Université Cadi Ayyad – Marrakech
Ecole Supérieure de Technologie – Safi
Département : Informatique
Filière : Génie Informatique

Compte Rendu TP2 Java

Gestion des employés et congés

Réalisé par : **TISSAFI IDRISSI Zineb**

Encadré par : **Mme EL KHOURCHI Asmaa**

Année universitaire : 2024-2025

1 Introduction

Dans le cadre de ce TP, nous avons exploré les concepts fondamentaux de la gestion des employés et des congés. Ce projet nous a permis de concevoir et d'implémenter une solution fonctionnelle simulant un système de gestion efficace pour suivre les employés, gérer leurs données, et organiser leurs congés. Ce compte rendu détaille les différentes étapes parcourues pour réaliser ce travail. Chaque étape est accompagnée de descriptions et d'explications. Pour une consultation complète du code source du projet et des étapes détaillées, vous pouvez accéder au dépôt GitHub via le lien suivant :

2 Structure du Projet

Le projet suit une architecture en couches bien organisée pour garantir la modularité et la maintenabilité. Les principaux packages sont :

- **Controller :** Contient les classes comme **EmployeController** et **HolidayController**, qui gèrent la logique métier et la communication entre la vue et les données.
- **DAO (Data Access Object) :** Gère l'accès aux données, avec des classes comme **DBConnexion** pour la connexion à la base de données et **EmployeDAOImpl/ HolidayDAOImpl** pour les opérations spécifiques.
- **Model :** Définit les entités métier telles que Employe, Holiday, et des classes associées comme **Post, Role,** et **TypeHoliday**.
- **View :** Contient les classes pour l'interface utilisateur, comme **EmployeeHolidayView**, pour afficher et manipuler les données.
- **Main :** La classe principale **(Main.java)** initialise et orchestre l'application.
- **Referenced Libraries :** Inclut le connecteur MySQL (mysql-connector-j-9.1.0.jar) pour gérer les interactions avec la base de données.

Cette structure garantit une séparation claire des responsabilités et facilite la maintenance et l'évolution du projet.

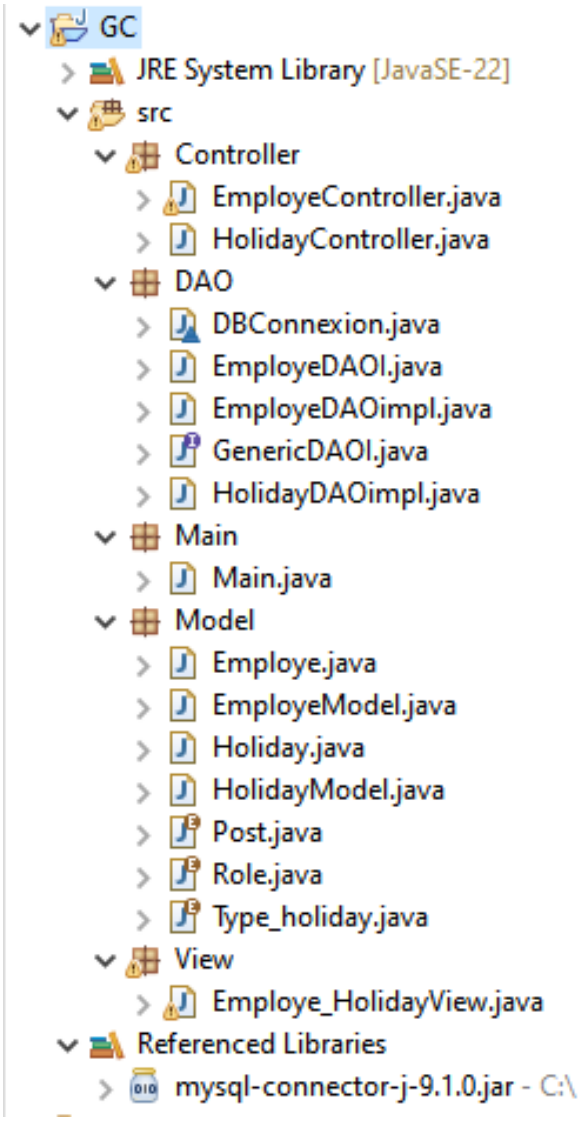

Figure 1: Structure du projet en couches

3 Réalisation

3.1 Script base de donnée :

Ce script a été conçu sur MySQL Workbench pour la création d'une base de données optimisée, nommée **GES**. Il comprend notamment deux tables principales : **Employee** et **Holiday,** soigneusement structurées pour gérer les informations des employés et leurs congés.

```sql
CREATE DATABASE Ges;
USE Ges;
CREATE TABLE `employe` (
  `id` int(11) NOT NULL,
  `nom` varchar(100) NOT NULL,
  `prenom` varchar(100) NOT NULL,
  `email` varchar(150) NOT NULL,
  `telephone` varchar(15) DEFAULT NULL,
  `salaire` decimal(10,2) NOT NULL,
  `role` enum('ADMIN','MANAGER','EMPLOYEE') NOT NULL,
  `poste` enum('DEVELOPER','DESIGNER','MARKETING','OTHER') NOT NULL,
  `solde` int(11) NOT NULL
) ;
-- Création de la table "holiday"
CREATE TABLE `holiday` (
  `id` int(11) NOT NULL,
  `id_employe` int(11) NOT NULL,
  `startdate` date NOT NULL,
  `enddate` date NOT NULL,
```

Figure 2: Script SQL de la base de données

Aperçu des tables de la base de données GES Ce screen illustre les deux tables principales de la base de données GES :

1.Employee : dédiée à la gestion des informations relatives aux employés (identifiants, noms, postes, etc.).

2.Holiday : conçue pour suivre et organiser les congés attribués aux employés (dates, Type, etc.). Ces tables sont soigneusement reliées pour garantir une gestion optimale et simplifier l'interaction avec les données via JDBC.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
employe	Parcourir Structure Rechercher Insérer Vider Supprimer	7	InnoDB	utf8mb4_general_ci	32,0 kio	-
holiday	Parcourir Structure Rechercher Insérer Vider Supprimer	8	InnoDB	utf8mb4_general_ci	32,0 kio	-
login	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	48,0 kio	-
3 tables	Somme	15	InnoDB	utf8mb4_general_ci	112,0 kio	0 o

Tout cocher — Avec la sélection :

Figure 3: Structure des tables Employee et Holiday de la base de données GES.

4 Architecture MVC (Model-View-Controller)

L'architecture MVC sépare une application en trois composants :

- **Model (Modèle)** : Gère les données et la logique métier.
- **View (Vue)** : Affiche les données et l'interface utilisateur.
- **Controller (Contrôleur)** : Relie le modèle et la vue, gérant les requêtes et la logique.

4.1 Couche Model : Implémentation du Model(Employee)

J'ai conçu une classe **Employe** avec des attributs pour stocker les informations essentielles d'un employé, comme le nom, le prénom, et l'email. J'ai inclus des **constructeur**s, **des getters** et **des setters** pour faciliter la manipulation des données, ainsi que des énumérations pour représenter les rôles et postes.

```
1  package Model;
2
3  public class Employe{
4      private int id;
5      private String nom;
6      private String prenom;
7      private String email;
8      private String telephone;
9      private double salaire;
10     private Role role;
11     private Post poste ;
12     private int solde ;
13
14     public Employe(int id ,String nom, String prenom, String email,
           String telephone, double salaire, Role role, Post post ,int
           solde){
15         this.id = id;
16         this.nom = nom;
17         this.prenom = prenom;
18         this.email = email;
19         this.telephone = telephone;
20         this.salaire = salaire;
21         this.role = role;
22         this.poste = post;
23         this.solde = solde;
24     }
25     public int getId(){
26         return id;
27     }
28
29     public void setId(int id){
30         this.id = id;
```

```
31      }
32      public String getNom() {
33          return nom;
34      }
35
36      public void setNom(String nom) {
37          this.nom = nom;
38      }
39
40      public String getPrenom() {
41          return prenom;
42      }
43
44      public void setPrenom(String prenom) {
45          this.prenom = prenom;
46      }
47
48      public String getEmail() {
49          return email;
50      }
51
52      public void setEmail(String email) {
53          this.email = email;
54      }
55
56      public String getTelephone() {
57          return telephone;
58      }
59
60      public void setTelephone(String telephone) {
61          this.telephone = telephone;
62      }
63
64      public double getSalaire() {
65          return salaire;
66      }
67
68      public void setSalaire(double salaire) {
69          this.salaire = salaire;
```

```
70      }
71
72      public Role getRole() {
73          return role;
74      }
75
76      public void setRole(Role role) {
77          this.role = role;
78      }
79
80      public Post getPost() {
81          return poste;
82      }
83
84      public void setPost(Post post) {
85          this.poste = post;
86      }
87
88      public void setSolde (int conge){
89          this.solde = conge;
90      }
91
92      public int getSolde(){
93          return solde;
94      }
95
96  }
```

enum Poste :

```
1  package Model;
2  public enum Post {
3          DEVELOPER, DESIGNER, MARKETING, OTHER
4  }
```

enum Role :

```
1  package Model;
2  public enum Role {
3      ADMIN, EMPLOYEE ,MANAGER
4  }
```

4.2 (Couche DAO) : Création de la connexion

J'ai créé une classe connexion qui gère la connexion à la base de données MySQL. La méthode getConnexion() établit une connexion si elle n'existe pas déjà, et closeConnexion() ferme la connexion lorsque c'est nécessaire. Cela permet de gérer la connexion de manière centralisée.

```
1  package DAO;
2  import java.sql.*;
3  class DBConnexion {
4  // D claration des informations n cessaires pour  tablir  une
       connexion   la base de donn es
5      public static final String url =
           "jdbc:mysql://localhost:3306/Ges?useSSL=false&serverTimezone=UTC";
6      public static final String user = "root";
7      public static final String password = "";
8      public static Connection conn = null;
9
10     public static Connection getConnexion() throws
           ClassNotFoundException {
11 // Si une connexion existe d j  , retourner cette connexion
12         if (conn != null) {
13             return conn;
14         }
15         try {
16
17             Class.forName("com.mysql.cj.jdbc.Driver");
18             conn = DriverManager.getConnection(url, user, password);
19             System.out.println("connexion r ussie");
20         } catch (SQLException e) {
21 // Si une erreur se produit lors de la connexion, une exception
       RuntimeException est lanc e
22             throw new RuntimeException("Error de connexion", e);
23         }
24         return conn;
25     }
26 }
```

4.3 (Couche DAO) : Creation de l'interface EmployeeDAOI

J'ai conçu l'interface **EmployeDAOI** pour définir les opérations de gestion des employés. Elle comprend des méthodes permettant de trouver un employé par son ID, de l'ajouter, de le modifier, de le supprimer, ainsi que d'obtenir la liste des rôles et des postes disponibles.

```
1 package DAO;
2 import Model.Employe;
3 import java.util.List;
4 public interface EmployeDAOI {
5   public boolean add(Employe e);
6   public void deleteEmploye(int id);
7   public void updateEmploye(Employe e);
8   public List<Employe> displayEmploye();
9 }
```

4.4 (Couche DAO) :Creation de class qui implémente l'interface (EmployeDAOImpl) :

J'ai implémenté la **class EmployeDAOImpl** pour gérer les opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) sur les employés dans la base de données. Cette classe inclut les méthodes suivantes :

1. 1. **add() :** Permet d'ajouter un employé à la base de données.
2. 2. **findById() :** Recherche un employé par son identifiant.
3. 3. **findAll() :** Récupère tous les employés de la base de données.
4. 4. **update() :** Met à jour les informations d'un employé existant.
5. 5. **delete() :** Supprime un employé en fonction de son ID.

