



Université Cadi Ayyad – Marrakech
Ecole Supérieure de Technologie – Safi
Département : Informatique
Filière : Génie Informatique

Compte Rendu TP3 Java

Gestion des employés et congés Entrées/Sorties



Réalisé par : TISSAFI IDRISSE Zineb

Encadré par : Mme EL KOURCHI Asmaa

Année universitaire : 2024-2025

1 Suite du TP 2 : Implémentation du Login

Voici la structure de la table login dans ma base de données :
















#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/> 1	id 	int(11)			Non	Aucun(e)		AUTO_INCREMENT	 Modifier  Supprimer  Plus
<input type="checkbox"/> 2	id_employe 	int(11)			Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/> 3	username 	varchar(50)	utf8mb4_general_ci		Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/> 4	password	varchar(255)	utf8mb4_general_ci		Non	Aucun(e)			 Modifier  Supprimer  Plus

Figure 1: Structure du table login

- **Model (login):**

La classe Login permet de gérer l'authentification des utilisateurs, en stockant le nom d'utilisateur et le mot de passe. Elle inclut des méthodes pour accéder et modifier ces informations, ainsi qu'une méthode verifyCredentials pour vérifier le mot de passe saisi. La méthode toString fournit une représentation textuelle des données de connexion.

```

1 package Model;
2 public class Login {
3     private String username;
4     private String password;
5
6     public Login(String username, String password) {
7         this.username = username;
8         this.password = password;
9     }
10
11     public String getUsername() {
12         return username;
13     }
14
15     public void setUsername(String username) {
16         this.username = username;
17     }
18
19     public String getPassword() {

```

```
20     return password;
21 }
22
23 public void setPassword(String password) {
24     this.password = password;
25 }
26
27 public boolean verifyCredentials(String storedPassword) {
28     return this.password.equals(storedPassword);
29 }
30
31 @Override
32 public String toString() {
33     return "Login [username=" + username + ", password=" + password
34         + " ]";
35 }
```

- DAO(loginDAOImpl):

La classe LoginDAOimpl implémente une méthode d'authentification qui vérifie si le nom d'utilisateur et le mot de passe fournis correspondent à ceux stockés dans la base de données. Elle utilise une requête SQL pour récupérer le mot de passe correspondant au nom d'utilisateur, puis compare les deux. Si les mots de passe correspondent, l'utilisateur est authentifié.

```
1 package DAO;
2
3 import java.sql.PreparedStatement;
4 import java.sql.ResultSet;
5 import java.sql.SQLException;
6
7 public class LoginDAOimpl {
8
9     // Method to check if the username and password are valid
10    public boolean authenticate(String username, String password) {
11        String sql = "SELECT password FROM login WHERE username = ?";
12        try (PreparedStatement stmt = DBConnexion.getConnexion().
            prepareStatement(sql)) {
```

```
13         stmt.setString(1, username);
14         try (ResultSet rs = stmt.executeQuery()) {
15             if (rs.next()) {
16                 String storedPassword = rs.getString("password");
17                 return storedPassword.equals(password); // Return
18                     true if passwords match
19             }
20         } catch (SQLException | ClassNotFoundException exception) {
21             System.err.println("Error during authentication: " +
22                 exception.getMessage());
23             exception.printStackTrace();
24         }
25         return false;
26     }
```

- **Model(loginModel):**

La classe LoginModel gère l'authentification des utilisateurs en utilisant la classe LoginDAOimpl pour valider les identifiants. Elle sépare ainsi la logique métier de l'accès aux données.

```
1 package Model;
2 import DAO.LoginDAOimpl;
3
4 public class LoginModel {
5     private LoginDAOimpl loginDAO;
6     public LoginModel(LoginDAOimpl loginDAO) {
7         this.loginDAO = loginDAO;
8     }
9     // Authentication method
10    public boolean authenticate(String username, String password) {
11        return loginDAO.authenticate(username, password); // Delegate
12            to DAO
13    }
14 }
```

- **Controller(LoginController):**

La classe LoginController gère l'interaction entre la vue et le modèle pour l'authentification, en vérifiant les informations de connexion saisies et en affichant des messages d'erreur si nécessaire.

```
1 package Controller;
2
3 import Model.LoginModel;
4 import View.LoginView;
5
6 import java.awt.event.ActionEvent;
7 import java.awt.event.ActionListener;
8
9 public class LoginController {
10
11     private final LoginView loginView;
12     private final LoginModel loginModel;
13
14     public LoginController(LoginView loginView, LoginModel loginModel)
15     {
16         this.loginView = loginView;
17         this.loginModel = loginModel;
18
19         // Attach the login button listener
20         this.loginView.addLoginListener(new LoginListener());
21     }
22
23     private class LoginListener implements ActionListener {
24
25         @Override
26         public void actionPerformed(ActionEvent e) {
27             // Get the username and password from the login view
28             String username = loginView.getUsername();
29             String password = loginView.getPassword();
30
31             // Validate input fields
32             if (username.isEmpty() || password.isEmpty()) {
33                 loginView.showError("Username or password cannot be
```

```
        empty.");
33         return;
34     }
35
36     // Attempt to authenticate the user using the model
37     boolean isAuthenticated = loginModel.authenticate(username,
38         password);
39
40     if (isAuthenticated) {
41         // If authentication is successful, notify the user
42         loginView.close(); // Close the login view or proceed
43         to the next view
44     } else {
45         // If authentication fails, show an error message
46         loginView.showError("Invalid username or password.
47         Please try again.");
48     }
49 }
```

- View (LoginView):

```
1 package View;
2 import javax.swing.*;
3 import java.awt.*;
4 import java.awt.event.ActionListener;
5 public class LoginView extends JFrame {
6
7     private JTextField usernameField;
8     private JPasswordField passwordField;
9     private JButton loginButton;
10
11     public LoginView() {
12         setTitle("Login");
13         setSize(300, 200);
14         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15
16         usernameField = new JTextField(20);
```

```
17     passwordField = new JPasswordField(20);
18     loginButton = new JButton("Login");
19
20     setLayout(new FlowLayout());
21     add(new JLabel("Username:"));
22     add(usernameField);
23     add(new JLabel("Password:"));
24     add(passwordField);
25     add(loginButton);
26
27     setLocationRelativeTo(null); // Center the window
28 }
29
30 public String getUsername() {
31     return usernameField.getText();
32 }
33
34 public String getPassword() {
35     return new String(passwordField.getPassword());
36 }
37
38 public void addLoginListener(ActionListener listener) {
39     loginButton.addActionListener(listener);
40 }
41
42 public void showError(String message) {
43     JOptionPane.showMessageDialog(this, message, "Error",
44                                   JOptionPane.ERROR_MESSAGE);
45 }
46
47 public void close() {
48     this.setVisible(false);
49 }
```

- Main :

