

**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 : LADDAD HAJAR**

**Encadré par : Mme. ELKOURCHI ASMAA**

**Année Universitaire : 2024/2025**

# Table des matières

<b>Inroduction</b>	<b>5</b>
<b>Outils &amp; environnement de travail</b>	<b>6</b>
1 Environnement de travail . . . . .	6
2 Outils de travail . . . . .	6
3 Language de Programmation . . . . .	7
<b>1 Réalisation</b>	<b>8</b>
1 Création de la base de donnée . . . . .	8
1.1 Script base de donnée . . . . .	8
2 Architecture MVC (Model-View-Controller) . . . . .	9
2.1 Model . . . . .	9
2.2 Enumeration.....	19
2.3 DAO.....	20
2.4 Controller.....	29
2.5 View.....	34
<b>2 Résultats</b>	<b>41</b>
1 Tables Créés.....	41
2 Résultats de la partie View.....	41
2.1 Pour Employee.....	41
2.2 Après Ajout .....	42
2.3 Après modification.....	43
2.4 Apres Suppression.....	44
2.5 Pour Holiday : .....	45
2.6 Après Ajout .....	45
2.7 Après modification.....	45
2.8 Apres Suppression.....	46
<b>3 Gestion des fichiers E/S</b>	<b>47</b>
1 interface" DataImportExport " .....	47
2 EmployeeDAOImpl.....	47
3 EmployeeModel"ajout des E/s" .....	48
4 EmployeeView .....	49
5 EmployeeController.....	53

<b>4</b>	<b>Résultats de la gestion des fichiers .</b>	<b>54</b>
1	Pour Button "Import" :.....	54
2	Pour Button "Export" :.....	56
<b>5</b>	<b>Conclusion générale</b>	<b>59</b>
<b>6</b>	<b>Références</b>	<b>60</b>

# Table des figures

1	intellij idea logo . . . . .	6
2	MySQL Workbench logo . . . . .	6
3	xampp logo . . . . .	6
4	java developpement kit logo . . . . .	7
5	java logo . . . . .	7
2.1	Tables de la base de donnée .....	41
2.2	Interface Utilisateur pour Employee.....	41
2.3	Interface Utilisateur pour Holiday.....	42
2.4	Resultat Ajout .....	42
2.5	Affichage de l' Ajout.....	43
2.6	Resultat de modification .....	43
2.7	Affichage de modification .....	43
2.8	Message pour s'assurer de la suppression.....	44
2.9	Resultat de la suppression .....	44
2.10	Affichage de la suppression.....	45
2.11	Resultat Ajout .....	45
2.12	Resultat de modification .....	46
2.13	Message pour s'assurer de la suppression.....	46
4.1	Processu d'importation .....	55
4.2	Fichier Impoté. ....	55
4.3	Affichage du fichier importé.....	56
4.4	Affichage du fichier importé.....	57
4.5	réussite d'exportation.....	58
4.6	Affichage du fichier exporté. ....	58

# 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

```
1  -- Cr ation de la base de donn es
2  create database gestion_des_employess;
3
4  -- Utilisation de la base de donnees
5  use gestion_des_employess;
6
7  -- Table Employee
8  CREATE TABLE employee (
9      id INT AUTO_INCREMENT PRIMARY KEY,
10     nom VARCHAR(50) NOT NULL,
11     prenom VARCHAR(50) NOT NULL,
12     salaire DECIMAL(10, 2) NOT NULL,
13     email VARCHAR(100) NOT NULL UNIQUE,
14     phone VARCHAR(15) NOT NULL,
15     role VARCHAR(200) not null,
16     poste VARCHAR(200) NOT NULL,
17     holidayBalance INTEGER DEFAULT 25
18 );
19 -- Table Holiday
20 create table holiday (
21     id int auto_increment primary key,
22     employee_id int not null,
23     type varchar(200),
24     start varchar(10) not null,
25     end varchar(10) not null,
26     CONSTRAINT fk_employee FOREIGN KEY (employee_id) REFERENCES employee(id) ON delete
27     cascade
28 );
```

Listing 1.1 – Script SQL de la base de données

- Ce script est écrit sur MySQL Workbench pour création la base de donnée pour être lié 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

```
1 package Model;
2
3 // Classe représentant un employé avec ses attributs et méthodes
4 public class Employee {
5     // Attributs de la classe Employee
6     private int id; // Identifiant unique de l'employé
7     private String nom; // Nom de l'employé
8     private String prenom; // Prénom de l'employé
9     private double salaire; // Salaire de l'employé
10    private String email; // Adresse e-mail de l'employé
11    private String phone; // Numéro de téléphone de l'employé
12    private Role role; // Rôle de l'employé (probablement une classe ou
        numération distincte)
13    private Poste poste; // Poste de l'employé (probablement une classe ou
        numération distincte)
14    private int holidayBalance; // Solde des congés de l'employé
15
16    // Constructeur pour initialiser les attributs de la classe
17    public Employee(int id, String nom, String prenom, double salaire, String
        email, String phone, Role role, Poste poste, int holidayBalance) {
18        this.id = id;
19        this.nom = nom;
20        this.prenom = prenom;
21        this.salaire = salaire;
22        this.email = email;
23        this.phone = phone;
24        this.role = role;
25        this.poste = poste;
26        this.holidayBalance = holidayBalance;
27    }
28
29    // Méthodes getter et setter pour accéder et modifier les attributs de l'
        employé
30
31    // Getter pour l'identifiant de l'employé
32    public int getId() {
33        return id;
34    }
35
36    // Setter pour l'identifiant de l'employé
37    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 representant 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
9     numration distincte)
10    private String start; // Date de d but du cong (au format String, ex: "YYYY-MM-DD
11    ")
12    private String end; // Date de fin du cong (au format String, ex: "YYYY-MM-DD")
13
14    // Constructeur principal pour initialiser les attributs de la classe
15    public Holiday(int id, int idEmployee, HolidayType type, String start, String end) {
16        this.id = id;
17        this.idEmployee = idEmployee;
18        this.type = type;
19        this.start = start;
20        this.end = end;
21    }
22
23    // Constructeur par d faut pour initialiser les attributs avec des valeurs nulles
24    ou par d faut
25    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
```

```
78     public void setEnd(String end) {
79         this.end = end;
80     }
81 }
```

