

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

**Rapport du TP N°3 java avancée**

---

# **Gestion des congés (E/S)**

---

**Réalisé par : EL HINA Soukaina**

**Encadré par : Mme.Leila Elkhrof**

**ANNÉE UNIVERSITAIRE : 2024/2025**

# Table des matières

<b>Introduction</b>	<b>4</b>
<b>Outils &amp; environnement de travail</b>	<b>5</b>
1 Environnement de travail . . . . .	5
2 Outils de travail . . . . .	5
3 Language de Programmation . . . . .	6
<b>1 Réalisation</b>	<b>7</b>
1 Création de la base de donnée . . . . .	7
1.1 Script base de donnée . . . . .	7
2 Architecture MVC (Model-View-Controller) . . . . .	9
2.1 Model . . . . .	9
2.2 DAO . . . . .	13
2.3 Controller . . . . .	17
2.4 View . . . . .	21
<b>2 Résultats</b>	<b>29</b>
1 Tables Créées . . . . .	29
2 Résultats de la partie View . . . . .	29
3 Résultats du Bouton Importer . . . . .	30
4 Résultats du Bouton Exporter . . . . .	30
<b>3 Conclusion générale</b>	<b>32</b>
<b>4 Références</b>	<b>32</b>

# 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 . . . . .	29
2.2	Interface Utilisateur . . . . .	29
2.3	– Interface de l’importation du fichier . . . . .	30
2.4	Résultat de l’importation du fichier . . . . .	30
2.5	Interface de l’exportation du fichier . . . . .	31
2.6	Résultat de l’exportation du fichier . . . . .	31

# Introduction

Ce travail pratique (TP) porte sur la conception d'une application Java dédiée à la gestion des entrées et sorties des employés, en adoptant l'architecture MVC (Model-View-Controller). Ce modèle de conception, qui distingue clairement les données, la logique métier et l'interface utilisateur, vise à renforcer la maîtrise des concepts fondamentaux de la programmation orientée objet (POO) tout en introduisant le développement d'interfaces graphiques à l'aide de la bibliothèque Swing. Cette approche garantit une meilleure organisation du code grâce à une séparation claire des responsabilités, tout en facilitant la maintenance et l'ajout de nouvelles fonctionnalités.

L'application, conçue pour optimiser la gestion des données des employés, offre une interface intuitive et conviviale. Elle simplifie les processus d'importation et d'exportation de données via des fichiers externes, permettant ainsi une gestion fluide et efficace. L'utilisation de l'architecture MVC assure une structuration rigoureuse du code, rendant le système évolutif et facile à entretenir.

Les principales fonctionnalités de l'application incluent :

- Importation des informations des employés depuis des fichiers externes.
- Exportation des données des employés pour des besoins d'archivage ou de partage.
- Gestion des entrées et sorties des employés à travers une interface utilisateur dédiée.

Ce projet constitue une démonstration pratique de l'efficacité de la POO et de l'architecture MVC dans la création d'applications robustes et évolutives. Il représente une étape clé dans la formation, préparant les apprenants à concevoir des systèmes 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
2 CREATE DATABASE gestion;
3
4 USE gestion;
5
6
7 CREATE TABLE `employe` (
8   `id` int(11) NOT NULL,
9   `nom` varchar(100) NOT NULL,
10  `prenom` varchar(100) NOT NULL,
11  `email` varchar(150) NOT NULL,
12  `telephone` varchar(15) DEFAULT NULL,
13  `salaire` decimal(10,2) NOT NULL,
14  `role` enum('ADMIN', 'MANAGER', 'EMPLOYEE') NOT NULL,
15  `poste` enum('DEVELOPER', 'DESIGNER', 'MARKETING', 'OTHER') NOT NULL,
16  `solde` int(11) NOT NULL
17 ) ;
18
19
20
21 CREATE TABLE `holiday` (
22   `id` int(11) NOT NULL,
23   `id_employe` int(11) NOT NULL,
24   `startdate` date DEFAULT NULL,
25   `enddate` date DEFAULT NULL,
26   `type` enum('ANNUAL', 'SICK', 'MATERNITY', 'OTHER') NOT NULL
27 ) ;
28
29 ALTER TABLE `employe`
30   MODIFY `id` int(11) NOT NULL AUTO_INCREMENT ;
31
32
33 ALTER TABLE `holiday`
34   MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
35 CREATE TABLE `login` (
36   `id` int(11) NOT NULL,
37   `username` varchar(50) NOT NULL,
```

```
38 `password` varchar(255) NOT NULL
39 ) ;
40 ALTER TABLE `login`
41 ADD PRIMARY KEY (`id`),
42 ADD UNIQUE KEY `username` (`username`);
43 ALTER TABLE `login`
44 MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
45 INSERT INTO login ( username, password)
46 VALUES ( 'utilisateur1', SHA2('password123', 256));
47 INSERT INTO login ( username, password)
48 VALUES ( 'utilisateur1', 'password123');
```

Listing 1.1 – Script SQL de la base de données

- Ce script est écrit sur MySQL Workbench pour créer la base de données pour être liée 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.

#### Employe

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

```
41
42     public String getPrenom() {
43         return prenom;
44     }
45
46     public void setPrenom(String prenom) {
47         this.prenom = prenom;
48     }
49
50     public String getEmail() {
51         return email;
52     }
53
54     public void setEmail(String email) {
55         this.email = email;
56     }
57
58     public String getTelephone() {
59         return telephone;
60     }
61
62     public void setTelephone(String telephone) {
63         this.telephone = telephone;
64     }
65
66     public double getSalaire() {
67         return salaire;
68     }
69
70     public void setSalaire(double salaire) {
71         this.salaire = salaire;
72     }
73
74     public Role getRole() {
75         return role;
76     }
77
78     public void setRole(Role role) {
79         this.role = role;
80     }
81
82     public Post getPost() {
83         return poste;
84     }
85
86     public void setPost(Post post) {
87         this.poste = post;
88     }
89
90     public void setSolde (int conge){
91         this.solde = conge;
92     }
93
94     public int getSolde(){
95         return solde;
```

