

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

**Rapport du TP N°2 java avancée**

---

# Gestion des congés

---

**Réalisé par : EL HINA Soukaina**

**Encadré par : Mme.Leila Elkhrof**

**ANNÉE UNIVERSITAIRE : 2024/2025**

# Table des matières

<b>Introduction</b>	<b>4</b>
<b>Outils &amp; environnement de travail</b>	<b>5</b>
1 Environnement de travail . . . . .	5
2 Outils de travail . . . . .	5
3 Language de Programmation . . . . .	6
<b>1 Réalisation</b>	<b>7</b>
1 Création de la base de donnée . . . . .	7
1.1 Script base de donnée . . . . .	7
2 Architecture MVC (Model-View-Controller) . . . . .	8
2.1 Model . . . . .	8
2.2 DAO . . . . .	10
2.3 Controller . . . . .	13
2.4 View . . . . .	18
<b>2 Résultats</b>	<b>25</b>
1 Tables Créées . . . . .	25
2 Résultats de la partie View . . . . .	25
3 Après Ajout . . . . .	26
4 Après modification . . . . .	27
5 Apres Suppression . . . . .	28
<b>3 Conclusion générale</b>	<b>30</b>
<b>4 Références</b>	<b>30</b>

# Table des figures

1	Eclipse logo . . . . .	5
2	MySQL Workbench logo . . . . .	5
3	xampp logo . . . . .	6
4	java developpement kit logo . . . . .	6
5	java logo . . . . .	6
2.1	Tables de la base de donnée . . . . .	25
2.2	Interface Utilisateur . . . . .	26
2.3	Resultat Ajout . . . . .	27
2.4	Resultat de modification . . . . .	27
2.5	Affichage de modification . . . . .	28
2.6	Resultat de suppression . . . . .	29

# Introduction

Ce travail pratique (TP) porte sur la conception et le développement d'une application Java pour la gestion des congés des employés. Ce projet s'appuie sur l'architecture MVC (Model-View-Controller) et intègre des notions avancées de programmation orientée objet (POO) ainsi que la manipulation d'interfaces graphiques à l'aide de la bibliothèque Swing..

L'objectif principal est de proposer une solution complète permettant une gestion centralisée et transparente des congés au sein d'une entreprise, tout en garantissant une séparation claire des responsabilités entre les différentes couches de l'application. Cette approche assure une maintenance facilitée et une évolutivité accrue.

Les fonctionnalités clés de l'application incluent :

- Ajout, modification et suppression des employés.
- Gestion des demandes de congés, incluant leur validation ou refus.
- Suivi des soldes de congés pour chaque employé.
- Affichage des informations sous forme tabulaire, favorisant une expérience utilisateur intuitive.

Ce projet représente une opportunité d'approfondir les compétences techniques dans la conception d'applications professionnelles, tout en explorant des concepts essentiels tels que les génériques, le DAO (Data Access Object) et les bonnes pratiques en conception logicielle.

# Outils & environnement de travail

## 1 Environnement de travail



FIGURE 1 – Eclipse logo

- **Eclipse** : Eclipse est un environnement de développement intégré (IDE) open-source, principalement utilisé pour le développement en Java, mais extensible à d'autres langages grâce à des plugins. Il offre des outils pour écrire, déboguer et gérer du code efficacement, avec une interface modulable et multiplateforme. Très polyvalent, Eclipse est prisé pour le développement d'applications web, mobiles et logicielles.

## 2 Outils de travail



FIGURE 2 – MySQL Workbench logo

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



FIGURE 3 – xampp logo

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



FIGURE 4 – java developpement kit logo

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

### 3 Language de Programmation



FIGURE 5 – java logo

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

# Réalisation

## 1 Création de la base de donnée

### 1.1 Script base de donnée

```
1 -- Cr ation de la base de donn es
2 CREATE DATABASE gestion;
3
4 -- Utilisation de la base de donn es
5 USE gestion;
6 CREATE TABLE employe (
7     id int(11) NOT NULL,
8     nom varchar(100) NOT NULL,
9     prenom varchar(100) NOT NULL,
10    email varchar(150) NOT NULL,
11    telephone varchar(15) DEFAULT NULL,
12    salaire decimal(10,2) NOT NULL,
13    role enum('ADMIN', 'MANAGER', 'EMPLOYEE') NOT NULL,
14    poste enum('DEVELOPER', 'DESIGNER', 'MARKETING', 'OTHER') NOT NULL,
15    solde int(11) NOT NULL
16 ) ;
17
18
19
20 CREATE TABLE `holiday` (
21     id int(11) NOT NULL,
22     id_employe int(11) NOT NULL,
23     startdate date DEFAULT NULL,
24     enddate date DEFAULT NULL,
25     type enum('ANNUAL', 'SICK', 'MATERNITY', 'OTHER') NOT NULL
26 ) ;
```

Listing 1.1 – Script SQL de la base de données

- Ce script est écrit sur MySQL Workbench pour création la base de donnée pour être lié à au code via le driver JDBC pour garantir la gestion .

## 2 Architecture MVC (Model-View-Controller)

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

### 2.1 Model

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

#### **HOLIDAY**

```
1 package model;
2
3 import java.sql.Date;
4
5
6 public class Holiday{
7     private int id_holiday;
8     private int id_employe;
9     private Date startDate;
10    private Date endDate;
11    private Type_holiday type;
12
13    public Holiday(int id_holiday, int id_employe, Date startDate, Date endDate ,
14        Type_holiday type){
15        this.id_holiday = id_holiday;
16        this.id_employe = id_employe;
17        this.startDate = startDate;
18        this.endDate = endDate;
19        this.type = type;
20    }
21    public int getId_holiday() {
22        return id_holiday;
23    }
24    public Date getStartDate() {
25        return startDate;
26    }
27    public Date getEndDate() {
28        return endDate;
29    }
30    public Type_holiday getType() {
31        return type;
32    }
33
34    public int getId_employe() {
35        return id_employe;
36    }
37
38    public Object getSolde() {
39        throw new UnsupportedOperationException("Not supported yet.");
40    }
```



```
41  
42 }
```