### EmployerLogic

```
1 package Model;
2
3 import java.util.List;
4 import DAO.EmployeeDAOImpl;
5 import Utilities.Utills;
6 import View.EmployeeView;
7
8 // Classe EmployeeModel : Contient la logique m tier pour g rer les employ s
9 public class EmployeeModel {
10     private EmployeeDAOImpl dao; // DAO (Data Access Object) pour interagir avec la base
        de donn es
11
12     // Constructeur pour initialiser le DAO
13     public EmployeeModel(EmployeeDAOImpl dao) {
14         this.dao = dao;
15     }
16
17     // M thode pour ajouter un nouvel employ
18     public void ajouterEmployee(String nom, String prenom, String salaire, String email,
        String phone, Role role, Poste poste) {
19         double salaireDouble = Utills.parseDouble(salaire); // Conversion du salaire en
        double
20         // Validation des champs obligatoires
21         if (nom.trim().isEmpty() || prenom.trim().isEmpty() || email.trim().isEmpty() ||
        phone.trim().isEmpty() || salaireDouble == 0) {
22             EmployeeView.AjouterFail("Veuillez remplir tous les champs.");
23             return;
24         }
25
26         // Validation de l'adresse e-mail
27         if (!email.matches("^([a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,})$")) {
28             EmployeeView.AjouterFail("Veuillez entrer une adresse email valide.");
29             return;
30         }
31
32         // Validation du num ro de t l phone
33         if (!phone.matches("^0\\d{9}$")) {
34             EmployeeView.AjouterFail("Le num ro de t l phone doit contenir 10
        chiffres");
35             return;
36         }
37
38         // Validation du salaire
39         if (salaireDouble < 0) {
40             EmployeeView.AjouterFail("Le salaire doit tre un nombre positif");
41             return;
42         }
43
44         // 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,
46     role, poste, 25);
47     dao.ajouter(employee); // Ajout de l'employ via le DAO
48 }
49 // Methode pour recuperer la liste des employes
50 public List<Employee> afficherEmployee() {
51     return dao.afficher();
52 }
53
54 // Methodes de recherche d'employes par differents criteres
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 // Methode pour supprimer un employe
80 public void supprimerEmployee(int id) {
81     if (EmployeeView.SupprimerConfirmation()) { // Demande de confirmation avant
82     suppression
83         dao.supprimer(id);
84     }
85 }
86
87 // Methode pour trouver un employe par son identifiant
88 public Employee findById(int id) {
89     return dao.findById(id);
90 }
91
92 // Methode pour mettre a jour un employe
93 public void updateEmployee(Employee employee, int id, String nom, String prenom,
94 String email, double salaire, String phone, Role role, Poste poste) {
95     // Validation : Au moins un champ doit etre renseigné
96     if (nom.trim().isEmpty() && prenom.trim().isEmpty() && email.trim().isEmpty() &&
97     phone.trim().isEmpty() && salaire == 0 && role == null && poste == null) {
98         EmployeeView.ModifierFail("Veuillez remplir au moins un champ.");
99     }
```

```
96         return;
97     }
98
99     // Mise    jour des champs non vides
100    if (!nom.trim().isEmpty()) employee.setNom(nom);
101    if (!prenom.trim().isEmpty()) employee.setPrenom(prenom);
102    if (!email.trim().isEmpty()) {
103        // Validation de l'adresse e-mail
104        if (!email.matches("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$")) {
105            EmployeeView.ModifierFail("Veuillez entrer une adresse email valide.");
106            return;
107        }
108        employee.setEmail(email);
109    }
110    if (salaire != 0) {
111        // Validation du salaire
112        if (salaire < 0) {
113            EmployeeView.ModifierFail("Le salaire doit tre un nombre positif");
114            return;
115        }
116        employee.setSalaire(salaire);
117    }
118    if (!phone.isEmpty()) {
119        // Validation du num ro de t l phone
120        if (!phone.matches("^0\\d{9}$")) {
121            EmployeeView.ModifierFail("Le num ro de t l phone doit contenir 10
chiffres");
122            return;
123        }
124        employee.setPhone(phone);
125    }
126    if (role != null) employee.setRole(role);
127    if (poste != null) employee.setPoste(poste);
128
129    // Mise    jour de l'employ via le DAO
130    dao.modifier(employee, id);
131 }
132 }
```

### HolidayModel

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
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
33     de la dur e du cong
34     if (startCheck(holiday.getStart())) { // V rifie que la date de d but est
35         valide
36         HolidayView.fail("La date de d but doit venir avant aujourd'hui.");
37         return;
38     }
39     if (days <= 0) { // V rifie que la date de fin est apr s la date de d but
40         HolidayView.fail("La date de fin doit venir apr s la date de d but.");
41         return;
42     }
43     // V rifie si l'employ a suffisamment de jours de cong disponibles
44     if (employee.getHolidayBalance() >= days) {
45         employee.setHolidayBalance(employee.getHolidayBalance() - days); // Mise
46         jour du solde de cong s
47         dao.ajouter(holiday); // Ajout du cong via le DAO
48         dao.modifierEmployeeBalance(employee, employee.getId()); // Mise jour du
49         solde dans la base de donn es
50     } else {
51         HolidayView.fail("Le nombre de jours de cong s disponibles est insuffisant
52         .");
53     }
54 }
55
56 // M thode pour v rifier si une date de d but est avant la date actuelle
57 public boolean startCheck(String startDateString) {
58     LocalDate startDate = LocalDate.parse(startDateString, DateTimeFormatter.
59     ofPattern("dd-MM-yyyy"));
60     return startDate.isBefore(LocalDate.now());
61 }
62
63 // M thode pour calculer la dur e d'un cong en jours
64 public int calculateHolidayTime(String startDateString, String endDateString) {
65     LocalDate startDate = LocalDate.parse(startDateString, DateTimeFormatter.
66     ofPattern("dd-MM-yyyy"));
67     LocalDate endDate = LocalDate.parse(endDateString, DateTimeFormatter.ofPattern("
68     dd-MM-yyyy"));
69     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 uction 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)

```
1 package Model;
2
3 public enum Role {
4     ADMIN,
5     MANAGER,
6     EMPLOYEE
7 }
```

### enums (Poste)

```
1 package Model;
2 public enum Poste {
3     INGENIEUR_ETUDE_ET_DEVELOPPEMENT,
4     TEAM_LEADER,
5     PILOTE
6 }
```

### enums (HolidayType)

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

## 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

```
1 package DAO;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 /**
8  * Classe pour g r e r la connexion      la base de donn es.
9  */
10 public class DBConnection {
11     // URL de connexion      la base de donn es MySQL
12     private static final String URL = "jdbc:mysql://localhost:3306/
gestion_des_employess";
13
14     // Nom d'utilisateur de la base de donn es
15     private static final String USER = "root";
16
17     // Mot de passe pour l'utilisateur de la base de donn es
18     private static final String PASSWORD = "";
19
20     // Objet Connection utilis pour tablir la connexion
21     private static Connection connection;
22
23     /**
24      * M thode pour obtenir une instance de connexion unique      la base de donn es.
25      *
26      * @return Connection : Objet Connection      la base de donn es.
27      */
28     public static Connection getConnection() {
29         // V rifie si la connexion n'a pas encore t tablie
30         if (connection == null) {
31             try {
32                 // Charge le driver JDBC pour MySQL
33                 Class.forName("com.mysql.cj.jdbc.Driver");
34
35                 // tablit la connexion avec les param tres donn s
36                 connection = DriverManager.getConnection(URL, USER, PASSWORD);
37             } catch (ClassNotFoundException | SQLException e) {
38                 // Affiche les erreurs ventuelles dans la console
39                 e.printStackTrace();
40
41                 // Lance une exception runtime si une erreur survient
42                 throw new RuntimeException("Error lors de la connexion");
43             }
44         }
45         return connection; // Retourne la connexion existante ou nouvellement cr e
46     }
47 }
```

### Generique DAOI

```
1 package DAO;
2
3 import java.util.List;
4 import Model.Employee;
5
6 /**
7  * Interface g n rique pour les op rations CRUD.
8  *
9  * @param <T> Type des entit s.
10  */
11 public interface GeneriqueDAOI<T> {
12
13     // Afficher toutes les entit s.
14     public List<T> afficher();
15
16     // Ajouter une nouvelle entit .
17     public void ajouter(T t);
18
19     // Modifier une entit existante par son ID.
20     public void modifier(T t, int id);
21
22     // Supprimer une entit par son ID.
23     public void supprimer(int id);
24
25     // Rechercher un Employee par son ID.
26     public Employee findById(int EmployeeId);
27 }
```

