

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 : Elajami Salma

Encadré par : 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'exportation la liste des employés	28
1.1 gestion des fichies au forme txt	29
2 L'importation	29
3 Conclusion générale	31
4 Références	32

Table des figures

1	intellij idea logo	5
2	MySQL Workbench logo	5
3	xampp logo	6
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 central en facilitant l'intégration et la sauvegarde des données de manière efficace.

Dans ce TP, nous allons explorer la manipulation des fichiers pour effectuer des opérations d'importation et d'exportation des données. L'objectif est de permettre au système de traiter des fichiers contenant des informations essentielles, telles que les données des employés ou des congés, afin de les intégrer dans la base de données ou de les exporter dans un format lisible et réutilisable.

Ces manipulations nécessitent la mise en œuvre de plusieurs étapes clés, notamment la lecture, l'écriture et la validation des fichiers, tout en garantissant une gestion rigoureuse des erreurs. Cela inclut la vérification des droits d'accès, le contrôle du format des fichiers et la gestion des exceptions pouvant survenir au cours de ces opérations.

Ces pratiques sont essentielles pour assurer la robustesse et la fiabilité du système, tout en offrant une interopérabilité efficace avec d'autres systèmes.

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_des_employs;
2 use gestion_des_employs;
3 -- Table Employee
4 CREATE TABLE employee (
5     id INT AUTO_INCREMENT PRIMARY KEY,
6     nom VARCHAR(50) NOT NULL,
7     prenom VARCHAR(50) NOT NULL,
8     salaire DECIMAL(10, 2) NOT NULL,
9     email VARCHAR(100) NOT NULL UNIQUE,
10    phone VARCHAR(15) NOT NULL,
11    role VARCHAR(200) not null,
12    poste VARCHAR(200) NOT NULL,
13    holidayBalance INTEGER DEFAULT 25
14 );
15
16 create table holiday (
17     id int auto_increment primary key,
18     employee_id int not null,
19     type varchar(200),
20     start varchar(10) not null,
21     end varchar(10) not null,
22     CONSTRAINT fk_employee FOREIGN KEY (employee_id) REFERENCES employee(id) ON delete
23     cascade
24 );
```

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

2.2.2 EmployeeDAOImpl

```
1 package DAO;
2
3 import java.io.BufferedReader;
4 import java.io.FileReader;
5 import java.sql.Connection;
6 import java.sql.PreparedStatement;
7 import java.sql.ResultSet;
8 import java.sql.SQLException;
9 import java.util.*;
10 import java.sql.*;
11 import java.io.*;
12
13 import Controller.EmployeeController;
14 import Model.Employee;
15 import Model.Poste;
16 import Model.Role;
17 import View.EmployeeView;
18
19 public class EmployeeDAOImpl implements EmployeeDAOI , GeneriqueDAOI<Employee>,
20    DataImportExport<Employee>{
21     private Connection connection;
```



```

21
22 public EmployeeDAOImpl() {
23     connection = DBConnection.getConnection();
24 }
25
26 @Override
27 public List<Employee> afficher() {
28     String SQL = "SELECT * FROM employee";
29     EmployeeController.viderLesChamps();
30     List<Employee> employees = new ArrayList<>();
31     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
32         try (ResultSet rset = stmt.executeQuery()) {
33             while (rset.next()) {
34                 int id = rset.getInt("id");
35                 String nom = rset.getString("nom");
36                 String prenom = rset.getString("prenom");
37                 double salaire = rset.getDouble("salaire");
38                 String email = rset.getString("email");
39                 String phone = rset.getString("phone");
40                 String role = rset.getString("role");
41                 String poste = rset.getString("poste");
42                 int holidayBalance = rset.getInt("holidayBalance");
43                 employees.add(new Employee(id, nom, prenom, salaire, email, phone,
Role.valueOf(role), Poste.valueOf(poste), holidayBalance));
44             }
45         }
46         catch (SQLException e) {
47             e.printStackTrace();
48         }
49         if (employees.isEmpty()) {
50             EmployeeView.AfficherFail("Aucun employé trouvé.");
51         }
52         return employees;
53     }
54     @Override
55     public void ajouter(Employee employee) {
56         String SQL = "INSERT INTO employee (nom, prenom, salaire, email, phone, role,
poste, holidayBalance) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
57         EmployeeController.viderLesChamps();
58         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
59             stmt.setString(1, employee.getNom());
60             stmt.setString(2, employee.getPrenom());
61             stmt.setDouble(3, employee.getSalaire());
62             stmt.setString(4, employee.getEmail());
63             stmt.setString(5, employee.getPhone());
64             stmt.setString(6, employee.getRole().name());
65             stmt.setString(7, employee.getPoste().name());
66             stmt.setInt(8, employee.getHolidayBalance());
67             stmt.executeUpdate();
68             EmployeeView.AjouterSuccess(employee);
69         } catch (SQLException e) {
70             e.printStackTrace();
71         }
72     }
73     @Override

```

