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

Rapport du TP Gestion des Congés

Développement d'une application de gestion des congés(MVC,DAO et E/S)

Date de réalisation : 27 Décembre

Réalisé par :

Mlle. BAIOUMY Hiba

Enseigné par :

M. EL ABDELLAOUI Said
Mme. KACHBAL Ilham

Année Universitaire : 2024/2025

Table des matières

1 Introduction **4**

2 Étapes de Développement **5**
1 Gestion des Données (DAO) 5
1.1 Interface Générique 5
1.2 Implémentation 5
2 Logique Métier (Model) 7
3 Interface Graphique (Vue) 9
4 Contrôleur (Controller) 9

3 Résultats **11**
1 Page d'acceuil 11
2 Importation de nouvelles données d'employés 12
2.1 Message de reussite 13
3 Exportation de la liste des employés vers un fichier CSV 14

4 Conclusion Générale **15**

Table des figures

3.1 Page d'acceuil 11
3.2 Importation 12
3.3 Importation 13
3.4 Exportation 14

Introduction

Dans ce TP, nous avons pour objectif d'étendre une application de gestion des congés en implémentant des mécanismes d'import/export de données tout en intégrant les notions de DAO et de MVC. Cette extension permettra notamment :

— L'exportation de la liste complète des employés.
— L'importation de nouvelles données d'employés.
— La gestion des fichiers au format `txt` ou `csv`.

Ces fonctionnalités seront réalisées à travers plusieurs étapes détaillées dans les sections suivantes.

Étapes de Développement

1 Gestion des Données (DAO)

Objectif : Créer une interface générique pour l'import/export de données et implémenter cette interface dans une classe dédiée.

1.1 Interface Générique

```java
public interface DataImportExport<T> {
  void importData(String fileName) throws IOException;
  void exportData(String fileName, List<T> data) throws IOException;

}
```

Listing 2.1 – Interface DataImportExport

1.2 Implémentation

```java
public class EmployeeDAOImpl implements DataImportExport<Employee> {
    @Override
  public void importData(String filePath) {
    String query = "INSERT INTO Employe (nom, prenom, email, telephone,
    ↪ salaire, rol, poste, holiday) VALUES (?,?,?,?,?,?,?,?)";
    try(BufferedReader reader = new BufferedReader(new FileReader(
    ↪ filePath));
       PreparedStatement pstmt = conn.getConnexion().prepareStatement(
    ↪ query)){

       String line = reader.readLine();
       while((line = reader.readLine()) != null) {
         String[] data = line.split(",");
         if(data.length == 5) {
           pstmt.setString(1, data[0].trim());
           pstmt.setString(2, data[1].trim());
           pstmt.setString(3, data[2].trim());
           pstmt.setString(4, data[3].trim());
           pstmt.setString(5, data[4].trim());
           pstmt.setString(6, data[5].trim());
           pstmt.setString(7, data[6].trim());
           pstmt.addBatch();
         }
```

```
21        }
22        pstmt.executeBatch();
23        System.out.println("Employees imported successfully !");
24      }catch(IOException | SQLException e) {
25        e.printStackTrace();
26      }
27
28    }
29
30    @Override
31    public void exportData(String fileName, List<Employe> data) throws
       ↪ IOException {
32
33      try(BufferedWriter writer = new BufferedWriter(new FileWriter(
       ↪ fileName)))
34      {
35        writer.write("nom, prenom, email, telephone, salaire, rol, poste,
       ↪ holiday");
36        writer.newLine();
37
38        for(Employe employe : data) {
39          String line = String.format("%s,%s,%s,%s,%s,%s,%.2f",
40              employe.getNom(),
41              employe.getPrenom(),
42              employe.getEmail(),
43              employe.getTelephone(),
44              employe.getRole(),
45              employe.getPoste(),
46              employe.getUsedBalance()
47              );
48          writer.write(line);
49          writer.newLine();
50        }
51      }
52    }
53
54    private boolean checkFileExists(File file) {
55      if(!file.exists()) {
56        throw new IllegalArgumentException("Le fichier n'existe pas: " +
       ↪ file.getPath());
57      }
58      return true;
59    }
60    private boolean checkIsFile(File file) {
61      if(!file.isFile()) {
62        throw new IllegalArgumentException("Le chemin specifie n'est pas un
```

```
    ↪  fichier :" + file.getPath());
63    }
64    return true;
65  }
66  private boolean checkIsReadable(File file) {
67  if (!file.canRead ()) {
68  throw new IllegalArgumentException("Le fichier n'est pas lisible: " +
    ↪ file.getPath ());
69  }
70  return true;
71
72  }
73
74 }
```

