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

Rapport du TP N°3 (Généricités, MVC, DAO et E/S)

Gestion des employés et des Congés (E/S)

Réalisé par : CHABLAOUI Salma

Encadré par : Mme. KACHBAL Ilham

Année Universitaire : 2024/2025

Table des matières

Table des figures

Introduction

Ce travail pratique (TP) s'intéresse à la création d'une application Java pour la gestion des entrées et sorties des employés. Il est structuré autour de l'architecture MVC (Model-View-Controller), un modèle de conception qui sépare clairement les données, la logique métier et l'interface utilisateur. Ce projet vise à renforcer la maîtrise des concepts de la programmation orientée objet (POO) et à développer des interfaces graphiques avec la bibliothèque Swing. L'approche suivie dans ce TP permet également de perfectionner la gestion du code et de garantir une organisation optimale grâce à une séparation des responsabilités.

L'application développée dans ce TP a pour but de faciliter la gestion des données des employés. Elle permet d'importer et d'exporter facilement les informations liées aux employés via des fichiers externes, avec une interface utilisateur intuitive et conviviale. Cette approche simplifie les processus de gestion et garantit une interaction fluide et efficace avec l'application. L'utilisation de l'architecture MVC assure une gestion structurée du code, permettant une maintenance aisée et l'ajout futur de nouvelles fonctionnalités.

Les principales fonctionnalités de l'application incluent :

- L'importation des informations des employés à partir de fichiers externes.
- L'exportation des données des employés vers des fichiers pour des besoins d'archivage ou de partage.
- La gestion des entrées et sorties des employés avec une interface dédiée.

Ce projet vise à démontrer l'efficacité de la programmation orientée objet et de l'architecture MVC dans la création d'applications logicielles robustes et évolutives. Il constitue une étape clé dans la formation, permettant de maîtriser les concepts fondamentaux avant d'aborder des projets plus complexes à l'avenir.

Environnement et outils utilisés

1 Environnement de travail

Figure 1 – Eclipse logo

,

- **Eclipse** : Eclipse est un environnement de développement intégré (IDE) open-source, principalement utilisé pour le développement Java, mais aussi extensible pour d'autres langages via des plugins. Il offre des fonctionnalités comme la complétion automatique, le débogage et la gestion de projets, facilitant ainsi la création et le test d'applications logicielles.

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 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 :

1.1 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.

Employe

```
1  package Model;
2
3  public class Employe{
4      private int id;
5      private String nom;
6      private String prenom;
7      private String email;
8      private String telephone;
9      private double salaire;
10     private Role role;
11     private Post poste ;
12     private int solde ;
13
14     public Employe(int id ,String nom, String prenom, String email, String
       telephone, double salaire, Role role, Post post ,int solde){
15         this.id = id;
16         this.nom = nom;
17         this.prenom = prenom;
18         this.email = email;
19         this.telephone = telephone;
20         this.salaire = salaire;
21         this.role = role;
22         this.poste = post;
23         this.solde = solde;
```

```
24      }
25
26
27      public int getId(){
28          return id;
29      }
30
31      public void setId(int id){
32          this.id = id;
33      }
34      public String getNom() {
35          return nom;
36      }
37
38      public void setNom(String nom) {
39          this.nom = nom;
40      }
41
42      public String getPrenom() {
43          return prenom;
44      }
45
46      public void setPrenom(String prenom) {
47          this.prenom = prenom;
48      }
49
50      public String getEmail() {
51          return email;
52      }
53
54      public void setEmail(String email) {
55          this.email = email;
56      }
57
58      public String getTelephone() {
59          return telephone;
60      }
61
62      public void setTelephone(String telephone) {
63          this.telephone = telephone;
64      }
65
66      public double getSalaire() {
67          return salaire;
68      }
69
70      public void setSalaire(double salaire) {
71          this.salaire = salaire;
72      }
```

```
73
74      public Role getRole() {
75          return role;
76      }
77
78      public void setRole(Role role) {
79          this.role = role;
80      }
81
82      public Post getPost() {
83          return poste;
84      }
85
86      public void setPost(Post post) {
87          this.poste = post;
88      }
89
90      public void setSolde (int conge){
91          this.solde = conge;
92      }
93
94      public int getSolde(){
95          return solde;
96      }
97  }
```

Post

```
1  package Model;
2
3  public enum Post {
4    DEVELOPER, DESIGNER, MARKETING, OTHER
5  }
```

Role

```
1  package Model;
2
3  public enum Role {
4      ADMIN, EMPLOYEE ,MANAGER
5  }
```

Employemodel

```
1  package Model;
2
```

```
3  import java.io.File;
4  import java.util.List;
5  import DAO.EmployeDAOimpl;
6
7  public class Employemodel {
8      private EmployeDAOimpl dao;
9      public Employemodel(EmployeDAOimpl dao) {
10         this.dao = dao;
11     }
12     // funtion of add Employe
13 public boolean addEmploye(int id ,String nom, String prenom, String email,
       String telephone, double salaire, Role role, Post post, int solde) {
14         if(salaire < 0 ){
15             System.out.println("Erreur : le salaire doit etre positif.");
16             return false;
17         }
18         if(id < 0 ){
19             System.out.println("Erreur : l'id doit etre positif.");
20             return false;
21         }
22         if(telephone.length() != 10){
23             System.out.println("Erreur : le telephone doit etre 10 num.");
24             return false;
25         }
26         if(!email.contains("@")){
27             System.out.println("Erreur : le mail doit contenir le @.");
28             return false;
29         }
30
31         Employe e = new Employe(id,nom, prenom, email, telephone, salaire,
       role, post ,solde);
32
33         dao.add(e);
34
35         return true;
36     }
37
38     // function of delete Employe :
39
40     public  boolean deleteEmploye(int id){
41         dao.delete(id);
42         return true;
43     }
44
45     // function of update Employe :
46
47     public boolean updateEmploye(int id, String nom, String prenom, String
       email, String telephone, double salaire, Role role, Post post , int
       solde) {
```

```
48
49          Employe e = new Employe(id,nom, prenom, email, telephone, salaire,
    role, post,solde);
50          dao.update(e);
51          return true;
52      }
53
54      //function of update solde Employe :
55
56      public boolean updateSolde(int id, int solde) {
57          dao.updateSolde(id, solde);
58          return true;
59      }
60
61      //function of display Employe :
62
63      public List<Employe> displayEmploye() {
64          List<Employe> Employes = dao.display();
65          return Employes;
66      }
67  private boolean checkFileExits(File file) {
68
69      if(!file.exists()) {
70        throw new IllegalArgumentException ("le fichier n'existe pas "+file.
    getPath());
71
72      }
73      return true;
74
75    }
76  private boolean checkIsFile(File file) {
77
78    if(!file.isFile()) {
79      throw new IllegalArgumentException ("le chemin specifie nest pas un
     fichier "+file.getPath());
80
81    }
82    return true;
83
84  }
85  private boolean checkIsReadebal(File file) {
86
87    if(!file.canRead()) {
88      throw new IllegalArgumentException ("le chemin specifie nest pas
     lisibles "+file.getPath());
89    }
90    return true;
91      }
92  }
```