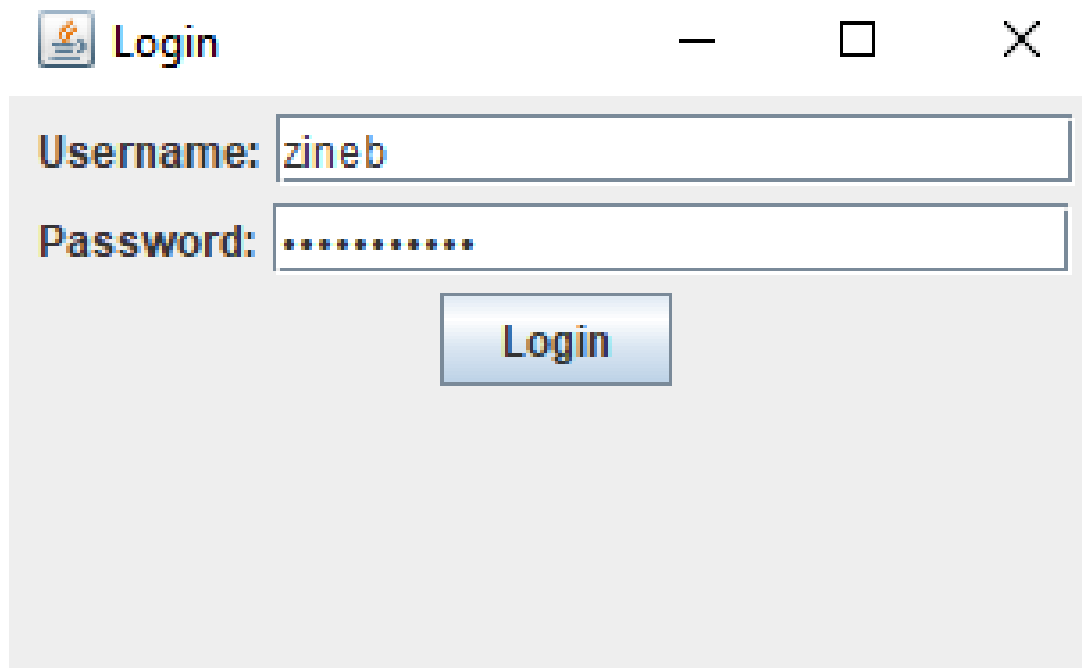
```
1     package Main;
2
```

```
3 import Controller.EmployeeController;
4 import Controller.HolidayController;
5 import Controller.LoginController;
6 import DAO.EmployeeDAOimpl;
7 import DAO.HolidayDAOimpl;
8 import DAO.LoginDAOimpl;
9 import Model.EmployeeModel;
10 import Model.HolidayModel;
11 import Model.LoginModel;
12 import View.Employee_HolidayView;
13 import View.LoginView;
14
15 public class Main {
16     public static void main(String[] args) {
17         // Create DAO instances
18         LoginDAOimpl loginDAO = new LoginDAOimpl();
19         EmployeeDAOimpl employeeDAO = new EmployeeDAOimpl();
20         HolidayDAOimpl holidayDAO = new HolidayDAOimpl();
21
22         // Create model instances
23         LoginModel loginModel = new LoginModel(loginDAO);
24         EmployeeModel employeeModel = new EmployeeModel(employeeDAO);
25         HolidayModel holidayModel = new HolidayModel(holidayDAO);
26
27         // Create views
28         LoginView loginView = new LoginView();
29         Employee_HolidayView employeeHolidayView = new
            Employee_HolidayView();
30
31         // Create controllers
32         new LoginController(loginView, loginModel);
33
34         // Show the login view
35         loginView.setVisible(true);
36
37         // Listen for login success to show the main application
38         loginView.addListener(e -> {
39             if (loginModel.authenticate(loginView.getUsername(),
                loginView.getPassword())) {
```



```
40         // Login successful
41         loginView.setVisible(false); // Hide login view
42
43         // Initialize controllers for employee and holiday
           management
44         new EmployeeController(employeeHolidayView, employeeModel)
           ;
45         new HolidayController(employeeHolidayView, holidayModel)
           ;
46
47         // Show the main application view
48         employeeHolidayView.setVisible(true);
49     } else {
50         // Login failed
51         loginView.showError("Invalid username or password.
           Please try again.");
52     }
53 });
54 }
55 }
```

1.1 Interface de login



The image shows a standard Windows-style login dialog box. The title bar at the top says 'Login' with a small icon on the left and standard minimize, maximize, and close buttons on the right. The main area has a light gray background. It contains two text input fields. The first is labeled 'Username:' and contains the text 'zineb'. The second is labeled 'Password:' and contains a series of dots to mask the password. Below these fields is a single button labeled 'Login' with a blue gradient and a 3D effect.

Figure 2: Login

1. Interface de Connexion

L'utilisateur saisit son nom d'utilisateur et son mot de passe. Un bouton "Login" permet de soumettre les informations

1. Accès Administrateur

Une fois connecté en tant qu'administrateur, l'utilisateur obtient un accès complet aux fonctionnalités de gestion des employés. Opérations Administratives

- **Ajout d'un Employé :** L'administrateur peut ajouter de nouveaux employés.
- **Suppression d'un Employé :** L'administrateur peut supprimer des employés existants.
- **Modification d'un Employé :** L'administrateur peut modifier les informations des employés.
- **Affichage des Employés :** L'administrateur peut afficher la liste des employés.








				id	id_employe	username	password	
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	11	11	zineb	password123
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	12	12	ayanadif	2005

Figure 3: Table Login

- **Cas d'erreur :**

En cas d'erreur, par exemple si l'utilisateur entre des informations incorrectes (nom d'utilisateur ou mot de passe erronés), un message d'erreur sera affiché dans la fenêtre de login, lui indiquant que l'authentification a échoué et lui demandant de vérifier ses informations et d'essayer à nouveau.

