Université Cadi Ayyad Marrakech

Ecole Supérieure de technologie-safi

Département : Informatique

Filière : Génie Informatique (GI)

TP Java Avance

Gestion des employée et des Conges

Effectué par :	*Encadré par :*
Louiass Salma	**Mme.Asmaa Elkourchi**

Année universitaire : **2023-2024**

Lien GitHub : https://github.com/Salma12louiass/Gestion_employee

Table des matières

Inroduction

Ce travail pratique (TP) vise le développement d'une application Java de gestion des employés et de leurs congés, en adoptant l'architecture MVC (Modèle-Vue-Contrôleur) et en intégrant des interfaces graphiques à l'aide de la bibliothèque Swing.

Ce projet s'inscrit dans une démarche pédagogique combinant l'apprentissage des bases de la programmation orientée objet (POO) et des bonnes pratiques de conception logicielle. L'objectif est de concevoir une application intuitive et modulaire, permettant de gérer efficacement les données des employés ainsi que leurs congés à travers des fonctionnalités telles que l'ajout, la modification, la suppression et l'affichage des informations, avec une prise en charge des demandes et états de congés.

Grâce à une structure bien définie et à une séparation claire des responsabilités, ce TP offre une opportunité d'approfondir les notions de maintenabilité et d'évolutivité, tout en préparant les étudiants à concevoir des applications logicielles complètes et adaptées aux besoins réels des entreprises.

Outils & environnement de travail

1 Environnement de travail

- **IntelliJ IDEA** : est un IDE développé par JetBrains, conçu principalement pour Java, mais prenant en charge d'autres langages comme Kotlin, Scala et Python. Il se distingue par ses outils intelligents, sa complétion de code avancée, son analyse en temps réel et son intégration avec des outils comme Maven, Gradle et Git, offrant une productivité accrue aux développeurs.

2 Outils de travail

- **MySQL Workbench** : est un outil graphique développé pour simplifier la conception, l'administration et la gestion des bases de données MySQL. Il offre une interface intuitive qui permet de manipuler les bases de données visuellement, sans dépendre uniquement des commandes en ligne, rendant les tâches plus accessibles et efficaces.

- **XAMPP** : est une solution open-source qui regroupe Apache, MySQL, PHP, et Perl pour faciliter le développement et le déploiement d'applications web. Il est conçu pour créer un environnement de serveur local rapide et simple, adapté aux développeurs souhaitant tester ou gérer des projets web sur leur machine avant le déploiement.

- **java developpement kit** : est un ensemble d'outils logiciels indispensables pour le développement d'applications Java. Il comprend des composants essentiels tels qu'un compilateur, une machine virtuelle Java (JVM), des bibliothèques standard, et des outils de débogage, permettant de coder, compiler, exécuter et tester des programmes Java efficacement.

3 Language de Programmation

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

Réalisation

1 Création de la base de donnée

1.1 Script base de donnée

```sql
-- phpMyAdmin SQL Dump
-- version 5.2.1
-- https://www.phpmyadmin.net/
--
-- Hôte : 127.0.0.1
-- Généré le : jeu. 19 déc. 2024 à 23:13
-- Version du serveur : 10.4.32-MariaDB
-- Version de PHP : 8.2.12

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Base de données : `tp7`
--

-- --------------------------------------------------------

--
-- Structure de la table `employe`
--

CREATE TABLE `employe` (
  `id` int(11) NOT NULL,
  `nom` varchar(100) DEFAULT NULL,
```

```
  `prenom` varchar(100) DEFAULT NULL,
  `email` varchar(100) DEFAULT NULL,
  `phone` varchar(20) DEFAULT NULL,
  `salaire` decimal(10,2) DEFAULT NULL,
  `role` enum('ADMIN','EMPLOYE') NOT NULL,
  `poste` enum('INGENIEURE_ETUDE_ET_DEVELOPPEMENT','TEAM_LEADER','PILOTE') NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Déchargement des données de la table `employe`
--

INSERT INTO `employe` (`id`, `nom`, `prenom`, `email`, `phone`, `salaire`, `role`, `poste`)
VALUES
(2, 'AMRANI', 'Nawal', 'nwlam@email.com', '0623456789', 7000.00, 'EMPLOYE',
'INGENIEURE_ETUDE_ET_DEVELOPPEMENT'),
(3, 'El Yassir', 'Rachid', 'rachid.elyassir@email.com', '0612345678', 5000.00, 'EMPLOYE',
'TEAM_LEADER');

-- --------------------------------------------------------

--
-- Structure de la table `holiday`
--

CREATE TABLE `holiday` (
  `id` int(11) NOT NULL,
  `employeeId` int(11) DEFAULT NULL,
  `startDate` date DEFAULT NULL,
  `endDate` date DEFAULT NULL,
  `type` enum('CONGE_MALADIE','CONGE_PAYE','CONGE_NON_PAYE') NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Déchargement des données de la table `holiday`
--

INSERT INTO `holiday` (`id`, `employeeId`, `startDate`, `endDate`, `type`) VALUES
(3, 2, '2024-12-01', '2024-12-10', 'CONGE_PAYE'),
(5, 3, '2024-12-15', '2024-12-20', 'CONGE_MALADIE');

--
-- Index pour les tables déchargées
--

--
```

```sql
-- Index pour la table `employe`
--
ALTER TABLE `employe`
  ADD PRIMARY KEY (`id`);

--
-- Index pour la table `holiday`
--
ALTER TABLE `holiday`
  ADD PRIMARY KEY (`id`),
  ADD KEY `employeeId` (`employeeId`);

--
-- AUTO_INCREMENT pour les tables déchargées
--

--
-- AUTO_INCREMENT pour la table `employe`
--
ALTER TABLE `employe`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;

--
-- AUTO_INCREMENT pour la table `holiday`
--
ALTER TABLE `holiday`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;

--
-- Contraintes pour les tables déchargées
--

--
-- Contraintes pour la table `holiday`
--
ALTER TABLE `holiday`
  ADD CONSTRAINT `holiday_ibfk_1` FOREIGN KEY (`employeeId`) REFERENCES `employe` (`id`) ON
DELETE CASCADE;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

2 Architecture MVC (Model-View-Controller)

2.1 Model:

Employee:

```
package Model;

public class Employee {
    private int id;
    private String nom;
    private String prenom;
    private String email;
    private String phone;
    private double salaire;
    private Role role;
    private Poste poste;
    private int solde;

    public Employee(String nom, String prenom, String email, String phone, double salaire, Role
role, Poste poste) {

        this.nom = nom;
        this.prenom = prenom;
        this.email = email;
        this.phone = phone;
        this.salaire = salaire;
        this.role = role;
        this.poste = poste;
    }

    public Employee() {

    }

    public Employee(String string, String string2, String string3, String string4, double
double1, int int1, Role role2, Poste poste2) {
        this.nom = string;
        this.prenom = string2;
        this.email = string3;
        this.phone = string4;
        this.salaire = double1;
```