   De plus, j'ai ajouté les méthodes **findAllRoles()** et **findAllPostes()** pour retourner les rôles et postes définis dans l'énumération Role et Poste.

```
1
2 package DAO;
3 import Model.Employe;
```

```
4  import Model.Post;
5  import Model.Role;
6  import java.sql.PreparedStatement;
7  import java.sql.ResultSet;
8  import java.sql.SQLException;
9  import java.util.ArrayList;
10 import java.util.List;
11
12
13 public class EmployeDAOimpl implements GenericDAOI<Employe> {
14 //Ajout des employ s
15     @Override
16     public void add(Employe e) {
17         String sql = "INSERT INTO employe (nom, prenom, email,
               telephone, salaire, role, poste , solde) VALUES (?, ?, ?,
               ?, ?, ?, ? , ?)";
18         try (PreparedStatement stmt =
               DBConnexion.getConnexion().prepareStatement(sql)) {
19             stmt.setString(1, e.getNom());
20             stmt.setString(2, e.getPrenom());
21             stmt.setString(3, e.getEmail());
22             stmt.setString(4, e.getTelephone());
23             stmt.setDouble(5, e.getSalaire());
24             stmt.setString(6, e.getRole().name());
25             stmt.setString(7, e.getPost().name());
26             stmt.setInt(8, e.getSolde());
27             stmt.executeUpdate();
28         } catch (SQLException exception) {
29             System.err.println("failed of add employe ");
30         } catch (ClassNotFoundException ex) {
31             System.err.println("failed connexion with data base");
32         }
33     }
34 //Suppression des employ s par ID.
35     @Override
36     public void delete(int id) {
37         String sql = "DELETE FROM employe WHERE id = ?";
38         try (PreparedStatement stmt =
               DBConnexion.getConnexion().prepareStatement(sql)) {
```

```
39              stmt.setInt(1,id);
40              stmt.executeUpdate();
41          } catch (SQLException exception) {
42              System.err.println("failed of delete employe");
43          } catch (ClassNotFoundException ex) {
44              System.err.println("failed connexion with data base");
45          }
46      }
47  // Mise    jour des informations des employ s
48      @Override
49      public void update(Employe e) {
50          String sql = "UPDATE employe SET nom = ?, prenom = ?, email =
                ?, telephone = ?, salaire = ?, role = ?, poste = ? WHERE id
                = ?";
51          try (PreparedStatement stmt =
                DBConnexion.getConnexion().prepareStatement(sql)) {
52              stmt.setString(1, e.getNom());
53              stmt.setString(2, e.getPrenom());
54              stmt.setString(3, e.getEmail());
55              stmt.setString(4, e.getTelephone());
56              stmt.setDouble(5, e.getSalaire());
57              stmt.setString(6, e.getRole().name());
58              stmt.setString(7, e.getPost().name());
59              stmt.setInt(8,e.getId());
60              stmt.executeUpdate();
61          } catch (SQLException exception) {
62              System.err.println("failed of update employe");
63          } catch (ClassNotFoundException ex) {
64              System.err.println("failed connexion with data base");
65          }
66      }
67  //Affichage de la liste de tous les employ s
68      @Override
69      public List<Employe> display() {
70          String sql = "SELECT * FROM employe";
71          List<Employe> Employes = new ArrayList<>();
72          try (PreparedStatement stmt =
                DBConnexion.getConnexion().prepareStatement(sql)) {
73              ResultSet re = stmt.executeQuery();
```

```
74              while (re.next()) {
75                  int id = re.getInt("id");
76                  String nom = re.getString("nom");
77                  String prenom = re.getString("prenom");
78                  String email = re.getString("email");
79                  String telephone = re.getString("telephone");
80                  double salaire = re.getDouble("salaire");
81                  String role = re.getString("role");
82                  String poste = re.getString("poste");
83                  int solde = re.getInt("solde");
84                  Employe e = new Employe(id,nom, prenom, email,
                        telephone, salaire, Role.valueOf(role),
                        Post.valueOf(poste),solde);
85                  Employes.add(e);
86              }
87              return Employes;
88          } catch (ClassNotFoundException ex) {
89              System.err.println("failed connexion with data base");
90              return null;
91          } catch (SQLException ex) {
92              System.err.println("failed of display employe");
93              return null;
94          }
95      }
96
97  //Mise    jour du solde des employ s
98      public void updateSolde(int id, int solde) {
99          String sql = "UPDATE employe SET solde = ? WHERE id = ?";
100         try (PreparedStatement stmt =
                DBConnexion.getConnexion().prepareStatement(sql)) {
101             stmt.setInt(1, solde);
102             stmt.setInt(2, id);
103             stmt.executeUpdate();
104         } catch (SQLException exception) {
105             System.err.println("failed of update solde employe");
106         } catch (ClassNotFoundException ex) {
107             System.err.println("failed connexion with data base");
108         }
109
```

```
}
```

4.5 (Couche Model) : EmployeModel

Dans la classe **EmployeModel**, je gère l'ajout d'un employé en effectuant des validations sur les données saisies. Je vérifie que le salaire est supérieur à zéro et que l'email contient un "@". Si ces conditions sont remplies, je crée un nouvel objet Employe et l'ajoute à la base de données via l'implémentation DAO (EmployeImpl). Ainsi, je m'assure de l'intégrité des données avant de les enregistrer.

```
package Model;
import DAO.EmployeDAOimpl;
import java.util.List;

public class EmployeModel {
    private EmployeDAOimpl dao;
    public EmployeModel(EmployeDAOimpl dao) {
        this.dao = dao;
    }
    // funtion of add Employe :
    public boolean addEmploye(int id ,String nom, String prenom,
       String email, String telephone, double salaire, Role role, Post
       post, int solde) {
        if(salaire < 0 ){
            System.out.println("Erreur : le salaire doit etre
               positif.");
            return false;
        }
        if(id < 0 ){
            System.out.println("Erreur : l'id doit etre positif.");
            return false;
        }
        if(telephone.length() != 10){
            System.out.println("Erreur : le telephone doit etre 10
               num.");
            return false;
        }
        if(!email.contains("@")){
```

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

4.6 (Couche Controller) : l'implémentation du Employeecontroller :

