

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

Rapport de Tp 3

Gestion de congés(E/S)

Réalisé par : Elgoudimi meryem

Encadré par : Mme.Elkhrof leila

AnnÉE UNIVERSITAIRE : 2024/2025

Table des matières

Introduction	4
Outils & environnement de travail	5
1 Environnement de travail	5
2 Outils de travail	5
3 Language de Programmation	6
1 Réalisation	7
1 Création de la base de donnée	7
1.1 Script base de donnée	7
2 Architecture MVC (Model-View-Controller)	8
2.1 DAO	8
2.2 Model.....	14
2.3 View.....	17
2.4 Controller.....	22
2 Resultat	28
1 L'importation.....	28
2 L'exportation la liste des employés.....	30
3 gestion des fichies au forme txt.....	32
3 Conclusion générale	33
4 Références	34

Table des figures

1	intellij idea logo	5
2	MySQL Workbench logo	5
3	xampp logo	5
4	java developpement kit logo	6
5	java logo	6

Introduction

La gestion des données dans une application repose souvent sur la capacité à échanger des informations avec des systèmes externes. Les opérations d'import et d'export de fichiers jouent un rôle essentiel dans ce contexte, permettant d'intégrer ou de sauvegarder des données en toute simplicité.

Dans ce TP, nous allons aborder la manipulation des fichiers en vue de réaliser des opérations d'importation et d'exportation des données. L'objectif est de permettre au système de traiter des fichiers contenant les informations nécessaires, tels que les données des employés ou des congés, pour les intégrer dans la base de données ou les exporter vers un format lisible et réutilisable.

Ces manipulations impliquent la lecture, l'écriture et la validation des fichiers, tout en assurant une gestion robuste des erreurs. Cela inclut la vérification des droits d'accès, le contrôle du format des fichiers, ainsi que la prise en charge des exceptions pouvant survenir lors de ces opérations.

Outils & environnement de travail

1 Environnement de travail



FIGURE 1 – intellij idea logo

- **IntelliJ idea** : est un environnement de développement intégré (IDE) développé par JetBrains, conçu principalement pour le développement en Java. Reconnu pour ses fonctionnalités intelligentes et sa grande efficacité, il prend également en charge de nombreux autres langages et frameworks comme Kotlin, Groovy, Scala, Python.

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

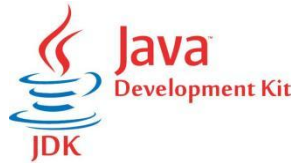


FIGURE 4 – java developpement kit logo

- **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 create database gestion;
2 use gestion;
3 -- Table Employee
4
5 CREATE TABLE `employee` ( `id` int(11) NOT NULL,
6 `nom` varchar(100) NOT NULL,
7 `prenom` varchar(100) NOT NULL,
8 `email` varchar(150) NOT NULL,
9 `telephone` varchar(15) DEFAULT NULL,
10 `salaire` decimal(10,2) NOT NULL,
11 `role` enum('ADMIN','MANAGER','EMPLOYEE') NOT NULL,
12 `poste` enum('DEVELOPER','DESIGNER','MARKETING','OTHER') NOT NULL,
13 `solde` int(11) NOT NULL
14 );
15 -- Table Holiday
16
17
18 CREATE TABLE `holiday` ( `id` int(11) NOT NULL,
19 `id
20 --
21 employee` int(11) NOT NULL,
22 `startdate` date DEFAULT NULL,
23 `enddate` date DEFAULT NULL,
24 `type` enum('ANNUAL','SICK','OTHERS') NOT NULL
25 );
```

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 DAO

Le DAO (Data Access Object) est un modèle de conception (design pattern) utilisé en développement logiciel pour isoler la logique d'accès aux données du reste de l'application. L'objectif principal du DAO est de séparer la couche de logique métier de la couche d'accès aux données, facilitant ainsi la gestion de la persistance des données (par exemple, les opérations CRUD : Création, Lecture, Mise à jour, Suppression).

Étape 1 : Gestion des données DAO

Créer une interface générique pour l'import/export DataImportExport et une implémentation de cette interface par la classe EmployeeDAOImpl.

2.2.1 DataImportExport

```
1 package Model;
2
3 import java.io.IOException;
4 import java.util.List;
5
6 public interface DataImportExport<T>{ no usages
7     void importData(String fileName) throws IOException; no usages
8
9     void exportData(String fileName, List<T> data) throws IOException; no usages
10 }
11
```

2.2.2 EmployeeDAOImpl


```

1 package DAO;
2
3 import Model.Employee;
4 import Model.Post;
5 import Model.Role;
6
7 import java.io.*;
8 import java.sql.*;
9 import java.util.ArrayList;
10 import java.util.List;
11
12 public class EmployeeDAOimpl implements GenericDAOI<Employee> { 15 usages
13     @Override
14     public void add(Employee e) {
15         String sql = "INSERT INTO employe (nom, prenom, email, telephone, salaire, role, poste, solde) VALUES ( ?, ?, ?, ?, ?, ?, ?, ? )";
16         try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(sql)) {
17             stmt.setString( parameterIndex: 1, e.getNom());
18             stmt.setString( parameterIndex: 2, e.getPrenom());
19             stmt.setString( parameterIndex: 3, e.getEmail());
20             stmt.setString( parameterIndex: 4, e.getTelephone());
21             stmt.setDouble( parameterIndex: 5, e.getSalaire());
22             stmt.setString( parameterIndex: 6, e.getRole().name());
23             stmt.setString( parameterIndex: 7, e.getPost().name());
24             stmt.setInt( parameterIndex: 8, e.getSolde());
25             stmt.executeUpdate();

```

```

12 public class EmployeeDAOimpl implements GenericDAOI<Employee> { 15 usages
14     public void add(Employee e) {
15         stmt.executeUpdate();
16     } catch (SQLException exception) {
17         System.err.println("Failed to add employee: " + exception.getMessage());
18         exception.printStackTrace();
19     } catch (ClassNotFoundException ex) {
20         System.err.println("Failed to connect to the database: " + ex.getMessage());
21         ex.printStackTrace();
22     }
23 }
24
25
26
27
28
29
30
31
32
33
34
35 ⚡ @Override 2 usages
36 public void delete(int id) {
37     String sql = "DELETE FROM employe WHERE id = ?";
38     try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(sql)) {
39         stmt.setInt( parameterIndex: 1, id);
40         stmt.executeUpdate();
41     } catch (SQLException exception) {
42         System.err.println("Failed to delete employee: " + exception.getMessage());
43     } catch (ClassNotFoundException ex) {
44         System.err.println("Failed to connect to the database: " + ex.getMessage());
45     }
46 }
47

```

@Override 2 usages