```
        this.solde=int1;
        this.role = role2;
        this.poste = poste2;  }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }
    public String getPrenom() { return prenom; }
    public void setPrenom(String prenom) { this.prenom = prenom; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone = phone; }
    public double getSalaire() { return salaire; }
    public void setSalaire(double salaire) { this.salaire = salaire; }
    public Role getRole() { return role; }
    public void setRole(Role role) { this.role = role; }
    public Poste getPoste() { return poste; }
    public void setPoste(Poste poste) { this.poste = poste; }
    public int getSolde() {
        return solde;
    }
    public void setnom(String nouveauNom) {
    }
}
```

Holiday :

```
package Model;

public class Holiday {
    private int id; // Identifiant du congé
    private int employeeId; // ID de l'employé
    private String employeeName; // Nom complet de l'employé
    private String startDate; // Date de début
    private String endDate;   // Date de fin
    private Type type;        // Type de congé (enum)

    // Constructeur avec employeeName pour listAll()
    public Holiday(int id, String employeeName, String startDate, String endDate, Type type) {
        this.id = id;
        this.employeeName = employeeName;
        this.startDate = startDate;
        this.endDate = endDate;
```

```
        this.type = type;
    }
    public Holiday(String employeeName, String startDate, String endDate, Type type) {
        this.employeeName = employeeName;
        this.startDate = startDate;
        this.endDate = endDate;
        this.type = type;
    }

    // Constructeur pour add() et update() (sans employeeName)
    public Holiday(int employeeId, String startDate, String endDate, Type type) {
        this.employeeId = employeeId;
        this.startDate = startDate;
        this.endDate = endDate;
        this.type = type;
    }

    // Getters et Setters
    public int getId() {
        return id;
    }

    public int getEmployeeId() {
        return employeeId;
    }

    public String getEmployeeName() {
        return employeeName; // Retourne le nom complet
    }

    public String getStartDate() {
        return startDate;
    }

    public String getEndDate() {
        return endDate;
    }

    public Type getType() {
        return type;
    }

    public void setEmployeeName(String employeeName) {
        this.employeeName = employeeName;
    }
}
```

Poste :

```
package Model;

public enum Poste {
    INGENIEURE_ETUDE_ET_DEVELOPPEMENT,
    TEAM_LEADER,
    PILOTE
}
```

Role :

```
package Model;


public enum Role{
    ADMIN,
    EMPLOYE
}
```

Type :

```


package Model;


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

2.2 View

EmployeeView :

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package View;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
```

```java
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;

public class EmployeeView extends JFrame {
    public JTable employeeTable;
    public JButton addButton;
    public JButton listButton;
    public JButton deleteButton;
    public JButton modifyButton;
    public JButton switchViewButton;
    public JTextField nameField;
    public JTextField surnameField;
    public JTextField emailField;
    public JTextField phoneField;
    public JTextField salaryField;
    public JComboBox<String> roleCombo;
    public JComboBox<String> posteCombo;

    public EmployeeView() {
        this.setTitle("Gestion des Employés");
        this.setSize(900, 600);
        this.setDefaultCloseOperation(3);
        this.setLayout(new BorderLayout());
        JPanel headerPanel = new JPanel(new BorderLayout());
        headerPanel.setBackground(new Color(58, 115, 173));
        JLabel headerLabel = new JLabel("Gestion des Employés", 0);
        headerLabel.setForeground(Color.WHITE);
        headerLabel.setFont(new Font("Arial", 1, 24));
        this.switchViewButton = this.createStyledButton("Gérer les Congés");
        this.switchViewButton.setBackground(new Color(255, 204, 102));
        this.switchViewButton.setForeground(Color.BLACK);
        headerPanel.add(headerLabel, "Center");
        headerPanel.add(this.switchViewButton, "East");
        this.add(headerPanel, "North");
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, 1));
        mainPanel.setBorder(new EmptyBorder(20, 20, 20, 20));
        mainPanel.setBackground(new Color(245, 245, 245));
        JPanel inputPanel = new JPanel(new GridLayout(0, 2, 10, 10));
        inputPanel.setBackground(new Color(245, 245, 245));
        inputPanel.add(new JLabel("Nom:"));
        this.nameField = new JTextField();
        this.styleTextField(this.nameField);
```

```
        inputPanel.add(this.nameField);
        inputPanel.add(new JLabel("Prénom:"));
        this.surnameField = new JTextField();
        this.styleTextField(this.surnameField);
        inputPanel.add(this.surnameField);
        inputPanel.add(new JLabel("Email:"));
        this.emailField = new JTextField();
        this.styleTextField(this.emailField);
        inputPanel.add(this.emailField);
        inputPanel.add(new JLabel("Téléphone:"));
        this.phoneField = new JTextField();
        this.styleTextField(this.phoneField);
        inputPanel.add(this.phoneField);
        inputPanel.add(new JLabel("Salaire:"));
        this.salaryField = new JTextField();
        this.styleTextField(this.salaryField);
        inputPanel.add(this.salaryField);
        inputPanel.add(new JLabel("Rôle:"));
        this.roleCombo = new JComboBox(new String[]{"Admin", "Employe"});
        this.styleComboBox(this.roleCombo);
        inputPanel.add(this.roleCombo);
        inputPanel.add(new JLabel("Poste:"));
        this.posteCombo = new JComboBox(new String[]{"INGENIEURE_ETUDE_ET_DEVELOPPEMENT",
"TEAM_LEADER", "PILOTE"});
        this.styleComboBox(this.posteCombo);
        inputPanel.add(this.posteCombo);
        mainPanel.add(inputPanel);
        JPanel tablePanel = new JPanel(new BorderLayout());
        tablePanel.setBorder(new EmptyBorder(10, 0, 10, 0));
        tablePanel.setBackground(Color.WHITE);
        JLabel tableTitle = new JLabel("Liste des Employés", 0);
        tableTitle.setFont(new Font("Arial", 1, 16));
        tablePanel.add(tableTitle, "North");
        this.employeeTable = new JTable();
        JScrollPane tableScrollPane = new JScrollPane(this.employeeTable);
        tablePanel.add(tableScrollPane, "Center");
        mainPanel.add(Box.createVerticalStrut(20));
        mainPanel.add(tablePanel);
        this.add(mainPanel, "Center");
        JPanel buttonPanel = new JPanel(new FlowLayout(1, 15, 10));
        buttonPanel.setBackground(new Color(240, 240, 240));
        this.addButton = this.createStyledButton("Ajouter");
        buttonPanel.add(this.addButton);
        this.listButton = this.createStyledButton("Afficher");
        buttonPanel.add(this.listButton);
        this.deleteButton = this.createStyledButton("Supprimer");
        buttonPanel.add(this.deleteButton);
        this.modifyButton = this.createStyledButton("Modifier");
        buttonPanel.add(this.modifyButton);
        this.add(buttonPanel, "South");
    }

    private void styleTextField(JTextField textField) {
        textField.setFont(new Font("Arial", 0, 14));

textField.setBorder(BorderFactory.createCompoundBorder(BorderFactory.createLineBorder(new
Color(200, 200, 200)), BorderFactory.createEmptyBorder(5, 5, 5, 5)));
    }
```