```

74     public List<Employee> findByEmail(String email) {
75         String SQL = "SELECT * FROM employee WHERE email = ?";
76         EmployeeController.viderLesChamps();
77         List<Employee> employees = new ArrayList<Employee>();
78         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
79             stmt.setString(1, email);
80             try (ResultSet rset = stmt.executeQuery()) {
81                 while(rset.next()) {
82                     employees.add(new Employee(rset.getInt("id"), rset.getString("nom"),
83                         rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
84                         getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
85                         getString("poste")), rset.getInt("holidayBalance")));
86                 }
87             } catch (SQLException e) {
88                 e.printStackTrace();
89             }
90             if (employees.isEmpty()) {
91                 EmployeeView.AfficherFail("Aucun employé a été trouvé avec cet adresse
92                 Email.");
93             }
94             return employees;
95         }
96     }
97     @Override
98     public List<Employee> findByFullName(String firstname, String lastname) {
99         String SQL = "SELECT * FROM employee WHERE nom = ? AND prenom = ?";
100         EmployeeController.viderLesChamps();
101         List<Employee> employees = new ArrayList<Employee>();
102         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
103             stmt.setString(1, lastname);
104             stmt.setString(2, firstname);
105             try (ResultSet rset = stmt.executeQuery()) {
106                 while(rset.next()) {
107                     employees.add(new Employee(rset.getInt("id"), rset.getString("nom"),
108                         rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
109                         getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
110                         getString("poste")), rset.getInt("holidayBalance")));
111                 }
112             } catch (SQLException e) {
113                 e.printStackTrace();
114             }
115             if (employees.isEmpty()) {
116                 EmployeeView.AfficherFail("Aucun employé a été trouvé avec ce nom et
117                 prenom.");
118             }
119             return employees;
120         }
121     }
122     @Override
123     public List<Employee> findByFirstName(String firstname) {
124         String SQL = "SELECT * FROM employee WHERE prenom = ?";
125         List<Employee> employees = new ArrayList<Employee>();
126         EmployeeController.viderLesChamps();
127         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
128             stmt.setString(1, firstname);

```

```

121         try (ResultSet rset = stmt.executeQuery()) {
122             while(rset.next()) {
123                 employees.add(new Employee(rset.getInt("id"), rset.getString("nom"),
124                     rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
125                     getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
126                     getString("poste")), rset.getInt("holidayBalance")));
127             }
128         } catch (SQLException e) {
129             e.printStackTrace();
130         }
131         if (employees.isEmpty()) {
132             EmployeeView.AfficherFail("Aucun employé trouvé avec ce prénom.");
133         }
134         return employees;
135     }
136     @Override
137     public List<Employee> findByLastName(String lastname) {
138         String SQL = "SELECT * FROM employee WHERE nom = ?";
139         List<Employee> employees = new ArrayList<Employee>();
140         EmployeeController.viderLesChamps();
141         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
142             stmt.setString(1, lastname);
143             try (ResultSet rset = stmt.executeQuery()) {
144                 while(rset.next()) {
145                     employees.add(new Employee(rset.getInt("id"), rset.getString("nom"),
146                         rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
147                         getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
148                         getString("poste")), rset.getInt("holidayBalance")));
149                 }
150             } catch (SQLException e) {
151                 e.printStackTrace();
152             }
153             if (employees.isEmpty()) {
154                 EmployeeView.AfficherFail("Aucun employé trouvé avec ce nom.");
155             }
156             return employees;
157         }
158     }
159     @Override
160     public List<Employee> findByPhone(String phone) {
161         String SQL = "SELECT * FROM employee WHERE phone = ?";
162         List<Employee> employees = new ArrayList<Employee>();
163         EmployeeController.viderLesChamps();
164         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
165             stmt.setString(1, phone);
166             try (ResultSet rset = stmt.executeQuery()) {
167                 while(rset.next()) {
168                     employees.add(new Employee(rset.getInt("id"), rset.getString("nom"),
169                         rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
170                         getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
171                         getString("poste")), rset.getInt("holidayBalance")));
172                 }
173             } catch (SQLException e) {

```