```

47
48
49 @Override
50 public void update(Employe e) {
51     String sql = "UPDATE employe SET nom = ?, prenom = ?, email = ?, telephone = ?, salaire = ?, role = ?, poste = ?, solde = ? WHERE id = ?";
52     try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(sql)) {
53         stmt.setString(1, e.getNom());
54         stmt.setString(2, e.getPrenom());
55         stmt.setString(3, e.getEmail());
56         stmt.setString(4, e.getTelephone());
57         stmt.setDouble(5, e.getSalaire());
58         stmt.setString(6, e.getRole().name());
59         stmt.setString(7, e.getPost().name());
60         stmt.setInt(8, e.getSolde());
61         stmt.setInt(9, e.getId());
62         stmt.executeUpdate();
63     } catch (SQLException exception) {
64         System.err.println("Failed to update employee: " + exception.getMessage());
65     } catch (ClassNotFoundException ex) {
66         System.err.println("Failed to connect to the database: " + ex.getMessage());
67     }
68 }

```

@Override 4 usages

```

69
70 @Override
71 public List<Employe> display() {
72     String sql = "SELECT * FROM employe";
73     List<Employe> employes = new ArrayList<>();
74     try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(sql)) {
75         ResultSet rs = stmt.executeQuery();
76         while (rs.next()) {
77             int id = rs.getInt("id");
78             String nom = rs.getString("nom");
79             String prenom = rs.getString("prenom");
80             String email = rs.getString("email");
81             String telephone = rs.getString("telephone");
82             double salaire = rs.getDouble("salaire");
83             String role = rs.getString("role");
84             String poste = rs.getString("poste");
85             int solde = rs.getInt("solde");
86             Employe e = new Employe(id, nom, prenom, email, telephone, salaire, Role.valueOf(role), Post.valueOf(poste), solde);
87             employes.add(e);
88         }
89     } catch (ClassNotFoundException ex) {
90         System.err.println("Failed to connect to the database: " + ex.getMessage());
91     } catch (SQLException ex) {
92     }
93 }

```

```

89         System.err.println("Failed to connect to the database: " + ex.getMessage());
90     } catch (SQLException ex) {
91         System.err.println("Failed to retrieve employees: " + ex.getMessage());
92     }
93     return employees;
94 }
95
96 public void updateSolde(int id, int solde) { 1 usage
97     String sql = "UPDATE employe SET solde = ? WHERE id = ?";
98     try (PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(sql)) {
99         stmt.setInt( parameterIndex: 1, solde);
100        stmt.setInt( parameterIndex: 2, id);
101        stmt.executeUpdate();
102    } catch (SQLException exception) {
103        System.err.println("Failed to update employee balance: " + exception.getMessage());
104    } catch (ClassNotFoundException ex) {
105        System.err.println("Failed to connect to the database: " + ex.getMessage());
106    }
107 }
108

```

```

109 public void importData(String filePath) { no usages
110     String sql = "INSERT INTO employe (nom, prenom, email, telephone, salaire, role, poste, solde) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
111     try (BufferedReader reader = new BufferedReader(new FileReader(filePath));
112         PreparedStatement stmt = DBConnexion.getConnexion().prepareStatement(sql)) {
113
114         String line = reader.readLine(); // Skip header line
115         while ((line = reader.readLine()) != null) {
116             String[] data = line.split( regex: "," );
117             if (data.length == 8) {
118                 stmt.setString( parameterIndex: 1, data[0].trim());
119                 stmt.setString( parameterIndex: 2, data[1].trim());
120                 stmt.setString( parameterIndex: 3, data[2].trim());
121                 stmt.setString( parameterIndex: 4, data[3].trim());
122                 stmt.setDouble( parameterIndex: 5, Double.parseDouble(data[4].trim()));
123                 stmt.setString( parameterIndex: 6, data[5].trim());
124                 stmt.setString( parameterIndex: 7, data[6].trim());
125                 stmt.setInt( parameterIndex: 8, Integer.parseInt(data[7].trim()));
126                 stmt.addBatch();
127             }
128         }
129         stmt.executeBatch();
130         System.out.println("Employees imported successfully!");
131     }

```



```

132     } catch (SQLException | IOException e) {
133         System.err.println("Failed to import data: " + e.getMessage());
134         e.printStackTrace();
135     } catch (ClassNotFoundException e) {
136         System.err.println("Failed to connect to the database: " + e.getMessage());
137     }
138 }
139
140 @ public void exportData(String fileName, List<Employee> data) { 1 usage
141     try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {
142         writer.write( str: "nom,prenom,email,telephone,salaire,role,poste,solde");
143         writer.newLine();
144         for (Employee employee : data) {
145             String line = String.format("%s,%s,%s,%s,%.2f,%s,%s,%d",
146                 employee.getNom(),
147                 employee.getPrenom(),
148                 employee.getEmail(),
149                 employee.getTelephone(),
150                 employee.getSalaire(),
151                 employee.getRole().name(),
152                 employee.getPost().name(),
153                 employee.getSolde());
154
155             employee.getPost().name(),
156             employee.getSolde());
157             writer.write(line);
158             writer.newLine();
159         }
160         System.out.println("Data exported successfully to " + fileName);
161     } catch (IOException exception) {
162         System.err.println("Failed to export data: " + exception.getMessage());
163         exception.printStackTrace();
164     }
165 }

```

2.2 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.

— Étape 2 : Logique métier

- Extension de la classe modèle (**EmployeeModel**) pour gérer l'import/export.
- Vérification de l'existence du fichier
 - On vérifie que le fichier n'est pas nul et qu'il existe dans le système.
 - Méthode utilisée : `file.exists()`
 - En cas d'absence du fichier, une exception est levée avec un message approprié.
- Vérification du type de fichier
 - On s'assure que le chemin indique bien un fichier.

- Méthode utilisée : `file.isFile()`
- **Vérification des droits de lecture**
 - On vérifie que l'application a les droits nécessaires pour lire le fichier.
 - Méthode utilisée : `file.canRead()`

2.1.1 EmployeModel

```
1 package Model;
2
3 import DAO.EmployeDAOimpl;
4 import java.io.BufferedReader;
5 import java.io.File;
6 import java.io.FileReader;
7 import java.io.IOException;
8 import java.util.List;
9
10 public class EmployeModel { 12 usages
11     private EmployeDAOimpl dao; 8 usages
12
13     // Constructeur
14     public EmployeModel(EmployeDAOimpl dao) { 7 usages
15         this.dao = dao;
16     }
17
18     // Fonction pour ajouter un employé
19     public boolean addEmploye(int id, String nom, String prenom, String email, String telephone, double salaire, Role role, Post post, int sold
20         if (salaire < 0) {
21             System.out.println("Erreur : le salaire doit être positif.");
22             return false;
23         }
24         if (id < 0) {
25             System.out.println("Erreur : l'id doit être positif.");
```

```
26         if (id < 0) {
27             System.out.println("Erreur : l'id doit être positif.");
28             return false;
29         }
30         if (telephone.length() != 10) {
31             System.out.println("Erreur : le téléphone doit être de 10 chiffres.");
32             return false;
33         }
34         if (!email.contains("@")) {
35             System.out.println("Erreur : le mail doit contenir un @.");
36             return false;
37         }
38         Employe e = new Employe(id, nom, prenom, email, telephone, salaire, role, post, solde);
39         dao.add(e);
40         return true;
41     }
42
43     // Fonction pour supprimer un employé
44     public boolean deleteEmploye(int id) { 1 usage
45         dao.delete(id);
46         return true;
47     }
48 }
```

