

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

gestion des congés(Mécanismes d'entrées/sorties)

Rapport du tp 3(java avancé)

Réalisé par : ELHIYANI Hanane

Encadré par : Mme.ELKHROF Leila

ANNÉE UNIVERSITAIRE : 2024/2025

Table des matières

Inroduction	4
Outils & environnement de travail	5
1 Language de programmation	5
2 Environnement de travail	5
1 Elaboration & Réalisation	7
1 Couche DAO	7
2 Couche Model	9
3 Couche View	10
4 Couche Controller	11
5 Couche DAO	12
6 Couche Model	13
7 Couche View	14
8 Couche Controller	14
2 Résultats	16
1 Button Import	16
2 Button Exporté	18
3 Conclusion	21

Table des figures

1	JAVA logo	5
2	intellij idea logo	5
3	xampp logo	6
2.1	Processus d'importation	17
2.2	Fichier importé	17
2.3	L'ajout de contenu du fichier importé au lite des employe	17
2.4	Processus d'exportation	18
2.5	Fichier exporté	19
2.6	Fichier importé	19
2.7	Resultat d'importation	20
2.8	Fichier exporté	21

Inroduction

Dans le cadre du TP n°3 intitulé « **Gestion de congés** », l'objectif est de poursuivre le développement de l'application de gestion des employés et des congés, déjà entamée lors des TP précédents. Après avoir implémenté les fonctionnalités de base de gestion des employés et des congés dans les **TP n°1** et **n°2**, ce troisième volet se concentre exclusivement sur l'ajout des fonctionnalités d'importation et d'exportation des données.

Ces nouvelles fonctionnalités visent à améliorer le système existant en permettant :

- **L'importation** de nouvelles données d'employés à partir de fichiers ;
- **L'exportation** de données, notamment la liste complète des employés ou celle des employés ayant des congés ;
- La gestion des fichiers au format **TXT** ou **CSV**, afin de faciliter le traitement et l'échange des informations.

Ce TP met l'accent sur l'intégration des notions de généricité et de sérialisation des données, tout en respectant l'architecture **MVC** et le modèle **DAO**. Le travail est structuré en plusieurs étapes, incluant la création d'une interface générique pour l'import/export, l'extension de la couche modèle, l'ajout d'éléments graphiques pour la vue, et la gestion des événements associés dans le contrôleur.

Cette extension permet d'optimiser l'application en offrant des outils pratiques pour la manipulation des données, tout en consolidant les acquis des TP précédents dans un contexte applicatif plus avancé.

Outils & environnement de travail

1 Language de programmation



FIGURE 1 – JAVA logo

- **Java** : est un langage de programmation orienté objet, sécurisé, portable et indépendant de la plateforme. Il est utilisé pour développer des applications web, mobiles et de bureau, grâce à sa machine virtuelle Java (JVM), qui permet d'exécuter le même code sur différentes plateformes.

2 Environnement de travail



FIGURE 2 – intellij idea logo

- **intellij idea** : est un environnement de développement intégré (IDE) puissant et populaire, principalement utilisé pour le développement en Java. Développé par JetBrains, il offre des fonctionnalités avancées telles que l'autocomplétion intelligente, la navigation dans le code, le débogage, le contrôle de version intégré et la prise en charge de plusieurs langages de programmation. IntelliJ IDEA est apprécié pour sa productivité accrue et son interface conviviale.



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



- **MySQL Workbench** : est un outil visuel de gestion de bases de données développé par Oracle. Il permet de concevoir, modéliser, administrer et interagir avec des bases de données MySQL.

Elaboration & Réalisation

Explications des Codes

Après avoir présenté les objectifs et les fonctionnalités générales de ce TP, nous allons maintenant détailler l'implémentation des différentes parties du code. Chaque section expliquera les ajouts réalisés dans chaque couche de l'application, en mettant l'accent sur les principes du modèle **MVC** et les spécificités de l'importation et de l'exportation des données. Ces explications permettront de mieux comprendre les choix techniques effectués et leur impact sur la structure et la performance de l'application.