### **HolidayLogic**

```
1 package model;  
2 import Controller.EmployeController;  
3 import java.util.List;  
4  
5 import DAO.HolidayDAOimpl;  
6 import java.sql.Date;  
7 public class HolidayModel {  
8     private HolidayDAOimpl dao;  
9  
10    public HolidayModel(HolidayDAOimpl dao) {  
11        this.dao = dao;  
12    }  
13  
14    public boolean addHoliday(int id, int id_employe, Date startdate, Date enddate,  
15    Type_holiday type , Employe targetEmploye) {  
16        if(startdate.after(enddate)) return false;  
17        if(startdate.equals(enddate)) return false;  
18        if(startdate.before(new Date(System.currentTimeMillis()))) return false;  
19        if(enddate.before(new Date(System.currentTimeMillis()))) return false;  
20  
21        long daysBetween = (enddate.toLocalDate().toEpochDay() - startdate.toLocalDate()  
22        .toEpochDay());  
23        if(daysBetween > targetEmploye.getSolde()) return false;  
24        EmployeController.updateSolde(targetEmploye.getId(), targetEmploye.getSolde() -  
25        (int) daysBetween);  
26        Holiday e = new Holiday(id, id_employe, startdate, enddate, type);  
27        dao.add(e);  
28        return true;  
29    }  
30  
31    public List<Holiday> displayHoliday() {  
32        List<Holiday> Holidays = dao.display();  
33        return Holidays;  
34    }  
35  
36    public boolean deleteHoliday(int id) {  
37        dao.delete(id);  
38        return true;  
39    }  
40  
41    public boolean updateHoliday(int id, int id_employe, Date startdate, Date enddate,  
42    Type_holiday type , Employe targetEmploye , int olddaysbetween ) {  
43        long daysBetween = (enddate.toLocalDate().toEpochDay() - startdate.toLocalDate()  
44        .toEpochDay());  
45        if(startdate.after(enddate)) return false;  
46        if(startdate.equals(enddate)) return false;  
47        if(startdate.before(new Date(System.currentTimeMillis()))) return false;
```

```
47         if(enddate.before(new Date(System.currentTimeMillis())) return false;
48
49         if(daysBetween > (targetEmploye.getSolde()+olddaysbetween)) return false;
50         EmployeeController.updateSolde(targetEmploye.getId(), (targetEmploye.getSolde()+
olddaysbetween) - (int) daysBetween);
51
52         Holiday e = new Holiday(id, id_employe, startdate, enddate, type);
53         dao.update(e);
54         return true;
55     }
56
57 }
58
59
60 package model;
61
62 public enum Post {
63     DEVELOPER, DESIGNER, MARKETING, OTHER
64 }
65 package model;
66
67 public enum Role {
68     ADMIN, EMPLOYEE ,MANAGER
69 }
70 package model;
71
72 public enum Type_holiday {
73     ANNUAL, SICK, MATERNITY, OTHER
74 }
```

## 2.2 DAO

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

### DBConnection

```
1 package DAO;
2
3 import java.sql.*;
4
5 class DBConnexion {
6     public static final String url = "jdbc:mysql://localhost:3306/gestion";
7     public static final String user = "root";
8     public static final String password = "";
9     public static Connection conn = null;
10
11     public static Connection getConnexion() throws ClassNotFoundException {
12         if (conn != null) {
13             return conn;
14         }
15         try {
16
17             Class.forName("com.mysql.cj.jdbc.Driver");
```

```
18         conn = DriverManager.getConnection(url, user, password);
19         System.out.println("correct");
20     } catch (SQLException e) {
21         throw new RuntimeException("Error de connexion", e);
22     }
23
24     return conn;
25 }
26 }
```

### GenericDAO

```
1 package DAO;
2
3 import java.util.List;
4 public interface GenericDAOI <T> {
5     public void add(T e);
6     public void delete(int id);
7     public void update(T e);
8     public List<T> display();
9 }
```

### HolidayInterface

```
1 package DAO;
2 import model.Holiday;
3 import model.Type_holiday;
4 import java.sql.Date;
5 import java.sql.PreparedStatement;
6 import java.sql.ResultSet;
7 import java.sql.SQLException;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class HolidayDAOimpl implements GenericDAOI<Holiday>{
12
13     @Override
14     public void add(Holiday e) {
15         String checkSoldeSql = "SELECT solde FROM employe WHERE id = ?";
16         String insertHolidaySql = "INSERT INTO holiday (id_employe, startdate,
17 enddate, type) VALUES (?, ?, ?, ?)";
18
19         try (PreparedStatement checkStmt = DBConnexion.getConnexion().
20 prepareStatement(checkSoldeSql)) {
21             // R cup rer le solde de cong de l'employ
22             checkStmt.setInt(1, e.getId_employe());
23             ResultSet rs = checkStmt.executeQuery();
24
25             if (rs.next()) {
26                 int solde = rs.getInt("solde");
27
28                 // Calculer le nombre de jours demand s
29                 long daysBetween = java.time.temporal.ChronoUnit.DAYS.between(
30 e.getStartDate().toLocalDate(),
```

```

29         e.getEndDate().toLocalDate()
30     );
31
32     if (daysBetween > solde) {
33         System.err.println("Le solde de cong est insuffisant.");
34         return;
35     }
36
37     // Ins rer la demande de cong
38     try (PreparedStatement insertStmt = DBConnexion.getConnexion().
prepareStatement(insertHolidaySql)) {
39         insertStmt.setInt(1, e.getId_employe());
40         insertStmt.setDate(2, e.getStartDate());
41         insertStmt.setDate(3, e.getEndDate());
42         insertStmt.setString(4, e.getType().name());
43
44         insertStmt.executeUpdate();
45
46         // Mettre jour le solde de cong
47         String updateSoldeSql = "UPDATE employe SET solde= solde - ?
WHERE id = ?";
48         try (PreparedStatement updateStmt = DBConnexion.getConnexion().
.prepareStatement(updateSoldeSql)) {
49             updateStmt.setInt(1, (int) daysBetween);
50             updateStmt.setInt(2, e.getId_employe());
51             updateStmt.executeUpdate();
52         }
53     }
54     } else {
55         System.err.println("Employ introuvable.");
56     }
57     } catch (SQLException | ClassNotFoundException exception) {
58         exception.printStackTrace();
59     }
60 }
61
62
63 @Override
64 public void delete(int id) {
65     String sql = "DELETE FROM holiday WHERE id = ?";
66     try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement (
sql)) {
67         stmt.setInt(1, id);
68         stmt.executeUpdate();
69     } catch (SQLException exception) {
70         System.err.println("failed of delete holiday");
71     } catch (ClassNotFoundException ex) {
72         System.err.println("failed connexion with data base");
73     }
74 }
75
76 @Override
77 public void update(Holiday e) {
78     String sql = "UPDATE holiday SET id_employe = ?, startdate = ?, enddate =
?, type = ? WHERE id = ?";

