Université Cadi Ayyad
École Supérieure De Technologie-Safi
Département : Informatique
Filière : genie informatique first year

Rapport du TPS 1/2/3 (java avancée)

Gestion des employés et des congés

Réalisé par : EN-NAHLI DOUAA

Encadré par : Mme. ELKHROF Leila

Année Universitaire : 2024/2025

Table des matières

Table des figures

Inroduction

Ce travail pratique (TP) se concentre sur le développement d'une application Java dédiée à la gestion des employés, en adoptant une structure basée sur l'architecture **MVC (Model-View-Controller)**. Ce projet s'inscrit dans le cadre de l'apprentissage des concepts fondamentaux de la programmation orientée objet (POO) et de la mise en œuvre d'interfaces graphiques avec la bibliothèque **Swing**. Il offre également une opportunité d'approfondir les compétences en conception logicielle et en organisation structurée du code pour garantir une séparation claire des responsabilités.

L'objectif principal est de développer une application intuitive et fonctionnelle permettant de manipuler des données d'employés et des congés. L'application est conçue pour gérer l'ajout, la modification, la suppression et l'affichage des informations des employés et de leurs congés, tout en assurant une interface utilisateur fluide et interactive. Grâce à l'implémentation des principes de l'architecture MVC, ce projet garantit une maintenance simplifiée et une évolutivité de l'application.

En complément, ce projet intègre la gestion des fichiers d'entrée/sortie (E/S) pour permettre l'importation et l'exportation des données des employés et de leurs congés. Cette fonctionnalité facilite le traitement des données en exploitant des fichiers au format CSV, permettant ainsi une manipulation efficace des informations, une sauvegarde externe, et une meilleure intégration avec d'autres systèmes. L'utilisation des fichiers E/S illustre également l'importance de la validation et de la gestion des erreurs pour assurer la fiabilité et la robustesse du système.

Les fonctionnalités principales incluent :

- Ajout d'employés avec des informations complètes.
- Modification des données des employés.
- Suppression des employés.
- Affichage de la liste des employés.

ainsi

- Ajout des congés avec des informations complètes.
- Modification des données des congés.
- Suppression des congés.
- Affichage de la liste des congés.

Cette application vise non seulement à développer une solution technique pour la gestion des employés et des congés, mais aussi à démontrer la puissance des concepts de la programmation orientée objet combinée à une architecture bien définie. L'intégration de la gestion des fichiers d'entrée/sortie (E/S) renforce la flexibilité de l'application en permettant l'importation et l'exportation des données. Elle constitue ainsi une étape fondamentale dans l'acquisition des compétences nécessaires pour des projets logiciels plus complexes à l'avenir, en combinant modularité, évolutivité et interaction efficace avec des sources de données externes.

Outils & environnement de travail

1 Environnement de travail

FIGURE 1 – intellij idea logo

,

- **Intellij idea** : est un environnement de développement intégré (IDE) développé par JetBrains, conçu principalement pour le développement en Java. Reconnu pour ses fonctionnalités intelligentes et sa grande efficacité, il prend également en charge de nombreux autres langages et frameworks comme Kotlin, Groovy, Scala, Python.

2 Outils de travail

FIGURE 2 – MySQL Workbench logo

- **MySQL Workbench** : un outil de travail graphique conçu pour faciliter la conception, l'administration, et la gestion des bases de données MySQL. Il fournit une interface utilisateur intuitive permettant de travailler avec des bases de données sans avoir à utiliser uniquement des commandes en ligne.

FIGURE 3 – xampp logo

- **xampp** : En parallèle, le projet vise à fournir des outils de gestion robustes pour le corps administratif, avec une fonctionnalité de multi-rôle, permettant à chaque agent d'accéder à un compte adapté à ses responsabilités spécifique

FIGURE 4 – java developpement kit logo

- **java developpement kit** : st un ensemble d'outils logiciels nécessaires pour développer des applications Java. Il inclut les composants essentiels pour coder, compiler, exécuter et déboguer des programmes Java.

3 Language de Programmation

FIGURE 5 – java logo

- **Java** : un langage de programmation orienté objet et une plateforme largement utilisée pour le développement d'applications logicielles. Il a été créé par Sun Microsystems (maintenant propriété d'Oracle) en 1995 et reste l'un des langages les plus populaires au monde, notamment pour les applications d'entreprise, le développement mobile (Android) et les applications web.

Réalisation

1 Création de la base de donnée

1.1 Script base de donnée

```sql
-- Cr ation de la base de donn es
create database gestion_des_employess;

-- Utilisation de la base de donnes
use gestion_des_employess;

-- Table Employee
CREATE TABLE employee (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
  salaire DECIMAL(10, 2) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
  phone VARCHAR(15) NOT NULL,
    role VARCHAR(200) not null,
    poste VARCHAR(200) NOT NULL,
    holidayBalance INTEGER DEFAULT 25
);
-- Table Holiday
create table holiday (
  id int auto_increment primary key,
  employee_id int not null,
    type varchar(200),
    start varchar(10) not null,
    end varchar(10) not null,
    CONSTRAINT fk_employee FOREIGN KEY (employee_id) REFERENCES employee(id) ON delete
   cascade
);
```

Listing 1.1 – Script SQL de la base de données

- Ce script est ecrit sur MySQL Workbench pour creation la base de donnée pour etre lier à au code via le driver JDBC pour garantir la gestion .

2 Architecture MVC (Model-View-Controller)

L'architecture MVC est un modèle de conception qui sépare les responsabilités au sein d'une application, facilitant ainsi la gestion et la maintenance du code. Elle repose sur trois composants principaux :

2.1 Model

Le modèle représente les données et la logique métier de l'application. Il gère l'accès aux données, effectue les calculs nécessaires et fournit les informations à la vue.

Employee

```
package Model;

// Classe repr sentant un employ  avec ses attributs et m thodes
public class Employee {
    // Attributs de la classe Employee
    private int id; // Identifiant unique de l'employ
    private String nom; // Nom de l'employ
    private String prenom; // Pr nom de l'employ
    private double salaire; // Salaire de l'employ
    private String email; // Adresse e-mail de l'employ
    private String phone; // Num ro de t l phone de l'employ
    private Role role; // R le de l'employ  (probablement une classe ou
      numration   distincte)
    private Poste poste; // Poste de l'employ  (probablement une classe ou
      numration   distincte)
    private int holidayBalance; // Solde des cong s de l'employ

    // Constructeur pour initialiser les attributs de la classe
    public Employee(int id, String nom, String prenom, double salaire, String
    email, String phone, Role role, Poste poste, int holidayBalance) {
        this.id = id;
        this.nom = nom;
        this.prenom = prenom;
        this.salaire = salaire;
        this.email = email;
        this.phone = phone;
        this.role = role;
        this.poste = poste;
        this.holidayBalance = holidayBalance;
    }

    // M thodes getter et setter pour acc der et modifier les attributs de l'
    employ

    // Getter pour l'identifiant de l'employ
    public int getId() {
        return id;
    }

    // Setter pour l'identifiant de l'employ
    public void setId(int id) {
```

```
38          this.id = id;
39      }
40
41      // Getter pour le nom de l'employ
42      public String getNom() {
43          return nom;
44      }
45
46      // Setter pour le nom de l'employ
47      public void setNom(String nom) {
48          this.nom = nom;
49      }
50
51      // Getter pour le pr nom de l'employ
52      public String getPrenom() {
53          return prenom;
54      }
55
56      // Setter pour le pr nom de l'employ
57      public void setPrenom(String prenom) {
58          this.prenom = prenom;
59      }
60
61      // Getter pour le salaire de l'employ
62      public double getSalaire() {
63          return salaire;
64      }
65
66      // Setter pour le salaire de l'employ
67      public void setSalaire(double salaire) {
68          this.salaire = salaire;
69      }
70
71      // Getter pour l'e-mail de l'employ
72      public String getEmail() {
73          return email;
74      }
75
76      // Setter pour l'e-mail de l'employ
77      public void setEmail(String email) {
78          this.email = email;
79      }
80
81      // Getter pour le num ro de t l phone de l'employ
82      public String getPhone() {
83          return phone;
84      }
85
86      // Setter pour le num ro de t l phone de l'employ
87      public void setPhone(String phone) {
88          this.phone = phone;
89      }
90
91      // Getter pour le r le de l'employ
92      public Role getRole() {
```

```
93          return role;
94      }
95
96      // Setter pour le r le de l'employ
97      public void setRole(Role role) {
98          this.role = role;
99      }
100
101     // Getter pour le poste de l'employ
102     public Poste getPoste() {
103         return poste;
104     }
105
106     // Setter pour le poste de l'employ
107     public void setPoste(Poste poste) {
108         this.poste = poste;
109     }
110
111     // Getter pour le solde des cong s
112     public int getHolidayBalance() {
113         return holidayBalance;
114     }
115
116     // Setter pour le solde des cong s
117     public void setHolidayBalance(int holidayBalance) {
118         this.holidayBalance = holidayBalance;
119     }
120 }
```

Holiday

```
1
2 package Model;
3 // Classe repr sentant les informations li es aux cong s d'un employ
4 public class Holiday {
5     // Attributs de la classe Holiday
6     private int id; // Identifiant unique du cong
7     private int idEmployee; // Identifiant de l'employ  concern
8     private HolidayType type; // Type de cong  (probablement une classe ou
        numration   distincte)
9     private String start; // Date de d but du cong  (au format String, ex: "YYYY-MM-DD
      ")
10    private String end; // Date de fin du cong  (au format String, ex: "YYYY-MM-DD")
11
12    // Constructeur principal pour initialiser les attributs de la classe
13    public Holiday(int id, int idEmployee, HolidayType type, String start, String end) {
14        this.id = id;
15        this.idEmployee = idEmployee;
16        this.type = type;
17        this.start = start;
18        this.end = end;
19    }
20
21    // Constructeur par d faut pour initialiser les attributs avec des valeurs nulles
     ou par d faut
22    public Holiday() {
```

```
23          this.id = 0; // Valeur par d faut pour l'identifiant
24          this.idEmployee = 0; // Valeur par d faut pour l'identifiant de l'employ
25          this.type = null; // Aucun type de cong  initialis
26          this.start = null; // Aucune date de d but initialis e
27          this.end = null; // Aucune date de fin initialis e
28      }
29
30      // M thodes getter et setter pour acc der et modifier les attributs
31
32      // Getter pour l'identifiant du cong
33      public int getId() {
34          return id;
35      }
36
37      // Setter pour l'identifiant du cong
38      public void setId(int id) {
39          this.id = id;
40      }
41
42      // Getter pour l'identifiant de l'employ
43      public int getIdEmployee() {
44          return idEmployee;
45      }
46
47      // Setter pour l'identifiant de l'employ
48      public void setIdEmployee(int idEmployee) {
49          this.idEmployee = idEmployee;
50      }
51
52      // Getter pour le type de cong
53      public HolidayType getType() {
54          return type;
55      }
56
57      // Setter pour le type de cong
58      public void setType(HolidayType type) {
59          this.type = type;
60      }
61
62      // Getter pour la date de d but du cong
63      public String getStart() {
64          return start;
65      }
66
67      // Setter pour la date de d but du cong
68      public void setStart(String start) {
69          this.start = start;
70      }
71
72      // Getter pour la date de fin du cong
73      public String getEnd() {
74          return end;
75      }
76
77      // Setter pour la date de fin du cong
```

```
    public void setEnd(String end) {
        this.end = end;
    }
}
```