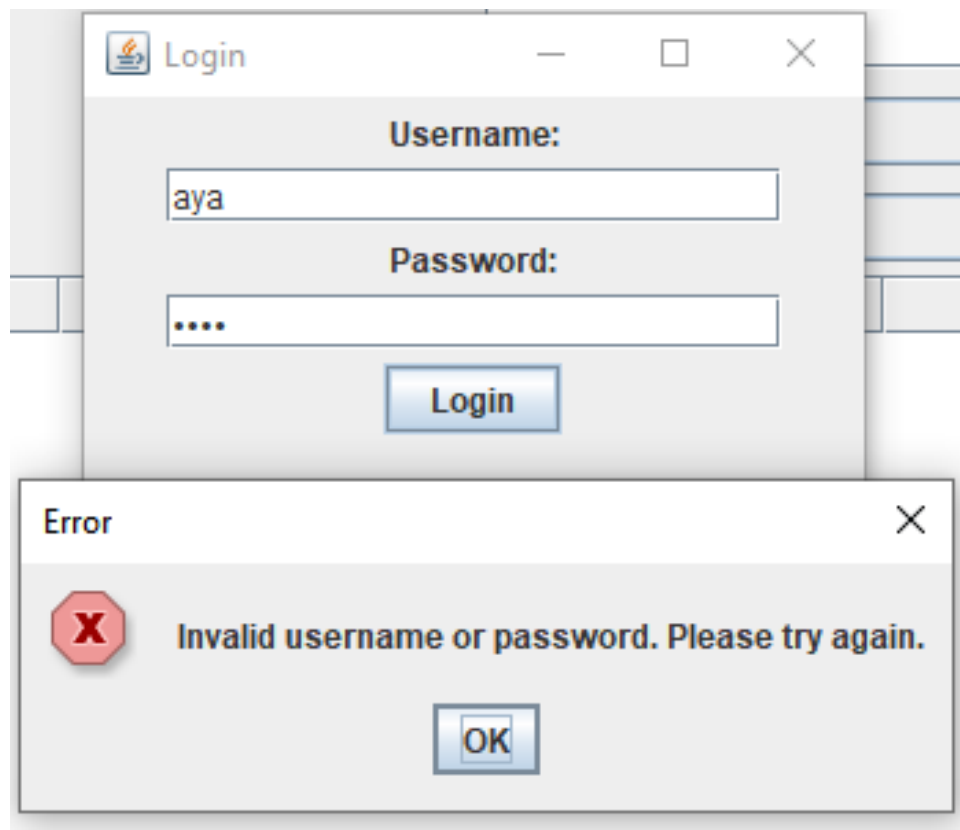


Figure 4: cas d'erreur

2 Gestion des Congés avec Fonctionnalités d'Importation et d'Exportation

2.1 structure de projet

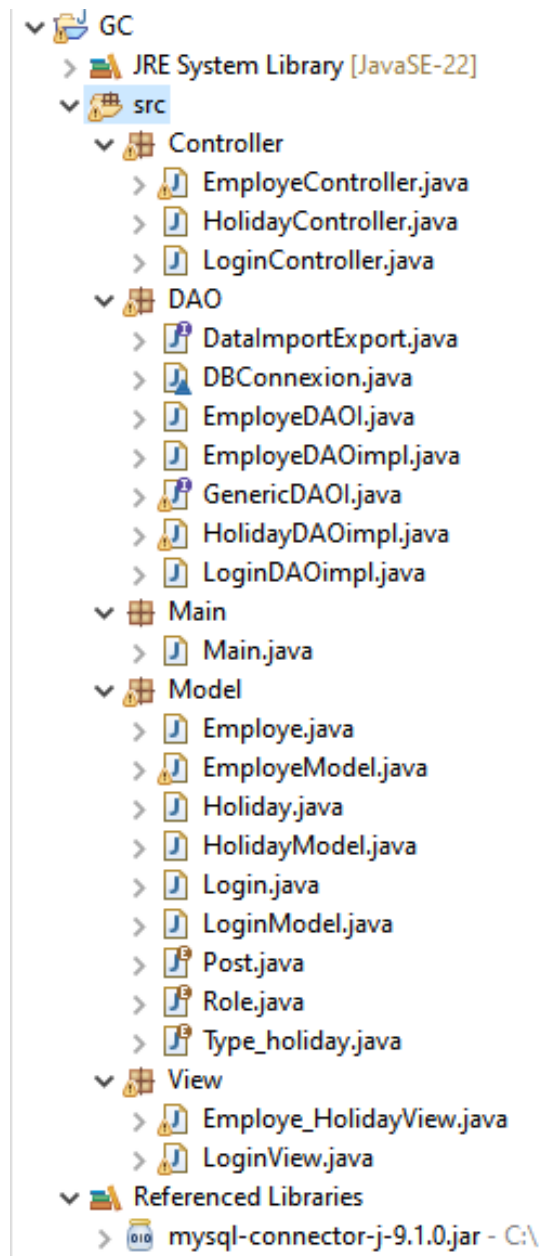


Figure 5: La structure

2.2 Gestion des Données (DAO)

1. Interface Générique pour l'Import/Export (DataImportExport)

```
1 package DAO;
2
3 import java.io.IOException;
4 import java.util.List;
5
6 public interface DataImportExport<T> {
7 void importData (String fileName)throws IOException;
8 void exportData (String fileName,List<T>data)throws IOException;
9 }
```

1. L'implémentation de cette interface par la classe : EmployeeDAOImpl

```
1         @Override
2 public void importData(String filePath) {
3     // Requête SQL pour insérer un nouvel employé dans la base de
4     // données
5     String query = "INSERT INTO Employe(nom, prenom, email, telephone,
6     salaire, role, poste) VALUES (?, ?, ?, ?, ?, ?, ?)";
7
8     // Tentative d'ouverture du fichier et de préparation de la
9     // requête SQL
10    try (BufferedReader reader = new BufferedReader(new FileReader(
11    filePath)); // Ouvre le fichier pour lecture
12
13        PreparedStatement psmt = DBConnexion.getConnexion().
14        prepareStatement(query)) { // Prépare la requête SQL
15
16        // Lire la première ligne (en-tête) et l'ignorer
17        String line = reader.readLine(); // Skip the header
18
19        // Lire chaque ligne du fichier CSV
20        while ((line = reader.readLine()) != null) {
21            // Séparer la ligne en fonction des virgules
22            String[] data = line.split(",");
23
24            // Vérifier que la ligne contient bien 7 éléments
25            if (data.length == 7) {
26                // Assigner les valeurs extraites de la ligne CSV aux
27                // paramètres de la requête SQL
28            }
29        }
30    }
```

```

21         psmt.setString(1, data[0].trim()); // nom
22         psmt.setString(2, data[1].trim()); // prenom
23         psmt.setString(3, data[2].trim()); // email
24         psmt.setString(4, data[3].trim()); // telephone
25         psmt.setString(5, data[4].trim()); // salaire
26         psmt.setString(6, data[5].trim()); // role
27         psmt.setString(7, data[6].trim()); // poste
28
29         // Ajouter la commande d'insertion dans le batch
30         psmt.addBatch();
31     } else {
32         // Afficher un message d'erreur si le format de la
33         // ligne est invalide
34         System.err.println("Invalid line format: " + line);
35     }
36
37     // Ex cuter toutes les commandes en une seule fois
38     psmt.executeBatch();
39     System.out.println("Employ s import s avec succ s.");
40 } catch (IOException | SQLException | ClassNotFoundException e) {
41     // Capturer et afficher les exceptions ventuelles
42     e.printStackTrace();
43 }
44 }
45
46 @Override
47 public void exportData(String fileName, List<Employe> data) throws
48     IOException {
49     // Tentative d' criture dans un fichier CSV
50     try (BufferedWriter writer = new BufferedWriter(new FileWriter(
51         fileName))) {
52         // crire l'en-t te du fichier CSV
53         writer.write("nom,prenom,email,telephone,role,poste,salaire");
54         writer.newLine();
55
56         // Parcourir la liste des employ s et les crire dans le
57         // fichier CSV
58         for (Employe employee : data) {