```
96     }
97 }
98 package model;
99
100 public enum Post {
101     DEVELOPER, DESIGNER, MARKETING, OTHER
102 }
103 package model;
104
105 public enum Role {
106     ADMIN, EMPLOYEE ,MANAGER
107 }
```

### Employemodel

```
1 package model;
2
3 import java.io.File;
4 import java.util.List;
5
6 import DAO.EmployeeDAOimpl;
7
8 public class Employemodel {
9     private EmployeeDAOimpl dao;
10    public Employemodel(EmployeeDAOimpl dao) {
11        this.dao = dao;
12    }
13
14    // funtion of add Employe :
15    public boolean addEmploye(int id ,String nom, String prenom, String email, String
    telephone, double salaire, Role role, Post post, int solde) {
16        if(salaire < 0 ){
17            System.out.println("Erreur : le salaire doit etre positif.");
18            return false;
19        }
20        if(id < 0 ){
21            System.out.println("Erreur : l'id doit etre positif.");
22            return false;
23        }
24        if(telephone.length() != 10){
25            System.out.println("Erreur : le telephone doit etre 10 num.");
26            return false;
27        }
28        if(!email.contains("@")){
29            System.out.println("Erreur : le mail doit contenir le @.");
30            return false;
31        }
32
33        Employe e = new Employe(id,nom, prenom, email, telephone, salaire, role, post ,
    solde);
34
35        dao.add(e);
36
37        return true;
38    }
39 }
```

```
40 // function of delete Employe :
41
42 public boolean deleteEmploye(int id){
43     dao.delete(id);
44     return true;
45 }
46
47 // function of update Employe :
48
49 public boolean updateEmploye(int id, String nom, String prenom, String email, String
    telephone, double salaire, Role role, Post post , int solde) {
50
51     Employe e = new Employe(id,nom, prenom, email, telephone, salaire, role, post,
    solde);
52     dao.update(e);
53     return true;
54 }
55
56 //function of update solde Employe :
57
58 public boolean updateSolde(int id, int solde) {
59     dao.updateSolde(id, solde);
60     return true;
61 }
62
63 //function of display Employe :
64
65 public List<Employe> displayEmploye() {
66     List<Employe> Employes = dao.display();
67     return Employes;
68 }
69 private boolean checkFileExists(File file) {
70
71     if(!file.exists()) {
72         throw new IllegalArgumentException ("le fichier n'existe pas "+file.getPath());
73     }
74     return true;
75 }
76
77 }
78 private boolean checkIsFile(File file) {
79
80     if(!file.isFile()) {
81         throw new IllegalArgumentException ("le chemin specifie nest pas un fichier "+file.
            getPath());
82     }
83     return true;
84 }
85
86 }
87 private boolean checkIsReadebal(File file) {
88
89     if(!file.canRead()) {
90         throw new IllegalArgumentException ("le chemin specifie nest pas lisibles "+file.
            getPath());
```

```
91  
92     }  
93     return true;  
94  
95 }  
96 }
```

## 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.*;  
4  
5 class DBConnexion {  
6     public static final String url = "jdbc:mysql://localhost:3306/gestion?useSSL=  
false&serverTimezone=UTC";  
7     public static final String user = "root";  
8     public static final String password = "";  
9     public static Connection conn = null;  
10  
11     public static Connection getConnexion() throws ClassNotFoundException {  
12         if (conn != null) {  
13             return conn;  
14         }  
15         try {  
16  
17             Class.forName("com.mysql.cj.jdbc.Driver");  
18             conn = DriverManager.getConnection(url, user, password);  
19             System.out.println("correct");  
20         } catch (SQLException e) {  
21             throw new RuntimeException("Error de connexion", e);  
22         }  
23  
24         return conn;  
25     }  
26 }
```

### GenericDAO

```
1 package DAO;  
2  
3 import java.util.List;  
4 public interface GenericDAOI <T> {  
5     public void add(T e);  
6     public void delete(int id);  
7     public void update(T e);  
8     public List<T> display();  
9 }
```