EmployerLogic

```
package Model;

import java.util.List;
import DAO.EmployeeDAOImpl;
import Utilities.Utils;
import View.EmployeeView;

// Classe EmployeeModel : Contient la logique m tier pour g rer les employ s
public class EmployeeModel {
    private EmployeeDAOImpl dao; // DAO (Data Access Object) pour interagir avec la base de donn es

    // Constructeur pour initialiser le DAO
    public EmployeeModel(EmployeeDAOImpl dao) {
        this.dao = dao;
    }

    // M thode pour ajouter un nouvel employ
    public void ajouterEmployee(String nom, String prenom, String salaire, String email, String phone, Role role, Poste poste) {
        double salaireDouble = Utils.parseDouble(salaire); // Conversion du salaire en double
        // Validation des champs obligatoires
        if (nom.trim().isEmpty() || prenom.trim().isEmpty() || email.trim().isEmpty() || phone.trim().isEmpty() || salaireDouble == 0) {
            EmployeeView.AjouterFail("Veuillez remplir tous les champs.");
            return;
        }

        // Validation de l'adresse e-mail
        if (!email.matches("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$")) {
            EmployeeView.AjouterFail("Veuillez entrer une adresse email valide.");
            return;
        }

        // Validation du num ro de t l phone
        if (!phone.matches("^0\\d{9}$")) {
            EmployeeView.AjouterFail("Le num ro de t l phone doit contenir 10 chiffres");
            return;
        }

        // Validation du salaire
        if (salaireDouble < 0) {
            EmployeeView.AjouterFail("Le salaire doit  tre  un nombre positif");
            return;
        }

        // Cr ation de l'objet Employee avec un solde de cong s initial de 25
```

```
45          Employee employee = new Employee(0, nom, prenom, salaireDouble, email, phone,
    role, poste, 25);
46          dao.ajouter(employee); // Ajout de l'employ  via le DAO
47      }
48
49      // M thode pour r cup rer la liste des employ s
50      public List<Employee> afficherEmployee() {
51          return dao.afficher();
52      }
53
54      // M thodes de recherche d'employ s par diff rents crit res
55      public List<Employee> findByEmail(String email) {
56          return dao.findByEmail(email);
57      }
58
59      public List<Employee> findByFullName(String firstname, String lastname) {
60          return dao.findByFullName(firstname, lastname);
61      }
62
63      public List<Employee> findByFirstName(String firstname) {
64          return dao.findByFirstName(firstname);
65      }
66
67      public List<Employee> findByLastName(String lastname) {
68          return dao.findByLastName(lastname);
69      }
70
71      public List<Employee> findByPhone(String phone) {
72          return dao.findByPhone(phone);
73      }
74
75      public List<Employee> findBySalaire(double salaire) {
76          return dao.findBySalaire(salaire);
77      }
78
79      // M thode pour supprimer un employ
80      public void supprimerEmployee(int id) {
81          if (EmployeeView.SupprimerConfirmation()) { // Demande de confirmation avant
    suppression
82              dao.supprimer(id);
83          }
84      }
85
86      // M thode pour trouver un employ  par son identifiant
87      public Employee findById(int id) {
88          return dao.findById(id);
89      }
90
91      // M thode pour mettre    jour un employ
92      public void updateEmployee(Employee employee, int id, String nom, String prenom,
    String email, double salaire, String phone, Role role, Poste poste) {
93          // Validation : Au moins un champ doit  tre  renseign
94          if (nom.trim().isEmpty() && prenom.trim().isEmpty() && email.trim().isEmpty() &&
     phone.trim().isEmpty() && salaire == 0 && role == null && poste == null) {
95              EmployeeView.ModifierFail("Veuillez remplir au moins un champ.");
```

```
            return;
        }

        // Mise    jour des champs non vides
        if (!nom.trim().isEmpty()) employee.setNom(nom);
        if (!prenom.trim().isEmpty()) employee.setPrenom(prenom);
        if (!email.trim().isEmpty()) {
            // Validation de l'adresse e-mail
            if (!email.matches("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$")) {
                EmployeeView.ModifierFail("Veuillez entrer une adresse email valide.");
                return;
            }
            employee.setEmail(email);
        }
        if (salaire != 0) {
            // Validation du salaire
            if (salaire < 0) {
                EmployeeView.ModifierFail("Le salaire doit  tre  un nombre positif");
                return;
            }
            employee.setSalaire(salaire);
        }
        if (!phone.isEmpty()) {
            // Validation du num ro de t l phone
            if (!phone.matches("^0\\d{9}$")) {
                EmployeeView.ModifierFail("Le num ro de t l phone doit contenir 10
    chiffres");
                return;
            }
            employee.setPhone(phone);
        }
        if (role != null) employee.setRole(role);
        if (poste != null) employee.setPoste(poste);

        // Mise    jour de l'employ  via le DAO
        dao.modifier(employee, id);
    }
}
```

HolidayModel

```
package Model;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.List;

import DAO.HolidayDAOImpl;
import View.HolidayView;

// Classe HolidayModel : Gestion des cong s (logique m tier)
public class HolidayModel {
    private HolidayDAOImpl dao; // DAO pour l'acc s aux donn es des cong s

    // Constructeur pour initialiser le DAO
```

```
16      public HolidayModel(HolidayDAOImpl dao) {
17          this.dao = dao;
18      }
19
20      // M thode pour afficher la liste des employ s
21      public List<Employee> afficherEmployee() {
22          return dao.afficherEmployee();
23      }
24
25      // M thode pour afficher la liste des cong s
26      public List<Holiday> afficher() {
27          return dao.afficher();
28      }
29
30      // M thode pour ajouter un nouveau cong
31      public void ajouterHoliday(Holiday holiday, Employee employee) {
32          int days = calculateHolidayTime(holiday.getStart(), holiday.getEnd()); // Calcul
        de la dur e du cong
33          if (startCheck(holiday.getStart())) { // V rifie que la date de d but est
    valide
34              HolidayView.fail("La date de d but doit venir avant aujourd'hui.");
35              return;
36          }
37          if (days <= 0) { // V rifie que la date de fin est apr s la date de d but
38              HolidayView.fail("La date de fin doit venir apr s la date de d but.");
39              return;
40          }
41
42          // V rifie si l'employ  a suffisamment de jours de cong  disponibles
43          if (employee.getHolidayBalance() >= days) {
44              employee.setHolidayBalance(employee.getHolidayBalance() - days); // Mise
    jour du solde de cong s
45              dao.ajouter(holiday); // Ajout du cong  via le DAO
46              dao.modifierEmployeeBalance(employee, employee.getId()); // Mise    jour du
    solde dans la base de donn es
47          } else {
48              HolidayView.fail("Le nombre de jours de cong s disponibles est insuffisant
    .");
49          }
50      }
51
52      // M thode pour v rifier si une date de d but est avant la date actuelle
53      public boolean startCheck(String startDateString) {
54          LocalDate startDate = LocalDate.parse(startDateString, DateTimeFormatter.
    ofPattern("dd-MM-yyyy"));
55          return startDate.isBefore(LocalDate.now());
56      }
57
58      // M thode pour calculer la dur e d'un cong  en jours
59      public int calculateHolidayTime(String startDateString, String endDateString) {
60          LocalDate startDate = LocalDate.parse(startDateString, DateTimeFormatter.
    ofPattern("dd-MM-yyyy"));
61          LocalDate endDate = LocalDate.parse(endDateString, DateTimeFormatter.ofPattern("
    dd-MM-yyyy"));
62          return (int) ChronoUnit.DAYS.between(startDate, endDate);
```

```
63      }
64
65      // M thode pour trouver un employ  par son identifiant
66      public Employee FindById(int EmployeeId) {
67          return dao.findById(EmployeeId);
68      }
69
70      // M thode pour modifier un cong  existant
71      public void ModifierHoliday(Holiday updatedHoliday, Holiday oldHoliday) {
72          int newDays = calculateHolidayTime(updatedHoliday.getStart(), updatedHoliday.
    getEnd());
73          int oldDays = calculateHolidayTime(oldHoliday.getStart(), oldHoliday.getEnd());
74
75          if (startCheck(updatedHoliday.getStart())) { // V rifie la validit  de la date
     de d but
76              HolidayView.fail("La date de d but doit venir avant aujourd'hui.");
77              return;
78          }
79          if (newDays <= 0) { // V rifie que la dur e est valide
80              HolidayView.fail("La date de fin doit venir apr s la date de d but.");
81              return;
82          }
83
84          // Gestion du solde de cong s pour l'ancien et le nouvel employ
85          Employee newEmployee = FindById(updatedHoliday.getIdEmployee());
86          Employee oldEmployee = FindById(oldHoliday.getIdEmployee());
87
88          if (newEmployee.getHolidayBalance() >= newDays) {
89              oldEmployee.setHolidayBalance(oldEmployee.getHolidayBalance() + oldDays); //
     R cup ration des anciens jours
90              dao.modifierEmployeeBalance(oldEmployee, oldEmployee.getId());
91              newEmployee.setHolidayBalance(newEmployee.getHolidayBalance() - newDays); //
     D duction des nouveaux jours
92              dao.modifierEmployeeBalance(newEmployee, newEmployee.getId());
93              dao.modifier(updatedHoliday, updatedHoliday.getId()); // Mise    jour du
    cong
94          } else {
95              HolidayView.fail("Le nombre de jours de cong s disponibles est insuffisant
    pour le nouvel employ .");
96              return;
97          }
98      }
99
100     // M thode pour r cup rer le solde de cong s d'un employ
101     public void modifierEmployeeBalanceRecover(int days, int EmployeeId) {
102         Employee employee = this.FindById(EmployeeId);
103         employee.setHolidayBalance(employee.getHolidayBalance() + days); // Ajoute des
    jours au solde
104         dao.modifierEmployeeBalance(employee, EmployeeId);
105     }
106
107     // M thode pour trouver un cong  par son identifiant
108     public Holiday FindHolidayById(int holidayId) {
109         return dao.FindHolidayById(holidayId);
110     }
```

```
111
112     // M thode pour supprimer un cong
113     public void supprimerHoliday(Holiday oldHoliday) {
114         int holidayId = oldHoliday.getId();
115         int oldDays = calculateHolidayTime(oldHoliday.getStart(), oldHoliday.getEnd());
116         Employee oldEmployee = FindById(oldHoliday.getIdEmployee());
117         oldEmployee.setHolidayBalance(oldEmployee.getHolidayBalance() + oldDays); //
    R cup ration des jours de cong
118         dao.modifierEmployeeBalance(oldEmployee, oldEmployee.getId());
119         dao.supprimer(holidayId); // Suppression du cong  via le DAO
120     }
121 }
```

2.2 Enumeration

enums (ROLE)

```
 package Model;

public enum Role {
    ADMIN,
    MANAGER,
    EMPLOYEE
}
```

enums (Poste)

```
package Model;
public enum Poste {
    INGENIEUR_ETUDE_ET_DEVELOPPEMENT,
    TEAM_LEADER,
    PILOTE
}
```

enums (HolidayType)

```
package Model;

public enum HolidayType {
    CONGE_PAYE,
    CONGE_NON_PAYE,
    CONGE_MALADIE
}
```

2.3 DAO

Le DAO est une couche qui permet de gérer l'interaction avec une base de données, en effectuant des opérations telles que la création, la lecture, la mise à jour et la suppression (CRUD) des données.

DBConnection

```
package DAO;

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

/**
 * Classe pour g rer la connexion    la base de donn es.
 */
public class DBConnection {
    // URL de connexion    la base de donn es MySQL
    private static final String URL = "jdbc:mysql://localhost:3306/
   gestion_des_employess";

    // Nom d'utilisateur de la base de donn es
    private static final String USER = "root";

    // Mot de passe pour l'utilisateur de la base de donn es
    private static final String PASSWORD = "";

    // Objet Connection utilis  pour  tablir  la connexion
    private static Connection connection;

    /**
     * M thode pour obtenir une instance de connexion unique    la base de donn es.
     *
     * @return Connection : Objet Connection    la base de donn es.
     */
    public static Connection getConnection() {
        // V rifie si la connexion n'a pas encore   t    tablie
        if (connection == null) {
            try {
                // Charge le driver JDBC pour MySQL
                Class.forName("com.mysql.cj.jdbc.Driver");

                //  tablit  la connexion avec les param tres donn s
                connection = DriverManager.getConnection(URL, USER, PASSWORD);
            } catch (ClassNotFoundException | SQLException e) {
                // Affiche les erreurs  ventuelles  dans la console
                e.printStackTrace();

                // Lance une exception runtime si une erreur survient
                throw new RuntimeException("Error lors de la connexion");
            }
        }
        return connection; // Retourne la connexion existante ou nouvellement  c r   e
    }
}
```

Generique DAOI

```
    package DAO;

import java.util.List;
import Model.Employee;

/**
 * Interface g n rique pour les op rations CRUD.
 *
 * @param <T> Type des entit s.
 */
public interface GeneriqueDAOI<T> {

    // Afficher toutes les entit s.
    public List<T> afficher();

    // Ajouter une nouvelle entit .
    public void ajouter(T t);

    // Modifier une entit  existante par son ID.
    public void modifier(T t, int id);

    // Supprimer une entit  par son ID.
    public void supprimer(int id);

    // Rechercher un Employee par son ID.
    public Employee findById(int EmployeeId);
}
```

