

**École Supérieure de Technologie Safi  
Génie Informatique**

## **Compte rendu**

---

### **TP 2**

### **Gestion des Employés et des Congés avec Architecture MVC et DAO**

**Java Avancée**

**Élaboré par :**

Khadija Kaydi

2024/2025

# Table des matières

<b>1</b>	<b>Contexte du TP2</b>	<b>3</b>
<b>2</b>	<b>Base de Données : Table conge</b>	<b>4</b>
2.1	Structure de la Table conge . . . . .	4
2.2	Relation avec la Table employe . . . . .	4
<b>3</b>	<b>Gestion des Congés dans le Projet Java</b>	<b>5</b>
3.1	Package Model . . . . .	5
3.1.1	Classe Holiday . . . . .	5
3.1.2	Class TypeH : Énumération des types de congés . . . . .	7
3.1.3	La classe HolidayModel . . . . .	7
3.2	Package DAO . . . . .	11
3.2.1	Classe Connexion . . . . .	11
3.2.2	Interface GenericDAOI . . . . .	12
3.2.3	Classe HolidayDAOimplement . . . . .	13
3.3	Package View . . . . .	18
3.3.1	Classe HolidayView . . . . .	18
3.3.2	Classe GestionEmployesConges . . . . .	21
3.4	Package Controller . . . . .	22
3.4.1	Classe HolidayController . . . . .	22
3.5	Package Main . . . . .	28
<b>4</b>	<b>Scripts d'Exécution du TP</b>	<b>29</b>
4.1	Ajouter Conge . . . . .	29
4.2	Modifier Conge . . . . .	30
4.3	Supprimer Conge . . . . .	31
4.4	Afficher Congé . . . . .	31
4.5	Scripts de l'Architecture MVC et DAO . . . . .	32
<b>5</b>	<b>Conclusion</b>	<b>33</b>

# 1 Contexte du TP2

Dans ce projet, nous avons conçu une interface pour la gestion des employés et des congés. Le TP2 étend le TP1 en ajoutant une fonctionnalité pour gérer les congés des employés.

L'interface se divise en deux parties :

- **Partie 1** : Gestion des employés (déjà traitée dans le TP1).
- **Partie 2** : Gestion des congés, permettant d'enregistrer, consulter et approuver les demandes de congés.

ID	Nom	Prenom	Telephone	Email	Salaire	Role	Poste	Solde
25	new	new	gv@gmail.com	1234567891	545.0	DIRECTION_...	DEVEOPPEU...	0.0
26	test	test	test@gmail.c...	1234567896	123.0	DIRECTION_...	DEVEOPPEU...	0.0
28	new	test	gv@gmail.com	1234569878	545.0	DIRECTION_...	DEVEOPPEU...	6.0
29	kaydi	kaydi	test@gmail.c...	1234567896	123.0	DIRECTION_...	DEVEOPPEU...	4.0
30	farah	test	gv@gmail.com	1234569878	545.0	DIRECTION_...	DEVEOPPEU...	11.0

FIGURE 1 – Fenêtre de gestion des employés et congés

## 2 Base de Données : Table conge

La table `conge` stocke les informations relatives aux congés des employés.

### 2.1 Structure de la Table conge

TABLE 1 – Structure de la table `conge`

#	Colonne	Type	Description
1	<code>id</code>	<code>int(11)</code>	Clé primaire
2	<code>date_debut</code>	<code>date</code>	Date de début
3	<code>date_fin</code>	<code>date</code>	Date de fin
4	<code>employee_id</code>	<code>int(11)</code>	Clé étrangère vers <code>employee</code>
5	<code>type</code>	<code>varchar(20)</code>	Type de congé
6	<code>nom_employe</code>	<code>varchar(50)</code>	Nom de l'employé (optionnel)

### 2.2 Relation avec la Table employee

La colonne `employee_id` référence un employé de la table `employee`, établissant ainsi une relation entre les congés et les employés.


















#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 <code>id</code> 	<code>int(11)</code>			Non	<i>Aucun(e)</i>		<code>AUTO_INCREMENT</code>	 Modifier  Supprimer  Plus
<input type="checkbox"/>	2 <code>date_debut</code>	<code>date</code>			Non	<i>Aucun(e)</i>			 Modifier  Supprimer  Plus
<input type="checkbox"/>	3 <code>date_fin</code>	<code>date</code>			Non	<i>Aucun(e)</i>			 Modifier  Supprimer  Plus
<input type="checkbox"/>	4 <code>Type</code>	<code>varchar(20)</code>	<code>utf8mb4_general_ci</code>		Oui	<code>NULL</code>			 Modifier  Supprimer  Plus
<input type="checkbox"/>	5 <code>employee_id</code> 	<code>int(11)</code>			Oui	<code>NULL</code>			 Modifier  Supprimer  Plus

FIGURE 2 – Script de la table `conge`

## 3 Gestion des Congés dans le Projet Java

Le projet Java, structuré selon l'architecture **MVC**, gère à la fois les employés et leurs congés, permettant de lier les informations dans une interface centralisée.

- **Model** : Entités du projet.
- **View** : Interface utilisateur.
- **Controller** : Gère les actions utilisateur et l'interaction avec le modèle.
- **DAO** : Accès aux données et gestion des requêtes.

### 3.1 Package Model

Le package `Model` contient les entités principales de la gestion des congés, notamment la classe `Holiday`.

#### 3.1.1 Classe Holiday

La classe `Holiday` représente un congé avec ces attributs, en plus les **getters**, **setters** et un constructeur.

```
1 package Model;
2
3 import java.time.LocalDate;
4
5 public class Holiday {
6
7     // Attributs privés de la classe Holiday
8     private int id; // Identifiant unique du congé
9     private LocalDate dateDebut; // Date de début du congé
10    private LocalDate dateFin; // Date de fin du congé
11    private TypeH type; // Type de congé (ex: maladie, annuel)
12    private String nom_employe; // Nom de l'employé
13    private int id_employe; // Identifiant de l'employé
14
15    // Constructeur sans identifiant (pour ajout d'un nouveau congé)
16    public Holiday(LocalDate dateDebut, LocalDate dateFin,
17        TypeH type, int id_employe, String nom) {
18        this.dateDebut = dateDebut;
19        this.dateFin = dateFin;
20        this.type = type;
21        this.id_employe = id_employe;
22        this.nom_employe = nom;
23    }
24
25    // Constructeur avec identifiant (pour récupérer un congé existant)
26    public Holiday(int id, LocalDate dateDebut, LocalDate dateFin,
27        TypeH type, int id_employe, String nom) {
28        this.id = id;
29        this.dateDebut = dateDebut;
30        this.dateFin = dateFin;
31        this.type = type;
32        this.id_employe = id_employe;
33        this.nom_employe = nom;
```

```
34     }
35
36     // Getter pour l'identifiant
37     public int getId() {
38         return id;
39     }
40
41     // Setter pour l'identifiant
42     public void setId(int id) {
43         this.id = id;
44     }
45
46     // Getter pour la date de début
47     public LocalDate getDateDebut() {
48         return dateDebut;
49     }
50
51     // Setter pour la date de début
52     public void setDateDebut(LocalDate dateDebut) {
53         this.dateDebut = dateDebut;
54     }
55
56     // Getter pour la date de fin
57     public LocalDate getDateFin() {
58         return dateFin;
59     }
60
61     // Setter pour la date de fin
62     public void setDateFin(LocalDate dateFin) {
63         this.dateFin = dateFin;
64     }
65
66     // Getter pour le type de congé
67     public TypeH getType() {
68         return type;
69     }
70
71     // Getter pour l'ID de l'employé
72     public int getId_employe() {
73         return id_employe;
74     }
75
76     // Setter pour l'ID de l'employé
77     public void setId_employe(int id_employe) {
78         this.id_employe = id_employe;
79     }
80
81     // Getter pour le nom de l'employé
82     public String getNom_employe() {
```

```

83     return nom_employe;
84 }
85
86 // Setter pour le nom de l'employé
87 public void setNom_employe(String nom_employe) {
88     this.nom_employe = nom_employe;
89 }
90 }

