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

gestion des employés

Rapport du tp 1(java avancé)

Réalisé par : ELHIYANI Hanane

Encadré par : Mme.ELKHROF Leila

Année Universitaire : 2024/2025

Table des matières

Table des figures

Inroduction

Dans le cadre de notre apprentissage des concepts avancés de développement d'applications, ce TP vise à mettre en pratique le modèle architectural **MVC (Model-View-Controller)** associé au **DAO (Data Access Object)**. Le projet consiste à développer une application de gestion des employés, permettant l'ajout, la modification, la suppression dans une base de données.

L'objectif principal est de séparer les différentes couches de l'application :
- Model : Gère la logique métier et l'interaction avec la base de données.
- View : Assure l'affichage et l'interaction utilisateur.
- Controller : Traite les requêtes de l'utilisateur et met à jour le modèle et la vue en conséquence.
- DAO :(Data Access Object) est une couche du modèle chargée des opérations de persistance. Elle gère l'accès aux données en exécutant les opérations CRUD (Create, Read, Update, Delete), tout en séparant la logique métier de l'accès à la base de données.

L'utilisation du DAO garantit une gestion centralisée et sécurisée des opérations sur les données, favorisant ainsi la maintenance et l'évolution de l'application. Ce TP permettra de mieux comprendre la structuration d'une application selon les bonnes pratiques du développement logiciel.

Outils & environnement de travail

1 Language de programmation

FIGURE 1 – JAVA logo

- **Java** : est un langage de programmation orienté objet, sécurisé, portable et indépendant de la plateforme. Il est utilisé pour développer des applications web, mobiles et de bureau, grâce à sa machine virtuelle Java (JVM), qui permet d'exécuter le même code sur différentes plateformes.

2 Environnement de travail

FIGURE 2 – intellij idea logo

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

FIGURE 3 – xampp logo

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

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

Elaboration & Réalisation

1 Création de la base de donnée

```
//
CREATE DATABASE EmployeDB;

//
USE EmployeDB;
//

CREATE TABLE Employes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    phone_number VARCHAR(20),
    salary DECIMAL(10, 2),
    role VARCHAR(50),
    poste VARCHAR(50)
);
//
CREATE TABLE Role (
  name varchar(30) NOT NULL
);
//
CREATE TABLE Poste (
  name varchar(30) NOT NULL
);
```

Listing 1.1 – Script SQL de la base de données

2 Implémentation de l'architecture MVC avec le pattern DAO

Les étapes pratiques suivies dans ce TP sont organisées selon l'architecture **MVC (Model-View-Controller)** combinée au **pattern DAO (Data Access Object)** pour assurer une bonne séparation des responsabilités. Après la création de la base de données, nous commençons par la gestion des données, en implémentant le DAO pour interagir avec la base. Les étapes sont les suivantes :

- **Étape 1 : Implémentation du modèle (couche Model)**
  Création de la classe `Employe` représentant l'entité métier, avec ses attributs, ses getters et setters.
- **Étape 2 : Gestion des données (DAO)**
  Création de l'interface `EmployerInterface` définissant les opérations CRUD, suivie de l'implémentation de la classe `EmployerDAO` pour la gestion des données de la base.
- **Étape 3 : Logique métier**
  Développement de la classe `EmployerLogic` contenant les règles métier et la gestion des opérations complexes.
- **Étape 4 : Interface graphique (couche View)**
  Création de l'interface utilisateur en utilisant `JTabbedPane` pour afficher les données et interagir avec l'application.
- **Étape 5 : Contrôleur (couche Controller)**
  Gestion des événements de l'interface utilisateur pour relier la vue et la logique métier.
- **Étape 6 : Main**
  Initialisation de l'application et lancement de l'interface principale.

Ces étapes assurent une application bien structurée, évolutive et facile à maintenir.

3 Couche Model

Dans cette étape, je vais me concentrer sur la partie « Modèle » de l'architecture MVC. Le Modèle représente les données et la logique métier de l'application. Pour cela, je vais créer un dossier nommé `Model` dans lequel je vais définir les énumérations `Poste` et `Role`. Ensuite, je vais créer la classe `Employe` avec un constructeur qui prend en paramètre tous les attributs nécessaires. Je vais également ajouter des getters et des setters pour accéder et modifier ces attributs facilement.

Dans le `EmployeModel`, la logique métier est implémentée pour valider les données avant leur manipulation. Par exemple, il est vérifié que l'email contient bien le caractère `@`, et que le numéro de téléphone comporte exactement 10 chiffres. Ces vérifications garantissent l'intégrité des données avant leur enregistrement ou mise à jour dans la base de données.

