Compte Rendu
Gestion des Employés en Java (DAO/MVC)

Douae SAMTI
https://github.com/douae000/TP-Gestion$_d$es$_e$mploy − s

December 13, 2024

Contents

1 Introduction

Dans ce projet, nous avons développé une application Java pour la gestion des employés, utilisant l'architecture DAO (Data Access Object) et le modèle MVC (Model-View-Controller). L'objectif principal était d'assurer la modularité, la maintenabilité et l'efficacité dans les opérations de gestion des employés.

2 Architecture et Étapes Suivies

2.1 Architecture du projet

L'architecture du projet suit le modèle MVC, divisant le code en trois couches principales :

1. **Modèle (Model)** : Contient la logique métier et les classes représentant les entités (par exemple, `Employee`, `Role`, `Poste`).
2. **Vue (View)** : Gère l'affichage et l'interaction utilisateur (par exemple, `EmployeeView`).
3. **Contrôleur (Controller)** : Relie le modèle à la vue et gère les interactions utilisateur (par exemple, `EmployeeController`).

2.2 Étapes suivies

Nous avons suivi les étapes suivantes :

1. **Conception de la base de données** :
   - Création d'une base de données MySQL contenant une table `Employe` avec des colonnes pour chaque attribut (nom, prénom, email, téléphone, salaire, rôle, poste).
2. **Développement du module DAO** :
   - Création de l'interface `EmployeeDAO` définissant les opérations de gestion des employés.
   - Implémentation de cette interface dans la classe `EmployeeDAOImpl`, en utilisant des requêtes SQL préparées.
3. **Développement des classes du modèle** :
   - Création de la classe `Employee` pour représenter les employés et des énumérations `Role` et `Poste` pour leurs rôles et postes respectifs.
4. **Création de la vue** :
   - Développement de l'interface utilisateur avec des champs pour entrer les détails des employés et des boutons pour effectuer des opérations CRUD.
5. **Développement du contrôleur** :
   - Gestion des actions utilisateur (ajout, suppression, modification, et affichage des employés).

3 Explications des Codes

3.1 Modèle (`Employee`, `Role`, `Poste`)

- **Objectif** : Représenter les données manipulées par l'application.
- **Détails** :
  - `Employee` contient des attributs comme `nom`, `prenom`, `email`, etc., avec des `getters` et `setters`.
  - `Role` et `Poste` sont des énumérations définissant des valeurs prédéfinies.

3.2 Classe `Employee`

Listing 1: Classe Employee

```
1
2  package Model;
3  // Classe repr sentant un employ  dans le syst me
4  public class Employee {
5      // Attributs priv s pour stocker les informations de l'employ
6      private int id;              // Identifiant unique de l'employ
7      private String nom;          // Nom de l'employ
8      private String prenom;       // Pr nom de l'employ
9      private String email;        // Adresse email de l'employ
10     private String phone;        // Num ro de t l phone de l'employ
11     private double salaire;      // Salaire de l'employ
12     private Role role;           // R le de l'employ  (associ     une
           autre classe)
13     private Poste poste;         // Poste occup  par l'employ  (
          associ     une autre classe)
14
15     // Constructeur avec param tres pour initialiser un nouvel
          employ
16     public Employee(String nom, String prenom, String email, String
          phone, double salaire, Role role, Poste poste) {
17         this.nom = nom;
18         this.prenom = prenom;
19         this.email = email;
20         this.phone = phone;
21         this.salaire = salaire;
22         this.role = role;
23         this.poste = poste;
24     }
25
26     // Constructeur par d faut (sans param tres)
27     public Employee() {
28         // Permet la cr ation d'un objet sans initialisation
              imm diate des attributs
29     }
30
31     // Accesseurs et mutateurs (getters et setters) pour chaque
          attribut
32
```

```
33      // Obtient l'identifiant de l'employ
34      public int getId() { return id; }
35
36      // D finit l'identifiant de l'employ
37      public void setId(int id) { this.id = id; }
38
39      // Obtient le nom de l'employ
40      public String getNom() { return nom; }
41
42      // D finit le nom de l'employ
43      public void setNom(String nom) { this.nom = nom; }
44
45      // Obtient le pr nom de l'employ
46      public String getPrenom() { return prenom; }
47
48      // D finit le pr nom de l'employ
49      public void setPrenom(String prenom) { this.prenom = prenom; }
50
51      // Obtient l'adresse email de l'employ
52      public String getEmail() { return email; }
53
54      // D finit l'adresse email de l'employ
55      public void setEmail(String email) { this.email = email; }
56
57      // Obtient le num ro de t l phone de l'employ
58      public String getPhone() { return phone; }
59
60      // D finit le num ro de t l phone de l'employ
61      public void setPhone(String phone) { this.phone = phone; }
62
63      // Obtient le salaire de l'employ
64      public double getSalaire() { return salaire; }
65
66      // D finit le salaire de l'employ
67      public void setSalaire(double salaire) { this.salaire = salaire; }
68
69      // Obtient le r le de l'employ
70      public Role getRole() { return role; }
71
72      // D finit le r le de l'employ
73      public void setRole(Role role) { this.role = role; }
74
75      // Obtient le poste de l'employ
76      public Poste getPoste() { return poste; }
77
78      // D finit le poste de l'employ
79      public void setPoste(Poste poste) { this.poste = poste; }
80
81      // M thode pour changer le nom de l'employ  (fonctionnalit
            incompl te)
82      public void setnom(String nouveauNom) {
83          // Actuellement, cette m thode est vide, elle pourrait  tre
                impl ment e
```

```
84          // pour ajouter des validations ou des fonctionnalit s
                sp cifiques
85      }
86  }
```