### EmployeInterface

```
1
2 package DAO;
3 import model.Employe;
4 import model.Post;
5 import model.Role;
6
7 import java.io.BufferedReader;
8 import java.io.BufferedWriter;
9 import java.io.FileReader;
10 import java.io.FileWriter;
11 import java.io.IOException;
12 import java.sql.Connection;
13 import java.sql.PreparedStatement;
14 import java.sql.ResultSet;
15 import java.sql.SQLException;
16 import java.util.ArrayList;
17 import java.util.List;
18
19
20 public class EmployeDAOimpl implements GenericDAOI<Employe> {
21
22     @Override
23     public void add(Employe e) {
24         // Do not include the 'id' column in the INSERT statement because it is
25         // AUTO_INCREMENT
26         String sql = "INSERT INTO employe (nom, prenom, email, telephone, salaire,
27             role, poste, solde) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
28         try (PreparedStatement stmt = DBConnexion.getConnection().prepareStatement(
29             sql)) {
30             stmt.setString(1, e.getNom());
31             stmt.setString(2, e.getPrenom());
32             stmt.setString(3, e.getEmail());
33             stmt.setString(4, e.getTelephone());
34             stmt.setDouble(5, e.getSalaire());
35             stmt.setString(6, e.getRole().name()); // Assuming Role is an enum
36             // and needs to be converted to string
37             stmt.setString(7, e.getPost().name()); // Assuming Post is an enum
38             // and needs to be converted to string
39             stmt.setInt(8, e.getSolde());
40             stmt.executeUpdate();
41         } catch (SQLException exception) {
42             System.err.println("Failed to add employee: " + exception.getMessage());
43             exception.printStackTrace(); // Prints the full stack trace for
44             // debugging
45         } catch (ClassNotFoundException ex) {
46             System.err.println("Failed to connect to the database: " + ex.
47                 getMessage());
48             ex.printStackTrace(); // Prints the full stack trace for debugging
49         }
50     }
51
52     @Override
```

```
46     public void delete(int id) {
47         String sql = "DELETE FROM employe WHERE id = ?";
48         try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement (
159         sql)) {
160             stmt.setInt(1,id);
161             stmt.executeUpdate();
162         } catch (SQLException exception) {
163             System.err.println("failed of delete employe");
164         } catch (ClassNotFoundException ex) {
165             System.err.println("failed connexion with data base");
166         }
167     }
168
169     @Override
170     public void update(Employe e) {
171         String sql = "UPDATE employe SET nom = ?, prenom = ?, email = ?, telephone
172         = ?, salaire = ?, role = ?, poste = ? WHERE id = ?";
173         try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement (
174         sql)) {
175             stmt.setString(1, e.getNom());
176             stmt.setString(2, e.getPrenom());
177             stmt.setString(3, e.getEmail());
178             stmt.setString(4, e.getTelephone());
179             stmt.setDouble(5, e.getSalaire());
180             stmt.setString(6, e.getRole().name());
181             stmt.setString(7, e.getPost().name());
182             stmt.setInt(8,e.getId());
183             stmt.executeUpdate();
184         } catch (SQLException exception) {
185             System.err.println("failed of update employe");
186         } catch (ClassNotFoundException ex) {
187             System.err.println("failed connexion with data base");
188         }
189     }
190
191     @Override
192     public List<Employe> display() {
193         String sql = "SELECT * FROM employe";
194         List<Employe> Employes = new ArrayList<>();
195         try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement (
196         sql)) {
197             ResultSet re = stmt.executeQuery();
198             while (re.next()) {
199                 int id = re.getInt("id");
200                 String nom = re.getString("nom");
201                 String prenom = re.getString("prenom");
202                 String email = re.getString("email");
203                 String telephone = re.getString("telephone");
204                 double salaire = re.getDouble("salaire");
205                 String role = re.getString("role");
206                 String poste = re.getString("poste");
207                 int solde = re.getInt("solde");
208                 Employe e = new Employe(id,nom, prenom, email, telephone, salaire,
209                 Role.valueOf(role), Post.valueOf(poste),solde);
210                 Employes.add(e);
211             }
212         }
213     }
214 }
```

```
96         return Employes;
97     } catch (ClassNotFoundException ex) {
98         System.err.println("failed connexion with data base");
99         return null;
100    } catch (SQLException ex) {
101        System.err.println("failed of display employee");
102        return null;
103    }
104 }
105
106
107 public void updateSolde(int id, int solde) {
108     String sql = "UPDATE employe SET solde = ? WHERE id = ?";
109     try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(
110 sql)) {
111         stmt.setInt(1, solde);
112         stmt.setInt(2, id);
113         stmt.executeUpdate();
114     } catch (SQLException exception) {
115         System.err.println("failed of update solde employee");
116     } catch (ClassNotFoundException ex) {
117         System.err.println("failed connexion with data base");
118     }
119 }
120 public void importData(String filePath) {
121     String query = "INSERT INTO Employe(nom, prenom, email, telephone, salaire
122 , role, poste) VALUES (?, ?, ?, ?, ?, ?, ?)";
123     Connection conn = null;
124     try (BufferedReader reader = new BufferedReader(new FileReader(filePath));
125         PreparedStatement ps = conn.prepareStatement(query)) {
126
127         String line = reader.readLine(); // Skip the header
128         while ((line = reader.readLine()) != null) {
129             String[] data = line.split(",");
130             if (data.length == 7) {
131                 ps.setString(1, data[0].trim()); // nom
132                 ps.setString(2, data[1].trim()); // prenom
133                 ps.setString(3, data[2].trim()); // email
134                 ps.setString(4, data[3].trim()); // telephone
135                 ps.setString(5, data[4].trim()); // salaire
136                 ps.setString(6, data[5].trim()); // role
137                 ps.setString(7, data[6].trim()); // poste
138                 ps.addBatch();
139             } else {
140                 System.err.println("Invalid line format: " + line);
141             }
142         }
143         ps.executeBatch();
144         System.out.println("Employe s import s avec succ s.");
145     } catch (IOException | SQLException e) {
146         e.printStackTrace();
147     }
148 }
149 public void exportData(String fileName, List<Employe> data) throws IOException
150 {
```



```
148     try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName)))
149     {
150         writer.write("nom, prenom, email, telephone, role, poste, salaire");
151         writer.newLine();
152         for (Employee employee : data) {
153             String line = String.format("%s,%s,%s,%s,%s,%s,%.2f",
154                 employee.getNom(),
155                 employee.getPrenom(),
156                 employee.getEmail(),
157                 employee.getTelephone(),
158                 employee.getRole(),
159                 employee.getPost(),
160                 employee.getSalaire());
161             writer.write(line);
162             writer.newLine();
163         }
164         System.out.println("Donn es  export es avec succ s.");
165     }
166 }
```

### 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.

#### EmployeInterface

```
1
2 package Controller;
3
4 import model.*;
5
6 import view.*;
7
8 import java.sql.Date;
9 import java.util.Calendar;
10 import java.util.List;
11
12 import javax.swing.table.DefaultTableModel;
13
14
15
16 public class EmployeController {
17
18     private final Employe_HolidayView View;
19     public static Employemodell model_employe ;
20     public static int id = 0;
21     public static int oldselectedrow = -1;
22     public static boolean test = false;
23     String nom = "";
24     String prenom = "";
```

```

25     String email = "";
26     String telephone = "";
27     double salaire = 0;
28     Role role = null;
29     Post poste = null;
30     int solde = 0;
31     boolean updatereussi = false;
32
33     public EmployeeController(Employee_HolidayView view, Employemodel model) {
34         this.View = view;
35         this.model_employe = model;
36
37         View.getaddButton_employe().addActionListener(e -> addEmployee());
38         View.getdeleteButton_employe().addActionListener(e -> deleteEmployee());
39         View.getupdateButton_employe().addActionListener(e -> updateEmployee());
40         View.getdisplayButton_employe().addActionListener(e -> displayEmployee());
41         Employee_HolidayView.Tableau.getSelectionModel().addListSelectionListener(e
-> updateEmployeebyselect());
42     }
43
44
45
46     public void displayEmployee() {
47         List<Employee> Employes = model_employe.displayEmployee();
48         if(Employes.isEmpty()){
49             View.afficherMessageErreur("Aucun employe.");
50         }
51         DefaultTableModel tableModel = (DefaultTableModel) Employee_HolidayView.
Tableau.getModel();
52         tableModel.setRowCount(0);
53         for(Employee e : Employes){
54             tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e
.getEmail(), e.getTelephone(), e.getSalaire(), e.getRole(), e.getPost(),e.
getSolde()});
55         }
56         View.remplaire_les_employes();
57     }
58
59
60     // function of add Employee :
61
62     private void addEmployee() {
63         String nom = View.getNom();
64         String prenom = View.getPrenom();
65         String email = View.getEmail();
66         String telephone = View.getTelephone();
67         double salaire = View.getSalaire();
68         Role role = View.getRole();
69         Post poste = View.getPoste();
70
71         View.viderChamps_em();
72         boolean addreussi = model_employe.addEmployee(0,nom, prenom, email,
telephone, salaire, role, poste ,25);
73
74         if(addreussi == true){