EmployerDAOImpl

```
package DAO;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.*;

import Controller.EmployeeController;
import Model.Employee;
import Model.Poste;
import Model.Role;
import View.EmployeeView;

/**
 * Impl mentation des interfaces EmployeeDAOI et GeneriqueDAOI.
 * Cette classe g re les op rations CRUD pour les employ s.
 */
public class EmployeeDAOImpl implements EmployeeDAOI, GeneriqueDAOI<Employee> {
    private Connection connection;

    // Initialisation de la connexion    la base de donn es.
    public EmployeeDAOImpl() {
```

```
24          connection = DBConnection.getConnection();
25      }
26
27      @Override
28      public List<Employee> afficher() {
29          // R cup re tous les employ s de la base de donn es.
30          String SQL = "SELECT * FROM employee";
31          EmployeeController.viderLesChamps();
32          List<Employee> employees = new ArrayList<>();
33          try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
34              try (ResultSet rset = stmt.executeQuery()) {
35                  while (rset.next()) {
36                      int id = rset.getInt("id");
37                      String nom = rset.getString("nom");
38                      String prenom = rset.getString("prenom");
39                      double salaire = rset.getDouble("salaire");
40                      String email = rset.getString("email");
41                      String phone = rset.getString("phone");
42                      String role = rset.getString("role");
43                      String poste = rset.getString("poste");
44                      int holidayBalance = rset.getInt("holidayBalance");
45                      employees.add(new Employee(id, nom, prenom, salaire, email, phone
    , Role.valueOf(role), Poste.valueOf(poste), holidayBalance));
46                  }
47              }
48          } catch (SQLException e) {
49              e.printStackTrace();
50          }
51          if (employees.isEmpty()) {
52              EmployeeView.AfficherFail("Aucun employ  a   t   trouv .");
53          }
54          return employees;
55      }
56
57      @Override
58      public void ajouter(Employee employee) {
59          // Ajoute un nouvel employ  dans la base de donn es.
60          String SQL = "INSERT INTO employee (nom, prenom, salaire, email, phone, role,
     poste, holidayBalance) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
61          EmployeeController.viderLesChamps();
62          try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
63              stmt.setString(1, employee.getNom());
64              stmt.setString(2, employee.getPrenom());
65              stmt.setDouble(3, employee.getSalaire());
66              stmt.setString(4, employee.getEmail());
67              stmt.setString(5, employee.getPhone());
68              stmt.setString(6, employee.getRole().name());
69              stmt.setString(7, employee.getPoste().name());
70              stmt.setInt(8, employee.getHolidayBalance());
71              stmt.executeUpdate();
72              EmployeeView.AjouterSuccess(employee);
73          } catch (SQLException e) {
74              e.printStackTrace();
75          }
76      }
```

```
77
78      @Override
79      public List<Employee> findByEmail(String email) {
80          // Recherche les employ s par email.
81          String SQL = "SELECT * FROM employee WHERE email = ?";
82          EmployeeController.viderLesChamps();
83          List<Employee> employees = new ArrayList<>();
84          try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
85              stmt.setString(1, email);
86              try (ResultSet rset = stmt.executeQuery()) {
87                  while (rset.next()) {
88                      employees.add(new Employee(rset.getInt("id"), rset.getString("nom
    "), rset.getString("prenom"),
89                              rset.getDouble("salaire"), rset.getString("email"), rset.
    getString("phone"),
90                              Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
    getString("poste")),
91                              rset.getInt("holidayBalance")));
92                  }
93              }
94          } catch (SQLException e) {
95              e.printStackTrace();
96          }
97          if (employees.isEmpty()) {
98              EmployeeView.AfficherFail("Aucun employ  a   t   trouv  avec cet
    adresse Email.");
99          }
100         return employees;
101     }
102
103     // Autres m thodes CRUD et de recherche impl ment es de mani re similaire,
104     // avec des commentaires succincts pour expliquer leur r le.
105
106     @Override
107     public Employee findById(int EmployeeId) {
108         // Recherche un employ  par son ID.
109         String SQL = "SELECT * FROM employee WHERE id = ?";
110         Employee employee = null;
111         EmployeeController.viderLesChamps();
112         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
113             stmt.setInt(1, EmployeeId);
114             try (ResultSet rset = stmt.executeQuery()) {
115                 if (rset.next()) {
116                     employee = new Employee(rset.getInt("id"), rset.getString("nom"),
     rset.getString("prenom"),
117                             rset.getDouble("salaire"), rset.getString("email"), rset.
    getString("phone"),
118                             Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
    getString("poste")),
119                             rset.getInt("holidayBalance"));
120                 }
121             } catch (SQLException e) {
122                 e.printStackTrace();
123             }
124         } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
        return employee;
    }

    @Override
    public void modifier(Employee employee, int EmployeeId) {
        // Met    jour les informations d'un employ  par son ID.
        String SQL = "UPDATE employee SET nom = ?, prenom = ?, salaire = ?, email =
   ?, phone = ?, role = ?, poste = ? WHERE id = ?";
        EmployeeController.viderLesChamps();
        try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
            stmt.setString(1, employee.getNom());
            stmt.setString(2, employee.getPrenom());
            stmt.setDouble(3, employee.getSalaire());
            stmt.setString(4, employee.getEmail());
            stmt.setString(5, employee.getPhone());
            stmt.setString(6, employee.getRole().name());
            stmt.setString(7, employee.getPoste().name());
            stmt.setInt(8, EmployeeId);
            stmt.executeUpdate();
            EmployeeView.ModifierSuccess();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void supprimer(int EmployeeId) {
        // Supprime un employ  par son ID.
        String SQL = "DELETE FROM employee WHERE id = ?";
        try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
            EmployeeController.viderLesChamps();
            stmt.setInt(1, EmployeeId);
            stmt.executeUpdate();
            EmployeeView.SupprimerSuccess();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

HolidayDAOImpl

```
package DAO;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

import Controller.EmployeeController;
import Model.Employee;
import Model.Holiday;
import Model.HolidayType;
```

```
import Model.Poste;
import View.EmployeeView;
import View.HolidayView;

public class HolidayDAOImpl implements GeneriqueDAOI<Holiday> {
    private Connection connection;

    // Constructeur : Initialise la connexion    la base de donn es
    public HolidayDAOImpl() {
        connection = DBConnection.getConnection();
    }

    // R cup re la liste de tous les employ s
    public List<Employee> afficherEmployee() {
        List<Employee> employees = new ArrayList<>();
        String query = "SELECT * FROM employee";

        try (PreparedStatement statement = connection.prepareStatement(query);
             ResultSet resultSet = statement.executeQuery()) {

            while (resultSet.next()) {
                // C r e un objet Employee    partir des donn es SQL
                int id = resultSet.getInt("id");
                String nom = resultSet.getString("nom");
                String prenom = resultSet.getString("prenom");
                double salaire = resultSet.getDouble("salaire");
                String email = resultSet.getString("email");
                String phone = resultSet.getString("phone");
                Model.Role role = Model.Role.valueOf(resultSet.getString("role"));
                Model.Poste poste = Poste.valueOf(resultSet.getString("poste"));
                int holidayBalance = resultSet.getInt("holidayBalance");
                Employee employee = new Employee(id, nom, prenom, salaire, email,
   phone, role, poste, holidayBalance);
                employees.add(employee);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return employees;
    }

    // R cup re la liste de tous les cong s
    public List<Holiday> afficher() {
        List<Holiday> holidays = new ArrayList<>();
        String query = "SELECT * FROM holiday";

        try (PreparedStatement statement = connection.prepareStatement(query);
             ResultSet resultSet = statement.executeQuery()) {

            while (resultSet.next()) {
                // C r e un objet Holiday    partir des donn es SQL
                int id = resultSet.getInt("id");
                int employeeId = resultSet.getInt("employee_id");
                HolidayType type = HolidayType.valueOf(resultSet.getString("type"));
```

```
65                String startDate = resultSet.getString("start");
66                String endDate = resultSet.getString("end");
67                Holiday holiday = new Holiday(id, employeeId, type, startDate,
    endDate);
68                holidays.add(holiday);
69            }
70        } catch (SQLException e) {
71            e.printStackTrace();
72        }
73
74        return holidays;
75    }
76
77    // Ajoute un nouveau cong  dans la base de donn es
78    @Override
79    public void ajouter(Holiday holiday) {
80        String SQL = "INSERT INTO holiday (employee_id, type, start, end) VALUES (?,
    ?, ?, ?)";
81        try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
82            stmt.setInt(1, holiday.getIdEmployee());
83            stmt.setString(2, holiday.getType().toString());
84            stmt.setString(3, holiday.getStart());
85            stmt.setString(4, holiday.getEnd());
86            stmt.executeUpdate();
87            HolidayView.success("Cong  ajout  avec succ s !");
88        } catch (SQLException e) {
89            e.printStackTrace();
90        }
91    }
92
93    // Modifie les informations d'un cong  existant
94    @Override
95    public void modifier(Holiday holiday, int holidayId) {
96        String SQL = "UPDATE holiday SET employee_id = ?, type = ?, start = ?, end =
    ? WHERE id = ?";
97        try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
98            stmt.setInt(1, holiday.getIdEmployee());
99            stmt.setString(2, holiday.getType().toString());
100           stmt.setString(3, holiday.getStart());
101           stmt.setString(4, holiday.getEnd());
102           stmt.setInt(5, holidayId);
103           stmt.executeUpdate();
104           HolidayView.success("Cong  modifi  avec succ s !");
105       } catch (SQLException e) {
106           e.printStackTrace();
107       }
108   }
109
110   // Met    jour le solde de cong s d'un employ
111   public void modifierEmployeeBalance(Employee employee, int EmployeeId) {
112       String SQL = "UPDATE employee SET holidayBalance = ? WHERE id = ?";
113       try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
114           stmt.setInt(1, employee.getHolidayBalance());
115           stmt.setInt(2, EmployeeId);
116           stmt.executeUpdate();
```

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

    // Supprime un cong  de la base de donn es
    @Override
    public void supprimer(int holidayId) {
        String SQL = "DELETE FROM holiday WHERE id = ?";
        try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
            stmt.setInt(1, holidayId);
            stmt.executeUpdate();
            HolidayView.success("Cong  supprim  avec succ s !");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Recherche un employ  par son ID
    @Override
    public Employee findById(int EmployeeId) {
        String SQL = "SELECT * FROM employee WHERE id = ?";
        Employee employee = null;
        EmployeeController.viderLesChamps();

        try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
            stmt.setInt(1, EmployeeId);
            try (ResultSet rset = stmt.executeQuery()) {
                if (rset.next()) {
                    // Cr e un objet Employee    partir des donn es SQL
                    int id = rset.getInt("id");
                    String nom = rset.getString("nom");
                    String prenom = rset.getString("prenom");
                    double salaire = rset.getDouble("salaire");
                    String email = rset.getString("email");
                    String phone = rset.getString("phone");
                    Model.Role role = Model.Role.valueOf(rset.getString("role"));
                    Model.Poste poste = Poste.valueOf(rset.getString("poste"));
                    int holidayBalance = rset.getInt("holidayBalance");
                    employee = new Employee(id, nom, prenom, salaire, email, phone,
role, poste, holidayBalance);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return employee;
    }

    // Recherche un cong  par son ID
    public Holiday FindHolidayById(int holidayId) {
        String SQL = "SELECT * FROM holiday WHERE id = ?";
        Holiday holiday = null;

        try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
```

```
171             stmt.setInt(1, holidayId);
172             try (ResultSet rset = stmt.executeQuery()) {
173                 if (rset.next()) {
174                     // Cr e un objet Holiday    partir des donn es SQL
175                     int id = rset.getInt("id");
176                     int idEmployee = rset.getInt("employee_id");
177                     Model.HolidayType type = Model.HolidayType.valueOf(rset.getString
    ("type"));
178                     String start = rset.getString("start");
179                     String end = rset.getString("end");
180                     holiday = new Holiday(id, idEmployee, type, start, end);
181                 }
182             }
183         } catch (SQLException e) {
184             e.printStackTrace();
185         }
186         return holiday;
187     }
188 }
```

2.4 Controller

Le contrôleur gère les actions de l'utilisateur. Il reçoit les événements de la vue, interagit avec le modèle pour effectuer des opérations (par exemple, ajout, modification, suppression de données), puis met à jour la vue en conséquence.

EmployeeController

```
public class EmployeeController {
    // Attribuer les mod les et les vues, et configurer les actions des boutons.
    public EmployeeController(EmployeeModel employeeModel, EmployeeView employeeView) {
        // Initialiser le mod le et la vue.
        this.employeeModel = employeeModel;
        EmployeeController.employeeView = employeeView;

        // Configurer les actions des boutons pour g rer les employ s.
        EmployeeController.employeeView.getAjouterButton().addActionListener(e -> this.ajouterEmployee());
        EmployeeController.employeeView.getAfficherButton().addActionListener(e -> this.handleAfficher());
        EmployeeController.employeeView.getSupprimerButton().addActionListener(e -> this.supprimerEmployee());
        EmployeeController.employeeView.getModifierButton().addActionListener(e -> this.updateEmployee());

        // Afficher les employ s lors de l'initialisation.
        this.afficherEmployee();
    }

    // Ajouter un nouvel employ .
    public void ajouterEmployee() {
        String nom  = employeeView.getNomField().getText();
        String prenom = employeeView.getPrenomField().getText();
        String salaire = employeeView.getSalaireField().getText();
        String email = employeeView.getEmailField().getText();
        String phone = employeeView.getPhoneField().getText();
        Role role = (Role) employeeView.getRoleComboBox().getSelectedItem();
        Poste poste = (Poste) employeeView.getPosteComboBox().getSelectedItem();

        employeeModel.ajouterEmployee(nom, prenom, salaire, email, phone, role , poste);
    }

    // Afficher tous les employ s.
    public void afficherEmployee() {
        List<Employee> employees = employeeModel.afficherEmployee();
        DefaultTableModel tableModel = (DefaultTableModel) employeeView.getTable().getModel();
        tableModel.setRowCount(0);
        for(Employee e : employees) {
            tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.getEmail(), e.getSalaire(), e.getPhone(), e.getRole(), e.getPoste(),e.getHolidayBalance()});
        }
```

```
    }

    // G rer les actions de recherche en fonction des champs remplis.
    private void handleAfficher() {
        if (allFieldsEmpty()) {
            this.afficherEmployee();
        } else {
            handleSearch();
        }
    }

    // V rifier si tous les champs sont vides.
    private boolean allFieldsEmpty() {
        return employeeView.getNomField().getText().isEmpty() &&
               employeeView.getPrenomField().getText().isEmpty() &&
               employeeView.getSalaireField().getText().isEmpty() &&
               employeeView.getEmailField().getText().isEmpty() &&
               employeeView.getPhoneField().getText().isEmpty();
    }

    // Effectuer une recherche en fonction des champs sp cifiques.
    private void handleSearch() {
        // Rechercher par nom complet.
        if (!employeeView.getNomField().getText().isEmpty() && !employeeView.
getPrenomField().getText().isEmpty()) {
            this.findByFullName(employeeView.getPrenomField().getText(), employeeView
.getNomField().getText());
        }
        // Rechercher par nom ou pr nom seul.
        if (!employeeView.getNomField().getText().isEmpty()) {
            this.findByLastName(employeeView.getNomField().getText());
        }
        if (!employeeView.getPrenomField().getText().isEmpty()) {
            this.findByFirstName(employeeView.getPrenomField().getText());
        }
        // Rechercher par d'autres crit res.
        if (!employeeView.getPhoneField().getText().isEmpty()) {
            this.findByPhone(employeeView.getPhoneField().getText());
        }
        if (!employeeView.getEmailField().getText().isEmpty()) {
            this.findByEmail(employeeView.getEmailField().getText());
        }
        if (!employeeView.getSalaireField().getText().isEmpty()) {
            this.findBySalaire(Double.parseDouble(employeeView.getSalaireField().
getText()));
        }
    }

    // Supprimer l'employ  s lectionn .
    public void supprimerEmployee() {
        int selectedRow = employeeView.getTable().getSelectedRow();
        if (selectedRow != -1) {
            int id = Integer.parseInt(employeeView.getTable().getModel().getValueAt(
selectedRow, 0).toString());
            employeeModel.supprimerEmployee(id);
```

```
        } else {
            EmployeeView.SupprimerFail("Veuillez choisir un employ .");
        }
        this.afficherEmployee();
    }

    // Mettre    jour les informations de l'employ  s lectionn .
    public void updateEmployee() {
        int selectedRow = employeeView.getTable().getSelectedRow();
        if (selectedRow != -1) {
            int id = Integer.parseInt(employeeView.getTable().getModel().getValueAt(
   selectedRow, 0).toString());
            String nom = employeeView.getNomField().getText();
            String prenom = employeeView.getPrenomField().getText();
            String email = employeeView.getEmailField().getText();
            double salaire = Utils.parseDouble(employeeView.getSalaireField().getText
   ());
            String phone = employeeView.getPhoneField().getText();
            Role role = (Role) employeeView.getRoleComboBox().getSelectedItem();
            Poste poste = (Poste) employeeView.getPosteComboBox().getSelectedItem();

            employeeModel.updateEmployee(employeeModel.findById(id), id, nom, prenom,
    email, salaire, phone, role, poste);
        } else {
            EmployeeView.ModifierFail("Veuillez choisir un employ .");
        }
    }

    // Vider les champs d'entr e.
    public static void viderLesChamps() {
        EmployeeView employeeView = EmployeeView.getInstance();
        employeeView.getNomField().setText("");
        employeeView.getPrenomField().setText("");
        employeeView.getSalaireField().setText("");
        employeeView.getEmailField().setText("");
        employeeView.getPhoneField().setText("");
        employeeView.getRoleComboBox().setSelectedIndex(-1);
        employeeView.getPosteComboBox().setSelectedIndex(-1);
    }
}
```

