



Université Cadi Ayyad  
École Supérieure De Technologie - Safi  
Département : Informatique  
Filière : Génie Informatique (GI)

---

# Compte Rendu TP1

## Gestion des Employés en Java (DAO/MVC)

---

Réalisée par : AMAL ELLAOUI

Encadrée par : Mme ILHAM KACHBAL // LEILA ELKHROF

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Architecture et Méthodologie</b>	<b>2</b>
2.1	Structure du projet . . . . .	2
2.2	Processus de développement . . . . .	2
<b>3</b>	<b>Analyse des composants du code</b>	<b>3</b>
3.1	Modèle (Employee, Role, Poste) . . . . .	3
3.2	Classe Employee . . . . .	3
3.3	Modèle (Employee, Role, Poste) . . . . .	5
3.4	Code Source . . . . .	5
3.5	Connexion à la base de données (DBConnection) . . . . .	5
3.6	Code Source . . . . .	6
3.7	DAO (EmployeeDAOImpl) . . . . .	6
3.8	Code Source . . . . .	6
3.9	DAO (EmployeeDAO) . . . . .	9
3.10	Vue (EmployeeView) . . . . .	9
3.11	Contrôleur (EmployeeController) . . . . .	11
3.12	Classe principale (Main) . . . . .	13
<b>4</b>	<b>Fonctionnement de l'application</b>	<b>13</b>
4.1	Ajout d'un employé . . . . .	13
4.2	Affichage des employés . . . . .	14
4.3	Suppression d'un employé . . . . .	15
4.4	Modification des informations d'un employé . . . . .	16
<b>5</b>	<b>Conclusion</b>	<b>17</b>

# Introduction

## Architecture et Méthodologie

### Structure du projet

L'architecture du projet suit le modèle MVC, qui organise le code en trois composants essentiels :

1. **Modèle (Model)** : Gère la logique métier et les entités de l'application (par exemple, `Employee`, `Role`, `Poste`).
2. **Vue (View)** : Responsable de l'affichage et de l'interaction avec l'utilisateur (par exemple, `EmployeeView`).
3. **Contrôleur (Controller)** : Assure la liaison entre le modèle et la vue, tout en traitant les actions des utilisateurs (par exemple, `EmployeeController`).

### Processus de développement

Nous avons suivi les étapes suivantes pour le développement :

1. **Conception de la base de données** :
  - Mise en place d'une base de données MySQL comprenant une table `Employee` avec des colonnes pour les attributs essentiels tels que : `nom`, `prenom`, `email`, `telephone`, `salaire`, `role`, et `poste`.
2. **Développement du module DAO** :
  - Création de l'interface `EmployeeDAO` qui définit les opérations nécessaires à la gestion des employés.
  - Implémentation de l'interface dans la classe `EmployeeDAOImpl`, en utilisant des requêtes SQL préparées.
3. **Création des classes du modèle** :
  - Développement de la classe `Employee` pour représenter les employés.
  - Création des énumérations `Role` et `Poste`, qui définissent des valeurs fixes pour les rôles et postes des employés.
4. **Développement de l'interface utilisateur** :
  - Conception de l'interface utilisateur avec des champs pour saisir les informations des employés et des boutons pour les opérations CRUD (Créer, Lire, Mettre à jour, Supprimer).
5. **Développement du contrôleur** :
  - Gestion des interactions des utilisateurs : ajout, suppression, modification et affichage des employés.

# Analyse des composants du code

## Modèle (Employee, Role, Poste)

- **Objectif** : Représenter les entités et les données manipulées par l'application.
- **Détails** :
  - La classe `Employee` contient des attributs comme `nom`, `prenom`, `email`, avec des méthodes `getters` et `setters`.
  - Les énumérations `Role` et `Poste` définissent des valeurs fixes et prédéfinies pour les rôles et les postes des employés.