3.3 Modèle (`Employee`, `Role`, `Poste`)

- **Objectif** : Représenter les données manipulées par l'application.
- **Détails** :
  - `Employee` contient des attributs comme `nom`, `prenom`, `email`, etc., avec des `getters` et `setters`.
  - `Role` et `Poste` sont des énumérations définissant des valeurs prédéfinies.

3.4 Code Source

Listing 2: Enumération Poste

Classe `Poste`

```
1  package Model;
2
3  public enum Poste {
4      INGENIEURE_ETUDE_ET_DEVELOPPEMENT,
5      TEAM_LEADER,
6      PILOTE
7  }
```

Listing 3: Enumération Role

Classe `Role`

```
1  package Model;
2
3  public enum Role {
4      ADMIN,
5      EMPLOYE
6  }
```

3.5 Connexion à la base de données (`DBConnection`)

- **Objectif** : Gérer la connexion à la base de données MySQL.
- **Détails** :
  - Utilise le driver JDBC pour se connecter à la base de données TP7.
  - Gère les exceptions SQL en affichant des messages d'erreur pertinents.

3.6 Code Source

Listing 4: Connexion à la base de données

Classe `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);
    }
}
```

3.7 DAO (`EmployeeDAOImpl`)

- **Objectif** : Effectuer les opérations CRUD sur la base de données.
- **Fonctionnalités principales** :
  - `add()` : Insère un nouvel employé dans la base de données.
  - `delete()` : Supprime un employé en fonction de son ID.
  - `listAll()` : Récupère tous les employés sous forme de liste.
  - `update()` : Met à jour un employé existant.

3.8 Code Source

Listing 5: DAO EmployeeDAOImpl

Classe `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 EmployeeDAOI {

    @Override
    public void add(Employee employee) {
        String sql = "INSERT␣INTO␣Employe␣(nom,␣prenom,␣email,␣phone,␣
            salaire,␣role,␣poste)␣VALUES␣(?,␣?,␣?,␣?,␣?,␣?,␣?)";
```

```
16          try (Connection conn = DBConnection.getConnection();
                PreparedStatement stmt = conn.prepareStatement(sql)) {
17              stmt.setString(1, employee.getNom());
18              stmt.setString(2, employee.getPrenom());
19              stmt.setString(3, employee.getEmail());
20              stmt.setString(4, employee.getPhone());
21              stmt.setDouble(5, employee.getSalaire());
22              stmt.setString(6, employee.getRole().name());
23              stmt.setString(7, employee.getPoste().name());
24              stmt.executeUpdate();
25          } catch (SQLException e) {
26              e.printStackTrace();
27          }
28      }
29
30      @Override
31      public void delete(int id) {
32          String sql = "DELETE FROM Employe WHERE id = ?";
33          try (Connection conn = DBConnection.getConnection();
                PreparedStatement stmt = conn.prepareStatement(sql)) {
34              stmt.setInt(1, id);
35              stmt.executeUpdate();
36          } catch (SQLException e) {
37              e.printStackTrace();
38          }
39      }
40
41      @Override
42      public List<Employee> listAll() {
43          List<Employee> employees = new ArrayList<>();
44          String sql = "SELECT * FROM Employe";
45          try (Connection conn = DBConnection.getConnection();
                PreparedStatement stmt = conn.prepareStatement(sql);
46                ResultSet rs = stmt.executeQuery()) {
47
48              while (rs.next()) {
49                  String roleStr = rs.getString("role").toUpperCase();
50                  String posteStr = rs.getString("poste").toUpperCase();
51
52                  Role role = null;
53                  Poste poste = null;
54                  try {
55                      role = Role.valueOf(roleStr);
56                  } catch (IllegalArgumentException e) {
57                      System.out.println("Role non valide : " + roleStr);
58                      role = Role.EMPLOYE; // Valeur par d faut
59                  }
60
61                  try {
62                      poste = Poste.valueOf(posteStr);
63                  } catch (IllegalArgumentException e) {
