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

Compte Rendu TP3

Gestion des Congés en Java (Généricités ,MVC,DAO et E/S)

Réalisée par : DOUAE SAMTI

Encadrée par : Mme Asmae Elkourchi

Contents

1 Introduction

Le système de gestion des congés a été intégré au projet existant de gestion des employés pour permettre une administration efficace des absences des employés. Cette fonctionnalité inclut la planification, la modification et le suivi des congés en fonction des types définis (payé, non payé, maladie). L'entreprise SEA souhaite étendre ce système en ajoutant la possibilité d'importer et d'exporter des données relatives aux employés et à leurs congés. Cette extension permettra notamment l'exportation de la liste complète des employés, l'importation de nouvelles données sur les employés, ainsi que l'exportation spécifique des employés ayant des congés. Le rapport suivant détaille la structure et le fonctionnement de cette extension, y compris les composants DAO, Modèle et Vue, selon le modèle MVC.

2 Les fonctionnalités ajoutées incluent

- L'exportation de la liste complète des employés.
- L'importation de nouvelles données sur les employés.
- L'exportation spécifique des employés ayant des congés.
- La gestion des fichiers au format TXT ou CSV.

L'implémentation de ce module sera réalisée en plusieurs étapes, allant de la gestion des données avec DAO à l'ajout d'une interface graphique pour l'importation et l'exportation des données.

3 Architecture et Organisation du Projet

Le projet d'extension de l'application de gestion des congés s'articule autour de plusieurs composants majeurs. Le principe de généricité est utilisé pour gérer l'import/export des données des employés à travers une interface. Chaque couche du projet (DAO, Modèle, Vue, Contrôleur) est mise à jour pour intégrer ces nouvelles fonctionnalités.

3.1 Organisation des Packages

L'architecture du projet est organisée en plusieurs packages qui correspondent aux différentes couches du modèle MVC :

- **DAO (Data Access Object)** : Ce package contient l'interface et la classe qui gère l'import/export des données des employés dans la base de données.
- **Model** : Ce package contient la logique métier, notamment la gestion des employés, ainsi que l'implémentation des méthodes pour vérifier et manipuler les fichiers d'import/export.
- **View** : Ce package gère l'interface graphique, où deux boutons d'import et d'export sont ajoutés à l'interface de gestion des employés.
- **Controller** : Ce package contient le contrôleur qui gère les événements associés aux boutons d'import/export.

4 Étapes Suivies

Le développement de ce module a été réalisé en six étapes principales :

4.1 Étape 1 : Gestion des données (DAO)

Dans cette première étape, nous avons créé une interface générique `DataImportExport<T>` qui définit deux méthodes principales :

- `importData(String fileName)` : Lit les données depuis un fichier.
- `exportData(String fileName, List<T> data)` : Exporte les données vers un fichier.

L'implémentation de cette interface est assurée par la classe `EmployeeDAOImpl` qui gère l'accès aux données des employés. Les données sont lues et écrites à l'aide de classes comme `BufferedReader` pour l'import et `BufferedWriter` pour l'export.

4.2 Étape 2 : Logique métier (Couche Model)

Dans cette couche, la classe `EmployeeModel` a été étendue pour gérer l'import et l'export des données. Des vérifications sont effectuées sur les fichiers pour s'assurer qu'ils existent, qu'ils sont du bon type et que l'application a les droits nécessaires pour y accéder.

4.3 Étape 3 : Interface graphique (Couche View)

L'interface graphique a été mise à jour pour inclure deux boutons permettant à l'utilisateur d'importer et d'exporter les données. Ces boutons sont placés dans un `FlowLayout` pour un alignement naturel.

4.4 Étape 4 : Contrôleur (Couche Controller)

Le contrôleur a été étendu pour gérer les événements des boutons d'import et d'export. Lorsque l'utilisateur clique sur "Importer", un fichier est sélectionné et les données sont lues puis ajoutées à la base de données. Pour l'exportation, les données sont récupérées, formatées et écrites dans un fichier que l'utilisateur peut télécharger.

4.5 Étape 5 : Main

Enfin, la méthode `main` initialise l'application et configure les différentes couches pour fonctionner ensemble. La gestion des événements et des actions de l'utilisateur est orchestrée pour garantir une bonne interactivité avec l'application.

4.6 Étape 6 : Test et Validation

Des tests ont été réalisés pour valider que l'import et l'export des données fonctionnent correctement. Des fichiers texte et CSV ont été utilisés pour vérifier la compatibilité avec les formats de données attendus.

5 Les modifications du code

Dans cette section, nous détaillons les modifications apportées au code pour intégrer les fonctionnalités d'importation et d'exportation des données. Ces modifications comprennent l'ajout d'une nouvelle interface, de méthodes spécifiques dans les classes existantes, et des vérifications nécessaires pour la gestion des fichiers.

5.1 Ajout de la classe DataImportExport

Une interface générique `DataImportExport` a été ajoutée pour définir les opérations d'importation et d'exportation de données. Cette interface se trouve dans le package `DAO` :

Listing 1: Interface DataImportExport

```
package DAO;

import java.io.IOException;
import java.util.List;

public interface DataImportExport<T> {

    void importData(String fileName) throws IOException;

    void exportData(String fileName, List<T> data) throws IOException;
}
```

5.2 Modifications dans EmployeDAOImpl

Des méthodes spécifiques à l'importation et à l'exportation ont été ajoutées dans la classe `EmployeDAOImpl` :

- Importation des bibliothèques nécessaires :

Listing 2: Imports ajoutés

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.File;
```

- Vérification de l'existence du fichier :

Listing 3: `checkFileExists`