1.2 DAO

Le DAO est une couche qui permet de gérer l'interaction avec une base de données, en effectuant des opérations telles que la création, la lecture, la mise à jour et la suppression (CRUD) des données.

- • **DBConnexion**

```java
package DAO;

import java.sql.*;

class DBConnexion {
    public static final String url = "jdbc:mysql://localhost:3306/gestion";
    public static final String user = "root";
    public static final String password = "";
    public static Connection conn = null;

    public static Connection getConnexion() throws ClassNotFoundException {
        if (conn != null) {
            return conn;
        }
        try {

            Class.forName("com.mysql.cj.jdbc.Driver");
            conn = DriverManager.getConnection(url, user, password);
            System.out.println("correct");
        } catch (SQLException e) {
            throw new RuntimeException("Error de connexion", e);
        }
        return conn;
    }
}
```

DataImportExport

```java
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;

}
```

EmployeDAOimpl

```
1  package DAO;
2  import Model.Employe;
3  import Model.Post;
4  import Model.Role;
5
6  import java.io.BufferedReader;
7  import java.io.BufferedWriter;
8  import java.io.FileReader;
9  import java.io.FileWriter;
10 import java.io.IOException;
11 import java.sql.PreparedStatement;
12 import java.sql.ResultSet;
13 import java.sql.SQLException;
14 import java.util.ArrayList;
15 import java.util.List;
16
17
18 public class EmployeDAOimpl implements GenericDAOI<Employe>,
       DataImportExport<Employe> {
19
20     @Override
21     public void add(Employe e) {
22         // Do not include the 'id' column in the INSERT statement
       because it is AUTO_INCREMENT
23         String sql = "INSERT INTO employe (nom, prenom, email, telephone
       , salaire, role, poste, solde) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
24         try (PreparedStatement stmt = DBConnexion.getConnexion().
       prepareStatement(sql)) {
25             stmt.setString(1, e.getNom());
26             stmt.setString(2, e.getPrenom());
27             stmt.setString(3, e.getEmail());
28             stmt.setString(4, e.getTelephone());
29             stmt.setDouble(5, e.getSalaire());
30             stmt.setString(6, e.getRole().name());  // Assuming Role is
       an enum and needs to be converted to string
31             stmt.setString(7, e.getPost().name());  // Assuming Post is
       an enum and needs to be converted to string
32             stmt.setInt(8, e.getSolde());
33             stmt.executeUpdate();
34         } catch (SQLException exception) {
35             System.err.println("Failed to add employee: " + exception.
       getMessage());
36             exception.printStackTrace();  // Prints the full stack trace
        for debugging
37         } catch (ClassNotFoundException ex) {
38             System.err.println("Failed to connect to the database: " +
```

```
ex.getMessage());
39              ex.printStackTrace();  // Prints the full stack trace for
debugging
40          }
41      }
42
43      @Override
44      public void delete(int id) {
45          String sql = "DELETE FROM employe WHERE id = ?";
46          try (PreparedStatement stmt = DBConnexion.getConnexion().
prepareStatement(sql)) {
47              stmt.setInt(1,id);
48              stmt.executeUpdate();
49          } catch (SQLException exception) {
50              System.err.println("failed of delete employe");
51          } catch (ClassNotFoundException ex) {
52              System.err.println("failed connexion with data base");
53          }
54      }
55
56      @Override
57      public void update(Employe e) {
58          String sql = "UPDATE employe SET nom = ?, prenom = ?, email = ?,
 telephone = ?, salaire = ?, role = ?, poste = ? WHERE id = ?";
59          try (PreparedStatement stmt = DBConnexion.getConnexion().
prepareStatement(sql)) {
60              stmt.setString(1, e.getNom());
61              stmt.setString(2, e.getPrenom());
62              stmt.setString(3, e.getEmail());
63              stmt.setString(4, e.getTelephone());
64              stmt.setDouble(5, e.getSalaire());
65              stmt.setString(6, e.getRole().name());
66              stmt.setString(7, e.getPost().name());
67              stmt.setInt(8,e.getId());
68              stmt.executeUpdate();
69          } catch (SQLException exception) {
70              System.err.println("failed of update employe");
71          } catch (ClassNotFoundException ex) {
72              System.err.println("failed connexion with data base");
73          }
74      }
75      @Override
76      public List<Employe> display() {
77          String sql = "SELECT * FROM employe";
78          List<Employe> Employes = new ArrayList<>();
79          try (PreparedStatement stmt = DBConnexion.getConnexion().
prepareStatement(sql)) {
80              ResultSet re = stmt.executeQuery();
81              while (re.next()) {
```

```
82                  int id = re.getInt("id");
83                  String nom = re.getString("nom");
84                  String prenom = re.getString("prenom");
85                  String email = re.getString("email");
86                  String telephone = re.getString("telephone");
87                  double salaire = re.getDouble("salaire");
88                  String role = re.getString("role");
89                  String poste = re.getString("poste");
90                  int solde = re.getInt("solde");
91                  Employe e = new Employe(id,nom, prenom, email, telephone
    , salaire, Role.valueOf(role), Post.valueOf(poste),solde);
92                  Employes.add(e);
93              }
94              return Employes;
95          } catch (ClassNotFoundException ex) {
96              System.err.println("failed connexion with data base");
97              return null;
98          } catch (SQLException ex) {
99              System.err.println("failed of display employe");
100             return null;
101         }
102     }
103
104
105     public void updateSolde(int id, int solde) {
106         String sql = "UPDATE employe SET solde = ? WHERE id = ?";
107         try (PreparedStatement stmt = DBConnexion.getConnexion().
    prepareStatement(sql)) {
108             stmt.setInt(1, solde);
109             stmt.setInt(2, id);
110             stmt.executeUpdate();
111         } catch (SQLException exception) {
112             System.err.println("failed of update solde employe");
113         } catch (ClassNotFoundException ex) {
114             System.err.println("failed connexion with data base");
115         }
116     }
117
118 @Override
119 public void importData(String filePath) throws IOException {
120     String query = "INSERT INTO Employe(nom, prenom, email, telephone,
    salaire, role, poste) VALUES (?, ?, ?, ?, ?, ?, ?)";
121         try (BufferedReader reader = new BufferedReader(new FileReader
    (filePath));
122              PreparedStatement ps = DBConnexion.getConnexion().
    prepareStatement(query)) {
123
124             String line = reader.readLine(); // Skip the header
125             while ((line = reader.readLine()) != null) {
```

```
126                     String[] data = line.split(",");
127                     if (data.length == 7) {
128                         ps.setString(1, data[0].trim()); // nom
129                         ps.setString(2, data[1].trim()); // prenom
130                         ps.setString(3, data[2].trim()); // email
131                         ps.setString(4, data[3].trim()); // telephone
132                         ps.setString(5, data[4].trim()); // salaire
133                         ps.setString(6, data[5].trim()); // role
134                         ps.setString(7, data[6].trim()); // poste
135                         ps.addBatch();
136                     } else {
137                         System.err.println("Invalid line format: " + line)
    ;
138                     }
139                 }
140                 ps.executeBatch();
141                 System.out.println("Employ s import s avec succ s.");
142             } catch (IOException | SQLException | ClassNotFoundException e
    ) {
143                 e.printStackTrace();
144             }
145
146     }
147
148     @Override
149     public void exportData(String fileName, List<Employe> data) throws
     IOException {
150       try (BufferedWriter writer = new BufferedWriter(new FileWriter(
     fileName))) {
151                 writer.write("nom,prenom,email,telephone,role,poste,
     salaire");
152                 writer.newLine();
153                 for (Employe employee : data) {
154                     String line = String.format("%s,%s,%s,%s,%s,%s,%.2f",
155                             employee.getNom(),
156                             employee.getPrenom(),
157                             employee.getEmail(),
158                             employee.getTelephone(),
159                             employee.getRole(),
160                             employee.getPost(),
161                             employee.getSalaire());
162                     writer.write(line);
163                     writer.newLine();
164                 }
165                 System.out.println("Donn es export es avec succ s.");
166             }
167     }
168 }
```