### EmployerDAOImpl

```
1 package DAO;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.util.*;
8
9 import Controller.EmployeeController;
10 import Model.Employee;
11 import Model.Poste;
12 import Model.Role;
13 import View.EmployeeView;
14
15 /**
16  * Implmentation des interfaces EmployeeDAOI et GeneriqueDAOI.
17  * Cette classe g re les op rations CRUD pour les employ s.
18  */
19 public class EmployeeDAOImpl implements EmployeeDAOI, GeneriqueDAOI<Employee> {
20     private Connection connection;
21
22     // Initialisation de la connexion la base de donn es.
23     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 employes 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"),
89 rset.getString("prenom"),
89                 rset.getDouble("salaire"), rset.getString("email"), rset.
90 getString("phone"),
90                 Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
91 getString("poste")),
91                 rset.getInt("holidayBalance")));
92             }
93         }
94     } catch (SQLException e) {
95         e.printStackTrace();
96     }
97     if (employees.isEmpty()) {
98         EmployeeView.AfficherFail("Aucun employe a t trouv avec cet
99 adresse Email.");
100     }
101     return employees;
102 }
103
104 // Autres mthodes CRUD et de recherche implmentes de mani re similaire,
105 // avec des commentaires succincts pour expliquer leur r le.
106
107 @Override
108 public Employee findById(int EmployeeId) {
109     // Recherche un employe par son ID.
110     String SQL = "SELECT * FROM employee WHERE id = ?";
111     Employee employee = null;
112     EmployeeController.viderLesChamps();
113     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
114         stmt.setInt(1, EmployeeId);
115         try (ResultSet rset = stmt.executeQuery()) {
116             if (rset.next()) {
117                 employee = new Employee(rset.getInt("id"), rset.getString("nom"),
118 rset.getString("prenom"),
119                 rset.getDouble("salaire"), rset.getString("email"), rset.
120 getString("phone"),
121                 Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
122 getString("poste")),
123                 rset.getInt("holidayBalance"));
124             }
125         }
126     } catch (SQLException e) {
127         e.printStackTrace();
128     }
129 }
```

```
125         e.printStackTrace();
126     }
127     return employee;
128 }
129
130 @Override
131 public void modifier(Employee employee, int EmployeeId) {
132     // Met à jour les informations d'un employé par son ID.
133     String SQL = "UPDATE employee SET nom = ?, prenom = ?, salaire = ?, email =
134     ?, phone = ?, role = ?, poste = ? WHERE id = ?";
135     EmployeeController.viderLesChamps();
136     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
137         stmt.setString(1, employee.getNom());
138         stmt.setString(2, employee.getPrenom());
139         stmt.setDouble(3, employee.getSalaire());
140         stmt.setString(4, employee.getEmail());
141         stmt.setString(5, employee.getPhone());
142         stmt.setString(6, employee.getRole().name());
143         stmt.setString(7, employee.getPoste().name());
144         stmt.setInt(8, EmployeeId);
145         stmt.executeUpdate();
146         EmployeeView.ModifierSuccess();
147     } catch (SQLException e) {
148         e.printStackTrace();
149     }
150
151 @Override
152 public void supprimer(int EmployeeId) {
153     // Supprime un employé par son ID.
154     String SQL = "DELETE FROM employee WHERE id = ?";
155     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
156         EmployeeController.viderLesChamps();
157         stmt.setInt(1, EmployeeId);
158         stmt.executeUpdate();
159         EmployeeView.SupprimerSuccess();
160     } catch (SQLException e) {
161         e.printStackTrace();
162     }
163 }
164 }
```

### HolidayDAOImpl

```
1 package DAO;
2
3 import java.sql.*;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 import Controller.EmployeeController;
8 import Model.Employee;
9 import Model.Holiday;
10 import Model.HolidayType;
```



```

11 import Model.Poste;
12 import View.EmployeeView;
13 import View.HolidayView;
14
15 public class HolidayDAOImpl implements GeneriqueDAOI<Holiday> {
16     private Connection connection;
17
18     // Constructeur : Initialise la connexion la base de donn es
19     public HolidayDAOImpl() {
20         connection = DBConnection.getConnection();
21     }
22
23     // R cup re la liste de tous les employ s
24     public List<Employee> afficherEmployee() {
25         List<Employee> employees = new ArrayList<>();
26         String query = "SELECT * FROM employee";
27
28         try (PreparedStatement statement = connection.prepareStatement(query);
29             ResultSet resultSet = statement.executeQuery()) {
30
31             while (resultSet.next()) {
32                 // Cr e un objet Employee partir des donn es SQL
33                 int id = resultSet.getInt("id");
34                 String nom = resultSet.getString("nom");
35                 String prenom = resultSet.getString("prenom");
36                 double salaire = resultSet.getDouble("salaire");
37                 String email = resultSet.getString("email");
38                 String phone = resultSet.getString("phone");
39                 Model.Role role = Model.Role.valueOf(resultSet.getString("role"));
40                 Model.Poste poste = Poste.valueOf(resultSet.getString("poste"));
41                 int holidayBalance = resultSet.getInt("holidayBalance");
42                 Employee employee = new Employee(id, nom, prenom, salaire, email,
phone, role, poste, holidayBalance);
43                 employees.add(employee);
44             }
45         } catch (SQLException e) {
46             e.printStackTrace();
47         }
48
49         return employees;
50     }
51
52     // R cup re la liste de tous les cong s
53     public List<Holiday> afficher() {
54         List<Holiday> holidays = new ArrayList<>();
55         String query = "SELECT * FROM holiday";
56
57         try (PreparedStatement statement = connection.prepareStatement(query);
58             ResultSet resultSet = statement.executeQuery()) {
59
60             while (resultSet.next()) {
61                 // Cr e un objet Holiday partir des donn es SQL
62                 int id = resultSet.getInt("id");
63                 int employeeId = resultSet.getInt("employee_id");
64                 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 congé dans la base de données
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("Congé ajout avec succès !");
88     } catch (SQLException e) {
89         e.printStackTrace();
90     }
91 }
92
93 // Modifie les informations d'un congé 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("Congé modifié avec succès !");
105    } catch (SQLException e) {
106        e.printStackTrace();
107    }
108 }
109
110 // Met à jour le solde de congés d'un employé
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();
```

```
117         } catch (SQLException e) {
118             e.printStackTrace();
119         }
120     }
121
122     // Supprime un congé de la base de données
123     @Override
124     public void supprimer(int holidayId) {
125         String SQL = "DELETE FROM holiday WHERE id = ?";
126         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
127             stmt.setInt(1, holidayId);
128             stmt.executeUpdate();
129             HolidayView.success("Congé supprimé avec succès !");
130         } catch (SQLException e) {
131             e.printStackTrace();
132         }
133     }
134
135     // Recherche un employé par son ID
136     @Override
137     public Employee findById(int EmployeeId) {
138         String SQL = "SELECT * FROM employee WHERE id = ?";
139         Employee employee = null;
140         EmployeeController.viderLesChamps();
141
142         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
143             stmt.setInt(1, EmployeeId);
144             try (ResultSet rset = stmt.executeQuery()) {
145                 if (rset.next()) {
146                     // Crée un objet Employee à partir des données SQL
147                     int id = rset.getInt("id");
148                     String nom = rset.getString("nom");
149                     String prenom = rset.getString("prenom");
150                     double salaire = rset.getDouble("salaire");
151                     String email = rset.getString("email");
152                     String phone = rset.getString("phone");
153                     Model.Role role = Model.Role.valueOf(rset.getString("role"));
154                     Model.Poste poste = Poste.valueOf(rset.getString("poste"));
155                     int holidayBalance = rset.getInt("holidayBalance");
156                     employee = new Employee(id, nom, prenom, salaire, email, phone,
157 role, poste, holidayBalance);
158                 }
159             } catch (SQLException e) {
160                 e.printStackTrace();
161             }
162             return employee;
163         }
164
165     // Recherche un congé par son ID
166     public Holiday FindHolidayById(int holidayId) {
167         String SQL = "SELECT * FROM holiday WHERE id = ?";
168         Holiday holiday = null;
169
170         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

```

1 public class EmployeeController {
2     // Attribuer les modèles et les vues, et configurer les actions des boutons.
3     public EmployeeController(EmployeeModel employeeModel, EmployeeView employeeView)
4     {
5         // Initialiser le modèle et la vue.
6         this.employeeModel = employeeModel;
7         EmployeeController.employeeView = employeeView;
8
9         // Configurer les actions des boutons pour gérer les employés.
10        EmployeeController.employeeView.getAjouterButton().addActionListener(e ->
11        this.ajouterEmployee());
12        EmployeeController.employeeView.getAfficherButton().addActionListener(e ->
13        this.handleAfficher());
14        EmployeeController.employeeView.getSupprimerButton().addActionListener(e ->
15        this.supprimerEmployee());
16        EmployeeController.employeeView.getModifierButton().addActionListener(e ->
17        this.updateEmployee());
18
19        // Afficher les employés lors de l'initialisation.
20        this.afficherEmployee();
21    }
22
23    // Ajouter un nouvel employé.
24    public void ajouterEmployee() {
25        String nom = employeeView.getNomField().getText();
26        String prenom = employeeView.getPrenomField().getText();
27        String salaire = employeeView.getSalaireField().getText();
28        String email = employeeView.getEmailField().getText();
29        String phone = employeeView.getPhoneField().getText();
30        Role role = (Role) employeeView.getRoleComboBox().getSelectedItem();
31        Poste poste = (Poste) employeeView.getPosteComboBox().getSelectedItem();
32
33        employeeModel.ajouterEmployee(nom, prenom, salaire, email, phone, role,
34        poste);
35    }
36
37    // Afficher tous les employés.
38    public void afficherEmployee() {
39        List<Employee> employees = employeeModel.afficherEmployee();
40        DefaultTableModel tableModel = (DefaultTableModel) employeeView.getTable().
41        getModel();
42        tableModel.setRowCount(0);
43        for(Employee e : employees) {
44            tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.
45            getEmail(), e.getSalaire(), e.getPhone(), e.getRole(), e.getPoste(), e.
46            getHolidayBalance()});
47        }
48    }
49

```

