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

Rapport du TP N°1 (MVC, DAO et Java Swing)

Gestion des employés

Réalisé par : CHABLAOUI Salma

Encadré par : Mme. KACHBAL Ilham

Année Universitaire : 2024/2025

Table des matières

Table des figures

Introduction

Ce travail pratique (TP) s'intéresse à la création d'une application Java pour la gestion des employés. Il est structuré autour de l'architecture **MVC (Model-View-Controller)**, un modèle de conception qui sépare clairement les données, la logique métier et l'interface utilisateur. Ce projet vise à renforcer la maîtrise des concepts de la programmation orientée objet (POO) et à développer des interfaces graphiques avec la bibliothèque **Swing**. L'approche suivie dans ce TP permet également de perfectionner la gestion du code et de garantir une organisation optimale à travers une séparation des responsabilités.

L'application développée dans ce TP a pour but de simplifier la gestion des employés. Elle permet de gérer les informations des employés, avec des fonctionnalités permettant leur ajout, modification, suppression et consultation. L'interface utilisateur, conçue pour être simple et agréable, assure une interaction fluide et efficace avec l'application. L'utilisation de l'architecture MVC facilite la gestion du code, permettant une maintenance facile et l'ajout futur de nouvelles fonctionnalités.

Les principales fonctionnalités de l'application incluent :

- Enregistrement des employés avec leurs données complètes.
- Modification des informations des employés existants.
- Suppression des employés du système.
- Affichage de la liste des employés présents dans la base de données.

Ce projet a pour objectif de démontrer l'efficacité de la programmation orientée objet et l'architecture MVC dans la création d'applications logicielles robustes et évolutives. Il représente une étape clé dans la formation nécessaire pour aborder des projets plus complexes à l'avenir.

Environnement et outils utilisés

1 Environnement de travail

Figure 1 – Eclipse logo

,

- **Eclipse** : Eclipse est un environnement de développement intégré (IDE) open-source, principalement utilisé pour le développement Java, mais aussi extensible pour d'autres langages via des plugins. Il offre des fonctionnalités comme la complétion automatique, le débogage et la gestion de projets, facilitant ainsi la création et le test d'applications logicielles.

2 Outils de travail

Figure 2 – MySQL Workbench logo

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

Figure 3 – xampp logo

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

FIGURE 4 – java developpement kit logo

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

3 Language de Programmation

FIGURE 5 – java logo

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

Réalisation

1 Création de la base de donnée

1.1 Script base de donnée

```sql
-- Cr ation de la base de donn es
CREATE DATABASE EmployeDB;

-- Utilisation de la base de donn es
USE EmployeDB;

-- Cr ation de la table des employ s
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)
);
```

Listing 1.1 – Script SQL de la base de données

- Ce script est ecrit sur MySQL Workbench pour creation la base de donnée pour etre lier à au code via le driver JDBC pour garantir la gestion .

2 Architecture MVC (Model-View-Controller)

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

2.1 Model

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

Employer

```
package Model;
public class Employe {
  private int id;
  public int getId() {
    return id;
  }




  public void setId(int id) {
    this.id = id;
  }
  public Employe(int id, String nom, String prenom, String email, String
   telephone, double salaire, Role role,
      Poste poste) {
    super();
    this.id = id;
    this.nom = nom;
    this.prenom = prenom;
    this.email = email;
    this.telephone = telephone;
    this.salaire = salaire;
    this.role = role;
    this.poste = poste;
  }
  private String nom;
  private String prenom;
  private String email;
  private String telephone;
  private double salaire;
  private Role role;
  private 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;
  }


```

```
46
47 //Getters & Setters
48   public String getNom() {
49     return nom;
50   }
51   public void setNom(String nom) {
52     this.nom=nom;
53   }
54
55
56   public String getPrenom() {
57     return prenom;
58   }
59
60   public String getEmail() {
61     return email;
62   }
63
64   public String getTelephone() {
65     return telephone;
66   }
67
68   public double getSalaire() {
69     return salaire;
70   }
71
72   public Role getRole() {
73     return role;
74   }
75
76   public Poste getPoste() {
77     return poste;
78   }
79
80   public void setPrenom(String prenom) {
81     this.prenom = prenom;
82   }
83
84   public void setEmail(String email) {
85     this.email = email;
86   }
87
88   public void setTelephone(String telephone) {
89     this.telephone = telephone;
90   }
91
92   public void setSalaire(double salaire) {
93     this.salaire = salaire;
94   }
```

```
95
96    public void setRole(Role role) {
97      this.role = role;
98    }
99
100   public void setPoste(Poste poste) {
101     this.poste = poste;
102   }
103
104
105   //Enum rations de Poste Role
106    public enum Role {
107     ADMIN,
108     EMPLOYE
109     }
110   public enum Poste {
111     INGENIEURE,
112     TEAM_LEADER,
113     PILOTE
114     }
115
116 }
```