```

167         e.printStackTrace();
168     }
169     if (employees.isEmpty()) {
170         EmployeeView.AfficherFail("Aucun employé a été trouvé avec ce numéro de
telephone.");
171     }
172     return employees;
173 }
174 @Override
175 public List<Employee> findBySalaire(double salaire) {
176     String SQL = "SELECT * FROM employee WHERE salaire = ?";
177     List<Employee> employees = new ArrayList<Employee>();
178     EmployeeController.viderLesChamps();
179     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
180         stmt.setDouble(1, salaire);
181         try (ResultSet rset = stmt.executeQuery()) {
182             while(rset.next()) {
183                 employees.add(new Employee(rset.getInt("id"), rset.getString("nom"),
rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
getString("poste")), rset.getInt("holidayBalance")));
184             }
185         }
186     } catch (SQLException e) {
187         e.printStackTrace();
188     }
189     if (employees.isEmpty()) {
190         EmployeeView.AfficherFail("Aucun employé a été trouvé avec ce salaire.");
191     }
192     return employees;
193 }
194 @Override
195 public Employee findById(int EmployeeId) {
196     String SQL = "SELECT * FROM employee WHERE id = ?";
197     Employee employee = null;
198     EmployeeController.viderLesChamps();
199     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
200         stmt.setInt(1, EmployeeId);
201         try (ResultSet rset = stmt.executeQuery()) {
202             if(rset.next()) {
203                 employee = new Employee(rset.getInt("id"), rset.getString("nom"),
rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
getString("poste")), rset.getInt("holidayBalance"));
204             }
205         } catch (SQLException e) {
206             e.printStackTrace();
207         }
208     } catch (SQLException e) {
209         e.printStackTrace();
210     }
211     return employee;
212 }
213 @Override

```

```

214     public void modifier(Employee employee, int EmployeeId) {
215         String SQL = "UPDATE employee SET nom = ?, prenom = ?, salaire = ?, email = ?,
phone = ?, role = ?, poste = ? WHERE id = ?";
216         EmployeeController.viderLesChamps();
217         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
218             stmt.setString(1, employee.getNom());
219             stmt.setString(2, employee.getPrenom());
220             stmt.setDouble(3, employee.getSalaire());
221             stmt.setString(4, employee.getEmail());
222             stmt.setString(5, employee.getPhone());
223             stmt.setString(6, employee.getRole().name());
224             stmt.setString(7, employee.getPoste().name());
225             stmt.setInt(8, EmployeeId);
226             stmt.executeUpdate();
227             EmployeeView.ModifierSuccess();
228         } catch (SQLException e) {
229             e.printStackTrace();
230         }
231     }
232
233     @Override
234     public void supprimer(int EmployeeId) {
235         String SQL = "DELETE FROM employee WHERE id = ?";
236         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
237             EmployeeController.viderLesChamps();
238             stmt.setInt(1, EmployeeId);
239             stmt.executeUpdate();
240             EmployeeView.SupprimerSuccess();
241         } catch (SQLException e) {
242             e.printStackTrace();
243         }
244     }
245
246     @Override
247     public void importData(String filePath){
248         String q="insert intro employee (prenom,nom,email,phone,role,poste,salaire)
values (?, ?, ?, ?, ?, ?, ?)";
249         try(BufferedReader reader= new BufferedReader(new FileReader(filePath));
250             PreparedStatement p=connection.prepareStatement(q)){
251             String line=reader.readLine();
252             while ((line=reader.readLine()) !=null){
253                 String[] data=line.split(",");
254                 if (data.length==5){
255                     p.setString(1,data[0].trim());
256                     p.setString(2,data[1].trim());
257                     p.setString(3,data[2].trim());
258                     p.setString(4,data[3].trim());
259                     p.setString(5,data[4].trim());
260                     p.setString(6,data[5].trim());
261                     p.setString(7,data[6].trim());
262
263                     p.addBatch();
264                 }
265             }
266             p.executeBatch();

```