1.3 View

EmployeHolidayView

```
1  package view;
2
3  import DAO.EmployeDAOimpl;
4  import Model.Employe;
5  import Model.Employemodel;
6  import Model.Post;
7  import Model.Role;
8  import Model.Type_holiday;
9  import java.awt.*;
10 import java.io.BufferedReader;
11 import java.io.FileReader;
12 import java.io.FileWriter;
13 import java.io.PrintWriter;
14
15 import javax.swing.*;
16 import javax.swing.table.DefaultTableModel;
17 import java.util.List;
18
19 public class EmployeHolidayView extends JFrame {
20
21     private JTabbedPane tabbedPane = new JTabbedPane();
22
23     private JPanel employeTab = new JPanel();
24     private JPanel holidayTab = new JPanel();
25
26     private JPanel Employepan = new JPanel();
27     private JPanel Holidaypan = new JPanel();
28     private JPanel Display_Table_employe = new JPanel();
29     private JPanel Display_Table_holiday = new JPanel();
30     private final JPanel Forme_employe = new JPanel();
31     private final JPanel Forme_holiday = new JPanel();
32     private JPanel panButton_employe = new JPanel();
33     private JPanel panButton_holiday = new JPanel();
34
35     // les labels du l'employe
36     private JLabel label_nom = new JLabel("Nom");
37     private JLabel label_prenom = new JLabel("Prenom");
38     private JLabel label_email = new JLabel("Email");
39     private JLabel label_tele = new JLabel("Telephone");
40     private JLabel label_salaire = new JLabel("Salaire");
41     private JLabel label_role = new JLabel("Role");
42     private JLabel label_poste = new JLabel("Poste");
43
44     // les labels du cong
45     private JLabel label_employe = new JLabel("Nom de l'employe");
```

```
46      private JLabel label_startDate = new JLabel("Date de debut (YYYY-MM-
        DD)");
47      private JLabel label_endDate = new JLabel("Date de fin (YYYY-MM-DD)"
        );
48      private JLabel label_type = new JLabel("Type");
49      private JComboBox<Type_holiday> TypeComboBox = new JComboBox<>(
        Type_holiday.values());
50
51      // les textfield du l'employe
52      private JTextField text_nom = new JTextField();
53      private JTextField text_prenom = new JTextField();
54      private JTextField text_email = new JTextField();
55      private JTextField text_tele = new JTextField();
56      private JTextField text_salaire = new JTextField();
57
58      private JComboBox<Role> roleComboBox = new JComboBox<>(Role.values()
        );
59      private JComboBox<Post> posteComboBox = new JComboBox<>(Post.values
        ());
60
61      // les textfield du 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      // le tableau de l'employe
84      JPanel pan0 = new JPanel(new BorderLayout());
85      public static String[] columnNames_employe = {"ID", "Nom", "Prenom",
        "Email", "T l phone", "Salaire", "Role", "Poste","solde"};
86      public static DefaultTableModel tableModel = new DefaultTableModel(
        columnNames_employe, 0);
87      public static JTable Tableau = new JTable(tableModel);
```

```
88
89      // le tableau du cong
90      JPanel pan1 = new JPanel(new BorderLayout());
91      public static String[] columnNames_holiday = {"ID", "nom_employe","
        date_debut","date_fin","type"};
92      public static DefaultTableModel tableModel1 = new DefaultTableModel(
        columnNames_holiday, 0);
93      public static JTable Tableau1 = new JTable(tableModel1);
94
95      public Employe_HolidayView() {
96
97          setTitle("Gestion des employes et des cong s");
98          setSize(1000, 600);
99          setDefaultCloseOperation(EXIT_ON_CLOSE);
100         setLocationRelativeTo(null);
101
102         add(tabbedPane);
103
104     // Employe Tab
105         employeTab.setLayout(new BorderLayout());
106         employeTab.add(Employepan, BorderLayout.CENTER);
107
108         Employepan.setLayout(new BorderLayout());
109         Employepan.add(Display_Table_employe, BorderLayout.CENTER);
110         Tableau.setFillsViewportHeight(true);
111         Dimension preferredSize = new Dimension(900, 500);
112         Tableau.setPreferredScrollableViewportSize(preferredSize);
113         pan0.add(new JScrollPane(Tableau), BorderLayout.CENTER);
114         Display_Table_employe.add(pan0);
115
116         Employepan.add(panButton_employe, BorderLayout.SOUTH);
117         panButton_employe.add(addButton_employe);
118         panButton_employe.add(updateButton_employe);
119         panButton_employe.add(deleteButton_employe);
120         panButton_employe.add(displayButton_employe);
121         panButton_employe.add(importButton_employe);
122         panButton_employe.add(exportButton_employe);
123
124
125         Employepan.add(Forme_employe, BorderLayout.NORTH);
126         Forme_employe.setLayout(new GridLayout(7, 2, 10, 10));
127         Forme_employe.add(label_nom);
128         Forme_employe.add(text_nom);
129         Forme_employe.add(label_prenom);
130         Forme_employe.add(text_prenom);
131         Forme_employe.add(label_email);
132         Forme_employe.add(text_email);
133         Forme_employe.add(label_tele);
134         Forme_employe.add(text_tele);
```

```
135         Forme_employe.add(label_salaire);
136         Forme_employe.add(text_salaire);
137         Forme_employe.add(label_role);