```

```

56      // Formater chaque ligne de données de l'employé en une
57      // chaîne CSV
58      String line = String.format("%s,%s,%s,%s,%s,%s,%.2f",
59          employee.getNom(),
60          employee.getPrenom(),
61          employee.getEmail(),
62          employee.getTelephone(),
63          employee.getRole(),
64          employee.getPost(),
65          employee.getSalaire());
66
67      // écrire la ligne dans le fichier
68      writer.write(line);
69      writer.newLine();
70
71      // Afficher un message indiquant que l'exportation a réussi
72      System.out.println("Données exportées avec succès.");
73  }
74  }

```

2.3 La logique de métier

1. Extension de la classe modèle (EmployeeModel) pour gérer l'import/export

```

1      // Vérifie si le fichier existe
2  private boolean checkFileExists(File file) {
3      // Si le fichier n'existe pas, une exception est lancée avec un
4      // message détaillant le chemin du fichier
5      if(!file.exists()) {
6          throw new IllegalArgumentException("Le fichier n'existe pas : "
7              + file.getPath());
8      }
9      return true; // Retourne true si le fichier existe
10 }

```

// Vérifie si le chemin spécifié correspond à un fichier et non un
 répertoire

```
11 private boolean checkIsFile(File file) {
12     // Si le chemin sp cifi n'est pas un fichier (c'est un
        // r pertoire), une exception est lanc e
13     if(!file.isFile()) {
14         throw new IllegalArgumentException("Le chemin sp cifi n'est
            pas un fichier : " + file.getPath());
15     }
16     return true; // Retourne true si c'est bien un fichier
17 }
18
19 // V rifie si le fichier peut tre lu
20 private boolean checkIsReadable(File file) {
21     // Si le fichier n'est pas lisible, une exception est lanc e avec
        // un message d taillant le chemin
22     if(!file.canRead()) {
23         throw new IllegalArgumentException("Le fichier sp cifi n'est
            pas lisible : " + file.getPath());
24     }
25     return true; // Retourne true si le fichier est lisible
26 }
```