```

267         System.out.println("Employees imported successfully!");
268     }catch (IOException|SQLException e){
269         e.printStackTrace();
270     }
271 }
272 @Override
273 public void exportData(String fileName,List<Employee> data ) throws IOException {
274     try(BufferedWriter writer=new BufferedWriter( new FileWriter(fileName))){
275         writer.write("prenom,nom,email,phone,rolr,poste,salaire");
276         writer.newLine();
277         for (Employee employee :data){
278             String line = String.format("%s,%s,%s,%s,%s,%s,%.2f",
279                 employee.getPrenom(),
280                 employee.getNom(),
281                 employee.getEmail(),
282                 employee.getPhone(),
283                 employee.getRole(),
284                 employee.getPoste(),
285                 employee.getSalaire());
286
287             writer.write(line);
288             writer.newLine();
289         }
290     }
291 }
292
293 }
294 }

```

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 EmployeeModel

```

1 package Model;
2
3 import java.util.List;
4 import java.io.File;

```

```

5 import DAO.EmployeeDAOImpl;
6 import Utilities.Utills;
7 import View.EmployeeView;
8 import DAO.*;
9 import java.lang.*;
10 import java.io.IOException;
11
12
13 public class EmployeeModel {
14     private EmployeeDAOImpl dao;
15     public EmployeeModel(EmployeeDAOImpl dao) {
16         this.dao = dao;
17     }
18
19     public void ajouterEmployee(String nom, String prenom, String salaire, String email,
20     String phone, Role role, Poste poste) {
21         double salaireDouble = Utills.parseDouble(salaire);
22         if(nom.trim().isEmpty() || prenom.trim().isEmpty() || email.trim().isEmpty() ||
23         phone.trim().isEmpty() || salaireDouble == 0) {
24             EmployeeView.AjouterFail("Veuillez remplir tous les champs.");
25             return;
26         }
27
28         if(!email.matches("^([a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,})$")) {
29             EmployeeView.AjouterFail("Veuillez entrer une adresse email valide.");
30             return;
31         }
32
33         if(!phone.matches("^0\\d{9}$")) {
34             EmployeeView.AjouterFail("Le num ro de t l phone doit contenir 10
35             chiffres");
36             return;
37         }
38
39         if(Utills.parseDouble(salaire) < 0 ){
40             EmployeeView.AjouterFail("Le salaire doit tre un nombre positif");
41             return;
42         }
43
44         Employee employee = new Employee(0, nom, prenom, salaireDouble, email, phone,
45         role, poste,25);
46         dao.ajouter(employee);
47     }
48     public List<Employee> afficherEmployee() {
49         return dao.afficher();
50     }
51     public List<Employee> findByEmail(String email) {
52         return dao.findByEmail(email);
53     }
54     public List<Employee> findByFullName(String firstname,String lastname) {
55         return dao.findByFullName(firstname,lastname);
56     }
57     public List<Employee> findByFirstName(String firstname) {
58         return dao.findByFirstName(firstname);
59     }
60 }

```

```

56     public List<Employee> findByLastName(String lastname) {
57         return dao.findByLastName(lastname);
58     }
59     public List<Employee> findByPhone(String phone) {
60         return dao.findByPhone(phone);
61     }
62     public List<Employee> findBySalaire(double salaire) {
63         return dao.findBySalaire(salaire);
64     }
65     public void supprimerEmployee(int id) {
66         if(EmployeeView.SupprimerConfirmation()){
67             dao.supprimer(id);
68         }
69         return;
70     }
71     public Employee findById(int id) {
72         return dao.findById(id);
73     }
74
75     public void updateEmployee(Employee employee,int id,String nom,String prenom,String
76     email,double salaire,String phone,Role role,Poste poste) {
77         if(nom.trim().isEmpty() && prenom.trim().isEmpty() && email.trim().isEmpty() &&
78     phone.trim().isEmpty() && salaire == 0 && role == null && poste == null) {
79             EmployeeView.ModifierFail("Veuillez remplir au moins un champ.");
80             return;
81         }
82         if(!nom.trim().isEmpty()) employee.setNom(nom);
83         if(!prenom.trim().isEmpty()) employee.setPrenom(prenom);
84         if(!email.trim().isEmpty()){
85             if(!email.matches("[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$")) {
86                 EmployeeView.ModifierFail("Veuillez entrer une adresse email valide.");
87                 return;
88             }
89             employee.setEmail(email);
90         }
91         if(salaire != 0) {
92             if(salaire < 0 ){
93                 EmployeeView.ModifierFail("Le salaire doit tre un nombre positif");
94                 return;
95             }
96             employee.setSalaire(salaire);
97         };
98         if(!phone.isEmpty()){
99             if(!phone.matches("^0\d{9}$")) {
100                 EmployeeView.ModifierFail("Le num ro de t l phone doit contenir 10
101 chiffres");
102                 return;
103             }
104             employee.setPhone(phone);
105         }
106         if(role != null) employee.setRole(role);
107         if(poste != null) employee.setPoste(poste);
108         dao.modifier(employee,id);
109     }

```