Employer

```
package Model;
public class Employe {
    private int id;
    private String nom;
    private String prenom;
    private String email;
    private String telephone;
    private double salaire;
    private Role role;
    private Poste poste;
    public Employe(int id,String nom,String prenom,String email,String telephone,
    double salaire,Role role,Poste poste) {
        this.id=id;
        this.nom=nom;
        this.prenom=prenom;
        this.email=email;
        this.telephone=telephone;
        this.salaire=salaire;
        this.role=role;
        this.poste=poste;
    }
    public Employe(String nom,String prenom,String email,String telephone,double
    salaire,Role role,Poste poste) {
        this.nom=nom;
        this.prenom=prenom;
        this.email=email;
        this.telephone=telephone;
        this.salaire=salaire;
        this.role=role;
        this.poste=poste;
    }
    public int getId() {
        return 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 getTelephone() {
```

```
        return telephone;
    }
    public void setTelephone(String telephone) {
        this.telephone = telephone;
    }
    public double getSalaire() {
        return salaire;
    }
    public void setSalaire(double salaire) {
        this.salaire = salaire;
    }
    public String getRole() {
        return role.name();
    }

    public String getPoste() {
        return poste.name();
    }
    public void setRole(Role role) {
        this.role = role;
    }
    public void setPoste(Poste poste) {
        this.poste = poste;
    }

}
```

EmployeModel

```
package Model;
import java.util.List;
import DAO.EmployeDAOImp;
public class EmployeModel {
    private EmployeDAOImp dao;

    public EmployeModel(EmployeDAOImp dao) {
        this.dao=dao;
    }
    public boolean add(String nom, String prenom, String email, String telephone, double
     salaire, Role role, Poste poste) {
        if (!email.contains("@")) {
            System.err.println("L'email que vous avez entr  est invalide.");
            return false;
        }

        if (telephone.length() != 10) {
            System.err.println("The phone number must contain exactly 10 digits.");
            return false;
        }

        Employe empl = new Employe(nom, prenom, email, telephone, salaire, role, poste);
        dao.add(empl);
        return true;
    }
    public boolean update(int id,String nom, String prenom, String email, String
```

```
    telephone, double salaire, Role role, Poste poste) {
        if (!email.contains("@")) {  // V rifie si '@' est manquant
            System.err.println("L'email que vous avez entr  est invalide.");
            return false; // Email invalide
        }
        // M thode pour v rifier NumeroTl
        if (telephone.length() != 10) {
            System.err.println("The phone number must contain exactly 10 digits.");
            return false;
        }

        Employe emp = new Employe(id,nom, prenom, email, telephone, salaire, role, poste
    );
        dao.update(emp);
        return true;
     }
     public boolean delete(int id) {
        dao.delete(id);
        return true;
     }
     public Object[][] employes(){
        List<Employe>emp=dao.employes();
        Object[][] tab =new Object[emp.size()][8];
        for (int i = 0; i < emp.size(); i++) {
            Employe empl = emp.get(i);
            tab[i][0] = empl.getId();
            tab[i][1] = empl.getNom();
            tab[i][2] = empl.getPrenom();
            tab[i][3] = empl.getEmail();
            tab[i][4] = empl.getTelephone();
            tab[i][5] = empl.getSalaire();
            tab[i][6] = empl.getRole();
            tab[i][7] = empl.getPoste();
        }
        return tab;
     }
}
```

4 Couche DAO

La couche **DAO** (Data Access Object) assure l'interaction entre l'application et la base de données. Elle contient trois classes principales : la classe `connexion` qui gère l'établissement de la connexion avec la base de données MySQL, l'interface générique `EmployeDAOI` qui définit les opérations CRUD (ajouter, mettre à jour, supprimer, et lister les employés), et la classe `EmployeDAOImp` qui implémente ces opérations en utilisant des requêtes SQL. Cette couche garantit la séparation des préoccupations en isolant la logique d'accès aux données de la logique métier.

connexion

```
package DAO;

import java.sql.Connection;
```

```
import java.sql.DriverManager;
import java.sql.SQLException;

public class connexion {
    private static final String URL="jdbc:mysql://localhost:3306/BDEmploye";
    private static final String USER="root";
    private static final String PASSWORD="";
    static Connection conn=null;
    public static Connection getConnexion() {
        try {
            conn= DriverManager.getConnection(URL,USER,PASSWORD);
        }catch (SQLException e){
            e.printStackTrace();
        }
        return conn;
    }

}
```