EmployerMOdel

```
1  package Model;
2
3  import javax.swing.JOptionPane;
4
5  import DAO.EmployeImpl;
6  import Model.Employe.Poste;
7  import Model.Employe.Role;
8
9  public class EmployeModel {
10   private EmployeImpl dao;
11
12   public EmployeModel(EmployeImpl dao) {
13   this.dao=dao;
14 }
15
16 //logique Metier
17 public boolean addEmploye(String nom,String pronom,String email,String telephone
     ,double salaire,Role role,Poste poste) {
18   // V rifie si le salaire est inf rieur ou  gal     z ro
19   if(salaire<=0) {
20       JOptionPane.showMessageDialog(null, "Salaire invalide il est inf rieure
     de z ro !!!!!!!");
21
22     return false;}
```

```
23
24      // V rifie si l'email est nul ou ne contient pas le caract re '@'
25    if (email == null || !email.contains("@")) {
26        JOptionPane.showMessageDialog(null, "L'email n'est pas valide Ajouter '@'
       !!!!!!!");
27
28
29          return false;
30      }
31
32    Employe NvEmploye = new Employe(nom,pronom,email,telephone,salaire,role,poste)
       ;
33    dao.add(NvEmploye);
34  return true;
35
36  }}
```

2.2 DAO

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

connexion

```
1  package DAO;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.SQLException;
6
7  public class connexion {
8    // D finition des param tres de connexion    la base de donn es
9      public static final String url = "jdbc:mysql://localhost:3306/salma";
10     public static final String user = "root";
11     public static final String password = "";
12     private static Connection conn = null;
13
14     // M thode pour  tablir  une connexion    la base de donn es
15     public static Connection getConnexion() {
16         if (conn == null) {
17             try {
18                 conn = DriverManager.getConnection(url, user, password);
19                 System.out.println("Connexion  tablie  avec succ s !");
20             } catch (SQLException e) {
21                 System.out.println("Erreur de connexion !!!!!");
22             }
23         }
24         return conn;
```

```
25      }
26
27      // M thode pour fermer la connexion    la base de donn es
28      public static void closeConnexion() {
29          if (conn != null) {
30              try {
31                  conn.close();
32                  conn = null;
33                  System.out.println("Connexion ferm e avec succ s !");
34              } catch (SQLException e) {
35                  System.out.println("Erreur lors de la fermeture de la
    connexion !!!!!");
36              }
37          }
38      }
39  }
```

EmployeI

```
1  package DAO;
2
3  import java.util.List;
4
5  import Model.Employe;
6  import Model.Employe.Poste;
7  import Model.Employe.Role;
8
9  public interface EmployeI {
10
11 List<Employe> findAll();
12 void add(Employe E);
13 void update(Employe E,int id);
14 void delete(int id);
15
16
17
18 }
```