```
46
47 // Fonction pour mettre à jour un employé
48 public boolean updateEmploye(int id, String nom, String prenom, String email, String telephone, double salaire, Role role, Post post, int solde) {
49     Employe e = new Employe(id, nom, prenom, email, telephone, salaire, role, post, solde);
50     dao.update(e);
51     return true;
52 }
53
54 // Fonction pour mettre à jour le solde d'un employé
55 public boolean updateSolde(int id, int solde) { 1 usage
56     dao.updateSolde(id, solde);
57     return true;
58 }
59
60 // Fonction pour afficher la liste des employés
61 public List<Employe> displayEmploye() { 8 usages
62     return dao.display();
63 }
64
65 // Fonction pour importer les employés depuis un fichier CSV
66 public void importData(String filePath) { 1 usage
67     File file = new File(filePath);
68
```

```
68
69 // Vérification de la validité du fichier
70 checkFileExists(file);
71 checkIsFile(file);
72 checkIsReadable(file);
73
74 try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
75     String line;
76     while ((line = reader.readLine()) != null) {
77         // Supposons que le format CSV soit : id, nom, prenom, email, telephone, salaire, role, poste, solde
78         String[] data = line.split(regex: ",");
79         if (data.length == 9) { // Vérification du nombre correct de champs
80             try {
81                 int id = Integer.parseInt(data[0].trim());
82                 String nom = data[1].trim();
83                 String prenom = data[2].trim();
84                 String email = data[3].trim();
85                 String telephone = data[4].trim();
86                 double salaire = Double.parseDouble(data[5].trim());
87                 String roleString = data[6].trim();
88                 String posteString = data[7].trim();
89                 int solde = Integer.parseInt(data[8].trim());
90
```

```

90
91 // Conversion de role et poste en enums
92 Role role = Role.valueOf(roleString.toUpperCase());
93 Post poste = Post.valueOf(posteString.toUpperCase());
94
95 // Création et ajout de l'employé
96 Employe employe = new Employe(id, nom, prenom, email, telephone, salaire, role, poste, solde);
97 dao.add(employe);
98 } catch (NumberFormatException e) {
99     System.out.println("Erreur de format dans les données du fichier.");
100 } catch (IllegalArgumentException e) {
101     System.out.println("Erreur de valeur d'enum dans le fichier: " + e.getMessage());
102 }
103 }
104 }
105 } catch (IOException e) {
106     System.out.println("Erreur lors de la lecture du fichier: " + e.getMessage());
107 }
108 }
109

```

```

109
110 // Fonction de validation du fichier : vérifie qu'il existe
111 @ ~ private boolean checkFileExists(File file) { 1 usage
112 ~     if (!file.exists()) {
113         throw new IllegalArgumentException("Le fichier n'existe pas : " + file.getPath());
114     }
115     return true;
116 }
117
118 // Vérifie que le chemin correspond bien à un fichier
119 @ ~ private boolean checkIsFile(File file) { 2 usages
120 ~     if (!file.isFile()) {
121         throw new IllegalArgumentException("Le chemin spécifié n'est pas un fichier : " + file.getPath());
122     }
123     return true;
124 }
125
126 // Vérifie que le fichier est lisible
127 @ ~ private boolean checkIsReadable(File file) { 2 usages
128 ~     if (!file.canRead()) {
129         throw new IllegalArgumentException("Le fichier n'est pas lisible : " + file.getPath());
130     }
131     return true;
132 }

```

```
126 // Vérifie que le fichier est lisible
127 @ private boolean checkIsReadable(File file) { 2 usages
128     if (!file.canRead()) {
129         throw new IllegalArgumentException("Le fichier n'est pas lisible : " + file.getPath());
130     }
131     return true;
132 }
133 public void exportData(String FileName , List<Employe> data) throws IOException { 1 usage
134     File file = new File(FileName);
135     checkIsReadable(file);
136     checkIsFile(file);
137     dao.exportData(FileName,data);
138 }
139 }
140
```

2.3 View

La Vue (View) dans l'architecture MVC (Model-View-Controller) est responsable de la présentation des données à l'utilisateur. Elle se charge d'afficher les informations contenues dans le modèle (par exemple, une liste de livres ou des détails d'un employé) sous une forme compréhensible et interactive. La vue ne contient pas de logique métier; elle se contente de recevoir les données et de les afficher de manière appropriée à l'utilisateur.

Étape 4 : Couche Vue - Modification de l'interface graphique

Pour créer une interface utilisateur simple et efficace, les étapes suivantes sont réalisées :

Ajout des boutons et champs nécessaires :

Un bouton d'importation pour charger les données.

Un bouton d'exportation pour sauvegarder les données.

Organisation de l'interface :


Utilisation d'un agencement naturel en appliquant le FlowLayout.

Disposition des éléments de manière intuitive pour améliorer l'expérience utilisateur.

2.3.1 Employe_HolidayView


```
1  package View;
2  import DAO.EmployeDAOimpl;
3  import Model.Employe;
4  import Model.EmployeModel;
5  import Model.Post;
6  import Model.Role;
7  import Model.Type_holiday;
8  import java.awt.*;
9  import javax.swing.*;
10 import javax.swing.table.DefaultTableModel;
11 import java.util.List;
12 public class Employe_HolidayView extends JFrame { 31 usages
13     // le tableau de employe et congé
14     private JTabbedPane tabbedPane = new JTabbedPane(); 3 usages
15     // les tabs
16     private JPanel employeTab = new JPanel(); 3 usages
17     private JPanel holidayTab = new JPanel(); 3 usages
18     // les panels
19     private JPanel Employepan = new JPanel(); 5 usages
20     private JPanel Holidaypan = new JPanel(); 5 usages
21     private JPanel Display_Table_employe = new JPanel(); 3 usages
22     private JPanel Display_Table_holiday = new JPanel(); 2 usages
23     private final JPanel Forme_employe = new JPanel(); 16 usages
24     private final JPanel Forme_holiday = new JPanel(); 10 usages
25     private JPanel panButton_employe = new JPanel(); 7 usages
```

```

49 // les textfield du congé
50 private JComboBox<String> text_employe = new JComboBox<>(); 8 usages
51 private JTextField text_startDate = new JTextField(""); 5 usages
52 private JTextField text_endDate = new JTextField(""); 5 usages
53 // les boutons du l'employe
54 private JButton addButton_employe = new JButton(text: "Ajouter"); 2 usages
55 private JButton updateButton_employe = new JButton(text: "Modifier"); 2 usages
56 private JButton deleteButton_employe = new JButton(text: "Supprimer"); 2 usages
57 private JButton displayButton_employe = new JButton(text: "Afficher"); 2 usages
58 //-----
59  private JButton importer = new JButton(text: "importer"); 2 usages
60 private JButton exporter = new JButton(text: "exporter"); 2 usages
61
62
63 // les boutons du congé
64 private JButton addButton_holiday = new JButton(text: "Ajouter"); 2 usages
65 private JButton updateButton_holiday = new JButton(text: "Modifier"); 2 usages
66 private JButton deleteButton_holiday = new JButton(text: "Supprimer"); 2 usages
67 private JButton displayButton_holiday = new JButton(text: "Afficher"); 2 usages
68

```