j'ai créé un contrôleur qui gère les actions pour les employés . Je m'occupe d'ajouter, modifier, supprimer et afficher les employés, tout en interagissant avec le modèle pour mettre à jour la base de données et la vue pour afficher les changements.

```
1  package Controller;
2  import Model.*;
3  import View.*;
4  import java.sql.Date;
5  import java.util.Calendar;
6  import java.util.List;
7  import javax.swing.table.DefaultTableModel;
8
9  public class EmployeController {
10     private final Employe_HolidayView View;
11     public static EmployeModel model_employe;
12     public static int id = 0;
13     public static int oldselectedrow = -1;
14     public static boolean test = false;
15     String nom = "";
16     String prenom = "";
17     String email = "";
18     String telephone = "";
19     double salaire = 0;
20     Role role = null;
21     Post poste = null;
22     int solde = 0;
23     boolean updatereussi = false;
24
25 // Constructeur pour initialiser la vue et le mod le , et ajouter les
       couteurs d' vnements
26     public EmployeController(Employe_HolidayView view, EmployeModel
           model) {
27         this.View = view;
28         this.model_employe = model;
29
```

```
30 //   couteurs   pour les boutons d'ajout, suppression, mise    jour et
   affichage
31         View.getaddButton_employe().addActionListener(e ->
               addEmploye());
32         View.getdeleteButton_employe().addActionListener(e ->
               deleteEmploye());
33         View.getupdateButton_employe().addActionListener(e ->
               updateEmploye());
34         View.getdisplayButton_employe().addActionListener(e ->
               displayEmploye());
35         Employe_HolidayView.Tableau.getSelectionModel().addList
36         SelectionListener(e -> updateEmployebyselect());
37     }
38
39 // Afficher tous les employ s dans le tableau
40     public void displayEmploye() {
41         List<Employe> Employes = model_employe.displayEmploye();
42         if (Employes.isEmpty()) {
43             View.afficherMessageErreur("Aucun employ  trouv .");
44         }
45         DefaultTableModel tableModel = (DefaultTableModel)
               Employe_HolidayView.Tableau.getModel();
46         tableModel.setRowCount(0); // R initialiser le tableau
47         for (Employe e : Employes) {
48             tableModel.addRow(new Object[]{e.getId(), e.getNom(),
                   e.getPrenom(), e.getEmail(), e.getTelephone(),
                   e.getSalaire(), e.getRole(), e.getPost(),
                   e.getSolde()});
49         }
50         View.remplaire_les_employes();
51     }
52
53 // Ajouter un nouvel employ
54     private void addEmploye() {
55 // R cup rer les donn es saisies dans la vue
56         String nom = View.getNom();
57         String prenom = View.getPrenom();
58         String email = View.getEmail();
59         String telephone = View.getTelephone();
```

```
60          double salaire = View.getSalaire();
61          Role role = View.getRole();
62          Post poste = View.getPoste();
63
64  // Vider les champs de saisie apr s la r cup ration des donn es
65          View.viderChamps_em();
66  // Ajouter l'employ  dans le mod le
67          boolean addreussi = model_employe.addEmploye(0, nom, prenom,
                email, telephone, salaire, role, poste, 25);
68  // Afficher un message de succ s ou d'erreur
69          if (addreussi) {
70              View.afficherMessageSucces("Employ  ajout  avec succ s
                    !");
71              displayEmploye();
72          } else {
73              View.afficherMessageErreur(" chec  de l'ajout de
                    l'employ .");
74          }
75      }
76  // Supprimer un employ
77      private void deleteEmploye() {
78          int selectedrow = Employe_HolidayView.Tableau.getSelectedRow();
79          if (selectedrow == -1) {
80              View.afficherMessageErreur("Veuillez s lectionner une
                    ligne.");
81          } else {
82  // R cup rer l'ID de l'employ     supprimer
83              int id = (int)
                    Employe_HolidayView.Tableau.getValueAt(selectedrow, 0);
84              if (model_employe.deleteEmploye(id)) {
85                  View.afficherMessageSucces("Employ  supprim  avec
                        succ s !");
86                  displayEmploye();
87              } else {
88                  View.afficherMessageErreur(" chec  de la suppression
                        de l'employ .");
89              }
90          }
91      }
```

```
92
93  // S lectionner un employ  pour mise    jour    partir du tableau
94      private void updateEmployebyselect() {
95          int selectedrow = Employe_HolidayView.Tableau.getSelectedRow();
96
97          if (selectedrow == -1) {
98              return; // Aucun employ  s lectionn
99          }
100         try {
101 // R cup rer les donn es de l'employ  s lectionn
102             id = (int)
                    Employe_HolidayView.Tableau.getValueAt(selectedrow, 0);
103             nom = (String)
                    Employe_HolidayView.Tableau.getValueAt(selectedrow, 1);
104             prenom = (String)
                    Employe_HolidayView.Tableau.getValueAt(selectedrow, 2);
105             email = (String)
                    Employe_HolidayView.Tableau.getValueAt(selectedrow, 3);
106             telephone = (String)
                    Employe_HolidayView.Tableau.getValueAt(selectedrow, 4);
107             salaire = (double)
                    Employe_HolidayView.Tableau.getValueAt(selectedrow, 5);
108             role = (Role)
                    Employe_HolidayView.Tableau.getValueAt(selectedrow, 6);
109             poste = (Post)
                    Employe_HolidayView.Tableau.getValueAt(selectedrow, 7);
110             solde = (int)
                    Employe_HolidayView.Tableau.getValueAt(selectedrow, 8);
111             View.remplaireChamps_em(id, nom, prenom, email, telephone,
                    salaire, role, poste);
112             test = true; // Indiquer qu'une ligne est s lectionn e
113         } catch (Exception e) {
114             View.afficherMessageErreur("Erreur lors de la
                    r cup ration des donn es.");
115         }
116     }
117
118 // Mettre    jour les informations d'un employ
119     private void updateEmploye() {
```

```
120         if (!test) {
121             View.afficherMessageErreur("Veuillez d'abord s lectionner
                   une ligne    modifier.");
122             return;
123         }
124         try {
125 // R cup rer les nouvelles donn es saisies
126             nom = View.getNom();
127             prenom = View.getPrenom();
128             email = View.getEmail();
129             telephone = View.getTelephone();
130             salaire = View.getSalaire();
131             role = View.getRole();
132             poste = View.getPoste();
133 
134 // Effectuer la mise    jour dans le mod le
135             boolean updateSuccessful = model_employe.updateEmploye(id,
                   nom, prenom, email, telephone, salaire, role, poste,
                   solde);
136 
137             if (updateSuccessful) {
138                 test = false; // R initialiser l'indicateur
139                 View.afficherMessageSucces("Employ  mis    jour avec
                       succ s.");
140                 displayEmploye();
141                 View.viderChamps_em(); // Vider les champs
142             } else {
143                 View.afficherMessageErreur("Erreur lors de la mise
                       jour de l'employ .");
144             }
145         } catch (Exception e) {
146             View.afficherMessageErreur("Erreur lors de la mise
                   jour.");
147         }
148     }
149 
150 // R initialiser le solde des cong s de tous les employ s au d but
    de l'ann e
151     public void resetSolde() {
```

```
152         Calendar now = Calendar.getInstance();
153         if (now.get(Calendar.DAY_OF_YEAR) == 1) {
154             for (Employe employe : model_employe.displayEmploye()) {
155                 updateSolde(employe.getId(), 25);
156             }
157         }
158     }
159 // Mettre    jour le solde des cong s pour un employ  sp cifique
160     public static void updateSolde(int id, int solde) {
161         boolean updateSuccessful = model_employe.updateSolde(id,
                solde);
162     }
163 }
```

4.7 Couche Model : Implémentation du Model(Holiday)

J'ai conçu une classe **Holiday** avec des attributs pour stocker les informations essentielles liées aux congés, comme l'ID du congé, l'ID de l'employé, les dates de début et de fin, ainsi que le type de congé. J'ai inclus des constructeurs, des getters pour récupérer ces informations, et une méthode getSolde() à implémenter pour gérer le solde des congés restants.

```
1  package Model;
2  import java.sql.Date;
3  public class Holiday{
4      private int id_holiday;
5      private int id_employe;
6      private Date startDate;
7      private Date endDate;
8      private Type_holiday type;
9  public Holiday(int id_holiday, int id_employe,Date startDate, Date
       endDate , Type_holiday type){
10     this.id_holiday = id_holiday;
11     this.id_employe = id_employe;
12     this.startDate = startDate;
13     this.endDate = endDate;
14     this.type = type;
15 }
```

```
public int getId_holiday() {
    return id_holiday;
}
public Date getStartDate() {
    return startDate;
}
public Date getEndDate() {
    return endDate;
}
public Type_holiday getType() {
    return type;
}
public int getId_employe() {
    return id_employe;
}

    public Object getSolde() {
        throw new UnsupportedOperationException("Not supported yet.");
    }

}
// enum TypeHoliday
public enum TypeHoliday {
        ANNUAL, SICK, MATERNITY, OTHER
}
```

4.8 Interface Générique pour la Gestion des Opérations CRUD

J'utilise l'interface GenericDAOI¡T¿ pour définir des opérations CRUD génériques, telles que add, delete, update et display, afin de gérer les entités de manière flexible et réutilisable.

```
package DAO;
import java.util.List;
public interface GenericDAOI <T> {
    public void add(T e);
    public void delete(int id);
    public void update(T e);
    public List<T> display();
}
```

4.9 Couche DAO) :Creation de class qui implémente l'interface (HolidayDAOImpl):

Dans cette implémentation, j'ai utilisé la classe **HolidayDAOimpl** pour gérer les opérations de base de données liées aux congés des employés. Les fonctionnalités incluent l'ajout d'une demande de congé avec vérification du solde, la suppression et la mise à jour des congés, ainsi que l'affichage de tous les congés. En utilisant JDBC avec des requêtes SQL préparées, j'ai assuré une gestion efficace et sécurisée des données, tout en respectant les règles de gestion métier.

```
1  package DAO;
2  import Model.Holiday;
3  import Model.Type_holiday;
4  import java.sql.Date;
5  import java.sql.PreparedStatement;
6  import java.sql.ResultSet;
7  import java.sql.SQLException;
8  import java.util.ArrayList;
9  import java.util.List;
10
11 public class HolidayDAOimpl implements GenericDAOI<Holiday>{
12    //M thode pour ajouter un cong  pour un employ
13     @Override
14     public void add(Holiday e) {
15         String checkSoldeSql = "SELECT solde FROM employe WHERE id =
               ?";
16         String insertHolidaySql = "INSERT INTO holiday (id_employe,
               startdate, enddate, type) VALUES (?, ?, ?, ?)";
17
18         try (PreparedStatement checkStmt =
               DBConnexion.getConnexion().prepareStatement(checkSoldeSql))
               {
19             // R cup rer le solde de cong  de l'employ
20             checkStmt.setInt(1, e.getId_employe());
21             ResultSet rs = checkStmt.executeQuery();
22
23             if (rs.next()) {
24                 int solde = rs.getInt("solde");
```

```
25
26              // Calculer le nombre de jours demand s
27                long daysBetween =
                      java.time.temporal.ChronoUnit.DAYS.between(
28                    e.getStartDate().toLocalDate(),
29                    e.getEndDate().toLocalDate()
30                );
31
32                if (daysBetween > solde) {
33                    System.err.println("Le solde de cong  est
                          insuffisant.");
34                    return;
35                }
36
37              // Ins rer la demande de cong
38                try (PreparedStatement insertStmt =
                      DBConnexion.getConnexion().prepareStatement
39                (insertHolidaySql)) {
40                    insertStmt.setInt(1, e.getId_employe());
41                    insertStmt.setDate(2, e.getStartDate());
42                    insertStmt.setDate(3, e.getEndDate());
43                    insertStmt.setString(4, e.getType().name());
44
45                    insertStmt.executeUpdate();
46
47                  // Mettre    jour le solde de cong
48                    String updateSoldeSql = "UPDATE employe SET solde=
                          solde - ? WHERE id = ?";
49                    try (PreparedStatement updateStmt
                          DBConnexion.getConnexion().prepareStatement
50                    (updateSoldeSql)) {
51                        updateStmt.setInt(1, (int) daysBetween);
52                        updateStmt.setInt(2, e.getId_employe());
53                        updateStmt.executeUpdate();
54                    }
55                }
56            } else {
57                System.err.println("Employ  introuvable.");
58            }
```

```
59          } catch (SQLException | ClassNotFoundException exception) {
60              exception.printStackTrace();
61          }
62      }
63      // M thode pour supprimer un cong  existant par son ID
64      @Override
65      public void delete(int id) {
66          String sql = "DELETE FROM holiday WHERE id = ?";
67          try (PreparedStatement stmt =
                DBConnexion.getConnexion().prepareStatement(sql)) {
68              stmt.setInt(1,id);
69              stmt.executeUpdate();
70          } catch (SQLException exception) {
71              System.err.println("failed of delete holiday");
72          } catch (ClassNotFoundException ex) {
73              System.err.println("failed connexion with data base");
74          }
75      }
76      // M thode pour mettre    jour un cong  existant
77      @Override
78      public void update(Holiday e) {
79          String sql = "UPDATE holiday SET id_employe = ?, startdate =
                ?, enddate = ?, type = ? WHERE id = ?";
80          try (PreparedStatement stmt =
                DBConnexion.getConnexion().prepareStatement(sql)){
81              stmt.setInt(1, e.getId_employe());
82              stmt.setDate(2, e.getStartDate());
83              stmt.setDate(3, e.getEndDate());
84              stmt.setString(4, e.getType().name());
85              stmt.setInt(5,e.getId_holiday());
86              stmt.executeUpdate();
87          } catch (SQLException exception) {
88              System.err.println("failed of update holiday");
89          } catch (ClassNotFoundException ex) {
90              System.err.println("failed connexion with data base");
91          }
92      }
93      // M thode pour afficher tous les cong s
94      @Override
```

```
95      public List<Holiday> display() {
96          String sql = "SELECT * FROM holiday";
97          List<Holiday> Holidays = new ArrayList<>();
98          try (PreparedStatement stmt =
                DBConnexion.getConnexion().prepareStatement(sql)) {
99              ResultSet re = stmt.executeQuery();
100             while (re.next()) {
101                 int id = re.getInt("id");
102                 int id_employe = re.getInt("id_employe");
103                 Date startdate = re.getDate("startdate");
104                 Date enddate = re.getDate("enddate");
105                 String type = re.getString("type");
106                 Holiday e = new Holiday(id, id_employe, startdate,
                        enddate, Type_holiday.valueOf(type));
107                 Holidays.add(e);
108             }
109         } catch (ClassNotFoundException ex) {
110             System.err.println("Failed to connect with database: " +
                    ex.getMessage());
111         } catch (SQLException ex) {
112             System.err.println("Failed to fetch holidays: " +
                    ex.getMessage());
113         }
114         return Holidays;
115     }
116
117 }
```