```

108
109
110
111     private boolean checkFileExists(File file){
112         if(!file.exists()){
113             throw new IllegalArgumentException("le fichier n'est pas existe:"+file.
getPath());
114         }
115         return true;}
116     private boolean checkIsFile(File file){
117         if (!file.isFile()){
118             throw new IllegalArgumentException("le chemin specifiee ne pas un fichier"+
file.getPath());
119         }
120         return true;
121     }
122     private boolean checkIsReadebal(File file){
123         if (!file.canRead()){
124             throw new IllegalArgumentException("le fichier n'est lisible "+file.getPath
());
125         } return true;
126     }
127     public void importData(String FileName)throws IOException {
128         File file = new File(FileName);
129         checkFileExists(file);
130         checkIsFile(file);
131         checkIsReadebal(file);
132         dao.importData(FileName);
133     }
134
135     public void exportData(String FileName , List<Employee> data) throws IOException {
136         File file = new File(FileName);
137         checkFileExists(file);
138         checkIsFile(file);
139         dao.exportData(FileName, data);
140     }
141 }

```

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 EmployeeView

```

1 package View;
2
3 import javax.swing.*;
4 import javax.swing.table.DefaultTableModel;
5
6 import Model.Employee;
7 import Model.Poste;
8 import Model.Role;
9 import java.awt.*;
10 public class EmployeeView extends JFrame {
11     protected static final EmployeeView INSTANCE = new EmployeeView();
12     protected JPanel General = new JPanel();
13     protected JPanel GeneralUp = new JPanel();
14     protected JPanel GeneralDown = new JPanel();
15     protected JPanel ListContainer = new JPanel();
16     protected JPanel ButtonsContainer = new JPanel();
17     protected DefaultTableModel tableModel = new DefaultTableModel(new String[]{"Id", "
Nom", "Prenom", "Email", "Salaire", "Phone", "Role", "Poste", "Holiday Balance"}, 0)
18     {
19         @Override
20         public boolean isCellEditable(int row, int column) {
21             return false;
22         }
23     };
24     protected JTable Tableau = new JTable(tableModel);
25     protected JButton Ajouter = new JButton("Ajouter");
26     protected JButton Modifier = new JButton("Modifier");
27     protected JButton Supprimer = new JButton("Supprimer");
28     protected JButton Afficher = new JButton("Afficher");
29     protected JButton importe = new JButton("import");
30     protected JButton export = new JButton("export");
31     protected JLabel NomLabel;
32     protected JTextField Nom;
33     protected JLabel PrenomLabel;
34     protected JTextField Prenom;
35     protected JLabel EmailLabel;
36     protected JTextField Email;
37     protected JLabel TelephoneLabel;
38     protected JTextField Telephone;
39     protected JLabel SalaireLabel;
40     protected JTextField Salaire;
41     protected JLabel RoleLabel;
42     protected JComboBox<Role> RoleComboBox;
43     protected JLabel PosteLabel;
44     protected JComboBox<Poste> PosteComboBox;

```