```

```

79         try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement (
            sql)){
80             stmt.setInt(1, e.getId_employe());
81             stmt.setDate(2, e.getStartDate());
82             stmt.setDate(3, e.getEndDate());
83             stmt.setString(4, e.getType().name());
84             stmt.setInt(5,e.getId_holiday());
85             stmt.executeUpdate();
86         } catch (SQLException exception) {
87             System.err.println("failed of update holiday");
88         } catch (ClassNotFoundException ex) {
89             System.err.println("failed connexion with data base");
90         }
91     }
92
93     @Override
94     public List<Holiday> display() {
95         String sql = "SELECT * FROM holiday";
96         List<Holiday> Holidays = new ArrayList<>();
97         try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement (
            sql)) {
98             ResultSet re = stmt.executeQuery();
99             while (re.next()) {
100                 int id = re.getInt("id");
101                 int id_employe = re.getInt("id_employe");
102                 Date startdate = re.getDate("startdate");
103                 Date enddate = re.getDate("enddate");
104                 String type = re.getString("type");
105                 Holiday e = new Holiday(id, id_employe, startdate, enddate,
                    Type_holiday.valueOf(type));
106                 Holidays.add(e);
107             }
108         } catch (ClassNotFoundException ex) {
109             System.err.println("Failed to connect with database: " + ex.getMessage
                ());
110         } catch (SQLException ex) {
111             System.err.println("Failed to fetch holidays: " + ex.getMessage());
112         }
113         return Holidays; // Retourne une liste vide si une erreur se produit
114     }
115
116 }

```

## 2.3 Controller

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

### HolidayInterface

```

1 package Controller;
2

```

```

3 import DAO.EmployeeDAOimpl;
4 import model.*;
5 import view.*;
6 import java.sql.Date;
7 import java.util.List;
8 import javax.swing.table.DefaultTableModel;
9
10
11 public class HolidayController {
12
13     private final Employee_HolidayView View;
14     public HolidayModel model_holiday;
15     public static int id = 0;
16     public static int oldselectedrow = -1;
17     public static boolean test = false;
18     int id_employe = 0;
19     String nom_employe = "";
20     public static String OldstartDate = null;
21     public static String OldendDate = null;
22     Type_holiday type = null;
23     int oldsolde = 0;
24     int solde = 0;
25     boolean updatereussi = false;
26     Employee targetEmployee = null;
27
28     public HolidayController(Employee_HolidayView view, HolidayModel model) {
29         this.View = view;
30         this.model_holiday= model;
31
32         View.getdeleteButton_holiday().addActionListener(e -> deleteHoliday());
33         View.getupdateButton_holiday().addActionListener(e -> updateHoliday());
34         Employee_HolidayView.Tableau1.getSelectionModel().addListSelectionListener(
35 e -> updateHolidaybyselect());
36         View.getaddButton_holiday().addActionListener(e -> addHoliday());
37         View.getdisplayButton_holiday().addActionListener(e -> displayHoliday());
38     }
39
40     private void addHoliday() {
41         int id_employe = View.getId_employe();
42         Date startDate = Date.valueOf(View.getStartDate());
43         Date endDate = Date.valueOf(View.getEndDate());
44         Type_holiday type = View.getType_holiday();
45
46         View.viderChamps_ho();
47
48         Employee targetEmployee = null;
49
50         // Rechercher l'employe correspondant
51         for (Employee employee : new Employemodell(new EmployeeDAOimpl()).
52 displayEmployee()) {
53             if (employee.getId() == id_employe) {
54                 targetEmployee = employee;
55                 break;
56             }
57         }
58     }
59 }