2.4 Interface graphique(couche View)

1. L'ajout des boutons d'importation et d'exportation

```
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 Employee_HolidayView extends JFrame {
20
21     private JTabbedPane tabbedPane = new JTabbedPane();
22
23     private JPanel employeeTab = 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 cong
45     private JLabel label_employe = new JLabel("Nom de l'employee");
46     private JLabel label_startDate = new JLabel("Date de debut (YYYY-MM
47         -DD)");
48     private JLabel label_endDate = new JLabel("Date de fin (YYYY-MM-DD
49         ");
50     private JLabel label_type = new JLabel("Type");
51     private JComboBox<Type_holiday> TypeComboBox = new JComboBox<>(
52         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 conge
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", "Telephone", "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 Employee_HolidayView() {
97
98     setTitle("Gestion des employes et des conges");
99     setSize(1000, 600);
100     setDefaultCloseOperation(EXIT_ON_CLOSE);
101     setLocationRelativeTo(null);
102
103     add(tabbedPane);
104
105 // Employee 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.setFillableViewportHeight(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", employeeTab);
178     tabbedPane.addTab("Holiday", holidayTab);
179     importButton_employe.addActionListener(e -> {
180         JFileChooser fileChooser = new JFileChooser();
181         if (fileChooser.showOpenDialog(this) == JFileChooser.
182             APPROVE_OPTION) {
183             importData(tableModel, fileChooser.getSelectedFile().
184                 getPath());
185         }
186     });
187
188     exportButton_employe.addActionListener(e -> {
189         JFileChooser fileChooser = new JFileChooser();
190         if (fileChooser.showSaveDialog(this) == JFileChooser.
191             APPROVE_OPTION) {
192             exportData(tableModel, fileChooser.getSelectedFile().
193                 getPath());
194         }
195     });
196
197     importButton_holiday.addActionListener(e -> {
198         JFileChooser fileChooser = new JFileChooser();
199         if (fileChooser.showOpenDialog(this) == JFileChooser.
200             APPROVE_OPTION) {
```

```
195         importData(tableModel1, fileChooser.getSelectedFile().
196             getPath());
197     });
198
199     exportButton_holiday.addActionListener(e -> {
200         JFileChooser fileChooser = new JFileChooser();
201         if (fileChooser.showSaveDialog(this) == JFileChooser.
202             APPROVE_OPTION) {
203             exportData(tableModel1, fileChooser.getSelectedFile().
204                 getPath());
205         }
206     });
207
208     remplaire_les_employes();
209     setVisible(true);
210 }
211
212 public void remplaire_les_employes () {
213     List<Employee> Employees = new EmployeeModel(new EmployeeDAOimpl())
214         .displayEmployee();
215     text_employe.removeAllItems();
216     for (Employee elem : Employees) {
217         text_employe.addItem(elem.getId() + " - " + elem.getNom()+"
218             "+elem.getPrenom());
219     }
220 }
221
222 // getters
223 public int getId_employe() {
224     return Integer.parseInt(text_employe.getSelectedItem().
225         toString().split(" - ")[0]);
226 }
227 public String getNom() {
228     return text_nom.getText();
229 }
230
231 public JTable getTable() {
232     return (JTable) Display_Table_employe.getComponent(0);
233 }
```

```
228
229     public String getPrenom() {
230         return text_prenom.getText();
231     }
232
233     public String getEmail() {
234         return text_email.getText();
235     }
236
237     public String getTelephone() {
238         return text_tele.getText();
239     }
240
241     public double getSalaire() {
242         return Double.parseDouble(text_salaire.getText());
243     }
244
245     public Role getRole() {
246         return (Role) roleComboBox.getSelectedItem();
247     }
248
249     public Post getPoste() {
250         return (Post) posteComboBox.getSelectedItem();
251     }
252
253     public JButton getaddButton_employe () {
254         return addButton_employe;
255     }
256
257     public JButton getupdateButton_employe () {
258         return updateButton_employe;
259     }
260
261     public JButton getdeleteButton_employe () {
262         return deleteButton_employe;
263     }
264
265     public JButton getdisplayButton_employe () {
266         return displayButton_employe;
```

```
267     }
268
269     public JButton getaddButton_holiday () {
270         return addButton_holiday;
271     }
272
273     public JButton getupdateButton_holiday () {
274         return updateButton_holiday;
275     }
276     public JButton getdeleteButton_holiday () {
277         return deleteButton_holiday;
278     }
279
280     public JButton getdisplayButton_holiday () {
281         return displayButton_holiday;
282     }
283     public String getStartDate () {
284         return text_startDate.getText();
285     }
286
287     public String getEndDate() {
288         return text_endDate.getText();
289     }
290
291     public Type_holiday getType_holiday(){
292         return (Type_holiday) TypeComboBox.getSelectedItem();
293     }
294
295     // methods d'affichage des messages
296     public void afficherMessageErreur(String message) {
297         JOptionPane.showMessageDialog(this, message, "Erreur",
298             JOptionPane.ERROR_MESSAGE);
299     }
300
301     public void afficherMessageSucces(String message) {
302         JOptionPane.showMessageDialog(this, message, "Succes",
303             JOptionPane.INFORMATION_MESSAGE);
304     }
```



```
304 // methodes de vider les champs
305 public void viderChamps_em() {
306     text_nom.setText("");
307     text_prenom.setText("");
308     text_email.setText("");
309     text_tele.setText("");
310     text_salaire.setText("");
311     roleComboBox.setSelectedIndex(0);
312     posteComboBox.setSelectedIndex(0);
313 }
314
315 public void viderChamps_ho() {
316     text_startDate.setText("");
317     text_endDate.setText("");
318     TypeComboBox.setSelectedIndex(0);
319 }
320
321 // methodes de remplir les champs
322 public void remplaireChamps_em (int id, String nom, String
323     prenom, String email, String telephone, double salaire, Role
324     role, Post poste) {
325     text_nom.setText(nom);
326     text_prenom.setText(prenom);
327     text_email.setText(email);
328     text_tele.setText(telephone);
329     text_salaire.setText(String.valueOf(salaire));
330     roleComboBox.setSelectedItem(role);
331     posteComboBox.setSelectedItem(poste);
332 }
333
334 public void remplaireChamps_ho(int id_employe, String
335     date_debut, String date_fin, Type_holiday type) {
336     List<Employe> Employes = new EmployeeModel(new
337         EmployeeDAOimpl()).displayEmployee();
338     text_employe.removeAllItems();
339     for (Employe elem : Employes) {
340         if (elem.getId() == id_employe) {
341             text_employe.addItem(elem.getId() + " - " + elem.
342                 getNom()+" "+elem.getPrenom());
343         }
344     }
345 }
```

```
338         text_employe.setSelectedItem(elem.getId() + " - " +
339             elem.getNom()+" "+elem.getPrenom());
340     }
341     text_startDate.setText(date_debut);
342     text_endDate.setText(date_fin);
343     TypeComboBox.setSelectedItem(type);
344 }
345
346 // methodes de test des champs
347 public boolean testChampsVide_em (){
348     return text_nom.getText().equals("") || text_prenom.getText()
349         .equals("") || text_email.getText().equals("") ||
350         text_tele.getText().equals("") || text_salaire.getText()
351         .equals("");
352 }
353
354 public boolean testChampsVide_ho () {
355     return text_employe.getSelectedItem().equals("") ||
356         text_startDate.getText().equals("") || text_endDate.
357         getText().equals("") || TypeComboBox.getSelectedItem().
358         equals("");
359 }
360
361 public void exportData(DefaultTableModel model, String fileName) {
362     try (PrintWriter writer = new PrintWriter(new FileWriter(
363         fileName))) {
364         for (int i = 0; i < model.getColumnCount(); i++) {
365             writer.print(model.getColumnNames(i));
366             if (i < model.getColumnCount() - 1) writer.print(",");
367         }
368         writer.println();
369         for (int i = 0; i < model.getRowCount(); i++) {
370             for (int j = 0; j < model.getColumnCount(); j++) {
371                 writer.print(model.getValueAt(i, j));
372                 if (j < model.getColumnCount() - 1) writer.print(",");
373             }
374             writer.println();
375         }
376     }
377 }
```

```

        ");
    }
    writer.println();
}
JOptionPane.showMessageDialog(this, "Donn es export es
avec succ s.");
} catch (Exception e) {
    JOptionPane.showMessageDialog(this, "Erreur lors de l'
exportation : " + e.getMessage(), "Erreur", JOptionPane
.ERROR_MESSAGE);
}
}

public void importData(DefaultTableModel model, String fileName) {
    try (BufferedReader reader = new BufferedReader(new FileReader(
fileName))) {
        model.setRowCount(0);
        String line = reader.readLine();
        while ((line = reader.readLine()) != null) {
            String[] data = line.split(",");
            model.addRow(data);
        }
        JOptionPane.showMessageDialog(this, "Donn es import es
avec succ s.");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Erreur lors de l'
importation : " + e.getMessage(), "Erreur", JOptionPane
.ERROR_MESSAGE);
    }
}
}
}

```

2.5 contrôleur(couche Controller)

```

1 package Controller;
2 import Model.*;
3 import View.*;
4 import java.sql.Date;