4.10 (Couche Model) : HolidayModel

J'ai créé la classe **HolidayModel** pour gérer les opérations liées aux congés des employés. Cette classe interagit avec le DAO HolidayDAOimpl pour effectuer des actions CRUD (ajouter, supprimer, mettre à jour et afficher) sur les congés. J'ai inclus des validations pour vérifier que les dates de congé sont correctes (la date de début avant la date de fin et que les dates ne sont pas passées). J'ai aussi vérifié que le solde de congé de l'employé est suffisant avant de procéder à l'ajout ou à la mise à jour. Enfin, j'ai mis à jour le solde de congé de l'employé après chaque modification.

```java
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;
    }
    // Fonction pour ajouter un cong  pour un employ , avec diverses validations
    public boolean addHoliday(int id, int id_employe, Date startdate, Date enddate, Type_holiday 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;

     // Calcul du nombre de jours entre la date de d but et la date de fin
        long daysBetween = (enddate.toLocalDate().toEpochDay() -
            startdate.toLocalDate().toEpochDay());
        if(daysBetween > targetEmploye.getSolde()) return false;
        EmployeController.updateSolde(targetEmploye.getId(),
            targetEmploye.getSolde() - (int) daysBetween);

     // Cr ation de l'objet Holiday et ajout dans la base de donn es via le DAO
        Holiday e = new Holiday(id, id_employe, startdate, enddate,
            type);
        dao.add(e);
        return true;
    }

    // Fonction pour afficher tous les cong s
```

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

4.11 (Couche Controller) : l'implementation du Holidaycontroller

```
1  package Controller;
2  import DAO.EmployeDAOimpl;
3  import Model.*;
4  import View.*;
5  import java.sql.Date;
6  import java.util.List;
7  import javax.swing.table.DefaultTableModel;
8
9  public class HolidayController {
10     private final Employe_HolidayView View;
11     public HolidayModel model_holiday;
12     public static int id = 0;
13     public static int oldselectedrow = -1;
14     public static boolean test = false;
15     int id_employe = 0;
16     String nom_employe = "";
17     public static String OldstartDate = null;
18     public static String OldendDate = null;
19     Type_holiday type = null;
20     int oldsolde = 0;
21     int solde = 0;
22     boolean updatereussi = false;
23     Employe targetEmploye = null;
24     // Constructeur
25     public HolidayController(Employe_HolidayView view, HolidayModel
           model) {
26         this.View = view;
27         this.model_holiday = model;
28
29         // Ajout des gestionnaires d'   vnements    aux boutons
30         View.getdeleteButton_holiday().addActionListener(e ->
               deleteHoliday());
31         View.getupdateButton_holiday().addActionListener(e ->
               updateHoliday());
32         Employe_HolidayView.Tableau1.getSelectionModel().
33         addListSelectionListener(e -> updateHolidaybyselect());
```

```
34          View.getaddButton_holiday().addActionListener(e ->
                addHoliday());
35          View.getdisplayButton_holiday().addActionListener(e ->
                displayHoliday());
36      }
37
38      // Ajouter un cong
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          Type_holiday type = View.getType_holiday();
44
45          View.viderChamps_ho(); // Vider les champs de saisie
46
47          // Recherche de l'employ  correspondant
48          Employe targetEmploye = null;
49          for (Employe employe : new EmployeModel(new
                EmployeDAOimpl()).displayEmploye()) {
50              if (employe.getId() == id_employe) {
51                  targetEmploye = employe;
52                  break;
53              }
54          }
55
56          if (targetEmploye == null) {
57              View.afficherMessageErreur("Cet employ  n'existe pas.");
58              return;
59          }
60
61          // Calcul de la dur e du cong
62          long daysBetween = java.time.temporal.ChronoUnit.DAYS.between(
63              startDate.toLocalDate(),
64              endDate.toLocalDate()
65          );
66
67          if (daysBetween <= 0) {
68              View.afficherMessageErreur("Les dates de d but et de fin
                    sont invalides.");
```

```
69              return;
70          }
71
72          // V rification du solde de cong
73          if (targetEmploye.getSolde() < daysBetween) {
74              View.afficherMessageErreur("Le solde de cong  de
                    l'employ  est insuffisant.");
75              return;
76          }
77
78          try {
79              // Tentative d'ajout du cong
80              boolean addReussi = model_holiday.addHoliday(0,
                    id_employe, startDate, endDate, type, targetEmploye);
81
82              if (addReussi) {
83                  targetEmploye.setSolde(targetEmploye.getSolde() -
                        (int) daysBetween); // Mise    jour du solde
84                  View.afficherMessageSucces("Holiday a   t   ajout e
                        avec succ s.");
85              } else {
86                  View.afficherMessageErreur(" chec  : L'ajout de la
                        Holiday a   chou  .");
87              }
88          } catch (Exception e) {
89              e.printStackTrace();
90              View.afficherMessageErreur("Erreur lors de l'ajout : " +
                    e.getMessage());
91          }
92      }
93
94      // Afficher les cong s
95      private void displayHoliday() {
96          List<Holiday> Holidays = model_holiday.displayHoliday();
97
98          if (Holidays == null || Holidays.isEmpty()) {
99              View.afficherMessageErreur("Aucune holiday.");
100             return; // Retour si la liste est vide
101         }
```

```
102
103         // Initialisation du tableau
104         DefaultTableModel tableModel1 = (DefaultTableModel)
                Employe_HolidayView.Tableau1.getModel();
105         tableModel1.setRowCount(0); // R initialisation des lignes du
                tableau
106
107         // Remplissage des donn es
108         for (Holiday e : Holidays) {
109             String nom_employe = null;
110             List<Employe> Employes = new EmployeModel(new
                    EmployeDAOimpl()).displayEmploye();
111             for (Employe em : Employes) {
112                 if (em.getId() == e.getId_employe()) {
113                     nom_employe = em.getId() + " - " + em.getNom() + "
                            " + em.getPrenom();
114                     break;
115                 }
116             }
117             tableModel1.addRow(new Object[]{e.getId_holiday(),
                    nom_employe, e.getStartDate(), e.getEndDate(),
                    e.getType()});
118         }
119         View.remplaire_les_employes();
120     }
121
122     // Supprimer un cong
123     private void deleteHoliday() {
124         int selectedrow =
                Employe_HolidayView.Tableau1.getSelectedRow();
125         if (selectedrow == -1) {
126             View.afficherMessageErreur("Veuillez selectionner une
                    ligne.");
127         } else {
128             // Logique de suppression d'un cong
129             int id = (int)
                    Employe_HolidayView.Tableau1.getValueAt(selectedrow, 0);
130             boolean deletereussi = model_holiday.deleteHoliday(id);
131             if (deletereussi) {
```

```
132                 View.afficherMessageSucces("Holiday a bien   t
                        supprim .");
133                 displayHoliday(); // Rafra chir l'affichage
134             } else {
135                 View.afficherMessageErreur("Holiday n'a pas   t
                        supprim .");
136             }
137         }
138     }
139
140     // S lection d'une ligne pour mise    jour
141     private void updateHolidaybyselect() {
142         int selectedrow =
                Employe_HolidayView.Tableau1.getSelectedRow();
143         if (selectedrow == -1) {
144             return;
145         }
146         try {
147             id = (int)
                    Employe_HolidayView.Tableau1.getValueAt(selectedrow, 0);
148             nom_employe = (String)
                    Employe_HolidayView.Tableau1.getValueAt(selectedrow, 1);
149             id_employe = Integer.parseInt(nom_employe.split(" - ")[0]);
150             OldstartDate =
                    String.valueOf(Employe_HolidayView.Tableau1.getValueAt
151             (selectedrow, 2));
152             OldendDate =
                    String.valueOf(Employe_HolidayView.Tableau1.getValueAt
153             (selectedrow, 3));
154             type = (Type_holiday)
                    Employe_HolidayView.Tableau1.getValueAt(selectedrow, 4);
155             View.remplaireChamps_ho(id_employe, OldstartDate,
                    OldendDate, type);
156             test = true; // Marquer la s lection
157         } catch (Exception e) {
158             View.afficherMessageErreur("Erreur lors de la
                    r cup ration des donn es");
159         }
160     }
```

```
161
162     // Mise    jour d'un cong
163     private void updateHoliday() {
164         if (!test) {
165             View.afficherMessageErreur("Veuillez d'abord s lectionner
                    une ligne    modifier.");
166             return;
167         }
168         try {
169             nom_employe = View.getNom();
170             Date startDate_holiday = Date.valueOf(View.getStartDate());
171             Date endDate_holiday = Date.valueOf(View.getEndDate());
172             type = View.getType_holiday();
173             id_employe = View.getId_employe();
174
175             int olddaysbetween = (int)
                    (Date.valueOf(OldendDate).toLocalDate().toEpochDay() -
                    Date.valueOf(OldstartDate).toLocalDate().toEpochDay());
176
177             for (Employe employe : new EmployeModel(new
                    EmployeDAOimpl()).displayEmploye()) {
178                 if (employe.getId() == id_employe) {
179                     targetEmploye = employe;
180                     break;
181                 }
182             }
183
184             boolean updateSuccessful = model_holiday.updateHoliday(id,
                    id_employe, startDate_holiday, endDate_holiday, type,
                    targetEmploye, olddaysbetween);
185
186             if (updateSuccessful) {
187                 test = false; // R initialiser le test
188                 View.afficherMessageSucces("Holiday a   t   modifi
                        avec succ s.");
189                 displayHoliday(); // Rafra chir l'affichage
190                 View.viderChamps_ho(); // Vider les champs
191             } else {
192                 View.afficherMessageErreur("Erreur lors de la mise
```

```
                    jour de holiday.");
193             }
194         } catch (Exception e) {
195             View.afficherMessageErreur("Erreur lors de la mise
                    jour");
196         }
197     }
198 }
```

4.12 (Couche View) :Interface graphique EmployeHolidayView

La classe **EmployeHolidayView** est une interface graphique (GUI) qui permet de gérer les employés et leurs congés à l'aide de deux onglets distincts : un pour les employés et un pour les congés.