## 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 }

```

## 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.

## Code Source

Listing 2: Énumération Poste

```

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

```

Listing 3: Énumération Role

```

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

```

## 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.

## Code Source

Listing 4: Connexion à la base de données

```
1 package DAO;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class DBConnection {
8     private static final String URL = "jdbc:mysql://localhost:3306/TP7";
9     ;
10    private static final String USER = "root";
11    private static final String PASSWORD = "";
12
13    public static Connection getConnection() throws SQLException {
14        return DriverManager.getConnection(URL, USER, PASSWORD);
15    }
16 }
```

## 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.

## Code Source

Listing 5: DAO EmployeeDAOImpl

```
1 package DAO;
2
3 import Model.Employee;
4 import Model.Poste;
5 import Model.Role;
6
7 import java.sql.*;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class EmployeeDAOImpl implements EmployeeDAOI {
12
13     @Override
14     public void add(Employee employee) {
15         String sql = "INSERT INTO Employee (nom, prenom, email, phone, salaire, role, poste) VALUES (?, ?, ?, ?, ?, ?)";
```

```

16     try (Connection conn = DBConnection.getConnection();
17         PreparedStatement stmt = conn.prepareStatement(sql)) {
18         stmt.setString(1, employee.getNom());
19         stmt.setString(2, employee.getPrenom());
20         stmt.setString(3, employee.getEmail());
21         stmt.setString(4, employee.getPhone());
22         stmt.setDouble(5, employee.getSalaire());
23         stmt.setString(6, employee.getRole().name());
24         stmt.setString(7, employee.getPoste().name());
25         stmt.executeUpdate();
26     } catch (SQLException e) {
27         e.printStackTrace();
28     }
29
30     @Override
31     public void delete(int id) {
32         String sql = "DELETE FROM Employee WHERE id=?";
33         try (Connection conn = DBConnection.getConnection();
34             PreparedStatement stmt = conn.prepareStatement(sql)) {
35             stmt.setInt(1, id);
36             stmt.executeUpdate();
37         } catch (SQLException e) {
38             e.printStackTrace();
39         }
40
41     @Override
42     public List<Employee> listAll() {
43         List<Employee> employees = new ArrayList<>();
44         String sql = "SELECT * FROM Employee";
45         try (Connection conn = DBConnection.getConnection();
46             PreparedStatement stmt = conn.prepareStatement(sql);
47             ResultSet rs = stmt.executeQuery()) {
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.EMPLOYEE; // 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);
65                 }

```



```

65         poste = Poste.INGENIEURE_ETUDE_ET_DEVELOPPEMENT; //
           Valeur par d faut
66     }
67
68     Employee employee = new Employee(
69         rs.getString("nom"),
70         rs.getString("prenom"),
71         rs.getString("email"),
72         rs.getString("phone"),
73         rs.getDouble("salaire"),
74         role,
75         poste
76     );
77     employee.setId(rs.getInt("id"));
78     employees.add(employee);
79 }
80 } catch (SQLException e) {
81     e.printStackTrace();
82 }
83 return employees;
84 }
85
86 @Override
87 public Employee findById(int id) {
88     String sql = "SELECT * FROM Employe WHERE id = ?";
89     try (Connection conn = DBConnection.getConnection();
90         PreparedStatement stmt = conn.prepareStatement(sql)) {
91         stmt.setInt(1, id);
92         ResultSet rs = stmt.executeQuery();
93         if (rs.next()) {
94             return new Employee(
95                 rs.getString("nom"),
96                 rs.getString("prenom"),
97                 rs.getString("email"),
98                 rs.getString("phone"),
99                 rs.getDouble("salaire"),
100                 Role.valueOf(rs.getString("role")),
101                 Poste.valueOf(rs.getString("poste"))
102             );
103         }
104     } catch (SQLException e) {
105         e.printStackTrace();
106     }
107     return null;
108 }
109
110 @Override
111 public void update(Employee employee, int id) {
112     String sql = "UPDATE Employe SET nom = ?, prenom = ?, email = ?,
113         phone = ?, salaire = ?, role = ?, poste = ? WHERE id = ?"
114         ;

```

```

113         try (Connection conn = DBConnection.getConnection();
114             PreparedStatement stmt = conn.prepareStatement(sql)) {
115             stmt.setString(1, employee.getNom());
116             stmt.setString(2, employee.getPrenom());
117             stmt.setString(3, employee.getEmail());
118             stmt.setString(4, employee.getPhone());
119             stmt.setDouble(5, employee.getSalaire());
120             stmt.setString(6, employee.getRole().name());
121             stmt.setString(7, employee.getPoste().name());
122             stmt.setInt(8, id);
123
124             int rowsUpdated = stmt.executeUpdate();
125
126             if (rowsUpdated > 0) {
127                 System.out.println("L'employ   a t   mis   jour avec
128                                     succ  s.");
129             } else {
130                 System.out.println("Aucun employ   trouv   avec cet ID.
131                                     ");
132             }
133         } catch (SQLException e) {
134             e.printStackTrace();
135         }
136     }

```

listings xcolor

## DAO (EmployeeDAO)

Code source :