138         Forme_employe.add(roleComboBox);
139         Forme_employe.add(label_poste);
140         Forme_employe.add(posteComboBox);
141 
142     // Holiday Tab
143         holidayTab.setLayout(new BorderLayout());
144         holidayTab.add(Holidaypan, BorderLayout.CENTER);
145         Holidaypan.setLayout(new BorderLayout());
146         Holidaypan.add(Display_Table_holiday, BorderLayout.CENTER);
147 
148         Tableau1.setFillsViewportHeight(true);
149         Tableau1.setPreferredScrollableViewportSize(preferredSize);
150         pan1.add(new JScrollPane(Tableau1), BorderLayout.CENTER);
151         Display_Table_holiday.add(pan1);
152 
153         Holidaypan.add(Forme_holiday, BorderLayout.NORTH);
154         Forme_holiday.setLayout(new GridLayout(4, 2, 10, 10));
155         Forme_holiday.add(label_employe);
156         Forme_holiday.add(text_employe);
157         Forme_holiday.add(label_startDate);
158         Forme_holiday.add(text_startDate);
159         Forme_holiday.add(label_endDate);
160         Forme_holiday.add(text_endDate);
161         Forme_holiday.add(label_type);
162         Forme_holiday.add(TypeComboBox);
163 
164         Holidaypan.add(panButton_holiday, BorderLayout.SOUTH);
165         panButton_holiday.add(addButton_holiday);
166         panButton_holiday.add(updateButton_holiday);
167         panButton_holiday.add(deleteButton_holiday);
168         panButton_holiday.add(displayButton_holiday);
169 
170 
171 
172     // TabbedPane
173         tabbedPane.addTab("Employe", employeTab);
174         tabbedPane.addTab("Holiday", holidayTab);
175         importButton_employe.addActionListener(e -> {
176             JFileChooser fileChooser = new JFileChooser();
177             if (fileChooser.showOpenDialog(this) == JFileChooser.
    APPROVE_OPTION) {
178                 importData(tableModel, fileChooser.getSelectedFile().
    getPath());
179             }
180         });
181 
```

```
182         exportButton_employe.addActionListener(e -> {
183             JFileChooser fileChooser = new JFileChooser();
184             if (fileChooser.showSaveDialog(this) == JFileChooser.
    APPROVE_OPTION) {
185                 exportData(tableModel, fileChooser.getSelectedFile().
    getPath());
186             }
187         });
188         importButton_holiday.addActionListener(e -> {
189             JFileChooser fileChooser = new JFileChooser();
190             if (fileChooser.showOpenDialog(this) == JFileChooser.
    APPROVE_OPTION) {
191                 importData(tableModel1, fileChooser.getSelectedFile().
    getPath());
192             }
193         });
194 
195         exportButton_holiday.addActionListener(e -> {
196             JFileChooser fileChooser = new JFileChooser();
197             if (fileChooser.showSaveDialog(this) == JFileChooser.
    APPROVE_OPTION) {
198                 exportData(tableModel1, fileChooser.getSelectedFile().
    getPath());
199             }
200         });
201 
202         remplaire_les_employes();
203         setVisible(true);
204      }
205      public void remplaire_les_employes () {
206         List<Employe> Employes = new Employemodel(new EmployeDAOimpl()).
    displayEmploye();
207        text_employe.removeAllItems();
208        for (Employe elem : Employes) {
209            text_employe.addItem(elem.getId() + " - " + elem.getNom()+" "
    +elem.getPrenom());
210        }
211      }
212 
213      // getters
214         public int getId_employe() {
215             return Integer.parseInt(text_employe.getSelectedItem().
    toString().split(" - ")[0]);
216         }
217         public String getNom() {
218             return text_nom.getText();
219         }
220 
221         public JTable getTable() {
```

```
222             return (JTable) Display_Table_employe.getComponent(0);
223         }
224
225         public String getPrenom() {
226             return text_prenom.getText();
227         }
228
229         public String getEmail() {
230             return text_email.getText();
231         }
232
233         public String getTelephone() {
234             return text_tele.getText();
235         }
236
237         public double getSalaire() {
238             return Double.parseDouble(text_salaire.getText());
239         }
240
241         public Role getRole() {
242             return (Role) roleComboBox.getSelectedItem();
243         }
244
245         public Post getPoste() {
246             return (Post) posteComboBox.getSelectedItem();
247         }
248
249         public JButton getaddButton_employe () {
250             return addButton_employe;
251         }
252
253         public JButton getupdateButton_employe () {
254             return updateButton_employe;
255         }
256
257         public JButton getdeleteButton_employe () {
258             return deleteButton_employe;
259         }
260
261         public JButton getdisplayButton_employe () {
262             return displayButton_employe;
263         }
264
265         public JButton getaddButton_holiday () {
266             return addButton_holiday;
267         }
268
269         public JButton getupdateButton_holiday () {
270             return updateButton_holiday;
```

```
271         }
272         public JButton getdeleteButton_holiday () {
273             return deleteButton_holiday;