```

```

75         View.afficherMessageSucces("L'employe a bien ete ajoutee.");
76         displayEmploye();
77     }else{
78         View.afficherMessageErreur("L'employe n'a pas ete ajoutee.");
79     }
80 }
81
82
83
84 // function of delete Employe :
85
86 private void deleteEmploye(){
87     int selectedrow = Employee_HolidayView.Tableau.getSelectedRow();
88     if(selectedrow == -1){
89         View.afficherMessageErreur("Veuillez selectionner une ligne.");
90     }else{
91         int id = (int) Employee_HolidayView.Tableau.getValueAt(selectedrow, 0);
92         if(model_employe.deleteEmploye(id)){
93             View.afficherMessageSucces("L'employe a bien ete supprimer.");
94             displayEmploye();
95         }else{
96             View.afficherMessageErreur("L'employe n'a pas ete supprimer.");
97         }
98     }
99 }
100
101 // function of Update :
102
103 private void updateEmployebyselect(){
104     int selectedrow = Employee_HolidayView.Tableau.getSelectedRow();
105
106     if (selectedrow == -1) {
107         return;
108     }
109     try{
110         id = (int) Employee_HolidayView.Tableau.getValueAt(selectedrow, 0);
111         nom = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow, 1);
112         prenom = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow,
113 2);
114         email = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow,
115 3);
116         telephone = (String) Employee_HolidayView.Tableau.getValueAt (
117 selectedrow, 4);
118         salaire = (double) Employee_HolidayView.Tableau.getValueAt(selectedrow,
119 5);
120         role = (Role) Employee_HolidayView.Tableau.getValueAt(selectedrow, 6);
121         poste = (Post) Employee_HolidayView.Tableau.getValueAt(selectedrow, 7);
122         solde = (int) Employee_HolidayView.Tableau.getValueAt(selectedrow, 8);
123         View.remplaireChamps_em(id, nom, prenom, email, telephone, salaire,
124 role, poste);
125         test = true;
126     }catch(Exception e){
127         View.afficherMessageErreur("Erreur lors de la r cupration des
128 donnes ");
129     }

```

```
124     }
125
126     private void updateEmploye(){
127         if (!test) {
128             View.afficherMessageErreur("Veuillez d'abord s lectionner une ligne
modifier.");
129             return;
130         }
131         try {
132             nom = View.getNom();
133             prenom = View.getPrenom();
134             email = View.getEmail();
135             telephone = View.getTelephone();
136             salaire = View.getSalaire();
137             role = View.getRole();
138             poste = View.getPoste();
139
140             boolean updateSuccessful = model_employe.updateEmploye(id, nom, prenom
, email, telephone, salaire, role, poste , solde);
141
142             if (updateSuccessful) {
143                 test = false;
144                 View.afficherMessageSucces("L'employ a t modifi avec
succs.");
145                 displayEmploye();
146                 View.viderChamps_em();
147             } else {
148                 View.afficherMessageErreur("Erreur lors de la mise jour de l'
employ.");
149             }
150             } catch (Exception e) {
151
152                 View.afficherMessageErreur("Erreur lors de la mise jour");
153             }
154         }
155
156     public void resetSolde(){
157         Calendar now = Calendar.getInstance();
158         if(now.get(Calendar.DAY_OF_YEAR) == 1){
159             for (Employe employe : model_employe.displayEmploye()) {
160                 updateSolde(employe.getId(), 25);
161             }
162         }
163     }
164
165     public static void updateSolde(int id , int solde){
166         boolean updateSuccessful = model_employe.updateSolde(id, solde);
167     }
168
169 }
```