```

69 // le tableau de l'employe
70 JPanel pan0 = new JPanel(new BorderLayout()); 2 usages
71 public static String[] columnNames_employe = {"ID", "Nom", "Prenom", "Email", "Téléphone", "Salaire", "Role", "Poste", "solde"}; 1 usage
72 public static DefaultTableModel tableModel = new DefaultTableModel(columnNames_employe, rowCount: 0); 1 usage
73 public static JTable Tableau = new JTable(tableModel); 17 usages
74 // le tableau du congé
75 JPanel pan1 = new JPanel(new BorderLayout()); 2 usages
76 public static String[] columnNames_holiday = {"ID", "nom_employe", "date_debut", "date_fin", "type"}; 1 usage
77 public static DefaultTableModel tableModel1 = new DefaultTableModel(columnNames_holiday, rowCount: 0); 1 usage
78 public static JTable Tableau1 = new JTable(tableModel1); 14 usages
79  public Employe_HolidayView() { 1 usage
80     setTitle("Gestion des employes et des congés");
81     setSize(width: 1000, height: 600);
82     setDefaultCloseOperation(EXIT_ON_CLOSE);
83     setLocationRelativeTo(null);
84     add(tabbedPane);
85     // Employe Tab
86     employeTab.setLayout(new BorderLayout());
87     employeTab.add(Employepan, BorderLayout.CENTER);
88
89     Employepan.setLayout(new BorderLayout());
90     Employepan.add(Display_Table_employe, BorderLayout.CENTER);
91     Tableau.setFillViewportHeight(true);

```

```

90     Employepan.add(Display_Table_employe, BorderLayout.CENTER);
91     Tableaui.setFillsViewportHeight(true);
92     Dimension preferredSize = new Dimension( width: 900, height: 500);
93     Tableaui.setPreferredScrollableViewportSize(preferredSize);
94     pan0.add(new JScrollPane(Tableaui), BorderLayout.CENTER);
95     Display_Table_employe.add(pan0);
96     Employepan.add(panButton_employe, BorderLayout.SOUTH);
97     panButton_employe.add(addButton_employe);
98     panButton_employe.add(updateButton_employe);
99     panButton_employe.add(deleteButton_employe);
100    panButton_employe.add(displayButton_employe);
101
102    panButton_employe.add(importer);
103    panButton_employe.add(exporter);
104
105    Employepan.add(Forme_employe, BorderLayout.NORTH);
106    Forme_employe.setLayout(new GridLayout( rows: 7, cols: 2, hgap: 10, vgap: 10));
107    Forme_employe.add(label_nom);
108    Forme_employe.add(text_nom);
109    Forme_employe.add(label_prenom);
110    Forme_employe.add(text_prenom);
111    Forme_employe.add(label_email);
112    Forme_employe.add(text_email);

```

```

112    Forme_employe.add(text_email);
113    Forme_employe.add(label_tele);
114    Forme_employe.add(text_tele);
115    Forme_employe.add(label_salaire);
116    Forme_employe.add(text_salaire);
117    Forme_employe.add(label_role);
118    Forme_employe.add(roleComboBox);
119    Forme_employe.add(label_poste);
120    Forme_employe.add(posteComboBox);
121    // Holiday Tab
122    holidayTab.setLayout(new BorderLayout());
123    holidayTab.add(Holidaypan, BorderLayout.CENTER);
124    Holidaypan.setLayout(new BorderLayout());
125    Holidaypan.add(Display_Table_holiday, BorderLayout.CENTER);
126    Tableaui1.setFillsViewportHeight(true);
127    Tableaui1.setPreferredScrollableViewportSize(preferredSize);
128    pan1.add(new JScrollPane(Tableaui1), BorderLayout.CENTER);
129    Display_Table_holiday.add(pan1);
130    Holidaypan.add(Forme_holiday, BorderLayout.NORTH);
131    Forme_holiday.setLayout(new GridLayout( rows: 4, cols: 2, hgap: 10, vgap: 10));
132    Forme_holiday.add(label_employe);
133    Forme_holiday.add(text_employe);
134    Forme_holiday.add(label_startDate);

```



```

133     Forme_holiday.add(text_employe);
134     Forme_holiday.add(label_startDate);
135     Forme_holiday.add(text_startDate);
136     Forme_holiday.add(label_endDate);
137     Forme_holiday.add(text_endDate);
138     Forme_holiday.add(label_type);
139     Forme_holiday.add(TypeComboBox);
140     Holidaypan.add(panButton_holiday, BorderLayout.SOUTH);
141     panButton_holiday.add(addButton_holiday);
142     panButton_holiday.add(updateButton_holiday);
143     panButton_holiday.add(deleteButton_holiday);
144     panButton_holiday.add(displayButton_holiday);
145     // TabbedPane
146     tabbedPane.addTab( title: "Employe", employeTab);
147     tabbedPane.addTab( title: "Holiday", holidayTab);
148
149     remplace_les_employes();
150     setVisible(true);
151 }
152 public void remplace_les_employes () { 3 usages
153     List<Employe> Employes = new EmployeeModel(new EmployeeDAOimpl()).displayEmployee();
154     text_employe.removeAllItems();
155     for (Employe elem : Employes) {

```

```

154     text_employe.removeAllItems();
155     for (Employee elem : Employes) {
156         text_employe.addItem(elem.getId() + " - " + elem.getNom()+" "+elem.getPrenom());
157     }
158 }
159 // getters
160 public int getId_employe() { 2 usages
161     return Integer.parseInt(text_employe.getSelectedItem().toString().split(regex: " - ")[0]);
162 }
163 public String getNom() { 3 usages
164     return text_nom.getText();
165 }
166 public JTable getTable() { no usages
167     return (JTable) Display_Table_employe.getComponent(n: 0);
168 }
169 public String getPrenom() { 2 usages
170     return text_prenom.getText();
171 }
172 public String getEmail() { 2 usages
173     return text_email.getText();
174 }
175 public String getTelephone() { 2 usages
176     return text_tele.getText();
177 }

```

