

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

**gestion des congés**

---

# Rapport du tp 2( java avancé)

---

**Réalisé par : ELHIYANI Hanane**

**Encadré par : Mme.ELKHROF Leila**

**ANNÉE UNIVERSITAIRE : 2024/2025**

# Table des matières

<b>Introduction</b>	<b>4</b>
<b>Fonctionnalités principales</b>	<b>5</b>
<b>Outils &amp; environnement de travail</b>	<b>5</b>
1    Language de programmation . . . . .	5
2    Environnement de travail . . . . .	6
<b>1 Elaboration &amp; Réalisation</b>	<b>7</b>
1    Implémentation de l'architecture MVC avec le pattern DAO . . . . .	8
2    Couche Model . . . . .	8
3    Couche DAO . . . . .	14
4    Couche Controller . . . . .	22
5    Couche View . . . . .	25
<b>2 Résultats</b>	<b>30</b>
1    Button ajouter . . . . .	31
2    Button Modifier . . . . .	32
<b>3 Conclusion</b>	<b>33</b>

# Table des figures

1	JAVA logo . . . . .	5
2	intellij idea logo . . . . .	6
3	xampp logo . . . . .	6
2.1	Resultat ajout de conge . . . . .	30
2.2	change de balance . . . . .	31
2.3	Echoue de l'ajout de conge . . . . .	31
2.4	la cause de l'echoue de l'ajout . . . . .	32
2.5	Resultat du Modification . . . . .	32
2.6	Resultat du Modification dans Employe . . . . .	32

# Inroduction

Ce deuxième travail pratique (TP N°2) intitulé *Gestion de Congé* a pour objectif de mettre en pratique les concepts de généricité, le modèle MVC (Model-View-Controller), et l'architecture DAO (Data Access Object). Il s'agit de développer une application de gestion des congés avec une interface graphique en utilisant Swing.

Les étapes de ce TP visent à guider les apprenants dans l'implémentation d'une solution complète, de la définition du modèle de données à la création de l'interface utilisateur, en passant par la gestion des données et la logique métier. Ce TP constitue ainsi une opportunité d'approfondir les notions théoriques tout en renforçant les compétences techniques à travers une mise en œuvre pratique.

# Fonctionnalités principales :

L'application propose deux fonctionnalités clés :

1. **Gestion des Employés** : Un module complet permettant l'ajout, la modification, la suppression et la consultation des informations des employés, assurant une utilisation fluide et intuitive (déjà détaillé dans le TP 1).
2. **Gestion des Congés** : Une interface dédiée à la gestion et à l'approbation des demandes de congés, parfaitement synchronisée avec la base de données pour assurer une administration cohérente et efficace.

Ces fonctionnalités sont implémentées selon le modèle architectural MVC (Model-View-Controller), garantissant une séparation claire des responsabilités entre la logique métier, l'interface utilisateur et l'accès aux données.

## Outils & environnement de travail

### 1 Language de programmation



FIGURE 1 – JAVA logo

- **Java** : est un langage de programmation orienté objet, sécurisé, portable et indépendant de la plateforme. Il est utilisé pour développer des applications web, mobiles et de bureau, grâce à sa machine

virtuelle Java (JVM), qui permet d'exécuter le même code sur différentes plateformes.

## 2 Environnement de travail



FIGURE 2 – intellij idea logo

- **intellij idea** : est un environnement de développement intégré (IDE) puissant et populaire, principalement utilisé pour le développement en Java. Développé par JetBrains, il offre des fonctionnalités avancées telles que l'autocomplétion intelligente, la navigation dans le code, le débogage, le contrôle de version intégré et la prise en charge de plusieurs langages de programmation. IntelliJ IDEA est apprécié pour sa productivité accrue et son interface conviviale.



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



- **MySQL Workbench** : est un outil visuel de gestion de bases de données développé par Oracle. Il permet de concevoir, modéliser, administrer et interagir avec des bases de données MySQL.

# Elaboration & Réalisation

## Création de la base de donnée : Ajout de la table `Holiday` dans la base de données

Le TP n°2, intitulé *Gestion de Congé*, fait suite au TP n°1, *Gestion des Employés*. Dans ce projet, l'objectif est d'étendre la base de données existante pour y intégrer une nouvelle fonctionnalité liée à la gestion des congés des employés.

Pour cela, une nouvelle table `Holiday` a été ajoutée à la base de données. Cette table est dédiée à l'enregistrement des demandes de congé des employés, ainsi qu'à la gestion de l'approbation ou du rejet de ces demandes. Elle permettra de gérer les informations relatives aux congés, telles que la date de début et de fin, le type de congé, ainsi que l'identifiant de l'employé qui fait la demande.

```
1 //
2
3
4 //
5 USE EmployeeDB;
6 //
7
8 CREATE TABLE holiday(
9     id int primary key auto_increment,
10    employeeId int,
11    CONSTRAINT fk_nom_employe
12    FOREIGN KEY (employeeId) REFERENCES employee(id)
13        ON UPDATE CASCADE
14        ON DELETE RESTRICT,
15    type varchar(100) not null,
16    startdate varchar(50) NOT NULL,
17    enddate varchar(50) NOT NULL
18
19 );
```

Listing 1.1 – Script SQL de la base de données

# 1 Implémentation de l'architecture MVC avec le pattern DAO

## 2 Couche Model

- **Holiday :**

La classe `Holiday` représente les congés des employés dans le modèle de l'application. Elle comprend plusieurs attributs, notamment `id` pour identifier chaque congé, `employeeId` comme clé étrangère associée à un employé, `type` pour définir le type de congé, ainsi que `startDate` et `endDate` pour spécifier les dates de début et de fin du congé. Deux constructeurs ont été mis en place : un constructeur complet permettant d'initialiser tous les champs, utile pour gérer des congés existants, et un constructeur sans `id`, utilisé lors de l'ajout de nouveaux congés où l'identifiant est généré automatiquement.

