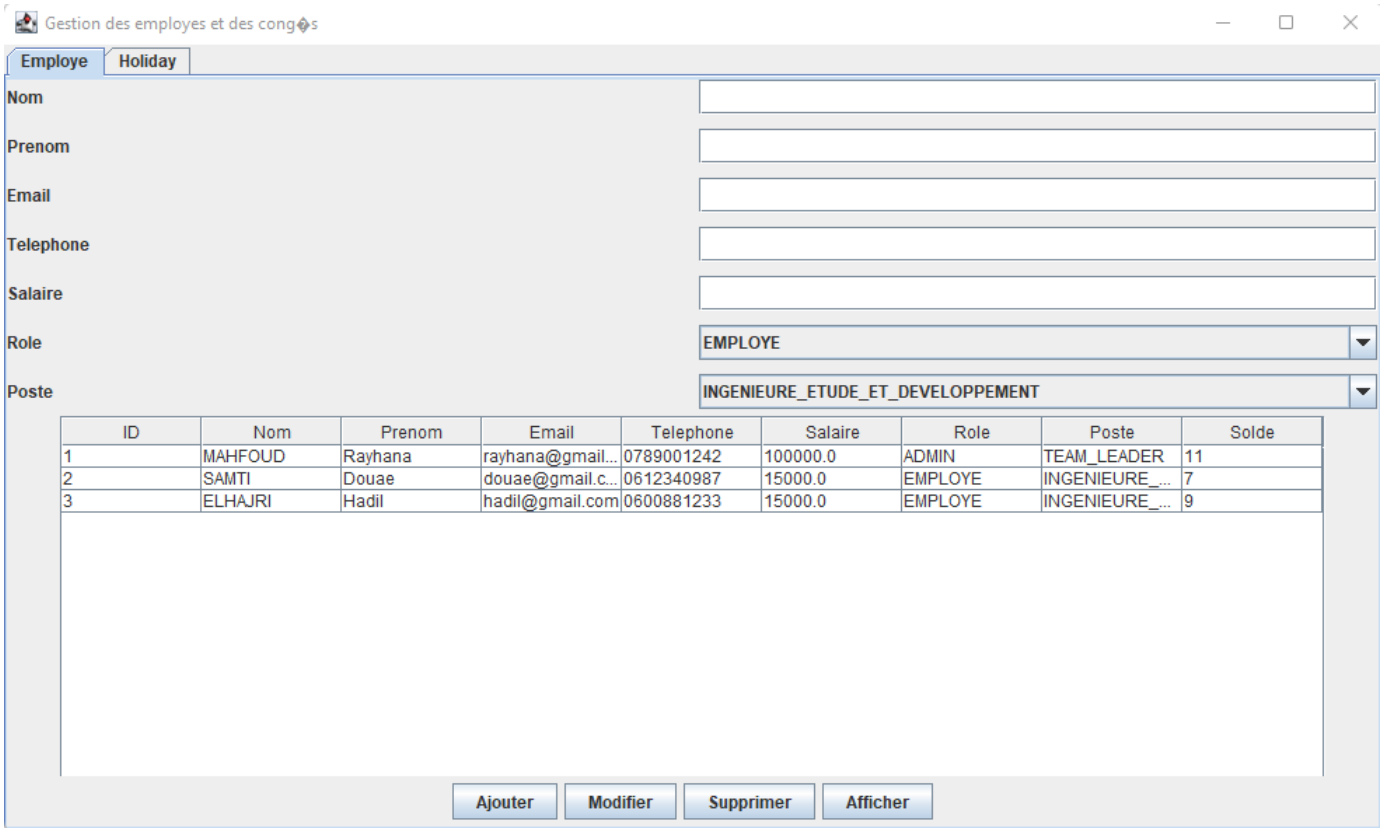

Figure 2.16: decrementation de conges

Conclusion Générale

Ce projet de gestion des ressources humaines offre une solution complète et fonctionnelle pour la gestion des employés et des congés. En intégrant des fonctionnalités essentielles comme l'ajout, la modification et la suppression des employés et des congés, ainsi que des mécanismes de validation pour éviter les chevauchements de dates de congé, l'application garantit une administration fluide et efficace des ressources humaines. Grâce à l'utilisation de l'architecture MVC, nous avons assuré une séparation claire des responsabilités, rendant l'application plus modulaire et maintenable.

L'implémentation du modèle DAO a permis de gérer efficacement les interactions avec la base de données, assurant une persistance des données fiable et sécurisée.

Avec une interface utilisateur bien conçue et des processus automatisés pour le calcul des soldes de congés, cette application constitue un outil précieux pour toute organisation cherchant à optimiser la gestion de ses ressources humaines.