

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

Rapport du TP N°1 java avancée

Gestion des employés

Réalisé par : EL HINA Soukaina

Encadré par : Mme. Leila Elkhrof

ANNÉE UNIVERSITAIRE : 2024/2025

Table des matières

Introduction	4
Outils & environnement de travail	5
1 Environnement de travail	5
2 Outils de travail	5
3 Language de Programmation	6
1 Réalisation	7
1 Création de la base de donnée	7
1.1 Script base de donnée	7
2 Architecture MVC (Model-View-Controller)	8
2.1 Model	8
2.2 DAO	11
2.3 Controller	15
2.4 View	17
2 Résultats	21
1 Tables Créées	21
2 Résultats de la partie View	21
3 Après Ajout	22
4 Après modification	22
5 Apres Suppression	23
3 Conclusion générale	25
4 Références	25

Table des figures

1	Eclipse logo	5
2	MySQL Workbench logo	5
3	xampp logo	6
4	java developpement kit logo	6
5	java logo	6
2.1	Tables de la base de donnée	21
2.2	Interface Utilisateur	22
2.3	Resultat Ajout	22
2.4	Resultat de modification	23
2.5	Affichage de modification	23
2.6	Resultat de suppression	24

Introduction

Ce travail pratique (TP) se concentre sur le développement d'une application Java dédiée à la gestion des employés, en adoptant une structure basée sur l'architecture **MVC (Model-View-Controller)**. Ce projet s'inscrit dans le cadre de l'apprentissage des concepts fondamentaux de la programmation orientée objet (POO) et de la mise en œuvre d'interfaces graphiques avec la bibliothèque **Swing**. Il offre également une opportunité d'approfondir les compétences en conception logicielle et en organisation structurée du code pour garantir une séparation claire des responsabilités.

L'objectif principal est de développer une application intuitive et fonctionnelle permettant de manipuler des données d'employés. L'application est conçue pour gérer l'ajout, la modification, la suppression et l'affichage des informations des employés, tout en assurant une interface utilisateur fluide et interactive. Grâce à l'implémentation des principes de l'architecture MVC, ce projet garantit une maintenance simplifiée et une évolutivité de l'application.

Les fonctionnalités principales incluent :

- Ajout d'employés avec des informations complètes.
- Modification des données des employés.
- Suppression des employés.
- Affichage de la liste des employés.

Cette application vise non seulement à développer une solution technique pour la gestion des employés, mais aussi à démontrer la puissance des concepts de la programmation orientée objet combinée à une architecture bien définie. Elle constitue une étape fondamentale dans l'acquisition des compétences nécessaires pour des projets logiciels plus complexes à l'avenir.

Outils & environnement de travail

1 Environnement de travail



FIGURE 1 – Eclipse logo

- **Eclipse** : Eclipse est un environnement de développement intégré (IDE) open-source, principalement utilisé pour le développement en Java, mais extensible à d'autres langages grâce à des plugins. Il offre des outils pour écrire, déboguer et gérer du code efficacement, avec une interface modulable et multiplateforme. Très polyvalent, Eclipse est prisé pour le développement d'applications web, mobiles et 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

```
1 -- Cr ation de la base de donn es
2 CREATE DATABASE Employe;
3
4 -- Utilisation de la base de donn es
5 USE Employe;
6
7 -- Cr ation de la table des employ s
8 CREATE TABLE `employes` (
9   `id` int(11) NOT NULL,
10  `nom` varchar(50) NOT NULL,
11  `prenom` varchar(50) NOT NULL,
12  `email` varchar(100) NOT NULL,
13  `telephone` varchar(15) NOT NULL,
14  `salaire` double(10,2) NOT NULL,
15  `role` varchar(50) NOT NULL,
16  `poste` varchar(50) NOT NULL
17 )
18 );
```