274         }
275 
276         public JButton getdisplayButton_holiday () {
277             return displayButton_holiday;
278         }
279         public String getStartDate () {
280             return text_startDate.getText();
281         }
282 
283         public String getEndDate() {
284             return text_endDate.getText();
285         }
286 
287         public Type_holiday getType_holiday(){
288             return (Type_holiday) TypeComboBox.getSelectedItem();
289         }
290 
291      // methods d'affichage des messages
292         public void afficherMessageErreur(String message) {
293             JOptionPane.showMessageDialog(this, message, "Erreur",
    JOptionPane.ERROR_MESSAGE);
294         }
295 
296         public void afficherMessageSucces(String message) {
297             JOptionPane.showMessageDialog(this, message, "Succes",
    JOptionPane.INFORMATION_MESSAGE);
298         }
299      // methodes de vider les champs
300         public void viderChamps_em() {
301             text_nom.setText("");
302             text_prenom.setText("");
303             text_email.setText("");
304             text_tele.setText("");
305             text_salaire.setText("");
306             roleComboBox.setSelectedIndex(0);
307             posteComboBox.setSelectedIndex(0);
308         }
309         public void viderChamps_ho() {
310             text_startDate.setText("");
311             text_endDate.setText("");
312             TypeComboBox.setSelectedIndex(0);
313         }
314 
315      // methodes de remplir les champs
316         public void remplaireChamps_em (int id, String nom, String
    prenom, String email, String telephone, double salaire, Role role,
```

```
Post poste) {
317             text_nom.setText(nom);
318             text_prenom.setText(prenom);
319             text_email.setText(email);
320             text_tele.setText(telephone);
321             text_salaire.setText(String.valueOf(salaire));
322             roleComboBox.setSelectedItem(role);
323             posteComboBox.setSelectedItem(poste);
324         }
325
326         public void remplaireChamps_ho(int id_employe, String date_debut
, String date_fin, Type_holiday type) {
327             List<Employe> Employes = new Employemodel(new EmployeDAOimpl
()).displayEmploye();
328             text_employe.removeAllItems();
329             for (Employe elem : Employes) {
330                 if (elem.getId() == id_employe) {
331                     text_employe.addItem(elem.getId() + " - " + elem.
getNom()+" "+elem.getPrenom());
332                     text_employe.setSelectedItem(elem.getId() + " - " +
elem.getNom()+" "+elem.getPrenom());
333                 }
334             }
335             text_startDate.setText(date_debut);
336             text_endDate.setText(date_fin);
337             TypeComboBox.setSelectedItem(type);
338         }
339
340      // methodes de test des champs
341         public boolean testChampsVide_em (){
342             return text_nom.getText().equals("") || text_prenom.getText
().equals("") || text_email.getText().equals("") || text_tele.getText
().equals("") || text_salaire.getText().equals("");
343         }
344
345         public boolean testChampsVide_ho () {
346             return text_employe.getSelectedItem().equals("") ||
text_startDate.getText().equals("") || text_endDate.getText().equals(
"") || TypeComboBox.getSelectedItem().equals("");
347         }
348
349      public void exportData(DefaultTableModel model, String fileName) {
350         try (PrintWriter writer = new PrintWriter(new FileWriter(
fileName))) {
351             for (int i = 0; i < model.getColumnCount(); i++) {
352                 writer.print(model.getColumnName(i));
353                 if (i < model.getColumnCount() - 1) writer.print(",");
354             }
355             writer.println();
```

```
356             for (int i = 0; i < model.getRowCount(); i++) {
357                 for (int j = 0; j < model.getColumnCount(); j++) {
358                     writer.print(model.getValueAt(i, j));
359                     if (j < model.getColumnCount() - 1) writer.print(","
    );
360                 }
361                 writer.println();
362             }
363             JOptionPane.showMessageDialog(this, "Donn es  export es
    avec  succ s.");
364         } catch (Exception e) {
365             JOptionPane.showMessageDialog(this, "Erreur  lors  de  l'
    exportation :  " + e.getMessage(), "Erreur", JOptionPane.
    ERROR_MESSAGE);
366         }
367     }
368
369     public void importData(DefaultTableModel model, String fileName) {
370         try (BufferedReader reader = new BufferedReader(new FileReader(
    fileName))) {
371             model.setRowCount(0);
372             String line = reader.readLine();
373             while ((line = reader.readLine()) != null) {
374                 String[] data = line.split(",");
375                 model.addRow(data);
376             }
377             JOptionPane.showMessageDialog(this, "Donn es  import es
    avec  succ s.");
378         } catch (Exception e) {
379             JOptionPane.showMessageDialog(this, "Erreur  lors  de  l'
    importation :  " + e.getMessage(), "Erreur", JOptionPane.
    ERROR_MESSAGE);
380         }
381     }
382 }
```