```

39     }
40
41     // Gérer les actions de recherche en fonction des champs remplis.
42     private void handleAfficher() {
43         if (allFieldsEmpty()) {
44             this.afficherEmployee();
45         } else {
46             handleSearch();
47         }
48     }
49
50     // Vérifier si tous les champs sont vides.
51     private boolean allFieldsEmpty() {
52         return employeeView.getNomField().getText().isEmpty() &&
53             employeeView.getPrenomField().getText().isEmpty() &&
54             employeeView.getSalaireField().getText().isEmpty() &&
55             employeeView.getEmailField().getText().isEmpty() &&
56             employeeView.getPhoneField().getText().isEmpty();
57     }
58
59     // Effectuer une recherche en fonction des champs spécifiques.
60     private void handleSearch() {
61         // Rechercher par nom complet.
62         if (!employeeView.getNomField().getText().isEmpty() && !employeeView.
63             getPrenomField().getText().isEmpty()) {
64             this.findByFullName(employeeView.getPrenomField().getText(), employeeView
65                 .getNomField().getText());
66         }
67         // Rechercher par nom ou prénom seul.
68         if (!employeeView.getNomField().getText().isEmpty()) {
69             this.findByLastName(employeeView.getNomField().getText());
70         }
71         if (!employeeView.getPrenomField().getText().isEmpty()) {
72             this.findByFirstName(employeeView.getPrenomField().getText());
73         }
74         // Rechercher par d'autres critères.
75         if (!employeeView.getPhoneField().getText().isEmpty()) {
76             this.findByPhone(employeeView.getPhoneField().getText());
77         }
78         if (!employeeView.getEmailField().getText().isEmpty()) {
79             this.findByEmail(employeeView.getEmailField().getText());
80         }
81         if (!employeeView.getSalaireField().getText().isEmpty()) {
82             this.findBySalaire(Double.parseDouble(employeeView.getSalaireField().
83                 getText()));
84         }
85     }
86
87     // Supprimer l'employé sélectionné.
88     public void supprimerEmployee() {
89         int selectedRow = employeeView.getTable().getSelectedRow();
90         if (selectedRow != -1) {
91             int id = Integer.parseInt(employeeView.getTable().getModel().getValueAt(
92                 selectedRow, 0).toString());
93             employeeModel.supprimerEmployee(id);
94         }
95     }

```

```
90     } else {
91         EmployeeView.SupprimerFail("Veuillez choisir un employ .");
92     }
93     this.afficherEmployee();
94 }
95
96 // Mettre jour les informations de l'employ s lectionn .
97 public void updateEmployee() {
98     int selectedRow = employeeView.getTable().getSelectedRow();
99     if (selectedRow != -1) {
100         int id = Integer.parseInt(employeeView.getTable().getModel().getValueAt(
101 selectedRow, 0).toString());
102         String nom = employeeView.getNomField().getText();
103         String prenom = employeeView.getPrenomField().getText();
104         String email = employeeView.getEmailField().getText();
105         double salaire = Utils.parseDouble(employeeView.getSalaireField().getText
106 ());
107         String phone = employeeView.getPhoneField().getText();
108         Role role = (Role) employeeView.getRoleComboBox().getSelectedItem();
109         Poste poste = (Poste) employeeView.getPosteComboBox().getSelectedItem();
110
111         employeeModel.updateEmployee(employeeModel.findById(id), id, nom, prenom,
112 email, salaire, phone, role, poste);
113     } else {
114         EmployeeView.ModifierFail("Veuillez choisir un employ .");
115     }
116 }
117
118 // Vider les champs d'entr e.
119 public static void viderLesChamps() {
120     EmployeeView employeeView = EmployeeView.getInstance();
121     employeeView.getNomField().setText("");
122     employeeView.getPrenomField().setText("");
123     employeeView.getSalaireField().setText("");
124     employeeView.getEmailField().setText("");
125     employeeView.getPhoneField().setText("");
126     employeeView.getRoleComboBox().setSelectedIndex(-1);
127     employeeView.getPosteComboBox().setSelectedIndex(-1);
128 }
```

### HolidayController

```
1 package Controller;
2
3 // Controller class to manage holiday-related actions and interactions between the model
4 and the view.
5
6 import java.util.List;
7
8 import javax.swing.DefaultComboBoxModel;
9 import javax.swing.JComboBox;
10 import javax.swing.table.DefaultTableModel;
11
12 import Model.Employee;
13 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(
67 selectedRow, 0).toString());
68     Holiday oldHoliday = holidayModel.FindHolidayById(idHoliday);
69     Holiday updatedHoliday = new Holiday();
70     updatedHoliday.setId(idHoliday);
71     updatedHoliday.setIdEmployee(Integer.parseInt(holidayView.getNomEmployeComboBox()
72 ().getSelectedItem().toString().split(" - ")[0]));
73     updatedHoliday.setType((HolidayType) holidayView.getTypeComboBox().
74 getSelectedItem());
75     updatedHoliday.setStart(holidayView.getDateDebut());
76     updatedHoliday.setEnd(holidayView.getDateFin());
77     holidayModel.ModifierHoliday(updatedHoliday, oldHoliday); // Update holiday in
78 the model.
79     this.afficherHoliday(); // Refresh the table.
80 }
81
82 // Deletes a selected holiday.
83 public void supprimerHoliday() {
84     int selectedRow = holidayView.getTable().getSelectedRow();
85     if (selectedRow == -1) {
86         HolidayView.fail("Veuillez S lectionner une ligne.");
87         return;
88     } else {
89         int idHoliday = Integer.parseInt(holidayView.getTable().getModel().
90 getValueAt(selectedRow, 0).toString());
91         Holiday oldHoliday = holidayModel.FindHolidayById(idHoliday);
92         holidayModel.supprimerHoliday(oldHoliday); // Remove the holiday from the
93 model.
94     }
95     this.afficherHoliday(); // Refresh the holiday table.
96 }
97
98 // Populates the employee ComboBox with all available employees.
99 public void setEmployeesInComboBox() {
100     List<Employee> employees = holidayModel.afficherEmployee();
101     DefaultComboBoxModel<String> comboBoxModel = new DefaultComboBoxModel<>();
102
103     for (Employee e : employees) {
104         comboBoxModel.addElement(e.getId() + " - " + e.getNom() + " " + e.getPrenom
105 ());
106     }
107
108     holidayView.getNomEmployeComboBox().setModel(comboBoxModel);
109 }
110 }
```

## 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

```
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     {
19         @Override
20         public boolean isCellEditable(int row, int column) {
21             return false;
22         }
23     };
24     protected JTable Tableau = new JTable(tableModel);
25     protected JButton Ajouter = new JButton("Ajouter");
26     protected JButton Modifier = new JButton("Modifier");
27     protected JButton Supprimer = new JButton("Supprimer");
28     protected JButton Afficher = new JButton("Afficher");
29     protected JLabel NomLabel;
30     protected JTextField Nom;
31     protected JLabel PrenomLabel;
32     protected JTextField Prenom;
33     protected JLabel EmailLabel;
34     protected JTextField Email;
35     protected JLabel TelephoneLabel;
36     protected JTextField Telephone;
37     protected JLabel SalaireLabel;
38     protected JTextField Salaire;
39     protected JLabel RoleLabel;
40     protected JComboBox<Role> RoleComboBox;
41     protected JLabel PosteLabel;
42     protected JComboBox<Poste> PosteComboBox;
43
44     public EmployeeView() {
45         setTitle("Gestion des employe s");
46         setDefaultCloseOperation(EXIT_ON_CLOSE);
47         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.setFillViewportHeight(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
succes");
99 }
100 public static void AjouterFail(String message){

```