```
    private void styleComboBox(JComboBox<String> comboBox) {
        comboBox.setFont(new Font("Arial", 0, 14));
        comboBox.setBackground(Color.WHITE);
        comboBox.setBorder(BorderFactory.createLineBorder(new Color(200, 200, 200)));
    }

    private JButton createStyledButton(String text) {
        final JButton button = new JButton(text);
        button.setFont(new Font("Arial", 0, 14));
        button.setBackground(new Color(70, 130, 180));
        button.setForeground(Color.WHITE);
        button.setFocusPainted(false);
        button.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
        button.setCursor(Cursor.getPredefinedCursor(12));
        button.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent evt) {
                button.setBackground(new Color(100, 160, 210));
            }

            public void mouseExited(MouseEvent evt) {
                button.setBackground(new Color(70, 130, 180));
            }
        });
        return button;
    }
}
```

HolidayView :

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package View;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;
```

```
public class HolidayView extends JFrame {
    public JTable holidayTable;
    public JButton addButton;
    public JButton listButton;
    public JButton deleteButton;
    public JButton modifyButton;
    public JButton switchViewButton;
    public JComboBox<String> employeeNameComboBox;
    public JTextField startDateField;
    public JTextField endDateField;
    public JComboBox<String> typeCombo;

    public HolidayView() {
        this.setTitle("Gestion des Congés");
        this.setSize(900, 600);
        this.setDefaultCloseOperation(3);
        this.setLayout(new BorderLayout());
        JPanel headerPanel = new JPanel(new BorderLayout());
        headerPanel.setBackground(new Color(58, 115, 173));
        JLabel headerLabel = new JLabel("Gestion des Congés", 0);
        headerLabel.setForeground(Color.WHITE);
        headerLabel.setFont(new Font("Arial", 1, 24));
        this.switchViewButton = this.createStyledButton("Gérer les Employés");
        this.switchViewButton.setBackground(new Color(255, 204, 102));
        this.switchViewButton.setForeground(Color.BLACK);
        headerPanel.add(headerLabel, "Center");
        headerPanel.add(this.switchViewButton, "East");
        this.add(headerPanel, "North");
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, 1));
        mainPanel.setBorder(new EmptyBorder(20, 20, 20, 20));
        mainPanel.setBackground(new Color(245, 245, 245));
        JPanel inputPanel = new JPanel(new GridLayout(0, 2, 10, 10));
        inputPanel.setBackground(new Color(245, 245, 245));
        inputPanel.add(new JLabel("Employé Nom Complet:"));
        this.employeeNameComboBox = new JComboBox();
        this.styleComboBox(this.employeeNameComboBox);
        inputPanel.add(this.employeeNameComboBox);
        inputPanel.add(new JLabel("Date Début:"));
        this.startDateField = new JTextField();
        this.styleTextField(this.startDateField);
        inputPanel.add(this.startDateField);
        inputPanel.add(new JLabel("Date Fin:"));
        this.endDateField = new JTextField();
        this.styleTextField(this.endDateField);
        inputPanel.add(this.endDateField);
        inputPanel.add(new JLabel("Type de Congé:"));
        this.typeCombo = new JComboBox(new String[]{"CONGE_PAYE", "CONGE_MALADIE",
"CONGE_NON_PAYE"});
        this.styleComboBox(this.typeCombo);
        inputPanel.add(this.typeCombo);
        mainPanel.add(inputPanel);
        JPanel tablePanel = new JPanel(new BorderLayout());
        tablePanel.setBorder(new EmptyBorder(10, 0, 10, 0));
        tablePanel.setBackground(Color.WHITE);
        JLabel tableTitle = new JLabel("Liste des Congés", 0);
        tableTitle.setFont(new Font("Arial", 1, 16));
```

```
        tablePanel.add(tableTitle, "North");
        this.holidayTable = new JTable();
        JScrollPane tableScrollPane = new JScrollPane(this.holidayTable);
        tablePanel.add(tableScrollPane, "Center");
        mainPanel.add(Box.createVerticalStrut(20));
        mainPanel.add(tablePanel);
        this.add(mainPanel, "Center");
        JPanel buttonPanel = new JPanel(new FlowLayout(1, 15, 10));
        buttonPanel.setBackground(new Color(240, 240, 240));
        this.addButton = this.createStyledButton("Ajouter");
        buttonPanel.add(this.addButton);
        this.listButton = this.createStyledButton("Afficher");
        buttonPanel.add(this.listButton);
        this.deleteButton = this.createStyledButton("Supprimer");
        buttonPanel.add(this.deleteButton);
        this.modifyButton = this.createStyledButton("Modifier");
        buttonPanel.add(this.modifyButton);
        this.add(buttonPanel, "South");
    }

    private void styleTextField(JTextField textField) {
        textField.setFont(new Font("Arial", 0, 14));

textField.setBorder(BorderFactory.createCompoundBorder(BorderFactory.createLineBorder(new
Color(200, 200, 200)), BorderFactory.createEmptyBorder(5, 5, 5, 5)));
    }

    private void styleComboBox(JComboBox<String> comboBox) {
        comboBox.setFont(new Font("Arial", 0, 14));
        comboBox.setBackground(Color.WHITE);
        comboBox.setBorder(BorderFactory.createLineBorder(new Color(200, 200, 200)));
    }

    private JButton createStyledButton(String text) {
        final JButton button = new JButton(text);
        button.setFont(new Font("Arial", 0, 14));
        button.setBackground(new Color(70, 130, 180));
        button.setForeground(Color.WHITE);
        button.setFocusPainted(false);
        button.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
        button.setCursor(Cursor.getPredefinedCursor(12));
        button.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent evt) {
                button.setBackground(new Color(100, 160, 210));
            }

            public void mouseExited(MouseEvent evt) {
                button.setBackground(new Color(70, 130, 180));
            }
        });
        return button;
    }
}
```