- **Fonctionnalités :**

Onglet Employé : Permet d'ajouter, modifier, supprimer ou afficher les informations des employés (nom, prénom, email, téléphone, salaire, rôle, poste). Affiche une liste des employés sous forme de tableau (JTable). Offre des champs pour saisir ou modifier les informations d'un employé.

Onglet Congé : Permet d'ajouter, modifier, supprimer ou afficher les informations des congés (employé, dates de début et de fin, type de congé). Affiche un tableau des congés. Offre des champs pour saisir ou modifier les informations d'un congé.

```
1  package View;
2
3  import DAO.EmployeDAOimpl;
4  import Model.Employe;
5  import Model.EmployeModel;
6  import Model.Post;
7  import Model.Role;
8  import Model.Type_holiday;
9  import java.awt.*;
10 import javax.swing.*;
11 import javax.swing.table.DefaultTableModel;
12 import java.util.List;
13
```

```
14 public class Employe_HolidayView extends JFrame {
15     // le tableau de employe et cong
16     private JTabbedPane tabbedPane = new JTabbedPane();
17 
18     // les tabs
19     private JPanel employeTab = new JPanel();
20     private JPanel holidayTab = new JPanel();
21 
22     // les panels
23     private JPanel Employepan = new JPanel();
24     private JPanel Holidaypan = new JPanel();
25     private JPanel Display_Table_employe = new JPanel();
26     private JPanel Display_Table_holiday = new JPanel();
27     private final JPanel Forme_employe = new JPanel();
28     private final JPanel Forme_holiday = new JPanel();
29     private JPanel panButton_employe = new JPanel();
30     private JPanel panButton_holiday = new JPanel();
31 
32     // les labels du l'employe
33     private JLabel label_nom = new JLabel("Nom");
34     private JLabel label_prenom = new JLabel("Prenom");
35     private JLabel label_email = new JLabel("Email");
36     private JLabel label_tele = new JLabel("Telephone");
37     private JLabel label_salaire = new JLabel("Salaire");
38     private JLabel label_role = new JLabel("Role");
39     private JLabel label_poste = new JLabel("Poste");
40 
41     // les labels du cong
42     private JLabel label_employe = new JLabel("Nom de l'employ ");
43     private JLabel label_startDate = new JLabel("Date de debut");
44     private JLabel label_endDate = new JLabel("Date de fin");
45     private JLabel label_type = new JLabel("Type");
46     private JComboBox<Type_holiday> TypeComboBox = new
        JComboBox<>(Type_holiday.values());
47 
48     // les textfield du l'employe
49     private JTextField text_nom = new JTextField();
50     private JTextField text_prenom = new JTextField();
51     private JTextField text_email = new JTextField();
```

```
52      private JTextField text_tele = new JTextField();
53      private JTextField text_salaire = new JTextField();
54
55      private JComboBox<Role> roleComboBox = new
            JComboBox<>(Role.values());
56      private JComboBox<Post> posteComboBox = new
            JComboBox<>(Post.values());
57
58      // les textfield du cong
59      private JComboBox<String> text_employe = new JComboBox<>();
60      private JTextField text_startDate = new
            JTextField("year-month-day");
61      private JTextField text_endDate = new JTextField("year-month-day");
62
63      // les boutons du l'employe
64      private JButton addButton_employe = new JButton("Ajouter");
65      private JButton updateButton_employe = new JButton("Modifier");
66      private JButton deleteButton_employe = new JButton("Supprimer");
67      private JButton displayButton_employe = new JButton("Afficher");
68
69      // les boutons du cong
70      private JButton addButton_holiday = new JButton("Ajouter");
71      private JButton updateButton_holiday = new JButton("Modifier");
72      private JButton deleteButton_holiday = new JButton("Supprimer");
73      private JButton displayButton_holiday = new JButton("Afficher");
74
75      // le tableau de l'employe
76      JPanel pan0 = new JPanel(new BorderLayout());
77      public static String[] columnNames_employe = {"ID", "Nom",
            "Prenom", "Email", "T l phone", "Salaire", "Role",
            "Poste","solde"};
78      public static DefaultTableModel tableModel = new
            DefaultTableModel(columnNames_employe, 0);
79      public static JTable Tableau = new JTable(tableModel);
80
81      // le tableau du cong
82      JPanel pan1 = new JPanel(new BorderLayout());
83      public static String[] columnNames_holiday = {"ID",
            "nom_employe","date_debut","date_fin","type"};
```

```
84      public static DefaultTableModel tableModel1 = new
            DefaultTableModel(columnNames_holiday, 0);
85      public static JTable Tableau1 = new JTable(tableModel1);
86
87      public Employe_HolidayView() {
88
89          setTitle("Gestion des employes et des cong s");
90          setSize(1000, 600);
91          setDefaultCloseOperation(EXIT_ON_CLOSE);
92          setLocationRelativeTo(null);
93
94          add(tabbedPane);
95
96          // Employe Tab
97          employeTab.setLayout(new BorderLayout());
98          employeTab.add(Employepan, BorderLayout.CENTER);
99
100         Employepan.setLayout(new BorderLayout());
101         Employepan.add(Display_Table_employe, BorderLayout.CENTER);
102         Tableau.setFillsViewportHeight(true);
103         Dimension preferredSize = new Dimension(900, 500);
104         Tableau.setPreferredScrollableViewportSize(preferredSize);
105         pan0.add(new JScrollPane(Tableau), BorderLayout.CENTER);
106         Display_Table_employe.add(pan0);
107
108         Employepan.add(panButton_employe, BorderLayout.SOUTH);
109         panButton_employe.add(addButton_employe);
110         panButton_employe.add(updateButton_employe);
111         panButton_employe.add(deleteButton_employe);
112         panButton_employe.add(displayButton_employe);
113
114         Employepan.add(Forme_employe, BorderLayout.NORTH);
115         Forme_employe.setLayout(new GridLayout(7, 2, 10, 10));
116         Forme_employe.add(label_nom);
117         Forme_employe.add(text_nom);
118         Forme_employe.add(label_prenom);
119         Forme_employe.add(text_prenom);
120         Forme_employe.add(label_email);
121         Forme_employe.add(text_email);
```

```
122         Forme_employe.add(label_tele);
123         Forme_employe.add(text_tele);
124         Forme_employe.add(label_salaire);
125         Forme_employe.add(text_salaire);
126         Forme_employe.add(label_role);
127         Forme_employe.add(roleComboBox);
128         Forme_employe.add(label_poste);
129         Forme_employe.add(posteComboBox);
130 
131         // Holiday Tab
132         holidayTab.setLayout(new BorderLayout());
133         holidayTab.add(Holidaypan, BorderLayout.CENTER);
134         Holidaypan.setLayout(new BorderLayout());
135         Holidaypan.add(Display_Table_holiday, BorderLayout.CENTER);
136 
137         Tableau1.setFillsViewportHeight(true);
138         Tableau1.setPreferredScrollableViewportSize(preferredSize);
139         pan1.add(new JScrollPane(Tableau1), BorderLayout.CENTER);
140         Display_Table_holiday.add(pan1);
141 
142         Holidaypan.add(Forme_holiday, BorderLayout.NORTH);
143         Forme_holiday.setLayout(new GridLayout(4, 2, 10, 10));
144         Forme_holiday.add(label_employe);
145         Forme_holiday.add(text_employe);
146         Forme_holiday.add(label_startDate);
147         Forme_holiday.add(text_startDate);
148         Forme_holiday.add(label_endDate);
149         Forme_holiday.add(text_endDate);
150         Forme_holiday.add(label_type);
151         Forme_holiday.add(TypeComboBox);
152 
153         Holidaypan.add(panButton_holiday, BorderLayout.SOUTH);
154         panButton_holiday.add(addButton_holiday);
155         panButton_holiday.add(updateButton_holiday);
156         panButton_holiday.add(deleteButton_holiday);
157         panButton_holiday.add(displayButton_holiday);
158 
159         // TabbedPane
160         tabbedPane.addTab("Employe", employeTab);
```

```
161         tabbedPane.addTab("Holiday", holidayTab);
162 
163         remplaire_les_employes();
164         setVisible(true);
165     }
166 
167     public void remplaire_les_employes () {
168         List<Employe> Employes = new EmployeModel(new
                EmployeDAOimpl()).displayEmploye();
169        text_employe.removeAllItems();
170        for (Employe elem : Employes) {
171            text_employe.addItem(elem.getId() + " - " + elem.getNom()+"
                   "+elem.getPrenom());
172        }
173     }
174         // getters
175         public int getId_employe() {
176             return Integer.parseInt(text_employe.getSelectedItem().
177             toString().split(" - ")[0]);
178         }
179         public String getNom() {
180             return text_nom.getText();
181         }
182 
183         public JTable getTable() {
184             return (JTable) Display_Table_employe.getComponent(0);
185         }
186 
187         public String getPrenom() {
188             return text_prenom.getText();
189         }
190 
191         public String getEmail() {
192             return text_email.getText();
193         }
194 
195         public String getTelephone() {
196             return text_tele.getText();
197         }
```

```
198
199         public double getSalaire() {
200             return Double.parseDouble(text_salaire.getText());
201         }
202
203         public Role getRole() {
204             return (Role) roleComboBox.getSelectedItem();
205         }
206
207         public Post getPoste() {
208             return (Post) posteComboBox.getSelectedItem();
209         }
210
211         public JButton getaddButton_employe () {
212             return addButton_employe;
213         }
214
215         public JButton getupdateButton_employe () {
216             return updateButton_employe;
217         }
218
219         public JButton getdeleteButton_employe () {
220             return deleteButton_employe;
221         }
222
223         public JButton getdisplayButton_employe () {
224             return displayButton_employe;
225         }
226
227         public JButton getaddButton_holiday () {
228             return addButton_holiday;
229         }
230
231         public JButton getupdateButton_holiday () {
232             return updateButton_holiday;
233         }
234         public JButton getdeleteButton_holiday () {
235             return deleteButton_holiday;
236         }
```

```
237
238         public JButton getdisplayButton_holiday () {
239             return displayButton_holiday;
240         }
241         public String getStartDate () {
242             return text_startDate.getText();
243         }
244
245         public String getEndDate() {
246             return text_endDate.getText();
247         }
248
249         public Type_holiday getType_holiday(){
250             return (Type_holiday) TypeComboBox.getSelectedItem();
251         }
252
253         // methods d'affichage des messages
254         public void afficherMessageErreur(String message) {
255             JOptionPane.showMessageDialog(this, message, "Erreur",
                    JOptionPane.ERROR_MESSAGE);
256         }
257
258         public void afficherMessageSucces(String message) {
259             JOptionPane.showMessageDialog(this, message, "Succ s",
                    JOptionPane.INFORMATION_MESSAGE);
260         }
261
262         // methodes de vider les champs
263         public void viderChamps_em() {
264             text_nom.setText("");
265             text_prenom.setText("");
266             text_email.setText("");
267             text_tele.setText("");
268             text_salaire.setText("");
269             roleComboBox.setSelectedIndex(0);
270             posteComboBox.setSelectedIndex(0);
271         }
272
273         public void viderChamps_ho() {
```

```
274             text_startDate.setText("");
275             text_endDate.setText("");
276             TypeComboBox.setSelectedIndex(0);
277         }
278 
279         // methodes de remplir les champs
280         public void remplaireChamps_em (int id, String nom, String
                prenom, String email, String telephone, double salaire,
                Role role, Post poste) {
281             text_nom.setText(nom);
282             text_prenom.setText(prenom);
283             text_email.setText(email);
284             text_tele.setText(telephone);
285             text_salaire.setText(String.valueOf(salaire));
286             roleComboBox.setSelectedItem(role);
287             posteComboBox.setSelectedItem(poste);
288         }
289 
290         public void remplaireChamps_ho(int id_employe, String
                date_debut, String date_fin, Type_holiday type) {
291             List<Employe> Employes = new EmployeModel(new
                    EmployeDAOimpl()).displayEmploye();
292             text_employe.removeAllItems();
293             for (Employe elem : Employes) {
294                 if (elem.getId() == id_employe) {
295                     text_employe.addItem(elem.getId() + " - " +
                            elem.getNom()+" "+elem.getPrenom());
296                     text_employe.setSelectedItem(elem.getId() + " - "
                            + elem.getNom()+" "+elem.getPrenom());
297                 }
298             }
299             text_startDate.setText(date_debut);
300             text_endDate.setText(date_fin);
301             TypeComboBox.setSelectedItem(type);
302         }
303 
304         // methodes de test des champs
305         public boolean testChampsVide_em (){
306             return text_nom.getText().equals("") ||
```

```
                text_prenom.getText().equals("") ||
                text_email.getText().equals("") ||
                text_tele.getText().equals("") ||
                text_salaire.getText().equals("");
307         }
308
309         public boolean testChampsVide_ho () {
310             return text_employe.getSelectedItem().equals("") ||
                    text_startDate.getText().equals("") ||
                    text_endDate.getText().equals("") ||
                    TypeComboBox.getSelectedItem().equals("");
311         }
312
313
314 }
```