```

### 3.1.2 Class TypeH : Enumération des types de congés

La classe TypeH définit les types de congés disponibles.  
Voici le code source de la classe d'énumération :

```

1 package Model;
2
3 public enum TypeH {
4     Conge_payee,
5     conge_non_payee,
6     conge_malade;
7 }

```

### 3.1.3 La classe HolidayModel

La classe HolidayModel est responsable de la logique métier de la gestion des congés.  
Voici le code source de la classe HolidayModel :

```

1 package Model;
2
3 import java.sql.Date;
4 import java.time.LocalDate;
5 import java.time.temporal.ChronoUnit;
6 import javax.swing.JOptionPane;
7 import DAO.EmployeeDAOImplement;
8 import DAO.HolidayDAOImplement;
9
10 public class HolidayModel {
11     private HolidayDAOImplement dao; // Déclaration de l'instance de DAO
12
13     // Constructeur pour initialiser le modèle avec l'implémentation de
14     ↪ HolidayDAO
15     public HolidayModel(HolidayDAOImplement dao) {
16         this.dao = dao; // Initialisation de l'instance de HolidayDAO
17     }
18
19     // Méthode pour ajouter un congé
20     public void addHoliday(LocalDate dateDebut, LocalDate dateFin, TypeH
21     ↪ type, int id_employe) {

```

```

20     try {
21         // Vérification si la date de fin est après la date de début
22         if (ChronoUnit.DAYS.between(dateDebut, dateFin) < 0) {
23             JOptionPane.showMessageDialog(null, "Erreur : la date de fin
24                 ↳ doit être après la date de début.");
25             return;
26         }
27
28         long differenceInDays = ChronoUnit.DAYS.between(dateDebut,
29             ↳ dateFin); // Calcul du nombre de jours entre les deux
30             ↳ dates
31
32         // Vérification de l'intervalle existant pour éviter le
33             ↳ chevauchement des congés
34         int overlapDays = dao.getOverlapDays(id_employe, dateDebut,
35             ↳ dateFin);
36
37         // Si l'intervalle de dates existe déjà, on l'avertit à
38             ↳ l'utilisateur
39         if (overlapDays == 0) {
40             JOptionPane.showMessageDialog(null, "L'intervalle de congé
41                 ↳ existe déjà !");
42             return;
43         }
44
45         // Si l'intervalle n'est pas trouvé et le solde est suffisant,
46             ↳ on ajoute le congé
47         if (overlapDays != -1) {
48             if (dao.soldeActuel(id_employe) < overlapDays) {
49                 JOptionPane.showMessageDialog(null, "Le nombre de jours
50                     ↳ de congé est insuffisant !");
51                 return;
52             }
53
54             Holiday holiday = new Holiday(dateDebut, dateFin, type,
55                 ↳ id_employe, null);
56             dao.addGeneric(holiday); // Ajouter le congé
57             dao.ModifierSolde(id_employe, overlapDays); // Modifier le
58                 ↳ solde d'employé
59         } else {
60             // Si l'intervalle n'est pas trouvé et que le solde est
61                 ↳ suffisant, on ajoute un congé classique
62             if (dao.soldeActuel(id_employe) < differenceInDays) {
63                 JOptionPane.showMessageDialog(null, "Le nombre de jours
64                     ↳ de congé est insuffisant !");
65                 return;
66             }
67
68             Holiday holiday = new Holiday(dateDebut, dateFin, type,
69                 ↳ id_employe, null);

```



```

55         dao.addGeneric(holiday); // Ajouter le congé
56         dao.ModifierSolde(id_employe, differenceInDays); //
           ↳ Modifier le solde d'employé
57     }
58
59     } catch (Exception e) {
60         JOptionPane.showMessageDialog(null, "Erreur inattendue lors de
           ↳ l'ajout du congé.");
61         e.printStackTrace(); // Afficher l'erreur dans la console
62     }
63 }
64
65 // Méthode pour modifier un congé
66 public void modifyHoliday(int id, LocalDate dateDebut, LocalDate
           ↳ dateFin, TypeH type, int id_employe) {
67     Holiday holiday = new Holiday(id, dateDebut, dateFin, type,
           ↳ id_employe, null);
68
69     // Vérifier si le congé existe
70     if (!dao.allHoliday(id)) {
71         JOptionPane.showMessageDialog(null, "Le congé n'existe pas");
72         return;
73     }
74
75     long differenceInDays = ChronoUnit.DAYS.between(dateDebut, dateFin)
           ↳ + 1; // Calcul du nombre de jours total
76
77     // Vérification du solde d'employé avant de modifier le congé
78     if (dao.soldeActuel(id_employe) < differenceInDays) {
79         JOptionPane.showMessageDialog(null, "Le nombre de jours de congé
           ↳ est insuffisant !");
80         return;
81     }
82
83     // Vérification si l'intervalle existe déjà
84     int overlapDays = dao.getOverlapDays(id_employe, dateDebut,
           ↳ dateFin);
85     if (overlapDays == 0) {
86         JOptionPane.showMessageDialog(null, "L'intervalle de congé
           ↳ existe déjà !");
87         return;
88     }
89
90     // Modification du congé si aucune erreur de chevauchement et
           ↳ solde suffisant
91     if (overlapDays != -1) {
92         if (dao.soldeActuel(id_employe) < overlapDays) {
93             JOptionPane.showMessageDialog(null, "Le nombre de jours de
           ↳ congé est insuffisant !");

```

```

94         return;
95     }
96     dao.modifyGeneric(holiday); // Modifier le congé
97     dao.ModifierSolde(id_employe, overlapDays); // Mettre à jour
98     ↪ le solde
99 } else {
100     dao.modifyGeneric(holiday); // Modifier le congé
101     dao.ModifierSolde(id_employe, differenceInDays); // Mettre à
102     ↪ jour le solde
103 }
104 }
105
106 // Méthode pour supprimer un congé
107 public void deleteHoliday(int id) {
108     dao.deleteGeneric(id); // Suppression du congé
109 }
110
111 // Méthode pour récupérer tous les congés sous forme d'un tableau
112 ↪ d'objets
113 public Object[][] getAllHolidays() {
114     List<Object[]> holidays = dao.getAllGeneric(); // Récupérer tous
115     ↪ les congés
116     Object[][] holidayData = new Object[holidays.size()][5]; //
117     ↪ Initialiser le tableau de données
118
119     // Remplir le tableau avec les informations des congés
120     for (int i = 0; i < holidays.size(); i++) {
121         Object[] row = holidays.get(i);
122         holidayData[i][0] = row[0];
123         holidayData[i][1] = row[1];
124         holidayData[i][2] = row[2];
125         holidayData[i][3] = row[3];
126         holidayData[i][4] = row[5];
127     }
128
129     return holidayData; // Retourner le tableau de congés
130 }
131
132 // Méthode pour récupérer les IDs des employés
133 public List<String> getIds() {
134     return dao.FindById(); // Récupérer tous les IDs des employés
135 }
136
137 // Méthode pour calculer la différence en jours entre deux dates
138 public int differentInDays(LocalDate dateFin, LocalDate dateDebut) {
139     return (int) ChronoUnit.DAYS.between(dateDebut, dateFin); //
140     ↪ Calculer la différence en jours
141 }

```

```

137 // Méthode pour vérifier le chevauchement des dates
138 public int getOverlapDays(int employeeId, LocalDate newStartDate,
    ↪ LocalDate newEndDate) {
139     return dao.getOverlapDays(employeeId, newStartDate, newEndDate); //
    ↪ Vérifier le chevauchement avec les congés existants
140 }
141 }

```

## 3.2 Package DAO

Le package DAO (Data Access Object) est responsable de l'accès aux données dans l'application. Il contient les classes et interfaces nécessaires pour effectuer des opérations CRUD sur les entités comme `Employees` et `Holiday`.