EmployeImpl

```
1  package DAO;
2
3  import java.sql.Connection;
4  import java.sql.PreparedStatement;
5  import java.sql.ResultSet;
6  import java.sql.SQLException;
7  import java.sql.Statement;
8  import java.util.ArrayList;
```

```
9  import java.util.Arrays;
10 import java.util.List;
11
12 import Model.Employe.Role;
13
14 import Model.Employe;
15 import Model.Employe.Poste;
16
17
18 public class EmployeImpl implements EmployeI {
19     private Connection conn;
20
21     public EmployeImpl() {
22         this.conn = connexion.getConnexion();
23     }
24
25     // Ajouter un nouvel employ  dans la base de donn es
26     @Override
27     public void add(Employe E) {
28         String Query = "INSERT INTO Employee(nom, prenom, email, telephone,
       salaire, role, poste) VALUES(?, ?, ?, ?, ?, ?, ?)";
29
30         try (PreparedStatement stmt = conn.prepareStatement(Query)) {
31             stmt.setString(1, E.getNom());
32             stmt.setString(2, E.getPrenom());
33             stmt.setString(3, E.getEmail());
34             stmt.setString(4, E.getTelephone());
35             stmt.setDouble(5, E.getSalaire());
36             stmt.setString(6, E.getRole().name());
37             stmt.setString(7, E.getPoste().name());
38             stmt.executeUpdate();
39             System.out.println("Employ  ajout  avec succ s !");
40         } catch (SQLException e) {
41             System.out.println("Erreur lors de l'ajout de l'employ  !");
42
43
44         }
45     }
46
47
48     // R cup rer la liste de tous les employ s dans la base de donn es
49     @Override
50   public List<Employe> findAll() {
51       List<Employe> employes = new ArrayList<>();
52       String query = "SELECT * FROM Employee";
53       try (Statement stmt = conn.createStatement();
54            ResultSet rs = stmt.executeQuery(query)) {
55           while (rs.next()) {
56               employes.add(new Employe(
```

```
                    rs.getInt("id"),
                      rs.getString("nom"),
                      rs.getString("prenom"),
                      rs.getString("email"),
                      rs.getString("telephone"),
                      rs.getDouble("salaire"),
                      Employe.Role.valueOf(rs.getString("role")),
                      Poste.valueOf(rs.getString("poste"))
                  ));
              }
          } catch (SQLException e) {
              System.out.println("Erreur lors de la r cup ration de tous les employ s !!!!!");

          }
          return employes;
      }

 // Mettre    jour les informations d'un employ  existant
@Override
public void update(Employe E, int id) {
     String query = "UPDATE Employee SET nom = ?, prenom = ?, email = ?, telephone = ?, salaire = ?, role = ?, poste = ? WHERE id = ?";
     try (PreparedStatement stmt = conn.prepareStatement(query)) {

          stmt.setString(1, E.getNom());
          stmt.setString(2, E.getPrenom());
          stmt.setString(3, E.getEmail());
          stmt.setString(4, E.getTelephone());
          stmt.setDouble(5, E.getSalaire());
          stmt.setString(6, E.getRole().name());
          stmt.setString(7, E.getPoste().name());
          stmt.setInt(8, id);
          stmt.executeUpdate();

          System.out.println("Employe modifier avec succ s !");
     } catch (SQLException e) {
          System.out.println("Erreur lors de la modefication de l'employe !!!!!");

     }
}

// Supprimer un employ  de la base de donn es par son ID
@Override
public void delete(int id) {
     String query = "DELETE FROM Employee WHERE id = ?";
     try (PreparedStatement stmt = conn.prepareStatement(query)) {
          stmt.setInt(1, id);
```

```
103             stmt.executeUpdate();
104             System.out.println("Employe supprim  avec succ s !");
105         } catch (SQLException e) {
106             System.out.println("Erreur lors de la suppression de l'employe !!!!
        ");
107 
108         }
109 }
110 
111 
112 
113 }
```