4.13 (Main) : Application principale

J'ai créé la classe Main pour initialiser l'application en suivant le modèle MVC. Elle lie la vue, le modèle et le contrôleur pour assurer le bon fonctionnement de l'interface et des actions des employés et des congés (holidays). Grâce à cette structure, l'application gère efficacement les informations liées aux employés et à leurs congés tout en garantissant une séparation claire des responsabilités.

```
1  package Main;
2  import Controller.*;
3  import DAO.*;
4  import Model.*;
5  import View.*;
6  public class Main {
7      public static void main(String[] args) {
8          Employe_HolidayView view = new Employe_HolidayView();
9          EmployeDAOimpl dao = new EmployeDAOimpl();
10         HolidayDAOimpl dao_holiday = new HolidayDAOimpl();
11         EmployeModel model_employe = new EmployeModel(dao);
12         HolidayModel model_holiday = new HolidayModel(dao_holiday);
13         new EmployeController(view, model_employe);
14         new HolidayController(view, model_holiday);
```

```
15
16      }
17  }
```

5 Démonstration Visuelle:

5.1 Interface Employé:

J'ai conçu cette interface pour gérer efficacement les employés. Elle comprend :

- **Champs de saisie :** Pour entrer les informations essentielles (nom, rôle, poste, etc.).
- **Menus déroulants :** Pour sélectionner facilement le rôle et le poste.
- **Tableau :** Affiche les données des employés de manière claire et ordonnée.
- **Boutons fonctionnels :** Ajouter, Modifier, Supprimer et Afficher pour gérer les informations.

L'interface est intuitive et centralise les tâches de gestion des employés.

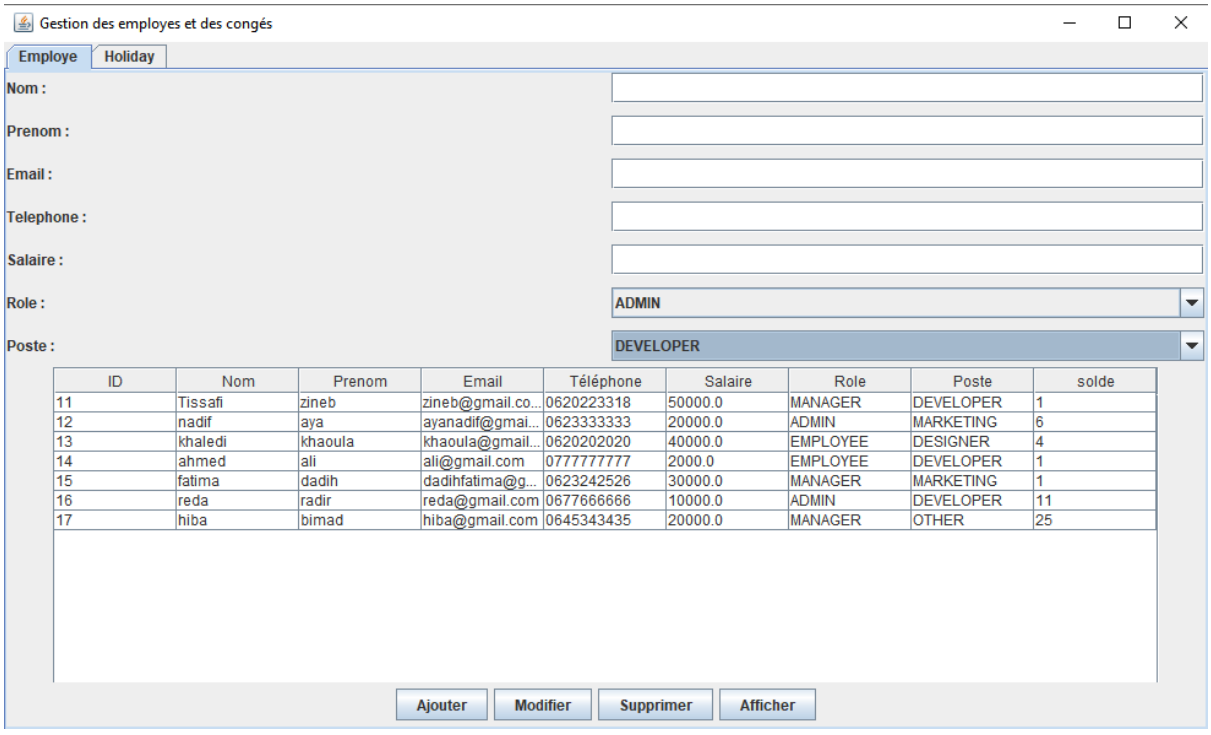

Figure 4: Interface Employé

5.1.1 Le cas d'ajout

Pour effectuer un ajout réussi, il est essentiel de respecter certaines règles. Le numéro de téléphone doit contenir exactement **10 chiffres**, et l'adresse **email doit suivre le format standard**, c'est-à-dire inclure le symbole "@" afin de respecter la logique métier. Ces validations garantissent que les informations saisies sont conformes aux attentes et permettent un traitement correct des données.

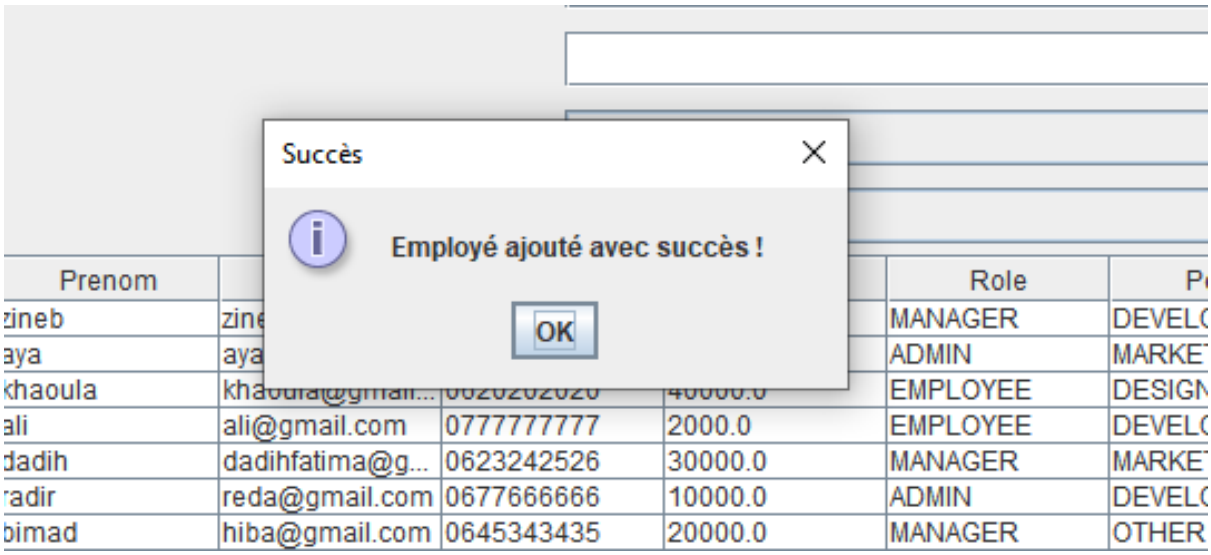

Figure 5: Le cas d'ajout avec succès

5.1.2 Le cas de modification

Pour modifier un employé, il faut tout d'abord **sélectionner une ligne**. Une fois la ligne sélectionnée, les informations de l'employé seront automatiquement remplies.