2.3 Controller

EmployeeController :

```java
package Controller;

import DAO.GenericDAO;
import DAO.EmployeeDAOImpl;
import Model.Employee;
import Model.Poste;
import Model.Role;
import View.EmployeeView;
import View.HolidayView;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

public class EmployeeController {
    private final EmployeeView view;
    private final GenericDAO<Employee> dao;
    private final HolidayView holidayView;

    public EmployeeController(EmployeeView view, HolidayView holidayView) {
        this.view = view;
        this.dao = new EmployeeDAOImpl();
        this.holidayView = holidayView;

        // Écouteur pour le bouton Ajouter
        view.addButton.addActionListener(e -> addEmployee());

        // Écouteur pour le bouton Lister
        view.listButton.addActionListener(e -> listEmployees());

        // Écouteur pour le bouton Supprimer
        view.deleteButton.addActionListener(e -> deleteEmployee());

        // Écouteur pour le bouton Modifier
        view.modifyButton.addActionListener(e -> modifyEmployee());

        // ActionListener pour le bouton "Gérer les Congés"
        view.switchViewButton.addActionListener(e -> {
            view.setVisible(false); // Cacher la vue des employés
            holidayView.setVisible(true); // Afficher la vue des congés
        });

        // Ajouter un écouteur de sélection sur la table des employés
```

```
        view.employeeTable.getSelectionModel().addListSelectionListener(e -> {
            if (!e.getValueIsAdjusting()) {
                int selectedRow = view.employeeTable.getSelectedRow();
                if (selectedRow != -1) {
                    // Récupérer les données de la ligne sélectionnée
                    int id = (int) view.employeeTable.getValueAt(selectedRow, 0);
                    String nom = (String) view.employeeTable.getValueAt(selectedRow, 1);
                    String prenom = (String) view.employeeTable.getValueAt(selectedRow, 2);
                    String email = (String) view.employeeTable.getValueAt(selectedRow, 3);
                    String phone = (String) view.employeeTable.getValueAt(selectedRow, 4);
                    double salaire = (double) view.employeeTable.getValueAt(selectedRow, 5);
                    Role role = Role.valueOf(view.employeeTable.getValueAt(selectedRow,
6).toString());
                    Poste poste = Poste.valueOf(view.employeeTable.getValueAt(selectedRow,
7).toString());

                    // Remplir les champs de modification avec les valeurs de la ligne
sélectionnée
                    view.nameField.setText(nom);
                    view.surnameField.setText(prenom);
                    view.emailField.setText(email);
                    view.phoneField.setText(phone);
                    view.salaryField.setText(String.valueOf(salaire));
                    view.roleCombo.setSelectedItem(role.toString());
                    view.posteCombo.setSelectedItem(poste.toString());

                    // Sauvegarder l'ID de l'employé dans le bouton de modification pour une
mise à jour
                    view.modifyButton.setActionCommand(String.valueOf(id));
                }
            }
        });

        // Charger la liste des employés au démarrage
        listEmployees();
    }

    // Méthode pour ajouter un employé avec validation
    private void addEmployee() {
        try {
            String nom = view.nameField.getText().trim();
            String prenom = view.surnameField.getText().trim();
            String email = view.emailField.getText().trim();
            String phone = view.phoneField.getText().trim();
            String salaireText = view.salaryField.getText().trim();
```

```
            // Validation des champs
            if (nom.isEmpty() || prenom.isEmpty() || email.isEmpty() || phone.isEmpty() ||
salaireText.isEmpty()) {
                JOptionPane.showMessageDialog(view, "Tous les champs sont obligatoires.");
                return;
            }
            if (!email.contains("@")) {
                JOptionPane.showMessageDialog(view, "Veuillez entrer une adresse email
valide.");
                return;
            }
            double salaire = Double.parseDouble(salaireText);

            Role role = Role.valueOf(view.roleCombo.getSelectedItem().toString().toUpperCase());
            Poste poste =
Poste.valueOf(view.posteCombo.getSelectedItem().toString().toUpperCase());

            Employee employee = new Employee(nom, prenom, email, phone, salaire, role, poste);
            dao.add(employee);
            JOptionPane.showMessageDialog(view, "Employé ajouté avec succès.");
            listEmployees(); // Rafraîchir la liste
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(view, "Salaire invalide.");
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(view, "Erreur: " + ex.getMessage());
        }
    }

    private void listEmployees() {
        List<Employee> employees = dao.listAll();
        String[] columnNames = {"ID", "Nom", "Prénom", "Email", "Téléphone", "Salaire", "Rôle",
"Poste", "solde"};
        DefaultTableModel model = new DefaultTableModel(columnNames, 0);

        for (Employee emp : employees) {
            Object[] row = {emp.getId(), emp.getNom(), emp.getPrenom(), emp.getEmail(),
emp.getPhone(), emp.getSalaire(), emp.getRole(), emp.getPoste(), emp.getSolde()};
            model.addRow(row);
        }

        view.employeeTable.setModel(model);
    }

    private void deleteEmployee() {
        try {
            // Demander l'ID de l'employé à supprimer
```

```
            String idInput = JOptionPane.showInputDialog(view,
                    "Entrez l'ID de l'employé à supprimer :",
                    "Suppression d'un employé",
                    JOptionPane.QUESTION_MESSAGE);

            if (idInput == null || idInput.trim().isEmpty()) {
                JOptionPane.showMessageDialog(view, "Aucun ID saisi. Suppression annulée.");
                return;
            }

            int id = Integer.parseInt(idInput.trim());

            // Demander confirmation avant de supprimer
            int confirm = JOptionPane.showConfirmDialog(view,
                    "Êtes-vous sûr de vouloir supprimer l'employé avec l'ID " + id + " ?",
                    "Confirmation de suppression",
                    JOptionPane.YES_NO_OPTION);

            if (confirm == JOptionPane.YES_OPTION) {
                // Supprimer l'employé
                dao.delete(id);
                JOptionPane.showMessageDialog(view, "Employé supprimé avec succès.");
                listEmployees(); // Rafraîchir la liste
            } else {
                JOptionPane.showMessageDialog(view, "Suppression annulée.");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(view, "ID invalide. Veuillez entrer un nombre
valide.");
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(view, "Erreur : " + ex.getMessage());
        }
    }
    private void modifyEmployee() {
        try {
            // Récupérer l'ID de l'employé à partir du bouton de modification
            String actionCommand = view.modifyButton.getActionCommand();
            if (actionCommand != null && !actionCommand.trim().isEmpty()) {
                int id = Integer.parseInt(actionCommand.trim());

                String nom = view.nameField.getText().trim();
                String prenom = view.surnameField.getText().trim();
                String email = view.emailField.getText().trim();
                String phone = view.phoneField.getText().trim();
                String salaireText = view.salaryField.getText().trim();
```

```
                // Validation des champs
                if (nom.isEmpty() || prenom.isEmpty() || email.isEmpty() || phone.isEmpty() ||
salaireText.isEmpty()) {
                    JOptionPane.showMessageDialog(view, "Tous les champs sont obligatoires.");
                    return;
                }
                if (!email.contains("@")) {
                    JOptionPane.showMessageDialog(view, "Veuillez entrer une adresse email
valide.");
                    return;
                }
                double salaire = Double.parseDouble(salaireText);

                Role role =
Role.valueOf(view.roleCombo.getSelectedItem().toString().toUpperCase());
                Poste poste =
Poste.valueOf(view.posteCombo.getSelectedItem().toString().toUpperCase());

                Employee updatedEmployee = new Employee(nom, prenom, email, phone, salaire,
role, poste);
                dao.update(updatedEmployee, id);

                JOptionPane.showMessageDialog(view, "Employé mis à jour avec succès.");
                listEmployees(); // Rafraîchir la liste
            } else {
                JOptionPane.showMessageDialog(view, "Veuillez sélectionner un employé à
modifier.");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(view, "Salaire invalide.");
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(view, "Erreur: " + ex.getMessage());
        }
    }
}
```

HolidayController :

```
package Controller;

import DAO.HolidayDAOImpl;
import Model.Holiday;
import Model.Type;
import View.HolidayView;

import javax.swing.*;
```

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.List;

public class HolidayController {
    private final HolidayView view;
    private final HolidayDAOImpl dao;

    public HolidayController(HolidayView view) {
        this.view = view;
        this.dao = new HolidayDAOImpl();

        loadEmployeeNames();
        refreshHolidayTable();

        view.addButton.addActionListener(e -> addHoliday());
        view.deleteButton.addActionListener(e -> deleteHoliday());
        view.modifyButton.addActionListener(e -> modifyHoliday());