2.3 Controller

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

EmployeController

```
1  package controller;
2  
3  
4  
5  import java.util.List;
6  
7  import javax.swing.JOptionPane;
8  import javax.swing.table.DefaultTableModel;
9  
10 import DAO.EmployeImpl;
11 import Model.Employe;
12 import Model.Employe.Poste;
13 import Model.Employe.Role;
14 
15 import Model.EmployeModel;
16 import View.EmployeView;
17 
18 public class EmployeController {
19 private EmployeModel model;
20 private EmployeView view;
21 
22 public EmployeController(EmployeModel model,EmployeView view) {
23   this.model=model;
24   this.view=view;
25 
26   this.view.btnAjouter.addActionListener(e->addEmploye());
27   this.view.btnModifier.addActionListener(e->updateEmploye());
```

```
28    this.view.btnAfficher.addActionListener(e -> afficherEmploye());
29    this.view.btnSupprimer.addActionListener(e -> supprimerEmploye());
30
31
32
33  }
34  //M thode pour ajouter un employ
35  private void addEmploye() {
36    String nom=view.getNom();
37    String prenom=view.getPrenom();
38    String email=view.getEmail();
39    String telephone=view.getTelephone();
40    double salaire =view.getSalaire();
41    Poste poste=view.getPoste();
42    Role role=view.getRole();
43
44
45    boolean addEmploye=model.addEmploye(nom, prenom, email, telephone,salaire
       , role, poste);
46      afficherEmploye();
47
48    if(addEmploye) JOptionPane.showMessageDialog(null, "Employ  Ajout  avec
        succ s !");
49    else JOptionPane.showMessageDialog(null, "Echec d'ajout d'employe !!!!!")
       ;
50  }
51
52  //M thode pour modifier un employ
53  private void updateEmploye() {
54    int selectedRow = view.table.getSelectedRow();
55
56      int id = (int) view.table.getValueAt(selectedRow, 0);
57
58    String nom=view.getNom();
59    String prenom=view.getPrenom();
60    String email=view.getEmail();
61    String telephone=view.getTelephone();
62    double salaire =view.getSalaire();
63    Poste poste=view.getPoste();
64    Role role=view.getRole();
65
66      Employe employe = new Employe(nom, prenom, email, telephone, salaire,
       role, poste);
67        EmployeImpl employeImpl = new EmployeImpl();
68
69        employeImpl.update(employe,id);
70          afficherEmploye();
71
72        JOptionPane.showMessageDialog(null, "Employ  modifi  avec succ s !
```

```
    ");
  }

// M thode pour afficher tous les employ s dans la table
public void afficherEmploye() {
 EmployeImpl employeImpl = new EmployeImpl();
      List<Employe> employes = employeImpl.findAll();

     DefaultTableModel model = (DefaultTableModel) view.table.getModel();
      model.setRowCount(0);

      for (Employe employe : employes) {
          model.addRow(new Object[]{
            employe.getId(),
              employe.getNom(),
              employe.getPrenom(),
              employe.getEmail(),
              employe.getTelephone(),
              employe.getSalaire(),
              employe.getRole(),
              employe.getPoste()
          });
      }
  }

//M thode pour supprimer un employ
public void supprimerEmploye() {
int selectedRow = view.table.getSelectedRow();
    if (selectedRow == -1) {

        JOptionPane.showMessageDialog(null, "Veuillez s lectionner un
    employ     supprimer !");

    }
    int id =view.getId(view.table);
    EmployeImpl employeImpl = new EmployeImpl();


        employeImpl.delete(id);
        afficherEmploye();
        JOptionPane.showMessageDialog(null, "Employ  supprim  avec
    succ s !");


        }
}
```

2.4 MAIN

Main

```
1  package DAO;
2
3  import java.util.List;
4
5  import Model.Employe;
6  import Model.Employe.Poste;
7  import Model.Employe.Role;
8
9  public interface EmployeI {
10
11 List<Employe> findAll();
12 void add(Employe E);
13 void update(Employe E,int id);
14 void delete(int id);
15
16
17
18 }
```

Résultats

1 Résultats de la partie View

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

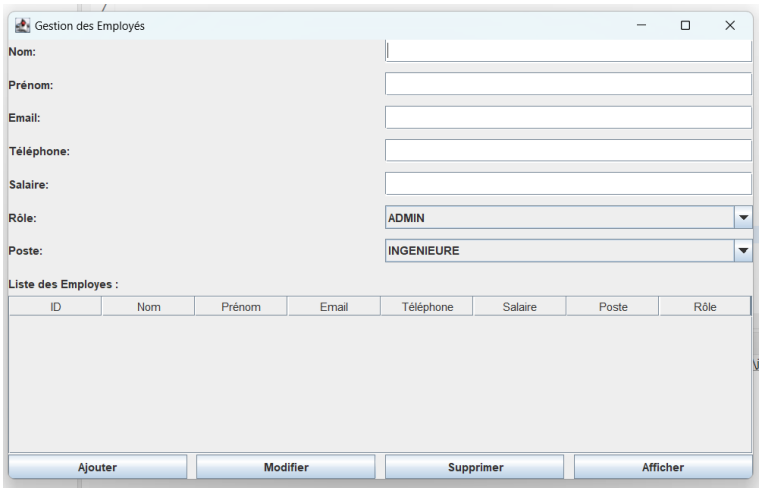
- 

FIGURE 2.1 – Interface Utilisateur

2 Affichage

Au début du travail pratique, une liste des employés est affichée avec leurs informations principales. L'utilisateur peut parcourir cette liste et sélectionner un employé. Un bouton "Afficher" permet de visualiser les détails de l'employé sélectionné.