Pour faciliter la manipulation des données, des getters et setters sont disponibles pour chaque attribut, assurant un accès et une modification sécurisés. Enfin, des méthodes supplémentaires, comme `getFormattedStartDate` et `getFormattedEndDate`, permettent de formater les dates selon un format spécifié ou de gérer les valeurs nulles, garantissant ainsi une gestion cohérente et structurée des informations liées aux congés.



```
1 package Model;
2
3 // Classe Holiday
4 public class Holiday {
5     private int id;
6     private int employeeId; // Cl     trangre     vers l'Employ
7     private Type type;
8     private String startDate;
9     private String endDate;
10
11     // Constructeur avec tous les champs
12     public Holiday(int id, int employeeId, Type type, String startDate, String
endDate) {
13         this.id = id;
14         this.employeeId = employeeId;
15         this.type = type;
16         this.startDate = startDate;
17         this.endDate = endDate;
18     }
19
20     // Constructeur sans "id" (par exemple pour l'ajout d'un nouveau cong )
21     public Holiday(int employeeId, Type type, String startDate, String endDate) {
22         this.employeeId = employeeId;
23         this.type = type;
24         this.startDate = startDate;
25         this.endDate = endDate;
26     }
27
28     // Getter et setter pour "id"
29     public int getId() {
30         return id;
31     }
32
33     public void setId(int id) {
34         this.id = id;
35     }
36
37     // Getter et setter pour "employeeId"
38     public int getEmployeeId() {
39         return employeeId;
40     }
41
42     public void setEmployeeId(int employeeId) {
43         this.employeeId = employeeId;
44     }
45
46     // Getter et setter pour "type"
47     public Type getType() {
48         return type;
49     }
50
51     public void setType(Type type) {
52         this.type = type;
53     }
54 }
```

```
55 // Getter et setter pour "startDate"
56 public String getStartDate() {
57     return startDate;
58 }
59
60 public void setStartDate(String startDate) {
61     this.startDate = startDate;
62 }
63
64 // Getter et setter pour "endDate"
65 public String getEndDate() {
66     return endDate;
67 }
68
69 public void setEndDate(String endDate) {
70     this.endDate = endDate;
71 }
72 public String getFormattedStartDate(String dateFormat) {
73     return startDate;
74 }
75
76 public String getFormattedEndDate(String dateFormat) {
77     return endDate != null ? endDate : "Non retourn ";
78 }
79
80
81 }
```

- **HolidayModel**

La classe `HolidayModel` représente la couche métier dans la gestion des congés, reliant l'interface utilisateur à la couche d'accès aux données (DAO). Elle fournit des méthodes pour ajouter, modifier, supprimer et afficher les congés des employés, tout en intégrant des validations pour garantir l'intégrité des données. Avant d'ajouter ou de modifier un congé, elle effectue plusieurs vérifications, notamment la validité des plages de dates, l'absence de chevauchements avec d'autres congés, et la disponibilité du solde de congés de l'employé. Elle assure également que les dates saisies respectent les contraintes, comme ne pas débiter avant la date actuelle.

De plus, la classe propose une méthode pour récupérer tous les congés sous forme de tableau, adapté à l'affichage dans une interface utilisateur. Enfin, elle intègre des fonctionnalités pour calculer la durée des congés et gérer les erreurs, rendant le système de gestion des congés fiable et efficace.

```
1 package Model;
2
3 import DAO.HolidayDAOImpl;
4 import java.text.ParseException;
5 import java.text.SimpleDateFormat;
6 import java.time.LocalDate;
7 import java.util.Collections;
8 import java.util.Date;
9 import java.util.List;
10
11 public class HolidayModel {
12     private final HolidayDAOImpl dao;
13     private static final String DATE_FORMAT = "yyyy-MM-dd";
14
15     public HolidayModel(HolidayDAOImpl dao) {
16         this.dao = dao;
17     }
18
19     // Ajouter un nouveau cong
20     public boolean add(String employee, Type type, String startDate, String endDate) {
21         if (!isValidDateRange(startDate, endDate)) {
22             return false; // Invalid date range, returning false
23         }
24
25         // Convertir les noms d'employ en IDs correspondants
26         int employeeId = dao.getEmployeeIdByName(employee);
27
28         if (employeeId == -1) {
29             return false; // Si aucun ID correspondant trouv , retour false
30         }
31
32         if (isStartBeforeToday(startDate)) {
33             System.err.println("La date de d but doit venir apr s aujourd'hui.");
34             return false;
35         }
36
37         // Vrification des chevauchements
38         if (hasHolidayOverlap(employeeId, startDate, endDate)) {
39             System.err.println("L'employ a d j un cong pendant cette p riode.");
40             return false;
41         }
42
43         // Vrifier la balance de cong
44         int holidayDays = calculateHolidayDuration(startDate, endDate);
45         if (!hasSufficientHolidayBalance(employeeId, holidayDays)) {
46             System.err.println("Le nombre de jours de cong est insuffisant.");
47             return false;
48         }
49
50         Holiday holiday = new Holiday(employeeId, type, startDate, endDate);
51         try {
52             dao.add(holiday); // Appeler la m thode DAO pour ajouter un cong
53             return true;
54         } catch (Exception e) {
55             logError("Erreur lors de l'ajout du cong ", e);
56         }
57     }
58 }
```

```
56         return false;
57     }
58 }
59
60 // V rifier si la date de d but est avant aujourd'hui
61 private boolean isStartBeforeToday(String startDate) {
62     LocalDate start = LocalDate.parse(startDate);
63     return start.isBefore(LocalDate.now());
64 }
65
66 // V rifier les chevauchements de cong s
67 private boolean hasHolidayOverlap(int employeeId, String startDate, String endDate) {
68     List<Holiday> employeeHolidays = dao.getHolidaysByEmployeeId(employeeId);
69     LocalDate newStart = LocalDate.parse(startDate);
70     LocalDate newEnd = LocalDate.parse(endDate);
71
72     for (Holiday holiday : employeeHolidays) {
73         LocalDate currentStart = LocalDate.parse(holiday.getStartDate());
74         LocalDate currentEnd = LocalDate.parse(holiday.getEndDate());
75
76         if ((newStart.isBefore(currentEnd) && newEnd.isAfter(currentStart))) {
77             return true; // Il y a un chevauchement
78         }
79     }
80     return false; // Pas de chevauchement
81 }
82
83 // V rifier si l'employ e a un solde de cong suffisant
84 private boolean hasSufficientHolidayBalance(int employeeId, int holidayDays) {
85     // R cup rer la balance de cong de l'employ e partir de la table Employee
86     int currentBalance = dao.getHolidayBalance(employeeId);
87
88     // V rifier si la balance de cong est suffisante
89     if (currentBalance >= holidayDays) {
90         return true;
91     } else {
92         System.err.println("Solde insuffisant. L'employ e n'a que " + currentBalance
93 + " jours de cong .");
94         return false;
95     }
96 }
97
98 // Calculer le nombre de jours entre deux dates
99 private int calculateHolidayDuration(String startDate, String endDate) {
100     try {
101         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
102         long start = sdf.parse(startDate).getTime();
103         long end = sdf.parse(endDate).getTime();
104         long diffInMillis = end - start;
105         return (int) (diffInMillis / (1000 * 60 * 60 * 24)) + 1; // Inclure le
premier et le dernier jour
106     } catch (ParseException e) {
107         e.printStackTrace();
108         return 0;
109     }
110 }
```

```

109     }
110     // Mettre jour un cong existant
111     public boolean update(int holidayId, String employee, Type type, String startDate,
112                           String endDate) {
113         if (!isValidDateRange(startDate, endDate)) {
114             return false; // Invalid date range, returning false
115         }
116
117         int employeeId = dao.getEmployeeIdByName(employee);
118         if (employeeId == -1) {
119             return false; // Si aucun ID correspondant trouv , retour false
120         }
121
122         if (isStartBeforeToday(startDate)) {
123             System.err.println("La date de d but doit venir apr s aujourd'hui.");
124             return false;
125         }
126
127
128         int holidayDays = calculateHolidayDuration(startDate, endDate);
129         if (!hasSufficientHolidayBalance(employeeId, holidayDays)) {
130             System.err.println("Le nombre de jours de cong est insuffisant.");
131             return false;
132         }
133
134         Holiday holiday = new Holiday(holidayId, employeeId, type, startDate, endDate);
135         try {
136             dao.update(holiday); // Appeler la m thode DAO pour mettre jour le
137             cong
138             return true;
139         } catch (Exception e) {
140             logError("Erreur lors de la mise jour du cong ", e);
141             return false;
142         }
143
144     // Supprimer un cong
145     public boolean delete(int holidayId) {
146         try {
147             dao.delete(holidayId); // Appeler la m thode DAO pour supprimer le cong
148             return true;
149         } catch (Exception e) {
150             logError("Erreur lors de la suppression du cong ", e);
151             return false;
152         }
153     }
154
155     // R cup rer tous les cong s sous forme de tableau pour l'affichage
156     public Object[][] getAllHolidaysAsTableData() {
157         List<Holiday> holidayList = dao.getAll();
158         Object[][] tab = new Object[holidayList.size()][5]; // 5 colonnes : id, employ
159         , type, d but , fin
160
161         for (int i = 0; i < holidayList.size(); i++) {