64                      System.out.println("Poste non valide : " + posteStr
                            );
```

```
                    poste = Poste.INGENIEURE_ETUDE_ET_DEVELOPPEMENT; // Valeur par d faut
                }

                Employee employee = new Employee(
                        rs.getString("nom"),
                        rs.getString("prenom"),
                        rs.getString("email"),
                        rs.getString("phone"),
                        rs.getDouble("salaire"),
                        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());
            stmt.setString(7, employee.getPoste().name());
            stmt.setInt(8, id);

            int rowsUpdated = stmt.executeUpdate();

            if (rowsUpdated > 0) {
                System.out.println("L'employ a  t  mis  jour avec succ s.");
            } else {
                System.out.println("Aucun employ  trouv  avec cet ID.");
            }

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

}
```

listings xcolor

3.9 DAO (EmployeeDAO)

Code source :

```
// Interface d finissant les op rations de gestion des employ s
package DAO;

import Model.Employee;
import java.util.List;

public interface EmployeeDAOI {
    void add(Employee employee); // Ajouter un employ
    void delete(int id);         // Supprimer un employ
    List<Employee> listAll();    // Lister tous les employ s
    Employee findById(int id);   // Trouver un employ  par ID
    void update(Employee employee, int id); // Mettre    jour un
        employ
}
```

3.10 Vue (EmployeeView)

Objectif : Fournir une interface utilisateur pour interagir avec l'application.
Détails :

- Contient des champs de saisie pour les détails des employés (nom, prénom, etc.).
- Des boutons permettent d'exécuter les différentes actions CRUD.

Code source :

```
package View;

import javax.swing.*;
import java.awt.*;

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

    public EmployeeView() {
        setTitle("Gestion des Employ s");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        JPanel inputPanel = new JPanel(new GridLayout(0, 2, 10, 10));

        inputPanel.add(new JLabel("Nom:"));
        nameField = new JTextField();
        inputPanel.add(nameField);

        inputPanel.add(new JLabel("Pr nom:"));
        surnameField = new JTextField();
        inputPanel.add(surnameField);

        inputPanel.add(new JLabel("Email:"));
        emailField = new JTextField();
        inputPanel.add(emailField);

        inputPanel.add(new JLabel("T l phone:"));
        phoneField = new JTextField();
        inputPanel.add(phoneField);

        inputPanel.add(new JLabel("Salaire:"));
        salaryField = new JTextField();
        inputPanel.add(salaryField);

        inputPanel.add(new JLabel("R le:"));
        roleCombo = new JComboBox<>(new String[]{"Admin", "Employe"});
        inputPanel.add(roleCombo);

        inputPanel.add(new JLabel("Poste:"));
        posteCombo = new JComboBox<>(new String[]{
            "INGENIEURE_ETUDE_ET_DEVELOPPEMENT",
            "TEAM_LEADER",
            "PILOTE"
```

```
        });
        inputPanel.add(posteCombo);

        add(inputPanel, BorderLayout.NORTH);

        employeeTable = new JTable();
        add(new JScrollPane(employeeTable), BorderLayout.CENTER);

        JPanel buttonPanel = new JPanel();
        addButton = new JButton("Ajouter");
        buttonPanel.add(addButton);
        listButton = new JButton("Afficher");
        buttonPanel.add(listButton);
        deleteButton = new JButton("Supprimer");
        buttonPanel.add(deleteButton);
        modifyButton = new JButton("Modifier");
        buttonPanel.add(modifyButton);

        add(buttonPanel, BorderLayout.SOUTH);
    }
}
```

3.11 Contrôleur (EmployeeController)

Objectif : Gérer les interactions utilisateur et coordonner les actions entre la vue et le modèle.