```

177     }
178     public double getSalaire() { 2 usages
179         return Double.parseDouble(text_salaire.getText());
180     }
181     public Role getRole() { 2 usages
182         return (Role) roleComboBox.getSelectedItem();
183     }
184     public Post getPoste() { 2 usages
185         return (Post) posteComboBox.getSelectedItem();
186     }
187     public JButton getaddButton_employe () { 1 usage
188         return addButton_employe;
189     }
190     public JButton getupdateButton_employe () { 1 usage
191         return updateButton_employe;
192     }
193     public JButton getdeleteButton_employe () { 1 usage
194         return deleteButton_employe;
195     }
196     public JButton getdisplayButton_employe () { 1 usage
197         return displayButton_employe;
198     }
199     public JButton getimporter () { 1 usage
200         return importer;
201     }

```

```

202
203  ✓ public JButton getexporter () { 1 usage
204      return exporter;
205  }
206  ✓ public JButton getaddButton_holiday () { 1 usage
207      return addButton_holiday;
208  }
209  ✓ public JButton getupdateButton_holiday () { 1 usage
210      return updateButton_holiday;
211  }
212  ✓ public JButton getdeleteButton_holiday () { 1 usage
213      return deleteButton_holiday;
214  }
215  ✓ public JButton getDisplayButton_holiday () { 1 usage
216      return displayButton_holiday;
217  }
218  ✓ public String getStartDate () { 2 usages
219      return text_startDate.getText();
220  }
221  ✓ public String getEndDate() { 2 usages
222      return text_endDate.getText();
223  }
224  ✓ public Type_holiday getType_holiday() { 2 usages
225      return (Type_holiday) TypeComboBox.getSelectedItem();

```

```

226  }
227  // methods d'affichage des messages
228  public void afficherMessageErreur(String message) { 21 usages
229      JOptionPane.showMessageDialog( parentComponent: this, message, title: "Erreur", JOptionPane.ERROR_MESSAGE);
230  }
231  public void afficherMessageSucces(String message) { 8 usages
232      JOptionPane.showMessageDialog( parentComponent: this, message, title: "Succès", JOptionPane.INFORMATION_MESSAGE);
233  }
234
235  // methodes de vider les champs
236  public void viderChamps_em() { 2 usages
237      text_nom.setText("");
238      text_prenom.setText("");
239      text_email.setText("");
240      text_tele.setText("");
241      text_salaire.setText("");
242      roleComboBox.setSelectedIndex(0);
243      posteComboBox.setSelectedIndex(0);
244  }
245  public void viderChamps_ho() { 2 usages
246      text_startDate.setText("");
247      text_endDate.setText("");
248      TypeComboBox.setSelectedIndex(0);
249  }

```

```

249     }
250     // methodes de remplir les champs
251     public void remplaireChamps_em (int id, String nom, String prenom, String email, String telephone, double salaire, Role role, Post poste) {
252         text_nom.setText(nom);
253         text_prenom.setText(prenom);
254         text_email.setText(email);
255         text_tele.setText(telephone);
256         text_salaire.setText(String.valueOf(salaire));
257         roleComboBox.setSelectedItem(role);
258         posteComboBox.setSelectedItem(poste);
259     }
260     public void remplaireChamps_ho(int id_employe, String date_debut, String date_fin, Type_holiday type) { 1 usage
261         List<Employe> Employes = new EmployeModel(new EmployeDAOimpl()).displayEmploye();
262         text_employe.removeAllItems();
263         for (Employe elem : Employes) {
264             if (elem.getId() == id_employe) {
265                 text_employe.addItem(elem.getId() + " - " + elem.getNom()+" "+elem.getPrenom());
266                 text_employe.setSelectedItem(elem.getId() + " - " + elem.getNom()+" "+elem.getPrenom());
267             }
268         }
269         text_startDate.setText(date_debut);
270         text_endDate.setText(date_fin);
271         TypeComboBox.setSelectedItem(type);
272     }
273     // methodes de test des champs
274     public boolean testChampsVide_em () { no usages
275         return text_nom.getText().equals("") || text_prenom.getText().equals("") || text_email.getText().equals("") || text_tele.getText().equals("") || text_salaire.getText().equals("") || roleComboBox.getSelectedItem().equals("") || posteComboBox.getSelectedItem().equals("");
276     }
277     public boolean testChampsVide_ho () { no usages
278         return text_employe.getSelectedItem().equals("") || text_startDate.getText().equals("") || text_endDate.getText().equals("") || TypeComboBox.getSelectedItem().equals("");
279     }
280 }

```

2.4 Controller

Le Controller dans le contexte de l'architecture MVC (Model-View-Controller) joue un rôle clé en tant que médiateur entre la Vue (interface utilisateur) et le Modèle (logique métier et gestion des données). Il est responsable de la gestion des entrées utilisateur, de la logique de contrôle et de la coordination entre le modèle et la vue. Le controller reçoit les actions de l'utilisateur (comme un clic sur un bouton ou la soumission d'un formulaire), traite ces actions (en interagissant avec le modèle si nécessaire) et met à jour la vue.

2.2.5 EmployeeController


```

1  package controller;
2
3  import Model.*;
4  import View.*;
5  import javax.swing.JFileChooser;
6  import javax.swing.filechooser.FileNameExtensionFilter;
7  import DAO.EmployeeDAOimpl;
8  import java.util.Calendar;
9  import Model.EmployeeModel;
10 import java.util.List;
11 import javax.swing.table.DefaultTableModel;
12
13 public class EmployeeController { 5 usages
14     private final Employee_HolidayView View; 41 usages
15     public static EmployeeModel model_employe; 9 usages
16     public static int id = 0;
17     public static int oldselectedrow = -1; no usages
18     public static boolean test = false; 3 usages
19     String nom = ""; 4 usages
20     String prenom = ""; 4 usages
21     String email = ""; 4 usages
22     String telephone = ""; 4 usages
23     double salaire = 0; 4 usages
24     Role role = null; 4 usages
25     Post poste = null; 4 usages
26     int solde = 0; 2 usages

```

```

25 Post poste = null; 4 usages
26 int solde = 0; 2 usages
27 boolean updatereussi = false; no usages
28
29 public EmployeeController(Employee_HolidayView view, EmployeeModel model) { 1 usage
30     this.View = view;
31     this.model_employe = model;
32     View.getaddButton_employe().addActionListener( ActionEvent e -> addEmployee());
33     View.getdeleteButton_employe().addActionListener( ActionEvent e -> deleteEmployee());
34     View.getupdateButton_employe().addActionListener( ActionEvent e -> updateEmployee());
35     View.getdisplayButton_employe().addActionListener( ActionEvent e -> displayEmployee());
36
37     View.getimporter().addActionListener( ActionEvent e -> handleImport());
38
39     View.getexporter().addActionListener( ActionEvent e -> handleExport());
40
41
42     Employee_HolidayView.Tableau.getSelectionModel().addListSelectionListener( ListSelectionEvent e -> updateEmployeebyselect()
43 }
44
45 public void displayEmployee() { 4 usages
46     List<Employee> Employes = model_employe.displayEmployee();
47     if (Employes.isEmpty()) {
48         View.afficherMessageErreur("Aucun employe.");
49     }

```