```

```
56
57     if (targetEmploye == null) {
58         View.afficherMessageErreur("Cet employe n'existe pas.");
59         return;
60     }
61
62     // Calculer la dur e du cong e demand
63     long daysBetween = java.time.temporal.ChronoUnit.DAYS.between(
64         startDate.toLocalDate(),
65         endDate.toLocalDate()
66     );
67
68     if (daysBetween <= 0) {
69         View.afficherMessageErreur("Les dates de d but et de fin sont
70 invalides.");
71         return;
72     }
73
74     // V rifier si le solde de cong e est suffisant
75     if (targetEmploye.getSolde() < daysBetween) {
76         View.afficherMessageErreur("Le solde de cong e de l'employe est
77 insuffisant.");
78         return;
79     }
80
81     try {
82         // Ajouter la demande de cong e
83         boolean addReussi = model_holiday.addHoliday(0, id_employe, startDate,
84 endDate, type, targetEmploye);
85
86         if (addReussi) {
87             // R duire le solde de cong e apr s l'ajout r ussi
88             targetEmploye.setSolde(targetEmploye.getSolde() - (int)
89 daysBetween);
90             View.afficherMessageSucces("Holiday a bien t ajout e.");
91         } else {
92             View.afficherMessageErreur("Holiday n'a pas t ajout e.");
93         }
94     } catch (Exception e) {
95         e.printStackTrace();
96         View.afficherMessageErreur("Erreur lors de l'ajout : " + e.getMessage
97 ());
98     }
99
100     }
101
102     private void displayHoliday() {
103         List<Holiday> Holidays = model_holiday.displayHoliday();
104
105         if (Holidays == null || Holidays.isEmpty()) {
106             View.afficherMessageErreur("Aucune holiday.");
107             return; // Retourner pour ne pas continuer l'ex cution si la liste
108 est vide
109         }
110
111         DefaultTableModel tableModel1 = (DefaultTableModel) Employee_HolidayView.
```

```

105     Tableau1.getModel();
106     tableModel1.setRowCount(0); // Clear existing rows in the table
107
108     for (Holiday e : Holidays) {
109         String nom_employe = null;
110         List<Employe> Employes = new Employemodell(new EmployeeDAOimpl()).
displayEmploye();
111         for (Employe em : Employes) {
112             if (em.getId() == e.getId_employe()) {
113                 nom_employe = em.getId() + " - " + em.getNom() + " " + em.
getPrenom();
114                 break;
115             }
116             // Ajout de la ligne dans le tableau
117             tableModel1.addRow(new Object[]{e.getId_holiday(), nom_employe, e.
getStartDate(), e.getEndDate(), e.getType()});
118         }
119         View.remplaire_les_employes();
120     }
121
122
123     private void deleteHoliday(){
124         int selectedrow = Employee_HolidayView.Tableau1.getSelectedRow();
125         if(selectedrow == -1){
126             View.afficherMessageErreur("Veuillez selectionner une ligne.");
127         }else{
128             int id = (int) Employee_HolidayView.Tableau1.getValueAt(selectedrow, 0)
;
129             int id_employe = Integer.parseInt((Employee_HolidayView.Tableau1.
getValueAt(selectedrow, 1)).toString().split(" - ")[0]);
130             int olddaysbetween = (int) ( (Date.valueOf(OldendDate).toLocalDate().
toEpochDay() - Date.valueOf(Oldstartdate).toLocalDate().toEpochDay()));
131             for(Employee e : new Employemodell(new EmployeeDAOimpl()).displayEmploye
()){
132                 if(e.getId() == id_employe){
133                     solde = e.getSolde();
134                     break;
135                 }
136             }
137             EmployeeController.updateSolde(id_employe, solde+olddaysbetween);
138             boolean deletereussi = model_holiday.deleteHoliday(id);
139             if(deletereussi){
140                 View.afficherMessageSucces("Holiday a bien ete supprimer.");
141                 displayHoliday();
142             }else{
143                 View.afficherMessageErreur("Holiday n'a pas ete supprimer.");
144             }
145         }
146     }
147
148     private void updateHolidaybyselect(){
149         int selectedrow = Employee_HolidayView.Tableau1.getSelectedRow();
150
151         if (selectedrow == -1) {

```



```

152         return;
153     }
154     try{
155         id = (int) Employee_HolidayView.Tableau1.getValueAt(selectedrow, 0);
156         nom_employe = (String) Employee_HolidayView.Tableau1.getValueAt(
selectedrow, 1);
157         id_employe = Integer.parseInt(nom_employe.split(" - ")[0]);
158         OldstartDate = String.valueOf(Employee_HolidayView.Tableau1.getValueAt(
selectedrow, 2));
159         OldendDate = String.valueOf(Employee_HolidayView.Tableau1.getValueAt(
selectedrow, 3));
160         type = (Type_holiday) Employee_HolidayView.Tableau1.getValueAt(
selectedrow, 4);
161         View.remplaireChamps_ho(id_employe, OldstartDate, OldendDate, type);
162         test = true;
163     }catch(Exception e){
164         View.afficherMessageErreur("Erreur lors de la r cup ration des
donn es");
165     }
166 }
167
168
169 private void updateHoliday(){
170     if (!test) {
171         View.afficherMessageErreur("Veuillez d'abord s lectionner une ligne
modifier.");
172         return;
173     }
174     try {
175         nom_employe = View.getNom();
176         Date startDate_holiday = Date.valueOf(View.getStartDate());
177         Date endDate_holiday = Date.valueOf(View.getEndDate());
178         type = View.getType_holiday();
179         id_employe = View.getId_employe();
180
181         int olddaysbetween = (int) ( (Date.valueOf(OldendDate).toLocalDate().
toEpochDay() - Date.valueOf(OldstartDate).toLocalDate().toEpochDay()));
182
183
184         for (Employee employee : new Employemodel(new EmployeeDAOimpl()).
displayEmployee()) {
185             if (employee.getId() == id_employe) {
186                 targetEmployee = employee;
187                 break;
188             }
189         }
190
191         boolean updateSuccessful = model_holiday.updateHoliday(id, id_employe,
startDate_holiday, endDate_holiday, type , targetEmployee , olddaysbetween);
192
193         if (updateSuccessful) {
194             test = false;
195             View.afficherMessageSucces("Holiday a t modifi avec succ s
.");
196             displayHoliday();

```

```
197         View.viderChamps_ho();
198     } else {
199         View.afficherMessageErreur("Erreur lors de la mise      jour de
holiday.");
200     }
201     } catch (Exception e) {
202
203         View.afficherMessageErreur("Erreur lors de la mise      jour");
204     }
205 }
206 }
```

## 2.4 View

Une View en Java désigne une partie visible de l'application, utilisée pour afficher des informations et inter-agir avec l'utilisateur, comme un bouton ou un champ de texte.