HolidayController

```
package Controller;

// Controller class to manage holiday-related actions and interactions between the model
    and the view.

import java.util.List;

import javax.swing.DefaultComboBoxModel;
import javax.swing.JComboBox;
import javax.swing.table.DefaultTableModel;

import Model.Employee;
import Model.Holiday;
```

```
13 import Model.HolidayModel;
14 import Model.HolidayType;
15 import View.HolidayView;
16
17 public class HolidayController {
18     private HolidayModel holidayModel;
19     private HolidayView holidayView;
20
21     // Constructor to initialize the controller with the holiday model and view.
22     public HolidayController(HolidayModel model, HolidayView view) {
23         this.holidayModel = model;
24         this.holidayView = view;
25         setEmployeesInComboBox(); // Populate the employee ComboBox with available
    employees.
26         holidayView.getAjouterButton().addActionListener(e -> this.ajouterHoliday()); //
     Add event listener for adding a holiday.
27         holidayView.getAfficherButton().addActionListener(e -> this.afficherHoliday());
    // Add event listener for displaying holidays.
28         holidayView.getModifierButton().addActionListener(e -> this.ModifierHoliday());
    // Add event listener for modifying a holiday.
29         holidayView.getSupprimerButton().addActionListener(e -> this.supprimerHoliday())
    ; // Add event listener for deleting a holiday.
30         this.afficherHoliday(); // Display holidays upon initialization.
31     }
32
33     // Adds a new holiday for an employee.
34     public void ajouterHoliday() {
35         JComboBox<String> nom = holidayView.getNomEmployeComboBox();
36         int Employeeid = Integer.parseInt(nom.getSelectedItem().toString().split(" - ")
    [0]);
37         HolidayType type = (HolidayType) holidayView.getTypeComboBox().getSelectedItem()
    ;
38         String dateDebut = holidayView.getDateDebut();
39         String dateFin = holidayView.getDateFin();
40         Holiday holiday = new Holiday(1, Employeeid, type, dateDebut, dateFin);
41         Employee employee = holidayModel.FindById(Employeeid);
42         holidayModel.ajouterHoliday(holiday, employee); // Save the holiday in the model
    .
43         this.afficherHoliday(); // Refresh the holiday table.
44     }
45
46     // Displays all holidays in the view's table.
47     public void afficherHoliday() {
48         DefaultTableModel model = (DefaultTableModel) holidayView.getHolidayTable().
    getModel();
49         Employee employee;
50         model.setRowCount(0);
51         List<Holiday> holidays = holidayModel.afficher(); // Retrieve holidays from the
    model.
52         for (Holiday holiday : holidays) {
53             employee = holidayModel.FindById(holiday.getIdEmployee());
54             model.addRow(new Object[]{holiday.getId(), employee.getNom() + " " +
    employee.getPrenom(), holiday.getType(), holiday.getStart(), holiday.getEnd()});
55         }
56     }
```

```
57
58      // Modifies an existing holiday.
59      public void ModifierHoliday() {
60          int selectedRow = holidayView.getTable().getSelectedRow();
61
62          if (selectedRow == -1) {
63              HolidayView.fail("Veuillez s lectionner une ligne.");
64              return;
65          }
66          int idHoliday = Integer.parseInt(holidayView.getTable().getModel().getValueAt(
    selectedRow, 0).toString());
67          Holiday oldHoliday = holidayModel.FindHolidayById(idHoliday);
68          Holiday updatedHoliday = new Holiday();
69          updatedHoliday.setId(idHoliday);
70          updatedHoliday.setIdEmployee(Integer.parseInt(holidayView.getNomEmployeComboBox
    ().getSelectedItem().toString().split(" - ")[0]));
71          updatedHoliday.setType((HolidayType) holidayView.getTypeComboBox().
    getSelectedItem());
72          updatedHoliday.setStart(holidayView.getDateDebut());
73          updatedHoliday.setEnd(holidayView.getDateFin());
74          holidayModel.ModifierHoliday(updatedHoliday, oldHoliday); // Update holiday in
    the model.
75          this.afficherHoliday(); // Refresh the table.
76      }
77
78      // Deletes a selected holiday.
79      public void supprimerHoliday() {
80          int selectedRow = holidayView.getTable().getSelectedRow();
81          if (selectedRow == -1) {
82              HolidayView.fail("Veuillez S lectionner une ligne.");
83              return;
84          } else {
85              int idHoliday = Integer.parseInt(holidayView.getTable().getModel().
    getValueAt(selectedRow, 0).toString());
86              Holiday oldHoliday = holidayModel.FindHolidayById(idHoliday);
87              holidayModel.supprimerHoliday(oldHoliday); // Remove the holiday from the
    model.
88          }
89          this.afficherHoliday(); // Refresh the holiday table.
90      }
91
92      // Populates the employee ComboBox with all available employees.
93      public void setEmployeesInComboBox() {
94          List<Employee> employees = holidayModel.afficherEmployee();
95          DefaultComboBoxModel<String> comboBoxModel = new DefaultComboBoxModel<>();
96
97          for (Employee e : employees) {
98              comboBoxModel.addElement(e.getId() + " - " + e.getNom() + " " + e.getPrenom
    ());
99          }
100
101         holidayView.getNomEmployeComboBox().setModel(comboBoxModel);
102     }
103 }
```

2.5 View

Dans le cadre du modèle MVC (Modèle-Vue-Contrôleur), la Vue (View) joue un rôle crucial en tant qu'interface utilisateur. Elle est responsable de la présentation des données à l'utilisateur et de la gestion de ses interactions avec l'application, sans contenir de logique métier.

EmployeeView

```
package View;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;

import Model.Employee;
import Model.Poste;
import Model.Role;
import java.awt.*;
public class EmployeeView extends JFrame {
    protected static final EmployeeView INSTANCE = new EmployeeView();
    protected JPanel General = new JPanel();
    protected JPanel GeneralUp = new JPanel();
    protected JPanel GeneralDown = new JPanel();
    protected JPanel ListContainer = new JPanel();
    protected JPanel ButtonsContainer = new JPanel();
    protected DefaultTableModel tableModel = new DefaultTableModel(new String[]{"Id","
    Nom", "Prenom", "Email", "Salaire", "Phone", "Role", "Poste", "Holiday Balance"}, 0)
    {
        @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
    };
    protected JTable Tableau = new JTable(tableModel);
    protected JButton Ajouter = new JButton("Ajouter");
    protected JButton Modifier = new JButton("Modifier");
    protected JButton Supprimer = new JButton("Supprimer");
    protected JButton Afficher = new JButton("Afficher");
    protected JLabel NomLabel;
    protected JTextField Nom;
    protected JLabel PrenomLabel;
    protected JTextField Prenom;
    protected JLabel EmailLabel;
    protected JTextField Email;
    protected JLabel TelephoneLabel;
    protected JTextField Telephone;
    protected JLabel SalaireLabel;
    protected JTextField Salaire;
    protected JLabel RoleLabel;
    protected JComboBox<Role> RoleComboBox;
    protected JLabel PosteLabel;
    protected JComboBox<Poste> PosteComboBox;

    public EmployeeView() {
        setTitle("Gestion des employe s");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(930, 520);
```

```
47          setLocationRelativeTo(null);
48          add(General);
49          General.setLayout(new BorderLayout());
50          General.add(GeneralUp, BorderLayout.NORTH);
51          General.add(GeneralDown, BorderLayout.CENTER);
52          GeneralUp.setLayout(new GridLayout(7,2));
53          GeneralUp.setBorder(BorderFactory.createEmptyBorder(10, 18, 10, 18));
54          NomLabel = new JLabel("Nom");
55          Nom = new JTextField();
56          GeneralUp.add(NomLabel);
57          GeneralUp.add(Nom);
58          PrenomLabel = new JLabel("Pre nom");
59          Prenom = new JTextField();
60          GeneralUp.add(PrenomLabel);
61          GeneralUp.add(Prenom);
62          EmailLabel = new JLabel("Email");
63          Email = new JTextField();
64          GeneralUp.add(EmailLabel);
65          GeneralUp.add(Email);
66          TelephoneLabel = new JLabel("Te le phone");
67          Telephone = new JTextField();
68          GeneralUp.add(TelephoneLabel);
69          GeneralUp.add(Telephone);
70          SalaireLabel = new JLabel("Salaire");
71          Salaire = new JTextField();
72          GeneralUp.add(SalaireLabel);
73          GeneralUp.add(Salaire);
74          RoleLabel = new JLabel("Role");
75          RoleComboBox = new JComboBox<>(Role.values());
76          GeneralUp.add(RoleLabel);
77          GeneralUp.add(RoleComboBox);
78          PosteLabel = new JLabel("Poste");
79          PosteComboBox = new JComboBox<>(Poste.values());
80          GeneralUp.add(PosteLabel);
81          GeneralUp.add(PosteComboBox);
82          GeneralDown.setLayout(new BorderLayout());
83          GeneralDown.add(ListContainer, BorderLayout.CENTER);
84          ListContainer.setLayout(new FlowLayout());
85          Dimension preferredSize = new Dimension(EmployeeView.this.getWidth() - 50,500);
86          Tableau.setPreferredScrollableViewportSize(preferredSize);
87          Tableau.setFillsViewportHeight(true);
88          ListContainer.add(new JScrollPane(Tableau));
89          GeneralDown.add(ButtonsContainer, BorderLayout.SOUTH);
90          ButtonsContainer.setLayout(new FlowLayout());
91          ButtonsContainer.add(Ajouter);
92          ButtonsContainer.add(Modifier);
93          ButtonsContainer.add(Supprimer);
94          ButtonsContainer.add(Afficher);
95          setVisible(true);
96      }
97      public static void AjouterSuccess(Employee employee){
98          JOptionPane.showMessageDialog(null, "L'employe  a  e  te   ajoute  avec
        succe s");
99      }
100     public static void AjouterFail(String message){
```

```
101         JOptionPane.showMessageDialog(null, "L'employe  n'a pas  e  t e   ajoute . " +
    message);
102     }
103     public static void AfficherFail(String message){
104         JOptionPane.showMessageDialog(null, message);
105     }
106     public static void SupprimerSuccess(){
107         JOptionPane.showMessageDialog(null, "L'employ  a bien  te  supprim .");
108     }
109     public static void SupprimerFail(String message){
110         JOptionPane.showMessageDialog(null, message);
111     }
112     public static void ModifierSuccess(){
113         JOptionPane.showMessageDialog(null, "L'employ  a bien   t   modifi .");
114     }
115     public static void ModifierFail(String message){
116         JOptionPane.showMessageDialog(null, message);
117     }
118     protected void CacherColumn(int index){
119         Tableau.getColumnModel().getColumn(index).setMinWidth(0);
120         Tableau.getColumnModel().getColumn(index).setMaxWidth(0);
121         Tableau.getColumnModel().getColumn(index).setWidth(0);
122     }
123     public static boolean SupprimerConfirmation(){
124         int choice = JOptionPane.showOptionDialog(null, " tes -vous  s  r  de supprimer
    cet  employe ?", "Confirmation", JOptionPane.YES_NO_OPTION, JOptionPane.
    QUESTION_MESSAGE, null, new String[]{"Oui", "Non"}, "Non");
125         return choice == JOptionPane.YES_OPTION;
126     }
127     public JTable getTable() {
128         return Tableau;
129     }
130     public JButton getAjouterButton() {
131         return Ajouter;
132     }
133
134     public JButton getModifierButton() {
135         return Modifier;
136     }
137
138     public JButton getSupprimerButton() {
139         return Supprimer;
140     }
141
142     public JButton getAfficherButton() {
143         return Afficher;
144     }
145
146     public JTextField getNomField() {
147         return Nom;
148     }
149
150     public void setNomField(JTextField nomField) {
151         Nom = nomField;
152     }
```

```
153
154     public JTextField getPrenomField() {
155         return Prenom;
156     }
157
158     public void setPrenomField(JTextField prenomField) {
159         Prenom = prenomField;
160     }
161
162     public JTextField getSalaireField() {
163         return Salaire;
164     }
165
166     public void setSalaireField(JTextField salaireField) {
167         Salaire = salaireField;
168     }
169
170     public JTextField getEmailField() {
171         return Email;
172     }
173
174     public void setEmailField(JTextField emailField) {
175         Email = emailField;
176     }
177
178     public JTextField getPhoneField() {
179         return Telephone;
180     }
181
182     public void setPhoneField(JTextField phoneField) {
183         Telephone = phoneField;
184     }
185
186     public JComboBox<Role> getRoleComboBox() {
187         return RoleComboBox;
188     }
189
190     public void setRoleComboBox(JComboBox<Role> roleComboBox) {
191         RoleComboBox = roleComboBox;
192     }
193
194     public JComboBox<Poste> getPosteComboBox() {
195         return PosteComboBox;
196     }
197
198     public void setPosteComboBox(JComboBox<Poste> posteComboBox) {
199         PosteComboBox = posteComboBox;
200     }
201     public static EmployeeView getInstance() {
202         return INSTANCE;
203     }
204 }
```