```

```

161         Holiday holiday = holidayList.get(i);
162         tab[i][0] = holiday.getId();
163         String employeeName = dao.getNameEmployeeById(holiday.getEmployeeId());
164         tab[i][1] = employeeName;
165         tab[i][2] = holiday.getType();
166         tab[i][3] = holiday.getFormattedStartDate (DATE_FORMAT);
167         tab[i][4] = holiday.getFormattedEndDate (DATE_FORMAT);
168     }
169
170     return tab;
171 }
172
173 // Mthode pour journaliser les erreurs
174 private void logError(String message, Exception e) {
175     System.err.println(message);
176     e.printStackTrace();
177 }
178 private boolean isValidDateRange(String startDate, String endDate) {
179     SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd"); // Le format des
180     dates utilis
181     try {
182         // Convertir les cha nes de dates en objets Date
183         java.util.Date start = sdf.parse(startDate);
184         java.util.Date end = sdf.parse(endDate);
185
186         // V rifier si la date de fin est apr s la date de d but
187         return !start.after(end);
188     } catch (ParseException e) {
189         e.printStackTrace();
190     }
191     return false; // Retourner false si les dates sont invalides ou la conversion
192     choue
193 }
194
195 // HolidayModel.java
196 public List<String> getEmployes() {
197     try {
198         return dao.getAllEmployeeNames();
199     } catch (Exception e) {
200         logError("Erreur lors de la r cup ration des employ s", e);
201         return Collections.emptyList(); // Retourne une liste vide en cas d'erreur
202     }
203 }

```

### 3 Couche DAO

- **connexion :**

La classe `connexion` fait partie du package DAO (*Data Access Object*) et est responsable de la gestion de la connexion à la base de données. Elle utilise le pilote JDBC pour se connecter à une base de données MySQL, en définissant les informations nécessaires, telles que l'URL, le nom d'utilisateur (USER) et le mot de passe (PASSWORD).

La méthode statique `getConnexion()` établit et retourne une connexion à la base de données. Si une erreur survient lors de la tentative de connexion, elle capture l'exception `SQLException` et affiche un message d'erreur dans la console, permettant ainsi un débogage plus facile. Cette classe centralise la gestion des connexions, offrant une réutilisabilité et une maintenance simplifiée dans l'application.

```
1 package DAO;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class connexion {
8     private static final String URL="jdbc:mysql://localhost:3306/BDEmploye";
9     private static final String USER="root";
10    private static final String PASSWORD="";
11    static Connection conn=null;
12    public static Connection getConnexion() {
13        try {
14            conn= DriverManager.getConnection(URL,USER,PASSWORD);
15        }catch (SQLException e){
16            e.printStackTrace();
17        }
18        return conn;
19    }
20
21 }
```

- **GeniriqueDAOI**

L'interface `GeniriqueDAOI` est une interface générique définie dans le package DAO. Elle joue un rôle central dans la gestion des opérations CRUD (*Create, Read, Update, Delete*) en définissant des méthodes génériques : `add(T t)`, `update(T t)`, `delete(int id)`, et `getAll()`. Ces méthodes sont utilisées pour manipuler les entités métier de l'application, telles que `Employe` et `Holiday`.