1.4 Controller

Le contrôleur gère les actions de l'utilisateur. Il reçoit les événements de la vue, interagit avec le modèle pour effectuer des opérations, puis met à jour la vue en conséquence.

EmployeController

```
1 package Controller;
2
3 import Model.*;
4
5 import view.*;
```

```
6
7  import java.sql.Date;
8  import java.util.Calendar;
9  import java.util.List;
10
11 import javax.swing.table.DefaultTableModel;
12
13 public class EmployeController {
14
15     private final Employe_HolidayView View;
16     public static Employemodel model_employe ;
17     public static int id = 0;
18     public static int oldselectedrow = -1;
19     public static boolean test = false;
20     String nom = "";
21     String prenom = "";
22     String email = "";
23     String telephone = "";
24     double salaire = 0;
25     Role role = null;
26     Post poste = null;
27     int solde = 0;
28     boolean updatereussi = false;
29
30     public EmployeController(Employe_HolidayView view, Employemodel
    model) {
31         this.View = view;
32         this.model_employe = model;
33
34         View.getaddButton_employe().addActionListener(e -> addEmploye())
    ;
35         View.getdeleteButton_employe().addActionListener(e ->
    deleteEmploye());
36         View.getupdateButton_employe().addActionListener(e ->
    updateEmploye());
37         View.getdisplayButton_employe().addActionListener(e ->
    displayEmploye());
38         Employe_HolidayView.Tableau.getSelectionModel().
    addListSelectionListener(e -> updateEmployebyselect());
39      }
40
41
42
43     public void displayEmploye() {
44         List<Employe> Employes = model_employe.displayEmploye();
45         if(Employes.isEmpty()){
46             View.afficherMessageErreur("Aucun employe.");
47         }
48         DefaultTableModel tableModel = (DefaultTableModel)
```

```
Employe_HolidayView.Tableau.getModel();
49          tableModel.setRowCount(0);
50          for(Employe e : Employes){
51              tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.
    getPrenom(), e.getEmail(), e.getTelephone(), e.getSalaire(), e.
    getRole(), e.getPost(),e.getSolde()});
52          }
53          View.remplaire_les_employes();
54      }
55
56
57      // function of add Employe :
58
59      private void addEmploye() {
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
68          View.viderChamps_em();
69          boolean addreussi = model_employe.addEmploye(0,nom, prenom,
    email, telephone, salaire, role, poste ,25);
70
71          if(addreussi == true){
72              View.afficherMessageSucces("L'employe a bien ete ajoutee.");
73              displayEmploye();
74          }else{
75              View.afficherMessageErreur("L'employe n'a pas ete ajoutee.")
    ;
76          }
77      }
78
79
80
81      // function of delete Employe :
82
83      private void deleteEmploye(){
84          int selectedrow = Employe_HolidayView.Tableau.getSelectedRow();
85          if(selectedrow == -1){
86              View.afficherMessageErreur("Veuillez selectionner une ligne.
    ");
87          }else{
88              int id = (int) Employe_HolidayView.Tableau.getValueAt(
    selectedrow, 0);
89              if(model_employe.deleteEmploye(id)){
90                  View.afficherMessageSucces("L'employe a bien ete
```

```
supprimer.");
91                  displayEmploye();
92              }else{
93                  View.afficherMessageErreur("L'employe n'a pas ete
supprimer.");
94              }
95          }
96      }
97
98      // function of Update :
99
100     private void updateEmployebyselect(){
101         int selectedrow = Employe_HolidayView.Tableau.getSelectedRow();
102
103         if (selectedrow == -1) {
104             return;
105         }
106         try{
107             id = (int) Employe_HolidayView.Tableau.getValueAt(
selectedrow, 0);
108             nom = (String) Employe_HolidayView.Tableau.getValueAt(
selectedrow, 1);
109             prenom = (String) Employe_HolidayView.Tableau.getValueAt(
selectedrow, 2);
110             email = (String) Employe_HolidayView.Tableau.getValueAt(
selectedrow, 3);
111             telephone = (String) Employe_HolidayView.Tableau.getValueAt(
selectedrow, 4);
112             salaire = (double) Employe_HolidayView.Tableau.getValueAt(
selectedrow, 5);
113             role = (Role) Employe_HolidayView.Tableau.getValueAt(
selectedrow, 6);
114             poste = (Post) Employe_HolidayView.Tableau.getValueAt(
selectedrow, 7);
115             solde = (int) Employe_HolidayView.Tableau.getValueAt(
selectedrow, 8);
116             View.remplaireChamps_em(id, nom, prenom, email, telephone,
salaire, role, poste);
117             test = true;
118         }catch(Exception e){
119              View.afficherMessageErreur("Erreur lors de la recup ration
 des donn es");
120         }
121     }
122
123     private void updateEmploye(){
124         if (!test) {
125             View.afficherMessageErreur("Veuillez d'abord selectionner
une ligne a modifier.");
```

```
126             return;
127         }
128         try {
129             nom = View.getNom();
130             prenom = View.getPrenom();
131             email = View.getEmail();
132             telephone = View.getTelephone();
133             salaire = View.getSalaire();
134             role = View.getRole();
135             poste = View.getPoste();
136 
137             boolean updateSuccessful = model_employe.updateEmploye(id,
    nom, prenom, email, telephone, salaire, role, poste , solde);
138 
139             if (updateSuccessful) {
140                 test = false;
141                 View.afficherMessageSucces("L'employe a   t   modifi
    avec succ s.");
142                 displayEmploye();
143                 View.viderChamps_em();
144             } else {
145                 View.afficherMessageErreur("Erreur lors de la mise a
    jour de l'employe.");
146             }
147         } catch (Exception e) {
148 
149             View.afficherMessageErreur("Erreur lors de la mise a jour");
150         }
151      }
152 
153      public void resetSolde(){
154         Calendar now = Calendar.getInstance();
155         if(now.get(Calendar.DAY_OF_YEAR) == 1){
156             for (Employe employe : model_employe.displayEmploye()) {
157                 updateSolde(employe.getId(), 25);
158             }
159         }
160      }
161 
162      public static void updateSolde(int  id , int solde){
163         boolean updateSuccessful = model_employe.updateSolde(id, solde);
164      }
165 
166 }
```