        // Ajout d'un écouteur de sélection sur la table
        view.holidayTable.getSelectionModel().addListSelectionListener(e -> {
            if (!e.getValueIsAdjusting()) { // Vérifie si la sélection est terminée
                int selectedRow = view.holidayTable.getSelectedRow();
                if (selectedRow != -1) {
                    // Récupérer l'ID et les autres informations de la ligne sélectionnée
                    int id = (int) view.holidayTable.getValueAt(selectedRow, 0); // Récupère
l'ID depuis la colonne 0
                    String employeeName = (String) view.holidayTable.getValueAt(selectedRow, 1);
                    String startDate = (String) view.holidayTable.getValueAt(selectedRow, 2);
                    String endDate = (String) view.holidayTable.getValueAt(selectedRow, 3);
                    Type type = Type.valueOf(view.holidayTable.getValueAt(selectedRow,
4).toString());

                    // Remplir les champs de modification avec ces valeurs
                    view.employeeNameComboBox.setSelectedItem(employeeName);
                    view.startDateField.setText(startDate);
                    view.endDateField.setText(endDate);
                    view.typeCombo.setSelectedItem(type.toString());

                    // Modifier l'action du bouton de modification pour utiliser l'ID de la
ligne sélectionnée
                    view.modifyButton.setActionCommand(String.valueOf(id));
                }
            }
        });
    }
```

```
    private void loadEmployeeNames() {
        view.employeeNameComboBox.removeAllItems();
        List<String> names = dao.getAllEmployeeNames();

        if (names.isEmpty()) {
            System.out.println("Aucun employé trouvé.");
        } else {
            System.out.println("Noms des employés chargés : " + names);
            for (String name : names) {
                view.employeeNameComboBox.addItem(name);
            }
        }
    }

    private void refreshHolidayTable() {
        List<Holiday> holidays = dao.listAll();
        String[] columnNames = {"ID", "Employé", "Date Début", "Date Fin", "Type"};
        Object[][] data = new Object[holidays.size()][5];

        for (int i = 0; i < holidays.size(); i++) {
            Holiday h = holidays.get(i);
            data[i] = new Object[]{h.getId(), h.getEmployeeName(), h.getStartDate(),
h.getEndDate(), h.getType()};
        }

        view.holidayTable.setModel(new javax.swing.table.DefaultTableModel(data, columnNames));
    }

    private boolean isValidDate(String date) {
        try {
            DateTimeFormatter formatter = DateTimeFormatter.ISO_LOCAL_DATE;
            LocalDate.parse(date, formatter);
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    private boolean isEndDateAfterStartDate(String startDate, String endDate) {
        DateTimeFormatter formatter = DateTimeFormatter.ISO_LOCAL_DATE;
        LocalDate start = LocalDate.parse(startDate, formatter);
        LocalDate end = LocalDate.parse(endDate, formatter);

        return end.isAfter(start);
    }
```

```
    private void addHoliday() {
        try {
            String employeeName = (String) view.employeeNameComboBox.getSelectedItem();
            String startDate = view.startDateField.getText();
            String endDate = view.endDateField.getText();
            Type type = Type.valueOf(view.typeCombo.getSelectedItem().toString().toUpperCase());

            if (!isValidDate(startDate) || !isValidDate(endDate)) {
                throw new IllegalArgumentException("Les dates doivent être au format YYYY-MM-
DD.");
            }

            if (!isEndDateAfterStartDate(startDate, endDate)) {
                throw new IllegalArgumentException("La date de fin doit être supérieure à la
date de début.");
            }

            Holiday holiday = new Holiday(employeeName, startDate, endDate, type);
            dao.add(holiday);
            loadEmployeeNames();
            refreshHolidayTable();
            JOptionPane.showMessageDialog(view, "Congé ajouté avec succès.");
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(view, "Erreur : " + ex.getMessage());
        }
    }

    private void modifyHoliday() {
        try {
            // Récupérer l'ID du congé à partir de l'action du bouton
            String actionCommand = view.modifyButton.getActionCommand();
            if (actionCommand != null && !actionCommand.trim().isEmpty()) {
                int id = Integer.parseInt(actionCommand.trim());

                String employeeName = (String) view.employeeNameComboBox.getSelectedItem();
                String startDate = view.startDateField.getText();
                String endDate = view.endDateField.getText();
                Type type =
Type.valueOf(view.typeCombo.getSelectedItem().toString().toUpperCase());

                if (!isValidDate(startDate) || !isValidDate(endDate)) {
                    throw new IllegalArgumentException("Les dates doivent être au format YYYY-
MM-DD.");
                }
```

```
                if (!isEndDateAfterStartDate(startDate, endDate)) {
                    throw new IllegalArgumentException("La date de fin doit être supérieure à la
date de début.");
                }

                Holiday holiday = new Holiday(employeeName, startDate, endDate, type);
                dao.update(holiday, id);
                loadEmployeeNames();
                refreshHolidayTable();
                JOptionPane.showMessageDialog(view, "Congé modifié avec succès.");
            } else {
                JOptionPane.showMessageDialog(view, "Veuillez sélectionner un congé à
modifier.");
            }
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(view, "Erreur : " + ex.getMessage());
        }
    }

    private void deleteHoliday() {
        try {
            // Afficher un message pour entrer un ID de congé
            String input = JOptionPane.showInputDialog(view, "Veuillez entrer l'ID du congé à
supprimer:");
            if (input != null && !input.trim().isEmpty()) {
                int id = Integer.parseInt(input.trim());

                // Demander confirmation avant de supprimer
                int confirm = JOptionPane.showConfirmDialog(view,
                        "Êtes-vous sûr de vouloir supprimer le congé avec l'ID " + id + " ?",
                        "Confirmation de suppression",
                        JOptionPane.YES_NO_OPTION);

                if (confirm == JOptionPane.YES_OPTION) {
                    dao.delete(id); // Supprimer le congé
                    loadEmployeeNames();
                    refreshHolidayTable();
                    JOptionPane.showMessageDialog(view, "Congé supprimé avec succès.");
                } else {
                    JOptionPane.showMessageDialog(view, "Suppression annulée.");
                }
            } else {
                JOptionPane.showMessageDialog(view, "ID de congé non valide ou action
annulée.");
            }
        } catch (NumberFormatException ex) {
```

```
            JOptionPane.showMessageDialog(view, "ID invalide. Veuillez entrer un nombre
valide.");
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(view, "Erreur : " + ex.getMessage());
        }
    }

}
```

viewController :

```
package Controller;

import View.EmployeeView;
import View.HolidayView;

public class ViewController {
    private final EmployeeView employeeView;
    private final HolidayView holidayView;

    public ViewController() {
        this.employeeView = new EmployeeView();
        this.holidayView = new HolidayView();