### **EmployeHolidayview**

```
1 package view;
2
3 import DAO.EmployeDAOimpl;
4 import model.Employe;
5 import model.Employemodel;
6 import model.Post;
7 import model.Role;
8 import model.Type_holiday;
9 import java.awt.*;
10 import javax.swing.*;
11 import javax.swing.table.DefaultTableModel;
12 import java.util.List;
13
14 public class Employe_HolidayView extends JFrame {
15
16     // le tableau de employe et cong
17     private JTabbedPane tabbedPane = new JTabbedPane();
18
19     // les tabs
20     private JPanel employeTab = new JPanel();
21     private JPanel holidayTab = new JPanel();
22
23     // les panels
24     private JPanel Employepan = new JPanel();
25     private JPanel Holidaypan = new JPanel();
26     private JPanel Display_Table_employe = new JPanel();
27     private JPanel Display_Table_holiday = new JPanel();
28     private final JPanel Forme_employe = new JPanel();
29     private final JPanel Forme_holiday = new JPanel();
30     private JPanel panButton_employe = new JPanel();
31     private JPanel panButton_holiday = new JPanel();
32
33     // les labels du l'employe
34     private JLabel label_nom = new JLabel("Nom");
```

```
35 private JLabel label_prenom = new JLabel("Prenom");
36 private JLabel label_email = new JLabel("Email");
37 private JLabel label_tele = new JLabel("Telephone");
38 private JLabel label_salaire = new JLabel("Salaire");
39 private JLabel label_role = new JLabel("Role");
40 private JLabel label_poste = new JLabel("Poste");
41
42 // les labels du cong
43 private JLabel label_employe = new JLabel("Nom de l'employe ");
44 private JLabel label_startDate = new JLabel("Date de debut (YYYY-MM-DD)");
45 private JLabel label_endDate = new JLabel("Date de fin (YYYY-MM-DD)");
46 private JLabel label_type = new JLabel("Type");
47 private JComboBox<Type_holiday> TypeComboBox = new JComboBox<>(Type_holiday.
values());
48
49 // les textfield du l'employe
50 private JTextField text_nom = new JTextField();
51 private JTextField text_prenom = new JTextField();
52 private JTextField text_email = new JTextField();
53 private JTextField text_tele = new JTextField();
54 private JTextField text_salaire = new JTextField();
55
56 private JComboBox<Role> roleComboBox = new JComboBox<>(Role.values());
57 private JComboBox<Post> posteComboBox = new JComboBox<>(Post.values());
58
59 // les textfield du cong
60 private JComboBox<String> text_employe = new JComboBox<>();
61 private JTextField text_startDate = new JTextField("");
62 private JTextField text_endDate = new JTextField("");
63
64 // les boutons du l'employe
65 private JButton addButton_employe = new JButton("Ajouter");
66 private JButton updateButton_employe = new JButton("Modifier");
67 private JButton deleteButton_employe = new JButton("Supprimer");
68 private JButton displayButton_employe = new JButton("Afficher");
69
70 // les boutons du cong
71 private JButton addButton_holiday = new JButton("Ajouter");
72 private JButton updateButton_holiday = new JButton("Modifier");
73 private JButton deleteButton_holiday = new JButton("Supprimer");
74 private JButton displayButton_holiday = new JButton("Afficher");
75
76
77 // le tableau de l'employe
78 JPanel pan0 = new JPanel(new BorderLayout());
79 public static String[] columnNames_employe = {"ID", "Nom", "Prenom", "Email",
"T l phone", "Salaire", "Role", "Poste", "solde"};
80 public static DefaultTableModel tableModel = new DefaultTableModel(
columnNames_employe, 0);
81 public static JTable Tableau = new JTable(tableModel);
82
83 // le tableau du cong
84 JPanel pan1 = new JPanel(new BorderLayout());
85 public static String[] columnNames_holiday = {"ID", "nom_employe", "date_debut
", "date_fin", "type"};
```

```

86     public static DefaultTableModel tableModell = new DefaultTableModel (
columnNames_holiday, 0);
87     public static JTable Tableau1 = new JTable(tableModell);
88
89     public Employee_HolidayView() {
90
91         setTitle("Gestion des employes et des cong s");
92         setSize(1000, 600);
93         setDefaultCloseOperation(EXIT_ON_CLOSE);
94         setLocationRelativeTo(null);
95
96         add(tabbedPane);
97
98         // Employee Tab
99         employeeTab.setLayout(new BorderLayout());
100        employeeTab.add(Employepan, BorderLayout.CENTER);
101
102        Employepan.setLayout(new BorderLayout());
103        Employepan.add(Display_Table_employe, BorderLayout.CENTER);
104        Tableau.setFillViewportHeight(true);
105        Dimension preferredSize = new Dimension(900, 500);
106        Tableau.setPreferredScrollableViewportSize(preferredSize);
107        pan0.add(new JScrollPane(Tableau), BorderLayout.CENTER);
108        Display_Table_employe.add(pan0);
109
110        Employepan.add(panButton_employe, BorderLayout.SOUTH);
111        panButton_employe.add(addButton_employe);
112        panButton_employe.add(updateButton_employe);
113        panButton_employe.add(deleteButton_employe);
114        panButton_employe.add(displayButton_employe);
115
116        Employepan.add(Forme_employe, BorderLayout.NORTH);
117        Forme_employe.setLayout(new GridLayout(7, 2, 10, 10));
118        Forme_employe.add(label_nom);
119        Forme_employe.add(text_nom);
120        Forme_employe.add(label_prenom);
121        Forme_employe.add(text_prenom);
122        Forme_employe.add(label_email);
123        Forme_employe.add(text_email);
124        Forme_employe.add(label_tele);
125        Forme_employe.add(text_tele);
126        Forme_employe.add(label_salaire);
127        Forme_employe.add(text_salaire);
128        Forme_employe.add(label_role);
129        Forme_employe.add(roleComboBox);
130        Forme_employe.add(label_poste);
131        Forme_employe.add(posteComboBox);
132
133        // Holiday Tab
134        holidayTab.setLayout(new BorderLayout());
135        holidayTab.add(Holidaypan, BorderLayout.CENTER);
136        Holidaypan.setLayout(new BorderLayout());
137        Holidaypan.add(Display_Table_holiday, BorderLayout.CENTER);
138
139        Tableau1.setFillViewportHeight(true);