L'utilisation d'une interface générique permet de centraliser la définition des opérations communes, évitant ainsi la redondance dans le code. Grâce à cette approche, les mêmes méthodes peuvent être implémentées pour différentes entités, ce qui réduit le risque d'erreurs, améliore la maintenabilité, et facilite l'ajout de nouvelles entités sans dupliquer les méthodes. Cette conception assure une gestion plus efficace et cohérente des données de l'application.

```

1 package DAO;
2
3 import java.util.List;
4
5 public interface GeniriqueDAOI<T>{
6     public void add(T t);
7     public void update(T t);
8     public void delete(int id);
9     public List<T> getAll();
10
11 }

```

### • HolidayDAOImpl

La classe `HolidayDAOImpl` représente une implémentation d'accès aux données permettant de gérer les congés des employés dans une base de données. Elle propose des fonctionnalités essentielles telles que l'ajout, la mise à jour, la suppression et la récupération des congés. De plus, elle prend en charge la gestion des balances de congés en calculant automatiquement la durée des congés et en ajustant les soldes des employés. En utilisant des requêtes SQL préparées, cette classe garantit une interaction sécurisée avec la base de données et une gestion efficace des données relatives aux congés.

```

1
2 import Model.Holiday;
3 import Model.Type;
4 import java.sql.*;
5 import java.util.ArrayList;
6 import java.util.List;
7
8 public class HolidayDAOImpl {
9
10     private static connexion conn;
11
12     public HolidayDAOImpl() {
13         conn = new connexion();
14     }
15
16     // Ajouter un cong
17     public void add(Holiday holiday) {
18         String sql = "INSERT INTO holiday (employeeId, type, startdate, enddate)
19 VALUES (?, ?, ?, ?)";
20         try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
21             // Ajouter le cong dans la base de données
22             stmt.setInt(1, holiday.getEmployeeId());
23             stmt.setString(2, holiday.getType().name());
24             stmt.setString(3, holiday.getStartDate());
25             stmt.setString(4, holiday.getEndDate());
26             stmt.executeUpdate();
27
28             // Calculer la durée du cong
29             int duration = calculateHolidayDuration(holiday.getStartDate(), holiday.getEndDate());
30
31             // Mettre à jour la balance
32             updateHolidayBalanceAfterAddition(holiday.getEmployeeId(), duration);
33
34         } catch (SQLException e) {
35             e.printStackTrace();
36         }
37     }
38
39     // Calculer la durée du cong
40     private int calculateHolidayDuration(Date startDate, Date endDate) {
41         return (endDate.getTime() - startDate.getTime()) / (24 * 60 * 60 * 1000);
42     }
43
44     // Mettre à jour la balance
45     private void updateHolidayBalanceAfterAddition(int employeeId, int duration) {
46         // ...
47     }
48 }

```



```

32         } catch (SQLException e) {
33             e.printStackTrace();
34         }
35     }
36
37     // Mettre jour un cong
38     public void update(Holiday holiday) {
39         String sql = "UPDATE holiday SET employeeId = ?, type = ?, startdate = ?,
endddate = ? WHERE id = ?";
40         try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
41             stmt.setInt(1, holiday.getEmployeeId());
42             stmt.setString(2, holiday.getType().name());
43             stmt.setString(3, holiday.getStartDate());
44             stmt.setString(4, holiday.getEndDate());
45             stmt.setInt(5, holiday.getId());
46             stmt.executeUpdate();
47             int duration = calculateHolidayDuration(holiday.getStartDate(), holiday.
getEndDate());
48
49             // Mettre jour la balance
50             updateHolidayBalanceAfterAddition(holiday.getEmployeeId(), duration);
51         } catch (SQLException e) {
52             e.printStackTrace();
53         }
54     }
55
56     // Supprimer un cong
57     public void delete(int id) {
58         String sql = "DELETE FROM holiday WHERE id = ?";
59         try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
60             stmt.setInt(1, id);
61             stmt.executeUpdate();
62         } catch (SQLException e) {
63             e.printStackTrace();
64         }
65     }
66
67     // R cup rer tous les cong s pour un employ
68     public List<Holiday> getByEmployeeId(int employeeId) {
69         List<Holiday> holidays = new ArrayList<>();
70         String sql = "SELECT * FROM holiday WHERE employeeId = ?";
71         try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
72             stmt.setInt(1, employeeId);
73             try (ResultSet rslt = stmt.executeQuery()) {
74                 while (rslt.next()) {
75                     holidays.add(new Holiday(
76                         rslt.getInt("id"),
77                         rslt.getInt("employeeId"),
78                         Type.valueOf(rslt.getString("type")),
79                         rslt.getString("startdate"),
80                         rslt.getString("enddate")
81                     ));
82                 }
83             }
84         } catch (SQLException e) {