1 Couche DAO

- **DataImportExport :**

Dans le cadre de la gestion des données, une interface générique `DataImportExport` a été créée pour définir des méthodes d'importation et d'exportation des données. Cette interface déclare deux méthodes principales : `importData(String fileName)`, qui permet d'importer des données depuis un fichier, et `exportData(String fileName, List<T> data)`, qui permet d'exporter des données vers un fichier. L'interface est ensuite implémentée par des classes spécifiques telles que `EmployeeDAOImp` et `HolidayDAOImp`.

```

1 package DAO;
2
3 import Model.Employe;
4
5 import java.io.IOException;
6 import java.util.List;
7 public interface DataImportExport<T> {
8     void importData(String fileName) throws IOException;
9     void exportData(String fileName, List<T> data) throws IOException;
10
11 }
12 }

```

• EmployeDAOImpl

Dans le cadre de la gestion des employés, deux fonctionnalités essentielles ont été ajoutées pour l'importation et l'exportation des données. La méthode `importData(String filePath)` permet d'importer des informations sur les employés depuis un fichier CSV ou un fichier texte. Elle lit chaque ligne du fichier, sépare les valeurs par des virgules et les insère dans la base de données. Cette méthode utilise un `BufferedReader` pour lire les fichiers et un `PreparedStatement` pour exécuter les requêtes d'insertion.

La méthode `exportData(String fileName, List<Employe> data)` permet d'exporter les données des employés vers un fichier CSV ou texte. Elle écrit les informations de chaque employé dans un fichier, en respectant un format précis (nom, prénom, email, téléphone, salaire, rôle, poste, solde). Un `BufferedWriter` est utilisé pour écrire ces données ligne par ligne dans le fichier spécifié.

Ces fonctionnalités ont été ajoutées pour faciliter la gestion des données des employés, permettant ainsi d'importer et d'exporter facilement les informations depuis et vers des fichiers CSV ou texte.

```

1 @Override
2     public void importData(String filePath) {
3         String sql = "INSERT INTO Employee (nom, prenom, email, phone, salaire,
4             role, poste, used_balance) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
5         try (BufferedReader reader = new BufferedReader(new FileReader(filePath));
6             PreparedStatement stmt = connexion.getConnexion().prepareStatement(sql)) {
7             String line = reader.readLine();
8             while ((line = reader.readLine()) != null) {
9                 String[] data = line.split(",");
10                System.out.println("la long de data :"+data.length );
11                if (data.length == 8) {
12
13                    stmt.setString(1, data[0].trim());
14                    stmt.setString(2, data[1].trim());
15                    stmt.setString(3, data[2].trim());
16                    stmt.setString(4, data[3].trim());
17                    stmt.setDouble(5, Double.parseDouble(data[4].trim()));
18                    stmt.setString(6, data[5].trim());
19                    stmt.setString(7, data[6].trim());
20                    stmt.setInt(8, Integer.parseInt(data[7].trim()));
21                    stmt.addBatch();
22                }
23            }
24        }
25    }

```



```

21         stmt.executeBatch();
22     }
23     System.out.println("Employees imported successfully !");
24
25     } catch (SQLException | IOException e) {
26         e.printStackTrace();
27     }
28
29 }
30 @Override
31 public void exportData (String fileName, List<Employee> data) throws IOException
32 {
33     try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName)))
34     {
35         writer.write("nom, prenom, email, phone, salaire, role, poste, solde");
36         writer.newLine();
37         for (Employee employee : data) {
38             String line = String.format("%s,%s,%s,%s,%.2f,%s,%s,%d",
39                 employee.getNom(),
40                 employee.getPrenom(),
41                 employee.getEmail(),
42                 employee.getTelephone(),
43                 employee.getSalaire(),
44                 employee.getRole(),
45                 employee.getPoste(),
46                 employee.getBalance()
47             );
48             writer.write(line);
49             writer.newLine();
50         }
51     }
52 }
53
54
55 }