```

50     DefaultTableModel tableModel = (DefaultTableModel) Employee_HolidayView.Tableau.getModel();
51     tableModel.setRowCount(0);
52     for (Employee e : Employes) {
53         tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.getEmail(), e.getTelephone(), e.getSalaire(), e.getRole(), e.get
54     }
55     View.remplaire_les_employes();
56 }
57
58 // Function to add an employee
59 private void addEmployee() { 1 usage
60     String nom = View.getNom();
61     String prenom = View.getPrenom();
62     String email = View.getEmail();
63     String telephone = View.getTelephone();
64     double salaire = View.getSalaire();
65     Role role = View.getRole();
66     Post poste = View.getPoste();
67     View.viderChamps_em();
68     boolean addeussi = model_employe.addEmployee(0, nom, prenom, email, telephone, salaire, role, poste, solde: 25);
69     if (addeussi) {
70         View.afficherMessageSucces("L'employe a bien ete ajoutez.");
71         displayEmployee();
72     } else {

```

```

71     displayEmployee(),
72 } else {
73     View.afficherMessageErreur("L'employe n'a pas ete ajoutee.");
74 }
75 }
76
77 // Function to delete an employee
78 private void deleteEmployee() { 1 usage
79     int selectedrow = Employee_HolidayView.Tableau.getSelectedRow();
80     if (selectedrow == -1) {
81         View.afficherMessageErreur("Veuillez selectionner une ligne.");
82     } else {
83         int id = (int) Employee_HolidayView.Tableau.getValueAt(selectedrow, column: 0);
84         if (model_employe.deleteEmployee(id)) {
85             View.afficherMessageSucces("L'employe a bien ete supprimer.");
86             displayEmployee();
87         } else {
88             View.afficherMessageErreur("L'employe n'a pas ete supprimer.");
89         }
90     }
91 }
92

```

```

93 // Function to update employee details by selecting a row
94 private void updateEmployeebyselect() { 1 usage
95     int selectedrow = Employee_HolidayView.Tableau.getSelectedRow();
96     if (selectedrow == -1) {
97         return;
98     }
99     try {
100         id = (int) Employee_HolidayView.Tableau.getValueAt(selectedrow, column: 0);
101         nom = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow, column: 1);
102         prenom = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow, column: 2);
103         email = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow, column: 3);
104         telephone = (String) Employee_HolidayView.Tableau.getValueAt(selectedrow, column: 4);
105         salaire = (double) Employee_HolidayView.Tableau.getValueAt(selectedrow, column: 5);
106         role = (Role) Employee_HolidayView.Tableau.getValueAt(selectedrow, column: 6);
107         poste = (Post) Employee_HolidayView.Tableau.getValueAt(selectedrow, column: 7);
108         solde = (int) Employee_HolidayView.Tableau.getValueAt(selectedrow, column: 8);
109         View.remplaireChamps_em(id, nom, prenom, email, telephone, salaire, role, poste);
110         test = true;
111     } catch (Exception e) {
112         View.afficherMessageErreur("Erreur lors de la récupération des données");
113     }
114 }
115

```

```

116 // Function to update an employee's details
117 private void updateEmployee() { 1 usage
118     if (!test) {
119         View.afficherMessageErreur("Veuillez d'abord sélectionner une ligne à modifier.");
120         return;
121     }
122     try {
123         nom = View.getNom();
124         prenom = View.getPrenom();
125         email = View.getEmail();
126         telephone = View.getTelephone();
127         salaire = View.getSalaire();
128         role = View.getRole();
129         poste = View.getPoste();
130
131         boolean updateSuccessful = model_employe.updateEmployee(id, nom, prenom, email, telephone, salaire, role, poste, solde);
132
133         if (updateSuccessful) {
134             test = false;
135             View.afficherMessageSucces("L'employé a été modifié avec succès.");
136             displayEmployee();
137             View.viderChamps_em();
138         } else {
139             View.afficherMessageErreur("Erreur lors de la mise à jour de l'employé.");
140         }
141     }
142 }
143

```



```

138     } else {
139         View.afficherMessageErreur("Erreur lors de la mise à jour de l'employé.");
140     }
141 } catch (Exception e) {
142     View.afficherMessageErreur("Erreur lors de la mise à jour");
143 }
144 }
145
146 public void resetSolde() { no usages
147     Calendar now = Calendar.getInstance();
148     if (now.get(Calendar.DAY_OF_YEAR) == 1) {
149         for (Employee employee : model_employe.displayEmploye()) {
150             updateSolde(employee.getId(), solde: 25);
151         }
152     }
153 }
154
155 public static void updateSolde(int id, int solde) { 4 usages
156     boolean updateSuccessful = model_employe.updateSolde(id, solde);
157 }
158

```