```

```
85         e.printStackTrace();
86     }
87     return holidays;
88 }
89
90 // R cup rer un cong par ID
91 public Holiday getById(int id) {
92     Holiday holiday = null;
93     String sql = "SELECT * FROM holiday WHERE id = ?";
94     try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
95         stmt.setInt(1, id);
96         try (ResultSet rslt = stmt.executeQuery()) {
97             if (rslt.next()) {
98                 holiday = new Holiday(
99                     rslt.getInt("id"),
100                    rslt.getInt("employeeId"),
101                    Type.valueOf(rslt.getString("type")),
102                    rslt.getString("startdate"),
103                    rslt.getString("enddate")
104                );
105            }
106        }
107    } catch (SQLException e) {
108        e.printStackTrace();
109    }
110    return holiday;
111 }
112
113 // R cup rer tous les cong s
114 public List<Holiday> getAll() {
115     List<Holiday> holidays = new ArrayList<>();
116     String sql = "SELECT * FROM holiday";
117     try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql);
118         ResultSet rslt = stmt.executeQuery()) {
119         while (rslt.next()) {
120             holidays.add(new Holiday(
121                 rslt.getInt("id"),
122                 rslt.getInt("employeeId"),
123                 Type.valueOf(rslt.getString("type")),
124                 rslt.getString("startdate"),
125                 rslt.getString("enddate")
126             ));
127         }
128     } catch (SQLException e) {
129         e.printStackTrace();
130     }
131     return holidays;
132 }
133
134 // R cup rer le nom de l'employ par son ID
135 public String getNameEmployeeById(int employeeId) {
136     String sql = "SELECT nom FROM Employee WHERE id = ?";
137     try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
138         stmt.setInt(1, employeeId);
139         try (ResultSet rslt = stmt.executeQuery()) {
```

```
140         if (rslt.next()) {
141             return rslt.getString("nom");
142         }
143     }
144     } catch (SQLException e) {
145         e.printStackTrace();
146     }
147     return null;
148 }
149
150 // R cup rer l'ID de l'employ par son nom
151 public int getEmployeeIdByName(String nom) {
152     String sql = "SELECT id FROM Employee WHERE nom = ?";
153     try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
154         stmt.setString(1, nom);
155         try (ResultSet rslt = stmt.executeQuery()) {
156             if (rslt.next()) {
157                 return rslt.getInt("id");
158             }
159         }
160     } catch (SQLException e) {
161         e.printStackTrace();
162     }
163     return -1;
164 }
165
166 public List<String> getEmployees() {
167     return fetchSingleColumn("SELECT nom FROM Employee", "nom");
168 }
169
170 public List<String> getAllEmployeeNames() {
171     return fetchSingleColumn("SELECT nom FROM Employee", "nom");
172 }
173
174 public List<String> fetchSingleColumn(String sql, String columnLabel) {
175     List<String> results = new ArrayList<>();
176     try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql);
177         ResultSet rs = stmt.executeQuery()) {
178         while (rs.next()) {
179             results.add(rs.getString(columnLabel));
180         }
181     } catch (SQLException e) {
182         logError(e, "Error fetching data for column: " + columnLabel);
183     }
184     return
185         results;
186 }
187
188 private void logError(Exception e, String message) {
189     System.err.println(message);
190     e.printStackTrace();
191 }
192
193 // R cup rer la balance de cong pour un employ
194 public int getHolidayBalance(int employeeId) {
```

```

195     int balance = 25;
196     String sql = "SELECT used_balance FROM Employee WHERE id = ?";
197     try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
198         stmt.setInt(1, employeeId);
199         try (ResultSet rs = stmt.executeQuery()) {
200             if (rs.next()) {
201                 balance = rs.getInt("used_balance");
202             }
203         }
204     } catch (SQLException e) {
205         e.printStackTrace();
206     }
207     return balance;
208 }
209
210 // Mettre jour la balance de cong apr s l'ajout d'un cong
211 private void updateHolidayBalanceAfterAddition(int employeeId, int duration) {
212     int currentBalance = getHolidayBalance(employeeId);
213     int newBalance = currentBalance - duration;
214     if (newBalance < 0) {
215         throw new IllegalArgumentException("Insufficient holiday balance.");
216     }
217     updateHolidayBalance(employeeId, newBalance);
218 }
219
220 // Mettre jour la balance de cong dans la base de donn es
221 public void updateHolidayBalance(int employeeId, int newBalance) {
222     String sql = "UPDATE Employee SET used_balance = ? WHERE id = ?";
223     try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
224         stmt.setInt(1, newBalance);
225         stmt.setInt(2, employeeId);
226         stmt.executeUpdate();
227     } catch (SQLException e) {
228         e.printStackTrace();
229     }
230 }
231
232 // Calculer la dur e d'un cong en jours
233 private int calculateHolidayDuration(String startDate, String endDate) {
234     try {
235         java.util.Date start = new java.text.SimpleDateFormat("yyyy-MM-dd").parse(
236             startDate);
237         java.util.Date end = new java.text.SimpleDateFormat("yyyy-MM-dd").parse(
238             endDate);
239         long diff = end.getTime() - start.getTime();
240         return (int) (diff / (1000 * 60 * 60 * 24)) + 1;
241     } catch (Exception e) {
242         e.printStackTrace();
243         return 0;
244     }
245 }
246
247 public List<Holiday> getHolidaysByEmployeeId(int employeeId) {
248     List<Holiday> holidays = new ArrayList<>();
249     String sql = "SELECT * FROM holiday WHERE employeeId = ?";
250     try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {

```

```
248         stmt.setInt(1, employeeId); // Ajouter l'identifiant de l'employé dans la
requête
249         try (ResultSet rs = stmt.executeQuery()) {
250             while (rs.next()) {
251                 holidays.add(new Holiday(
252                     rs.getInt("id"),
253                     rs.getInt("employeeId"),
254                     Type.valueOf(rs.getString("type")), // Convertir en
Type (enum)
255                     rs.getString("startdate"),
256                     rs.getString("enddate")
257                 ));
258             }
259         }
260     } catch (SQLException e) {
261         e.printStackTrace();
262     }
263     return holidays;
264 }
265 }
```

## 4 Couche Controller

- **HolidayControll**

La classe `HolidayController` joue un rôle central dans l'architecture MVC (Modèle-Vue-Contrôleur) de l'application de gestion des congés, agissant comme un intermédiaire entre la vue (interface utilisateur) et le modèle (gestion des données). Elle initialise les données de la vue, telles que la liste des employés, et ajoute des écouteurs d'événements pour gérer les actions des boutons (ajouter, supprimer, modifier, afficher). Lorsqu'un utilisateur sélectionne une ligne dans la table des congés, les champs de saisie sont automatiquement remplis avec les données correspondantes.

Le contrôleur permet d'ajouter, modifier ou supprimer un congé, après avoir validé et formaté les données, tout en affichant des messages de réussite ou d'échec selon les actions réalisées. Il valide également que tous les champs sont remplis, que les dates respectent le format attendu (yyyy-MM-dd), et gère les erreurs en affichant des messages clairs en cas de problèmes.

Cette classe garantit une séparation claire des responsabilités entre l'interface utilisateur et la logique métier, assurant ainsi la maintenabilité du code et une expérience utilisateur fluide.

```

267
268 import Model.*;
269 import View.HolidayView;
270
271 import java.text.ParseException;
272 import java.text.SimpleDateFormat;
273 import java.util.List;
274
275 public class HolidayController {
276     private HolidayView view;
277     private HolidayModel model;
278
279     public HolidayController(HolidayView view, HolidayModel model) {
280         this.view = view;
281         this.model = model;
282
283         // Récupérer les employés et remplir la combobox
284         List<String> employees = model.getEmployes();
285         view.setEmployes(employees);
286
287         // Ajouter les écouteurs pour les boutons
288         this.view.getAjouterButton().addActionListener(e -> addHoliday());
289         this.view.getSupprimerButton().addActionListener(e -> deleteHoliday());
290         this.view.getModifierButton().addActionListener(e -> updateHoliday());
291         this.view.getAfficherButton().addActionListener(e -> showHolidays());
292
293         // Ajouter un écouteur pour remplir les champs lors de la sélection dans la
294         // table
295         this.view.getTable().getSelectionModel().addListSelectionListener(e -> {
296             int selectedRow = view.getTable().getSelectedRow();
297             if (selectedRow != -1) {
298                 view.setEmploye(view.getTable().getValueAt(selectedRow, 1).toString());
299             }
300             view.getTypeComboBox().setSelectedItem(Type.valueOf(view.getTable().