```
private void checkFileExists(File file) throws IOException {
    if (file == null || !file.exists()) {
        throw new IOException("Erreur␣:␣Le␣fichier␣n'existe␣pas.")
            ;
    }
    if (!file.isFile()) {
        throw new IOException("Erreur␣:␣Le␣chemin␣ne␣correspond␣
            pas␣\u00e0␣un␣fichier␣valide.");
    }
    if (!file.canRead()) {
```

```
        throw new IOException("Erreur : Le fichier ne peut pas \u00eatre lu.");
    }
}
```

- Méthode d'importation des données :

Listing 4: `importData` dans EmployeDAOImpl

```
public void importData(String fileName) throws IOException {
    File file = new File(fileName);
    checkFileExists(file);

    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split(",");
            if (parts.length == 9) {
                int id = Integer.parseInt(parts[0]);
                String nom = parts[1];
                String prenom = parts[2];
                String email = parts[3];
                String telephone = parts[4];
                double salaire = Double.parseDouble(parts[5]);
                Role role = Role.valueOf(parts[6]);
                Poste poste = Poste.valueOf(parts[7]);
                int solde = Integer.parseInt(parts[8]);

                addEmploye(id, nom, prenom, email, telephone, salaire, role, poste, solde);
            }
        }
    } catch (IOException | NumberFormatException e) {
        throw new IOException("Erreur lors de l'importation des donn es : " + e.getMessage(), e);
    }
}
```

- Méthode d'exportation des données :

Listing 5: `exportData` dans EmployeDAOImpl

```
public void exportData(String fileName, List<Employe> data) throws IOException {
    File file = new File(fileName);

    try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
        for (Employe employe : data) {
            String line = String.format("%d,%s,%s,%s,%s,%.2f,%s,%s,%d",
                    employe.getId(),
                    employe.getNom(),
                    employe.getPrenom(),
```

```
                        employe.getEmail(),
                        employe.getTelephone(),
                        employe.getSalaire(),
                        employe.getRole(),
                        employe.getPoste(),
                        employe.getSolde());
                writer.write(line);
                writer.newLine();
            }
        } catch (IOException e) {
            throw new IOException("Erreur lors de l'exportation des
                donn es : " + e.getMessage(), e);
        }
}
```

5.3 Modifications dans EmployeModel

Dans la classe `EmployeModel`, les méthodes `importData` et `exportData` ont été ajoutées pour intégrer les fonctionnalités avec `EmployeDAOImpl` :

Listing 6: Modifications dans `EmployeModel`

```
public void importEmployesFromFile(String fileName) {
    try {
        employeDAO.importData(fileName);
        System.out.println("Importation des employ s r ussie.");
    } catch (IOException e) {
        System.err.println("Erreur lors de l'importation : " + e.
            getMessage());
    }
}

public void exportEmployesToFile(String fileName) {
    try {
        List<Employe> employes = employeDAO.getAllEmployes();
        employeDAO.exportData(fileName, employes);
        System.out.println("Exportation des employ s r ussie.");
    } catch (IOException e) {
        System.err.println("Erreur lors de l'exportation : " + e.
            getMessage());
    }
}
```

5.4 View : ViewMain

La classe `MainView` est une fenêtre principale qui permet de gérer les employés et leurs congés. Elle utilise le framework `Swing` pour l'interface graphique. Elle comporte deux onglets principaux : un pour la gestion des employés et un autre pour la gestion des congés. Chaque onglet contient des panels et des boutons permettant de gérer les entités correspondantes.

Code Source

```
package View;

```

```java
import DAO.EmployeDAOImpl;
import Model.Employe;
import Model.EmployeModel;
import Model.Poste;
import Model.Role;
import Model.HolidayType;
import java.awt.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.util.List;

public class MainView extends JFrame {

    private JTabbedPane tabbedPane = new JTabbedPane();

    private JPanel employeTab = new JPanel();
    private JPanel holidayTab = new JPanel();

    private JPanel Employepan = new JPanel();
    private JPanel Holidaypan = new JPanel();
    private JPanel Display_Table_employe = new JPanel();
    private JPanel Display_Table_holiday = new JPanel();
    private final JPanel Forme_employe = new JPanel();
    private final JPanel Forme_holiday = new JPanel();
    private JPanel panButton_employe = new JPanel();
    private JPanel panButton_holiday = new JPanel();

    // les labels du l'employe
    private JLabel label_nom = new JLabel("Nom");
    private JLabel label_prenom = new JLabel("Prenom");
    private JLabel label_email = new JLabel("Email");
    private JLabel label_tele = new JLabel("Telephone");
    private JLabel label_salaire = new JLabel("Salaire");
    private JLabel label_role = new JLabel("Role");
    private JLabel label_poste = new JLabel("Poste");

    // les labels du cong
    private JLabel label_employe = new JLabel("Nom de l'employ ");
    private JLabel label_startDate = new JLabel("Date de debut
        (YYYY-MM-DD)");
    private JLabel label_endDate = new JLabel("Date de fin
        (YYYY-MM-DD)");
    private JLabel label_type = new JLabel("Type");
    private JComboBox<HolidayType> TypeComboBox = new
        JComboBox<>(HolidayType.values());

    // les textfield du l'employe
    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<Poste> posteComboBox = new
           JComboBox<>(Poste.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", "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 MainView() {
97
98          setTitle("Gestion des employes et des congs");
```