## 2.4 View

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

### **EmployeHolidayview**

```
1 package view;
2
3 import DAO.EmployeDAOimpl;
4 import model.Employe;
5 import model.Employemodel;
6 import model.Post;
7 import model.Role;
8 import model.Type_holiday;
9 import java.awt.*;
10 import java.io.BufferedReader;
11 import java.io.FileReader;
12 import java.io.FileWriter;
13 import java.io.PrintWriter;
14
15 import javax.swing.*;
16 import javax.swing.table.DefaultTableModel;
17 import java.util.List;
18
19 public class Employe_HolidayView extends JFrame {
20
21     private JTabbedPane tabbedPane = new JTabbedPane();
22
23     private JPanel employeTab = new JPanel();
24     private JPanel holidayTab = new JPanel();
25
26     private JPanel Employepan = new JPanel();
27     private JPanel Holidaypan = new JPanel();
28     private JPanel Display_Table_employe = new JPanel();
29     private JPanel Display_Table_holiday = new JPanel();
30     private final JPanel Forme_employe = new JPanel();
31     private final JPanel Forme_holiday = new JPanel();
32     private JPanel panButton_employe = new JPanel();
33     private JPanel panButton_holiday = new JPanel();
34
35     // les labels du l'employe
36     private JLabel label_nom = new JLabel("Nom");
37     private JLabel label_prenom = new JLabel("Prenom");
38     private JLabel label_email = new JLabel("Email");
39     private JLabel label_tele = new JLabel("Telephone");
40     private JLabel label_salaire = new JLabel("Salaire");
41     private JLabel label_role = new JLabel("Role");
42     private JLabel label_poste = new JLabel("Poste");
43
44     // les labels du c o n g
45     private JLabel label_employe = new JLabel("Nom de l'employ ");
46     private JLabel label_startDate = new JLabel("Date de debut (YYYY-MM-DD)");
47     private JLabel label_endDate = new JLabel("Date de fin (YYYY-MM-DD)");
48     private JLabel label_type = new JLabel("Type");
```

```
49     private JComboBox<Type_holiday> TypeComboBox = new JComboBox<>(Type_holiday.
values());
50
51     // les textfield du l'employe
52     private JTextField text_nom = new JTextField();
53     private JTextField text_prenom = new JTextField();
54     private JTextField text_email = new JTextField();
55     private JTextField text_tele = new JTextField();
56     private JTextField text_salaire = new JTextField();
57
58     private JComboBox<Role> roleComboBox = new JComboBox<>(Role.values());
59     private JComboBox<Post> posteComboBox = new JComboBox<>(Post.values());
60
61     // les textfield du cong
62     private JComboBox<String> text_employe = new JComboBox<>();
63     private JTextField text_startDate = new JTextField("");
64     private JTextField text_endDate = new JTextField("");
65
66     // les boutons du l'employe
67     private JButton addButton_employe = new JButton("Ajouter");
68     private JButton updateButton_employe = new JButton("Modifier");
69     private JButton deleteButton_employe = new JButton("Supprimer");
70     private JButton displayButton_employe = new JButton("Afficher");
71     public JButton importButton_employe = new JButton("Importer");
72     public JButton exportButton_employe = new JButton("Exporter");
73
74     // les boutons du cong
75     private JButton addButton_holiday = new JButton("Ajouter");
76     private JButton updateButton_holiday = new JButton("Modifier");
77     private JButton deleteButton_holiday = new JButton("Supprimer");
78     private JButton displayButton_holiday = new JButton("Afficher");
79     public JButton importButton_holiday = new JButton("Importer");
80     public JButton exportButton_holiday = new JButton("Exporter");
81
82
83
84     // le tableau de l'employe
85     JPanel pan0 = new JPanel(new BorderLayout());
86     public static String[] columnNames_employe = {"ID", "Nom", "Prenom", "Email",
" T lphone ", "Salaire", "Role", "Poste", "solde"};
87     public static DefaultTableModel tableModel = new DefaultTableModel(
columnNames_employe, 0);
88     public static JTable Tableau = new JTable(tableModel);
89
90     // le tableau du cong
91     JPanel pan1 = new JPanel(new BorderLayout());
92     public static String[] columnNames_holiday = {"ID", "nom_employe", "date_debut
", "date_fin", "type"};
93     public static DefaultTableModel tableModel1 = new DefaultTableModel(
columnNames_holiday, 0);
94     public static JTable Tableau1 = new JTable(tableModel1);
95
96     public Employe_HolidayView() {
97
98         setTitle("Gestion des employes et des cong s");
```

```

99      setSize(1000, 600);
100     setDefaultCloseOperation(EXIT_ON_CLOSE);
101     setLocationRelativeTo(null);
102
103     add(tabbedPane);
104
105     // Employe Tab
106     employeTab.setLayout(new BorderLayout());
107     employeTab.add(Employepan, BorderLayout.CENTER);
108
109     Employepan.setLayout(new BorderLayout());
110     Employepan.add(Display_Table_employe, BorderLayout.CENTER);
111     Tableau.setFillViewportHeight(true);
112     Dimension preferredSize = new Dimension(900, 500);
113     Tableau.setPreferredScrollableViewportSize(preferredSize);
114     pan0.add(new JScrollPane(Tableau), BorderLayout.CENTER);
115     Display_Table_employe.add(pan0);
116
117     Employepan.add(panButton_employe, BorderLayout.SOUTH);
118     panButton_employe.add(addButton_employe);
119     panButton_employe.add(updateButton_employe);
120     panButton_employe.add(deleteButton_employe);
121     panButton_employe.add(displayButton_employe);
122     panButton_employe.add(importButton_employe);
123     panButton_employe.add(exportButton_employe);
124
125
126     Employepan.add(Forme_employe, BorderLayout.NORTH);
127     Forme_employe.setLayout(new GridLayout(7, 2, 10, 10));
128     Forme_employe.add(label_nom);
129     Forme_employe.add(text_nom);
130     Forme_employe.add(label_prenom);
131     Forme_employe.add(text_prenom);
132     Forme_employe.add(label_email);
133     Forme_employe.add(text_email);
134     Forme_employe.add(label_tele);
135     Forme_employe.add(text_tele);
136     Forme_employe.add(label_salaire);
137     Forme_employe.add(text_salaire);
138     Forme_employe.add(label_role);
139     Forme_employe.add(roleComboBox);
140     Forme_employe.add(label_poste);
141     Forme_employe.add(posteComboBox);
142
143     // Holiday Tab
144     holidayTab.setLayout(new BorderLayout());
145     holidayTab.add(Holidaypan, BorderLayout.CENTER);
146     Holidaypan.setLayout(new BorderLayout());
147     Holidaypan.add(Display_Table_holiday, BorderLayout.CENTER);
148
149     Tableau1.setFillViewportHeight(true);
150     Tableau1.setPreferredScrollableViewportSize(preferredSize);
151     pan1.add(new JScrollPane(Tableau1), BorderLayout.CENTER);
152     Display_Table_holiday.add(pan1);
153