### 3.2.1 Classe Connexion

La classe `Connexion` est partagée entre les entités `Employees` et `Holiday`. Elle est utilisée pour établir une connexion à la base de données.

```

1 package DAO;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 /**
8  * Classe responsable de l'établissement de la connexion à la base de
    ↪ données.
9  */
10 public class Connexion {
11
12     private static final String URL = "jdbc:mysql://localhost:3306/ma_base";
13     private static final String USER = "root";
14     private static final String PASSWORD = "password";
15     private static Connection connection;
16
17     /**
18      * Méthode pour obtenir une instance unique de la connexion
    ↪ (Singleton).
19      * @return Une connexion à la base de données.
20      */
21     public static Connection getConnection() {
22         if (connection == null) {
23             try {
24                 connection = DriverManager.getConnection(URL, USER,
    ↪ PASSWORD);
25             } catch (SQLException e) {
26                 e.printStackTrace();
27             }
28         }

```

```

29         return connection;
30     }
31 }

```

### 3.2.2 Interface GenericDAOI

L'interface `GenericDAOI` est une interface générique qui définit les méthodes nécessaires pour effectuer des opérations CRUD (*Create, Read, Update, Delete*). Elle est conçue pour être flexible et applicable à plusieurs entités.

```

1  package DAO;
2
3  import java.util.List;
4
5  /**
6   * Interface générique pour les opérations CRUD.
7   * @param <T> Type générique représentant une entité (ex: Employees,
8   *   ↪ Holiday).
9   */
10 public interface GenericDAOI<T> {
11
12     /**
13      * Ajoute un objet de type générique à la base de données.
14      * @param emp L'objet à ajouter.
15      */
16     public void addGeneric(T emp);
17
18     /**
19      * Modifie un objet existant dans la base de données.
20      * @param emp L'objet modifié.
21      */
22     public void modifyGeneric(T emp);
23
24     /**
25      * Supprime un objet de la base de données grâce à son identifiant.
26      * @param id L'identifiant unique de l'objet.
27      */
28     public void deleteGeneric(int id);
29
30     /**
31      * Récupère tous les objets d'une entité depuis la base de données.
32      * @return Une liste d'objets de type générique.
33      */
34     public List<T> getAllGeneric();
35 }

```

### 3.2.3 Classe HolidayDAOimplement

La classe HolidayDAOimplement est l'implémentation concrète de l'interface GenericDAOI pour la classe Holiday. Elle permet d'exécuter les opérations CRUD pour l'entité Holiday en interagissant avec la base de données.

```
1 package DAO;
2
3 import java.sql.*;
4 import java.time.LocalDate;
5 import java.time.temporal.ChronoUnit;
6 import java.util.ArrayList;
7 import java.util.List;
8 import javax.swing.JOptionPane;
9 import Model.*;
10
11 public class HolidayDAOimplement implements GenericDAOI<Holiday> {
12
13     private Connect c; // Objet de connexion à la base de données
14     public EmployeeDAOimplement emp1 = new EmployeeDAOimplement(); // DAO
15     ↪ pour les employés
16
17     public HolidayDAOimplement() {
18         this.c = new Connect(); // Initialisation de la connexion à la
19         ↪ base de données
20     }
21
22     EmployeeDAOimplement emp = new EmployeeDAOimplement();
23
24     // Ajouter un congé
25     @Override
26     public void addGeneric(Holiday holiday) {
27         String sql = "INSERT INTO
28         ↪ holiday(date_debut,date_fin,Type,employe_id) VALUES (?, ?, ?,
29         ↪ ?)"; // SQL pour insérer un nouveau congé
30
31         try (PreparedStatement st = c.getConnect().prepareStatement(sql)) {
32             // Définir les paramètres de la requête
33             st.setDate(1, Date.valueOf(holiday.getDateDebut()));
34             st.setDate(2, Date.valueOf(holiday.getDateFin()));
35             st.setString(3, holiday.getType().name());
36             st.setInt(4, holiday.getId_employe());
37
38             // Exécution de la requête
39             st.executeUpdate();
40             JOptionPane.showMessageDialog(null, "Congé ajouté avec succès
41             ↪ !");
42         } catch (SQLException e) {
43             JOptionPane.showMessageDialog(null, "Erreur dans l'ajout d'un
44             ↪ holiday", "Erreur", JOptionPane.ERROR_MESSAGE);
45         }
46     }
47 }
```

```

39     }
40 }
41
42 // Modifier un congé
43 @Override
44 public void modifyGeneric(Holiday holiday) {
45     String sql = "UPDATE Holiday SET date_debut = ?, date_fin = ?, Type
    ↪ = ?, employe_id = ? WHERE id = ?"; // SQL pour mettre à jour
    ↪ un congé
46
47     try (PreparedStatement st = c.getConnect().prepareStatement(sql)) {
48         // Définir les paramètres de la requête
49         st.setDate(1, Date.valueOf(holiday.getDateDebut()));
50         st.setDate(2, Date.valueOf(holiday.getDateFin()));
51         st.setString(3, holiday.getType().name());
52         st.setInt(4, holiday.getId_employe());
53         st.setInt(5, holiday.getId());
54
55         // Exécution de la requête
56         int rowsAffected = st.executeUpdate();
57         JOptionPane.showMessageDialog(null, "Holiday modifié avec succès
    ↪ !", "Message", JOptionPane.INFORMATION_MESSAGE);
58
59     } catch (SQLException e) {
60         System.err.println("Error updating holiday: " + e.getMessage());
61     }
62 }
63
64 // Supprimer un congé
65 @Override
66 public void deleteGeneric(int id) {
67     String sql = "DELETE FROM Holiday WHERE id = ?"; // SQL pour
    ↪ supprimer un congé
68
69     try (PreparedStatement st = c.getConnect().prepareStatement(sql)) {
70         st.setInt(1, id); // Définir l'ID du congé à supprimer
71
72         // Exécution de la requête
73         int rowsAffected = st.executeUpdate();
74         JOptionPane.showMessageDialog(null, "Holiday supprimé avec
    ↪ succès !", "Message", JOptionPane.INFORMATION_MESSAGE);
75
76     } catch (SQLException e) {
77         System.err.println("Error deleting holiday: " + e.getMessage());
78     }
79 }
80
81 // Récupérer tous les congés avec jointure entre Holiday et Employee
82 @Override