```

```
5 import java.util.Calendar;
6 import java.util.List;
7 import javax.swing.table.DefaultTableModel;
8 public class EmployeeController {
9     private final Employee_HolidayView View;
10    public static EmployeeModel model_employe ;
11    public static int id = 0;
12    public static int oldselectedrow = -1;
13    public static boolean test = false;
14    String nom = "";
15    String prenom = "";
16    String email = "";
17    String telephone = "";
18    double salaire = 0;
19    Role role = null;
20    Post poste = null;
21    int solde = 0;
22    boolean updatereussi = false;
23    public EmployeeController(Employee_HolidayView view, EmployeeModel
24        model) {
25        this.View = view;
26        this.model_employe = model;
27
28        View.getaddButton_employe().addActionListener(e -> addEmployee()
29            );
30        View.getdeleteButton_employe().addActionListener(e ->
31            deleteEmployee());
32        View.getupdateButton_employe().addActionListener(e ->
33            updateEmployee());
34        View.getdisplayButton_employe().addActionListener(e ->
35            displayEmployee());
36        Employee_HolidayView.Tableau.getSelectionModel().
37            addListSelectionListener(e -> updateEmployeebyselect());
38    }
39    public void displayEmployee() {
40        List<Employee> Employes = model_employe.displayEmployee();
41        if(Employes.isEmpty()){
42            View.afficherMessageErreur("Aucun employe.");
43        }
44    }
```

```
38      DefaultTableModel tableModel = (DefaultTableModel)
        Employee_HolidayView.Tableau.getModel();
39      tableModel.setRowCount(0);
40      for(Employe e : Employes){
41          tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.
            getPrenom(), e.getEmail(), e.getTelephone(), e.
            getSalaire(), e.getRole(), e.getPost(),e.getSolde()});
42      }
43      View.remplaire_les_employes();
44  }
45  // function of add Employe
46  private void addEmploye() {
47      String nom = View.getNom();
48      String prenom = View.getPrenom();
49      String email = View.getEmail();
50      String telephone = View.getTelephone();
51      double salaire = View.getSalaire();
52      Role role = View.getRole();
53      Post poste = View.getPoste();
54      View.viderChamps_em();
55      boolean adresseussi = model_employe.addEmploye(0,nom, prenom,
        email, telephone, salaire, role, poste ,25);
56      if(adresseussi == true){
57          View.afficherMessageSucces("L'employe a bien ete ajoutee.")
            ;
58          displayEmploye();
59      }else{
60          View.afficherMessageErreur("L'employe n'a pas ete ajoutee."
            );
61      }
62  }
63  // function of delete Employe :
64  private void deleteEmploye(){
65      int selectedrow = Employee_HolidayView.Tableau.getSelectedRow();
66      if(selectedrow == -1){
67          View.afficherMessageErreur("Veuillez selectionner une ligne
            .");
68      }else{
```

```
69         int id = (int) Employee_HolidayView.Tableau.getValueAt(  
70             selectedrow, 0);  
71         if(model_employe.deleteEmploye(id)){  
72             View.afficherMessageSucces("L'employe a bien ete  
73                 supprimer.");  
74             displayEmploye();  
75         }else{  
76             View.afficherMessageErreur("L'employe n'a pas ete  
77                 supprimer.");  
78         }  
79     }  
80     // function of Update :  
81     private void updateEmployebyselect(){  
82         int selectedrow = Employee_HolidayView.Tableau.getSelectedRow();  
83         if (selectedrow == -1) {  
84             return;  
85         }  
86         try{  
87             id = (int) Employee_HolidayView.Tableau.getValueAt(  
88                 selectedrow, 0);  
89             nom = (String) Employee_HolidayView.Tableau.getValueAt(  
90                 selectedrow, 1);  
91             prenom = (String) Employee_HolidayView.Tableau.getValueAt(  
92                 selectedrow, 2);  
93             email = (String) Employee_HolidayView.Tableau.getValueAt(  
94                 selectedrow, 3);  
95             telephone = (String) Employee_HolidayView.Tableau.getValueAt(  
96                 selectedrow, 4);  
97             salaire = (double) Employee_HolidayView.Tableau.getValueAt(  
98                 selectedrow, 5);  
99             role = (Role) Employee_HolidayView.Tableau.getValueAt(  
100                 selectedrow, 6);  
101             poste = (Post) Employee_HolidayView.Tableau.getValueAt(  
102                 selectedrow, 7);  
103             solde = (int) Employee_HolidayView.Tableau.getValueAt(  
104                 selectedrow, 8);  
105             View.remplaireChamps_em(id, nom, prenom, email, telephone,  
106                 salaire, role, poste);
```

```
95         test = true;
96     } catch (Exception e) {
97         View.afficherMessageErreur("Erreur lors de la
           r   cupration   des   d o n n e s ");
98     }
99 }
100 private void updateEmploye(){
101     if (!test) {
102         View.afficherMessageErreur("Veuillez d'abord selectionner
           une ligne a modifier.");
103         return;
104     }
105     try {
106         nom = View.getNom();
107         prenom = View.getPrenom();
108         email = View.getEmail();
109         telephone = View.getTelephone();
110         salaire = View.getSalaire();
111         role = View.getRole();
112         poste = View.getPoste();
113
114         boolean updateSuccessful = model_employe.updateEmploye(id,
           nom, prenom, email, telephone, salaire, role, poste ,
           solde);
115
116         if (updateSuccessful) {
117             test = false;
118             View.afficherMessageSucces("L'employe a ete modifie
           avec succes.");
119             displayEmploye();
120             View.viderChamps_em();
121         } else {
122             View.afficherMessageErreur("Erreur lors de la mise a
           jour de l'employee.");
123         }
124     } catch (Exception e) {
125
126         View.afficherMessageErreur("Erreur lors de la mise a jour")
           ;

```

```
127     }
128 }
129 public void resetSolde(){
130     Calendar now = Calendar.getInstance();
131     if(now.get(Calendar.DAY_OF_YEAR) == 1){
132         for (Employee employe : model_employe.displayEmploye()) {
133             updateSolde(employe.getId(), 25);
134         }
135     }
136 }
137 public static void updateSolde(int id , int solde){
138     boolean updateSuccessful = model_employe.updateSolde(id, solde)
139     ;
140 }
141 }
```

2.6 Main

Le main reset toujours le même.

3 Vue d'Ensemble de l'Interface (Export/Import)

Dans cette interface employé, nous avons ajouté deux boutons : l'un pour l'exportation et l'autre pour l'importation.

Gestion des employes et des congés

Employe Holiday

Nom

Prenom

Email

Telephone

Salaire

Role

Poste

ID	Nom	Prenom	Email	Telephone	Salaire	Role	Poste	solde
11	Tissafi	zineb	zineb@gmail.co...	0620223318	50000.0	MANAGER	DEVELOPER	1
12	nadif	aya	ayanadif@gmail...	0623333333	20000.0	ADMIN	MARKETING	6
13	khaledi	khaoula	khaoula@gmail...	0620202020	40000.0	EMPLOYEE	DESIGNER	4
14	ahmed	ali	ali@gmail.com	0777777777	2000.0	EMPLOYEE	DEVELOPER	1
15	fatima	dadih	dadihfatima@g...	0623242526	30000.0	MANAGER	MARKETING	1
16	reda	radir	reda@gmail.com	0677666666	10000.0	ADMIN	DEVELOPER	11
17	hiba	bimad	hiba@gmail.com	0645343435	20000.0	MANAGER	OTHER	25
19	Sanae	safir	sanae@gmail.c...	0707070707	20000.0	MANAGER	MARKETING	11
21	Tissafi	zineb	tiss@gmail.com	0620223316	5000.0	MANAGER	MARKETING	25

Ajouter Modifier Supprimer Afficher Importer Exporter

Figure 6: Interface Employé

3.1 Le cas de l'exportation :

Lorsque je clique sur le bouton **”Exporter”**, une fenêtre de dialogue s’ouvre, permettant de sélectionner l’emplacement où enregistrer le fichier exporté, de spécifier son nom et de confirmer l’opération en cliquant sur **”Save”**