```
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.setFillsViewportHeight(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.setFillsViewportHeight(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(tableModel1,
                        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(tableModel1,
                        fileChooser.getSelectedFile().getPath());
203             }
204         });
205 
206         remplaire_les_employes();
207         setVisible(true);
208     }
209 
210     public void remplaire_les_employes () {
211         List<Employe> Employes = new EmployeModel(new
                EmployeDAOImpl()).displayEmploye();
212        text_employe.removeAllItems();
213        for (Employe 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 Poste getPoste() {
255             return (Poste) posteComboBox.getSelectedItem();
256         }
257 
258         public JButton getaddButton_employe () {
259             return addButton_employe;
260         }
261 
262         public JButton getupdateButton_employe () {
263             return updateButton_employe;
264         }
265 
266         public JButton getdeleteButton_employe () {
267             return deleteButton_employe;
268         }
269 
270         public JButton getdisplayButton_employe () {
271             return displayButton_employe;
272         }
273 
274         public JButton getaddButton_holiday () {
275             return addButton_holiday;
276         }
277 
278         public JButton getupdateButton_holiday () {
279             return updateButton_holiday;
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 HolidayType getHolidayType(){
297             return (HolidayType) 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, "Succes",
                    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 remplaireChamps_em (int id, String nom, String
                prenom, String email, String telephone, double salaire,
                Role role, Poste 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 remplaireChamps_ho(int id_employe, String
                date_debut, String date_fin, HolidayType type) {
338             List<Employe> Employes = new EmployeModel(new
                    EmployeDAOImpl()).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.getColumnName(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, "Exportation avec
                succ s.");
378         } catch (Exception e) {
379             JOptionPane.showMessageDialog(this, "Erreur lors de
                l'exportation : " + e.getMessage(), "Erreur",
                JOptionPane.ERROR_MESSAGE);
380         }
381     }
382 
383     public void importData(DefaultTableModel model, String fileName) {
384         try (BufferedReader reader = new BufferedReader(new
```

```
            FileReader(fileName))) {
385             model.setRowCount(0);
386             String line = reader.readLine();
387             while ((line = reader.readLine()) != null) {
388                 String[] data = line.split(",");
389                 model.addRow(data);
390             }
391             JOptionPane.showMessageDialog(this, "Importation avec
                    succ s.");
392         } catch (Exception e) {
393             JOptionPane.showMessageDialog(this, "Erreur lors de
                    l'importation : " + e.getMessage(), "Erreur",
                    JOptionPane.ERROR_MESSAGE);
394         }
395     }
396 }
```

5.5 Ajout de la classe Login

La classe `Login` représente un utilisateur avec des attributs pour le nom d'utilisateur (`username`) et le mot de passe (`password`). Elle inclut des méthodes pour manipuler ces attributs et vérifier la correspondance des identifiants.

```
1  package Model;
2
3  public class Login {
4
5      private String username;
6      private String password;
7
8      public Login(String username, String password) {
9          this.username = username;
10         this.password = password;
11     }
12
13     public String getUsername() {
14         return username;
15     }
16
17     public void setUsername(String username) {
18         this.username = username;
19     }
20
21     public String getPassword() {
22         return password;
23     }
24
25     public void setPassword(String password) {
26         this.password = password;
27     }
28
29     public boolean verifyCredentials(String storedPassword) {
30         return this.password.equals(storedPassword);
```

```
31      }
32
33      @Override
34      public String toString() {
35          return "Login [username=" + username + ", password=" +
                password + "]";
36      }
37  }
```

5.6 Ajout de la classe `LoginModel`

La classe `LoginModel` joue le rôle de couche intermédiaire pour gérer les opérations d'authentification. Elle utilise un DAO (`LoginDAOimpl`) pour accéder à la base de données et vérifier les identifiants.

```
1   package Model;
2
3   import DAO.LoginDAOimpl;
4
5   public class LoginModel {
6
7       private LoginDAOimpl loginDAO;
8
9       public LoginModel(LoginDAOimpl loginDAO) {
10          this.loginDAO = loginDAO;
11      }
12
13
14      public boolean authenticate(String username, String password) {
15          return loginDAO.authenticate(username, password);
16      }
17  }
```

5.7 Ajout de la classe `LoginDAOimpl`

La classe `LoginDAOimpl` gère les interactions avec la base de données pour vérifier les identifiants des utilisateurs. Elle utilise une requête SQL pour récupérer le mot de passe correspondant à un nom d'utilisateur.

```
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
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()) {
```

```
                if (rs.next()) {
                    String storedPassword = rs.getString("password");
                    return storedPassword.equals(password);
                }
            }
        } catch (SQLException | ClassNotFoundException exception) {
            System.err.println("Error during authentication: " +
               exception.getMessage());
            exception.printStackTrace();
        }
        return false;
    }
}
```

5.8 Ajout de la classe `LoginView`

La classe `LoginView` fournit une interface graphique pour la connexion des utilisateurs. Elle contient des champs de saisie pour le nom d'utilisateur et le mot de passe, ainsi qu'un bouton pour soumettre les informations.

```
package View;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;

public class LoginView extends JFrame {

    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;

    public LoginView() {
        setTitle("Login");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        usernameField = new JTextField(20);
        passwordField = new JPasswordField(20);
        loginButton = new JButton("Login");

        setLayout(new FlowLayout());
        add(new JLabel("Username:"));
        add(usernameField);
        add(new JLabel("Password:"));
        add(passwordField);
        add(loginButton);

        setLocationRelativeTo(null);
    }

    public String getUsername() {
        return usernameField.getText();
```

```
    }

    public String getPassword() {
        return new String(passwordField.getPassword());
    }

    public void addLoginListener(ActionListener listener) {
        loginButton.addActionListener(listener);
    }

    public void showError(String message) {
        JOptionPane.showMessageDialog(this, message, "Error",
            JOptionPane.ERROR_MESSAGE);
    }

    public void close() {
        this.setVisible(false);
    }
}
```

5.9 Ajout de la classe LoginController

La classe `LoginController` gère les interactions entre la vue (`LoginView`) et le modèle (`LoginModel`). Elle capture les actions de l'utilisateur et déclenche les processus d'authentification.