```

```

83 public List<Object[]> getAllGeneric() {
84     List<Object[]> infos = new ArrayList<>();
85     String sql = "SELECT h.id, h.date_debut, h.date_fin, h.Type,
86         ↪ h.employe_id, e.solde, e.nom " +
87         ↪ "FROM Holiday h JOIN Employees e ON h.employe_id =
88         ↪ e.id"; // SQL pour récupérer les congés avec les
89         ↪ infos des employés
90
91     try (PreparedStatement st = c.getConnect().prepareStatement(sql);
92         ResultSet rs = st.executeQuery()) {
93
94         // Parcours du résultat
95         while (rs.next()) {
96             infos.add(new Object[] {
97                 rs.getInt("id"), // ID du congé
98                 rs.getDate("date_debut"), // Date de début
99                 rs.getDate("date_fin"), // Date de fin
100                 rs.getString("Type"), // Type de congé
101                 rs.getDouble("solde"), // Solde de congé de
102                 ↪ l'employé
103                 rs.getString("nom") // Nom de l'employé
104             });
105         }
106
107     } catch (SQLException e) {
108         System.err.println("Erreur lors de la récupération des congés :
109         ↪ " + e.getMessage());
110     }
111
112     return infos; // Retourner les données récupérées
113 }
114
115 // Récupérer les IDs des employés
116 public List<String> FindById() {
117     List<String> namesWithId = new ArrayList<>();
118     String query = "SELECT nom, prenom, id FROM employees"; // SQL pour
119     ↪ récupérer les IDs et noms des employés
120
121     try (PreparedStatement ps = c.getConnect().prepareStatement(query);
122         ResultSet rs = ps.executeQuery()) {
123
124         // Parcours du résultat
125         while (rs.next()) {
126             String fullName = rs.getString("nom") + " " +
127                 ↪ rs.getString("prenom"); // Nom complet de l'employé
128             int id = rs.getInt("id"); // ID de l'employé
129             namesWithId.add(id + " - " + fullName); // Ajouter à la
130             ↪ liste
131         }
132     }

```

```

124     } catch (Exception e) {
125         e.printStackTrace();
126     }
127
128     return namesWithId; // Retourner la liste des employés avec leurs
129         ↪ IDs
130 }
131
132 // Vérifier si un congé existe
133 public boolean allHoliday(int id) {
134     String query = "SELECT * FROM holiday WHERE id = ?"; // SQL pour
135         ↪ vérifier si un congé existe
136
137     try (PreparedStatement ps = c.getConnection().prepareStatement(query))
138     {
139         ps.setInt(1, id); // Définir l'ID du congé
140
141         // Exécuter la requête
142         try (ResultSet rs = ps.executeQuery()) {
143             if (rs.next()) {
144                 return true; // Le congé existe
145             }
146         }
147     } catch (Exception e) {
148         System.out.println("Erreur lors de la vérification de l'ID de
149             ↪ congé : " + e.getMessage());
150         e.printStackTrace();
151     }
152
153     return false; // Le congé n'existe pas
154 }
155
156 // Récupérer le solde actuel de congé d'un employé
157 public double soldeActuel(int id) {
158     return emp1.SoldeActuel(id); // Appel à la méthode SoldeActuel de
159         ↪ l'EmployeeDAOImplement
160 }
161
162 // Modifier le solde de congé d'un employé
163 public void ModifierSolde(int id, double soldeDemande) {
164     emp1.ModifierSolde(id, soldeDemande); // Appel à la méthode
165         ↪ ModifierSolde de l'EmployeeDAOImplement
166 }
167
168 // Vérifier le chevauchement des congés
169 public int getOverlapDays(int employeeId, LocalDate newStartDate,
170     ↪ LocalDate newEndDate) {
171     String query = "SELECT date_debut, date_fin FROM holiday WHERE
172         ↪ employe_id = ?"; // SQL pour récupérer les congés existants de
173         ↪ l'employé

```



```

166 try (PreparedStatement ps = c.getConnect().prepareStatement(query))
167     ↪ {
168     ps.setInt(1, employeeId); // Définir l'ID de l'employé
169     try (ResultSet rs = ps.executeQuery()) {
170         while (rs.next()) {
171             LocalDate existingStartDate =
172                 ↪ rs.getDate("date_debut").toLocalDate();
173             LocalDate existingEndDate =
174                 ↪ rs.getDate("date_fin").toLocalDate();
175
176             // Cas 1 : Le nouveau congé est exactement identique à
177             ↪ l'existant
178             if (newStartDate.isEqual(existingStartDate) &&
179                 ↪ newEndDate.isEqual(existingEndDate)) {
180                 return 0;
181             }
182
183             // Cas 2 : Le nouveau congé est complètement inclus
184             ↪ dans un congé existant
185             if (!newStartDate.isBefore(existingStartDate) &&
186                 ↪ !newEndDate.isAfter(existingEndDate)) {
187                 return 0;
188             }
189
190             // Cas 3 : Le nouveau congé s'étend au-delà de la fin
191             ↪ d'un congé existant
192             if (newStartDate.isBefore(existingEndDate) &&
193                 ↪ newEndDate.isAfter(existingEndDate)) {
194                 long additionalDays =
195                     ↪ ChronoUnit.DAYS.between(existingEndDate,
196                     ↪ newEndDate);
197                 return (int) additionalDays;
198             }
199
200             // Cas 4 : Les congés ne se chevauchent pas
201             if (newEndDate.isBefore(existingStartDate) ||
202                 ↪ newStartDate.isAfter(existingEndDate)) {
203                 continue; // Pas de chevauchement
204             }
205         }
206     }
207 } catch (SQLException e) {
208     System.err.println("Erreur lors de la vérification des
209         ↪ chevauchements : " + e.getMessage());
210     e.printStackTrace();
211 }
212
213 // Aucun chevauchement trouvé

```

```

202         return -1;
203     }
204 }
205

```

### 3.3 Package View

Le package **View** contient la classe **HolidayView**, responsable de l’affichage et de la gestion des congés dans l’interface utilisateur. Elle permet à l’utilisateur de visualiser, ajouter, modifier et supprimer des congés.

#### 3.3.1 Classe HolidayView

La classe **HolidayView** fournit une interface permettant de manipuler les congés via un tableau et des boutons d’action pour l’ajout, la modification et la suppression des congés. Elle interagit avec le modèle pour récupérer et envoyer les données vers la base de données.

**Code de la classe HolidayView :** .

```

1  package View;
2
3  import javax.swing.*;
4  import javax.swing.table.DefaultTableModel;
5  import Model.TypeH;
6  import java.awt.*;
7  import java.awt.event.ActionListener;
8  import java.util.Date;
9
10 public class HolidayView {
11     // Déclaration des panels
12     public JPanel panel1;
13     public JPanel panel2;
14     public JPanel panel3;
15
16     // Déclaration des colonnes pour le tableau
17     String[] columnNames = {"ID", "Date de début", "Date de fin", "Type",
18         ↪ "nom_employe"};
19
20     // Déclaration des composants pour l'interface utilisateur
21     public JLabel lblId, lblDateDebut, lblDateFin, lblType, lblIdEmploye;
22     public JTextField idField;
23     public JTextField txtDateDebut, txtDateFin;
24     public JComboBox<TypeH> cmbType; // ComboBox pour sélectionner le type
25         ↪ de congé
26     public JComboBox<String> cmbIdEmploye; // ComboBox pour sélectionner
27         ↪ l'employé
28     public JTable table; // Tableau pour afficher les congés
29     public DefaultTableModel model; // Modèle du tableau
30     public JScrollPane scrollPane; // Permet de défiler dans le tableau
31     public JButton btnAjouter, btnModifier, btnSupprimer, btnAfficher; //
32         ↪ Boutons pour ajouter, modifier, supprimer et afficher les congés