EmployerDAO

```
package DAO;

import java.util.List;

import Model.Employe;

public interface EmployeDAOI<T>{
    public void add(T emp);
    public void update(T emp);
    public void delete(int id);
    public List<T> employes();
}
```

EmployeDAOImp

```
package DAO;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;


import javax.swing.JOptionPane;

import Model.Employe;
import Model.Poste;
import Model.Role;

public class EmployeDAOImp implements EmployeDAOI<Employe>{
    private static connexion conn;
```

```
18
19      public EmployeDAOImp() {
20          conn=new connexion();
21      }
22
23      @Override
24      public void add(Employe emp) {
25          String sql = "INSERT INTO Employee (nom, prenom, email, phone, salaire,
        role, poste) VALUES (?, ?, ?, ?, ?, ?, ?)";
26          try (PreparedStatement stmt = connexion.getConnexion().prepareStatement(
        sql)) {
27              stmt.setString(1, emp.getNom());
28              stmt.setString(2, emp.getPrenom());
29              stmt.setString(3, emp.getEmail());
30              stmt.setString(4, emp.getTelephone());
31              stmt.setDouble(5, emp.getSalaire());
32              stmt.setString(6, emp.getRole()); // Convertir en String si Role est
        une   numration
33              stmt.setString(7, emp.getPoste()); // Convertir en String si Poste est
         une   numration
34
35              stmt.executeUpdate();
36
37          } catch (SQLException e) {
38
39              e.printStackTrace();
40          }
41      }
42
43      @Override
44      public void update(Employe emp) {
45          String sql = "UPDATE Employee SET nom = ? , prenom = ? , email = ?,phone =
         ?, salaire = ?, role = ?, poste = ? WHERE id = ?";
46          try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
47              stmt.setString(1, emp.getNom());
48              stmt.setString(2, emp.getPrenom());
49              stmt.setString(3, emp.getEmail());
50              stmt.setString(4, emp.getTelephone());
51              stmt.setDouble(5, emp.getSalaire());
52              stmt.setString(6, emp.getRole());
53              stmt.setString(7, emp.getPoste());
54              stmt.setInt(8, emp.getId());
55              stmt.executeUpdate();
56
57          } catch (SQLException e) {
58              e.printStackTrace();
59          }
60      }
61      @Override
62      public void delete(int id) {
63          String sql = "DELETE FROM Employee WHERE id = ?";
64          try (PreparedStatement stmt = conn.getConnexion().prepareStatement(sql)) {
65              stmt.setInt(1, id);
66              stmt.executeUpdate();
67
```

```
68          } catch (SQLException e) {
69              e.printStackTrace();
70          }
71      }
72      public List<Employe> employes() {
73          List<Employe> emp =new ArrayList<>();
74          String sql = "SELECT * FROM Employee";
75          try (PreparedStatement stmt = connexion.getConnexion().prepareStatement(
        sql);
76                   ResultSet rslt=stmt.executeQuery()){
77              while(rslt.next()) {
78                  emp.add(new Employe(
79                          rslt.getInt("id"),
80                          rslt.getString("nom"),
81                          rslt.getString("prenom"),
82                          rslt.getString("email"),
83                          rslt.getString("phone"),
84                          rslt.getDouble("salaire"),
85                          Role.valueOf(rslt.getString("role")),
86                          Poste.valueOf(rslt.getString("poste"))
87                  ));
88              }
89          }catch(SQLException e) {
90             e.printStackTrace();
91          }
92          return emp;
93      }
94  }
```

5 Couche Controller

La couche **Controller** agit comme un médiateur entre la vue (`View`) et le modèle (`Model`). Elle reçoit les actions de l'utilisateur (comme les clics sur les boutons) et appelle les méthodes appropriées du modèle pour modifier les données ou effectuer des actions. Dans le cadre de la gestion des employés, le `EmployeController` écoute les actions sur la vue, telles que l'ajout, la mise à jour, la suppression et l'affichage des employés. Par exemple, la méthode `add()` permet d'ajouter un employé en vérifiant les données saisies, puis elle appelle la méthode du modèle pour effectuer l'ajout dans la base de données. La méthode `update()` met à jour les informations d'un employé, tandis que `delete()` permet de supprimer un employé sélectionné. Enfin, la méthode `employes()` récupère et affiche la liste des employés actuels dans la vue. Cette architecture garantit une séparation claire entre la gestion des données et l'interface utilisateur.