Fonctionnalités principales :

- `addEmployee()`: Ajoute un nouvel employé en récupérant les entrées utilisateur et en appelant `EmployeeDAOImpl.add`.
- `listEmployees()`: Affiche tous les employés en remplissant un tableau dans l'interface utilisateur.
- `deleteEmployee()`: Supprime un employé en fonction de son ID.
- `modifyEmployee()`: Met à jour les informations d'un employé existant.

Code source :

```
package Controller;

import DAO.EmployeeDAOI;
import DAO.EmployeeDAOImpl;
import Model.Employee;
import Model.Poste;
import Model.Role;
import View.EmployeeView;

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 EmployeeDAOI dao;

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

        view.addButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                addEmployee();
            }
        });

        view.listButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                listEmployees();
            }
        });

        view.deleteButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                deleteEmployee();
            }
        });

        view.modifyButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                modifyEmployee();
            }
        });
    }

    private void addEmployee() {
        try {
            String nom = view.nameField.getText();
            String prenom = view.surnameField.getText();
            String email = view.emailField.getText();
            String phone = view.phoneField.getText();
            double salaire =
               Double.parseDouble(view.salaryField.getText());
            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);
```

```
65              dao.add(employee);
66              JOptionPane.showMessageDialog(view, "Employ  ajout  avec
                    succ s.");
67          } catch (Exception ex) {
68              JOptionPane.showMessageDialog(view, "Erreur: " +
                    ex.getMessage());
69          }
70      }
71  }
```

3.12 Classe principale (Main)

Objectif : Démarrer l'application.

Détails :

- Crée une instance de `EmployeeView` et la passe à `EmployeeController`.
- Révèle l'interface utilisateur avec `setVisible(true)`.

Code source :

```
1  // Classe principale pour d marrer l'application
2  package Main;
3
4  import Controller.EmployeeController;
5  import View.EmployeeView;
6
7  public class Main {
8      public static void main(String[] args) {
9          // Cr ation de la vue principale
10         EmployeeView view = new EmployeeView();
11
12         // Lien entre la vue et le contr leur
13         new EmployeeController(view);
14
15         // Affichage de l'interface utilisateur
16         view.setVisible(true);
17     }
18 }
```

4 Fonctionnement de l'application

Cette section présente les interfaces graphiques du projet et leurs fonctionnalités principales : ajout, affichage, suppression, et modification.

4.1 Ajout d'un employé

La capture d'écran ci-dessous illustre l'interface permettant d'ajouter un employé. L'utilisateur peut remplir les champs requis (nom, poste, etc.) et cliquer sur le bouton `Ajouter` pour enregistrer les informations dans le système.