```

101     JOptionPane.showMessageDialog(null, "L'employe n'a pas e te 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

```
1 package View;
2
3 import javax.swing.*;
4 import javax.swing.table.DefaultTableModel;
5 import Model.Employee;
6 import Model.HolidayType;
7 import Model.Role;
8
9 import java.awt.*;
10
11 public class HolidayView extends JFrame {
12     private static final HolidayView INSTANCE = new HolidayView();
13     private JPanel generalPanel = new JPanel();
14     private JLabel nomEmployeLabel = new JLabel("Nom de l'employé ");
15     private JComboBox<String> nomEmployeComboBox = new JComboBox<>();
16     private JLabel typeLabel = new JLabel("Type");
17     private JComboBox<HolidayType> typeComboBox = new JComboBox<>(HolidayType.values());
18     private JLabel dateDebutLabel = new JLabel("Date de début");
19     private JTextField dateDebut = new JTextField("YYYY-MM-DD");
20     private JLabel dateFinLabel = new JLabel("Date de fin");
21     private JTextField dateFin = new JTextField("YYYY-MM-DD");
22     private DefaultTableModel tableModel = new DefaultTableModel(new String[]{"Id", "Employé", "Type", "Date de début", "Date de fin"}, 0){
23         @Override
24         public boolean isCellEditable(int row, int column) {
25             return false;
26         }
27     };
28     protected JTable holidayTable = new JTable(tableModel);
29     private JScrollPane tableScrollPane = new JScrollPane(holidayTable);
30     private JButton ajouterButton = new JButton("Ajouter");
31     private JButton modifierButton = new JButton("Modifier");
32     private JButton supprimerButton = new JButton("Supprimer");
33     private JButton afficherButton = new JButton("Afficher");
34     private JPanel inputPanel = new JPanel();
35     private JPanel buttonPanel = new JPanel();
36
37     public HolidayView() {
38         setTitle("Gestion des holidays");
39         setDefaultCloseOperation(EXIT_ON_CLOSE);
40         setSize(930, 520);
41         setLocationRelativeTo(null);
42         setVisible(true);
43
44         generalPanel.setLayout(new BorderLayout());
45         inputPanel.setLayout(new GridLayout(5, 2, 10, 10));
46         inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 0, 10));
47
48         inputPanel.add(nomEmployeLabel);
49         inputPanel.add(nomEmployeComboBox);
50         inputPanel.add(typeLabel);
51         inputPanel.add(typeComboBox);
52         inputPanel.add(dateDebutLabel);
53         inputPanel.add(dateDebut);
```

```
54     inputPanel.add(dateFinLabel);
55     inputPanel.add(dateFin);
56     generalPanel.add(inputPanel, BorderLayout.NORTH);
57
58     tableScrollPane.setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 10));
59     generalPanel.add(tableScrollPane, BorderLayout.CENTER);
60
61     buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 10));
62     buttonPanel.add(ajouterButton);
63     buttonPanel.add(modifierButton);
64     buttonPanel.add(supprimerButton);
65     buttonPanel.add(afficherButton);
66     generalPanel.add(buttonPanel, BorderLayout.SOUTH);
67
68     add(generalPanel);
69
70     holidayTable.setFillViewportHeight(true);
71     holidayTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
72 }
73
74 public JComboBox<String> getNomEmployeComboBox() {
75     return nomEmployeComboBox;
76 }
77 public JComboBox<HolidayType> getTypeComboBox() {
78     return typeComboBox;
79 }
80 public String getDateDebut() {
81     return dateDebut.getText();
82 }
83 public String getDateFin() {
84     return dateFin.getText();
85 }
86 public JButton getAfficherButton() {
87     return afficherButton;
88 }
89 public JButton getAjouterButton() {
90     return ajouterButton;
91 }
92 public JButton getModifierButton() {
93     return modifierButton;
94 }
95
96 public JButton getSupprimerButton() {
97     return supprimerButton;
98 }
99 public JTable getHolidayTable() {
100     return holidayTable;
101 }
102 public static HolidayView getInstance() {
103     return INSTANCE;
104 }
105 public static void success(String message) {
106     JOptionPane.showMessageDialog(null, message, "Success", JOptionPane.
107     INFORMATION_MESSAGE);
108 }
```



```
108     public static void fail(String message) {
109         JOptionPane.showMessageDialog(null, message, "Error", JOptionPane.ERROR_MESSAGE)
110     ;
111     }
112     public JTable getTable() {
113         return holidayTable;
114     }
115 }
```

### **PanelsView**

```
1 package View;
2
3 import javax.swing.*;
4
5 public class PanelsView extends JFrame {
6     private static PanelsView INSTANCE = new PanelsView();
7     private JTabbedPane tabbedPane = new JTabbedPane();
8     private EmployeeView employeeView = EmployeeView.getInstance();
9     private HolidayView holidayView = HolidayView.getInstance();
10
11     public PanelsView() {
12         setTitle("Admin Dashboard - Gestion des Employ s et Cong s");
13         setDefaultCloseOperation(EXIT_ON_CLOSE);
14         setSize(930, 520);
15         setLocationRelativeTo(null);
16         employeeView.dispose();
17         holidayView.dispose();
18         tabbedPane.addTab("Gestion des Employ s", employeeView.getContentPane());
19         tabbedPane.addTab("Gestion des Cong s", holidayView.getContentPane());
20         add(tabbedPane);
21         setVisible(true);
22     }
23     public static PanelsView getInstance() {
24         return INSTANCE;
25     }
26 }
```

# Résultats

## 1 Tables Créées

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> employee	★ Parcourir Structure Rechercher Insérer Vider Supprimer	2	InnoDB	utf8mb4_general_ci	32,0 kio	-
<input type="checkbox"/> 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

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

Id	Nom	Prénom	Email	Salaire	Phone	Role	Poste	Holiday Balance
4	Laddad	Hajar	hajarladdad@g...	23000.0	0612345678	ADMIN	INGENIEUR_E...	25

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

Id	Nom	Prénom	Email	Salaire	Phone	Role	Poste	Holiday Balance
4	Laddad	Hajar	hajarladdad@g...	23000.0	0612345678	ADMIN	INGENIEUR_E...	25

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.