Listing 2.2 – Classe EmployeeDAOImpl

2 Logique Métier (Model)

```
1 public class EmployeModel{
2     public List<Employe> importData(String filePath) {
3       List<Employe> employes = new ArrayList<>();
4       int failureCount = 0;
5
6       try (BufferedReader reader = new BufferedReader(new FileReader(
    ↪ filePath))) {
7           String line = reader.readLine();
8
9           while ((line = reader.readLine()) != null) {
10              String[] data = line.split(",");
11              if (data.length != 7) {
12                  System.err.println("Erreur : Nombre de colonnes
    ↪ invalide dans la ligne : " + line);
13                  failureCount++;
14                  continue;
15              }
16
17              try {
18                  // Lecture des donn es
19                  String nom = data[0].trim();
20                  String prenom = data[1].trim();
21                  String email = data[2].trim();
22                  String telephone = data[3].trim();
23                  double salaire = Double.parseDouble(data[4].trim());
24                  Rol role = Rol.valueOf(data[5].trim().toUpperCase());
```

```
                    Poste poste = Poste.valueOf(data[6].trim().toUpperCase
    ↪ ());

                    // Validation des donn es
                    if (email == null || !(email.contains("@"))) {
                        System.err.println("Erreur : email invalide dans la
    ↪  ligne : " + line);
                        failureCount++;
                        continue;
                    }
                    if (telephone.length() != 10) {
                        System.err.println("Erreur : num ro de t l phone
    ↪  invalide dans la ligne : " + line);
                        failureCount++;
                        continue;
                    }
                    if (salaire < 0) {
                        System.err.println("Erreur : salaire n gatif dans
    ↪ la ligne : " + line);
                        failureCount++;
                        continue;
                    }

                    // Cr er un objet Employe valide
                    Employe employe = new Employe(0, nom, prenom, email,
    ↪ telephone, salaire, role, poste);
                    employes.add(employe);

                } catch (Exception e) {
                    System.err.println("Erreur lors du traitement de la
    ↪ ligne : " + line);
                    e.printStackTrace();
                    failureCount++;
                }
            }

            System.out.println("Importation termin e !");
            System.out.println("Employ s valides : " + employes.size());
            System.out.println("Employ s   chous   : " + failureCount);

        } catch (IOException e) {
            e.printStackTrace();
        }

        return employes;
    }
```

```
}
```

Listing 2.3 – Classe EmployeM

3 Interface Graphique (Vue)

Objectif : Ajouter des boutons d'import/export dans l'interface utilisateur.

Exemple d'interface utilisateur :

— Bouton "Exporter en CSV".
— Bouton "Importer depuis un fichier".

4 Contrôleur (Controller)

Objectif : Gérer les événements liés aux boutons d'import/export.

```
public class EmployeController {
    this.view.importer.addActionListener(new ActionListener() {
        @Override

        public void actionPerformed(ActionEvent e) {
            handleImport();
        }});

    // ActionListener pour le bouton "Exporter"
    this.view.exporter.addActionListener(new ActionListener() {
        @Override

        public void actionPerformed(ActionEvent e) {
            handleExport(); // Appelle la m thode handleExport pour
    ↪ g rer l'exportation
        }
    });
}

  private void handleImport () {
  JFileChooser fileChooser = new JFileChooser ();
  fileChooser. setFileFilter (new FileNameExtensionFilter("Fichiers CSV",
   ↪  "txt"));

  if(fileChooser.showOpenDialog(view) == JFileChooser.APPROVE_OPTION) {
  try {
  String filePath = fileChooser.getSelectedFile().getAbsolutePath();
  employeModel.importData(filePath);
  view.afficherMessageSuccess("Importation r ussie");
```

```
28
29    }catch (Exception ex) {
30      view.afficherMessageError("Erreur lors de l'importation: " + ex.
       ↪ getMessage());
31    }
32    }
33 }
34
35 private void handleExport() {
36    JFileChooser fileChooser = new JFileChooser();
37    fileChooser.setFileFilter(new FileNameExtensionFilter("Fichiers CSV","
       ↪ csv"));
38
39    if (fileChooser.showSaveDialog(view) == JFileChooser.APPROVE_OPTION) {
40    try {
41    String filePath = fileChooser.getSelectedFile().getAbsolutePath();
42    List<String> holidays = new ArrayList<>();
43    if(!filePath.toLowerCase().endsWith(".txt")) {
44    filePath += ".txt";
45    }
46    List<Object[]> employe = employeModel.display();
47    holidayModel.exportData(filePath, holidays);
48    view.afficherMessageSuccess("Exportation r u s s i e");
49
50    }catch (IOException ex) {
51    view.afficherMessageError("Erreur lors de l'exportation : " +ex.
       ↪ getMessage());
52 }
53    }
54 }
```