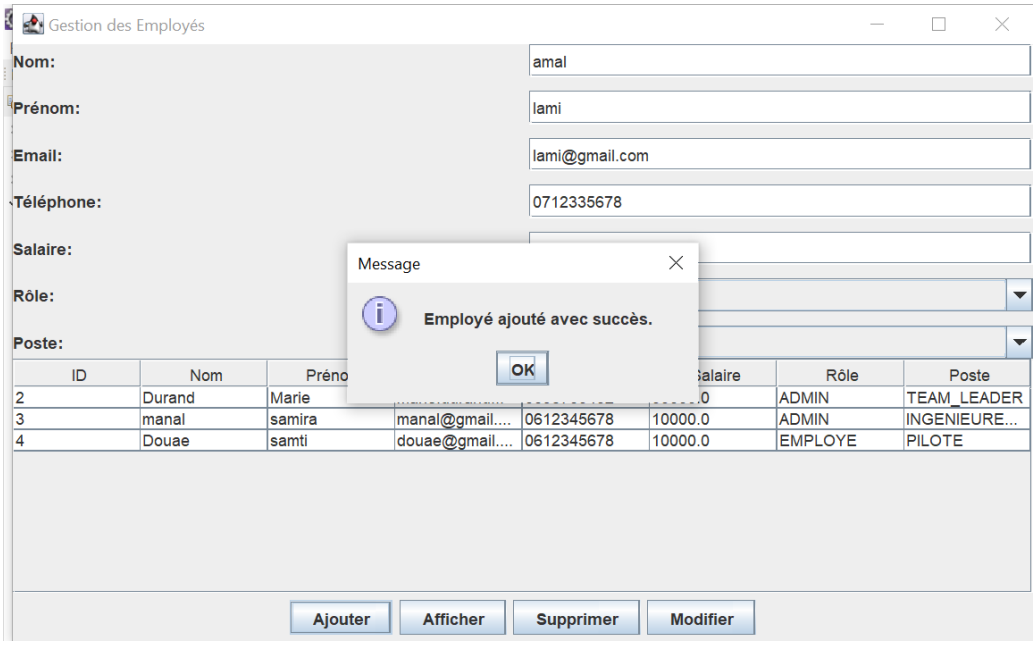

Figure 1: Interface pour l'ajout d'un employé.

4.2 Affichage des employés

L'interface graphique pour l'affichage permet de visualiser tous les employés enregistrés dans le système. Elle comprend une table listant les informations importantes telles que le nom, l'ID, et le poste.

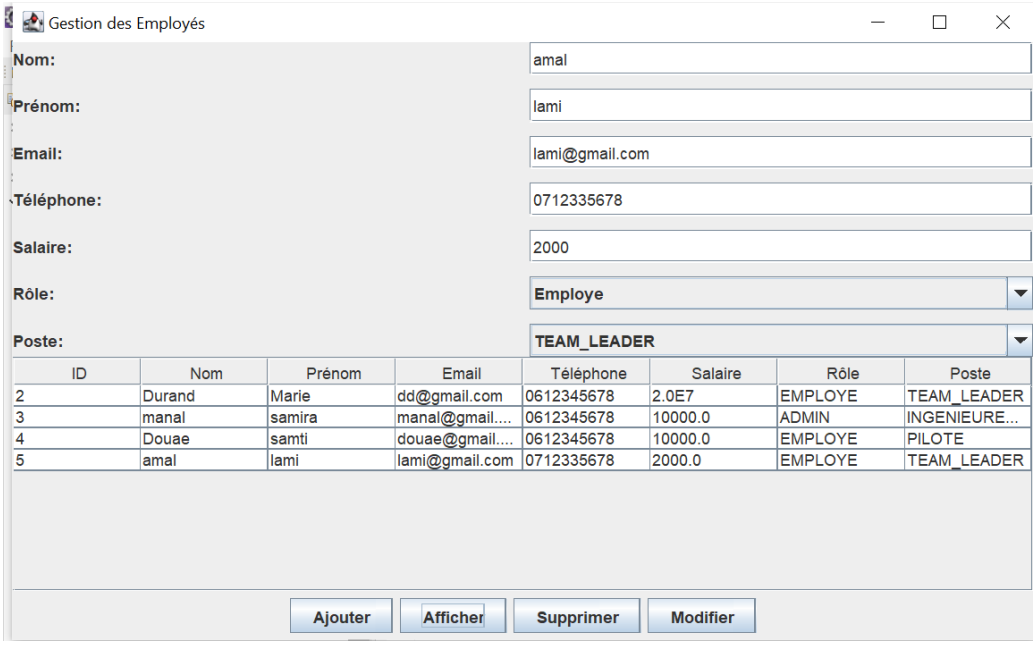

Figure 2: Interface pour l'affichage des employés.

4.3 Suppression d'un employé

L'option de suppression permet de sélectionner un employé dans la liste et de le retirer du système. Une confirmation peut être requise avant de supprimer l'employé.