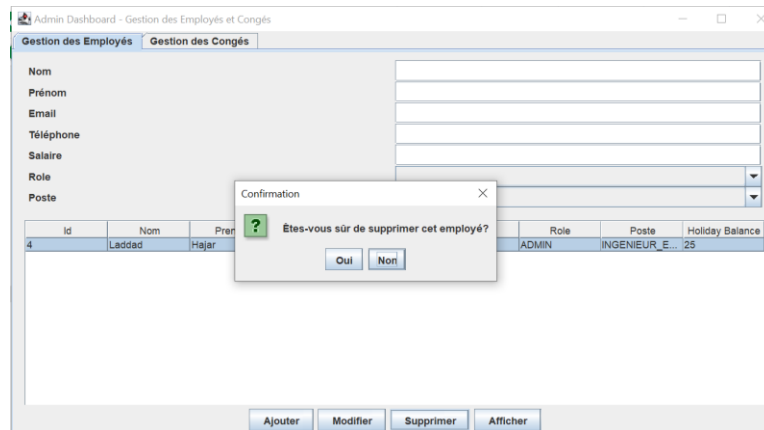


FIGURE 2.8 – Message pour s'assurer de la suppression

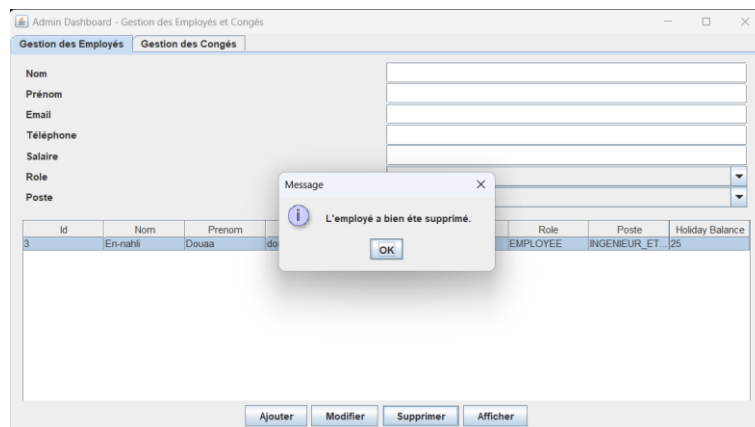


FIGURE 2.9 – Resultat de la suppression

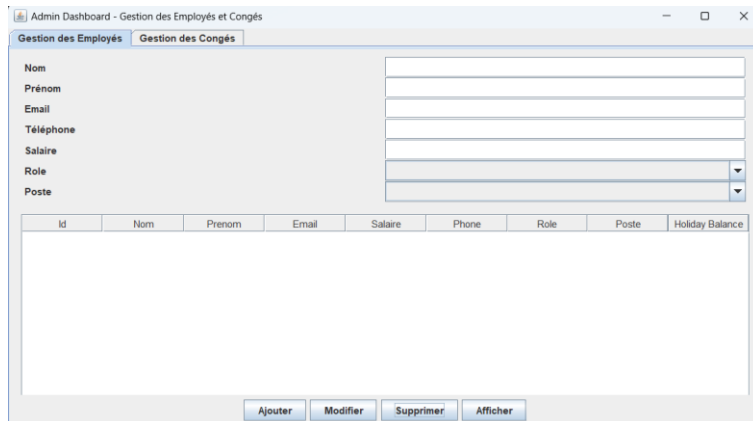


FIGURE 2.10 – Affichage de la suppression

## 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.

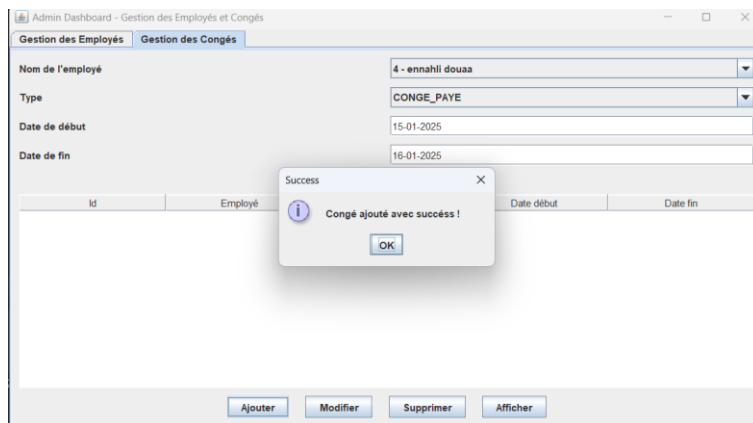


FIGURE 2.11 – Resultat Ajout

## 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.

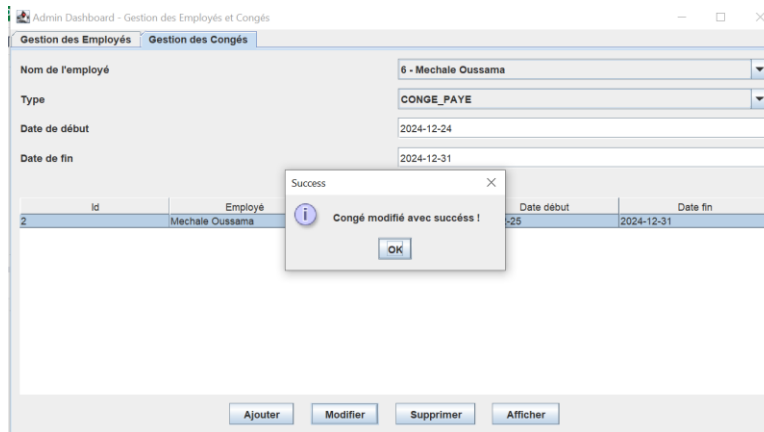


FIGURE 2.12 – Resultat de modification

## 2.8 Après 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.

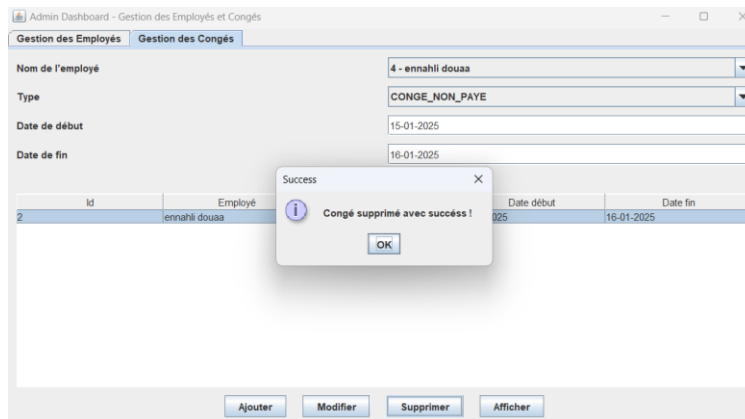


FIGURE 2.13 – Message pour s'assurer de la suppression

# Gestion des fichiers E/S

## 1 interface" DataImportExport "

```
1 package DAO;
2
3 import java.io.IOException;
4 import java.util.List;
5
6 public interface DataImportExport <T> {
7     void importData(String fileName) throws IOException;
8     void exportData(String fileName , List<T> data) throws IOException;
9 }
```

## 2 EmployeeDAOImpl

```
1 @Override
2 public void importData(String filePath) {
3     String query ="INSERT INTO employee (nom, prenom, salaire, email, phone, role,
4 poste) VALUES (?, ?, ?, ?, ?, ?, ?)" ;
5     try(BufferedReader reader =new BufferedReader(new FileReader(filePath)) ;
6 PreparedStatement pstmt = connection.prepareStatement(query)){
7         String Line = reader.readLine();
8         while((Line = reader.readLine()) != null){
9             String[] data =Line.split(",");
10             if (data.length ==7){
11                 pstmt.setString(1, data[0].trim());
12                 pstmt.setString(2, data[1].trim());
13                 pstmt.setString(3, data[2].trim());
14                 pstmt.setString(4, data[3].trim());
15                 pstmt.setString(5, data[4].trim());
16                 pstmt.setString(6, data[5].trim());
17                 pstmt.setString(7, data[6].trim());
18                 pstmt.addBatch();
19             }
20             System.out.println("data imported "+data.length);
21         }
22         pstmt.executeBatch();
23     }
24 }
```



```
22         System.out.println("Employee imported successfully");
23
24
25         }catch(IOException | SQLException e){
26             e.printStackTrace();
27         }
28
29     }
30
31     @Override
32     public void exportData(String fileName, List<Employee> data) throws IOException {
33         try(BufferedWriter writer =new BufferedWriter(new FileWriter(fileName))){
34             writer.write("nom, prenom, salaire, email, phone, role, poste");
35             writer.newLine();
36             for(Employee employee : data){
37                 String Line = String.format("%s,%s,%.2f,%s,%s,%s,%s",
38                     employee.getNom(),
39                     employee.getPrenom(),
40                     employee.getSalaire(),
41                     employee.getEmail(),
42                     employee.getPhone(),
43                     employee.getRole(),
44                     employee.getPoste());
45                 writer.write(Line);
46                 writer.newLine();
47             }
48         }
49     }
50 }
```

### 3 EmployeeModel"ajout des E/s"

```
1     private boolean checkFileExists(File file) {
2
3         if(!file.exists()) {
4             throw new IllegalArgumentException ("le fichier n'existe pas "+file.getPath
5             ());
6         }
7         return true;
8     }
9
10    private boolean checkIsFile(File file) {
11
12        if(!file.isFile()) {
13            throw new IllegalArgumentException ("le chemin specifie nest pas un fichier
14            "+file.getPath());
15        }
16        return true;
17    }
18
19    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     {
19         @Override
20         public boolean isCellEditable(int row, int column) {
21             return false;
22         }
23     };
24     protected JTable Tableau = new JTable(tableModel);
25     protected JButton Ajouter = new JButton("Ajouter");
26     protected JButton Import = new JButton("Import");
27     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("Pre 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.setFillViewportHeight(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
104 succe s");
105 }
106 public static void AjouterFail(String message){
107     JOptionPane.showMessageDialog(null, "L'employe n'a pas e te ajoute . " +
108 message);
109 }
110 public static void AfficherFail(String message){
111     JOptionPane.showMessageDialog(null, message);
112 }
113 public static void SupprimerSuccess(){
114     JOptionPane.showMessageDialog(null, "L'employe a bien te supprim .");
115 }
116 public static void SupprimerFail(String message){
117     JOptionPane.showMessageDialog(null, message);
118 }
119 public static void ModifierSuccess(){
120     JOptionPane.showMessageDialog(null, "L'employe a bien t modifi .");
121 }
122 public static void ModifierFail(String message){
123     JOptionPane.showMessageDialog(null, message);
124 }
125 protected void CacherColumn(int index){
126     Tableau.getColumnModel().getColumn(index).setMinWidth(0);
127     Tableau.getColumnModel().getColumn(index).setMaxWidth(0);
128     Tableau.getColumnModel().getColumn(index).setWidth(0);
129 }
130 public static boolean SupprimerConfirmation(){
131     int choice = JOptionPane.showOptionDialog(null, "tes -vous s r de supprimer
132 cet employe ?", "Confirmation", JOptionPane.YES_NO_OPTION, JOptionPane.
QUESTION_MESSAGE, null, new String[]{"Oui", "Non"}, "Non");
    return choice == JOptionPane.YES_OPTION;
}
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
```

```
188 public JTextField getPhoneField() {
189     return Telephone;
190 }
191
192 public void setPhoneField(JTextField phoneField) {
193     Telephone = phoneField;
194 }
195
196 public JComboBox<Role> getRoleComboBox() {
197     return RoleComboBox;
198 }
199
200 public void setRoleComboBox(JComboBox<Role> roleComboBox) {
201     RoleComboBox = roleComboBox;
202 }
203
204 public JComboBox<Poste> getPosteComboBox() {
205     return PosteComboBox;
206 }
207
208 public void setPosteComboBox(JComboBox<Poste> posteComboBox) {
209     PosteComboBox = posteComboBox;
210 }
211 public static EmployeeView getInstance() {
212     return INSTANCE;
213 }
214 }
```