Listing 2.4 – Gestion des événements

Résultats

1 Page d'acceuil

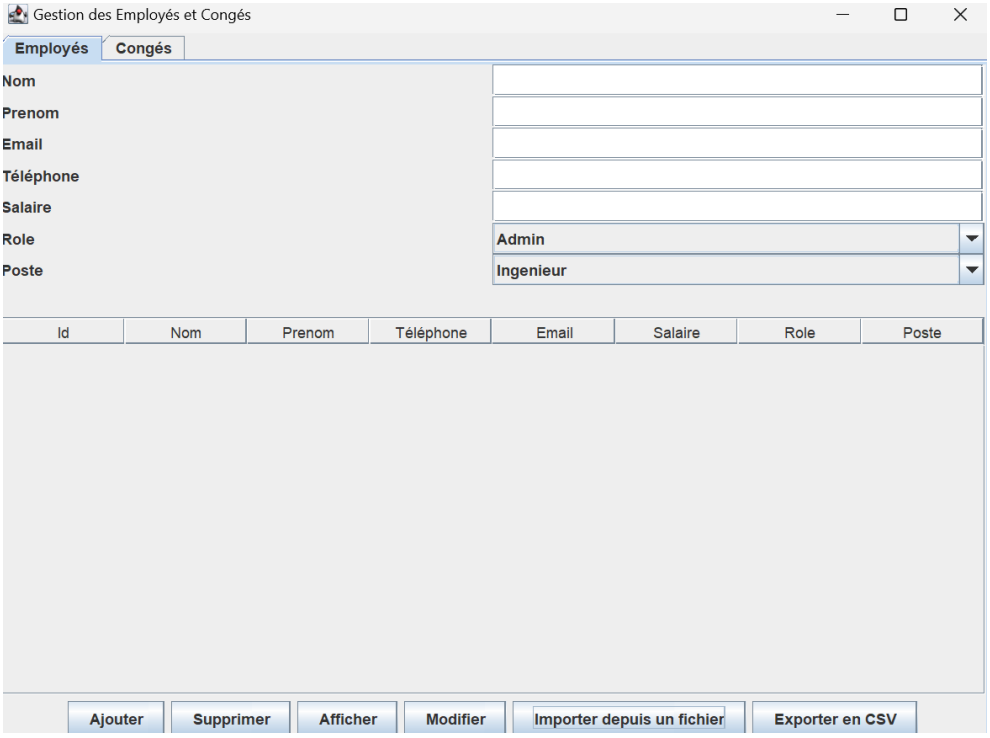

FIGURE 3.1 – Page d'acceuil

2 Importation de nouvelles données d'employés

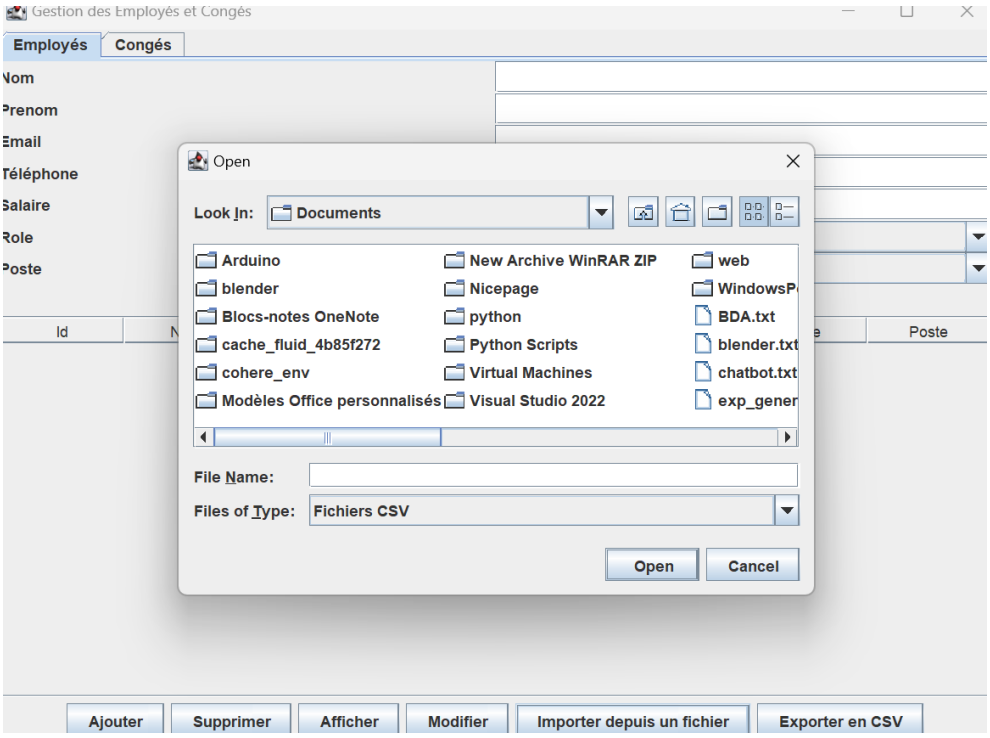

FIGURE 3.2 – Importation

2.1 Message de reussite

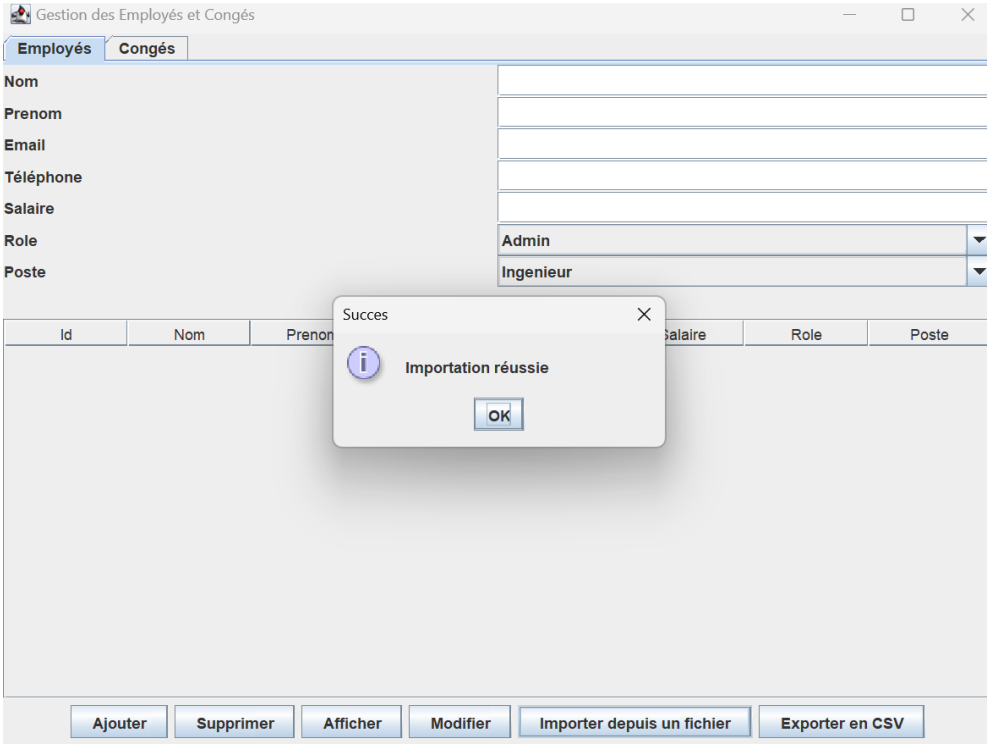

FIGURE 3.3 – Importation

3 Exportation de la liste des employés vers un fichier CSV

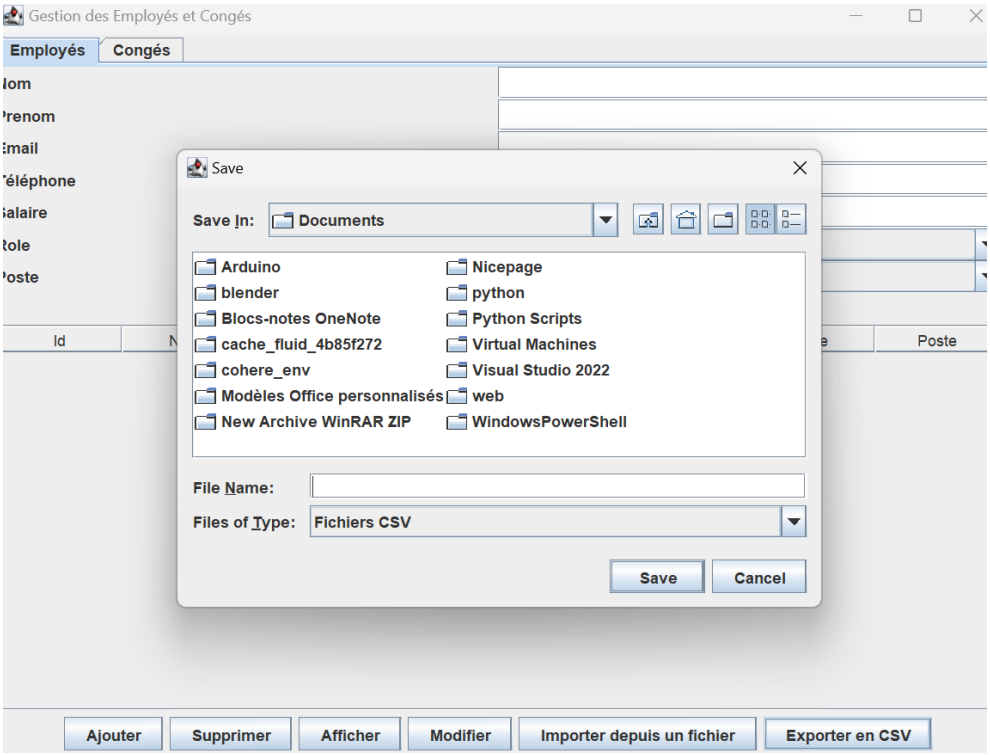

FIGURE 3.4 – Exportation

Conclusion Générale

Ce projet a permis de développer une application de gestion des employés offrant des fonctionnalités essentielles telles que l'ajout, la modification, la suppression et la consultation des informations des employés. Grâce à une interface simple et intuitive, l'utilisateur peut facilement interagir avec la base de données, assurant ainsi une gestion efficace des employés et de leurs congés.