        // Connect views for navigation
        employeeView.switchViewButton.addActionListener(e -> showHolidayView());
        holidayView.switchViewButton.addActionListener(e -> showEmployeeView());
    }

    public void showEmployeeView() {
        holidayView.setVisible(false);
        employeeView.setVisible(true);
    }

    public void showHolidayView() {
        employeeView.setVisible(false);
        holidayView.setVisible(true);
    }

    public static void main(String[] args) {
        new ViewController().showEmployeeView();
    }
}
```

3 Architecture DAO :

DBconnection :

```
package DAO;

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

public class DBConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/TP7";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

EmployeeDAOImpl :

```
package DAO;

import Model.Employee;
import Model.Poste;
import Model.Role;

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

public class EmployeeDAOImpl implements GenericDAO<Employee> {

    @Override
    public void add(Employee employee) {
        String sql = "INSERT INTO Employe (nom, prenom, email, phone, salaire, role, poste)
VALUES (?, ?, ?, ?, ?, ?, ?)";
        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement(sql)) {
            stmt.setString(1, employee.getNom());
            stmt.setString(2, employee.getPrenom());
            stmt.setString(3, employee.getEmail());
            stmt.setString(4, employee.getPhone());
            stmt.setDouble(5, employee.getSalaire());
```

```
            stmt.setString(6, employee.getRole().name());
            stmt.setString(7, employee.getPoste().name());
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void delete(int id) {
        String sql = "DELETE FROM Employe WHERE id = ?";
        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement(sql)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public List<Employee> listAll() {
        List<Employee> employees = new ArrayList<>();
        String sql = "SELECT * FROM Employe";
        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement(sql);
             ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                String roleStr = rs.getString("role").toUpperCase();
                String posteStr = rs.getString("poste").toUpperCase();

                Role role = null;
                Poste poste = null;
                try {
                    role = Role.valueOf(roleStr);
                } catch (IllegalArgumentException e) {
                    System.out.println("Role non valide : " + roleStr);
                    role = Role.EMPLOYE; // Valeur par défaut
                }

                try {
                    poste = Poste.valueOf(posteStr);
                } catch (IllegalArgumentException e) {
                    System.out.println("Poste non valide : " + posteStr);
                    poste = Poste.INGENIEURE_ETUDE_ET_DEVELOPPEMENT; // Valeur par défaut
```

```
                }

                Employee employee = new Employee(
                        rs.getString("nom"),
                        rs.getString("prenom"),
                        rs.getString("email"),
                        rs.getString("phone"),
                        rs.getDouble("salaire"),
                        rs.getInt("solde"),
                        role,
                        poste
                );
                employee.setId(rs.getInt("id"));
                employees.add(employee);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return employees;
    }




    @Override
    public Employee findById(int id) {
        String sql = "SELECT * FROM Employe WHERE id = ?";
        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement(sql)) {
            stmt.setInt(1, id);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return new Employee(
                        rs.getString("nom"),
                        rs.getString("prenom"),
                        rs.getString("email"),
                        rs.getString("phone"),
                        rs.getDouble("salaire"),
                        Role.valueOf(rs.getString("role")),
                        Poste.valueOf(rs.getString("poste"))
                );
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
        return null;
    }

    @Override
    public void update(Employee employee, int id) {
        String sql = "UPDATE Employe SET nom = ?, prenom = ?, email = ?, phone = ?, salaire = ?,
role = ?, poste = ? WHERE id = ?";

        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement(sql)) {
            stmt.setString(1, employee.getNom());
            stmt.setString(2, employee.getPrenom());
            stmt.setString(3, employee.getEmail());
            stmt.setString(4, employee.getPhone());
            stmt.setDouble(5, employee.getSalaire());
            stmt.setString(6, employee.getRole().name()); // Envoi du rôle en tant que chaîne
(avec la méthode .name())
            stmt.setString(7, employee.getPoste().name()); // Idem pour le poste
            stmt.setInt(8, id); // L'ID de l'employé à mettre à jour
            int rowsUpdated = stmt.executeUpdate();

            if (rowsUpdated > 0) {
                System.out.println("L'employé a été mis à jour avec succès.");
            } else {
                System.out.println("Aucun employé trouvé avec cet ID.");
            }

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

}
```

GenericDAO :

```
package DAO;

import java.util.List;
public interface GenericDAO<T> {
    void add(T entity); // Ajouter un objet
    void delete(int id); // Supprimer un objet par ID
    List<T> listAll(); // Lister tous les objets
    T findById(int id); // Trouver un objet par ID
    void update(T entity, int id); // Mettre à jour un objet
}
```

HolidayDAOImpl :

```
package DAO;

import Model.Holiday;
import Model.Type;

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

public class HolidayDAOImpl implements GenericDAO<Holiday> {

    // Constants for SQL queries
    private static final String INSERT_HOLIDAY_SQL = "INSERT INTO holiday (employeeId,
startDate, endDate, type) VALUES (?, ?, ?, ?)";
    private static final String DELETE_HOLIDAY_SQL = "DELETE FROM holiday WHERE id = ?";
    private static final String SELECT_ALL_HOLIDAY_SQL = "SELECT h.id, CONCAT(e.nom, ' ',
e.prenom) AS employeeName, h.startDate, h.endDate, h.type FROM holiday h JOIN employe e ON
h.employeeId = e.id";
    private static final String SELECT_HOLIDAY_BY_ID_SQL = "SELECT h.id, CONCAT(e.nom, ' ',
e.prenom) AS employeeName, h.startDate, h.endDate, h.type FROM holiday h JOIN employe e ON
h.employeeId = e.id WHERE h.id = ?";
    private static final String SELECT_EMPLOYEE_ID_BY_NAME_SQL = "SELECT id FROM employe WHERE
CONCAT(nom, ' ', prenom) = ?";
    // Méthode pour ajouter un congé
    @Override
    public void add(Holiday holiday) {
        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement(INSERT_HOLIDAY_SQL)) {
            int employeeId = getEmployeeIdByName(holiday.getEmployeeName());
            if (employeeId == -1) {
                System.out.println("Erreur : Employé introuvable.");
                return;
            }
            stmt.setInt(1, employeeId);
            stmt.setString(2, holiday.getStartDate());
            stmt.setString(3, holiday.getEndDate());
            stmt.setString(4, holiday.getType().name());
            stmt.executeUpdate();
            System.out.println("Congé ajouté avec succès.");
        } catch (SQLException e) {
            System.err.println("Erreur lors de l'ajout du congé : " + e.getMessage());
            e.printStackTrace();
```

```
        }
    }

    // Méthode pour supprimer un congé par ID
    @Override
    public void delete(int id) {
        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement(DELETE_HOLIDAY_SQL)) {
            stmt.setInt(1, id);
            int rowsDeleted = stmt.executeUpdate();
            if (rowsDeleted > 0) {
                System.out.println("Congé supprimé avec succès.");
            } else {
                System.out.println("Aucun congé trouvé avec cet ID.");
            }
        } catch (SQLException e) {
            System.err.println("Erreur lors de la suppression du congé : " + e.getMessage());
            e.printStackTrace();
        }
    }

    // Méthode pour lister tous les congés
    @Override
    public List<Holiday> listAll() {
        List<Holiday> holidays = new ArrayList<>();
        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement(SELECT_ALL_HOLIDAY_SQL); ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                Holiday holiday = new Holiday(
                        rs.getInt("id"),
                        rs.getString("employeeName"),
                        rs.getString("startDate"),
                        rs.getString("endDate"),
                        Type.valueOf(rs.getString("type"))
                );
                holidays.add(holiday);
            }
        } catch (SQLException e) {
            System.err.println("Erreur lors de la récupération des congés : " + e.getMessage());
            e.printStackTrace();
        }
        return holidays;
    }

    // Méthode pour trouver un congé par ID
    @Override
```

```
    public Holiday findById(int id) {
        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement(SELECT_HOLIDAY_BY_ID_SQL)) {
            stmt.setInt(1, id);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return new Holiday(
                        rs.getInt("id"),
                        rs.getString("employeeName"),
                        rs.getString("startDate"),
                        rs.getString("endDate"),
                        Type.valueOf(rs.getString("type"))
                );
            }
        } catch (SQLException e) {
            System.err.println("Erreur lors de la recherche du congé : " + e.getMessage());
            e.printStackTrace();
        }
        return null;
    }

    // Méthode pour mettre à jour un congé
    @Override
    public void update(Holiday holiday, int id) {
        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement("UPDATE holiday SET employeeId = ?, startDate = ?, endDate = ?, type = ?
WHERE id = ?")) {
            int employeeId = getEmployeeIdByName(holiday.getEmployeeName());
            if (employeeId == -1) {
                System.out.println("Erreur : Employé introuvable.");
                return;
            }
            stmt.setInt(1, employeeId);
            stmt.setString(2, holiday.getStartDate());
            stmt.setString(3, holiday.getEndDate());
            stmt.setString(4, holiday.getType().name());
            stmt.setInt(5, id);
            int rowsUpdated = stmt.executeUpdate();
            if (rowsUpdated > 0) {
                System.out.println("Congé mis à jour avec succès.");
            } else {
                System.out.println("Aucun congé trouvé avec cet ID.");
            }
        } catch (SQLException e) {
            System.err.println("Erreur lors de la mise à jour du congé : " + e.getMessage());
            e.printStackTrace();
```