```
package Controller;

import Model.LoginModel;
import View.LoginView;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class LoginController {

    private final LoginView loginView;
    private final LoginModel loginModel;

    public LoginController(LoginView loginView, LoginModel loginModel)
        {
        this.loginView = loginView;
        this.loginModel = loginModel;
        this.loginView.addLoginListener(new LoginListener());
    }

    private class LoginListener implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {

            String username = loginView.getUsername();
            String password = loginView.getPassword();

```

```
28
29              if (username.isEmpty() || password.isEmpty()) {
30                  loginView.showError("Username or password cannot be
                        empty.");
31                  return;
32              }
33
34
35              boolean isAuthenticated =
                    loginModel.authenticate(username, password);
36
37              if (isAuthenticated) {
38
39                  loginView.showError("Login successful!");
40                  loginView.close();
41              } else {
42
43                  loginView.showError("Invalid username or password.
                        Please try again.");
44              }
45          }
46      }
47  }
```

5.10 Modifications dans `Main`

Le fichier `Main.java` a été modifié pour intégrer les nouvelles fonctionnalités d'importation et d'exportation de données ainsi la page de connexion Login. Voici le code mis à jour :

```
1  package Main;
2
3  import Controller.EmployeController;
4  import Controller.HolidayController;
5  import Controller.LoginController;
6  import DAO.EmployeDAOImpl;
7  import DAO.HolidayDAOImpl;
8  import DAO.LoginDAOimpl;
9  import Model.EmployeModel;
10 import Model.HolidayModel;
11 import Model.LoginModel;
12 import View.MainView;
13 import View.LoginView;
14
15 public class Main {
16     public static void main(String[] args) {
17         LoginDAOimpl loginDAO = new LoginDAOimpl();
18         EmployeDAOImpl employeDAO = new EmployeDAOImpl();
19         HolidayDAOImpl holidayDAO = new HolidayDAOImpl();
20         LoginModel loginModel = new LoginModel(loginDAO);
21         EmployeModel employeModel = new EmployeModel(employeDAO);
22         HolidayModel holidayModel = new HolidayModel(holidayDAO);
23
24         LoginView loginView = new LoginView();
```