## 5 EmployeeController

Ajouter au premier que les actionlistener 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        }
13    }
```

```

13     }
14 }
15
16 public void handleExport() {
17     JFileChooser fileChooser = new JFileChooser();
18     fileChooser.setFileFilter(new FileNameExtensionFilter("Fichiers CSV", "csv"));
19
20     if (fileChooser.showSaveDialog(employeeView) == JFileChooser.APPROVE_OPTION) {
21         try {
22             String filePath = fileChooser.getSelectedFile().getAbsolutePath();
23             if (!filePath.toLowerCase().endsWith(".txt")) {
24                 filePath += ".txt";
25             }
26             List<Employee> employees = new EmployeeDAOImpl().afficher();
27             employeeModel.exportData(filePath, employees);
28             employeeView.AfficherFail("Exportation r ussie !");
29         } catch (Exception ex) {
30             employeeView.AfficherFail("Une erreur inattendue est survenue : " + ex.
31             getMessage());
32         }
33     }

```

## 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.

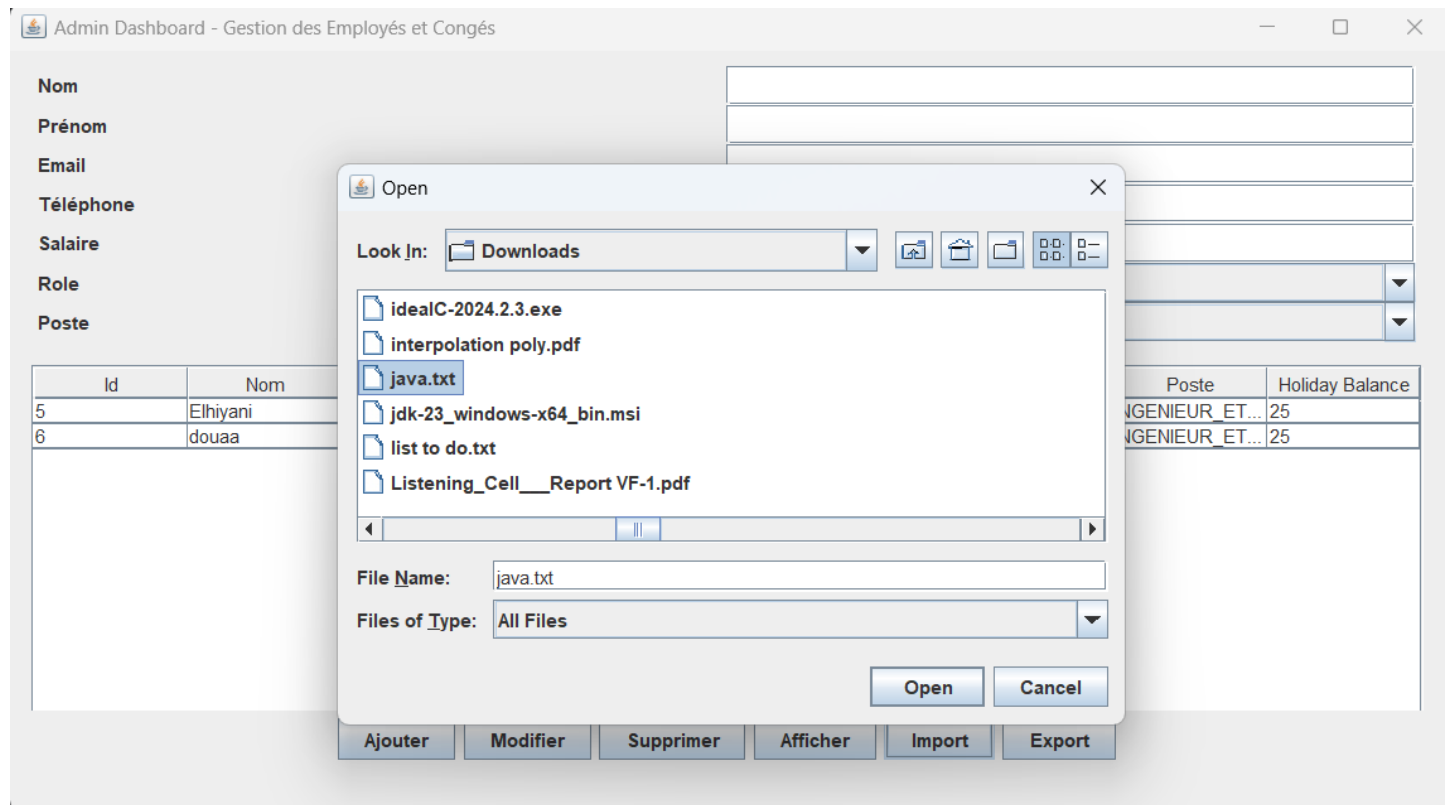


FIGURE 4.1 – Processu d’importation .

```
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.



Admin Dashboard - Gestion des Employés et Congés

Gestion des Employés
Gestion des Congés

Nom

Prénom

Email

Téléphone

Salaire

Role

Poste

Id	Nom	Prenom	Email	Salaire	Phone	Role	Poste	Holiday Balance
5	Elhiyani	hanane	hananeelhiyani@...	5600.0	0736037390	ADMIN	INGENIEUR_ET...	25
6	douaa	douaa	douaaennahli@g...	15200.0	0771567572	ADMIN	INGENIEUR_ET...	25
7	En-nahli	yahya	yahyaennahli@g...	50600.0	0670809050	ADMIN	TEAM_LEADER	25
8	Safaa	Safaa	safaasafaa@gm...	15200.0	0771567572	MANAGER	PILOTE	25
9	Bouarbine	amina	aminabouarbine...	500600.0	0636037390	ADMIN	TEAM_LEADER	25

Ajouter
Modifier
Supprimer
Afficher
Import
Export

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.