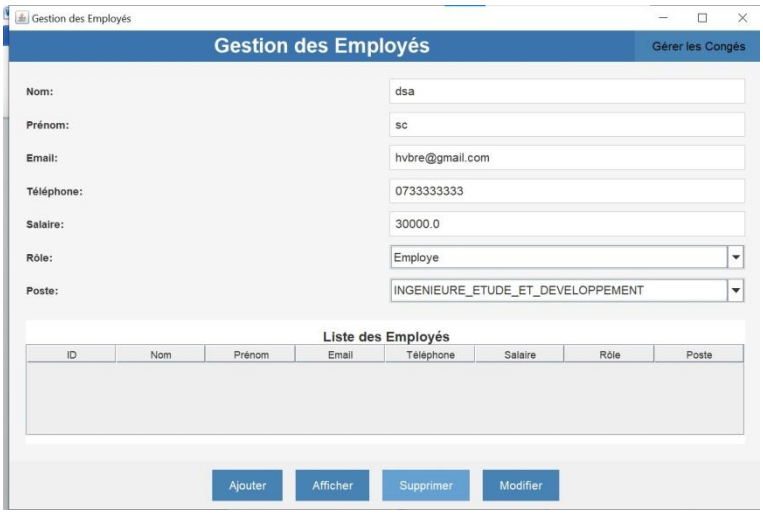
```
        }
    }

    // Méthode pour récupérer l'ID de l'employé par nom complet
    public int getEmployeeIdByName(String employeeName) {
        try (Connection conn = DBConnection.getConnection(); PreparedStatement stmt =
conn.prepareStatement(SELECT_EMPLOYEE_ID_BY_NAME_SQL)) {
            stmt.setString(1, employeeName);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return rs.getInt("id");
            }
        } catch (SQLException e) {
            System.err.println("Erreur lors de la récupération de l'ID employé : " +
e.getMessage());
            e.printStackTrace();
        }
        return -1;
    }

    // Méthode pour récupérer tous les noms des employés
    public List<String> getAllEmployeeNames() {
        List<String> employeeNames = new ArrayList<>();
        String query = "SELECT CONCAT(nom, ' ', prenom) AS fullName FROM employe";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(query);
             ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                String fullName = rs.getString("fullName");
                System.out.println("Nom récupéré : " + fullName); // Pour debug
                employeeNames.add(fullName);
            }
        } catch (SQLException e) {
            System.err.println("Erreur lors de la récupération des noms des employés : " +
e.getMessage());
        }

        return employeeNames;
    }



    // Méthode pour récupérer les types de congés (Enum)
    public List<Type> getAllHolidayTypes() {
```

```
        List<Type> holidayTypes = new ArrayList<>();
        for (Type type : Type.values()) {
            holidayTypes.add(type);
        }
        return holidayTypes;
    }
}
```

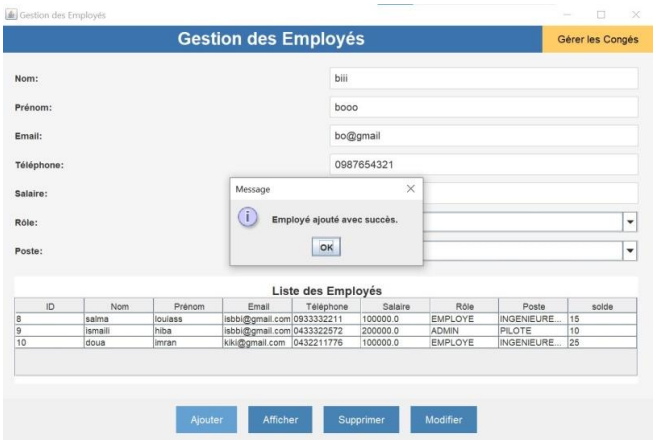
Main:

```
package Main;

import Controller.EmployeeController;
import Controller.HolidayController;
import View.EmployeeView;
import View.HolidayView;

public class Main {
    public static void main(String[] args) {
        // Créer les vues
        EmployeeView employeeView = new EmployeeView();
        HolidayView holidayView = new HolidayView();

        // Créer les contrôleurs pour chaque vue
        new EmployeeController(employeeView, holidayView);
        new HolidayController(holidayView);

        // Définir quelle vue sera affichée par défaut (exemple : vue des employés)
        employeeView.setVisible(true);

        // Ajouter un gestionnaire pour passer de l'une à l'autre
        employeeView.switchViewButton.addActionListener(e -> {
            employeeView.setVisible(false);
            holidayView.setVisible(true);
        });

        holidayView.switchViewButton.addActionListener(e -> {
            holidayView.setVisible(false);
            employeeView.setVisible(true);
        });
    }
}
```