```

```

154     Holidaypan.add(Forme_holiday, BorderLayout.NORTH);
155     Forme_holiday.setLayout(new GridLayout(4, 2, 10, 10));
156     Forme_holiday.add(label_employe);
157     Forme_holiday.add(text_employe);
158     Forme_holiday.add(label_startDate);
159     Forme_holiday.add(text_startDate);
160     Forme_holiday.add(label_endDate);
161     Forme_holiday.add(text_endDate);
162     Forme_holiday.add(label_type);
163     Forme_holiday.add(TypeComboBox);
164
165     Holidaypan.add(panButton_holiday, BorderLayout.SOUTH);
166     panButton_holiday.add(addButton_holiday);
167     panButton_holiday.add(updateButton_holiday);
168     panButton_holiday.add(deleteButton_holiday);
169     panButton_holiday.add(displayButton_holiday);
170     panButton_holiday.add(importButton_holiday);
171     panButton_holiday.add(exportButton_holiday);
172
173
174
175
176 // TabbedPane
177     tabbedPane.addTab("Employe", employeTab);
178     tabbedPane.addTab("Holiday", holidayTab);
179     importButton_employe.addActionListener(e -> {
180         JFileChooser fileChooser = new JFileChooser();
181         if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
182             importData(tableModel, fileChooser.getSelectedFile().getPath());
183         }
184     });
185
186     exportButton_employe.addActionListener(e -> {
187         JFileChooser fileChooser = new JFileChooser();
188         if (fileChooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
189             exportData(tableModel, fileChooser.getSelectedFile().getPath());
190         }
191     });
192     importButton_holiday.addActionListener(e -> {
193         JFileChooser fileChooser = new JFileChooser();
194         if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
195             importData(tableModell, fileChooser.getSelectedFile().getPath());
196         }
197     });
198
199     exportButton_holiday.addActionListener(e -> {
200         JFileChooser fileChooser = new JFileChooser();
201         if (fileChooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
202             exportData(tableModell, fileChooser.getSelectedFile().getPath());
203         }
204     });
205
206     remplace_les_employes();
207     setVisible(true);
208 }

```



```
209
210 public void remplaier_les_employes () {
211     List<Employee> Employes = new Employemodell(new EmployeeDAOimpl()).
displayEmployee();
212     text_employe.removeAllItems();
213     for (Employee elem : Employes) {
214         text_employe.addItem(elem.getId() + " - " + elem.getNom()+" "+elem.
getPrenom());
215     }
216 }
217
218
219
220 // getters
221
222
223     public int getId_employe() {
224         return Integer.parseInt(text_employe.getSelectedItem().toString().
split(" - ")[0]);
225     }
226     public String getNom() {
227         return text_nom.getText();
228     }
229
230     public JTable getTable() {
231         return (JTable) Display_Table_employe.getComponent(0);
232     }
233
234     public String getPrenom() {
235         return text_prenom.getText();
236     }
237
238     public String getEmail() {
239         return text_email.getText();
240     }
241
242     public String getTelephone() {
243         return text_tele.getText();
244     }
245
246     public double getSalaire() {
247         return Double.parseDouble(text_salaire.getText());
248     }
249
250     public Role getRole() {
251         return (Role) roleComboBox.getSelectedItem();
252     }
253
254     public Post getPoste() {
255         return (Post) posteComboBox.getSelectedItem();
256     }
257
258     public JButton getaddButton_employe () {
259         return addButton_employe;
260     }
```