```

2 Couche Model

- **EmployeeModel :**

Dans le cadre de l'amélioration de l'application de gestion des employés, la classe `EmployeeModel` a été enrichie par l'ajout de deux nouvelles fonctionnalités : l'importation et l'exportation des données. Ces fonctionnalités permettent à l'application de gérer les informations des employés stockées dans des fichiers au format texte ou CSV. La méthode `importData` permet de charger les données depuis un fichier externe après avoir effectué plusieurs vérifications, telles que l'existence du fichier, sa validité en tant que fichier et sa lisibilité. De son côté, la méthode `exportData` permet d'écrire les données des employés dans un fichier externe, facilitant ainsi leur sauvegarde ou leur partage. Ces ajouts viennent compléter la fonctionnalité existante de gestion des employés, offrant ainsi une plus

grande flexibilité dans le traitement des données tout en maintenant l'architecture basée sur le modèle MVC et DAO.

```
1 private boolean checkFileExists(File file) {
2
3     if(!file.exists()) {
4         throw new IllegalArgumentException ("le fichier n'existe pas "+file.
5         getPath());
6     }
7     return true;
8
9 }
10 private boolean checkIsFile(File file) {
11
12     if(!file.isFile()) {
13         throw new IllegalArgumentException ("le chemin specifie nest pas un
14         fichier "+file.getPath());
15     }
16     return true;
17
18 }
19 private boolean checkIsReadabal(File file) {
20
21     if(!file.canRead()) {
22         throw new IllegalArgumentException ("le chemin specifie nest pas
23         lisibles "+file.getPath());
24     }
25     return true;
26
27 }
28 public void importData(String FileName)throws IOException {
29     File file = new File(FileName);
30     checkFileExists(file);
31     checkIsFile(file);
32     checkIsReadabal(file);
33     dao.importData(FileName);
34 }
35
36 public void exportData(String FileName , List<Employe> data) throws
37 IOException {
38     File file = new File(FileName);
39     dao.exportData(FileName, data);
40 }
```

3 Couche View

- **EmployeView**

Dans la vue, nous avons ajouté simplement les deux boutons *Exporter* et *Importer*, et créé leurs getters afin de pouvoir les utiliser dans le contrôleur.

```

1 package View;
2 JButton Importer = new JButton("Importer");
3     JButton Exporter = new JButton("Exporter");
4     pan4.add(Importer);
5     pan4.add(Exporter);
6 public JButton getExporter(){ return Exporter ;}
7 public JButton getImporter(){ return Importer;}

```

4 Couche Controller

- **EmployeController**

Les modifications apportées pour la gestion de l'importation et de l'exportation des données dans Controller concernent principalement l'ajout de deux boutons dans la vue : Importer et Exporter. Dans le contrôleur, ces boutons sont associés aux méthodes `handleImport()` et `handleExport()`. La méthode d'importation permet à l'utilisateur de choisir un fichier CSV ou TXT via un dialogue de fichier, puis d'importer les données en utilisant la fonction `importData()` du modèle. La méthode d'exportation permet à l'utilisateur de sauvegarder les données des employés dans un fichier texte, en choisissant un emplacement et un nom via un autre dialogue de fichier, puis en utilisant la méthode `exportData()` du modèle pour générer le fichier. Ces ajouts facilitent l'intégration et l'exportation des données des employés.

```

1 package View;
2
3 this.view.getExporter().addActionListener(e ->handleExport());
4     this.view.getImporter().addActionListener(e -> handleImport());
5     public void handleImport() {
6         JFileChooser fileChooser = new JFileChooser();
7         fileChooser.setFileFilter(new FileNameExtensionFilter("Fichiers CSV", "csv", "txt"));
8
9         if (fileChooser.showOpenDialog(view) == JFileChooser.APPROVE_OPTION) {
10             try {
11                 String filePath = fileChooser.getSelectedFile().getAbsolutePath();
12                 model.importData(filePath);
13                 view.afficherMessageSucces("Importation russie !");
14             }catch(IOException ex){
15                 view.afficherMessageErreur("erreur lors de l'importation "+ex.
16                 getMessage());
17             }
18         }
19     }
20
21     public void handleExport() {
22         JFileChooser fileChooser = new JFileChooser();