```

```

29
30 // Constructeur de la vue
31 public HolidayView() {
32     // Initialisation des panels avec leur disposition respective
33     panel1 = new JPanel(new BorderLayout()); // Panel principal avec
        ↳ BorderLayout
34     panel2 = new JPanel(new GridLayout(6, 2)); // Panel pour les
        ↳ champs de saisie (GridLayout)
35     panel3 = new JPanel(new FlowLayout()); // Panel pour les boutons
        ↳ (FlowLayout)
36
37     // Création et configuration des composants (labels, textfields,
        ↳ comboboxes)
38     lblDateDebut = new JLabel("Date de début (yyyy-MM-dd):");
39     txtDateDebut = new JTextField(10);
40     lblDateFin = new JLabel("Date de fin (yyyy-MM-dd):");
41     txtDateFin = new JTextField(10);
42     lblType = new JLabel("Type de congé:");
43     cmbType = new JComboBox<>(TypeH.values()); // ComboBox avec les
        ↳ types de congés disponibles
44     lblIdEmploye = new JLabel("ID Employé:");
45     cmbIdEmploye = new JComboBox<>(); // ComboBox pour sélectionner
        ↳ l'employé (les IDs seront ajoutés dynamiquement)
46
47     // Ajouter les composants au panel2 (GridLayout)
48     panel2.add(lblDateDebut);
49     panel2.add(txtDateDebut);
50     panel2.add(lblDateFin);
51     panel2.add(txtDateFin);
52     panel2.add(lblType);
53     panel2.add(cmbType);
54     panel2.add(lblIdEmploye);
55     panel2.add(cmbIdEmploye);
56
57     // Initialisation du modèle de tableau avec les colonnes définies
58     model = new DefaultTableModel(columnNames, 0);
59     table = new JTable(model); // Création du tableau avec le modèle
60     scrollPane = new JScrollPane(table); // Ajout du tableau dans un
        ↳ JScrollPane pour permettre le défilement
61
62     // Création des boutons
63     btnAjouter = new JButton("Ajouter");
64     btnModifier = new JButton("Modifier");
65     btnSupprimer = new JButton("Supprimer");
66     btnAfficher = new JButton("Afficher");
67
68     // Ajouter les boutons au panel3
69     panel3.add(btnAjouter);
70     panel3.add(btnModifier);

```

```

71     panel3.add(btnSupprimer);
72     panel3.add(btnAfficher);
73
74     // Ajouter les panels au panel principal (panel1) avec
75     ↪ BorderLayout
76     panel1.add(panel2, BorderLayout.NORTH); // Panel des champs en
77     ↪ haut
78     panel1.add(scrollPane, BorderLayout.CENTER); // Tableau au centre
79     panel1.add(panel3, BorderLayout.SOUTH); // Boutons en bas
80 }
81
82 // Méthodes getter et setter pour les composants
83
84 // Retourner le tableau (JTable) pour y accéder ailleurs
85 public JTable getTable() {
86     return table;
87 }
88
89 // Retourner le panel principal de la vue
90 public JPanel getpan() {
91     return panel1;
92 }
93
94 // Définir un tableau à afficher
95 public void setTable(JTable table) {
96     this.table = table;
97 }
98
99 // Retourner les noms des colonnes du tableau
100 public String[] getColumnNames() {
101     return columnNames;
102 }
103
104 // Définir la ComboBox pour le type de congé
105 public void setCmbType(JComboBox<TypeH> cmbType) {
106     this.cmbType = cmbType;
107 }
108
109 // Ajouter un listener pour le bouton "Ajouter"
110 public void getAjouterListener(ActionListener e) {
111     btnAjouter.addActionListener(e);
112 }
113

```

### 3.3.2 Classe GestionEmployesConges

La classe `GestionEmployesConges` est une fenêtre principale qui gère l'interface de gestion des employés et des congés, en utilisant un `JTabbedPane` pour organiser les onglets.

- `initializeEmployeeManagement()` : Initialise la gestion des employés et ajoute l'onglet correspondant.
- `initializeHolidayManagement()` : Initialise la gestion des congés et ajoute l'onglet correspondant.

Voici le code correspondant :

```
1 package View;
2
3 import Controller.EmployeeController;
4 import Controller.HolidayController;
5 import Model.EmployeeModel;
6 import Model.HolidayModel;
7 import DAO.EmployeeDAOImplement;
8 import DAO.HolidayDAOImplement;
9 import javax.swing.*;
10
11 public class GestionEmployesConges extends JFrame {
12
13     private JTabbedPane tabbedPane; // Déclaration de l'onglet pour
14     l'interface
15
16     public GestionEmployesConges() {
17         this.setSize(800, 600); // Définir la taille de la fenêtre
18         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Fermer
19         l'application
20         à la fermeture de la fenêtre
21         this.setTitle("Gestion Employés et Congés"); // Titre de la
22         ↪ fenêtre
23
24         tabbedPane = new JTabbedPane(); // Initialiser l'onglet
25
26         initializeEmployeeManagement(); // Initialisation de la gestion
27         des employés
28         initializeHolidayManagement(); // Initialisation de la gestion
29         des congés
30
31         add(tabbedPane); // Ajouter l'onglet au JFrame
32     }
33
34     private void initializeEmployeeManagement() {
35         EmployeeDAOImplement employeeDAO = new EmployeeDAOImplement(); //
36         ↪ Créer
37         l'instance du DAO pour les employés
38         EmployeeModel employeeModel = new EmployeeModel(employeeDAO); //
39         ↪ Créer
```

```

37     le modèle des employés
38     EmployeesView employeeView = new EmployeesView(); // Créer
39     la vue des employés
40
41     EmployeeController employeeController = new
42     EmployeeController(employeeView, employeeModel); // Créer
43     le contrôleur pour les employés
44
45     // Ajouter l'onglet Employés
46     tabbedPane.addTab("Employés", employeeView.getPan());
47
48 }
49
50 private void initializeHolidayManagement() {
51     HolidayDAOimplement holidayDAO = new
52     HolidayDAOimplement(); // Créer
53     l'instance du DAO pour les congés
54     HolidayModel holidayModel = new HolidayModel(holidayDAO); // Créer
55     le modèle des congés
56     HolidayView holidayView = new HolidayView(); // Créer
57     la vue des congés
58
59     HolidayController holidayController = new HolidayController
60     (holidayView, holidayModel); // Créer le contrôleur pour les
61     ↪ congés
62
63     tabbedPane.addTab("Congés", holidayView.getpan()); // Ajouter
64     l'onglet Congés
65
66 }

```

## 3.4 Package Controller

### 3.4.1 Classe HolidayController

La classe `HolidayController` gère les événements liés à la vue des congés. Elle répond aux actions de l'utilisateur (ajout, modification, suppression de congés) et met à jour le modèle et la vue en conséquence. Elle interagit avec l'objet `HolidayView` pour l'affichage et l'objet `HolidayModel` pour la logique métier. et un objet `HolidayModel` qui contient la logique métier pour gérer les congés.

```

1 package Controller;
2
3 import Model.HolidayModel;
4 import View.HolidayView;
5 import Model.Employees;
6 import Model.TypeH;