EmployeController

```
package defau.Controller;
import Model.*;
import View.EmployeView;
public class EmployeController {
    private EmployeView view;
    private EmployeModel model;

    public EmployeController(EmployeView view, EmployeModel model) {
        this.view = view;
        this.model = model;

        this.view.getAjouterButton().addActionListener(e -> add());
        this.view.getModifierButton().addActionListener(e -> update());
        this.view.getSupprimerButton().addActionListener(e -> delete());
        this.view.getAfficherButton().addActionListener(e -> employes());
    }

    // M thode pour ajouter un employ
    private void add() {
        String nom = view.getNom().trim();
        String prenom = view.getPrenom().trim();
        String email = view.getEmail().trim();
        String telephone = view.getTelephone().trim();
        Role role = view.getRole();
        Poste poste = view.getPoste();
        Double salaire;
        try {

            salaire = view.getSalaire();

        } catch (NumberFormatException e) {
            view.afficherMessageErreur("Le salaire doit  tre  un nombre valide.");
            return;
        }
        // Appel au mod le pour ajouter l'employ
        boolean ajoutReussi = model.add( nom, prenom, email, telephone, salaire,
    role, poste);
        if (ajoutReussi) {
            view.afficherMessageSucces("Employ  ajout  avec succ s.");
```

```
39              // Actualiser la liste des employ s
40          } else {
41              view.afficherMessageErreur(" chec  de l'ajout de l'employ .");
42          }
43
44
45      }
46
47      // M thode pour mettre    jour un employ
48      private void update() {
49          int selectedRow = view.table.getSelectedRow();
50          if (selectedRow == -1) {
51              view.afficherMessageErreur("Veuillez s lectionner un employ
    modifier.");
52              return;
53          }
54          String nom = view.getNom().trim();
55          String prenom = view.getPrenom().trim();
56          String email = view.getEmail().trim();
57          String telephone = view.getTelephone().trim();
58          Double salaire ;
59          Role role = view.getRole();
60          Poste poste = view.getPoste();
61          int id = (int) view.model.getValueAt(selectedRow, 0);
62          try {
63
64              salaire = view.getSalaire();
65
66          } catch (NumberFormatException e) {
67              view.afficherMessageErreur("Le salaire doit  tre  un nombre valide.");
68              return;
69          }
70          boolean miseAJourReussie = model.update(id, nom, prenom, email, telephone,
     salaire, role, poste);
71          if (miseAJourReussie) {
72              view.afficherMessageSucces("Employ  modifi  avec succ s.");
73              employes();
74          } else {
75              view.afficherMessageErreur(" chec  de la modification de l'employ .")
    ;
76          }
77      }
78
79      // M thode pour supprimer un employ
80      private void delete() {
81
82              int selectedRow = view.table.getSelectedRow();
83              if (selectedRow == -1) {
84                  view.afficherMessageErreur("Veuillez s lectionner un employ
    supprimer.");
85                  return;
86              }
87              int id = (int) view.model.getValueAt(selectedRow, 0);
88
89              boolean suppressionReussie = model.delete(id);
```

```
            if (suppressionReussie) {
                view.afficherMessageSucces("Employ  supprim  avec succ s.");
                employes();

            } else {
                view.afficherMessageErreur(" chec  de la suppression de l'employ
    .");
            }
    }

    // M thode pour afficher tous les employ s
    private void employes() {
            Object[][] employes = model.employes();
            view.model.setRowCount(0); // Vider la table avant d'ajouter les
    donn es
            for (Object[] emp : employes) {
                view.model.addRow(emp);

            }
            System.out.println("bien affiche");

    }
}
```

6 Couche View

La couche **View** est responsable de l'affichage de l'interface utilisateur et de la gestion des interactions avec l'utilisateur. Dans le cas de l'application de gestion des employés, la classe `EmployeView` hérite de `JFrame` et contient tous les éléments graphiques nécessaires à l'interface utilisateur, tels que des champs de texte, des boutons, des labels et une table pour afficher les employés. La vue offre des boutons comme `Ajouter`, `Modifier`, `Supprimer`, et `Afficher` pour permettre à l'utilisateur d'interagir avec les données. Elle utilise des `JComboBox` pour permettre la sélection des rôles et des postes, ainsi qu'un `JTable` pour afficher la liste des employés. La classe inclut également des méthodes pour obtenir les valeurs des champs de texte et pour afficher des messages de succès ou d'erreur, facilitant ainsi la communication avec l'utilisateur. En résumé, la vue gère uniquement l'affichage et l'interaction avec l'utilisateur, en envoyant les actions appropriées au contrôleur pour que celui-ci puisse effectuer les modifications nécessaires sur les données.