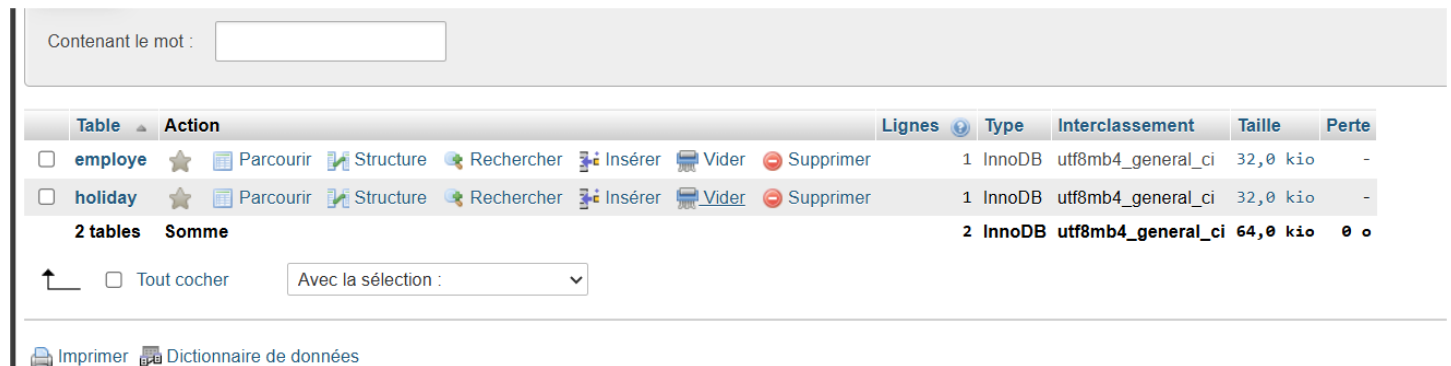
```
261     public JButton getupdateButton_employe () {
262         return updateButton_employe;
263     }
264
265     public JButton getdeleteButton_employe () {
266         return deleteButton_employe;
267     }
268
269     public JButton getdisplayButton_employe () {
270         return displayButton_employe;
271     }
272
273     public JButton getaddButton_holiday () {
274         return addButton_holiday;
275     }
276
277     public JButton getupdateButton_holiday () {
278         return updateButton_holiday;
279     }
280
281     public JButton getdeleteButton_holiday () {
282         return deleteButton_holiday;
283     }
284
285     public JButton getdisplayButton_holiday () {
286         return displayButton_holiday;
287     }
288     public String getStartDate () {
289         return text_startDate.getText();
290     }
291
292     public String getEndDate() {
293         return text_endDate.getText();
294     }
295
296     public Type_holiday getType_holiday(){
297         return (Type_holiday) TypeComboBox.getSelectedItem();
298     }
299
300     // methods d'affichage des messages
301     public void afficherMessageErreur(String message) {
302         JOptionPane.showMessageDialog(this, message, "Erreur", JOptionPane.
ERROR_MESSAGE);
303     }
304
305     public void afficherMessageSucces(String message) {
306         JOptionPane.showMessageDialog(this, message, " S u c c s ", JOptionPane.
INFORMATION_MESSAGE);
307     }
308
309     // methodes de vider les champs
310     public void viderChamps_em() {
311         text_nom.setText("");
312         text_prenom.setText("");
313         text_email.setText("");
```

```
314         text_tele.setText("");
315         text_salaire.setText("");
316         roleComboBox.setSelectedIndex(0);
317         posteComboBox.setSelectedIndex(0);
318     }
319
320     public void viderChamps_ho() {
321         text_startDate.setText("");
322         text_endDate.setText("");
323         TypeComboBox.setSelectedIndex(0);
324     }
325
326     // methodes de remplir les champs
327     public void rempilaireChamps_em (int id, String nom, String prenom, String
email, String telephone, double salaire, Role role, Post poste) {
328         text_nom.setText(nom);
329         text_prenom.setText(prenom);
330         text_email.setText(email);
331         text_tele.setText(telephone);
332         text_salaire.setText(String.valueOf(salaire));
333         roleComboBox.setSelectedItem(role);
334         posteComboBox.setSelectedItem(poste);
335     }
336
337     public void rempilaireChamps_ho(int id_employe, String date_debut, String
date_fin, Type_holiday type) {
338         List<Employe> Employes = new Employemodel(new EmployeeDAOimpl()).
displayEmploye();
339         text_employe.removeAllItems();
340         for (Employe elem : Employes) {
341             if (elem.getId() == id_employe) {
342                 text_employe.addItem(elem.getId() + " - " + elem.getNom()+" "+
elem.getPrenom());
343                 text_employe.setSelectedItem(elem.getId() + " - " + elem.
getNom()+" "+elem.getPrenom());
344             }
345         }
346         text_startDate.setText(date_debut);
347         text_endDate.setText(date_fin);
348         TypeComboBox.setSelectedItem(type);
349     }
350
351     // methodes de test des champs
352     public boolean testChampsVide_em () {
353         return text_nom.getText().equals("") || text_prenom.getText().equals
("") || text_email.getText().equals("") || text_tele.getText().equals("") ||
text_salaire.getText().equals("");
354     }
355
356     public boolean testChampsVide_ho () {
357         return text_employe.getSelectedItem().equals("") || text_startDate.
getText().equals("") || text_endDate.getText().equals("") || TypeComboBox.
getSelectedItem().equals("");
358     }
359
```

```
360
361
362
363 public void exportData(DefaultTableModel model, String fileName) {
364     try (PrintWriter writer = new PrintWriter(new FileWriter(fileName))) {
365         for (int i = 0; i < model.getColumnCount(); i++) {
366             writer.print(model.getColumnNames(i));
367             if (i < model.getColumnCount() - 1) writer.print(",");
368         }
369         writer.println();
370         for (int i = 0; i < model.getRowCount(); i++) {
371             for (int j = 0; j < model.getColumnCount(); j++) {
372                 writer.print(model.getValueAt(i, j));
373                 if (j < model.getColumnCount() - 1) writer.print(",");
374             }
375             writer.println();
376         }
377         JOptionPane.showMessageDialog(this, "Donn es export es avec succ s
378     .");
379     } catch (Exception e) {
380         JOptionPane.showMessageDialog(this, "Erreur lors de l'exportation: "
381     + e.getMessage(), "Erreur", JOptionPane.ERROR_MESSAGE);
382     }
383 }
384
385 public void importData(DefaultTableModel model, String fileName) {
386     try (BufferedReader reader = new BufferedReader(new FileReader(fileName)))
387     {
388         model.setRowCount(0);
389         String line = reader.readLine();
390         while ((line = reader.readLine()) != null) {
391             String[] data = line.split(",");
392             model.addRow(data);
393         }
394         JOptionPane.showMessageDialog(this, "Donn es import es avec succ s
395     .");
396     } catch (Exception e) {
397         JOptionPane.showMessageDialog(this, "Erreur lors de l'importation: "
398     + e.getMessage(), "Erreur", JOptionPane.ERROR_MESSAGE);
399     }
400 }
```

# Résultats

## 1 Tables Créées



Contenant le mot :

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> employe	★ Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	32,0 kio	-
<input type="checkbox"/> holiday	★ Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	32,0 kio	-
<b>2 tables</b>	<b>Somme</b>	<b>2</b>	<b>InnoDB</b>	<b>utf8mb4_general_ci</b>	<b>64,0 kio</b>	<b>0 o</b>

↑ ☐ Tout cocher Avec la sélection : ▼

Imprimer Dictionnaire de données

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.

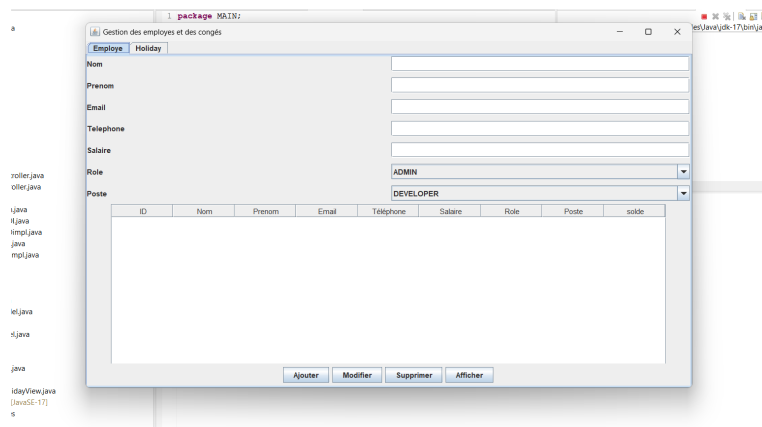


FIGURE 2.2 – Interface Utilisateur

### 3 Résultats du Bouton Importer

Le bouton Importer permet de charger les données des employés à partir d'un fichier "Importer.txt". Cette fonctionnalité est utile pour intégrer rapidement un grand volume de données dans l'application sans les saisir manuellement. Lorsqu'il est cliqué, le bouton ouvre une boîte de dialogue permettant de sélectionner le fichier à importer. Les données sont ensuite analysées et ajoutées à la base de données après vérification de leur validité.