HolidayView

```
package View;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import Model.Employee;
import Model.HolidayType;
import Model.Role;

import java.awt.*;

public class HolidayView extends JFrame {
    private static final HolidayView INSTANCE = new HolidayView();
    private JPanel generalPanel = new JPanel();
    private JLabel nomEmployeLabel = new JLabel("Nom de l'employ ");
    private JComboBox<String> nomEmployeComboBox = new JComboBox<>();
    private JLabel typeLabel = new JLabel("Type");
    private JComboBox<HolidayType> typeComboBox = new JComboBox<>(HolidayType.values());
    private JLabel dateDebutLabel = new JLabel("Date de d but");
    private JTextField dateDebut = new JTextField("YYYY-MM-DD");
    private JLabel dateFinLabel = new JLabel("Date de fin");
    private JTextField dateFin = new JTextField("YYYY-MM-DD");
    private DefaultTableModel tableModel = new DefaultTableModel(new String[]{"Id","
    Employ ","Type","Date d but","Date fin"}, 0){
        @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
    };
    protected JTable holidayTable = new JTable(tableModel);
    private JScrollPane tableScrollPane = new JScrollPane(holidayTable);
    private JButton ajouterButton = new JButton("Ajouter");
    private JButton modifierButton = new JButton("Modifier");
    private JButton supprimerButton = new JButton("Supprimer");
    private JButton afficherButton = new JButton("Afficher");
    private JPanel inputPanel = new JPanel();
    private JPanel buttonPanel = new JPanel();

    public HolidayView() {
        setTitle("Gestion des holidays");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(930, 520);
        setLocationRelativeTo(null);
        setVisible(true);

        generalPanel.setLayout(new BorderLayout());
        inputPanel.setLayout(new GridLayout(5, 2, 10, 10));
        inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 0, 10));

        inputPanel.add(nomEmployeLabel);
        inputPanel.add(nomEmployeComboBox);
        inputPanel.add(typeLabel);
        inputPanel.add(typeComboBox);
        inputPanel.add(dateDebutLabel);
        inputPanel.add(dateDebut);
```

```
        inputPanel.add(dateFinLabel);
        inputPanel.add(dateFin);
        generalPanel.add(inputPanel, BorderLayout.NORTH);

        tableScrollPane.setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 10));
        generalPanel.add(tableScrollPane, BorderLayout.CENTER);

        buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 10));
        buttonPanel.add(ajouterButton);
        buttonPanel.add(modifierButton);
        buttonPanel.add(supprimerButton);
        buttonPanel.add(afficherButton);
        generalPanel.add(buttonPanel, BorderLayout.SOUTH);

        add(generalPanel);

        holidayTable.setFillsViewportHeight(true);
        holidayTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    }
    public JComboBox<String> getNomEmployeComboBox() {
        return nomEmployeComboBox;
    }
    public JComboBox<HolidayType> getTypeComboBox() {
        return typeComboBox;
    }
    public String getDateDebut() {
        return dateDebut.getText();
    }
    public String getDateFin() {
        return dateFin.getText();
    }
    public JButton getAfficherButton() {
        return afficherButton;
    }
    public JButton getAjouterButton() {
        return ajouterButton;
    }

    public JButton getModifierButton() {
        return modifierButton;
    }

    public JButton getSupprimerButton() {
        return supprimerButton;
    }
    public JTable getHolidayTable() {
        return holidayTable;
    }
    public static HolidayView getInstance() {
        return INSTANCE;
    }
    public static void success(String message) {
        JOptionPane.showMessageDialog(null, message, "Success", JOptionPane.
    INFORMATION_MESSAGE);
    }
```

```
    public static void fail(String message) {
        JOptionPane.showMessageDialog(null, message, "Error", JOptionPane.ERROR_MESSAGE);
    }
    public JTable getTable() {
        return holidayTable;
    }
}
```

PanelsView

```
package View;

import javax.swing.*;

public class PanelsView extends JFrame {
    private static PanelsView INSTANCE = new PanelsView();
    private JTabbedPane tabbedPane = new JTabbedPane();
    private EmployeeView employeeView = EmployeeView.getInstance();
    private HolidayView holidayView = HolidayView.getInstance();

    public PanelsView() {
        setTitle("Admin Dashboard - Gestion des Employ s et Cong s");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(930, 520);
        setLocationRelativeTo(null);
        employeeView.dispose();
        holidayView.dispose();
        tabbedPane.addTab("Gestion des Employ s", employeeView.getContentPane());
        tabbedPane.addTab("Gestion des Cong s", holidayView.getContentPane());
        add(tabbedPane);
        setVisible(true);
    }
    public static PanelsView getInstance() {
        return INSTANCE;
    }
}
```

Résultats

1 Tables Crées

	Table	Action						Lignes	Type	Interclassement	Taille	Perte
☐	employee	☆	Parcourir	Structure	Rechercher	Insérer	Vider	Supprimer	2	InnoDB	utf8mb4_general_ci	32,0 kio	-
☐	holiday	☆	Parcourir	Structure	Rechercher	Insérer	Vider	Supprimer	0	InnoDB	utf8mb4_general_ci	32,0 kio	-
	2 tables	Somme							2	InnoDB	utf8mb4_general_ci	64,0 kio	0 o

FIGURE 2.1 – Tables de la base de donnée

2 Résultats de la partie View

2.1 Pour Employee

La couche View représente l'interface utilisateur de l'application et permet l'interaction entre l'utilisateur et le système. Dans ce projet, l'interface a été conçue avec le framework Swing en Java, qui fournit des composants graphiques riches et personnalisables.

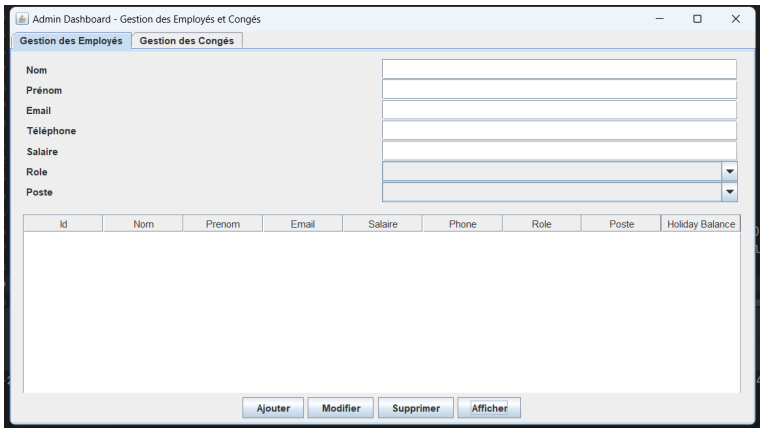

FIGURE 2.2 – Interface Utilisateur pour Employee

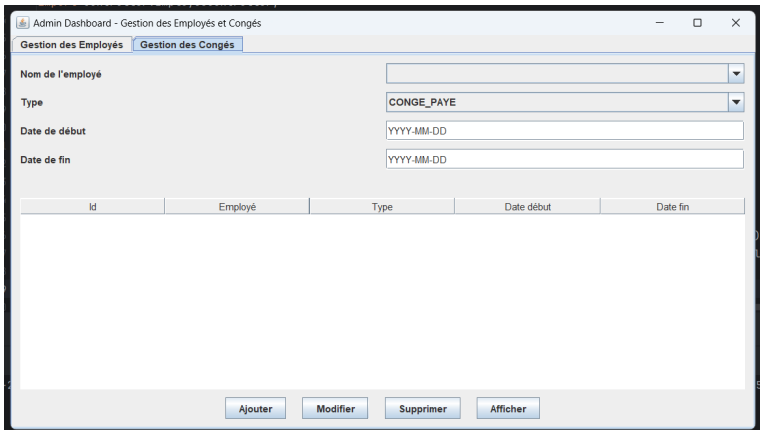

FIGURE 2.3 – Interface Utilisateur pour Holiday

2.2 Après Ajout

Après l'ajout d'un employé, les informations saisies par l'utilisateur dans le panneau d'entrée sont validées et transmises à la couche Controller, qui communique avec la logique métier pour enregistrer les données. Une fois l'opération réussie, la liste des employés est automatiquement mise à jour dans le panneau d'affichage, reflétant les changements en temps réel.

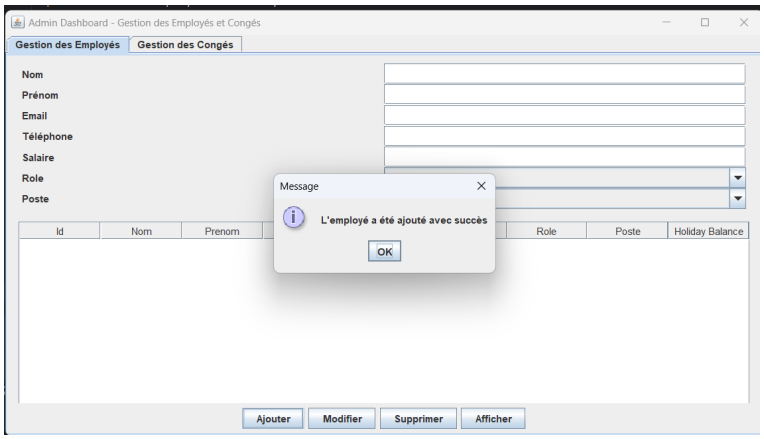

FIGURE 2.4 – Resultat Ajout

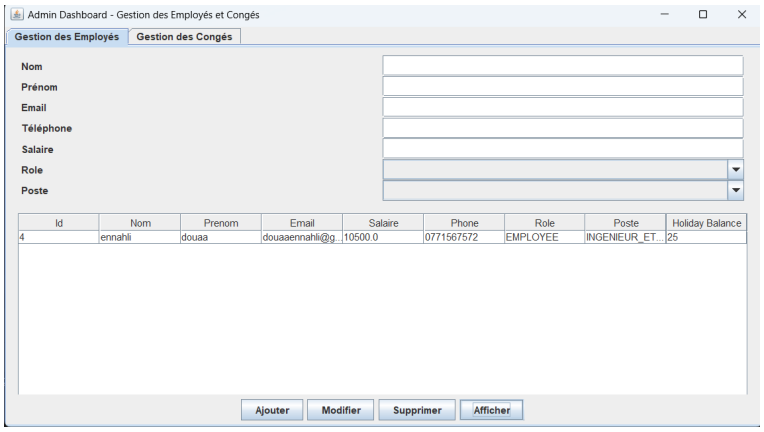

FIGURE 2.5 – Affichage de l'Ajout

2.3 Après modification

Après la mise à jour d'un employé, les nouvelles informations saisies par l'utilisateur dans le panneau d'entrée sont validées et transmises à la couche Controller, qui assure leur traitement via la logique métier.

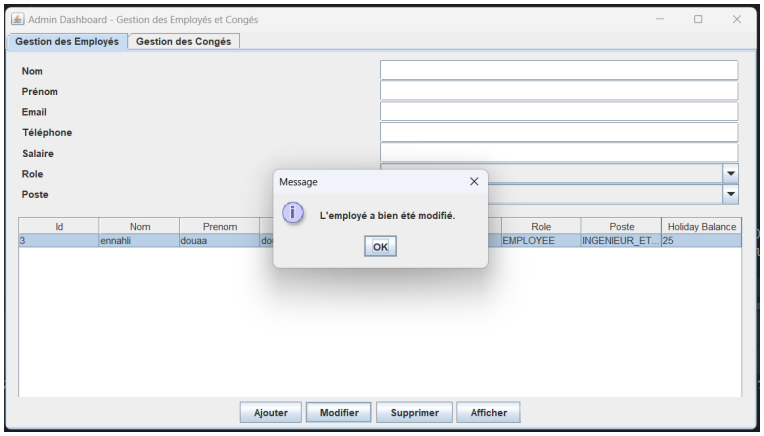

FIGURE 2.6 – Resultat de modification

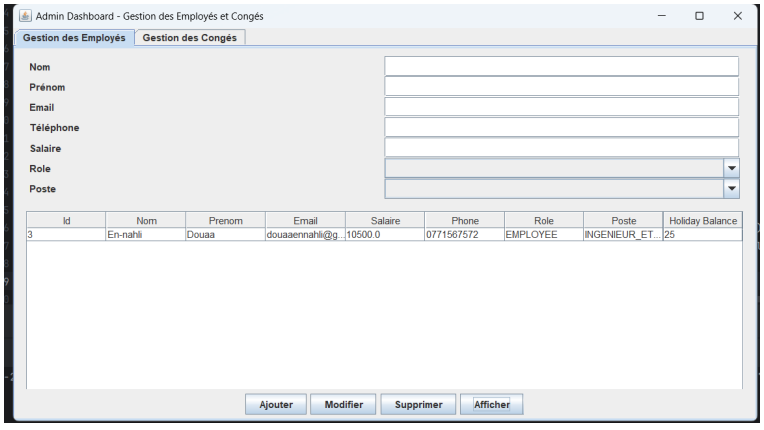

FIGURE 2.7 – Affichage de modification

2.4 Apres Suppression

Lorsqu'un employé est supprimé, l'utilisateur sélectionne l'employé concerné dans la liste affichée et confirme l'action en cliquant sur le bouton Supprimer. Cette demande est transmise à la couche Controller, qui s'assure de la suppression de l'enregistrement via la logique métier.