Listing 1.1 – Script SQL de la base de données

- Ce script est écrit sur MySQL Workbench pour création la base de donnée pour être lié 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

```
1 package model;
2 public class Employe {
3     private int id;
4     public int getId() {
5         return id;
6     }
7
8
9
10
11     public void setId(int id) {
12         this.id = id;
13     }
14     public Employe(int id, String nom, String prenom, String email, String telephone
15         , double salaire, Role role,
16         Poste poste) {
17         super();
18         this.id = id;
19         this.nom = nom;
20         this.prenom = prenom;
21         this.email = email;
22         this.telephone = telephone;
23         this.salaire = salaire;
24         this.role = role;
25         this.poste = poste;
26     }
27     private String nom;
28     private String prenom;
29     private String email;
30     private String telephone;
31     private double salaire;
32     private Role role;
33     private Poste poste;
34
35     public Employe(String nom, String prenom, String email,
36         String telephone, double salaire, Role role, Poste poste) {
37         this.nom=nom;
38         this.prenom=prenom;
39         this.email=email;
40         this.telephone=telephone;
41         this.salaire=salaire;
```



```
41     this.role=role;
42     this.poste=poste;
43 }
44
45
46
47
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 public enum Role {
105     ADMIN,
106     EMPLOYE
107 }
108 public enum Poste {
109     INGENIEURE,
110     TEAM_LEADER,
111     PILOTE
112 }
113
114 }
```

EmployerLogic

```
1 package model;
2
3 import javax.management.relation.Role;
4
5 import DAO.EmployeImpl;
6 import model.Employe.Poste;
7
8 public class Employemodel {
9     private EmployeImpl dao;
10
11     public Employemodel(EmployeImpl dao) {
12         this.dao=dao;
13     }
14
15     //logique Metier
16     public boolean addEmploye(String nom,String pronom,String email,String telephone,double
17         salaire,model.Employe.Role role,Poste poste) {
18         if(salaire<=0) {
19             System.out.println("Le salaire doit etre superieur de 0 !!!!!");
20             return false;}
21
22         if (email == null || !email.contains("@")) {
23             System.out.println("L'email n'est pas valide !");
24             return false;
25         }
26
27         Employe NvEmploye = new Employe(nom,pronom,email,telephone,salaire,role,poste);
28         dao.add(NvEmploye);
29     return true;
30 }
```

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.

DBConnection

```
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     public static final String url = "jdbc:mysql://localhost:3306/employe";
9     public static final String user = "root";
10    public static final String password = "";
11    private static Connection conn = null;
12
13    public static Connection getConnexion() {
14        if (conn == null) {
15            try {
16                conn = DriverManager.getConnection(url, user, password);
17                System.out.println("Connexion tablie avec succ s !");
18            } catch (SQLException e) {
19                System.out.println("Erreur de connexion !!!!!");
20                //e.printStackTrace();
21            }
22        }
23        return conn;
24    }
25
26    public static void closeConnexion() {
27        if (conn != null) {
28            try {
29                conn.close();
30                conn = null;
31                System.out.println("Connexion ferm e avec succ s !");
32            } catch (SQLException e) {
33                System.out.println("Erreur lors de la fermeture de la connexion
34                !!!!!");
35                //e.printStackTrace();
36            }
37        }
38    }
```

EmployerDAO

```
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 EmployeDAOI {
10
11 Employee findById(int employeeId);
12 List<Employe> findAll();
13 void add(Employe E);
14 void update(Employe E,int id);
15 void delete(int id);
16 List<Role>findAllRoles();
17 List<Poste>findAllPostes();
18 void updateSpecificField(int id, String fieldName, Object newValue);
19
20
21 }
```