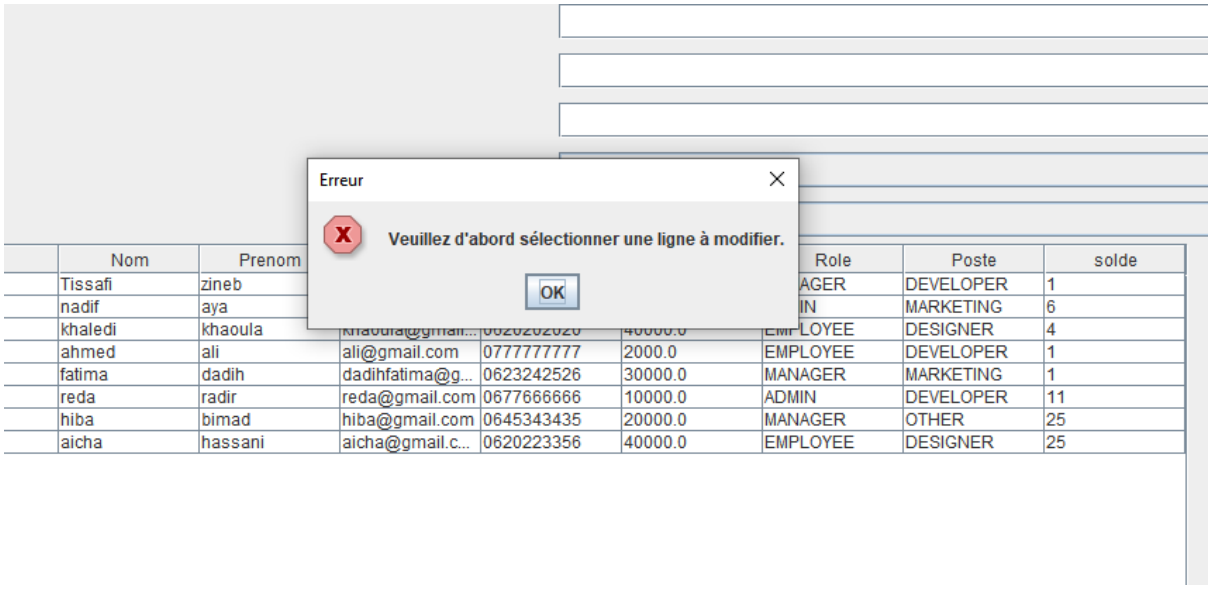

Figure 6: Avant la sélection d'une ligne

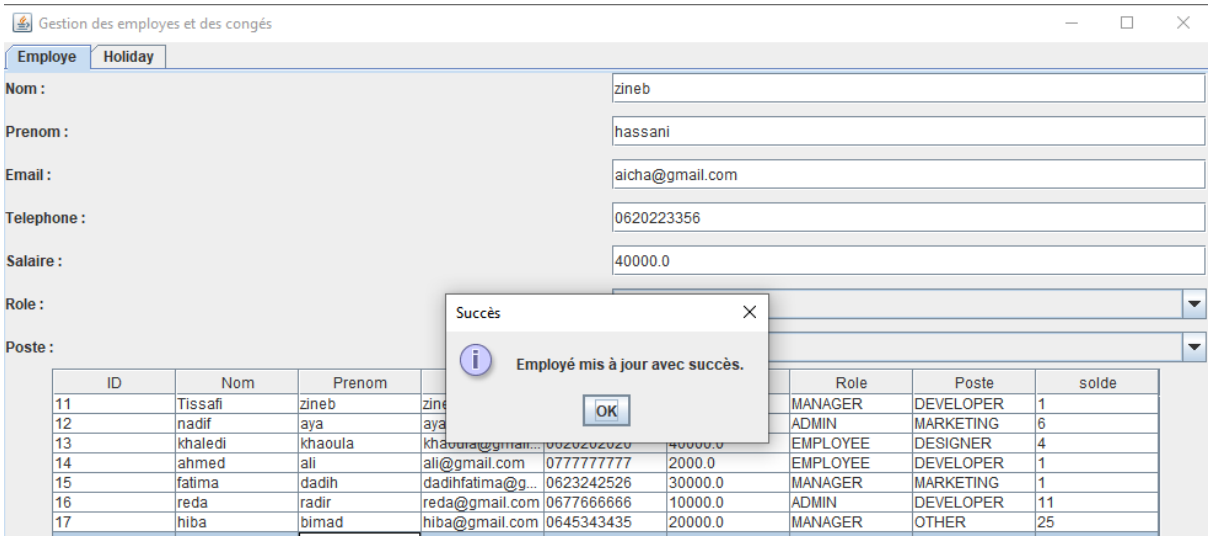

Figure 7: Cas modification avec succès

5.1.3 Le cas de suppression

Pour supprimer un employé, il faut d'abord sélectionner la ligne correspondant à l'employé que l'on souhaite supprimer, puis confirmer que l'on est bien sûr de vouloir supprimer cet employé

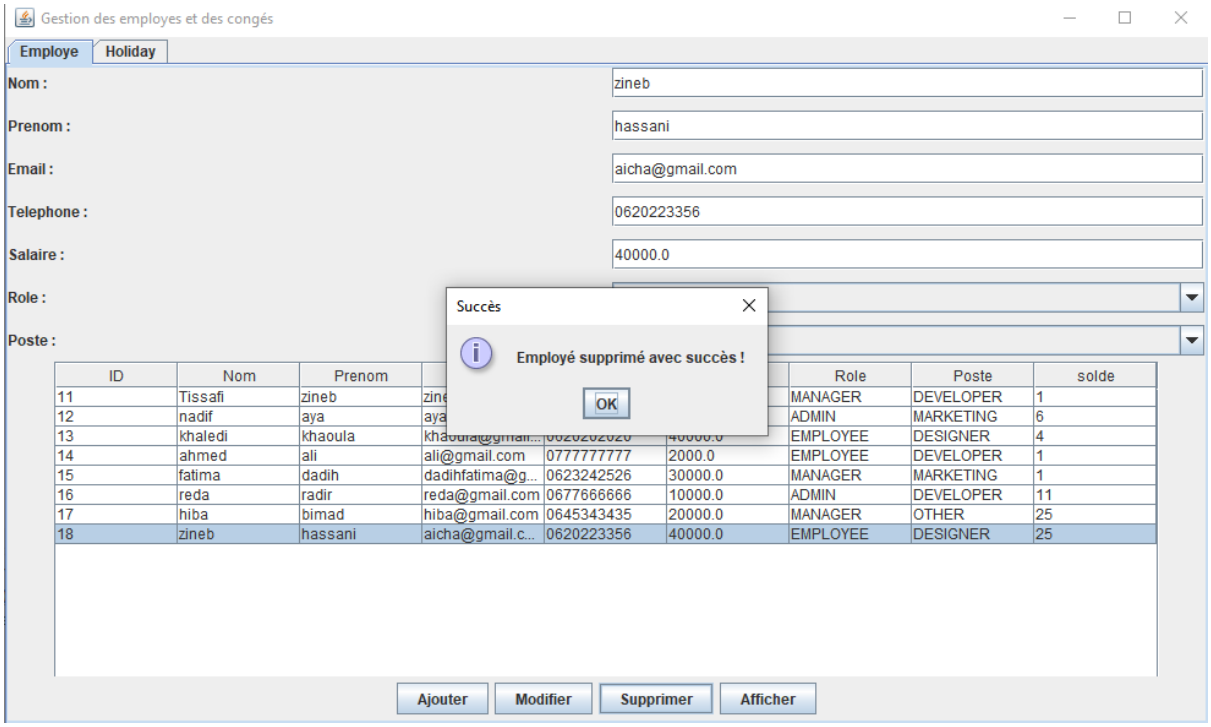

Figure 8: Cas suppression avec succès

5.1.4 Le cas d'affichage

Pour afficher les informations d'un employé, il suffit de cliquer sur le bouton **'Afficher'** après avoir sélectionné la ligne correspondante dans le tableau. Une fois cette action effectuée, les détails de l'employé seront automatiquement affichés. De plus, lors de l'exécution de l'interface, les employés ajoutés précédemment s'affichent directement dans le tableau, offrant ainsi une visualisation instantanée. Lors de l'ajout d'un nouvel employé, ses informations apparaîtront immédiatement après la validation de l'ajout.

5.2 Interface Holiday:

L'interface Holiday permet de gérer efficacement les congés des employés. Elle offre la possibilité de saisir les dates de début et de fin de congé ainsi que la catégorie correspondante. Une fois les informations validées, elles sont immédiatement affichées dans le tableau des congés, garantissant une gestion fluide et en temps réel des demandes de congé. Cette interface assure ainsi un suivi précis et rapide des congés des employés.