```

158
159 // Handle Import operation
160 public void handleImport() { 1 usage
161     JFileChooser fileChooser = new JFileChooser();
162     fileChooser.setFileFilter(new FileNameExtensionFilter( description: "Fichiers CSV", ...extensions: "csv", "txt"));
163
164     if (fileChooser.showOpenDialog(View) == JFileChooser.APPROVE_OPTION) {
165         String filePath = fileChooser.getSelectedFile().getAbsolutePath();
166         model_employe.importData(filePath); // Remplacer model par model_employe
167         View.afficherMessageSucces("Importation réussie !");
168     }
169 }
170
171 // Handle Export operation
172 public void handleExport() { 1 usage
173     JFileChooser fileChooser = new JFileChooser();
174     fileChooser.setFileFilter(new FileNameExtensionFilter( description: "Fichiers CSV", ...extensions: "csv"));
175
176     if (fileChooser.showSaveDialog(View) == JFileChooser.APPROVE_OPTION) {
177         try {
178             String filePath = fileChooser.getSelectedFile().getAbsolutePath();
179             if (!filePath.toLowerCase().endsWith(".txt")) {
180                 filePath += ".txt";
181             }

```

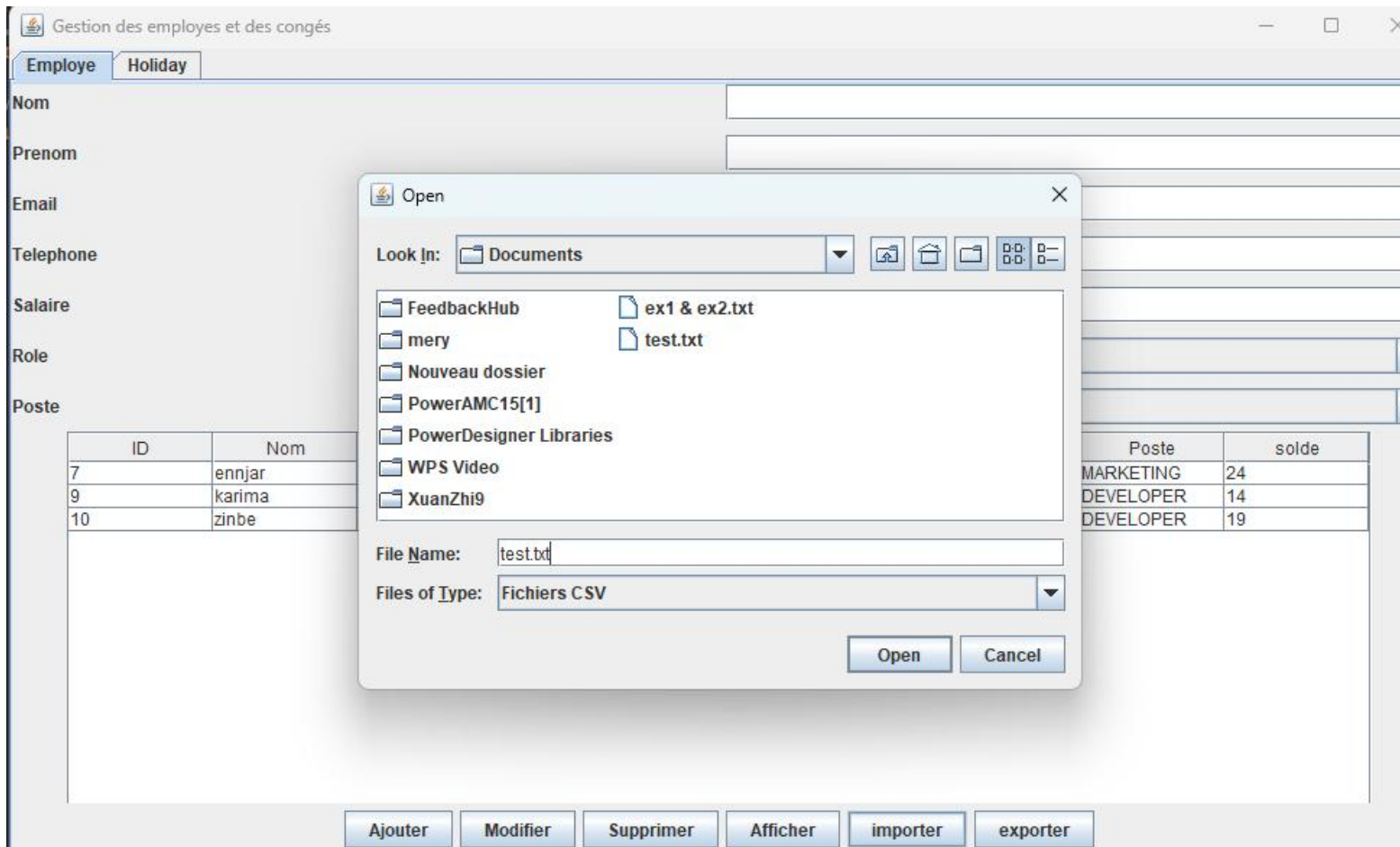
```