EmployerInterface

```
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 EmployeeImpl implements EmployeDAOI {
19     private Connection conn;
20
21     public EmployeeImpl() {
22         this.conn = Connexion.getConnexion();
23     }
24
25     @Override
26     public void add(Employe E) {
27         String Query = "INSERT INTO employes(nom, prenom, email, telephone,
28 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         }
```

```
36         stmt.setString(7, E.getPoste().name());
37         stmt.executeUpdate();
38         System.out.println("Employ ajout avec succ s !");
39     } catch (SQLException e) {
40         System.out.println("Erreur lors de l'ajout de l'employ !");
41         //e.printStackTrace();
42
43     }
44 }
45 @Override
46 public Employee findById(int employeeId) {
47     String query = "SELECT * FROM employes WHERE id = ?";
48     try (PreparedStatement stmt = conn.prepareStatement(query)) {
49         stmt.setInt(1, employeeId);
50         ResultSet rs = stmt.executeQuery();
51         if (rs.next()) {
52             return new Employee(
53                 rs.getString("nom"),
54                 rs.getString("prenom"),
55                 rs.getString("email"),
56                 rs.getString("telephone"),
57                 rs.getDouble("salaire"),
58                 Employee.Role.valueOf(rs.getString("role")),
59                 Poste.valueOf(rs.getString("poste"))
60             );
61         }
62     } catch (SQLException e) {
63         System.out.println("Erreur lors de la recherche de l'employ par ID
64         !!!!");
65         //e.printStackTrace();
66     }
67     return null;
68
69
70 @Override
71 public List<Employee> findAll() {
72     List<Employee> employes = new ArrayList<>();
73     String query = "SELECT * FROM employes";
74     try (Statement stmt = conn.createStatement();
75         ResultSet rs = stmt.executeQuery(query)) {
76         while (rs.next()) {
77             employes.add(new Employee(
78                 rs.getInt("id"),
79                 rs.getString("nom"),
80                 rs.getString("prenom"),
81                 rs.getString("email"),
82                 rs.getString("telephone"),
83                 rs.getDouble("salaire"),
84                 Employee.Role.valueOf(rs.getString("role")),
85                 Poste.valueOf(rs.getString("poste"))
86             ));
87         }
88     } catch (SQLException e) {
89         System.out.println("Erreur lors de la r cup ration de tous les
```

```

    employ s !!!!!");
90         //e.printStackTrace();
91     }
92     return employes;
93 }
94
95
96 @Override
97 public void update(Employe E, int id) {
98     String query = "UPDATE employes SET nom = ?, prenom = ?, email = ?, telephone
= ?, salaire = ?, role = ?, poste = ? WHERE id = ?";
99     try (PreparedStatement stmt = conn.prepareStatement(query)) {
100         stmt.setString(1, E.getNom());
101         stmt.setString(2, E.getPrenom());
102         stmt.setString(3, E.getEmail());
103         stmt.setString(4, E.getTelephone());
104         stmt.setDouble(5, E.getSalaire());
105         stmt.setString(6, E.getRole().name());
106         stmt.setString(7, E.getPoste().name());
107         stmt.setInt(8, id);
108         stmt.executeUpdate();
109         System.out.println("Employe modifier avec succ s !");
110     } catch (SQLException e) {
111         System.out.println("Erreur lors de la modification de l'employe !!!!!");
112         //e.printStackTrace();
113     }
114 }
115
116 @Override
117 public void delete(int id) {
118     String query = "DELETE FROM employes WHERE id = ?";
119     try (PreparedStatement stmt = conn.prepareStatement(query)) {
120         stmt.setInt(1, id);
121         stmt.executeUpdate();
122         System.out.println("Employe supprim avec succ s !");
123     } catch (SQLException e) {
124         System.out.println("Erreur lors de la suppression de l'employe !!!!!");
125         //e.printStackTrace();
126     }
127 }
128
129 @Override
130 public List<Employe.Role> findAllRoles() {
131     return Arrays.asList(Employe.Role.values());
132 }
133
134 @Override
135 public List<Poste> findAllPostes() {
136     return Arrays.asList(Poste.values());
137 }
138
139 @Override
140 public void updateSpecificField(int id, String fieldName, Object newValue) {
141     // TODO Auto-generated method stub
142

```

```
143 }  
144  
145 }
```

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.