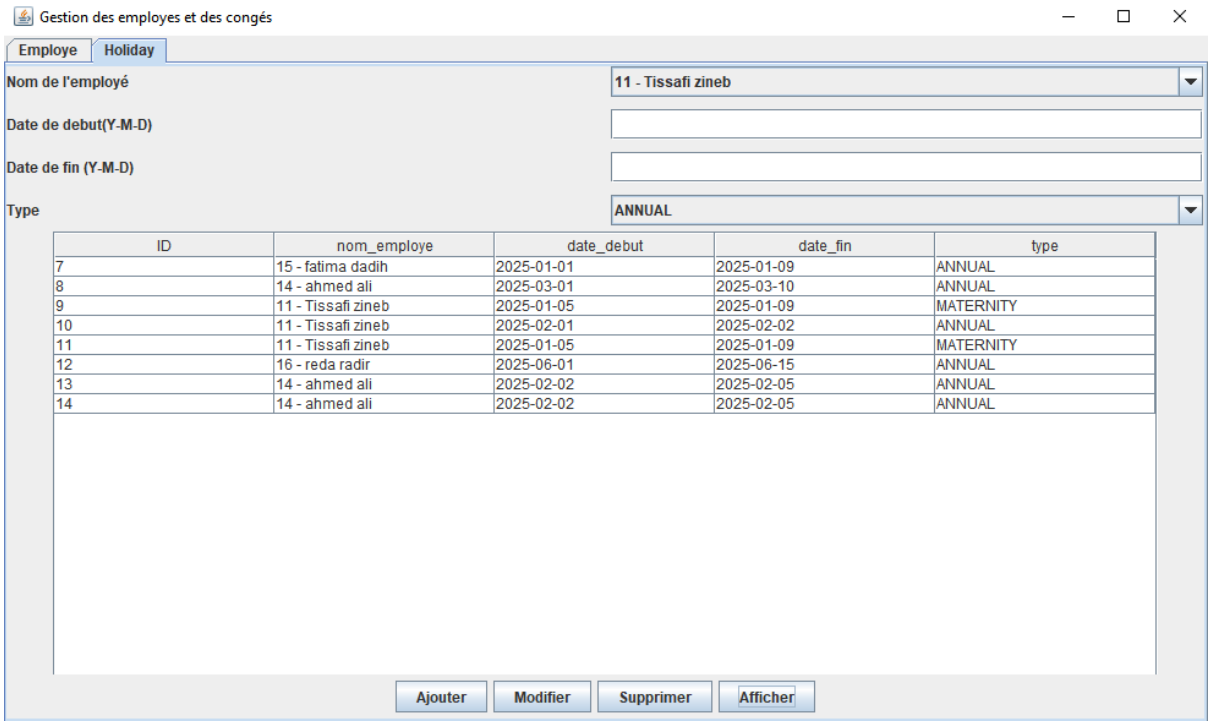

Figure 9: Interface Holiday

5.2.1 Le cas d'ajout

Pour ajouter un congé, il faut respecter les règles suivantes :

- **Format de la date :** La date de début doit être antérieure à la date de fin (exemple : 1er janvier 2024 **Inférieur** 5 janvier 2024).
- **Pas de chevauchement :** Un congé ne doit pas se superposer à une période déjà attribuée (exemple : pas de congé du 3 au 4 janvier si l'employé est déjà en congé du 1er au 5 janvier).

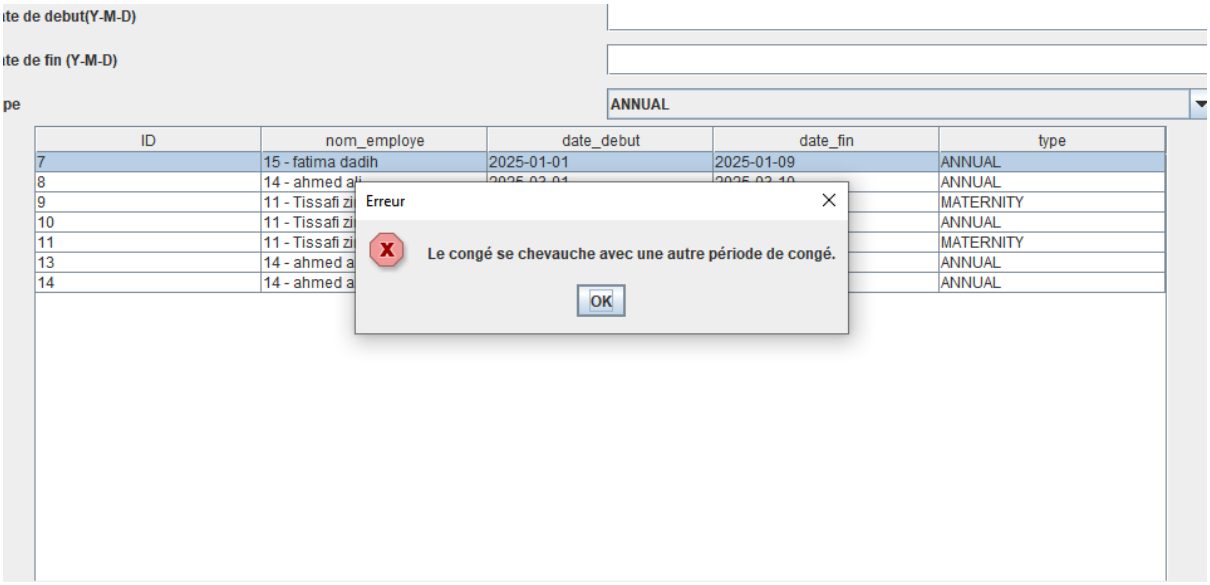

Figure 10: Le cas de chevauchement des dates pour les congés

Cela garantit une gestion correcte des congés.

5.2.2 Le cas de modification et suppression de congé

Pour modifier ou supprimer un congé, il faut suivre les étapes suivantes :

- **Modification :** Sélectionner le congé à modifier, puis ajuster les dates ou autres informations nécessaires. Il est important de vérifier que les nouvelles dates respectent les règles de logique (date de début **Inférieur** date de fin).

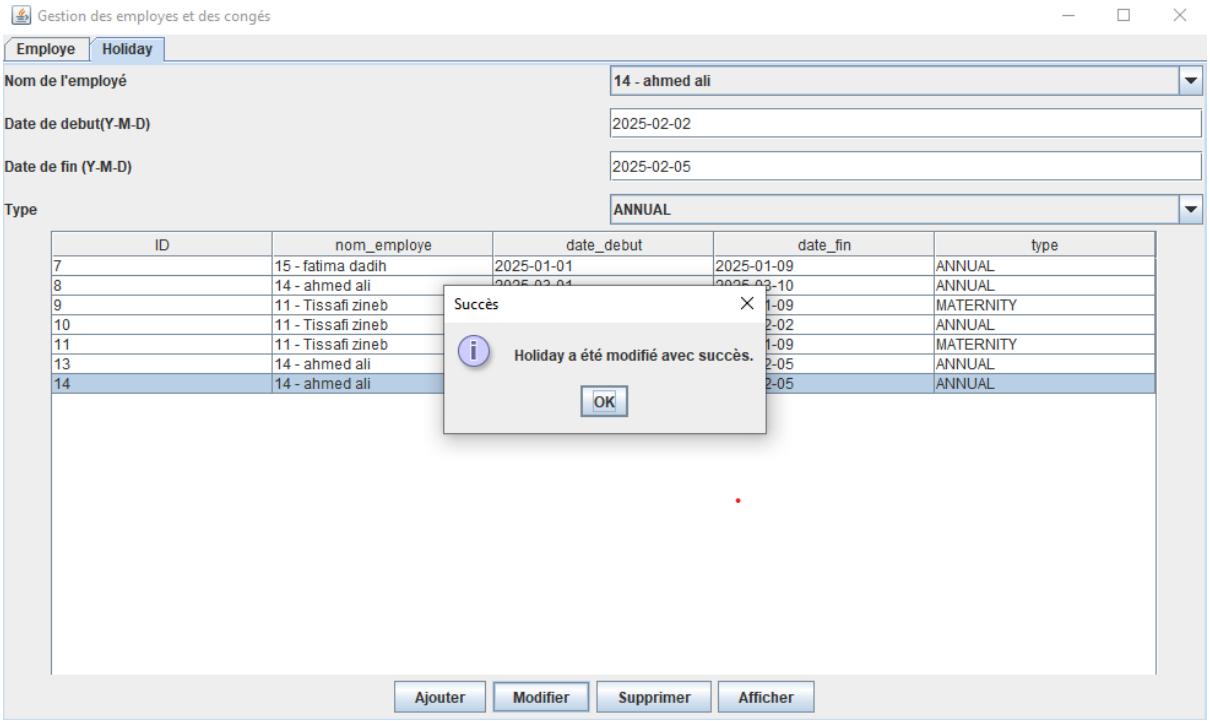

Figure 11: Le cas de modification d'un congé

- **Suppression :** Sélectionner le congé à supprimer et confirmer l'action. Assurez-vous que la suppression ne perturbe pas d'autres plannings ou informations importantes.

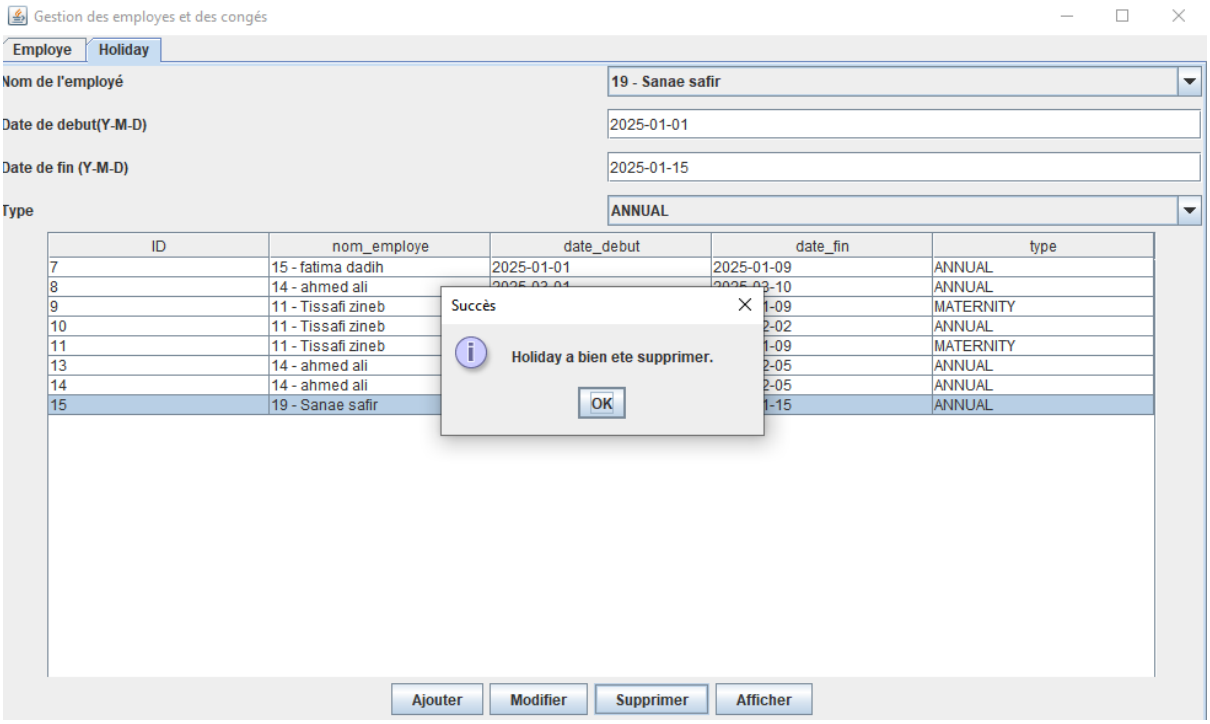

Figure 12: Le cas de Suppression d'un congé