```

```

23     fileChooser.setFileFilter(new FileNameExtensionFilter("Fichiers CSV", "csv
    "));
24
25     if (fileChooser.showSaveDialog(view) == JFileChooser.APPROVE_OPTION) {
26         try {
27             String filePath = fileChooser.getSelectedFile().getAbsolutePath();
28             if (!filePath.toLowerCase().endsWith(".txt")) {
29                 filePath += ".txt";
30             }
31             List<Employe> employes =new EmployeDAOImp().getAll();
32             model.exportData(filePath, employes);
33             view.afficherMessageSucces("Exportation russie !");
34         } catch (IOException ex) {
35             view.afficherMessageErreur("Une erreur inattendue est survenue : "
+ ex.getMessage());
36         }
37     }
38 }
39 }

```

Les mêmes Modifications sont appliquées dans la gestion des Conge (Holiday).

5 Couche DAO

• HolidayDAOImpl

```

1  @Override
2  public void importData(String filePath) {
3      String sql = "INSERT INTO holiday (employeeId, type, startdate, enddate)
VALUES (?, ?, ?, ?)";
4      try (BufferedReader reader = new BufferedReader(new FileReader(filePath));
      PreparedStatement stmt = connexion.getConnexion().prepareStatement(sql)) {
5          String line = reader.readLine();
6          while ((line = reader.readLine()) != null) {
7              String[] data = line.split(",");
8              System.out.println("la long de data :" + data.length);
9              if (data.length == 4) {
10
11                  stmt.setInt(1, Integer.parseInt(data[0].trim()));
12                  stmt.setString(2, data[1].trim());
13                  stmt.setString(3, data[2].trim());
14                  stmt.setString(4, data[3].trim());
15
16                  stmt.addBatch();
17              }
18              stmt.executeBatch();
19          }
20          System.out.println("Holiday imported successfully !");
21
22      } catch (SQLException | IOException e) {
23          e.printStackTrace();
24      }
25  }

```

```

26
27     @Override
28     public void exportData(String fileName, List<Holiday> data) throws
IOException {
29         try (BufferedWriter writer = new BufferedWriter(new FileWriter(
fileName))) {
30             writer.write("employeeId, type, startdate, enddate");
31             writer.newLine();
32             for (Holiday h : data) {
33                 String line = String.format("%d,%s,%s,%s",
34                     h.getEmployeeId(),
35                     h.getType(),
36                     h.getStartDate(),
37                     h.getEndDate()
38
39                 );
40
41                 writer.write(line);
42                 writer.newLine();
43             }
44         }
45     }
46 }

```

6 Couche Model

- **HolidayModel :**

```

1 private boolean checkFileExists(File file) {
2
3     if(!file.exists()) {
4         throw new IllegalArgumentException ("le fichier n'existe pas "+file.
getPath());
5
6     }
7     return true;
8
9 }
10 private boolean checkIsFile(File file) {
11
12     if(!file.isFile()) {
13         throw new IllegalArgumentException ("le chemin specifie nest pas un
fichier "+file.getPath());
14
15     }
16     return true;
17
18 }
19 private boolean checkIsReadebal(File file) {
20
21     if(!file.canRead()) {
22         throw new IllegalArgumentException ("le chemin specifie nest pas
lisibles "+file.getPath());
23

```

```

24     }
25     return true;
26
27 }
28 public void importData(String FileName)throws IOException {
29     File file = new File(FileName);
30     checkFileExists(file);
31     checkIsFile(file);
32     checkIsReadebal(file);
33     dao.importData(FileName);
34 }
35
36 public void exportData(String FileName , List<Holiday> data) throws
IOException {
37     File file = new File(FileName);
38     dao.exportData(FileName, data);
39 }