EmployerInterface

```
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 employerController {  
19     private Employemodel model;  
20     private Employeview view;  
21  
22     public employerController(Employemodel model, Employeview view) {  
23         this.model=model;  
24         this.view=view;  
25         this.view.btnAjouter.addActionListener(e->addEmploye());  
26         this.view.btnModifier.addActionListener(e->updateEmploye());  
27         this.view.btnAfficher.addActionListener(e -> afficherEmploye());  
28         this.view.btnSupprimer.addActionListener(e -> supprimerEmploye());  
29  
30  
31  
32     }  
33     private void addEmploye() {  
34         String nom=view.getNom();  
35         String prenom=view.getPrenom();  
36         String email=view.getEmail();  
37         String telephone=view.getTelephone();  
38         double salaire =view.getSalaire();  
39         Poste poste=view.getPoste();  
40         Role role=view.getRole();  
41
```

```
42
43
44 boolean addEmploye=model.addEmploye(nom, prenom, email, telephone,salaire, role,
    poste);
45 if(addEmploye) System.out.println("Employe ajoute avec Succes");
46 else System.out.println("Echec d'ajout d'employe !!!!!");
47 }
48
49
50
51
52
53
54 private void updateEmploye() {
55     int selectedRow = view.table.getSelectedRow();
56     //int id =view.getId(view.table);
57     int id = (int) view.table.getValueAt(selectedRow, 0);
58
59     String nom=view.getNom();
60     String prenom=view.getPrenom();
61     String email=view.getEmail();
62     String telephone=view.getTelephone();
63     double salaire =view.getSalaire();
64     Poste poste=view.getPoste();
65     Role role=view.getRole();
66
67     Employee employe = new Employee(nom, prenom, email, telephone, salaire, role,
    poste);
68     EmployeeImpl employeImpl = new EmployeeImpl();
69
70     employeImpl.update(employe,id);
71     // JOptionPane.showMessageDialog(null, "Employ modifi avec succ s !");
72
73
74
75
76 }
77
78 public void afficherEmploye() {
79     EmployeeImpl EmployerDAOimpl = new EmployeeImpl();
80     List<Employee> employees = EmployerDAOimpl.findAll();
81
82     DefaultTableModel model = (DefaultTableModel) view.table.getModel();
83     model.setRowCount(0);
84
85     for (Employee employe : employees) {
86         model.addRow(new Object[]{
87             employe.getId(),
88             employe.getNom(),
89             employe.getPrenom(),
90             employe.getEmail(),
91             employe.getTelephone(),
92             employe.getSalaire(),
93             employe.getRole(),
94             employe.getPoste()
```



```
95         });
96     }
97 }
98 public void supprimerEmploye() {
99     int selectedRow = view.table.getSelectedRow();
100     if (selectedRow == -1) {
101         JOptionPane.showMessageDialog(null, "Veuillez selectionner un employe
supprimer !");
102         return;
103     }
104     int id = view.getId(view.table);
105     //int id = (int) view.table.getValueAt(selectedRow, 0);
106     EmployeeImpl EmployerDAOimpl = new EmployeeImpl();
107
108     int confirmation = JOptionPane.showConfirmDialog(null, "Voulez-vous vraiment
supprimer cet employe ?", "Confirmation", JOptionPane.YES_NO_OPTION);
109     if (confirmation == JOptionPane.YES_OPTION) {
110         EmployerDAOimpl.delete(id);
111         JOptionPane.showMessageDialog(null, "Employe supprime avec succ s !");
112     }
113 }
114 }
115 }
```

2.4 View

Une View en Java désigne une partie visible de l'application, utilisée pour afficher des informations et interagir avec l'utilisateur, comme un bouton ou un champ de texte.