1.5 MAIN

Main

```
1  package Main;
2
3
4  import Controller.EmployeController;
5  import Controller.HolidayController;
6  import Controller.LoginController;
7  import DAO.EmployeDAOimpl;
8  import DAO.HolidayDAOimpl;
9  import DAO.LoginDAOimpl;
10 import Model.Employemodel;
11 import Model.HolidayModel;
12 import Model.LoginModel;
13 import view.Employe_HolidayView;
14 import view.LoginView;
15
16 public class Main {
17     public static void main(String[] args) {
18
19         LoginDAOimpl loginDAO = new LoginDAOimpl();
20         EmployeDAOimpl employeDAO = new EmployeDAOimpl();
21         HolidayDAOimpl holidayDAO = new HolidayDAOimpl();
22
23
24         LoginModel loginModel = new LoginModel(loginDAO);
25         Employemodel employeModel = new Employemodel(employeDAO);
26         HolidayModel holidayModel = new HolidayModel(holidayDAO);
27
28
29         LoginView loginView = new LoginView();
30         Employe_HolidayView employeHolidayView = new Employe_HolidayView
       ();
31
32
33         new LoginController(loginView, loginModel);
34
35         // Show the login view
36         loginView.setVisible(true);
37
38         // Listen for login success to show the main application
39         loginView.addLoginListener(e -> {
40             if (loginModel.authenticate(loginView.getUsername(),
       loginView.getPassword())) {
41                 // Login successful
42                 loginView.setVisible(false); // Hide login view
43
```

```
44                  // Initialize controllers for employee and holiday
    management
45                  new EmployeController(employeHolidayView, employeModel);
46                  new HolidayController(employeHolidayView, holidayModel);
47
48                  // Show the main application view
49                  employeHolidayView.setVisible(true);
50              } else {
51                  // Login failed
52                  loginView.showError("Invalid username or password.
    Please try again.");
53              }
54          });
55      }
56  }
```

Résultats

1 Résultats de la partie View

La couche View représente l'interface utilisateur de l'application et permet l'interaction entre l'utilisateur et le système. Dans ce projet, l'interface a été conçue avec le framework Swing en Java, qui fournit des composants graphiques riches et personnalisables.

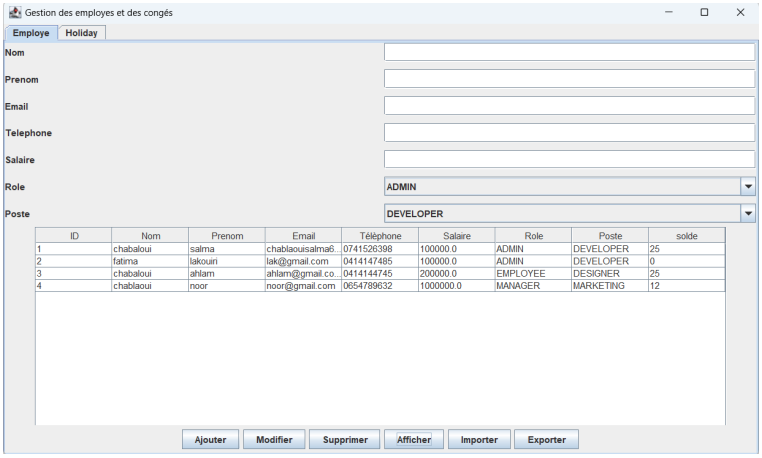

FIGURE 2.1 – Interface graphique des employes

2 Résultats du Bouton Importer

Le bouton Importer permet de charger les données des employés à partir d'un fichier "Importer.txt" . Cette fonctionnalité est utile pour intégrer rapidement un grand volume de données dans l'application sans les saisir manuellement. Lorsqu'il est cliqué, le bouton ouvre une boîte de dialogue permettant de sélectionner le fichier à importer. Les données sont ensuite analysées et ajoutées à la base de données après vérification de leur validité.