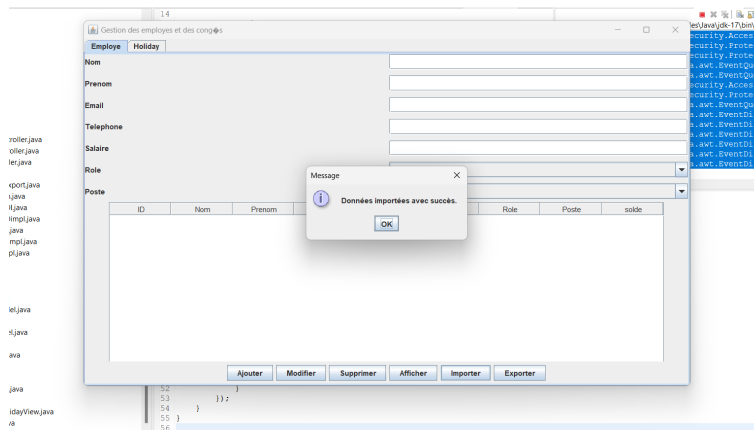


FIGURE 2.3 – Interface de l'importation du fichier

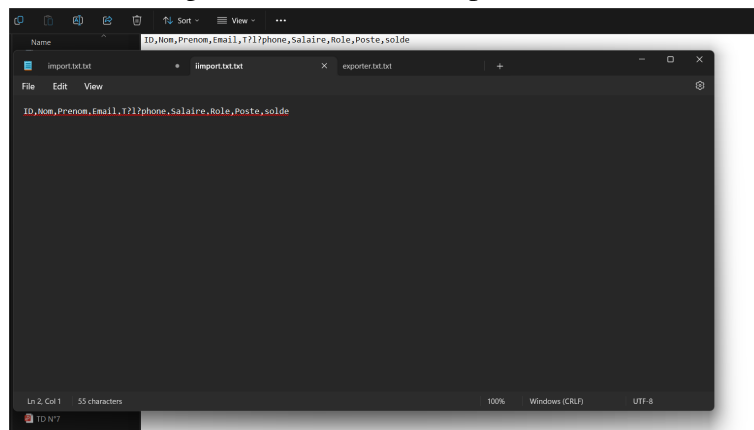


FIGURE 2.4 – Résultat de l'importation du fichier

### 4 Résultats du Bouton Exporter

Le bouton Exporter permet de sauvegarder les données des employés dans un fichier externe "exporter.txt". Cette fonctionnalité est idéale pour générer des rapports, partager des données ou créer des sauvegardes. En cliquant sur le bouton, l'utilisateur peut choisir un emplacement pour enregistrer le fichier exporté, qui contient toutes les informations des employés.

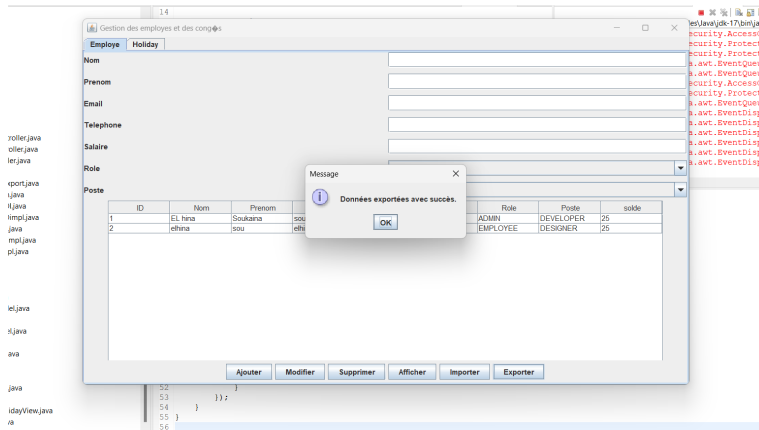


FIGURE 2.5 – Interface de l'exportation du fichier

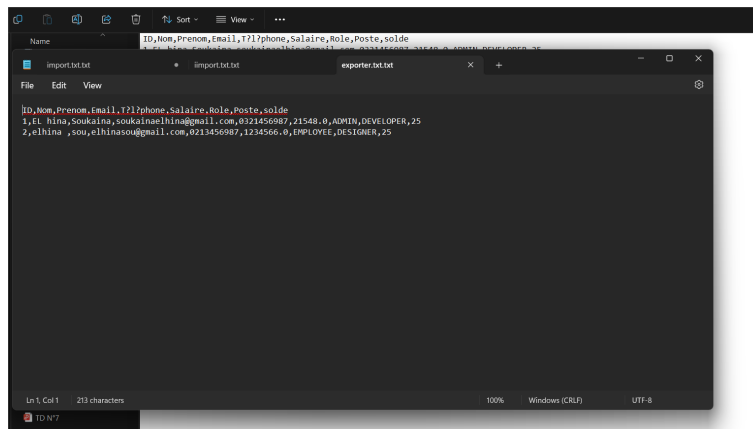


FIGURE 2.6 – Résultat de l'exportation du fichier

# Conclusion générale

Ce travail pratique a permis de concevoir et de développer une application complète dédiée à la gestion des entrées et sorties des employés, intégrant des fonctionnalités clés adaptées aux besoins d'une entreprise. Parmi celles-ci figurent l'importation des données des employés à partir de fichiers externes, l'exportation des données pour des besoins d'archivage ou de partage, ainsi qu'une gestion optimisée des informations relatives aux employés.

L'application a été conçue pour offrir une utilisation intuitive grâce à une interface graphique conviviale, simplifiant ainsi les processus de gestion. Des mécanismes de validation ont été mis en place pour garantir l'exactitude des données importées, notamment en vérifiant les formats des fichiers et les champs obligatoires. L'exportation produit des fichiers bien structurés, facilitant leur exploitation dans d'autres outils ou systèmes.

Ce projet a également été une opportunité d'approfondir nos compétences techniques dans plusieurs domaines. Nous avons renforcé notre maîtrise de la programmation orientée objet, notamment en appliquant le modèle d'architecture MVC, et perfectionné nos connaissances en bases de données grâce à leur intégration efficace avec Java via JDBC. Par ailleurs, le développement d'une interface graphique avec Swing a consolidé notre aptitude à concevoir des applications interactives et ergonomiques.

En répondant à un besoin concret, ce travail pratique illustre l'importance des solutions numériques pour optimiser la gestion des employés. Il constitue une expérience formatrice et enrichissante dans notre parcours professionnel.

# 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/>