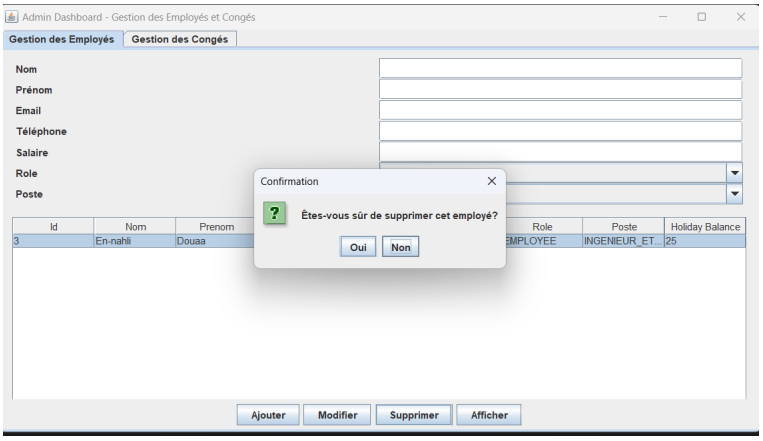

FIGURE 2.8 – Message pour s'assurer de la suppression

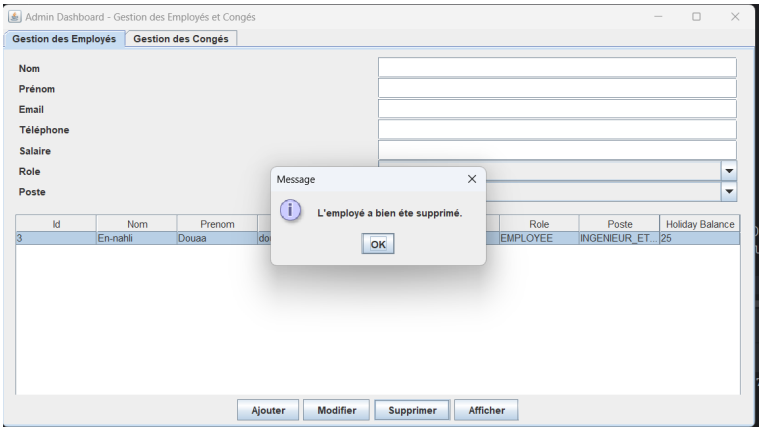

FIGURE 2.9 – Resultat de la suppression

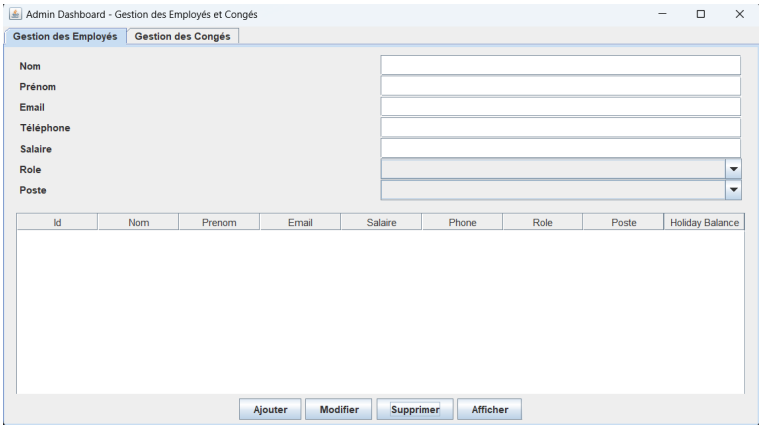

FIGURE 2.10 – Affichage de la suppression

2.5 Pour Holiday :

2.6 Après Ajout

Après l'ajout d'un employé, les informations saisies par l'utilisateur dans le panneau d'entrée sont validées et transmises à la couche Controller, qui communique avec la logique métier pour enregistrer les données. Une fois l'opération réussie, la liste des employés est automatiquement mise à jour dans le panneau d'affichage, reflétant les changements en temps réel.

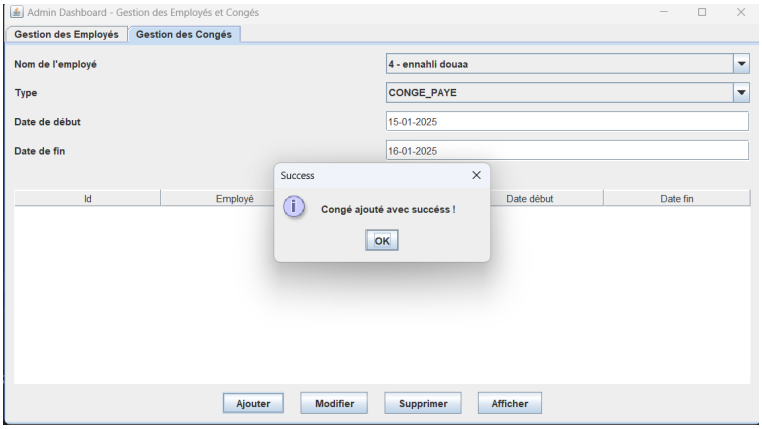

FIGURE 2.11 – Resultat Ajout

2.7 Après modification

Après la mise à jour d'un employé, les nouvelles informations saisies par l'utilisateur dans le panneau d'entrée sont validées et transmises à la couche Controller, qui assure leur traitement via la logique métier.

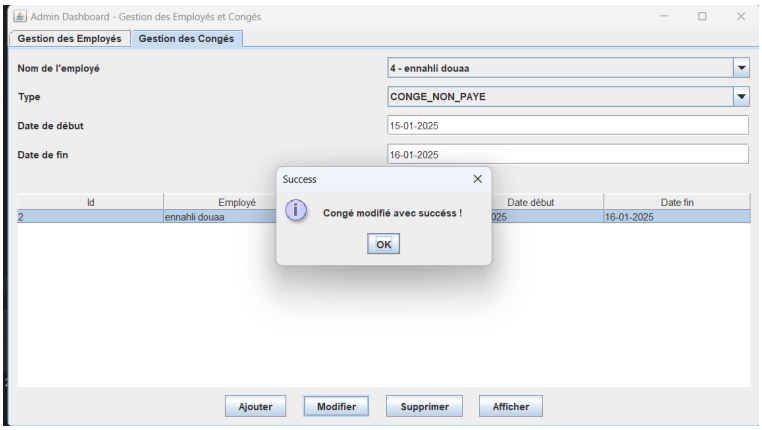

FIGURE 2.12 – Resultat de modification

2.8 Apres Suppression

Ajouter un congé : La méthode ajouterHoliday() récupère les informations saisies dans la vue (nom de l'employé, type de congé, dates de début et de fin). Elle crée ensuite un objet Holiday et l'ajoute via le modèle HolidayModel, puis met à jour l'affichage des congés.

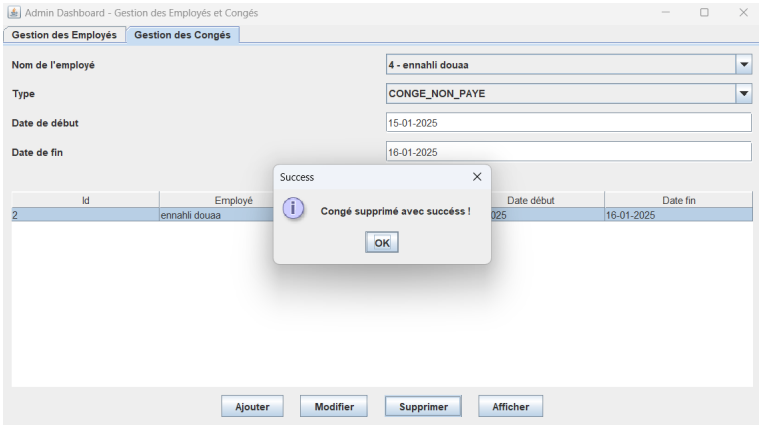

FIGURE 2.13 – Message pour s'assurer de la suppression

Gestion des fichiers E/S

1 interface" DataImportExport "

```
    package DAO;

import java.io.IOException;
import java.util.List;

public interface DataImportExport <T> {
    void importData(String fileName) throws IOException;
    void exportData(String fileName , List<T> data) throws IOException;
}
```

2 EmployeeDAOImpl

```
    @Override
    public void importData(String filePath) {
        String query ="INSERT INTO employee (nom, prenom, salaire, email, phone, role,
   poste) VALUES (?, ?, ?, ?, ?, ?, ?)" ;
        try(BufferedReader reader =new BufferedReader(new FileReader(filePath)) ;
   PreparedStatement pstmt = connection.prepareStatement(query)){
            String Line = reader.readLine();
            while((Line = reader.readLine()) != null){
                String[] data =Line.split(",");
                if (data.length ==7){
                    pstmt.setString(1, data[0].trim());
                    pstmt.setString(2, data[1].trim());
                    pstmt.setString(3, data[2].trim());
                    pstmt.setString(4, data[3].trim());
                    pstmt.setString(5, data[4].trim());
                    pstmt.setString(6, data[5].trim());
                    pstmt.setString(7, data[6].trim());
                    pstmt.addBatch();

                }
                System.out.println("data imported "+data.length);
            }
            pstmt.executeBatch();
```

```
            System.out.println("Employee imported successfully");


        }catch(IOException | SQLException e){
            e.printStackTrace();
        }

    }

    @Override
    public void exportData(String fileName, List<Employee> data) throws IOException {
     try(BufferedWriter writer =new BufferedWriter(new FileWriter(fileName))){
         writer.write("nom, prenom, salaire, email, phone, role, poste");
         writer.newLine();
         for(Employee employee : data){
             String Line = String.format("%s,%s,%.2f,%s,%s,%s,%s",
                     employee.getNom(),
                     employee.getPrenom(),
                     employee.getSalaire(),
                     employee.getEmail(),
                     employee.getPhone(),
                     employee.getRole(),
                     employee.getPoste());
             writer.write(Line);
             writer.newLine();
         }
     }

    }
```

3 EmployeeModel''ajout des E/s''

```
    private boolean checkFileExits(File file) {

        if(!file.exists()) {
            throw new IllegalArgumentException ("le fichier n'existe pas "+file.getPath
    ());

        }
        return true;

    }
    private boolean checkIsFile(File file) {

        if(!file.isFile()) {
            throw new IllegalArgumentException ("le chemin specifie nest pas un fichier
    "+file.getPath());

        }
        return true;

    }
    private boolean checkIsReadebal(File file) {
```

```
20
21          if(!file.canRead()) {
22              throw new IllegalArgumentException ("le chemin specifie nest pas lisibles "+
    file.getPath());
23
24          }
25          return true;
26
27      }
28      public void  importData(String FileName) throws IOException{
29          File file = new File(FileName);
30          checkFileExits(file);
31          checkIsFile(file);
32          checkIsReadebal(file);
33          dao.importData(FileName);
34      }
35
36      public void exportData(String FileName , List<Employee> data) throws IOException {
37          File file = new File(FileName);
38          dao.exportData(FileName, data);
39      }
```