L'application repose sur une architecture MVC, garantissant une séparation claire des responsabilités entre les différentes parties de l'application, ce qui facilite sa maintenance et son évolutivité. L'utilisation du modèle DAO (Data Access Object) permet une gestion optimale des interactions avec la base de données, offrant ainsi une abstraction des opérations de lecture, écriture, mise à jour et suppression des données, notamment pour les employés et les congés.

L'ajout de la gestion des congés permet de suivre le solde des congés de chaque employé et de gérer les demandes de congés, avec des informations détaillées telles que les dates, le type de congé et le lien avec les employés concernés. Cette fonctionnalité améliore considérablement la capacité de l'application à répondre aux besoins réels des entreprises.

L'implémentation de l'interface utilisateur a permis de garantir une expérience conviviale pour l'utilisateur, avec des fonctionnalités claires pour l'ajout, la modification, la suppression et l'affichage des employés et de leurs congés. Chaque opération est accompagnée de messages de confirmation, assurant ainsi une bonne communication avec l'utilisateur et minimisant les risques d'erreurs.

En conclusion, ce projet a permis d'acquérir des compétences en développement d'applications de gestion, en gestion de bases de données, ainsi qu'en conception d'interfaces utilisateurs simples et efficaces. L'ajout de la gestion des congés représente un enrichissement important des fonctionnalités, faisant de cette application une solution complète et professionnelle.

Le code source de l'application est disponible sur GitHub pour consultation et utilisation :
GitHub - Gestion des Employés