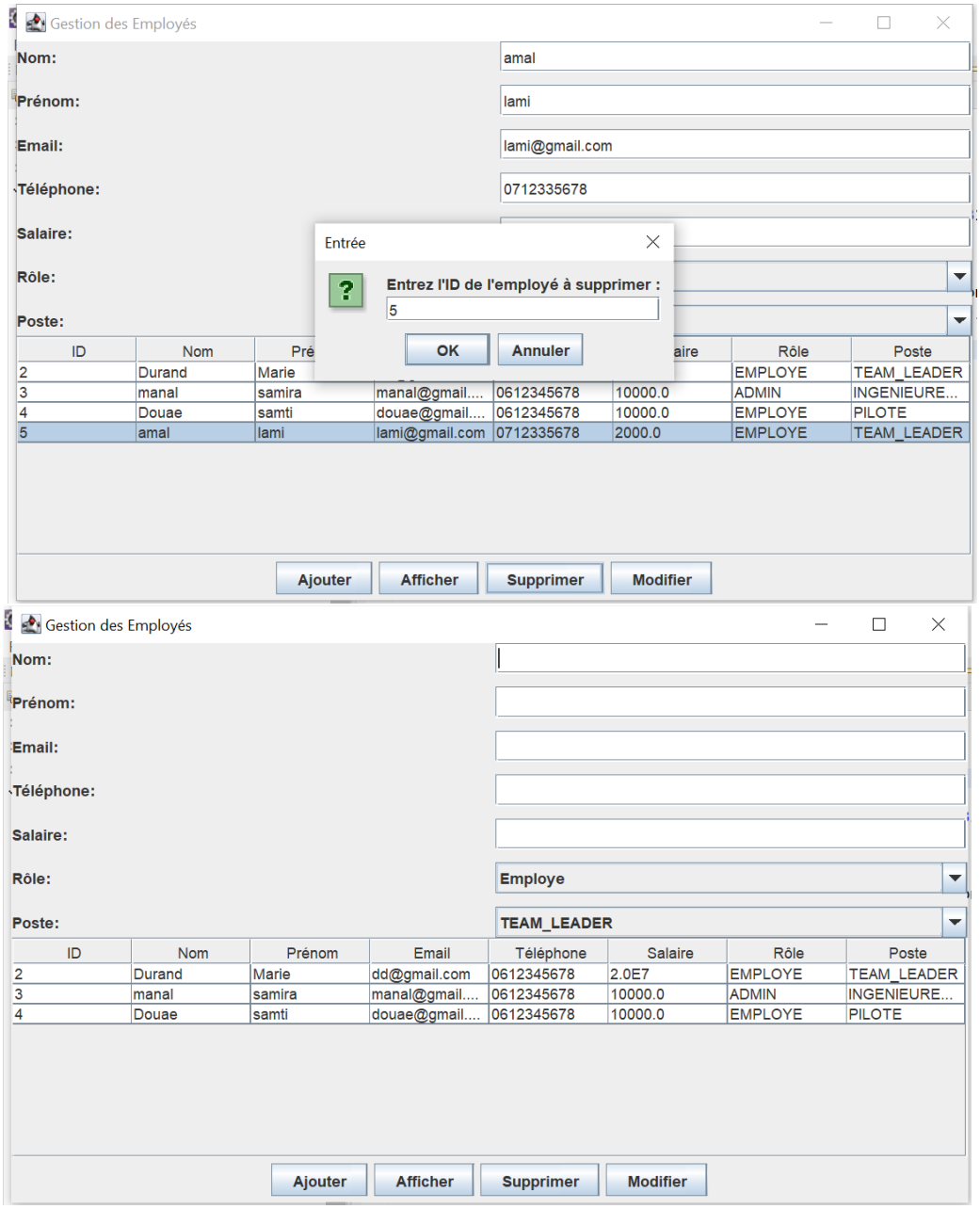

Figure 3: Interface pour la suppression d'un employé.

4.4 Modification des informations d'un employé

Cette fonctionnalité permet de modifier les informations d'un employé existant. L'utilisateur sélectionne un employé, met à jour les champs nécessaires, puis enregistre les modifications.