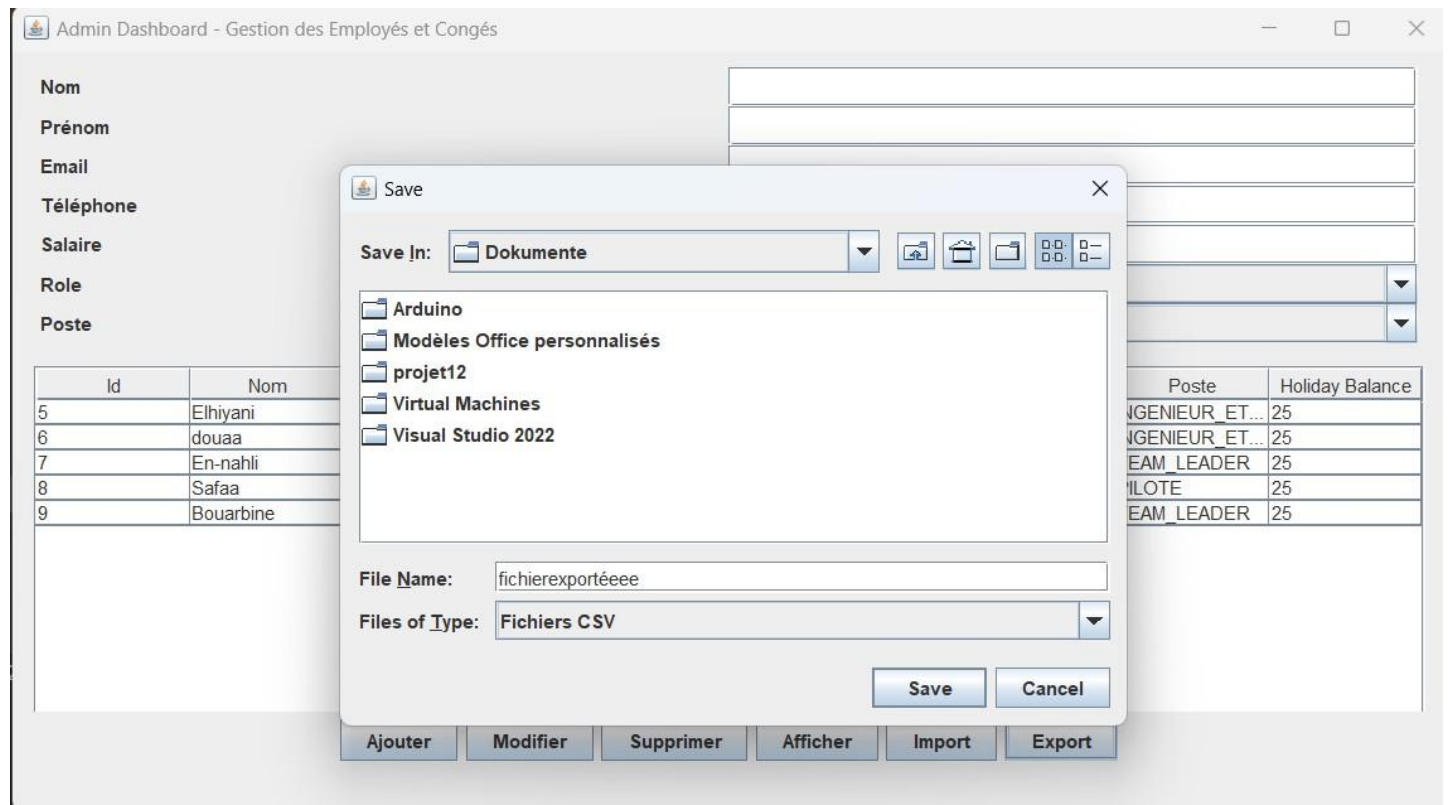


FIGURE 4.4 – Affichage du fichier importé.

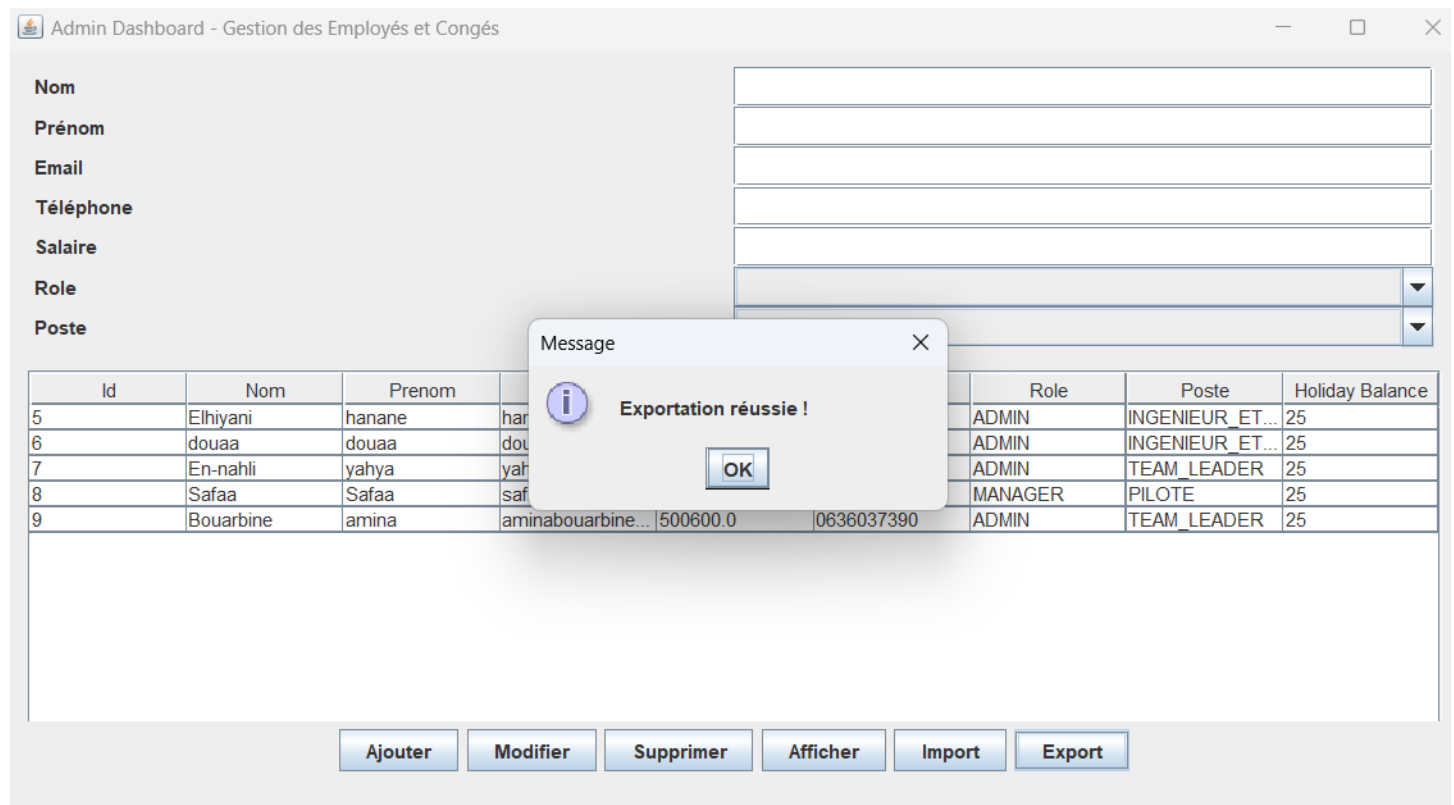


FIGURE 4.5 – réussite d’exportation.

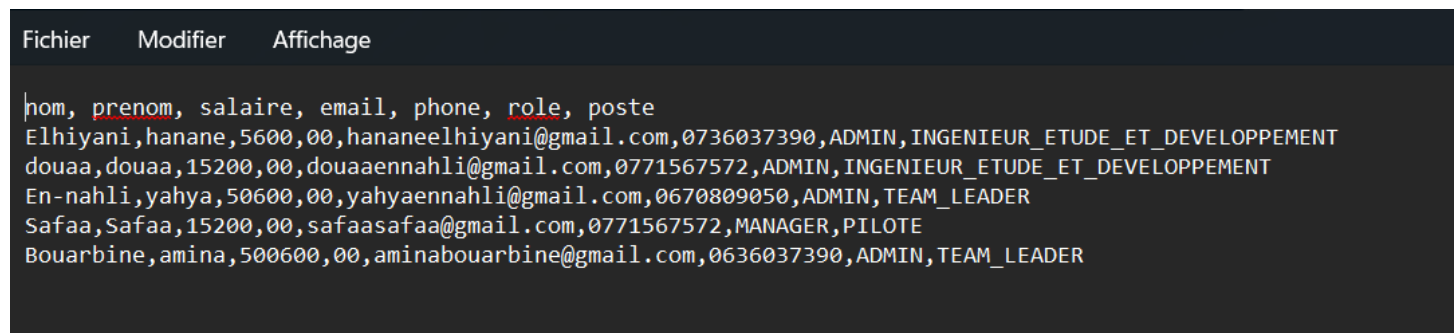


FIGURE 4.6 – Affichage du fichier exporté.

## 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 modulaire, 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/hwijer/TP-JAVA-final>