```

```

140     Tableaul.setPreferredScrollableViewportSize(preferredSize);
141     panl.add(new JScrollPane(Tableaul), BorderLayout.CENTER);
142     Display_Table_holiday.add(panl);
143
144     Holidaypan.add(Forme_holiday, BorderLayout.NORTH);
145     Forme_holiday.setLayout(new GridLayout(4, 2, 10, 10));
146     Forme_holiday.add(label_employe);
147     Forme_holiday.add(text_employe);
148     Forme_holiday.add(label_startDate);
149     Forme_holiday.add(text_startDate);
150     Forme_holiday.add(label_endDate);
151     Forme_holiday.add(text_endDate);
152     Forme_holiday.add(label_type);
153     Forme_holiday.add(TypeComboBox);
154
155     Holidaypan.add(panButton_holiday, BorderLayout.SOUTH);
156     panButton_holiday.add(addButton_holiday);
157     panButton_holiday.add(updateButton_holiday);
158     panButton_holiday.add(deleteButton_holiday);
159     panButton_holiday.add(displayButton_holiday);
160
161     // TabbedPane
162     tabbedPane.addTab("Employe", employeTab);
163     tabbedPane.addTab("Holiday", holidayTab);
164
165
166     remplaceire_les_employes();
167     setVisible(true);
168 }
169
170 public void remplaceire_les_employes () {
171     List<Employe> Employes = new Employemodell(new EmployeeDAOimpl()).
displayEmploye();
172     text_employe.removeAllItems();
173     for (Employe elem : Employes) {
174         text_employe.addItem(elem.getId() + " - " + elem.getNom()+" "+elem.
getPrenom());
175     }
176 }
177
178
179
180 // getters
181
182
183     public int getId_employe() {
184         return Integer.parseInt(text_employe.getSelectedItem().toString().
split(" - ")[0]);
185     }
186     public String getNom() {
187         return text_nom.getText();
188     }
189
190     public JTable getTable() {
191         return (JTable) Display_Table_employe.getComponent(0);

```

```
192     }
193
194     public String getPrenom() {
195         return text_prenom.getText();
196     }
197
198     public String getEmail() {
199         return text_email.getText();
200     }
201
202     public String getTelephone() {
203         return text_tele.getText();
204     }
205
206     public double getSalaire() {
207         return Double.parseDouble(text_salaire.getText());
208     }
209
210     public Role getRole() {
211         return (Role) roleComboBox.getSelectedItem();
212     }
213
214     public Post getPoste() {
215         return (Post) posteComboBox.getSelectedItem();
216     }
217
218     public JButton getaddButton_employe () {
219         return addButton_employe;
220     }
221
222     public JButton getupdateButton_employe () {
223         return updateButton_employe;
224     }
225
226     public JButton getdeleteButton_employe () {
227         return deleteButton_employe;
228     }
229
230     public JButton getdisplayButton_employe () {
231         return displayButton_employe;
232     }
233
234     public JButton getaddButton_holiday () {
235         return addButton_holiday;
236     }
237
238     public JButton getupdateButton_holiday () {
239         return updateButton_holiday;
240     }
241     public JButton getdeleteButton_holiday () {
242         return deleteButton_holiday;
243     }
244
245     public JButton getdisplayButton_holiday () {
246         return displayButton_holiday;
```

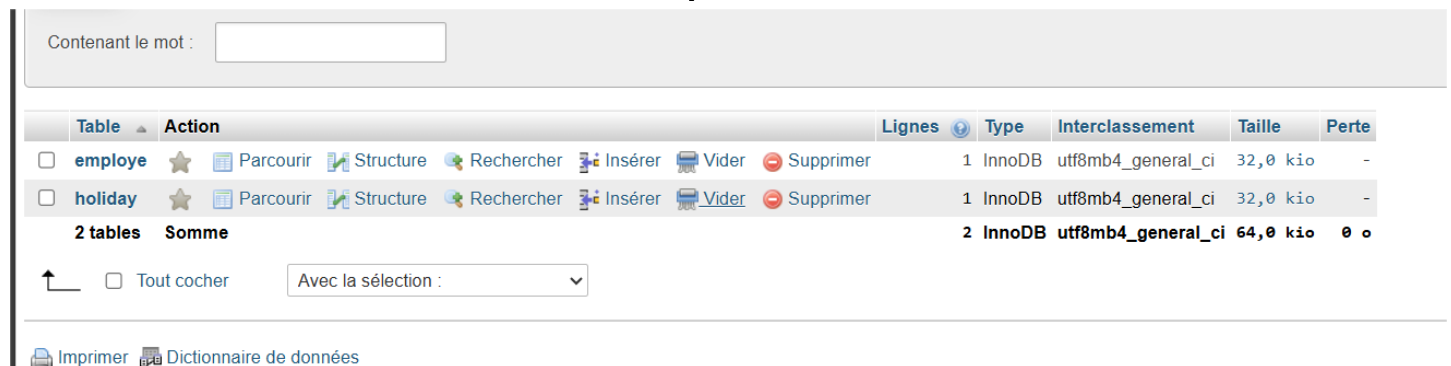
```
247     }
248     public String getStartDate () {
249         return text_startDate.getText ();
250     }
251
252     public String getEndDate() {
253         return text_endDate.getText ();
254     }
255
256     public Type_holiday getType_holiday(){
257         return (Type_holiday) TypeComboBox.getSelectedItemAt();
258     }
259
260     // methods d'affichage des messages
261     public void afficherMessageErreur(String message) {
262         JOptionPane.showMessageDialog(this, message, "Erreur", JOptionPane.
ERROR_MESSAGE);
263     }
264
265     public void afficherMessageSucces(String message) {
266         JOptionPane.showMessageDialog(this, message, "Succ s", JOptionPane.
INFORMATION_MESSAGE);
267     }
268
269     // methodes de vider les champs
270     public void viderChamps_em() {
271         text_nom.setText("");
272         text_prenom.setText("");
273         text_email.setText("");
274         text_tele.setText("");
275         text_salaire.setText("");
276         roleComboBox.setSelectedIndex(0);
277         posteComboBox.setSelectedIndex(0);
278     }
279
280     public void viderChamps_ho() {
281         text_startDate.setText("");
282         text_endDate.setText("");
283         TypeComboBox.setSelectedIndex(0);
284     }
285
286     // methodes de remplir les champs
287     public void remplaireChamps_em (int id, String nom, String prenom, String
email, String telephone, double salaire, Role role, Post poste) {
288         text_nom.setText (nom);
289         text_prenom.setText (prenom);
290         text_email.setText (email);
291         text_tele.setText (telephone);
292         text_salaire.setText (String.valueOf(salaire));
293         roleComboBox.setSelectedItem(role);
294         posteComboBox.setSelectedItem(poste);
295     }
296
297     public void remplaireChamps_ho(int id_employe, String date_debut, String
date_fin, Type_holiday type) {
```

```
298         List<Employee> Employees = new Employemodel(new EmployeeDAOimpl()).
displayEmploye();
299         text_employe.removeAllItems();
300         for (Employee elem : Employees) {
301             if (elem.getId() == id_employe) {
302                 text_employe.addItem(elem.getId() + " - " + elem.getNom()+" "+
elem.getPrenom());
303                 text_employe.setSelectedItem(elem.getId() + " - " + elem.
getNom()+" "+elem.getPrenom());
304             }
305         }
306         text_startDate.setText(date_debut);
307         text_endDate.setText(date_fin);
308         TypeComboBox.setSelectedItem(type);
309     }
310
311     // methodes de test des champs
312     public boolean testChampsVide_em () {
313         return text_nom.getText().equals("") || text_prenom.getText().equals
("") || text_email.getText().equals("") || text_tele.getText().equals("") ||
text_salaire.getText().equals("");
314     }
315
316     public boolean testChampsVide_ho () {
317         return text_employe.getSelectedItem().equals("") || text_startDate.
getText().equals("") || text_endDate.getText().equals("") || TypeComboBox.
getSelectedItem().equals("");
318     }
319
320 }
```