```

1 // Interface d'impl  mentant les op  rations de gestion des employ  s
2 package DAO;
3
4 import Model.Employee;
5 import java.util.List;
6
7 public interface EmployeeDAO {
8     void add(Employee employee); // Ajouter un employ  
9     void delete(int id);         // Supprimer un employ  
10    List<Employee> listAll();      // Lister tous les employ  s
11    Employee findById(int id);    // Trouver un employ   par ID
12    void update(Employee employee, int id); // Mettre    jour un
13                                     employ  
14 }

```

## Vue (EmployeeView)

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

**D  tails :**

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

Code source :

```

1 package View;
2
3 import javax.swing.*;
4 import java.awt.*;
5
6 public class EmployeeView extends JFrame {
7     public JTable employeeTable;
8     public JButton addButton, listButton, deleteButton, modifyButton;
9     public JTextField nameField, surnameField, emailField, phoneField,
10         salaryField;
11     public JComboBox<String> roleCombo, posteCombo;
12
13     public EmployeeView() {
14         setTitle("Gestion des Employés");
15         setSize(800, 600);
16         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17         setLayout(new BorderLayout());
18
19         JPanel inputPanel = new JPanel(new GridLayout(0, 2, 10, 10));
20
21         inputPanel.add(new JLabel("Nom:"));
22         nameField = new JTextField();
23         inputPanel.add(nameField);
24
25         inputPanel.add(new JLabel("Prénom:"));
26         surnameField = new JTextField();
27         inputPanel.add(surnameField);
28
29         inputPanel.add(new JLabel("Email:"));
30         emailField = new JTextField();
31         inputPanel.add(emailField);
32
33         inputPanel.add(new JLabel("Téléphone:"));
34         phoneField = new JTextField();
35         inputPanel.add(phoneField);
36
37         inputPanel.add(new JLabel("Salaire:"));
38         salaryField = new JTextField();
39         inputPanel.add(salaryField);
40
41         inputPanel.add(new JLabel("Rôle:"));
42         roleCombo = new JComboBox<>(new String[]{"Admin", "Employé"});
43         inputPanel.add(roleCombo);
44
45         inputPanel.add(new JLabel("Poste:"));
46         posteCombo = new JComboBox<>(new String[]{
47             "INGENIEURE_ETUDE_ET_DEVELOPPEMENT",
48             "TEAM_LEADER",
49             "PILOTE"
50         });
51     }
52 }

```

```

49     });
50     inputPanel.add(posteCombo);
51
52     add(inputPanel, BorderLayout.NORTH);
53
54     employeeTable = new JTable();
55     add(new JScrollPane(employeeTable), BorderLayout.CENTER);
56
57     JPanel buttonPanel = new JPanel();
58     addButton = new JButton("Ajouter");
59     buttonPanel.add(addButton);
60     listButton = new JButton("Afficher");
61     buttonPanel.add(listButton);
62     deleteButton = new JButton("Supprimer");
63     buttonPanel.add(deleteButton);
64     modifyButton = new JButton("Modifier");
65     buttonPanel.add(modifyButton);
66
67     add(buttonPanel, BorderLayout.SOUTH);
68 }
69 }

```

## 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 :

```

1 package Controller;
2
3 import DAO.EmployeeDAOI;
4 import DAO.EmployeeDAOImpl;
5 import Model.Employee;
6 import Model.Poste;
7 import Model.Role;
8 import View.EmployeeView;
9
10 import javax.swing.*;
11 import javax.swing.table.DefaultTableModel;
12 import java.awt.event.ActionEvent;
13 import java.awt.event.ActionListener;
14 import java.util.List;
15

```