View

```
package View;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import Model.Poste;
import Model.Role;


public class EmployeView extends JFrame {
    // D claration des panels
    JPanel pan1 = new JPanel();
```

```
    JPanel pan2 = new JPanel();
    JPanel pan4 = new JPanel();


    // Boutons
    JButton ajouter = new JButton("Ajouter");
    JButton modifier = new JButton("Modifier");
    JButton supprimer = new JButton("Supprimer");
    JButton afficher = new JButton("Afficher");

    // Labels
    JLabel nomLabel = new JLabel("Nom:");
    JLabel prenomLabel = new JLabel("Pr nom:");
    JLabel emailLabel = new JLabel("Email:");
    JLabel telephoneLabel = new JLabel("T l phone:");
    JLabel salaireLabel = new JLabel("Salaire:");
    JLabel roleLabel = new JLabel("R le:");
    JLabel posteLabel = new JLabel("Poste:");


    // Champs de texte
    JTextField nomField = new JTextField(20);
    JTextField prenomField = new JTextField(20);
    JTextField emailField = new JTextField(20);
    JTextField telephoneField = new JTextField(20);
    JTextField salaireField = new JTextField(20);

    public JTable table;
    public DefaultTableModel model;
    public JScrollPane scrollPane;


    public JComboBox<Role> roleComboBox = new JComboBox<>(Role.values());
    public JComboBox<Poste> posteComboBox = new JComboBox<>(Poste.values());

    public EmployeView() {
        // Configuration de la fen tre principale
        setTitle("Exemple de MySQL");
        setSize(500, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Configuration du panel principal
        pan1.setLayout(new BorderLayout());
        pan1.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); // Ajout
    d'un espace int rieur

        // Configuration du panel contenant les champs
        pan2.setLayout(new GridLayout(7, 2, 2, 2)); // R duction de l'espacement
    horizontal et vertical
        pan2.add(nomLabel);
        pan2.add(nomField);
        pan2.add(prenomLabel);
        pan2.add(prenomField);
        pan2.add(emailLabel);
        pan2.add(emailField);
```

```
66          pan2.add(telephoneLabel);
67          pan2.add(telephoneField);
68          pan2.add(salaireLabel);
69          pan2.add(salaireField);
70          pan2.add(roleLabel);
71          pan2.add(roleComboBox);
72          pan2.add(posteLabel);
73          pan2.add(posteComboBox);
74
75          // Configuration du panel pour les boutons
76          pan4.setLayout(new GridLayout(1, 4, 8, 8));
77          pan4.add(ajouter);
78          pan4.add(modifier);
79          pan4.add(supprimer);
80          pan4.add(afficher);
81
82          String[] columnNames = {"Id","Nom", "Prenom", "Telephone", "Email", "
    Salaire", "Role", "Poste"};
83          model = new DefaultTableModel(columnNames, 0);
84          table = new JTable(model);
85          scrollPane = new JScrollPane(table);
86
87
88          pan1.add(pan2, BorderLayout.NORTH);
89          pan1.add(pan4, BorderLayout.SOUTH);
90          pan1.add(scrollPane, BorderLayout.CENTER);   // Table en bas
91
92          add(pan1);
93
94          // Rendre la fen t re visible
95          setVisible(true);
96      }
97
98      // Getters for buttons
99      public JButton getAjouterButton() {
100         return ajouter;
101     }
102
103     public JButton getModifierButton() {
104         return modifier;
105     }
106
107     public JButton getSupprimerButton() {
108         return supprimer;
109     }
110
111     public JButton getAfficherButton() {
112         return afficher;
113     }
114
115     // Getters for fields
116     public String getNom() {
117         return nomField.getText();
118     }
119
```

```
120
121     public String getPrenom() {
122         return prenomField.getText();
123     }
124
125     public String getEmail() {
126         return emailField.getText();
127     }
128
129     public String getTelephone() {
130         return telephoneField.getText();
131     }
132
133     public double getSalaire() {
134         return Double.parseDouble(salaireField.getText());
135     }
136
137     public Role getRole() {
138         return (Role) roleComboBox.getSelectedItem();
139     }
140
141     public Poste getPoste() {
142         return (Poste) posteComboBox.getSelectedItem();
143     }
144
145     public void afficherMessageErreur(String message) {
146         JOptionPane.showMessageDialog(this, message, "Erreur", JOptionPane.
    ERROR_MESSAGE);
147     }
148
149     public void afficherMessageSucces(String message) {
150         JOptionPane.showMessageDialog(this, message, "Succ s", JOptionPane.
    INFORMATION_MESSAGE);
151     }
152
153
154 }
```