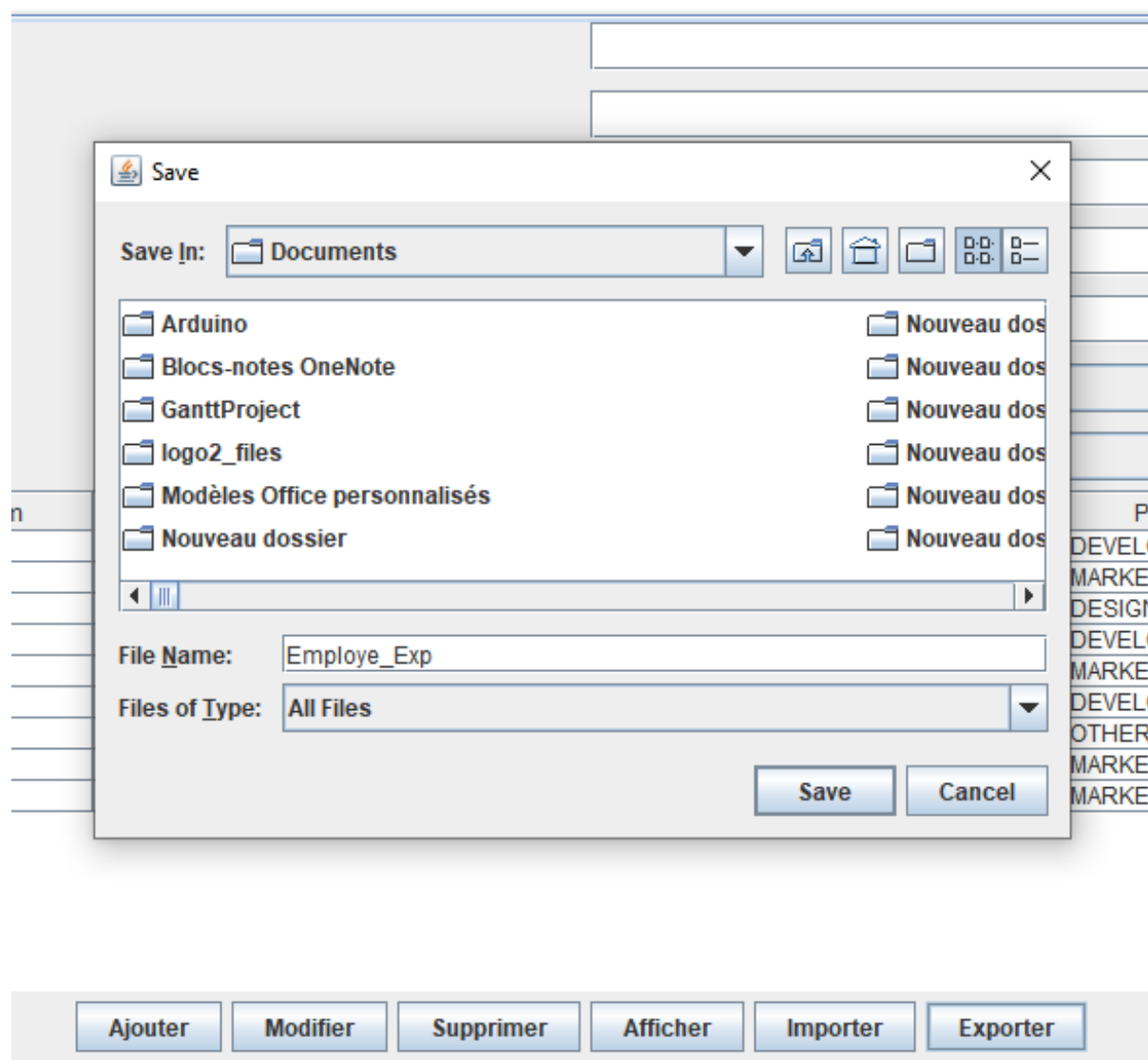


Figure 7: cas de l'exportation

- **Résultat de l'exportation**

Un fichier nommé "**EmployeeExp**" est créé et enregistré à l'emplacement choisi par l'utilisateur, contenant les données sélectionnées au format spécifié.



Figure 8: Résultat de l'exportation

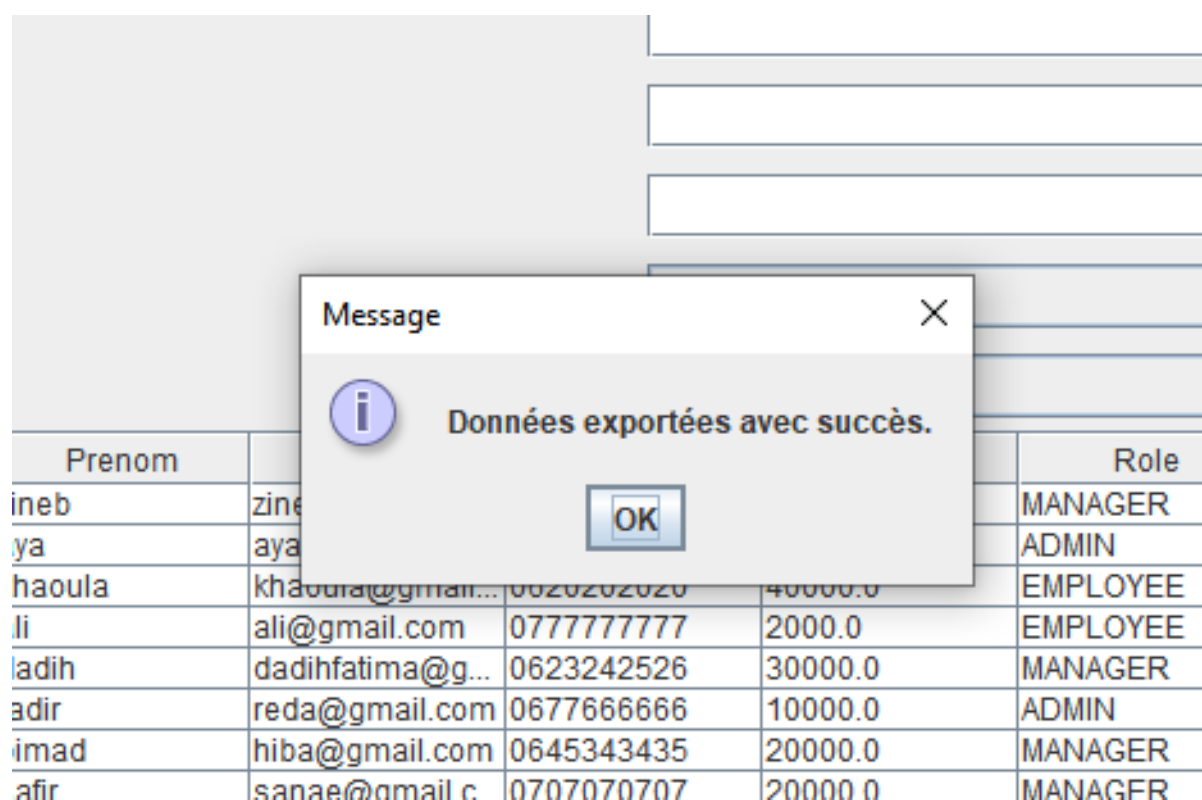


Figure 9: Message d'exportation réussie

3.2 Le cas de l'importation:

Voici le processus d'importation résumé :