```

```

7
8 import javax.swing.*;
9 import javax.swing.table.DefaultTableModel;
10
11 import java.awt.event.*;
12 import java.sql.Date;
13 import java.sql.SQLException;
14 import java.text.SimpleDateFormat;
15 import java.time.LocalDate;
16 import java.time.format.DateTimeFormatter;
17 import java.time.format.DateTimeParseException;
18 import java.util.List;
19
20 public class HolidayController {
21     private HolidayView view;
22     private HolidayModel model;
23     private int selectedEmployeeId = -1; // ID de l'employé sélectionné
24
25     // Constructeur du contrôleur, initialise la vue et le modèle
26     public HolidayController(HolidayView view, HolidayModel model) {
27         this.view = view;
28         this.model = model;
29
30         // Remplir la liste des IDs des employés dans la ComboBox
31         fillEmployeeIdsComboBox();
32
33         //////////////////////////////////////////// Button Ajouter
34         ↪ ////////////////////////////////////////////
35
36         // ActionListener pour le bouton "Ajouter"
37         this.view.btnAjouter.addActionListener(new ActionListener() {
38             @Override
39             public void actionPerformed(ActionEvent e) {
40                 try {
41                     // Récupérer l'ID de l'employé sélectionné
42                     Integer idEmploye = RetournIds();
43                     TypeH typeConges = (TypeH)
44                     ↪ view.cmbType.getSelectedItem(); // Récupérer le
45                     ↪ type de congé sélectionné
46                     String dateDebutStr =
47                     ↪ view.txtDateDebut.getText().trim();
48                     String dateFinStr = view.txtDateFin.getText().trim();
49
50                     // Validation du format des dates
51                     if (!isValidDate(dateDebutStr, "yyyy-MM-dd")) {
52                         JOptionPane.showMessageDialog(null, "La date de début
53                         ↪ est invalide. Veuillez utiliser le format
54                         ↪ yyyy-MM-dd.");
55                     }
56                     return;
57                 }
58             }
59         });
60     }
61 }

```

```

50         }
51         if (!isValidDate(dateFinStr, "yyyy-MM-dd")) {
52             JOptionPane.showMessageDialog(null, "La date de fin
53                 ↳ est invalide. Veuillez utiliser le format
54                 ↳ yyyy-MM-dd.");
55             return;
56         }
57
58         // Conversion des dates de String en LocalDate
59         LocalDate dateDebut = LocalDate.parse(dateDebutStr,
60             ↳ DateTimeFormatter.ofPattern("yyyy-MM-dd"));
61         LocalDate dateFin = LocalDate.parse(dateFinStr,
62             ↳ DateTimeFormatter.ofPattern("yyyy-MM-dd"));
63
64         // Vérifier que la date de fin est après la date de
65         ↳ début
66         if (!dateFin.isAfter(dateDebut)) {
67             JOptionPane.showMessageDialog(null, "La date de fin
68                 ↳ doit être strictement supérieure à la date de
69                 ↳ début.");
70             return;
71         }
72
73         // Ajouter le congé via le modèle
74         model.addHoliday(dateDebut, dateFin, typeConges,
75             ↳ idEmploye);
76
77         // Rafraîchir l'affichage et vider les champs
78         afficher();
79         view.txtDateDebut.setText("");
80         view.txtDateFin.setText("");
81
82     } catch (Exception ex) {
83         JOptionPane.showMessageDialog(null, "Une erreur est
84             ↳ survenue : " + ex.getMessage());
85     }
86 }
87
88 ////////////////////////////////// Selection
89 ↳ //////////////////////////////////
90
91 // ActionListener pour la sélection d'une ligne dans le tableau
92 view.table.getSelectionModel().addListSelectionListener(e -> {
93     if (!e.getValueIsAdjusting()) {
94         int selectedRow = view.table.getSelectedRow();
95         if (selectedRow != -1) {
96             selectedEmployeeId = (int)
97                 ↳ view.model.getValueAt(selectedRow, 0); // Récupérer
98                 ↳ l'ID de l'employé sélectionné

```



```

88      // Récupérer les dates de début et de fin à partir de
89      ↪ la ligne sélectionnée
90      Date dateDebut = (Date)
91      ↪ view.model.getValueAt(selectedRow, 1);
92      Date dateFin = (Date) view.model.getValueAt(selectedRow,
93      ↪ 2);
94
95      // Formatage des dates pour les afficher dans les
96      ↪ champs de texte
97      SimpleDateFormat dateFormat = new
98      ↪ SimpleDateFormat("yyyy-MM-dd");
99      String dateDebutStr = (dateDebut != null) ?
100      ↪ dateFormat.format(dateDebut) : "";
101      String dateFinStr = (dateFin != null) ?
102      ↪ dateFormat.format(dateFin) : "";
103
104      // Mettre à jour les champs de texte avec les dates
105      ↪ sélectionnées
106      view.txtDateDebut.setText(dateDebutStr);
107      view.txtDateFin.setText(dateFinStr);
108    }
109  });
110
111  //////////// Button Modifier ////////////
112
113  // ActionListener pour le bouton "Modifier"
114  this.view.btnModifier.addActionListener(new ActionListener() {
115      @Override
116      public void actionPerformed(ActionEvent e) {
117          try {
118              // Récupérer les informations de l'employé et des
119              ↪ dates
120              int idEmploye = RetourIds();
121              TypeH typeConges = (TypeH)
122              ↪ view.cmbType.getSelectedItem();
123              String dateDebutStr =
124              ↪ view.txtDateDebut.getText().trim();
125              String dateFinStr = view.txtDateFin.getText().trim();
126
127              // Validation des dates
128              if (!isValidDate(dateDebutStr) ||
129              ↪ !isValidDate(dateFinStr)) {
130                  JOptionPane.showMessageDialog(null, "Veuillez entrer
131                  ↪ des dates valides au format yyyy-MM-dd.");
132                  return;
133              }
134          }
135      }
136  });

```

```

124         // Conversion des dates en LocalDate
125         LocalDate dateDebut = LocalDate.parse(dateDebutStr,
126             ↪ DateTimeFormatter.ofPattern("yyyy-MM-dd"));
127         LocalDate dateFin = LocalDate.parse(dateFinStr,
128             ↪ DateTimeFormatter.ofPattern("yyyy-MM-dd"));
129
130         // Modifier les congés via le modèle
131         model.modifyHoliday(selectedEmployeeId, dateDebut,
132             ↪ dateFin, typeConges, idEmploye);
133
134         // Rafraîchir l'affichage
135         afficher();
136     } catch (Exception ex) {
137         JOptionPane.showMessageDialog(null, "Une erreur s'est
138             ↪ produite : " + ex.getMessage());
139         ex.printStackTrace();
140     }
141 }
142 });
143
144 ////////////////////////////////////////////////// Button Supprimer //////////////////////////////////////
145
146 // ActionListener pour le bouton "Supprimer"
147 this.view.btnSupprimer.addActionListener(new ActionListener() {
148     @Override
149     public void actionPerformed(ActionEvent e) {
150         int selectedRow = view.table.getSelectedRow();
151         if (selectedRow != -1) {
152             // Supprimer la ligne sélectionnée du tableau et du
153             ↪ modèle
154             int id =
155                 ↪ Integer.parseInt(view.model.getValueAt(selectedRow,
156                 ↪ 0).toString());
157             model.deleteHoliday(id);
158             view.model.removeRow(selectedRow);
159             afficher(); // Rafraîchir l'affichage
160         } else {
161             JOptionPane.showInputDialog(view, "Veuillez sélectionner
162                 ↪ une ligne à modifier.");
163         }
164     }
165 }
166 });
167
168 // ActionListener pour le bouton "Afficher"
169 this.view.btnAfficher.addActionListener(new ActionListener() {
170     @Override
171     public void actionPerformed(ActionEvent e) {
172         afficher(); // Rafraîchir l'affichage
173     }
174 }