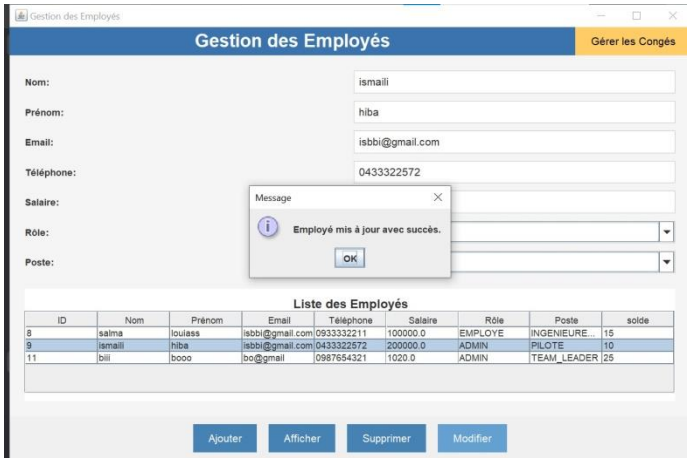
Résultats



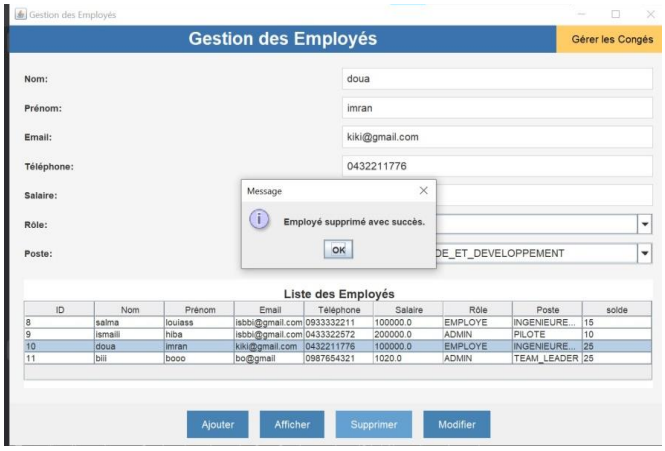
EmployeAdd



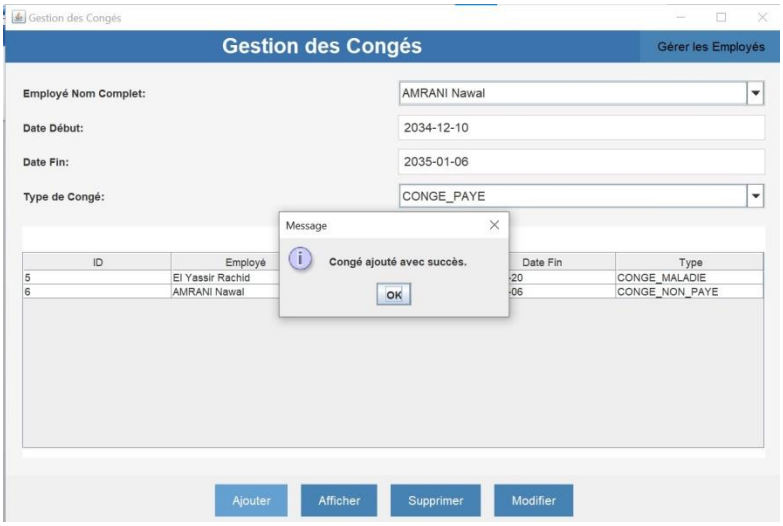
EmployeeModifier :



EmployeeSupp :

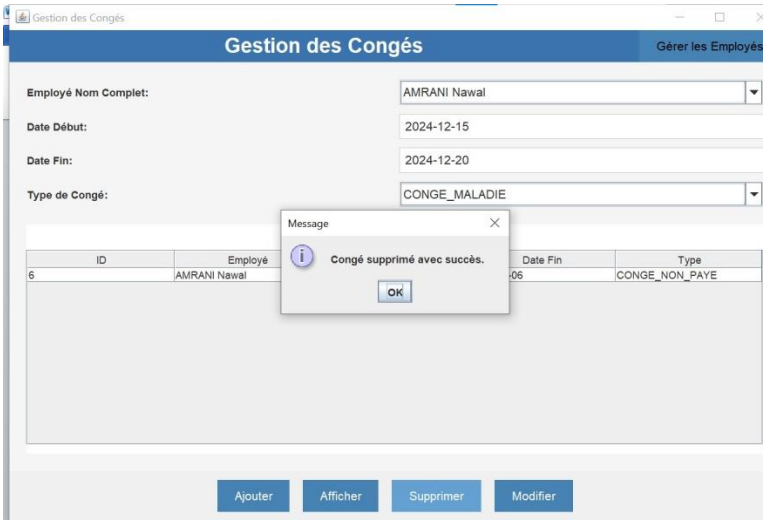


CongeAjouter :



CongeModifier :

CongeSupprimer :



Bilan de la séance de travaux pratiques : Gestion des employés et de leurs congés

Dans cette séance de travaux pratiques, nous avons développé une application Java complète pour la gestion des employés d'une entreprise ainsi que de leurs congés. L'application est structurée selon une architecture en couches (logique métier, DAO, contrôleur et interface utilisateur) et respecte les principes de la programmation orientée objet.

Tâches principales réalisées

1. Travail en équipe

Configuration initiale du projet : Mise en place de la structure du projet Java et définition des classes principales, incluant les classes pour les employés et leurs congés.

Implémentation de la couche DAO : Développement des méthodes pour la gestion des employés et des congés, notamment pour l'ajout, la mise à jour et la suppression.

Validation des requêtes SQL : Test et correction des requêtes SQL utilisées pour gérer les employés et enregistrer les demandes de congés.

Intégration entre les couches : Synchronisation des interactions entre la logique métier et le contrôleur pour assurer une gestion fluide des employés et de leurs congés.

2. Travail individuel

Conception de l'interface graphique :

Création de panneaux interactifs pour saisir et afficher les informations des employés.

Développement d'un module spécifique pour la gestion des congés, permettant de :

Ajouter une demande de congé pour un employé.

Afficher l'état des congés (en attente, approuvé ou refusé).

Validation des champs de saisie : Mise en place de contrôles pour garantir la validité des données saisies pour les employés et les congés (dates, types de congés, etc.).

Tests fonctionnels : Simulation des différentes actions telles que :

Ajout, modification, suppression et consultation des employés.

Création, mise à jour et suivi des demandes de congés.

3. Tests et validation

Vérification des données : Contrôle de la cohérence entre les données affichées dans l'interface et celles stockées dans la base de données, tant pour les employés que pour les congés.

Identification et correction de bugs :

Résolution d'erreurs liées au format des champs (par exemple, les dates de congés).

Correction des incohérences dans la gestion des relations entre employés et congés.

Bilan et compétences acquises

Cette séance a permis de renforcer nos compétences en programmation Java, particulièrement dans les domaines suivants :

Développement d'une architecture en couches pour une application modulaire et évolutive.

Gestion des interactions avec une base de données pour des entités liées (employés et congés).

Conception d'interfaces utilisateur conviviales avec Swing, intégrant des fonctionnalités avancées de gestion.

Mise en œuvre de validations robustes pour garantir l'intégrité des données.

Test et correction de fonctionnalités pour assurer la cohérence des flux de travail (gestion des employés et de leurs congés).

Ce TP nous a offert une expérience pratique et complète, préparant à des projets logiciels plus avancés et répondant à des besoins complexes en gestion d'entreprise.