```

7 Couche View

- **HolidayView**

```

1 package View;
2 JButton Importer = new JButton("Importer");
3     JButton Exporter = new JButton("Exporter");
4     pan4.add(Importer);
5     pan4.add(Exporter);
6 public JButton getExporter(){ return Exporter ;}
7 public JButton getImporter(){ return Importer;}
8

```

8 Couche Controller

- **HolidayController**

```

1 this.view.getExporter().addActionListener(e -> handleExport());
2 this.view.getImporter().addActionListener(e -> handleImport());
3 private String formatDate(String date) {
4     try {
5         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
6         return sdf.format(sdf.parse(date));
7     } catch (ParseException e) {
8         view.afficherMessageErreur("Format de date invalide.");
9         return null;
10    }
11 }
12
13 public void handleImport() {
14     JFileChooser fileChooser = new JFileChooser();
15     fileChooser.setFileFilter(new FileNameExtensionFilter("Fichiers CSV", "csv", "txt"));
16

```

```
17         if (fileChooser.showOpenDialog(view) == JFileChooser.APPROVE_OPTION) {
18             try {
19                 String filePath = fileChooser.getSelectedFile().getAbsolutePath();
20                 model.importData(filePath);
21                 view.afficherMessageSucces("Importation r ussie !");
22             } catch (IOException ex) {
23                 view.afficherMessageErreur("erreur lors de l'importation " + ex.
24                 getMessage());
25             }
26         }
27     }
28
29     public void handleExport() {
30         JFileChooser fileChooser = new JFileChooser();
31         fileChooser.setFileFilter(new FileNameExtensionFilter("Fichiers CSV", "csv
32         "));
33
34         if (fileChooser.showSaveDialog(view) == JFileChooser.APPROVE_OPTION) {
35             try {
36                 String filePath = fileChooser.getSelectedFile().getAbsolutePath();
37                 if (!filePath.toLowerCase().endsWith(".txt")) {
38                     filePath += ".txt";
39                 }
40                 List<Holiday> Holidays = new HolidayDAOImpl().getAll();
41                 model.exportData(filePath, Holidays);
42                 view.afficherMessageSucces("Exportation r ussie !");
43             } catch (IOException ex) {
44                 view.afficherMessageErreur("Une erreur inattendue est survenue : "
45                 + ex.getMessage());
46             }
47         }
48     }
```

Résultats

1 Button Import

Lorsque l'on clique sur le bouton "Importer", on sélectionne le fichier préalablement créé, qui contient déjà les informations des employés. Une fois le fichier sélectionné, son contenu est directement ajouté à la base de données et à la liste des employés.