179  ✓      if (!filePath.toLowerCase().endsWith(".txt")) {
180          filePath += ".txt";
181      }
182      List<Employe> employes = new EmployeDAOimpl().display();
183      model_employe.exportData(filePath, employes); // Remplacer model par model_employe
184      View.afficherMessageSucces("Exportation réussie !");
185  ✓  } catch (Exception ex) {
186      View.afficherMessageErreur("Une erreur inattendue est survenue : " + ex.getMessage());
187  }
188  }
189  }
190  }
191

```

Resultat

1 L'importation



Gestion des employes et des congés

Employee Holiday

Nom

Prenom

Email

Telephone

Salaire

Role

Poste

Succès

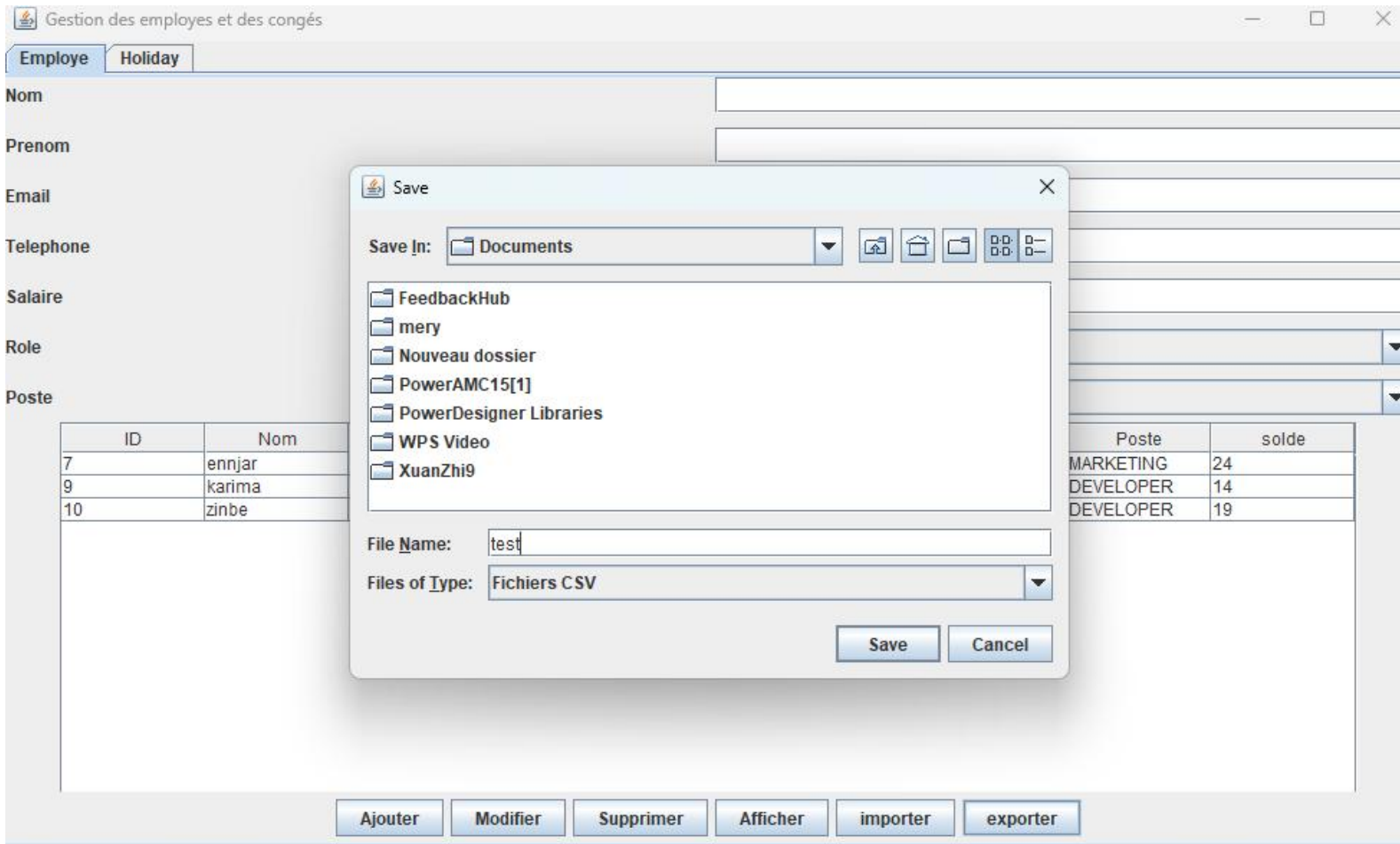
Importation réussie !

OK

ID	Nom	Prenom	Role	Poste	solde
7	ennjar	hiba	EMPLOYEE	MARKETING	24
9	karima	kamal	ADMIN	DEVELOPER	14
10	zinbe	wafi	ADMIN	DEVELOPER	19

Ajouter Modifier Supprimer Afficher importer exporter

2 L'exportation la liste des employés



Gestion des employes et des congés

Employee Holiday

Nom

Prenom

Email

Telephone

Salaire

Role

Poste

ID	Nom	Prenom	Role	Poste	solde
7	ennjar	hiba	EMPLOYEE	MARKETING	24
9	karima	kamal	ADMIN	DEVELOPER	14
10	zinbe	wafi	ADMIN	DEVELOPER	19

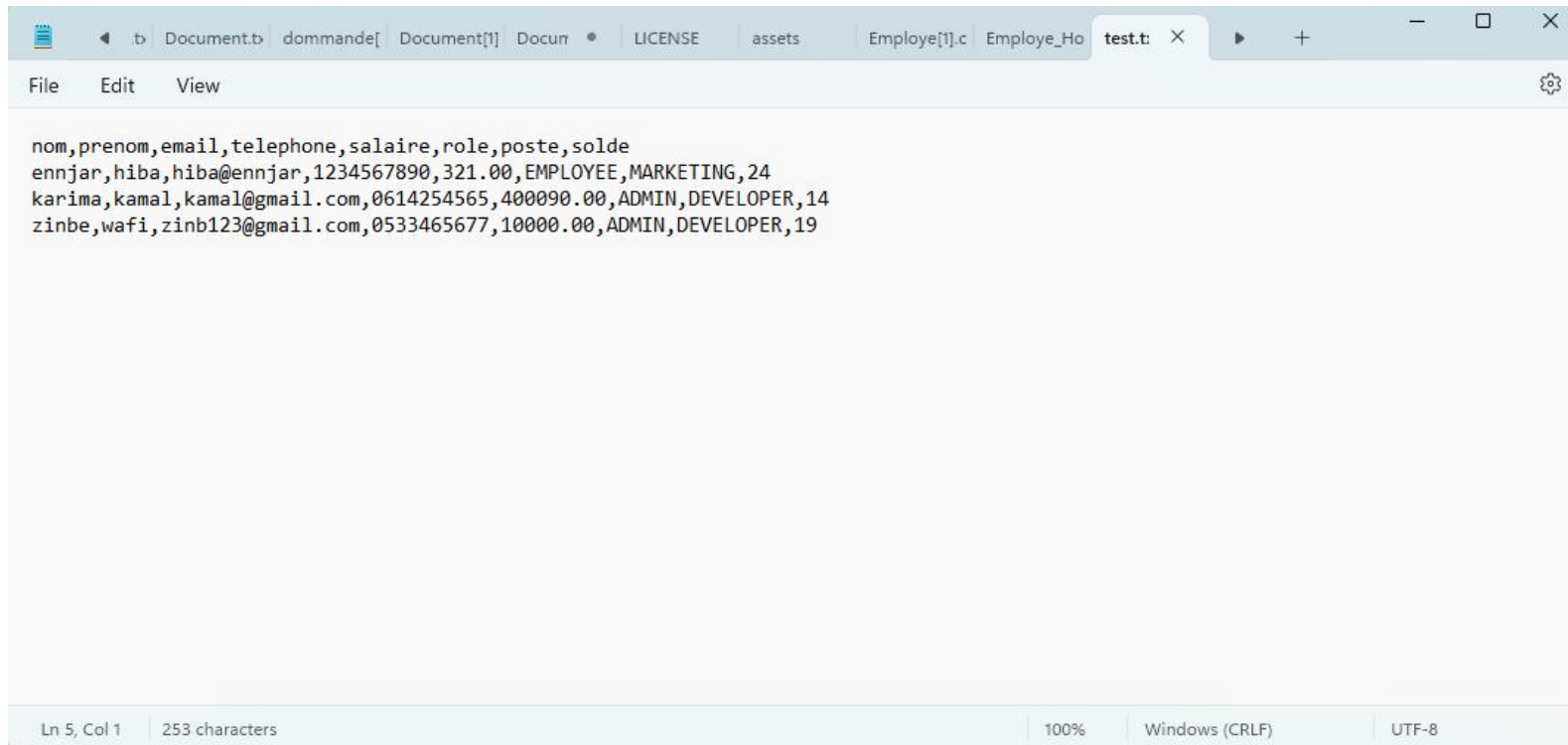
Ajouter Modifier Supprimer Afficher importer exporter

Succès

Exportation réussie !

OK

3 gestion des fichiers au forme txt



The screenshot shows a text editor window with a light blue header bar. The header bar contains a menu bar with 'File', 'Edit', and 'View'. Below the menu bar is a toolbar with various icons. The main area of the window displays the content of the file 'test.txt'. The content is a CSV file with 5 columns: nom, prenom, email, telephone, salaire, role, poste, solde. The data is as follows:

nom	prenom	email	telephone	salaire	role	poste	solde
ennjar	hiba	hiba@ennjar	1234567890	321.00	EMPLOYEE	MARKETING	24
karima	kamal	kamal@gmail.com	0614254565	400090.00	ADMIN	DEVELOPER	14
zinbe	wafi	zinb123@gmail.com	0533465677	10000.00	ADMIN	DEVELOPER	19

The status bar at the bottom of the window shows 'Ln 5, Col 1', '253 characters', '100%', 'Windows (CRLF)', and 'UTF-8'.

Conclusion générale

Ce TP nous a permis de découvrir et de mettre en œuvre les principes fondamentaux de la manipulation des fichiers dans un environnement applicatif. À travers les opérations d'importation et d'exportation, nous avons appris à gérer efficacement des données externes, en les intégrant dans notre système ou en les exportant vers des formats standardisés.

Nous avons également mis en pratique des concepts clés tels que la vérification des droits d'accès, le contrôle du format des fichiers et la gestion des exceptions pour assurer la robustesse et la fiabilité du système. Ces étapes sont essentielles pour garantir une expérience utilisateur fluide et une bonne interopérabilité entre les systèmes.

En conclusion, ce TP a renforcé nos compétences en programmation orientée objet, en particulier dans l'utilisation du modèle MVC et des DAO, tout en soulignant l'importance d'une gestion rigoureuse des données. Ces apprentissages sont indispensables pour concevoir des applications performantes et évolutives répondant aux besoins réels des utilisateurs.

Références

java :

— <https://www.java.com/en/download/>

intellij idea :

— <https://www.jetbrains.com/idea/download/?ref=freeStuffDevsection=windows>

XAMPP :

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

jdk 23 :

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