```

```

getValueAt(selectedRow, 2).toString()));
299         view.getStartDateField().setText(view.getTable().getValueAt(
selectedRow, 3).toString());
300         view.getEndDateField().setText(view.getTable().getValueAt(selectedRow
, 4).toString());
301     }
302     });
303 }
304
305 // Ajouter un cong
306 private void addHoliday() {
307     try {
308         if (validateInputs()) {
309             String employe = view.getEmploye().trim();
310             Type type = view.getType();
311             String startDateFormatted = formatDate(view.getStartDate().trim());
312             String endDateFormatted = formatDate(view.getEndDate().trim());
313
314             boolean ajoutReussi = model.add(employe, type, startDateFormatted,
endDateFormatted);
315             if (ajoutReussi) {
316                 view.afficherMessageSucces("Cong ajout avec succ s.");
317                 showHolidays();
318             } else {
319                 view.afficherMessageErreur("chec de l'ajout du cong .");
320             }
321         }
322     } catch (Exception e) {
323         view.afficherMessageErreur("Une erreur inattendue est survenue : " + e.
getMessage());
324     }
325 }
326 // Mettre jour un cong
327 private void updateHoliday() {
328     try {
329         int selectedRow = view.getTable().getSelectedRow();
330         if (selectedRow == -1) {
331             view.afficherMessageErreur("Veuillez slectionner un cong
mettre jour.");
332             return;
333         }
334
335         if (validateInputs()) {
336             int id = (int) view.getTable().getValueAt(selectedRow, 0);
337             String employe = view.getEmploye().trim();
338             Type type = view.getType();
339             String startDateFormatted = formatDate(view.getStartDate().trim());
340             String endDateFormatted = formatDate(view.getEndDate().trim());
341
342             boolean miseAJourReussie = model.update(id, employe, type,
startDateFormatted, endDateFormatted);
343             if (miseAJourReussie) {
344                 view.afficherMessageSucces("Cong mis jour avec succ s.");
345                 showHolidays();
346             } else {

```

```

347         view.afficherMessageErreur(" chec de la mise jour du cong
        .");
348     }
349 }
350 } catch (Exception e) {
351     view.afficherMessageErreur("Une erreur inattendue est survenue : " + e.
    getMessage());
352 }
353 }
354 // Supprimer un cong
355 private void deleteHoliday() {
356     int selectedRow = view.getTable().getSelectedRow();
357     if (selectedRow == -1) {
358         view.afficherMessageErreur("Veuillez slectionner un cong supprimer
        .");
359         return;
360     }
361     int id = (int) view.getTable().getValueAt(selectedRow, 0);
362
363     boolean suppressionReussie = model.delete(id);
364     if (suppressionReussie) {
365         view.afficherMessageSucces("Cong supprim avec succ s.");
366         showHolidays();
367     } else {
368         view.afficherMessageErreur(" chec de la suppression du cong .");
369     }
370 }
371
372 // Afficher tous les cong s
373 private void showHolidays() {
374     Object[][] holidays = model.getAllHolidaysAsTableData();
375     view.getModel().setRowCount(0); // Vider la table avant d'ajouter les
    nouvelles donn es
376     for (Object[] holiday : holidays) {
377         view.getModel().addRow(holiday);
378     }
379 }
380
381 // Valider les entr es des utilisateurs
382 private boolean validateInputs() {
383     String employe = view.getEmploye().trim();
384     Type type = view.getType();
385     String startDate = view.getStartDate().trim();
386     String endDate = view.getEndDate().trim();
387
388     if (employe.isEmpty() || type == null || startDate.isEmpty() || endDate.
    isEmpty()) {
389         view.afficherMessageErreur("Tous les champs doivent tre remplis.");
390         return false;
391     }
392
393     if (!isValidDate(startDate) || !isValidDate(endDate)) {
394         view.afficherMessageErreur("Les dates doivent tre au format AAAA-MM-JJ
        .");
395         return false;

```



```

396     }
397
398     return true;
399 }
400
401 // V rifier si une date est valide
402 private boolean isValidDate(String date) {
403     SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
404     sdf.setLenient(false);
405     try {
406         sdf.parse(date);
407         return true;
408     } catch (ParseException e) {
409         return false;
410     }
411 }
412
413 // Formater la date au format attendu
414 private String formatDate(String date) {
415     try {
416         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
417         return sdf.format(sdf.parse(date));
418     } catch (ParseException e) {
419         view.afficherMessageErreur("Format de date invalide.");
420         return null;
421     }
422 }
423 }

```

## 5 Couche View

### • HolidayView

La classe `HolidayView` est une interface graphique (GUI) en Java Swing pour la gestion des congés des employés. Elle hérite de `JPanel` et organise l'interface en plusieurs panneaux (`pan1`, `pan2`, `pan4`) pour afficher les champs de formulaire, les boutons et la table présentant les enregistrements des congés.

Cette classe contient plusieurs composants tels que des boutons pour ajouter, modifier, supprimer et afficher les congés, ainsi que des champs de texte pour saisir les dates de début et de fin des congés.

Elle utilise également des `JComboBox` pour sélectionner l'employé et le type de congé. La `JTable` sert à afficher les données des congés. Plusieurs méthodes `getter` et `setter` permettent d'interagir avec les composants de l'interface.

La classe inclut aussi des méthodes pour afficher des messages d'erreur et de succès à l'utilisateur via `JOptionPane`. De plus, la méthode `setEmployes` permet de remplir dynamiquement la liste des employés dans la combo box.

#### Composants principaux :

- `JButton` : pour ajouter, modifier, supprimer et afficher les congés.
- `TextField` : pour saisir les dates de début et de fin des congés.