```
25          MainView employeHolidayView = new MainView();
26
27          new LoginController(loginView, loginModel);
28
29          loginView.setVisible(true);
30          loginView.addLoginListener(e -> {
31              if (loginModel.authenticate(loginView.getUsername(),
                    loginView.getPassword())) {
32                  loginView.setVisible(false);
33
34                  new EmployeController(employeHolidayView,
                        employeModel);
35                  new HolidayController(employeHolidayView,
                        holidayModel);
36
37                  employeHolidayView.setVisible(true);
38              } else {
39                  loginView.showError("Invalid username or password.
                        Please try again.");
40              }
41          });
42      }
43  }
```

6 Fonctionnement de l'application

Cette section présente les interfaces graphiques du projet et leurs fonctionnalités principales :

6.1 Login

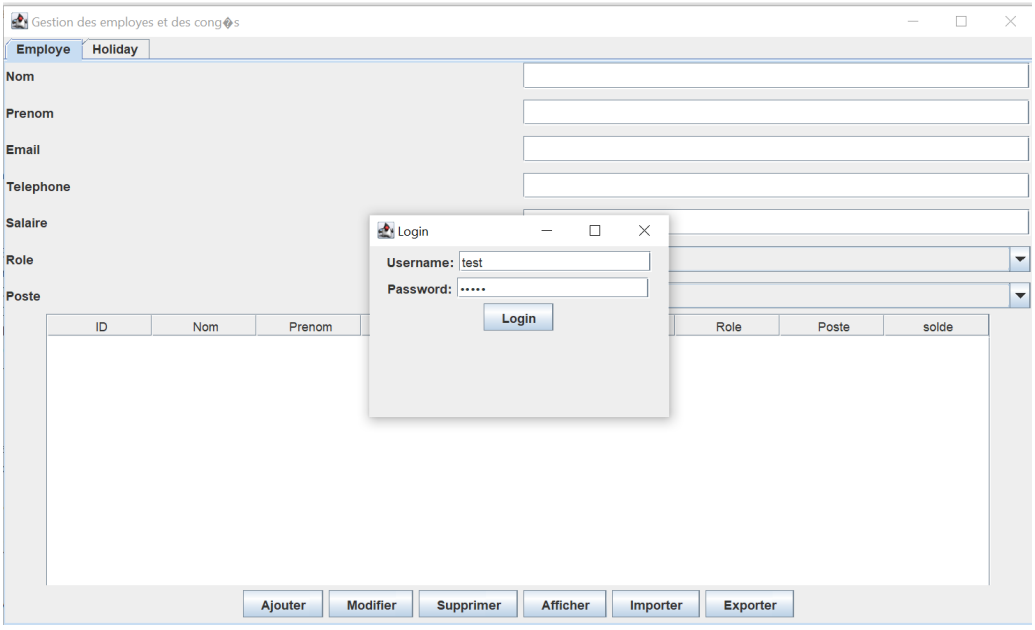

Figure 1: Interface pour la connexion .

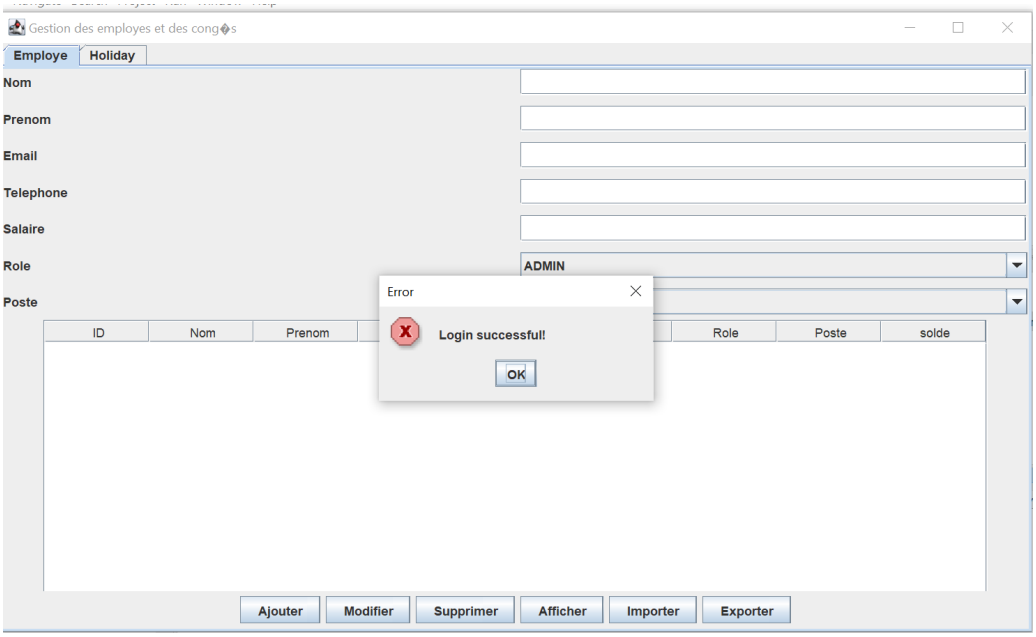

Figure 2: Interface pour la connexion .

6.2 Importation

Gestion des employés :

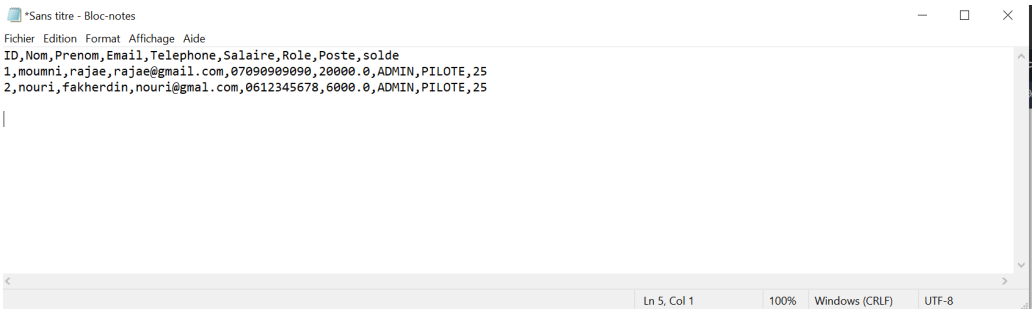

Figure 3: Fichier Importé .

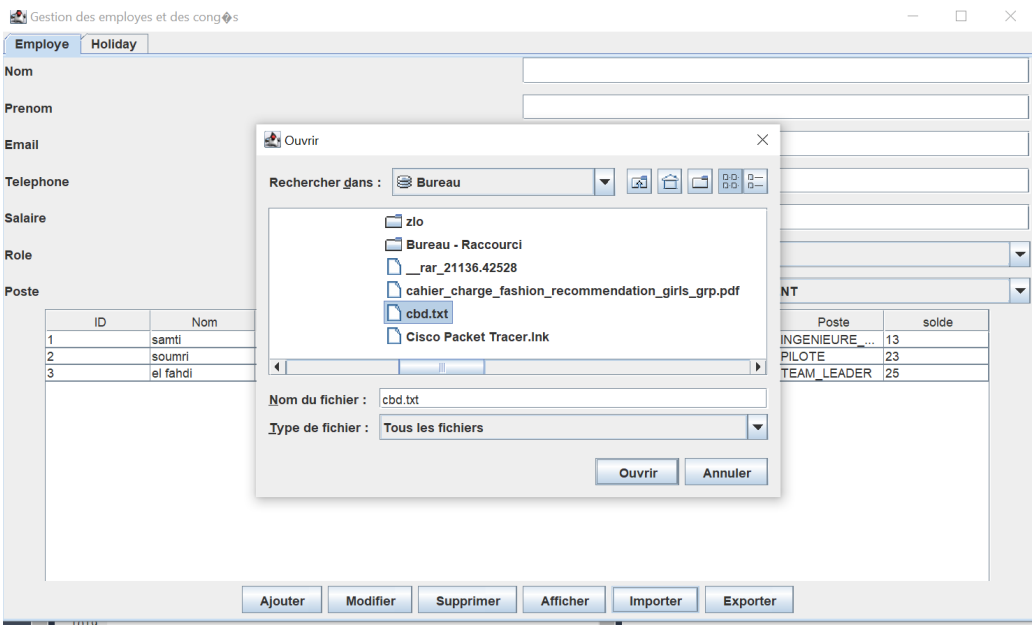

Figure 4: Interface pour l'importation.

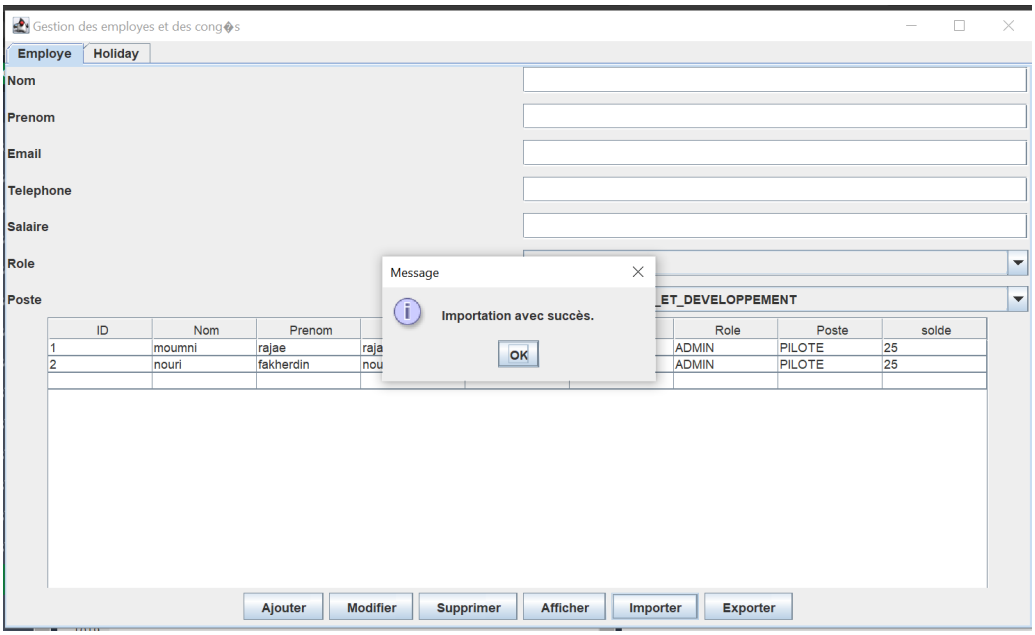

Figure 5: Interface pour l'importation.

Gestion des congés :

Figure 6: Fichier Importé .

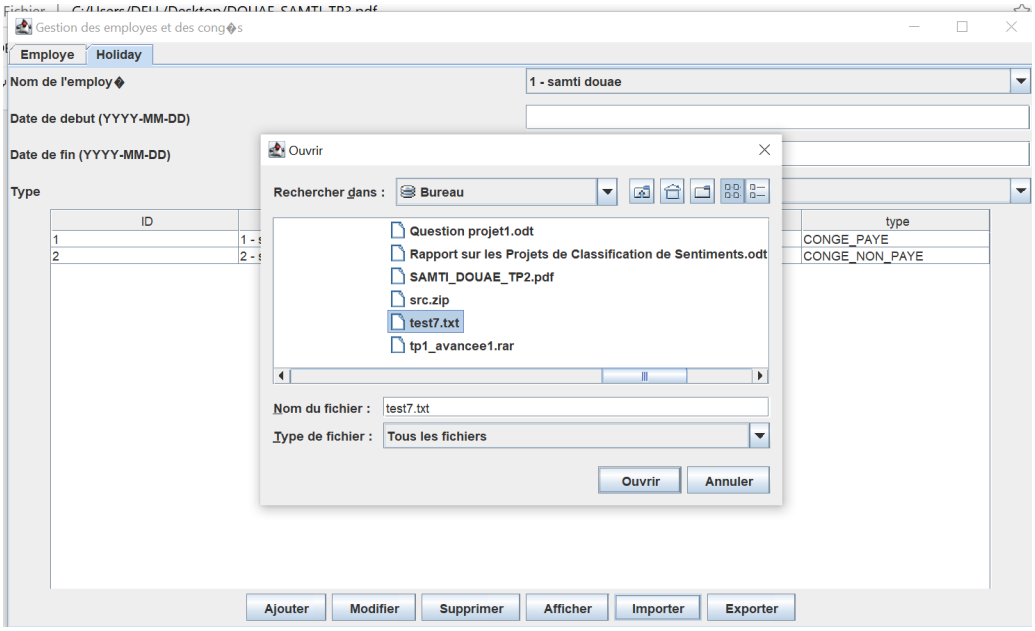

Figure 7: Interface pour l'importation.

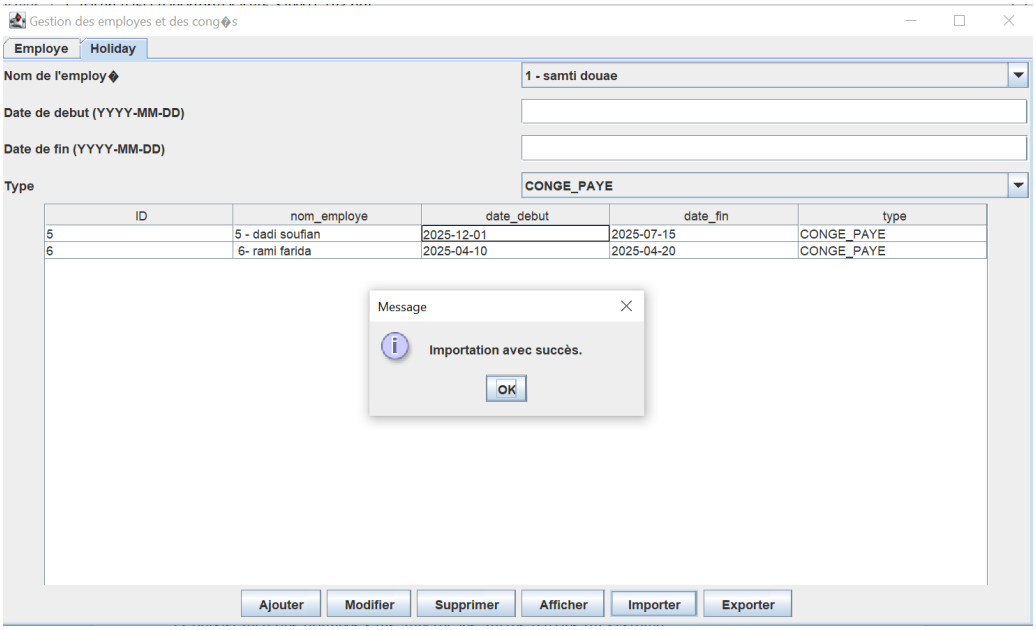

Figure 8: Interface pour l'importation.

6.3 Exportation