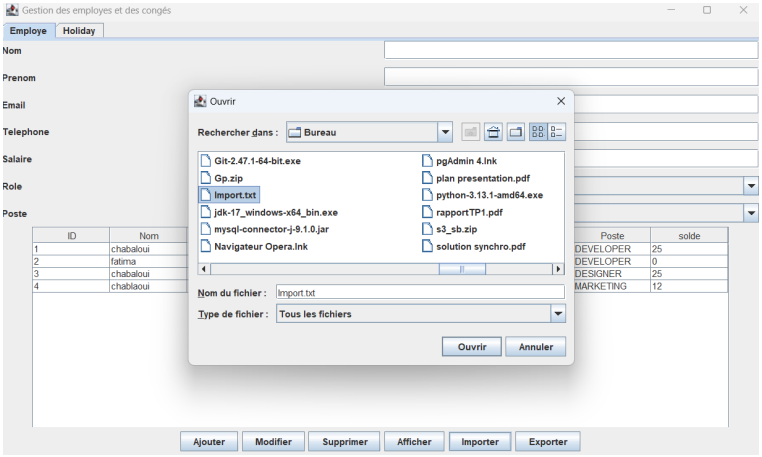

FIGURE 2.2 – Interface de l'importation du fichier

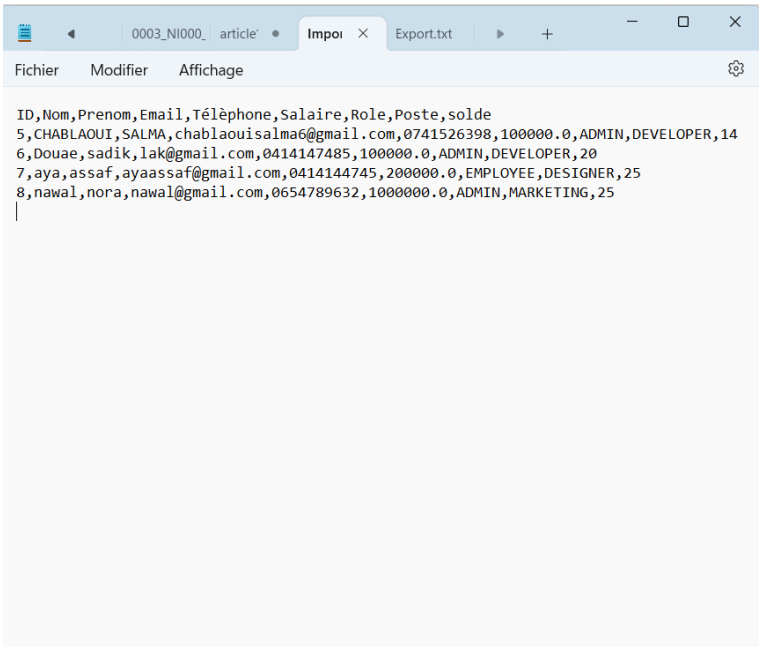

FIGURE 2.3 – Résultat de l'importation du fichier

3 Résultats du Bouton Exporter

Le bouton Exporter permet de sauvegarder les données des employés dans un fichier externe "exporter.txt" . Cette fonctionnalité est idéale pour générer des rapports, partager des données ou créer des sauvegardes. En cliquant sur le bouton, l'utilisateur peut choisir un emplacement pour enregistrer le fichier exporté, qui contient toutes les informations des employés.

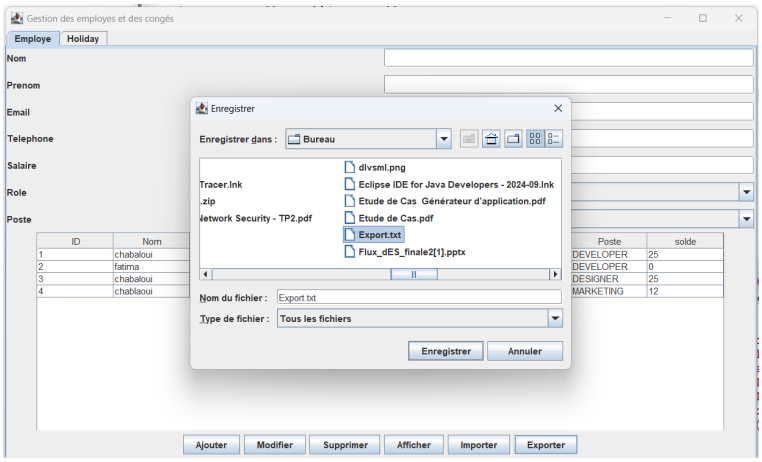

FIGURE 2.4 – Interface de l'exprtation du fichier

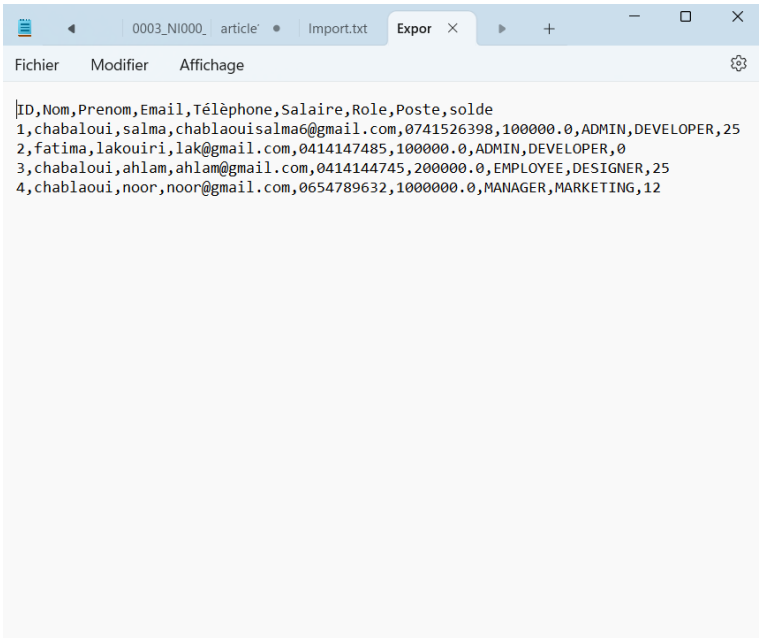

FIGURE 2.5 – Résultat de l'exportation du fichier

Conclusion générale

Ce travail pratique a permis de concevoir et de mettre en œuvre une application complète de gestion des entrées et sorties des employés, offrant des fonctionnalités essentielles adaptées aux besoins d'une entreprise. Parmi celles-ci, on retrouve l'importation des données des employés à partir de fichiers externes, l'exportation des données pour des besoins d'archivage ou de partage, et une gestion efficace des informations relatives aux employés.

L'application a été pensée pour garantir une utilisation intuitive, avec une interface graphique conviviale simplifiant les processus de gestion. Des validations ont été intégrées pour assurer l'exactitude des données importées, comme la vérification des formats des fichiers et des champs obligatoires. L'exportation permet de produire des fichiers bien structurés qui facilitent l'utilisation des données dans d'autres outils ou systèmes.

Ce projet nous a permis de développer nos compétences techniques dans plusieurs domaines. Nous avons consolidé nos connaissances en programmation orientée objet, notamment dans le cadre de l'architecture MVC, et nous avons approfondi notre maîtrise des bases de données, en les intégrant efficacement avec Java via JDBC. De plus, le développement d'une interface graphique avec Swing a renforcé notre aptitude à créer des applications interactives et conviviales.

En répondant à un besoin concret, ce travail pratique illustre l'importance des outils numériques pour optimiser la gestion des employés et constitue une expérience enrichissante pour notre formation professionnelle.

Références

java :

— https://www.java.com/en/download/
Eclipse :
— https://www.eclipse.org/downloads/
XAMPP :
— https://www.apachefriends.org/fr/index.html
jdk 23 :
— https://www.oracle.com/java/technologies/downloads/