```

```

165     });
166 }
167
168 // Méthode pour valider un format de date (yyyy-MM-dd)
169 private boolean isValidDate(String dateStr) {
170     try {
171         LocalDate.parse(dateStr,
172             ↪ DateTimeFormatter.ofPattern("yyyy-MM-dd"));
173         return true;
174     } catch (DateTimeParseException e) {
175         return false;
176     }
177 }
178
179 // Méthode pour remplir la ComboBox des IDs des employés
180 private void fillEmployeeIdsComboBox() {
181     List<String> employeeIds = model.getIds();
182     view.cmbIdEmploye.removeAllItems();
183
184     // Ajouter chaque ID d'employé dans la ComboBox
185     for (String id : employeeIds) {
186         view.cmbIdEmploye.addItem(id);
187     }
188 }
189
190 // Méthode pour afficher les congés dans le tableau
191 public void afficher() {
192     Object[][] holidayData = model.getAllHolidays(); // Récupérer les
193     ↪ données des congés
194     view.model = new DefaultTableModel(holidayData,
195     ↪ view.getColumnNames()); // Mettre à jour le modèle du tableau
196     view.getTable().setModel(view.model);
197     fillEmployeeIdsComboBox(); // Mettre à jour les IDs des employés
198     ↪ dans la ComboBox
199 }
200
201 // Méthode pour valider une date avec un format spécifié
202 private boolean isValidDate(String dateStr, String format) {
203     DateTimeFormatter formatter = DateTimeFormatter.ofPattern(format);
204     try {
205         LocalDate.parse(dateStr, formatter);
206         return true;
207     } catch (Exception e) {
208         return false;
209     }
210 }
211
212 // Méthode pour récupérer l'ID de l'employé sélectionné dans la
213 ↪ ComboBox

```

```

209     public Integer RetournIds() {
210         int idEmploye = -1;
211         String selectedItem =
            ↳ view.cmbIdEmploye.getSelectedItem().toString().trim();
212
213         if (selectedItem != null && !selectedItem.isEmpty()) {
214             try {
215                 // Extraire l'ID de la sélection (séparée par un tiret)
216                 String[] parts = selectedItem.split(" - ");
217                 if (parts.length == 2) {
218                     idEmploye = Integer.parseInt(parts[0].trim()); //
                        ↳ Extraire l'ID
219                 }
220             } catch (NumberFormatException e) {
221                 e.printStackTrace();
222             }
223         }
224
225         return idEmploye;
226     }
227 }
228

```

Code de la classe

### 3.5 Package Main

Le package Main contient la classe principale Main, qui est le point d'entrée de l'application. Cette classe est responsable de l'initialisation de l'interface graphique en lançant la fenêtre principale de l'application. Elle crée une instance de la classe **GestionEmployesConges** et l'affiche à l'écran. Cela permet de démarrer l'application avec les fonctionnalités de gestion des employés et des congés.

```

1  package Main;
2
3  import View.GestionEmployesConges; // Importer la classe
   ↳ GestionEmployesConges du package 'view'
4
5  public class Main {
6
7      public static void main(String[] args) {
8          // Créer une instance de la classe GestionEmployesConges et rendre
           ↳ la fenêtre visible
9          javax.swing.SwingUtilities.invokeLater(() -> {
10              new GestionEmployesConges().setVisible(true);
11          });
12     }
13 }

```

## 4 Scripts d'Exécution du TP

### 4.1 Ajouter Conge

Gestion Employés et Congés

Employés Congés

Date de début (yyyy-MM-dd): 2023-01-01

Date de fin (yyyy-MM-dd): 2023-01-02

Type de congé: Conge\_payee

ID Employé: 26 - test test

ID	Date de début	Date de fin	Type	nom_employe
116	2020-01-01	2020-01-08	conge_non_payee	new
117	2020-01-01	2020-01-10	Conge_payee	new
118	2020-01-07	2020-01-12	Conge_payee	new
119	2020-01-01	2020-01-05	Conge_payee	test
120	2020-01-01	2020-01-15	Conge_payee	test

Message

Congé ajouté avec succès !

OK

Ajouter Modifier Supprimer Afficher

FIGURE 3 – Bouton Ajouter

#### Gestion des Erreurs : Validation des Dates

Lors de la saisie des dates de congé, il est nécessaire de vérifier que la *date de fin* n'est pas antérieure à la *date de début*. Si cette condition est remplie, un message d'erreur doit être affiché pour informer l'utilisateur de l'incohérence. Voici l'exemple de validation et le message d'erreur associé :

Gestion Employés et Congés

Employés Congés

ID: 3

Date de début (yyyy-MM-dd): 2021-05-06

Date de fin (yyyy-MM-dd): 2014-08-06

Type de congé: Conge\_payee

ID Employé: 4

ID	Date de début	Date de fin	Type	nom_employe
1	2014-05-03	2014-06-02	Conge_payee	jksc
2	2014-05-06	2014-08-06	Conge_payee	test
3	2011-05-06	2014-08-06	Conge_payee	test
6	2004-02-05	2004-02-05	Conge_payee	jksc
7	2014-05-03	2014-06-02	Conge_payee	kfarah
8	2011-05-06	2014-08-06	Conge_payee	test
10	2011-05-06	2014-08-06	Conge_payee	test

Message

erreur dans les dates

OK

Ajouter Modifier Supprimer Afficher

FIGURE 4 – Valider les dates

## Validation des Jours Disponibles

Un autre critère important consiste à vérifier si l'employé possède un nombre de jours de congé suffisant. Voici les étapes de la validation :

1. Calculer le nombre de jours demandés entre **la date de début** et **la date de fin**.
2. Comparer ce nombre au **solde de jours de congé disponibles** de l'employé.
3. Si le solde est insuffisant, afficher un message d'erreur pour informer l'utilisateur.

The screenshot shows a software window titled "Gestion Employés et Congés" with two tabs: "Employés" and "Congés". The "Congés" tab is active, displaying a form with the following fields:

- ID: (empty)
- Date de début (yyyy-MM-dd): 2001-02-06
- Date de fin (yyyy-MM-dd): 2001-03-05
- Type de congé: Conge\_payee (dropdown menu)
- ID Employé: 1 (dropdown menu)

Below the form is a table with the following data:

ID	Date de début	Date de fin	Type	nom_employe
61	2000-02-01	2000-02-26	Conge_payee	kaydi
63	2000-06-05	2000-06-09	Conge_payee	kaydi
65	2020-02-01	2020-02-03	Conge_payee	kaydi
62	2000-02-03	2000-02-05	Conge_payee	mahdi

A message dialog box is displayed in the center of the window with the text: "Le nombre de jours de congé est insuffisant !" (The number of leave days is insufficient!). The dialog has an "OK" button.

At the bottom of the window, there are four buttons: "Ajouter", "Modifier", "Supprimer", and "Afficher".

FIGURE 5 – Valider le Conge

## 4.2 Modifier Conge

Le bouton **Modifier** permet de mettre à jour les informations d'un conge existant dans la base de données. Pour ce faire, l'utilisateur doit entrer l'**ID** de Conge dont il souhaite modifier les informations, puis remplir les nouveaux champs avec les données à jour .