Employeview

```
1 package view;
2
3 import javax.swing.*;
4 import javax.swing.table.DefaultTableModel;
5 import model.Employe.Role;
6 import model.Employe.Poste;
7
8 import java.awt.*;
9 import java.util.ArrayList;
10
11 public class Employeview extends JFrame {
12
13     private JPanel mainPanel, topPanel, centerPanel, bottomPanel;
14     private JLabel lblNom, lblPrenom, lblEmail, lblTelephone, lblSalaire, lblPoste
, lblRole;
15     private JTextField txtNom, txtPrenom, txtEmail, txtTelephone, txtSalaire;
16     private JComboBox<Poste> cbPoste;
17     private JComboBox<Role> cbRole;
18     public JTable table;
19     public JButton btnAjouter;
20     public JButton btnModifier;
21     public JButton btnSupprimer;
```

```
22 public JButton btnAfficher;
23
24 public Employeview() {
25
26     setTitle("Gestion des Employés");
27     setSize(600, 400);
28     setLocationRelativeTo(null);
29     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30     setLayout(new BorderLayout());
31
32     mainPanel = new JPanel(new BorderLayout());
33     topPanel = new JPanel(new GridLayout(7, 2, 10, 10));
34     centerPanel = new JPanel(new BorderLayout());
35     bottomPanel = new JPanel(new GridLayout(1, 4, 10, 10));
36
37     lblNom = new JLabel("Nom:");
38     lblPrenom = new JLabel("Prénom:");
39     lblEmail = new JLabel("Email:");
40     lblTelephone = new JLabel("Téléphone:");
41     lblSalaire = new JLabel("Salaire:");
42     lblPoste = new JLabel("Poste:");
43     lblRole = new JLabel("Rôle:");
44
45     txtNom = new JTextField();
46     txtPrenom = new JTextField();
47     txtEmail = new JTextField();
48     txtTelephone = new JTextField();
49     txtSalaire = new JTextField();
50
51     cbRole = new JComboBox<>(Role.values());
52     cbPoste = new JComboBox<>(Poste.values());
53
54
55
56     table = new JTable(new DefaultTableModel(new Object[]{"ID", "Nom", "Prénom", "Email", "Téléphone", "Salaire", "Poste", "Rôle"}, 0));
57     JScrollPane scrollPane = new JScrollPane(table);
58
59
60     btnAjouter = new JButton("Ajouter");
61     btnModifier = new JButton("Modifier");
62     btnSupprimer = new JButton("Supprimer");
63     btnAfficher = new JButton("Afficher");
64
65     topPanel.add(lblNom);
66     topPanel.add(txtNom);
67     topPanel.add(lblPrenom);
68     topPanel.add(txtPrenom);
69     topPanel.add(lblEmail);
70     topPanel.add(txtEmail);
71     topPanel.add(lblTelephone);
72     topPanel.add(txtTelephone);
73     topPanel.add(lblSalaire);
74     topPanel.add(txtSalaire);
75     topPanel.add(lblRole);
```

```

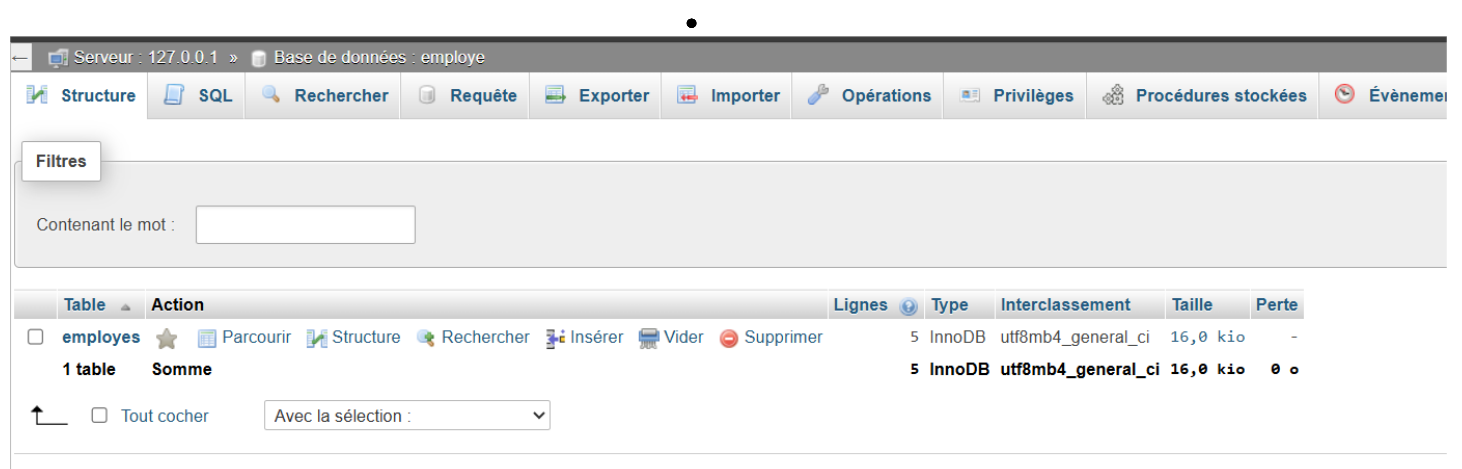
76     topPanel.add(cbRole);
77     topPanel.add(lblPoste);
78     topPanel.add(cbPoste);
79
80     centerPanel.add(scrollPane, BorderLayout.CENTER);
81
82     bottomPanel.add(btnAjouter);
83     bottomPanel.add(btnModifier);
84     bottomPanel.add(btnSupprimer);
85     bottomPanel.add(btnAfficher);
86
87     mainPanel.add(topPanel, BorderLayout.NORTH);
88     mainPanel.add(centerPanel, BorderLayout.CENTER);
89     mainPanel.add(bottomPanel, BorderLayout.SOUTH);
90
91     add(mainPanel);
92     setVisible(true);
93 }
94
95
96 public int getId(JTable table) {
97     int selectedRow = table.getSelectedRow();
98
99     if (selectedRow == -1) {
100         JOptionPane.showMessageDialog(null, "Veuillez selectionner une ligne
!");
101         return -1;
102     }
103     return (int) table.getValueAt(selectedRow, 0);
104
105 }
106
107     public String getNom() {
108         return txtNom.getText();
109     }
110     public String getPrenom() {
111         return txtPrenom.getText();
112     }
113     public String getEmail() {
114         return txtEmail.getText();
115     }
116     public String getTelephone() {
117         return txtTelephone.getText();
118     }
119     public double getSalaire() {
120         return Double.parseDouble(txtSalaire.getText());
121     }
122     public Role getRole() {
123         Role r=(Role) cbRole.getSelectedItem();
124         return r;
125     }
126     public Poste getPoste() {
127         Poste p=(Poste) cbPoste.getSelectedItem();
128         return p ;
129     }