Résultats

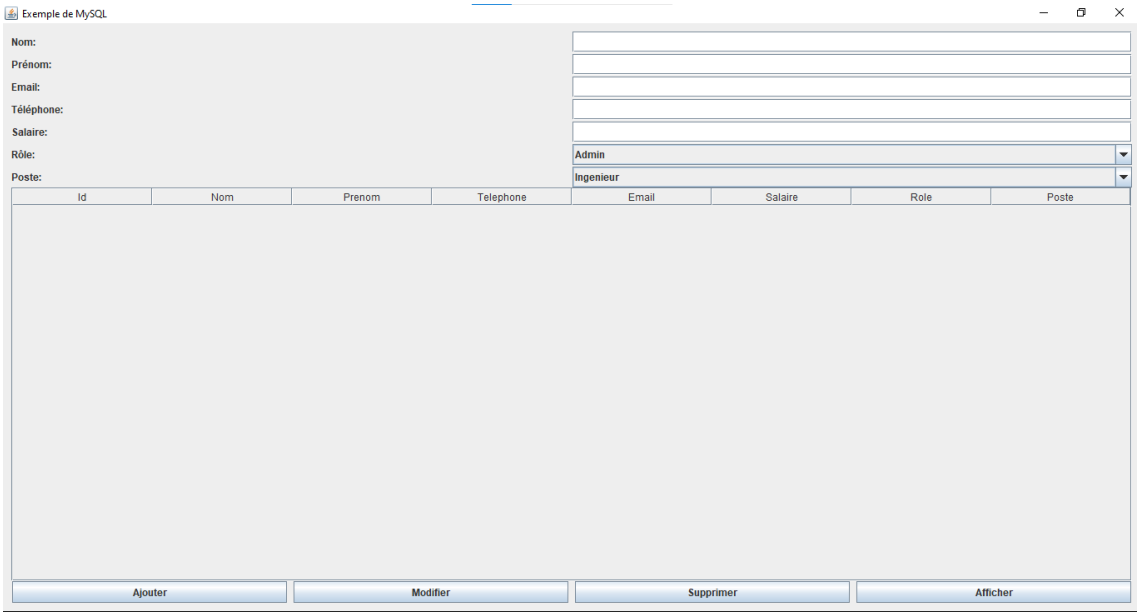

FIGURE 2.1 – Interface Utilisateur

1 Button ajouter

Lorsque l'utilisateur clique sur le bouton "Ajouter", les informations saisies sont enregistrées dans la base de données.