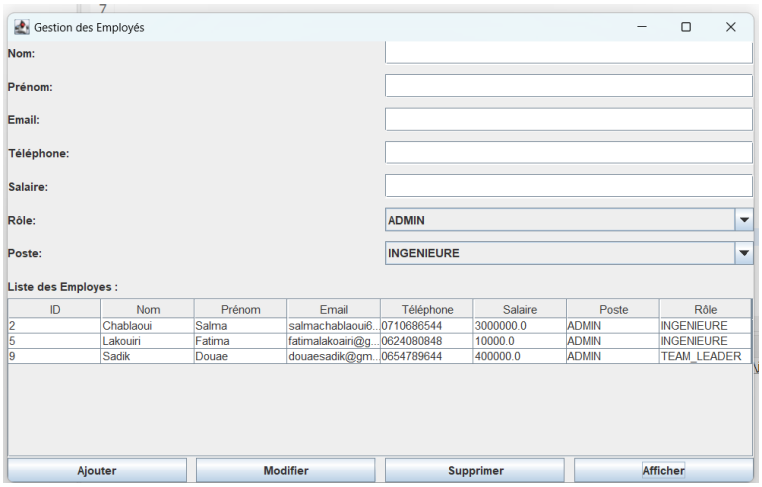


FIGURE 2.2 – Resultat d'afficher

3 Après Ajout

Après l'ajout d'un employé, les informations saisies par l'utilisateur dans le panneau d'entrée sont validées, notamment en vérifiant que le salaire est supérieur à 0 et que l'email est valide. Ces données sont ensuite transmises à la couche Controller, qui interagit avec la logique métier pour enregistrer les informations. Une fois l'opération réussie, la liste des employés est automatiquement mise à jour dans le panneau d'affichage, reflétant les changements en temps réel. Si une des validations échoue, un message d'erreur approprié est affiché à l'utilisateur pour l'informer des corrections nécessaires.