1. Cliquer sur "Importer".

2. Sélectionner le fichier à importer.
3. Vérifier la compatibilité du fichier.
4. Cliquer sur "Importer" pour l'ajouter à l'application.
5. Manipuler et sauvegarder le fichier une fois importé

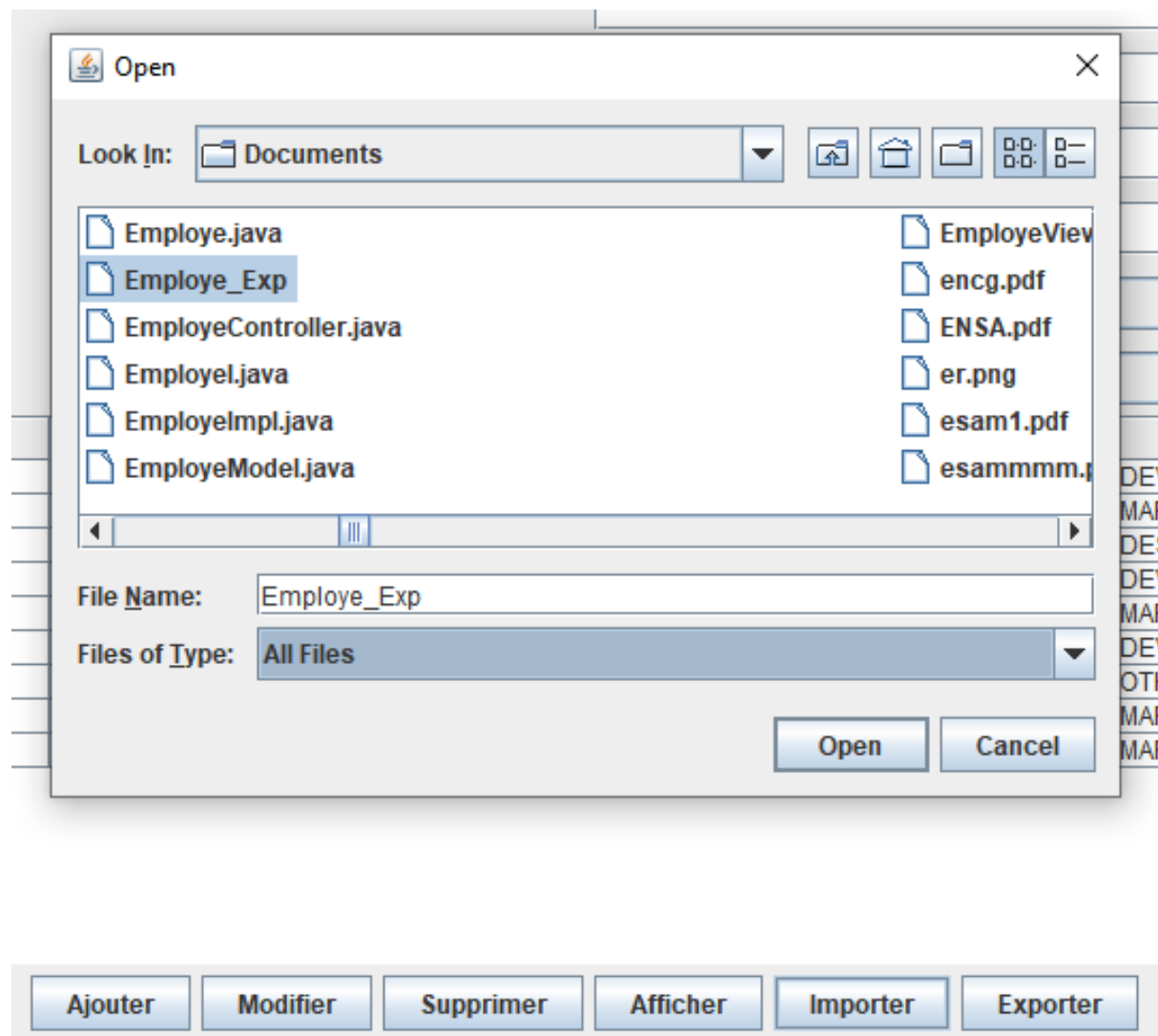


Figure 10: cas de l'importation

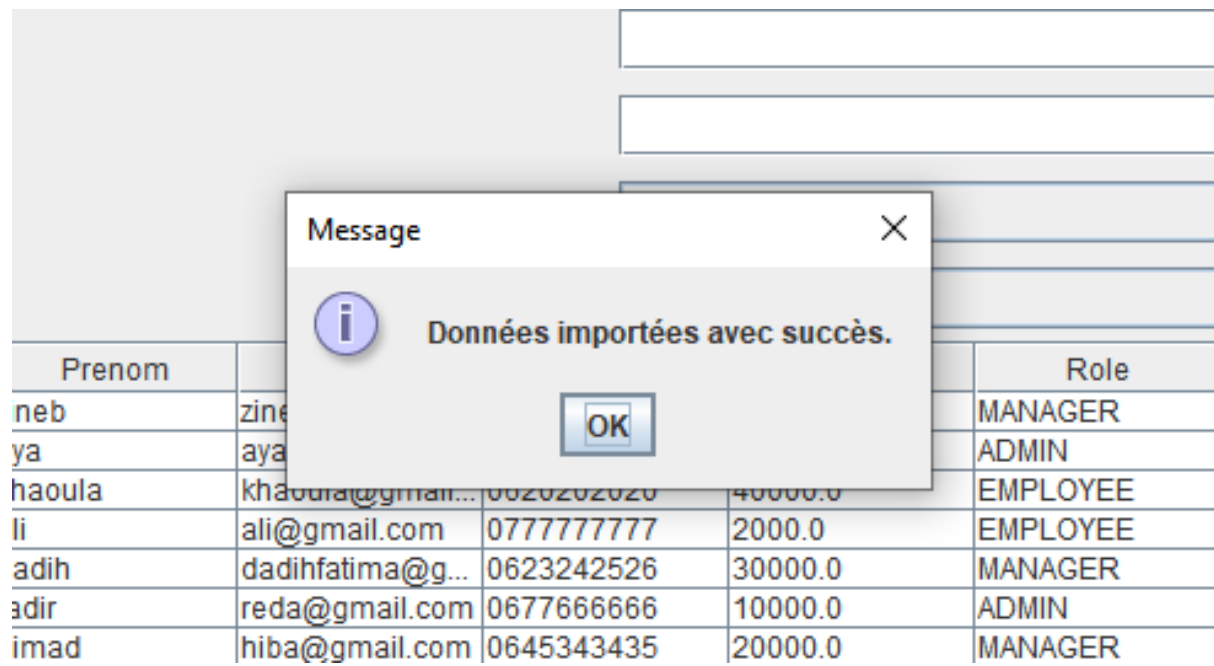


Figure 11: Message d'importation réussie