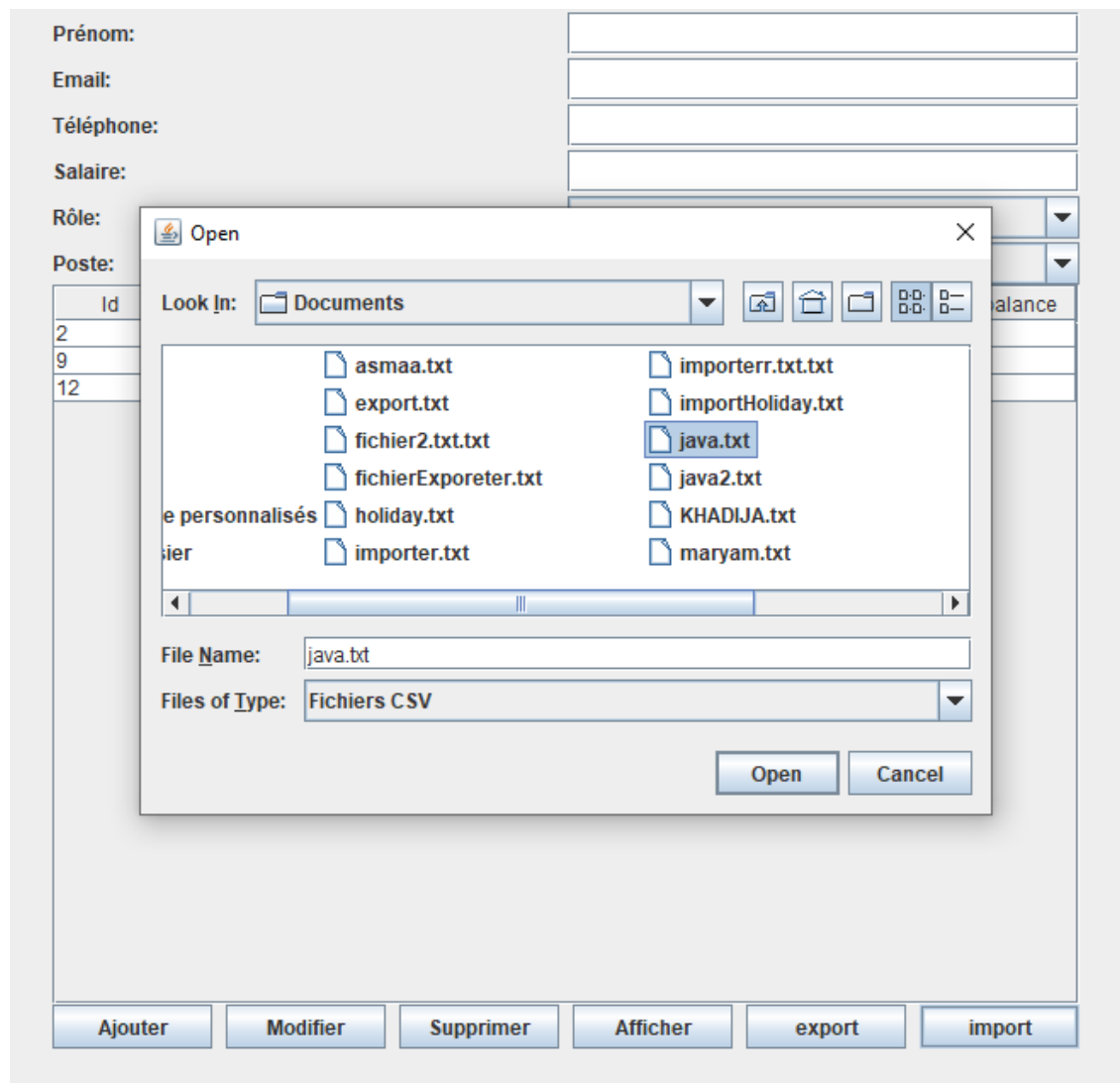


FIGURE 2.1 – Processus d’importation

```

Fichier  Edition  Format  Affichage  Aide
nom,prenom,email,phone,salaire,role,poste,used_balance
yussra,sdsq,yessra@gmail.com,0654578892,5678.00,ADMIN,Ingenieur,25

```

FIGURE 2.2 – Fichier importé

Id	Nom	Prenom	Telephone	Email	Salaire	Role	Poste	balance
2	hanan	hhdHJ	h@gmail....	0675457...	8790.0	ADMIN	Ingenieur	0
9	hananee	hhdHJ	h@gmail....	0675457...	8790.0	ADMIN	Ingenieur	14
12	Zineb	elhiyani	zineb@g...	0654578...	56789.0	ADMIN	Ingenieur	22
18	yussra	sdsq	yessra@...	0654578...	5678.0	ADMIN	Ingenieur	25

FIGURE 2.3 – L’ajout de contenu du fichier importé au lite des employe

2 Button Exporté

Lorsque l'on clique sur le bouton "Exporter", un fichier est créé automatiquement avec un nom spécifié, et les informations présentes dans la base de données sont enregistrées dans ce fichier.