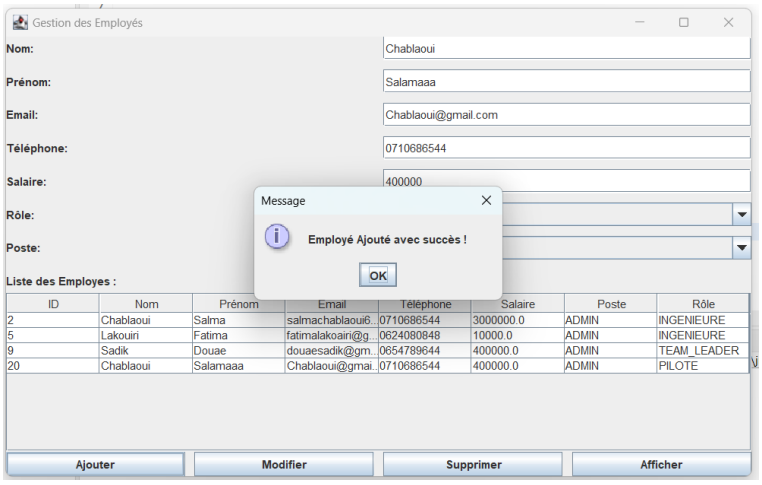


FIGURE 2.3 – Affichage de l'Ajout

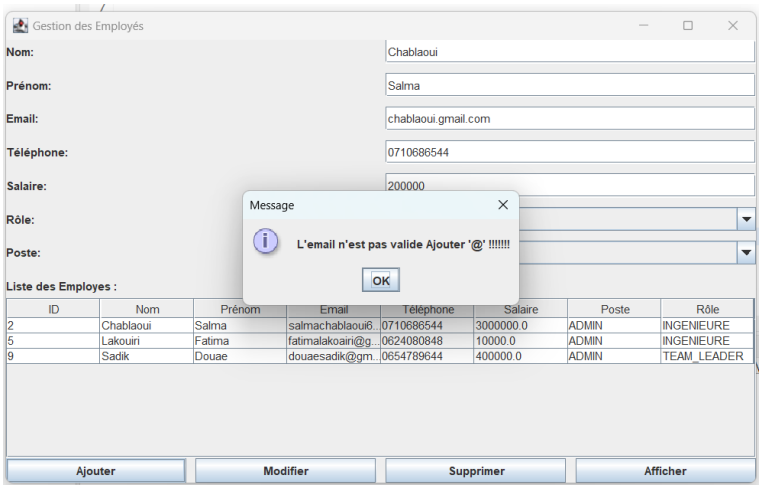


FIGURE 2.4 – Résultat en cas d'email invalide

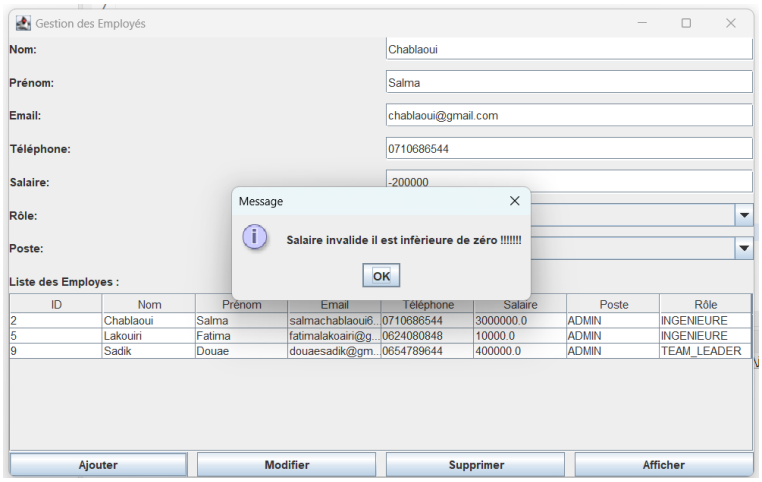


FIGURE 2.5 – Résultat en cas du salaire invalide

4 Après modification

Après la mise à jour d'un employé, les nouvelles informations saisies par l'utilisateur dans le panneau d'entrée sont validées et transmises à la couche Controller, qui assure leur traitement via la logique métier.

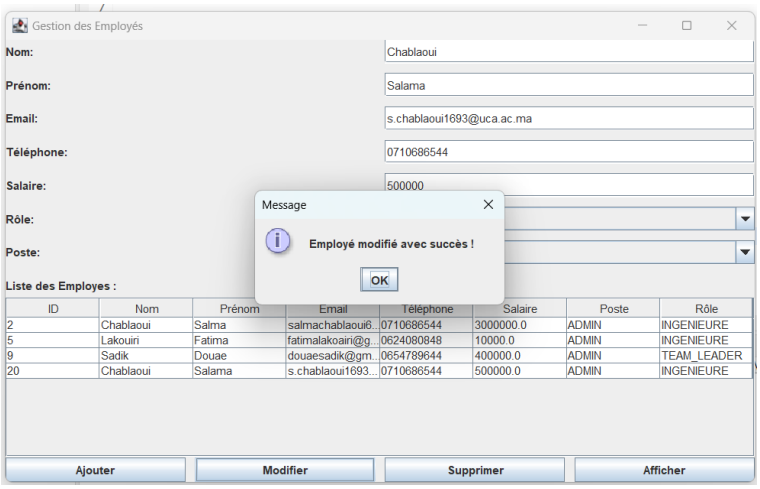


FIGURE 2.6 – Resultat de modification

5 Apres Suppression

Lorsqu'un employé est supprimé, l'utilisateur sélectionne l'employé concerné dans la liste affichée et confirme l'action en cliquant sur le bouton Supprimer. Cette demande est transmise à la couche Controller, qui s'assure de la suppression de l'enregistrement via la logique métier.

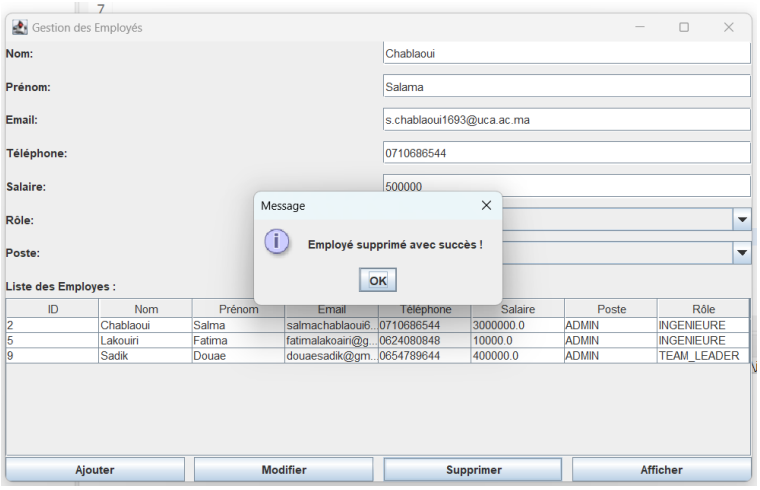


FIGURE 2.7 – Resultat de suppression

Tous ca est stocké dans la base de donnée voici le resultat.