# Résultats

## 1 Tables Créées



Contenant le mot :

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> employe	★ Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	32,0 kio	-
<input type="checkbox"/> holiday	★ Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	32,0 kio	-
2 tables	Somme	2	InnoDB	utf8mb4_general_ci	64,0 kio	0 o

⬆ ☐ Tout cocher Avec la sélection :

Imprimer Dictionnaire de données

FIGURE 2.1 – Tables de la base de donnée

## 2 Résultats de la partie View

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

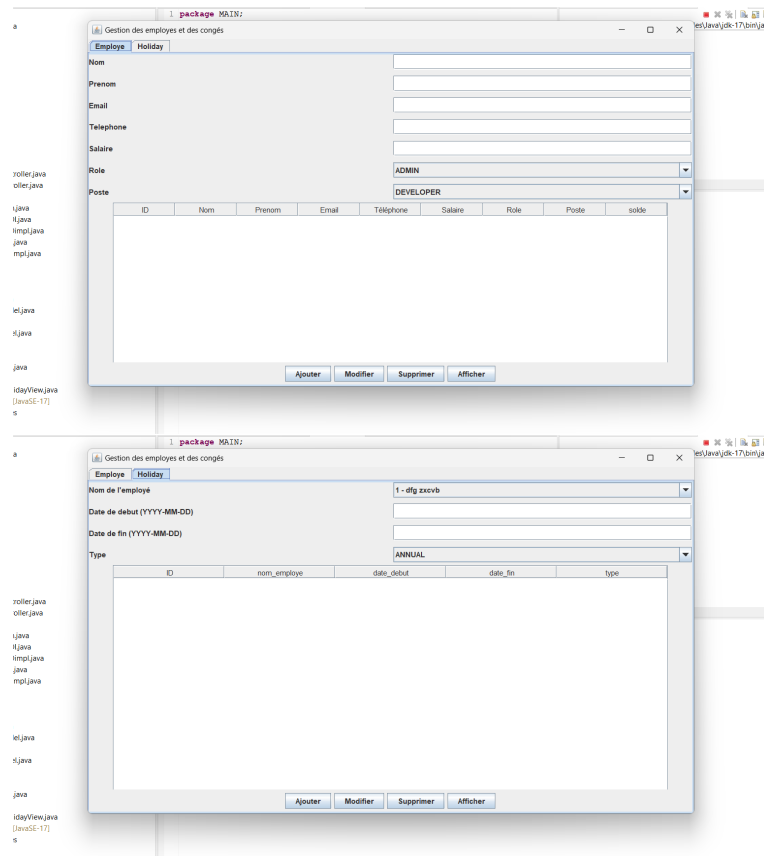


FIGURE 2.2 – Interface Utilisateur

### 3 Après Ajout

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

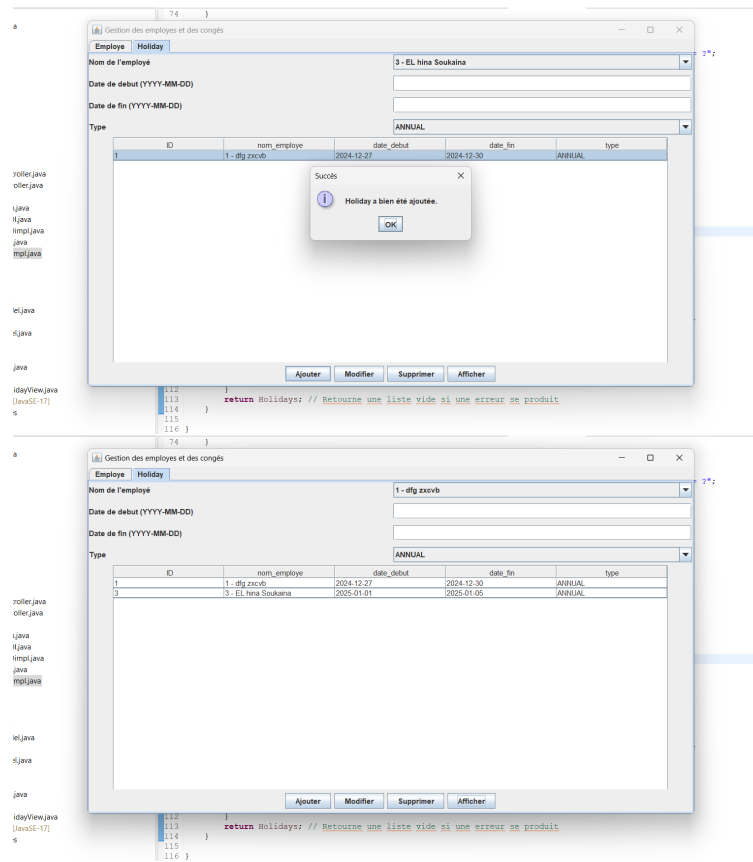


FIGURE 2.3 – Resultat Ajout

## 4 Après modification

Après la mise à jour d'un congé, les nouvelles informations saisies par l'utilisateur dans le panneau d'entrée sont validées et transmises à la couche Controller, qui assure leur traitement via la logique métier. Une fois la modification effectuée avec succès, la liste des congés est immédiatement actualisée dans le panneau d'affichage, garantissant ainsi une visualisation en temps réel des données mises à jour.