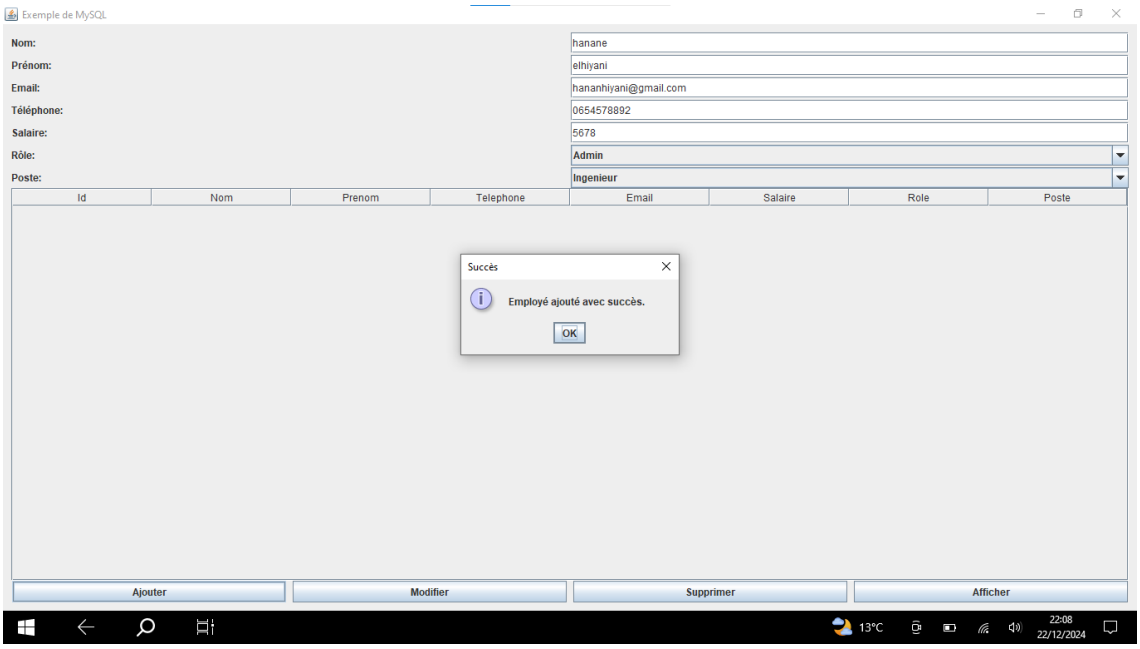

FIGURE 2.2 – Resultat Ajout

2 Button modifier

Pour modifier une entrée, l'utilisateur doit d'abord sélectionner la ligne correspondante dans le tableau de l'interface graphique. Ensuite, en cliquant sur le bouton "Modifier", les informations de cette ligne sont chargées dans les champs de saisie. Après avoir apporté les modifications nécessaires, un second clic sur le bouton "Modifier" met à jour les données dans la base de données et actualise le tableau.

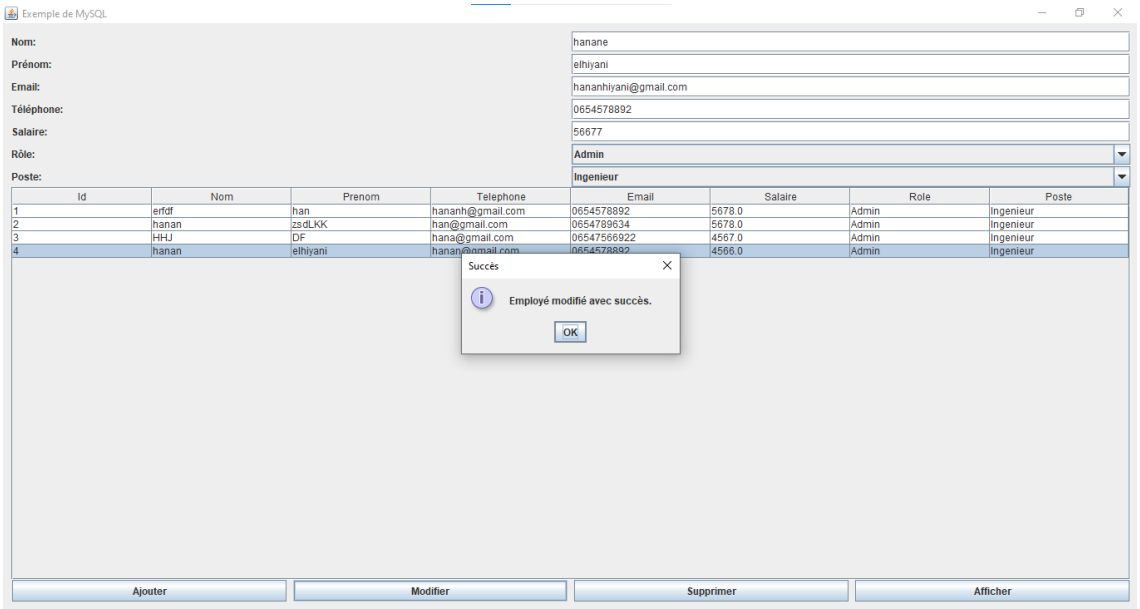

FIGURE 2.3 – Resultat de modification

3 Button afficher

Lorsque l'utilisateur clique sur ce bouton, tous les employés de la base de données seront affichés.