- JComboBox : pour sélectionner l'employé et le type de congé.
- JTable : pour afficher les enregistrements des congés.

```

1 package View;
2
3 import javax.swing.*;
4 import javax.swing.table.DefaultTableModel;
5 import java.awt.*;
6 import java.util.List;
7 import Model.Type;
8
9 public class HolidayView extends JPanel {
10     // D clARATION des panels
11     private JPanel pan1 = new JPanel();
12     private JPanel pan2 = new JPanel();
13     private JPanel pan4 = new JPanel();
14
15     // Boutons
16     private JButton ajouter = new JButton("Ajouter ");
17     private JButton modifier = new JButton("Modifier "); // Nouveau bouton pour
        modifier
18     private JButton supprimer = new JButton("Supprimer ");
19     private JButton afficher = new JButton("Afficher");
20
21     // Labels
22     private JLabel employeLabel = new JLabel("Employ :");
23     private JLabel typeLabel = new JLabel("Type de Cong :");
24     private JLabel startDateLabel = new JLabel("Date de D but:");
25     private JLabel endDateLabel = new JLabel("Date de Fin:");
26
27     // Champs de texte
28     private JTextField startDateField = new JTextField(20);
29     private JTextField endDateField = new JTextField(20);
30
31     // Combobox pour les employ s et le type de cong
32     private JComboBox<String> employeComboBox = new JComboBox<>();
33     private JComboBox<Type> typeComboBox = new JComboBox<>(Type.values());
34
35     // Table pour afficher les cong s
36     private JTable table;
37     private DefaultTableModel model;
38     private JScrollPane scrollPane;
39
40     public HolidayView() {
41         // Configuration du panel principal
42         pan1.setLayout(new BorderLayout());
43         pan1.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); //
        Ajout d'un espace int rieur
44
45         // Configuration du panel contenant les champs
46         pan2.setLayout(new GridLayout(5, 2, 2, 2)); // R duction de l'
        espacement horizontal et vertical
47         pan2.add(employeLabel);
48         pan2.add(employeComboBox);
49         pan2.add(typeLabel);
50         pan2.add(typeComboBox);

```

```
51         pan2.add(startDateLabel);
52         pan2.add(startDateField);
53         pan2.add(endDateLabel);
54         pan2.add(endDateField);
55
56         // Configuration du panel pour les boutons
57         pan4.setLayout(new GridLayout(1, 4, 8, 8)); // Ajustement de l'
espacement pour 4 boutons
58         pan4.add(ajouter);
59         pan4.add(modifier);
60         pan4.add(supprimer);
61         pan4.add(afficher);
62
63         // Table pour afficher les cong s
64         String[] columnNames = {"Id", "Employ ", "Type de Cong ", "Date de
D but ", "Date de Fin"};
65         model = new DefaultTableModel(columnNames, 0);
66         table = new JTable(model);
67         scrollPane = new JScrollPane(table);
68
69         // Ajout des panneaux au panel principal
70         pan1.add(pan2, BorderLayout.NORTH);
71         pan1.add(scrollPane, BorderLayout.CENTER);
72         pan1.add(pan4, BorderLayout.SOUTH);
73
74         // Ajout du panel principal      la vue
75         add(pan1);
76
77         // Rendre la fen tre visible
78         setVisible(true);
79     }
80
81     // Getters et setters pour les boutons
82     public JButton getAjouterButton() {
83         return ajouter;
84     }
85
86     public JButton getModifierButton() {
87         return modifier;
88     }
89
90     public JButton getSupprimerButton() {
91         return supprimer;
92     }
93
94     public JButton getAfficherButton() {
95         return afficher;
96     }
97
98     // Getters et setters pour les champs de texte
99     public JTextField getStartDateField() {
100         return startDateField;
101     }
102
103     public JTextField getEndDateField() {
```

```
104         return endDateField;
105     }
106
107     // Getters et setters pour les ComboBox
108     public JComboBox<String> getEmployeComboBox() {
109         return employeComboBox;
110     }
111
112     public JComboBox<Type> getTypeComboBox() {
113         return typeComboBox;
114     }
115
116     // Getters pour la table et son mod le
117     public JTable getTable() {
118         return table;
119     }
120
121     public DefaultTableModel getModel() {
122         return model;
123     }
124
125     // Mthodes pour afficher les messages d'erreur et de succ s
126     public void afficherMessageErreur(String message) {
127         JOptionPane.showMessageDialog(this, message, "Erreur", JOptionPane.
ERROR_MESSAGE);
128     }
129
130     public void afficherMessageSucces(String message) {
131         JOptionPane.showMessageDialog(this, message, "Succ s", JOptionPane.
INFORMATION_MESSAGE);
132     }
133
134     // Mthode pour d finir les employ s dans le ComboBox
135     public void setEmployes(List<String> employes) {
136         employeComboBox.setModel(new DefaultComboBoxModel<>(employes.toArray(
new String[0])));
137     }
138
139     // Mthode pour r cup rer l'employ s lectionn
140     public String getEmploye() {
141         return (String) employeComboBox.getSelectedItem();
142     }
143
144     // Mthode pour d finir l'employ s lectionn
145     public void setEmploye(String employe) {
146         employeComboBox.setSelectedItem(employe);
147     }
148
149     // Mthode pour r cup rer le type de cong s lectionn
150     public Type getType() {
151         return (Type) typeComboBox.getSelectedItem();
152     }
153
154     // Mthode pour d finir le type de cong s lectionn
155     public void setType(Type type) {
```

```
156         typeComboBox.setSelectedItem(type);
157     }
158
159     // Mthodes pour r cup rer et d finir les dates
160     public String getStartDate() {
161         return startDateField.getText();
162     }
163
164     public void setStartDate(String startDate) {
165         startDateField.setText(startDate);
166     }
167
168     public String getEndDate() {
169         return endDateField.getText();
170     }
171
172     public void setEndDate(String endDate) {
173         endDateField.setText(endDate);
174     }
175 }
```

- **PanelsView**

La classe `PanelsView` représente une interface graphique qui affiche une fenêtre contenant deux onglets. Le premier onglet est destiné à la gestion des employés et le second à la gestion des congés. Elle utilise un composant `JTabbedPane` pour organiser et afficher ces vues. Les vues `EmployeView` et `HolidayView` sont ajoutées respectivement sous chaque onglet, permettant une navigation fluide entre les deux sections.

# Résultats

Admin Dashboard - Gestion des Congés et des Employés

Gestion des Employés Gestion des Congés

Employé: hananee

Type de Congé: CONGE\_PAYE

Date de Début: 2024-12-27

Date de Fin: 2024-12-30

Id	Employé	Type de Congé	Date de Début	Date de Fin
----	---------	---------------	---------------	-------------

Succès

Congé ajouté avec succès.