4 EmployeeView

Ajout des buttons (Import et Export) et leurs labels . Remplacer le code de EmployeeView par ce code .

```
1     package View;
2
3  import javax.swing.*;
4  import javax.swing.table.DefaultTableModel;
5
6  import Model.Employee;
7  import Model.Poste;
8  import Model.Role;
9  import java.awt.*;
10 public class EmployeeView extends JFrame {
11     protected static final EmployeeView INSTANCE = new EmployeeView();
12     protected JPanel General = new JPanel();
13     protected JPanel GeneralUp = new JPanel();
14     protected JPanel GeneralDown = new JPanel();
15     protected JPanel ListContainer = new JPanel();
16     protected JPanel ButtonsContainer = new JPanel();
17     protected DefaultTableModel tableModel = new DefaultTableModel(new String[]{"Id","
    Nom", "Prenom", "Email", "Salaire", "Phone", "Role", "Poste", "Holiday Balance"}, 0)
    {
18         @Override
19             public boolean isCellEditable(int row, int column) {
20                 return false;
21             }
22     };
23     protected JTable Tableau = new JTable(tableModel);
24     protected JButton Ajouter = new JButton("Ajouter");
25     protected JButton Import = new JButton("Import");
26     protected JButton Export = new JButton("Export");
```

```
27      protected JButton Modifier = new JButton("Modifier");
28      protected JButton Supprimer = new JButton("Supprimer");
29      protected JButton Afficher = new JButton("Afficher");
30      protected JLabel NomLabel;
31      protected JTextField Nom;
32      protected JLabel PrenomLabel;
33      protected JTextField Prenom;
34      protected JLabel EmailLabel;
35      protected JTextField Email;
36      protected JLabel TelephoneLabel;
37      protected JTextField Telephone;
38      protected JLabel SalaireLabel;
39      protected JTextField Salaire;
40      protected JLabel RoleLabel;
41      protected JComboBox<Role> RoleComboBox;
42      protected JLabel PosteLabel;
43      protected JComboBox<Poste> PosteComboBox;
44
45      public EmployeeView() {
46          setTitle("Gestion des employe s");
47          setDefaultCloseOperation(EXIT_ON_CLOSE);
48          setSize(930, 520);
49          setLocationRelativeTo(null);
50          add(General);
51          General.setLayout(new BorderLayout());
52          General.add(GeneralUp, BorderLayout.NORTH);
53          General.add(GeneralDown, BorderLayout.CENTER);
54          GeneralUp.setLayout(new GridLayout(7,2));
55          GeneralUp.setBorder(BorderFactory.createEmptyBorder(10, 18, 10, 18));
56          NomLabel = new JLabel("Nom");
57          Nom = new JTextField();
58          GeneralUp.add(NomLabel);
59          GeneralUp.add(Nom);
60          PrenomLabel = new JLabel("P re nom");
61          Prenom = new JTextField();
62          GeneralUp.add(PrenomLabel);
63          GeneralUp.add(Prenom);
64          EmailLabel = new JLabel("Email");
65          Email = new JTextField();
66          GeneralUp.add(EmailLabel);
67          GeneralUp.add(Email);
68          TelephoneLabel = new JLabel("Te le phone");
69          Telephone = new JTextField();
70          GeneralUp.add(TelephoneLabel);
71          GeneralUp.add(Telephone);
72          SalaireLabel = new JLabel("Salaire");
73          Salaire = new JTextField();
74          GeneralUp.add(SalaireLabel);
75          GeneralUp.add(Salaire);
76          RoleLabel = new JLabel("Role");
77          RoleComboBox = new JComboBox<>(Role.values());
78          GeneralUp.add(RoleLabel);
79          GeneralUp.add(RoleComboBox);
80          PosteLabel = new JLabel("Poste");
81          PosteComboBox = new JComboBox<>(Poste.values());
```

```
82          GeneralUp.add(PosteLabel);
83          GeneralUp.add(PosteComboBox);
84          GeneralDown.setLayout(new BorderLayout());
85          GeneralDown.add(ListContainer, BorderLayout.CENTER);
86          ListContainer.setLayout(new FlowLayout());
87          Dimension preferredSize = new Dimension(EmployeeView.this.getWidth() - 50,500);
88          Tableau.setPreferredScrollableViewportSize(preferredSize);
89          Tableau.setFillsViewportHeight(true);
90          ListContainer.add(new JScrollPane(Tableau));
91          GeneralDown.add(ButtonsContainer, BorderLayout.SOUTH);
92          ButtonsContainer.setLayout(new FlowLayout());
93          ButtonsContainer.add(Ajouter);
94          ButtonsContainer.add(Modifier);
95          ButtonsContainer.add(Supprimer);
96          ButtonsContainer.add(Afficher);
97          ButtonsContainer.add(Import);
98          ButtonsContainer.add(Export);
99
100         setVisible(true);
101     }
102     public static void AjouterSuccess(Employee employee){
103         JOptionPane.showMessageDialog(null, "L'employe  a  e  te   ajoute  avec
        succe s");
104     }
105     public static void AjouterFail(String message){
106         JOptionPane.showMessageDialog(null, "L'employe  n'a pas  e  te   ajoute . " +
        message);
107     }
108     public static void AfficherFail(String message){
109         JOptionPane.showMessageDialog(null, message);
110     }
111     public static void SupprimerSuccess(){
112         JOptionPane.showMessageDialog(null, "L'employ  a bien  te  supprim .");
113     }
114     public static void SupprimerFail(String message){
115         JOptionPane.showMessageDialog(null, message);
116     }
117     public static void ModifierSuccess(){
118         JOptionPane.showMessageDialog(null, "L'employ  a bien   t   modifi .");
119     }
120     public static void ModifierFail(String message){
121         JOptionPane.showMessageDialog(null, message);
122     }
123     protected void CacherColumn(int index){
124         Tableau.getColumnModel().getColumn(index).setMinWidth(0);
125         Tableau.getColumnModel().getColumn(index).setMaxWidth(0);
126         Tableau.getColumnModel().getColumn(index).setWidth(0);
127     }
128     public static boolean SupprimerConfirmation(){
129         int choice = JOptionPane.showOptionDialog(null, " tes -vous  s  r  de supprimer
        cet  employe ?", "Confirmation", JOptionPane.YES_NO_OPTION, JOptionPane.
        QUESTION_MESSAGE, null, new String[]{"Oui", "Non"}, "Non");
130         return choice == JOptionPane.YES_OPTION;
131     }
132     public JTable getTable() {
```

```
133         return Tableau;
134     }
135     public JButton getAjouterButton() {
136         return Ajouter;
137     }
138 
139     public JButton getImport(){ return Import;}
140     public JButton getExport(){
141         return Export;
142     }
143 
144     public JButton getModifierButton() {
145         return Modifier;
146     }
147 
148     public JButton getSupprimerButton() {
149         return Supprimer;
150     }
151 
152     public JButton getAfficherButton() {
153         return Afficher;
154     }
155 
156     public JTextField getNomField() {
157         return Nom;
158     }
159 
160     public void setNomField(JTextField nomField) {
161         Nom = nomField;
162     }
163 
164     public JTextField getPrenomField() {
165         return Prenom;
166     }
167 
168     public void setPrenomField(JTextField prenomField) {
169         Prenom = prenomField;
170     }
171 
172     public JTextField getSalaireField() {
173         return Salaire;
174     }
175 
176     public void setSalaireField(JTextField salaireField) {
177         Salaire = salaireField;
178     }
179 
180     public JTextField getEmailField() {
181         return Email;
182     }
183 
184     public void setEmailField(JTextField emailField) {
185         Email = emailField;
186     }
187 
```

```
    public JTextField getPhoneField() {
        return Telephone;
    }

    public void setPhoneField(JTextField phoneField) {
        Telephone = phoneField;
    }

    public JComboBox<Role> getRoleComboBox() {
        return RoleComboBox;
    }

    public void setRoleComboBox(JComboBox<Role> roleComboBox) {
        RoleComboBox = roleComboBox;
    }

    public JComboBox<Poste> getPosteComboBox() {
        return PosteComboBox;
    }

    public void setPosteComboBox(JComboBox<Poste> posteComboBox) {
        PosteComboBox = posteComboBox;
    }
    public static EmployeeView getInstance() {
        return INSTANCE;
    }
}
```

5 EmployeeController

Ajouter au premier que les actionlistenner des buttons et creation des fonctions HandleImport et HandleExport precisamment au constructeur "EmployeeController ". ajouter ces codes suivants.

```
1     EmployeeController.employeeView.getImport().addActionListener(e -> this.
      handleImport());
2         EmployeeController.employeeView.getExport().addActionListener(e -> this.
      handleExport());
```

puis ces fonctionnalités.

```
1     public void handleImport() {
2         JFileChooser fileChooser = new JFileChooser();
3         fileChooser.setFileFilter(new FileNameExtensionFilter("Fichiers CSV", "csv", "
      txt"));
4
5         if (fileChooser.showOpenDialog(employeeView) == JFileChooser.APPROVE_OPTION) {
6             try {
7                 String filePath = fileChooser.getSelectedFile().getAbsolutePath();
8                 employeeModel.importData(filePath);
9                 employeeView.AfficherFail("Importation r ussie !");
10            } catch (IOException ex) {
11                employeeView.AfficherFail("Une erreur inattendue est survenue : " + ex.
      getMessage());
12            }
```

```
        }
    }

    public void handleExport()  {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileFilter(new FileNameExtensionFilter("Fichiers CSV", "csv"));

        if (fileChooser.showSaveDialog(employeeView) == JFileChooser.APPROVE_OPTION) {
            try {
                String filePath = fileChooser.getSelectedFile().getAbsolutePath();
                if (!filePath.toLowerCase().endsWith(".txt")) {
                    filePath += ".txt";
                }
                List<Employee> employes = new EmployeeDAOImpl().afficher();
                employeeModel.exportData(filePath, employes);
                employeeView.AfficherFail("Exportation r ussie !");
            } catch (Exception ex) {
                employeeView.AfficherFail("Une erreur inattendue est survenue : " + ex.
    getMessage());
            }
        }
    }
```

Résultats de la gestion des fichiers .

1 Pour Button "Import" :

Lorsqu'on clique sur le bouton "Import", on sélectionne un fichier préalablement créé contenant les informations des employés. Une fois le fichier sélectionné, son contenu est directement ajouté à la base de données ainsi qu'à la liste des employés.

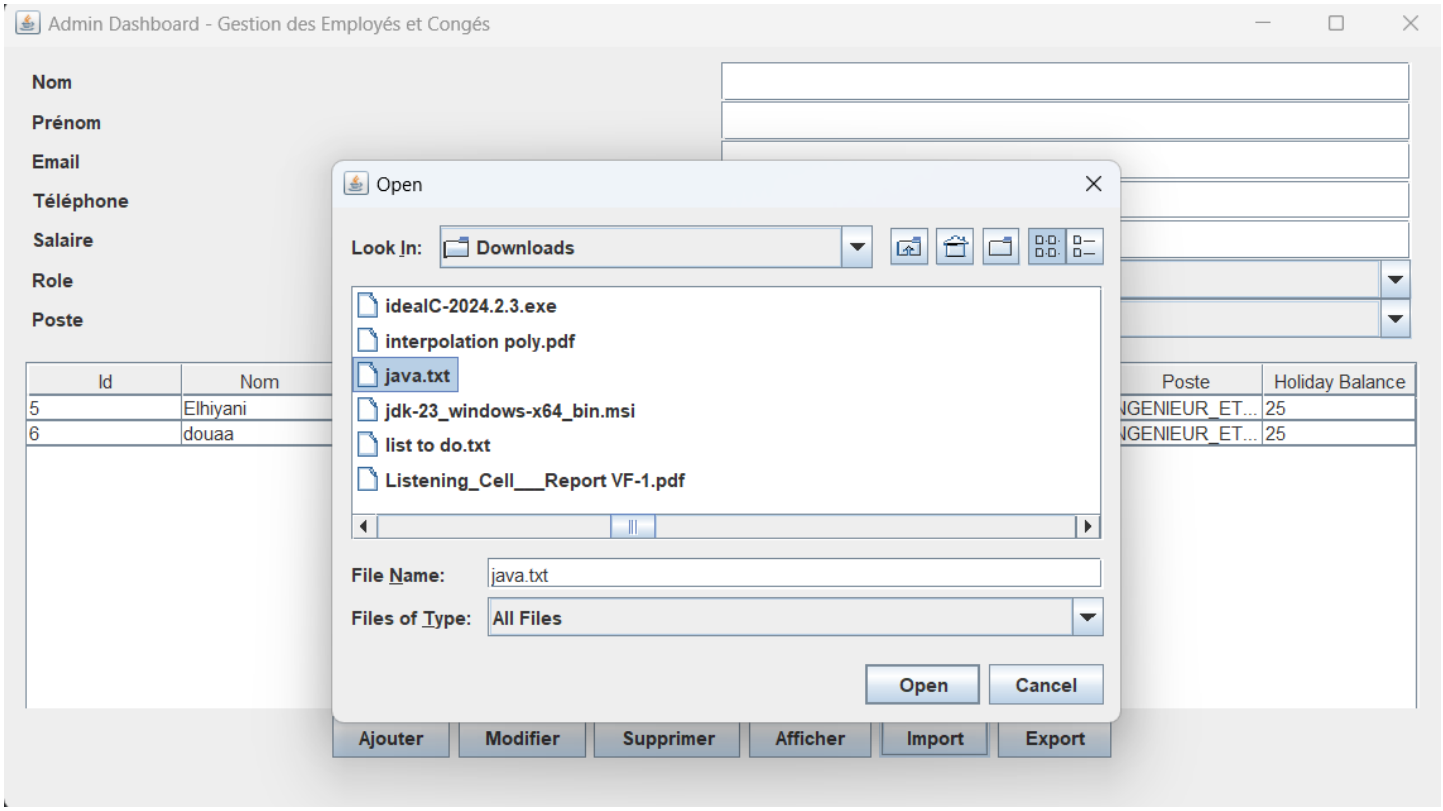

FIGURE 4.1 – Processu d'importation .

```
nom, prenom, salaire, email, phone, role, poste
En-nahli,yahya,50600.00,yahyaennahli@gmail.com,0670809050,ADMIN,TEAM_LEADER
Safaa,Safaa,15200.00,safaasafaa@gmail.com,0771567572,MANAGER,PILOTE
Bouarbine,amina,500600.00,aminabouarbine@gmail.com,0636037390,ADMIN,TEAM_LEADER
fatii,fatii,15200.00,fatiifatii@gmail.com,0606060606,EMPLOYEE,PILOTE
```

FIGURE 4.2 – Fichier Impoté.