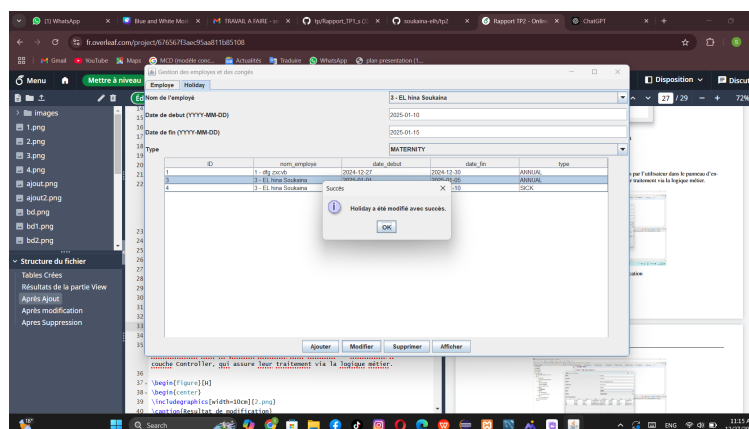


FIGURE 2.4 – Resultat de modification

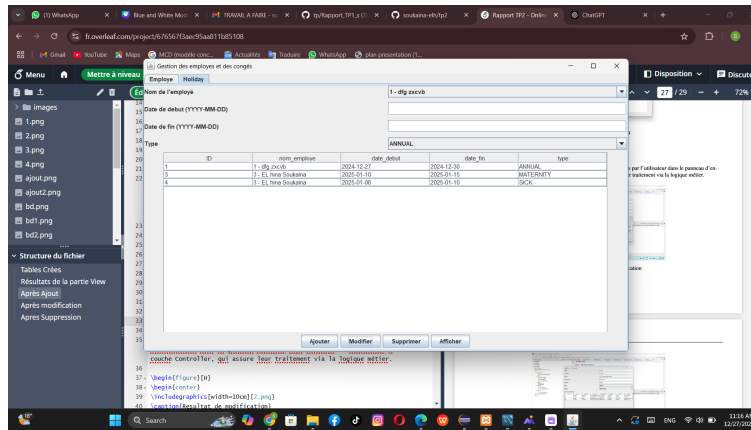


FIGURE 2.5 – Affichage de modification

## 5 Après Suppression

Lorsqu'un congé est supprimé, l'utilisateur sélectionne le congé concerné dans la liste affichée et confirme l'action en cliquant sur le bouton Supprimer. Cette demande est transmise à la couche Controller, qui s'assure de la suppression de l'enregistrement via la logique métier. Une fois l'opération terminée, la liste des congés est automatiquement mise à jour pour refléter les modifications en temps réel.

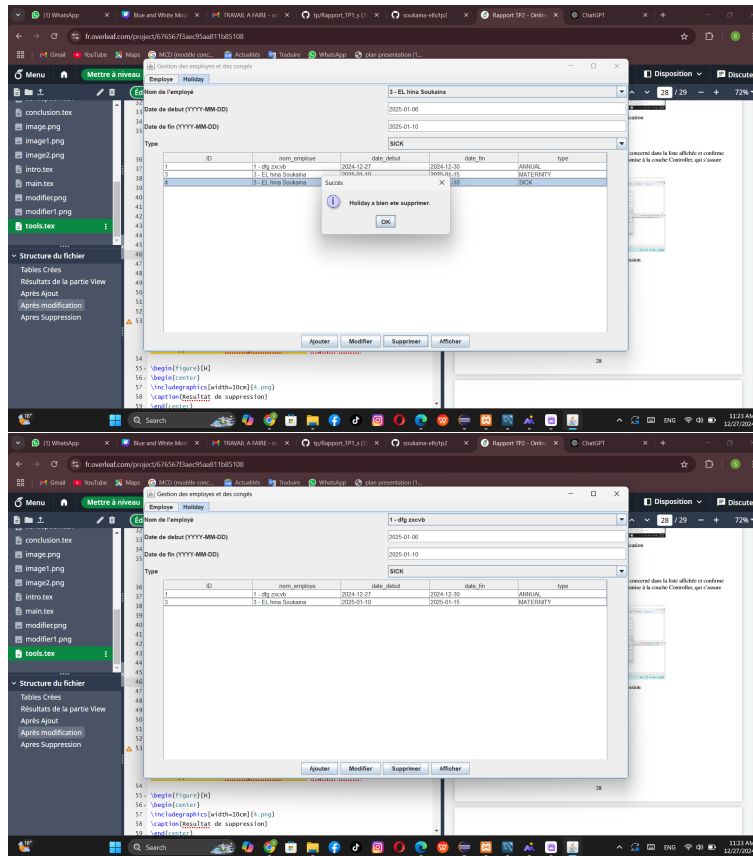


FIGURE 2.6 – Resultat de suppression



# Conclusion générale

Ce travail pratique (TP) a permis de concevoir et de développer une application de gestion des congés des employés en s'appuyant sur l'architecture MVC. Cette structure a facilité la séparation des responsabilités entre la gestion des données, la logique métier et l'interface utilisateur, garantissant ainsi une application modulaire, maintenable et évolutive.

L'implémentation des fonctionnalités principales, telles que la gestion des employés et des congés, a renforcé notre compréhension des principes de la programmation orientée objet, des génériques, et des bonnes pratiques de développement. De plus, l'utilisation de la bibliothèque Swing a permis de créer une interface graphique intuitive, favorisant une meilleure expérience utilisateur.

Ce TP illustre l'importance d'adopter une approche structurée dans la conception et le développement de logiciels, tout en mettant en avant l'impact d'une architecture bien définie sur la qualité et la pérennité des applications.

# Références

*java :*

— <https://www.java.com/en/download/>

*Eclipse :*

— <https://eclipse.fr.softonic.com/>

*XAMPP :*

— <https://www.apachefriends.org/fr/index.html>

*jdk 23 :*

— <https://www.oracle.com/java/technologies/downloads/>