```

16 public class EmployeeController {
17     private final EmployeeView view;
18     private final EmployeeDAOI dao;
19
20     public EmployeeController(EmployeeView view) {
21         this.view = view;
22         this.dao = new EmployeeDAOImpl();
23
24         view.addButton.addActionListener(new ActionListener() {
25             @Override
26             public void actionPerformed(ActionEvent e) {
27                 addEmployee();
28             }
29         });
30
31         view.listButton.addActionListener(new ActionListener() {
32             @Override
33             public void actionPerformed(ActionEvent e) {
34                 listEmployees();
35             }
36         });
37
38         view.deleteButton.addActionListener(new ActionListener() {
39             @Override
40             public void actionPerformed(ActionEvent e) {
41                 deleteEmployee();
42             }
43         });
44
45         view.modifyButton.addActionListener(new ActionListener() {
46             @Override
47             public void actionPerformed(ActionEvent e) {
48                 modifyEmployee();
49             }
50         });
51     }
52
53     private void addEmployee() {
54         try {
55             String nom = view.nameField.getText();
56             String prenom = view.surnameField.getText();
57             String email = view.emailField.getText();
58             String phone = view.phoneField.getText();
59             double salaire =
60                 Double.parseDouble(view.salaryField.getText());
61             Role role
62                 Role.valueOf(view.roleCombo.getSelectedItem().toString()
63                     .toUpperCase());
64             Poste poste =
65                 Poste.valueOf(view.posteCombo.getSelectedItem().toString()
66                     .toUpperCase());
67             Employee employee = new Employee(nom, prenom, email,
68                 phone, salaire, role, poste);

```

```

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

```

## 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 démarrer 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         // Création de la vue principale
10        EmployeeView view = new EmployeeView();
11
12        // Lien entre la vue et le contrôleur
13        new EmployeeController(view);
14
15        // Affichage de l'interface utilisateur
16        view.setVisible(true);
17    }
18 }

```

## Fonctionnement de l'application

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

### 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.

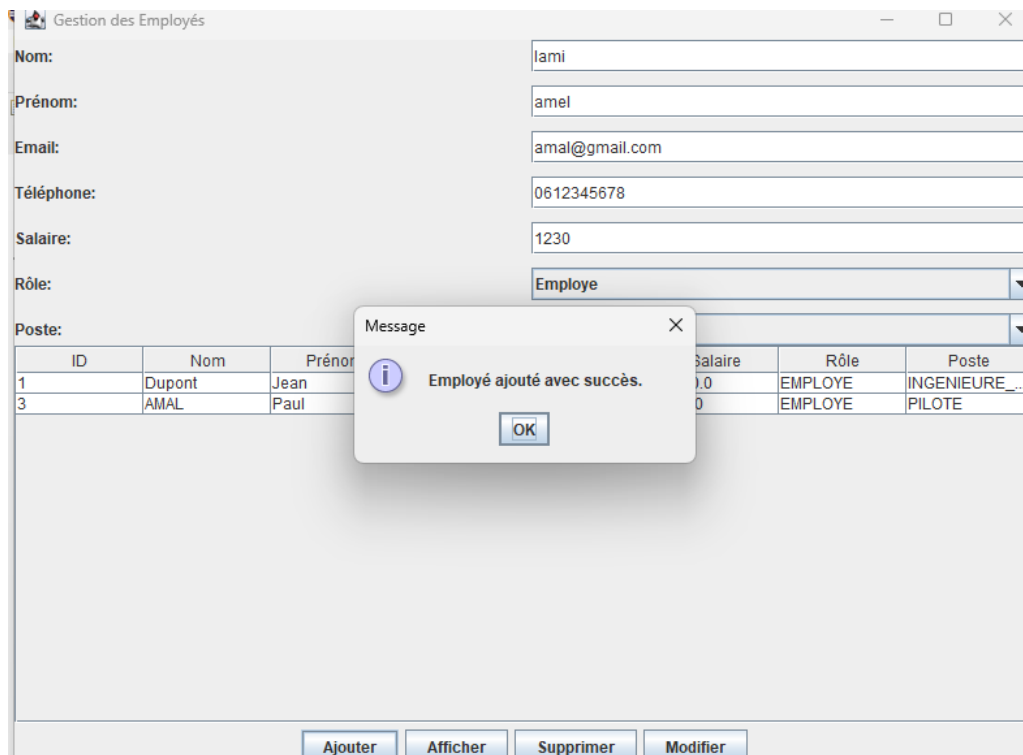


Figure 1: Interface pour l'ajout d'un employé.