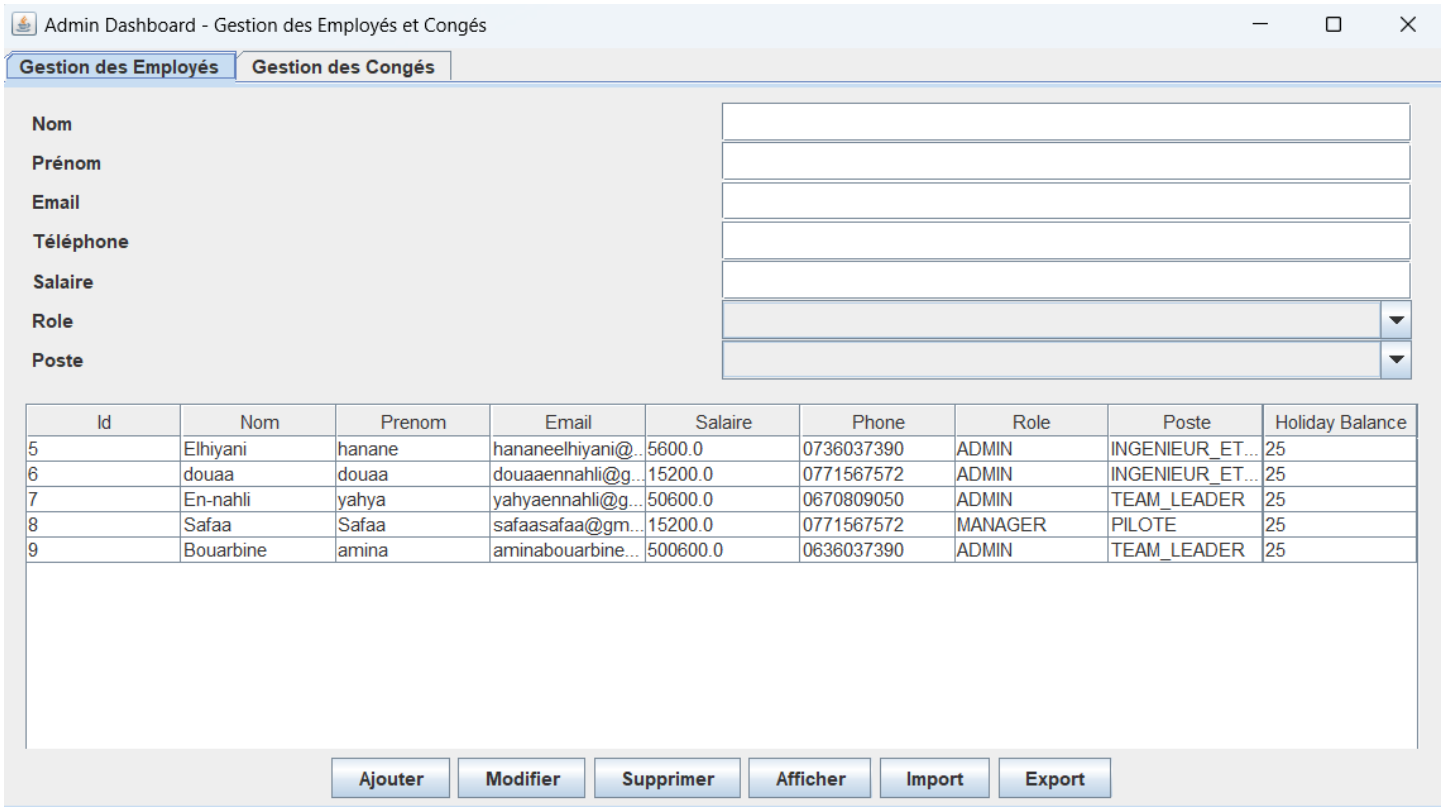

FIGURE 4.3 – Affichage du fichier importé.

2 Pour Button "Export" :

Lorsqu'on clique sur le bouton **"Exporter"**, un fichier est automatiquement créé avec un nom spécifié, et les informations contenues dans la base de données y sont enregistrées.

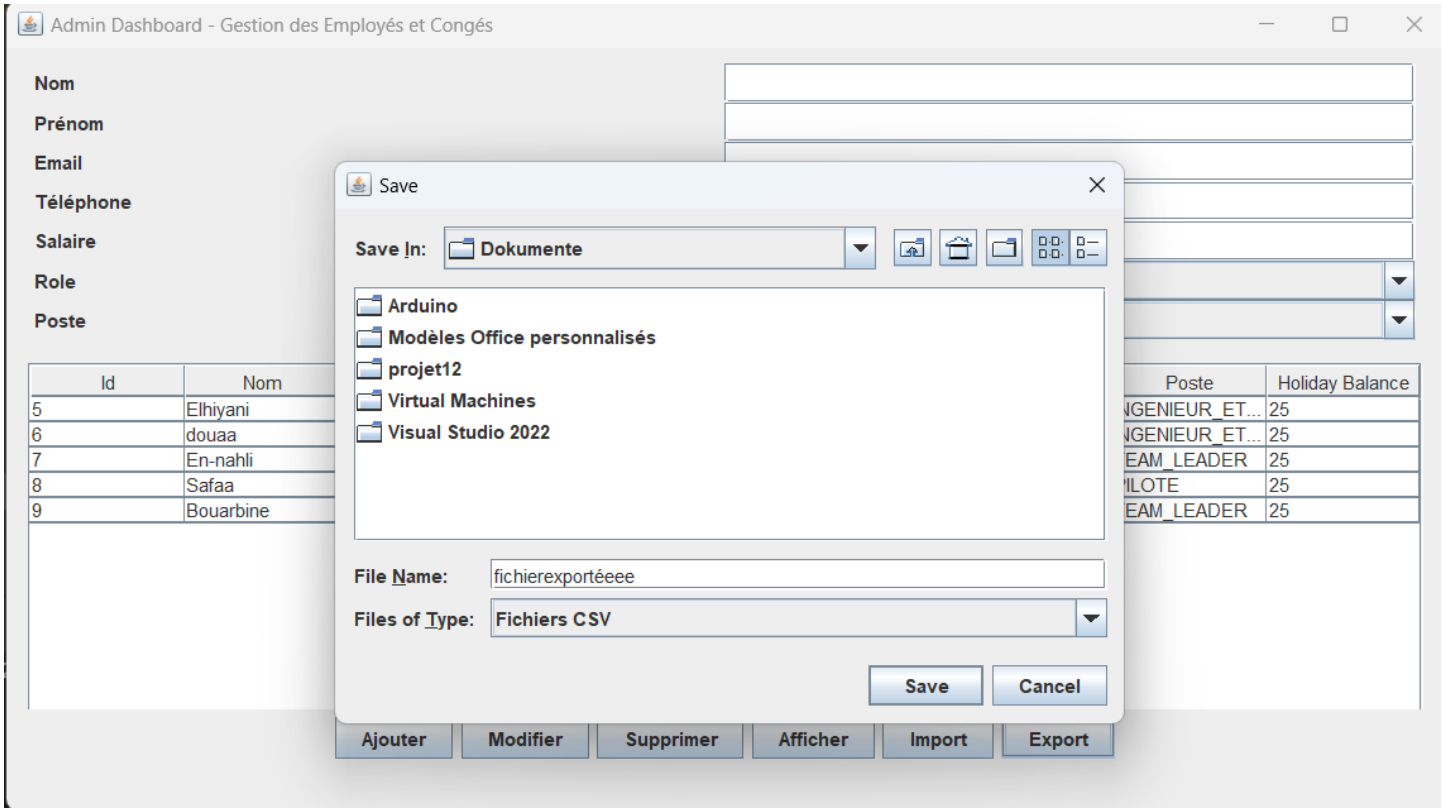

FIGURE 4.4 – Affichage du fichier importé.

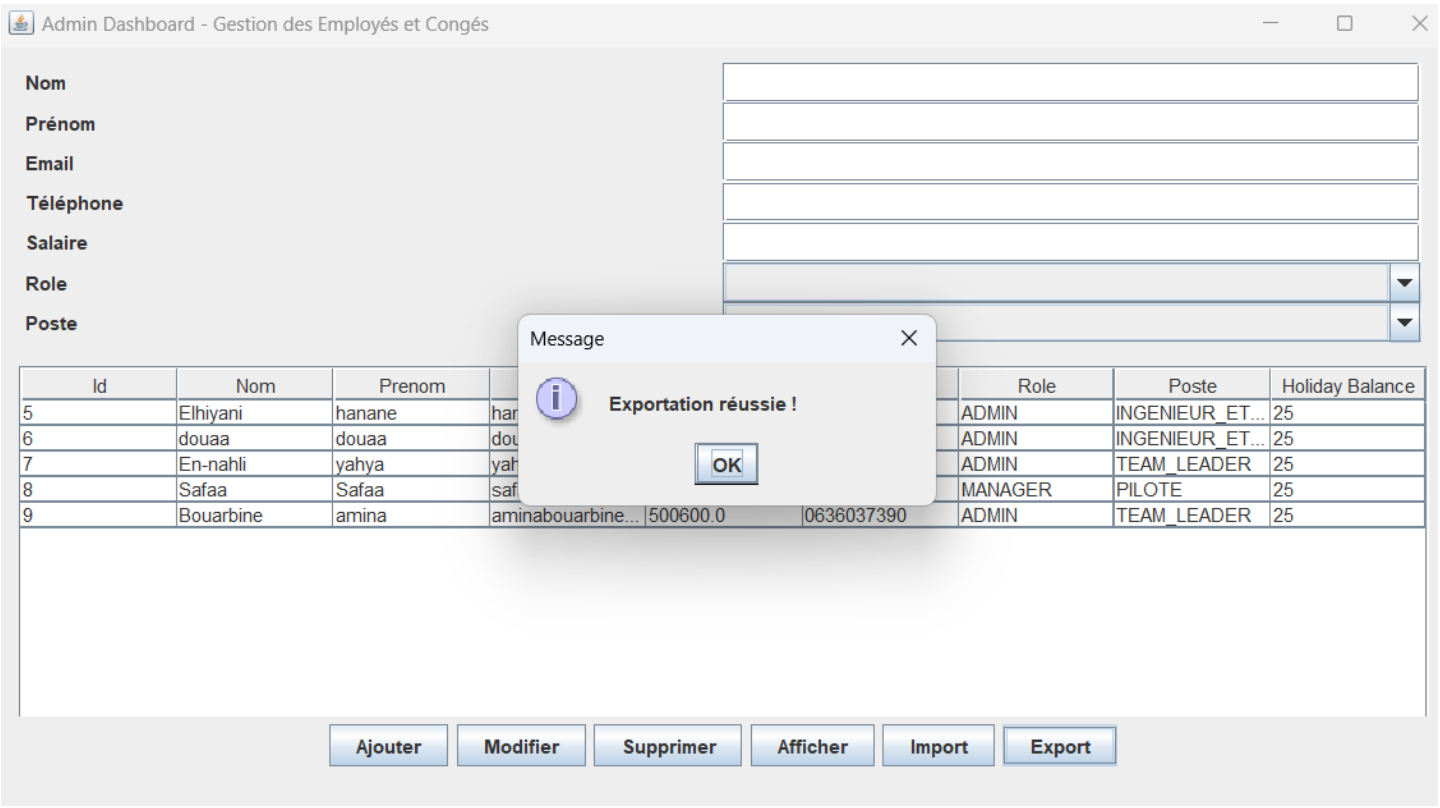

FIGURE 4.5 – réussite d'exportation.

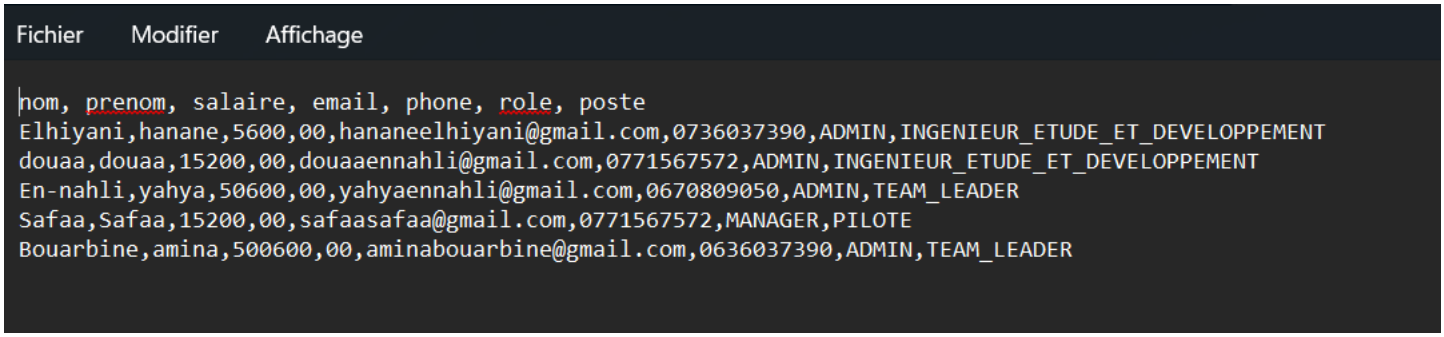

FIGURE 4.6 – Affichage du fichier exporté.

Conclusion générale

En conclusion, ce TP a permis de développer une application complète de gestion des employés et des congés en utilisant l'architecture **MVC**. Cette approche a permis une séparation claire des responsabilités entre la logique métier, l'interface utilisateur et la gestion des données, assurant ainsi une application modulable, facile à maintenir et à étendre. L'intégration des fonctionnalités d'ajout, de mise à jour et de suppression tant pour les employés que pour les congés a approfondi notre maîtrise des concepts de programmation orientée objet, de gestion des interfaces graphiques en Java, ainsi que de la gestion des différents types de données dans une application. De plus, l'implémentation des fichiers d'entrée/sortie (E/S) a permis d'ajouter des capacités d'importation et d'exportation de données au format CSV, renforçant l'utilité pratique et l'interopérabilité de l'application avec d'autres systèmes.

Ce travail pratique démontre l'importance de bien structurer le code pour construire des applications performantes et évolutives, tout en garantissant une expérience utilisateur fluide et efficace. Par ailleurs, il met en évidence le rôle crucial des bonnes pratiques de programmation et des mécanismes de gestion des données pour le développement de solutions logicielles robustes et professionnelles.

Références

java :

— https://www.java.com/en/download/
intellij idea :
— https://www.jetbrains.com/idea/download/?ref=freeStuffDevsection=windows
XAMPP :
— https://www.apachefriends.org/fr/index.html
jdk 23 :
— https://www.oracle.com/java/technologies/downloads/
Github :
— https://github.com/Ennahlidouaa/Gestion-des-employes—Gestion-des-cong-s.git
Github1 :
— https://github.com/Ennahlidouaa/gestion-employes-cong-s-MVC-DAO-E-S-.git