OK

Ajouter Modifier Supprimer Afficher

FIGURE 2.1 – Resultat ajout de conge

## 1 Button ajouter

Lorsque le bouton "Ajouter" est cliqué, les informations de congé sont récupérées et ajoutées à la base de données si les dates sont valides. Si la date de début et de fin respectent les conditions, le congé est enregistré, et le solde des jours de congé de l'employé est mis à jour en conséquence.

+	h	up	h@gm...	06754...	8790.0	ADMIN	Ingeni...	+
9	hanan...	hhdHJ	h@gm...	06754...	8790.0	ADMIN	Ingeni...	17


FIGURE 2.2 – change de balance

Lorsque le bouton "Ajouter" est cliqué, le système vérifie d'abord si le nombre de jours de congé demandé dépasse le solde disponible de l'employé. Si c'est le cas, un message d'erreur est affiché.

Employé:
Type de Congé:
Date de Début:
Date de Fin:

Id	Employé	Type de Congé	Date de Début	Date de Fin
4	hananee	CONGE_PAYE	2024-12-27	2024-12-30

Erreur


Échec de l'ajout du congé.

OK

FIGURE 2.3 – Echoue de l'ajout de conge

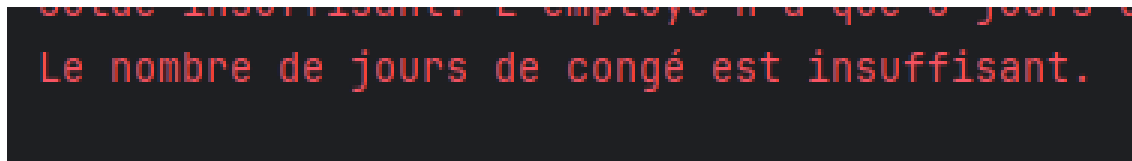


FIGURE 2.4 – la cause de l'échoue de l'ajout

## 2 Button Modifier

Lorsqu'un congé déjà ajouté est modifié, le système recalculera le solde de congé de l'employé en fonction des nouvelles dates. Cela permet de mettre à jour le nombre de jours de congé restants, prenant en compte la durée modifiée du congé.

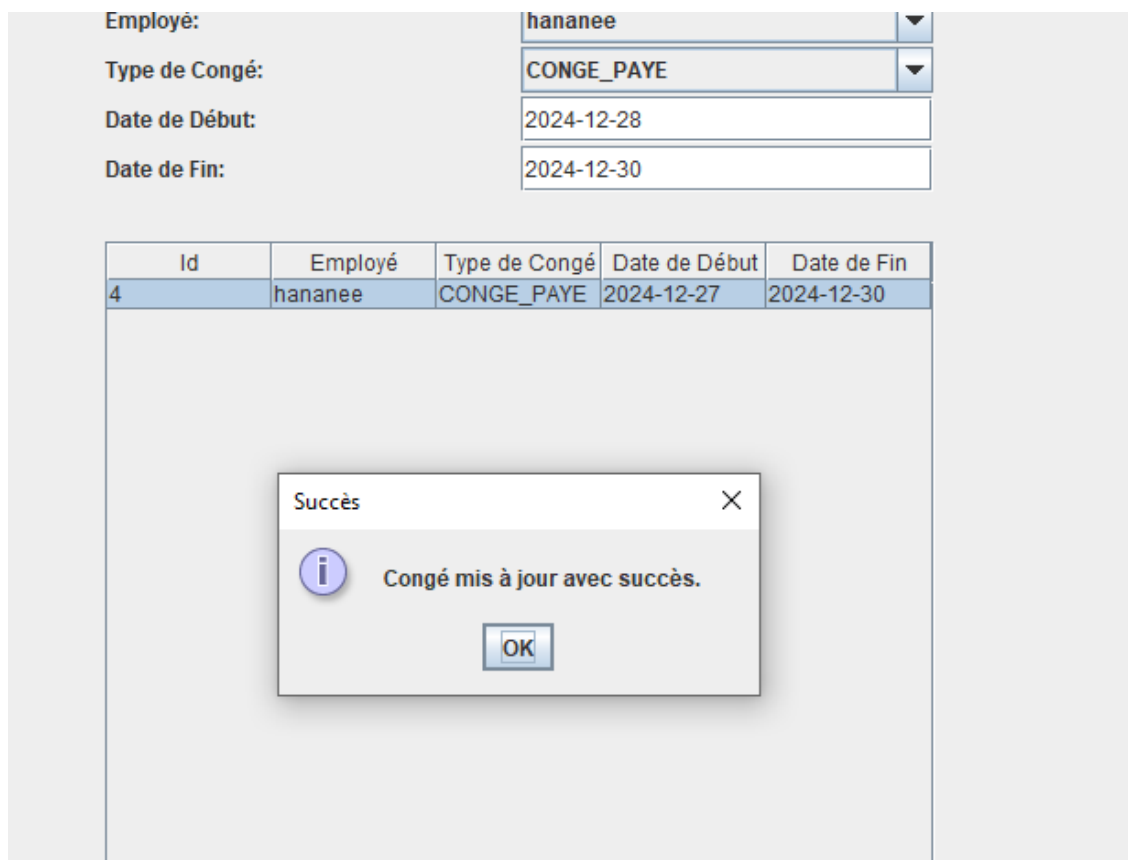


FIGURE 2.5 – Resultat du Modification

9	hanan...	hhdHJ	h@gm...	06754...	8790.0	ADMIN	Ingeni...	14
---	----------	-------	---------	----------	--------	-------	-----------	----

FIGURE 2.6 – Resultat du Modification dans Employe



## Conclusion

En conclusion, ce TP a permis de développer une application de gestion des employés et des congés en adoptant l'architecture MVC. Cette approche a facilité la séparation des responsabilités entre la logique métier, l'interface utilisateur et le traitement des données, garantissant ainsi une application modulaire, évolutive et facile à maintenir. L'application permet de gérer efficacement les employés et leurs congés, avec des fonctionnalités telles que l'ajout, la modification, la suppression et l'affichage des employés et des congés. De plus, le recalcul du solde de congé des employés lors des modifications garantit une gestion précise et dynamique des ressources humaines. L'intégration de ces fonctionnalités a renforcé notre maîtrise des concepts de programmation orientée objet et de gestion des interfaces graphiques en Java. Ce travail pratique met en évidence l'importance d'une organisation claire et structurée du code pour créer des applications robustes et performantes.