## 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.

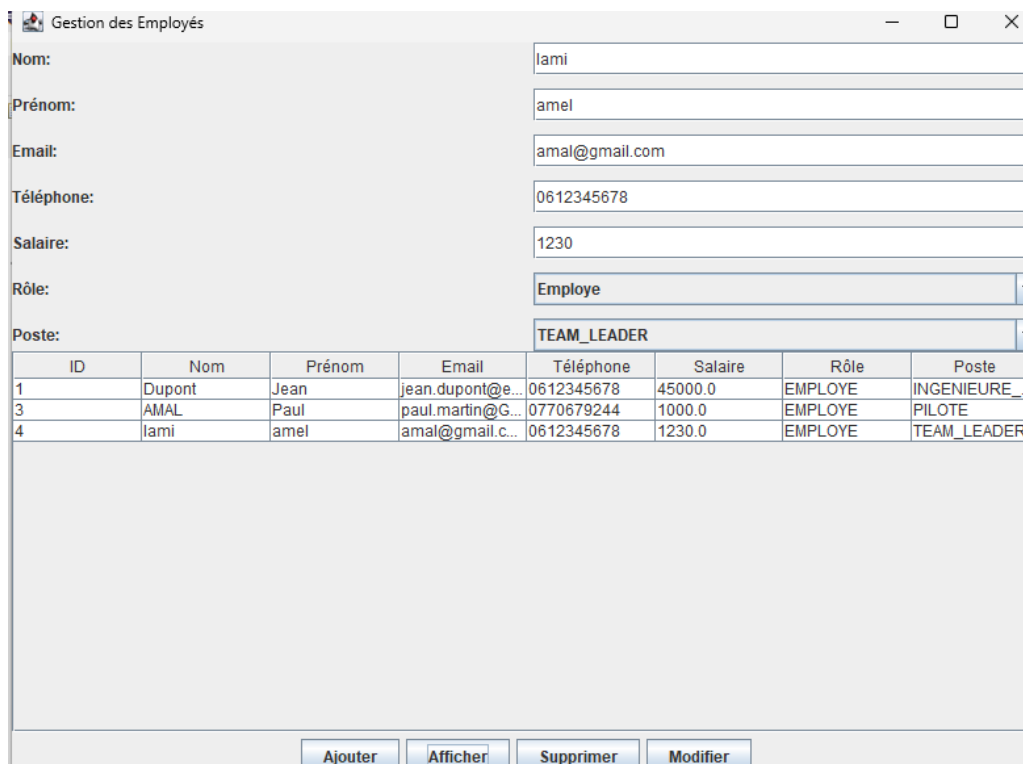


Figure 2: Interface pour l'affichage des employés.

## 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.

The screenshot shows the 'Gestion des Employés' application window. It features a form for adding or editing employee details and a table listing existing employees. A confirmation dialog box titled 'Entrée' is displayed in the center, asking for the ID of the employee to be deleted. The dialog has a green question mark icon, a text input field containing '1', and 'OK' and 'Annuler' buttons.

**Gestion des Employés**

Nom: lami

Prénom: amel

Email: amal@gmail.com

Téléphone: 0612345678

Salaire: 1230

Rôle: Employee

Poste:

ID	Nom	Prénom	Email	Téléphone	Salaire	Rôle	Poste
1	Dupont	Jean				EMPLOYE	INGENIEURE_...
3	AMAL	Paul				EMPLOYE	PILOTE
4	lami	amel				EMPLOYE	TEAM_LEADER

Ajouter Afficher Supprimer Modifier

**Entrée**

Entrez l'ID de l'employé à supprimer :

1

OK Annuler

**Gestion des Employés**

Nom: lami

Prénom: amel

Email: amal@gmail.com

Téléphone: 0612345678

Salaire: 1230

Rôle: Employee

Poste: TEAM\_LEADER

ID	Nom	Prénom	Email	Téléphone	Salaire	Rôle	Poste
3	AMAL	Paul	paul.martin@G...	0770679244	1000.0	EMPLOYE	PILOTE
4	lami	amel	amal@gmail.c...	0612345678	1230.0	EMPLOYE	TEAM_LEADER

Ajouter Afficher Supprimer Modifier

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



## 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.

The screenshot shows the 'Gestion des Employés' application interface. The form contains the following fields:

- Nom: lami
- Prénom: manal
- Email: amal@gmail.com
- Téléphone: 0612345678
- Salaire: 20000
- Rôle: Employee
- Poste: TEAM\_LEADER

A success message dialog is displayed in the center:

Message  
i Employé mis à jour avec succès.  
OK

Below the form is a table with the following data:

ID	Nom	Prénom	Email	Téléphone	Salaire	Rôle	Poste
3	AMAL	Paul	paul.martin@G...	0770679244	1000.0	EMPLOYE	PILOTE
4	lami	manal	amal@gmail.c...	0612345678	20000.0	EMPLOYE	TEAM_LEADER

At the bottom of the application are four buttons: Ajouter, Afficher, Supprimer, and Modifier.

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

## 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.