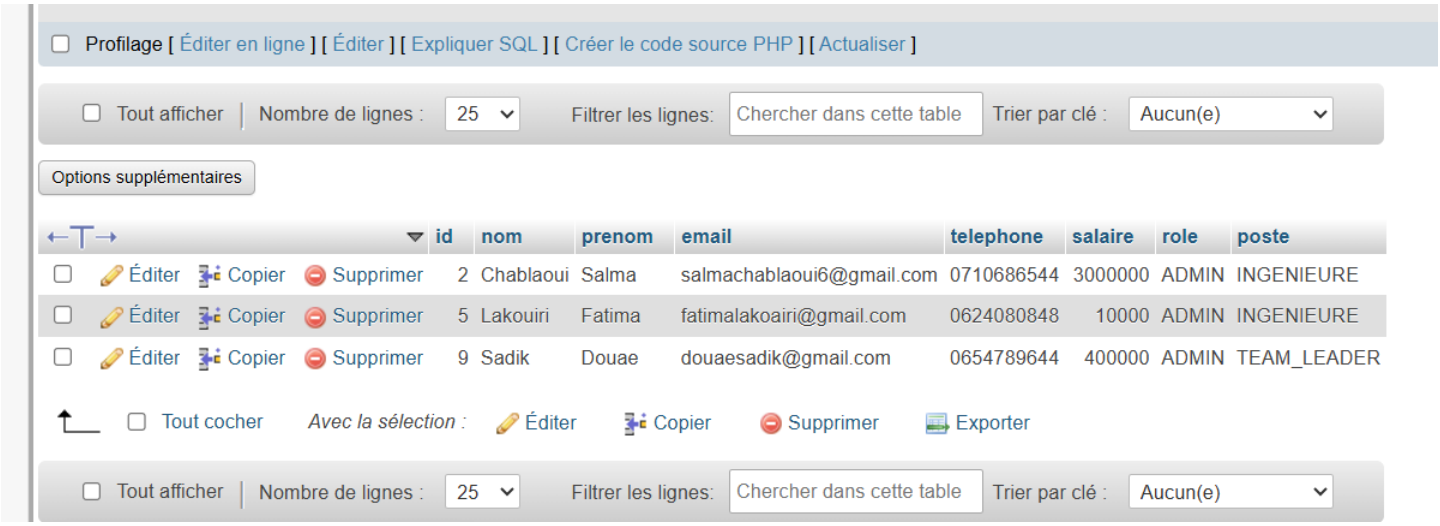


FIGURE 2.8 – affichage de tous ce qui est stocké à la base de donnée

Conclusion générale

En conclusion, ce travail pratique a permis de créer une application Java fonctionnelle pour la gestion des employés, en appliquant les principes de la programmation orientée objet et l'architecture MVC. Grâce à cette architecture, l'application est structurée de manière claire, assurant une séparation des responsabilités qui facilite la maintenance et l'évolution du code. Les fonctionnalités principales, telles que l'ajout, la modification, la suppression et l'affichage des informations des employés, ont été intégrées dans une interface fluide et interactive, offrant une expérience utilisateur optimale.

Ce projet a non seulement permis de renforcer la maîtrise des concepts techniques, mais a également mis en évidence l'importance de la structuration du code pour garantir des applications évolutives et durables. En intégrant ces principes dans la gestion d'un projet concret, ce TP constitue une étape clé dans l'acquisition des compétences nécessaires pour aborder des projets logiciels plus complexes à l'avenir, tout en développant des aptitudes en gestion de projet, conception logicielle et interaction utilisateur.

Références

java :

— https://www.java.com/en/download/

Eclipse :

— https://www.eclipse.org/downloads/

XAMPP :

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

jdk 23 :

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