```

```
130  
131     public static void main(String[] args) {  
132         new Employeeview();  
133     }  
134 }
```

Résultats

1 Tables Créées



The screenshot shows a database management interface. At the top, there's a toolbar with buttons for 'Structure', 'SQL', 'Rechercher', 'Requête', 'Exporter', 'Importer', 'Opérations', 'Privilèges', 'Procédures stockées', and 'Événement'. Below the toolbar is a 'Filtres' section with a text input field labeled 'Contenant le mot :'. The main area displays a table list with columns: 'Table', 'Action', 'Lignes', 'Type', 'Interclassement', 'Taille', and 'Perte'. The table list shows one table named 'employes' with 5 lines, InnoDB type, utf8mb4_general_ci collation, and a size of 16,0 kio. Below the table list, there are options to 'Tout cocher' and a dropdown menu labeled 'Avec la sélection :'. The 'employes' table is selected, and its details are shown in the table below.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> employes	Parcourir Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8mb4_general_ci	16,0 kio	-
1 table	Somme	5	InnoDB	utf8mb4_general_ci	16,0 kio	0 o

☐ Tout cocher Avec la sélection :

FIGURE 2.1 – Tables de la base de donnée

2 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.

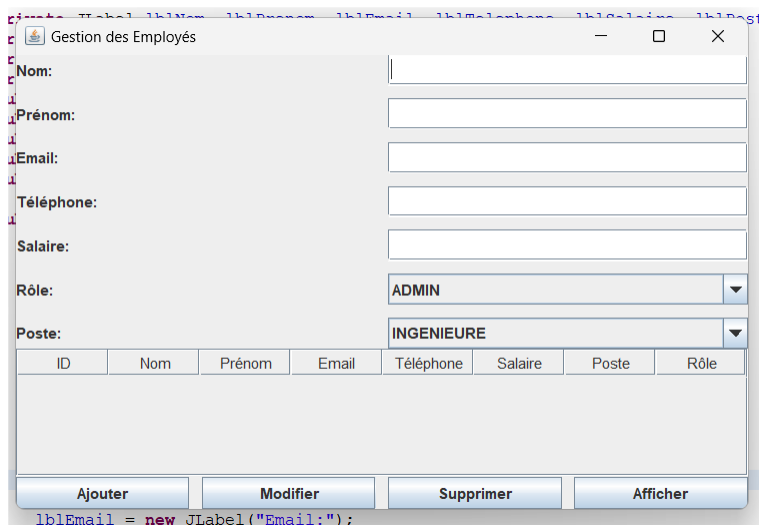


FIGURE 2.2 – Interface Utilisateur

3 Après Ajout

Après l'ajout d'un employé, les informations saisies par l'utilisateur dans le panneau d'entrée sont validées et transmises à la couche Controller, qui communique avec la logique métier pour enregistrer les données. 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.

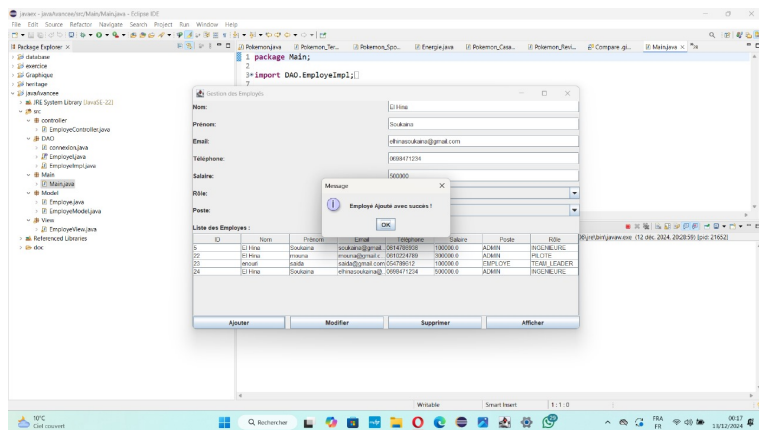


FIGURE 2.3 – Resultat Ajout

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.