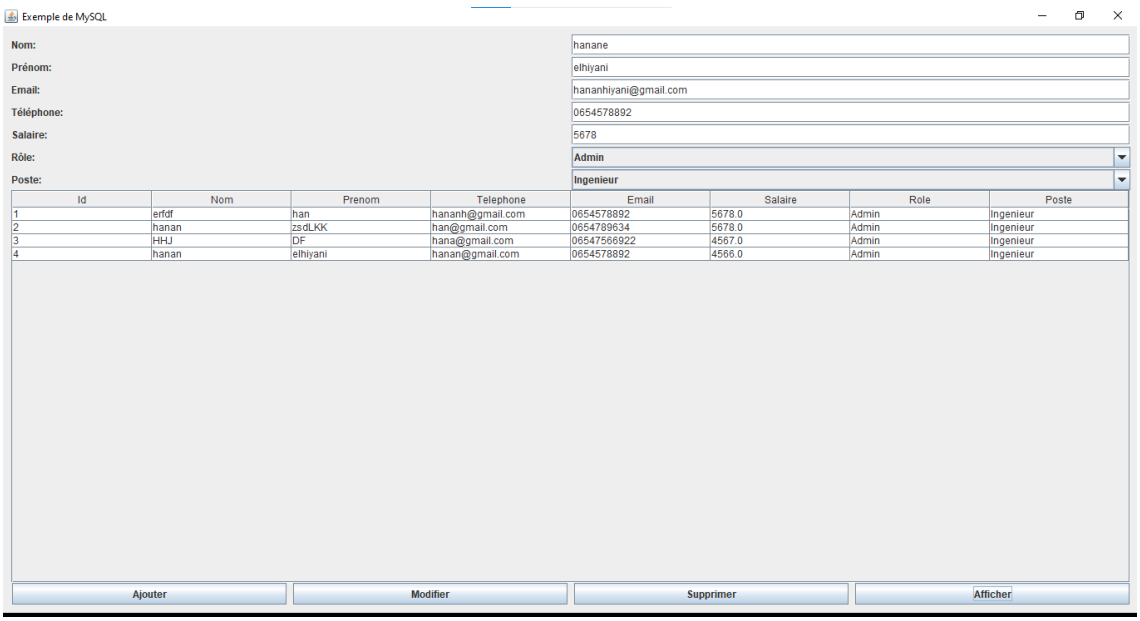

FIGURE 2.4 – Resultat de modification

4 Button supprimer

Pour supprimer une entrée, l'utilisateur doit sélectionner la ligne correspondante dans le tableau de l'interface graphique. Ensuite, en cliquant sur le bouton "Supprimer", un message de confirmation peut s'afficher pour éviter les suppressions accidentelles. Si l'utilisateur confirme, l'entrée est supprimée de la base de données et le tableau est mis à jour automatiquement.

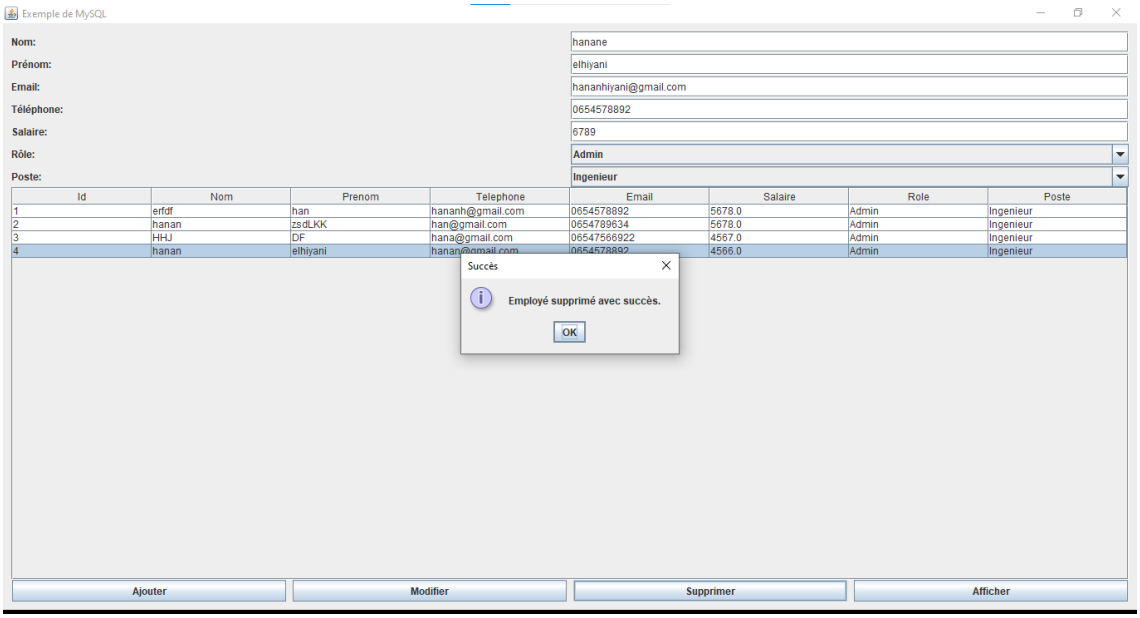

FIGURE 2.5 – Resultat de suppression

Conclusion

En conclusion, ce TP a permis de développer une application de gestion des employés en adoptant l'architecture **MVC**. Cette approche a facilité la séparation des responsabilités entre la logique métier, l'interface utilisateur et le traitement des données, garantissant ainsi une application modulaire, évolutive et facile à maintenir. L'intégration des fonctionnalités telles que l'ajout, la mise à jour et la suppression des employés a renforcé notre maîtrise des concepts de programmation orientée objet et de gestion des interfaces graphiques en Java. Ce travail pratique met en évidence l'importance d'une organisation claire et structurée du code pour créer des applications robustes et performantes.