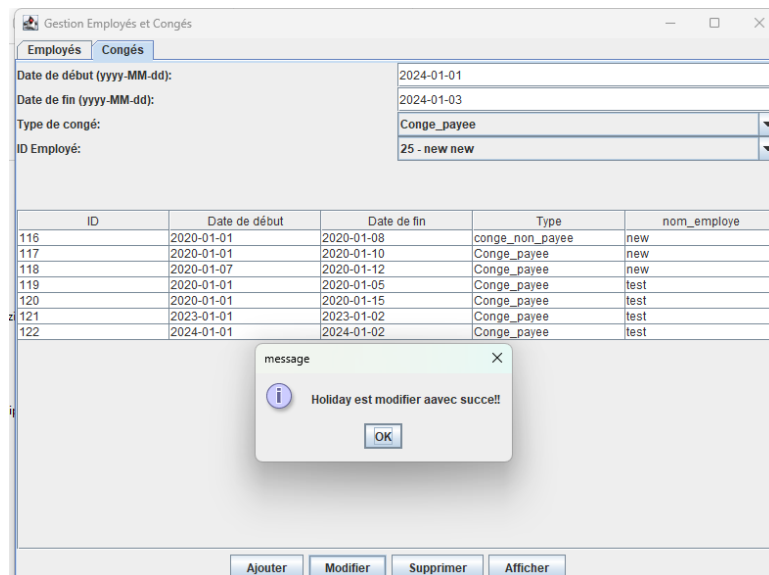


FIGURE 6 – Button Modifier

### 4.3 Supprimer Conge

Le bouton **Supprimer** permet de supprimer un congé de la base de données. Pour ce faire, l'utilisateur doit entrer l'**ID** du congé à supprimer.

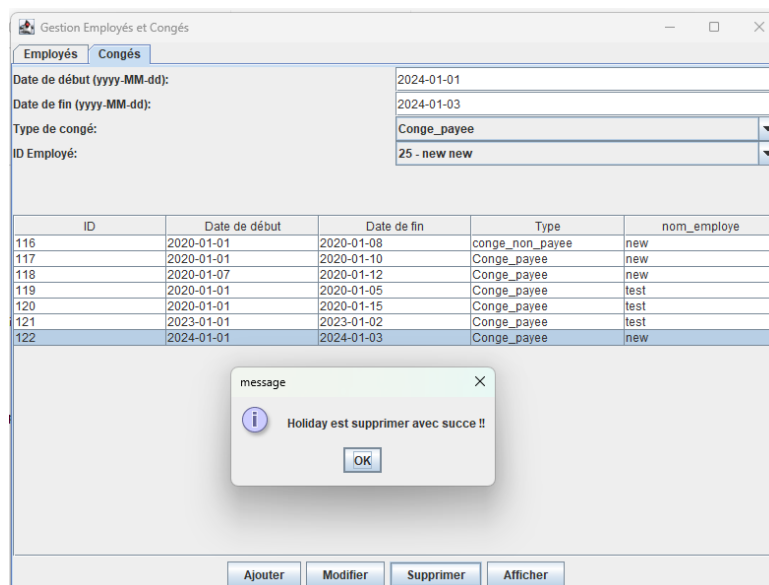


FIGURE 7 – Button Supprimer

### 4.4 Afficher Congé

Lorsque l'utilisateur appuie sur le bouton **Afficher**, les données des congés sont extraites de la base de données et affichées dans l'interface utilisateur.

Gestion Employés et Congés

Employés

Congés

Nom:

Prenom:

Telephone:

Email:

Salaire:

Roles:

DIRECTION\_GENERALE

Postes:

DEVEOPPEUR\_FULL\_STACK

ID	Nom	Prenom	Telephone	Email	Salaire	Role	Poste	Solde
25	new	new	gw@gmail.com	1234567891	545.0	DIRECTION_	DEVEOPPEU...	0.0
26	test	test	test@gmail.c.	1234567896	123.0	DIRECTION_	DEVEOPPEU...	0.0
28	new	test	gw@gmail.com	1234569878	545.0	DIRECTION_	DEVEOPPEU...	6.0
29	kaydi	kaydi	test@gmail.c.	1234567896	123.0	DIRECTION_	DEVEOPPEU...	4.0
30	farah	test	gw@gmail.com	1234569878	545.0	DIRECTION_	DEVEOPPEU...	11.0

Ajouter

Modifier

Supprimer

Afficher

FIGURE 8 – Button Afficher

## 4.5 Scripts de l'Architecture MVC et DAO



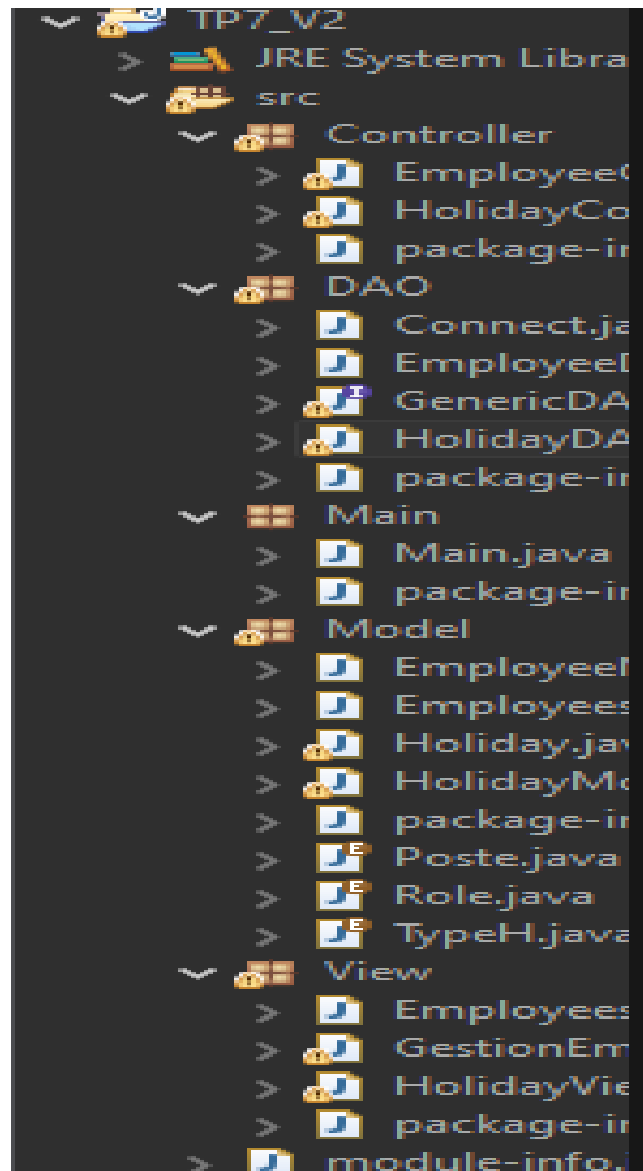


FIGURE 9 – Architecture MVC et DAO

## 5 Conclusion

Ce travail pratique a permis de mettre en place une application de gestion des employés et des congés dans une seule fenêtre, avec deux onglets dédiés à chaque gestion. L'application repose sur l'architecture MVC (Modèle-Vue-Contrôleur) et le design pattern DAO (Data Access Object), permettant une gestion efficace des données tout en garantissant une séparation claire des responsabilités entre la logique métier, l'interface utilisateur et l'accès aux données.

De plus, ce TP intègre l'utilisation de la généricité, ce qui permet de rendre l'application plus flexible et réutilisable pour d'autres entités similaires à "Employé" et "Congé". Cette approche simplifie également l'ajout de nouvelles fonctionnalités tout en minimisant le code redondant.

En résumé, l'application facilite la gestion des informations des employés et de leurs congés, avec une interface simple et une architecture bien structurée.

Pour accéder au code source complet de ce travail pratique, vous pouvez consulter le lien suivant sur GitHub :

Voici le lien vers mon projet GitHub