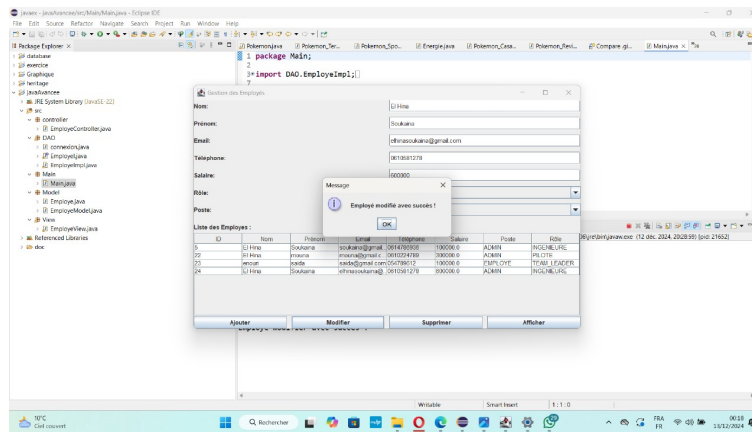


FIGURE 2.4 – Resultat de modification

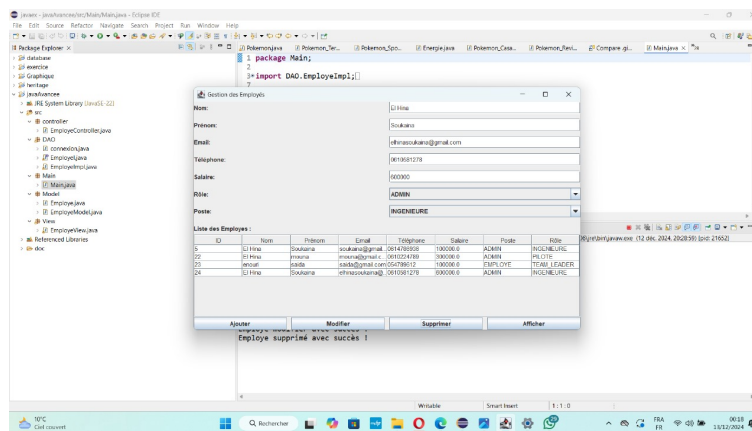


FIGURE 2.5 – Affichage de modification

5 Après 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.

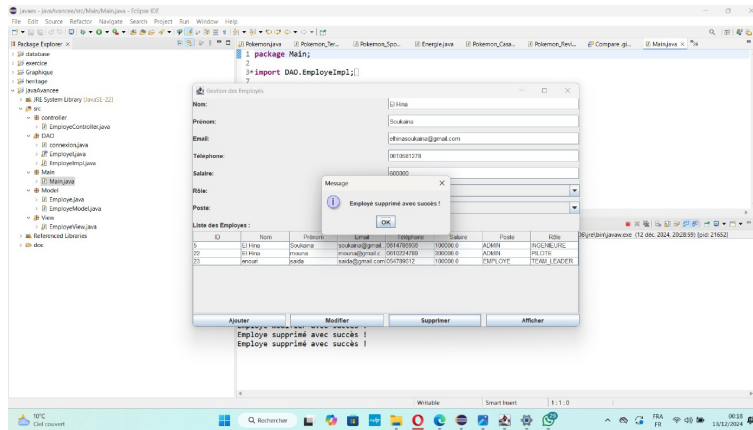


FIGURE 2.6 – Resultat de suppression

Conclusion générale

En conclusion, ce Tp a permis de mettre en œuvre une application de gestion des employés en utilisant l'architecture ****MVC****. Grâce à cette structure, nous avons séparé clairement les responsabilités entre la logique métier, l'interface utilisateur et le traitement des données, garantissant ainsi une application modulaire, maintenable et extensible. L'intégration de fonctionnalités telles que l'ajout, la mise à jour et la suppression d'employés a renforcé notre compréhension des concepts de programmation orientée objet et de gestion d'interfaces graphiques en Java. Ce travail pratique illustre l'importance de structurer et d'organiser le code pour développer des applications robustes et performantes.

Références

java :

— <https://www.java.com/en/download/>

Eclipse :

— <https://eclipse.fr.softonic.com/>

XAMPP :

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

jdk 23 :

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