Gestion des employés :

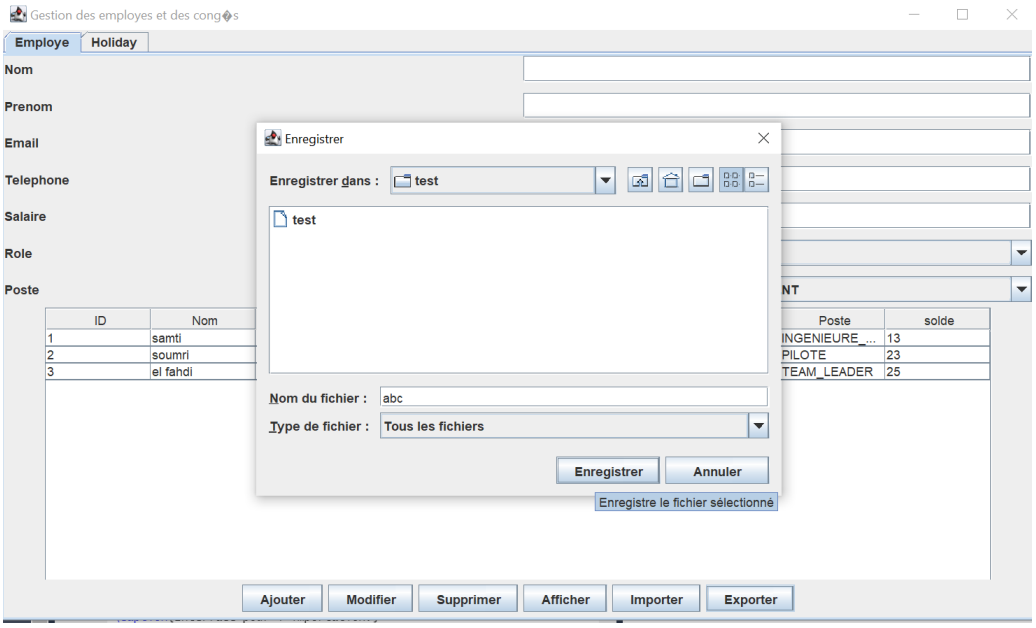

Figure 9: Interface pour l'exportation.

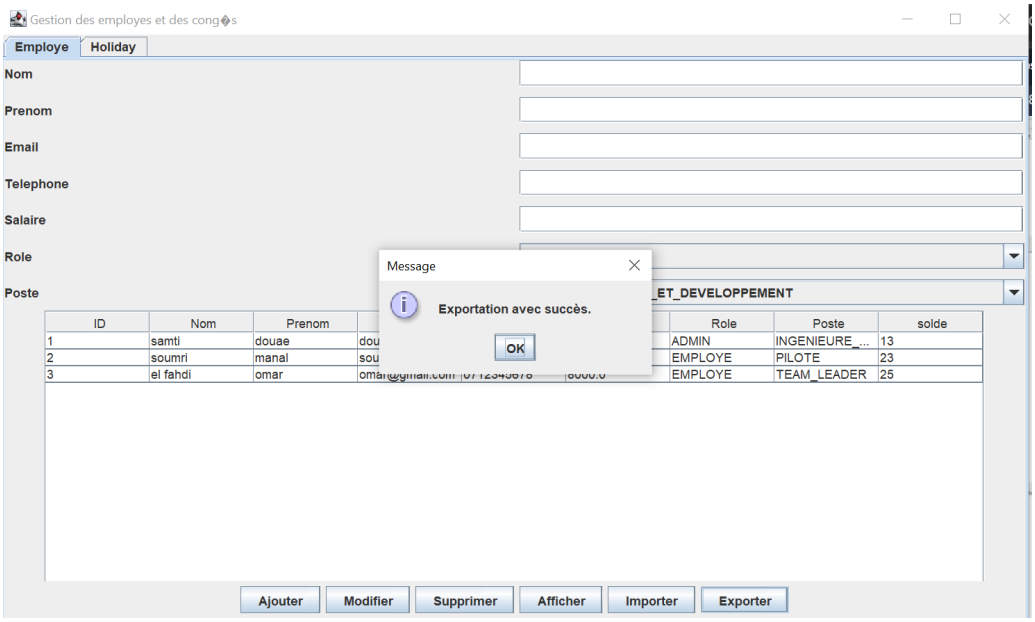

Figure 10: Interface pour l'exportation.

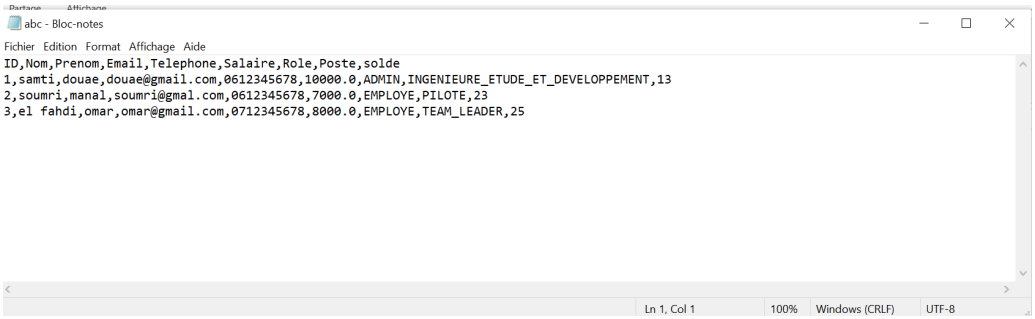

Figure 11: Fichier Exporte.

Gestion des congés :

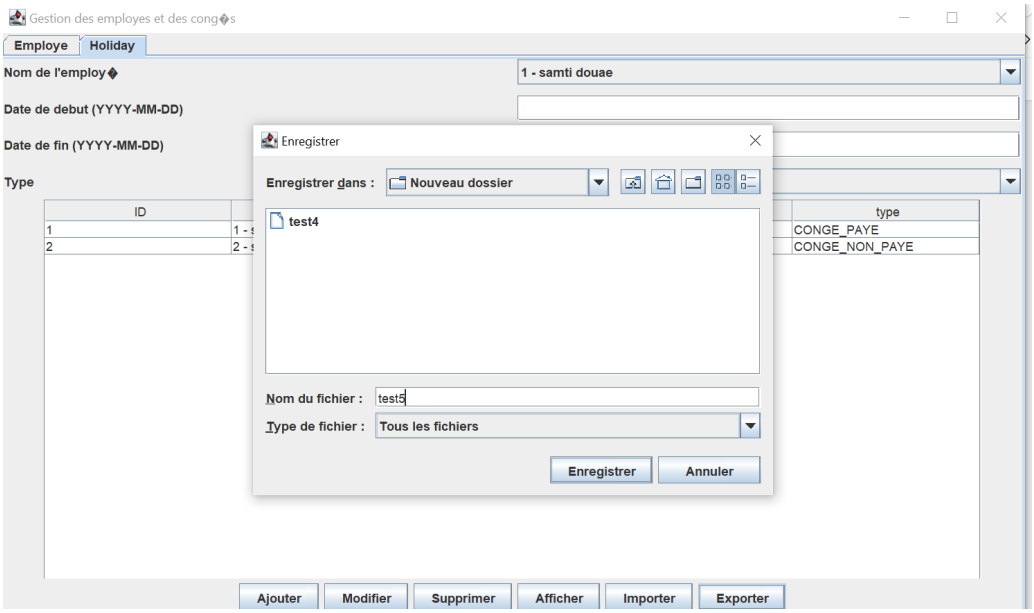

Figure 12: Interface pour l'exportation.

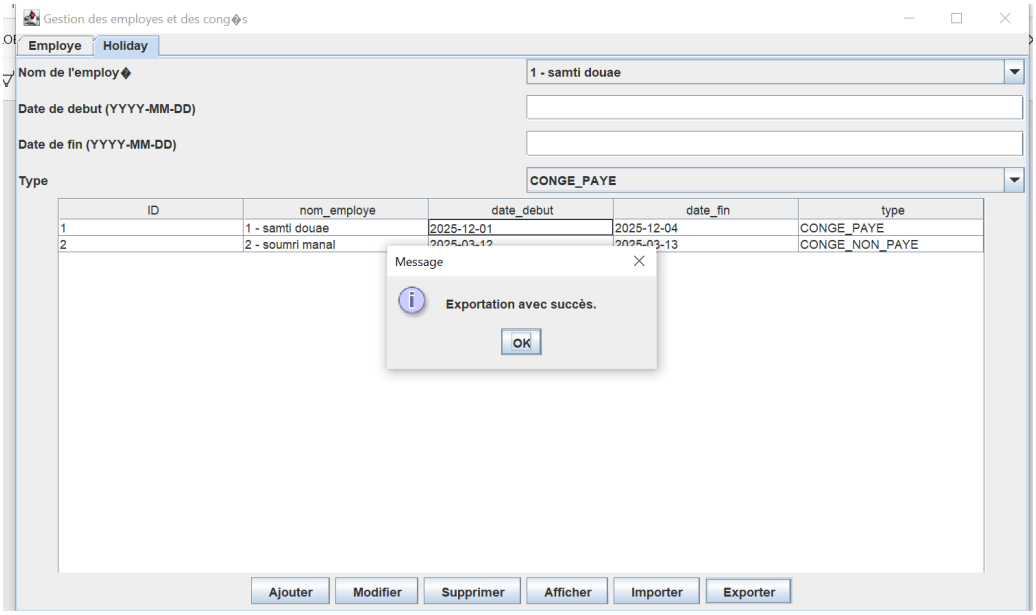

Figure 13: Interface pour l'exportation.

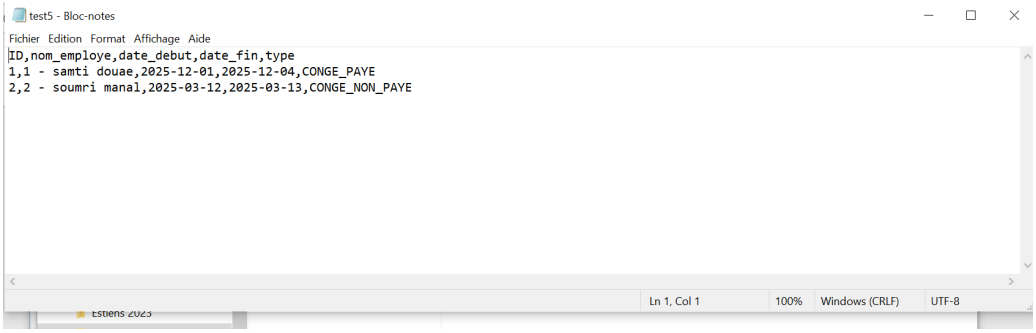

Figure 14: Fichier Exporte.

7 Conclusion

Ce projet a permis d'intégrer plusieurs composants essentiels d'une application Java orientée objet, combinant les modèles, les vues et les contrôleurs selon le paradigme MVC. La classe `Main` joue un rôle central en orchestrant l'interaction entre les différentes couches et en initiant les composants nécessaires pour garantir une navigation fluide dans l'application.

L'ajout des fonctionnalités de connexion a renforcé la sécurité de l'application tout en offrant une expérience utilisateur intuitive. Grâce à la modularité apportée par les DAO, il est désormais possible d'étendre facilement les fonctionnalités ou de modifier l'implémentation de la persistance des données sans affecter les autres parties du système.

En conclusion, ce projet illustre non seulement les concepts fondamentaux de la programmation orientée objet, mais aussi les bonnes pratiques de développement logiciel, telles que la séparation des responsabilités, la réutilisabilité du code, et l'interopérabilité des composants. Les fondations mises en place permettent d'envisager des évolutions futures pour répondre aux besoins croissants des utilisateurs.