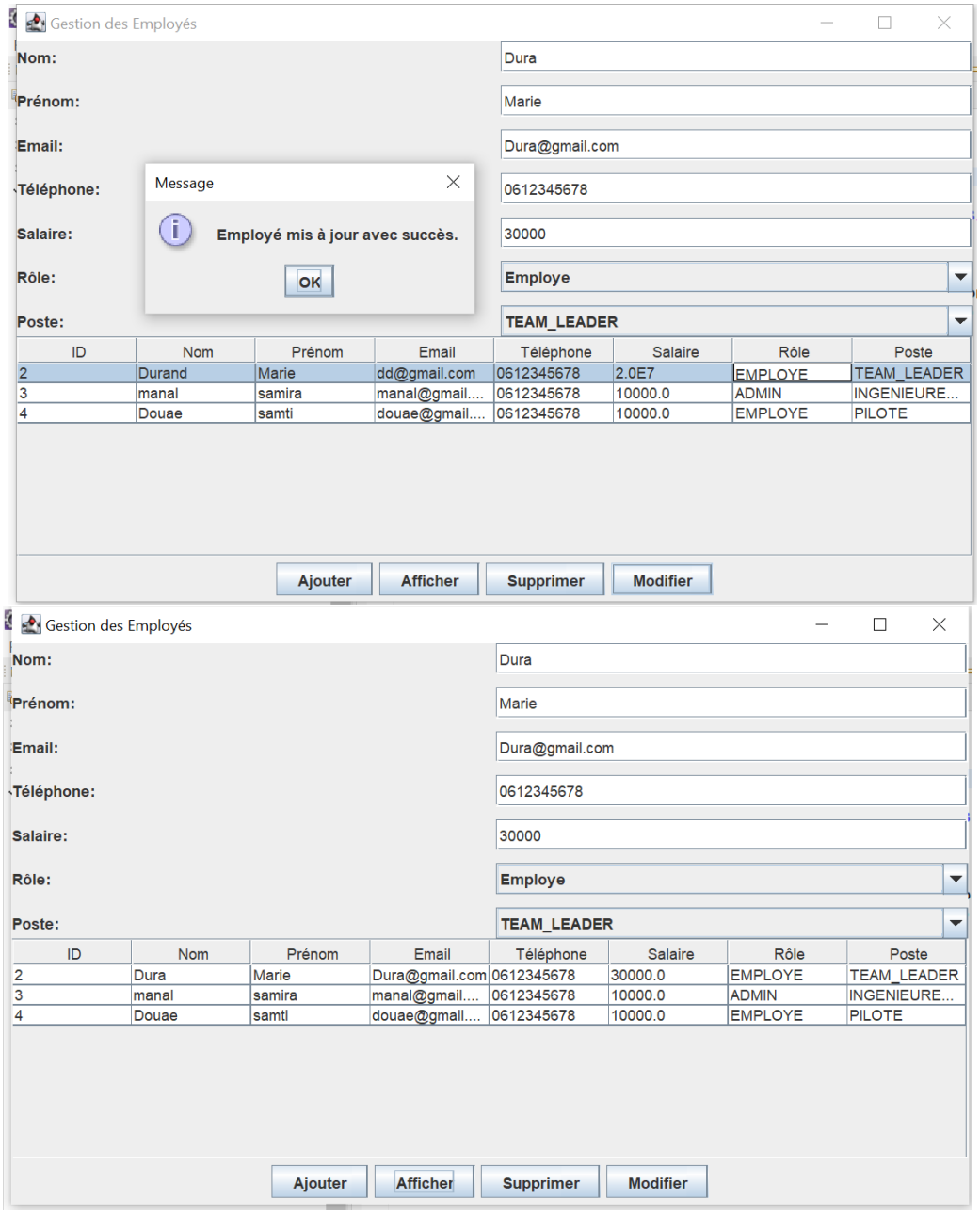

Figure 4: Interface pour la modification des informations d'un employé.

5 Conclusion

Ce projet a offert une opportunité concrète de mettre en œuvre le modèle d'architecture DAO/MVC pour développer une application robuste et bien structurée. La couche DAO (Data Access Object) a permis une gestion efficace des interactions avec la base de données, garantissant un accès centralisé et sécurisé aux données. La couche MVC (Model-View-Controller) a structuré l'application en séparant les responsabilités : le modèle gère les données et leur logique métier, la vue est dédiée à l'interface utilisateur, et le contrôleur assure la coordination entre les deux. Cette organisation modulaire a considérablement amélioré la maintenabilité du code, rendant le projet plus flexible face à des évolutions futures, comme l'ajout de nouvelles fonctionnalités ou la modification de l'interface graphique.