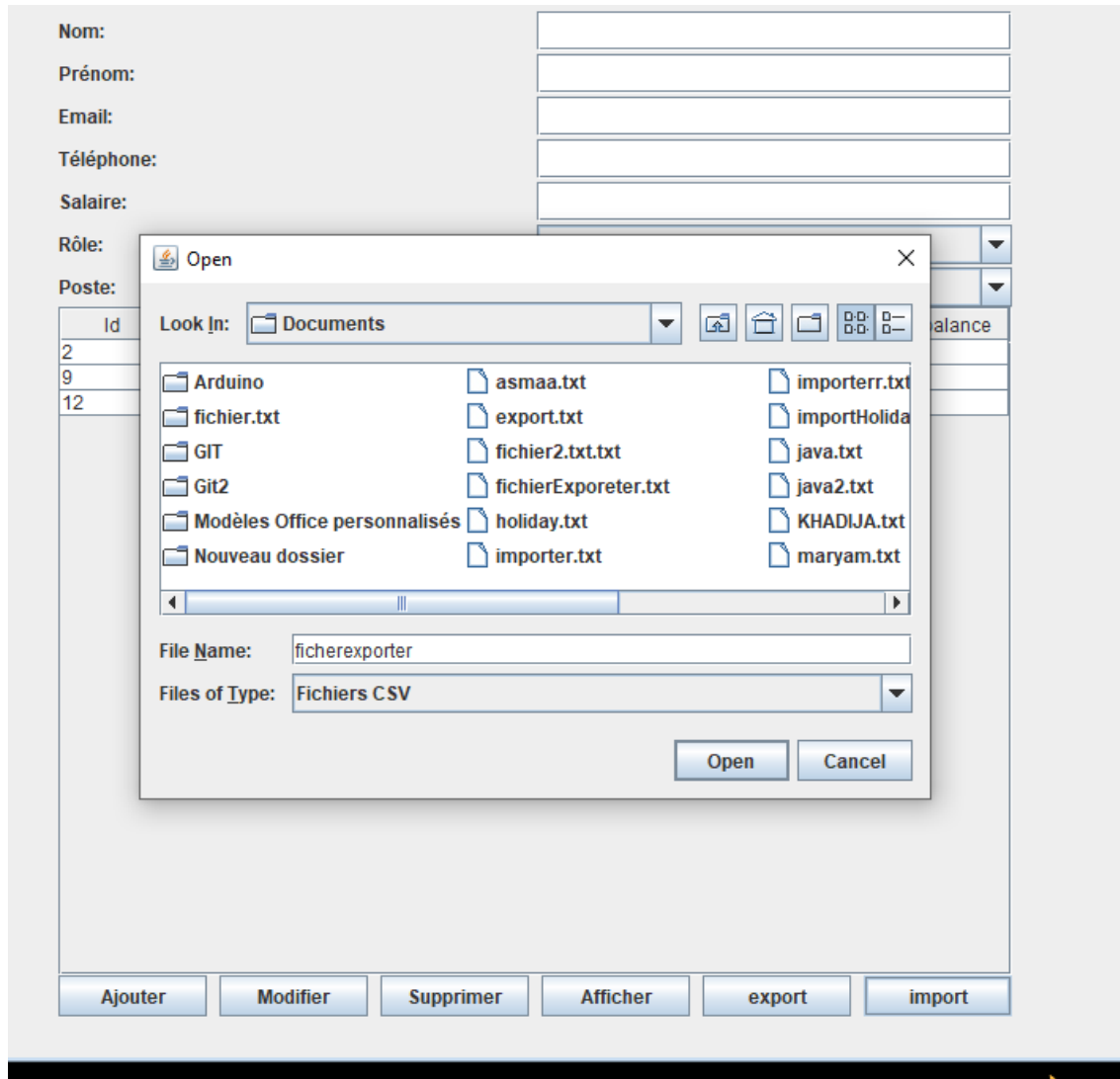
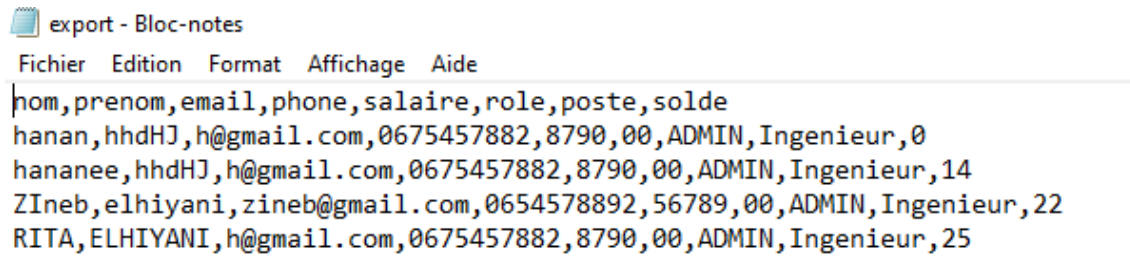


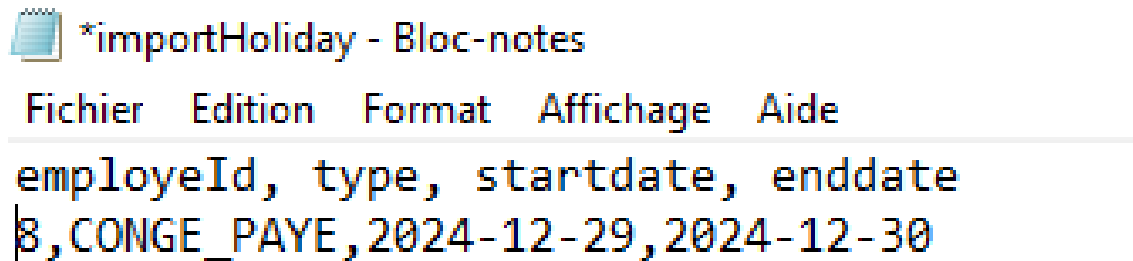
FIGURE 2.4 – Processus d'exportation



```
export - Bloc-notes
Fichier  Edition  Format  Affichage  Aide
nom,prenom,email,phone,salaire,role,poste,solde
hanan,hhdHJ,h@gmail.com,0675457882,8790,00,ADMIN,Ingenieur,0
hananee,hhdHJ,h@gmail.com,0675457882,8790,00,ADMIN,Ingenieur,14
ZIneb,elhiyani,zineb@gmail.com,0654578892,56789,00,ADMIN,Ingenieur,22
RITA,ELHIYANI,h@gmail.com,0675457882,8790,00,ADMIN,Ingenieur,25
```

FIGURE 2.5 – Fichier exporté

Les mêmes résultats pour les congés.



```
*importHoliday - Bloc-notes
Fichier  Edition  Format  Affichage  Aide
employeId, type, startdate, enddate
8,CONGE_PAYE,2024-12-29,2024-12-30
```

FIGURE 2.6 – Fichier importé

Employé:

Type de Congé:

Date de Début:

Date de Fin:

Id	Employé	Type de Congé	Date de Début	Date de Fin
4	hananee	CONGE_PAYE	2024-12-28	2024-12-30
6	Zineb	CONGE_PAYE	2025-01-01	2025-01-03
7	yussra	CONGE_PAYE	2024-12-29	2024-12-30
8	yussra	CONGE_PAYE	2024-12-29	2024-12-30

FIGURE 2.7 – Resultat d'importation

holiday - Bloc-notes

Fichier	Edition	Format	Affichage	Aide
employeeId, type, startdate, enddate				
9,	CONGE_PAYE,	2024-12-28,	2024-12-30	
12,	CONGE_PAYE,	2025-01-01,	2025-01-03	
18,	CONGE_PAYE,	2024-12-29,	2024-12-30	
8,	CONGE_PAYE,	2024-12-29,	2024-12-30	

FIGURE 2.8 – Fichier exporté

Conclusion

Ce travail pratique a permis de développer un système de gestion des employés avec des fonctionnalités d'importation et d'exportation de données. L'importation facilite l'ajout d'employés depuis un fichier, tandis que l'exportation permet de sauvegarder les données dans un fichier. Ce TP m'a permis d'améliorer mes compétences en gestion des bases de données et en interaction avec l'utilisateur, en appliquant des concepts théoriques à un projet concret.