```

45 public EmployeeView() {
46     setTitle("Gestion des employe s");
47     setDefaultCloseOperation(EXIT_ON_CLOSE);
48     setSize(930, 520);
49     setLocationRelativeTo(null);
50     add(General);
51     General.setLayout(new BorderLayout());
52     General.add(GeneralUp, BorderLayout.NORTH);
53     General.add(GeneralDown, BorderLayout.CENTER);
54     GeneralUp.setLayout(new GridLayout(7,2));
55     GeneralUp.setBorder(BorderFactory.createEmptyBorder(10, 18, 10, 18));
56     NomLabel = new JLabel("Nom");
57     Nom = new JTextField();
58     GeneralUp.add(NomLabel);
59     GeneralUp.add(Nom);
60     PrenomLabel = new JLabel("Pre nom");
61     Prenom = new JTextField();
62     GeneralUp.add(PrenomLabel);
63     GeneralUp.add(Prenom);
64     EmailLabel = new JLabel("Email");
65     Email = new JTextField();
66     GeneralUp.add(EmailLabel);
67     GeneralUp.add(Email);
68     TelephoneLabel = new JLabel("Te le phone");
69     Telephone = new JTextField();
70     GeneralUp.add(TelephoneLabel);
71     GeneralUp.add(Telephone);
72     SalaireLabel = new JLabel("Salaire");
73     Salaire = new JTextField();
74     GeneralUp.add(SalaireLabel);
75     GeneralUp.add(Salaire);
76     RoleLabel = new JLabel("Role");
77     RoleComboBox = new JComboBox<>(Role.values());
78     GeneralUp.add(RoleLabel);
79     GeneralUp.add(RoleComboBox);
80     PosteLabel = new JLabel("Poste");
81     PosteComboBox = new JComboBox<>(Poste.values());
82     GeneralUp.add(PosteLabel);
83     GeneralUp.add(PosteComboBox);
84     GeneralDown.setLayout(new BorderLayout());
85     GeneralDown.add(ListContainer, BorderLayout.CENTER);
86     ListContainer.setLayout(new FlowLayout());
87     Dimension preferredSize = new Dimension(EmployeeView.this.getWidth() - 50,500);
88     Tableau.setPreferredScrollableViewportSize(preferredSize);
89     Tableau.setFillViewportHeight(true);
90     ListContainer.add(new JScrollPane(Tableau));
91     GeneralDown.add(ButtonsContainer, BorderLayout.SOUTH);
92     ButtonsContainer.setLayout(new FlowLayout());
93     ButtonsContainer.add(Ajouter);
94     ButtonsContainer.add(Modifier);
95     ButtonsContainer.add(Supprimer);
96     ButtonsContainer.add(Afficher);
97     ButtonsContainer.add(importe);
98     ButtonsContainer.add(export);
99

```

```

100     setVisible(true);
101 }
102 public static void AjouterSuccess(Employee employee){
103     JOptionPane.showMessageDialog(null, "L'employe a e te ajoute avec
succes");
104 }
105 public static void AjouterFail(String message){
106     JOptionPane.showMessageDialog(null, "L'employe n'a pas e te ajoute . " +
message);
107 }
108 public static void AfficherFail(String message){
109     JOptionPane.showMessageDialog(null, message);
110 }
111 public static void SupprimerSuccess(){
112     JOptionPane.showMessageDialog(null, "L'employe a bien te supprim.");
113 }
114 public static void SupprimerFail(String message){
115     JOptionPane.showMessageDialog(null, message);
116 }
117 public static void ModifierSuccess(){
118     JOptionPane.showMessageDialog(null, "L'employe a bien t modifi.");
119 }
120 public static void ModifierFail(String message){
121     JOptionPane.showMessageDialog(null, message);
122 }
123
124 public static void ImportSuccess(){
125     JOptionPane.showMessageDialog(null, "L'employe a bien t modifi.");
126 }
127 public static void ImportFail(String message){
128     JOptionPane.showMessageDialog(null, message);
129 }
130
131
132 public static void ExportSuccess(){
133     JOptionPane.showMessageDialog(null, "L'employe a bien t modifi.");
134 }
135 public static void ExportFail(String message){
136     JOptionPane.showMessageDialog(null, message);
137 }
138 protected void CacherColumn(int index){
139     Tableau.getColumnModel().getColumn(index).setMinWidth(0);
140     Tableau.getColumnModel().getColumn(index).setMaxWidth(0);
141     Tableau.getColumnModel().getColumn(index).setWidth(0);
142 }
143 public static boolean SupprimerConfirmation(){
144     int choice = JOptionPane.showOptionDialog(null, "tes -vous s r de supprimer
cet employe ?", "Confirmation", JOptionPane.YES_NO_OPTION, JOptionPane.
QUESTION_MESSAGE, null, new String[]{"Oui", "Non"}, "Non");
145     return choice == JOptionPane.YES_OPTION;
146 }
147 public JTable getTable() {
148     return Tableau;
149 }
150 public JButton getAjouterButton() {

```

```
151         return Ajouter;
152     }
153
154     public JButton getModifierButton() {
155         return Modifier;
156     }
157
158     public JButton getSupprimerButton() {
159         return Supprimer;
160     }
161     public JButton getImportButton() {
162         return importe;
163     }
164     public JButton getexportButton() {
165         return export;
166     }
167
168     public JButton getAfficherButton() {
169         return Afficher;
170     }
171
172     public JTextField getNomField() {
173         return Nom;
174     }
175
176     public void setNomField(JTextField nomField) {
177         Nom = nomField;
178     }
179
180     public JTextField getPrenomField() {
181         return Prenom;
182     }
183
184     public void setPrenomField(JTextField prenomField) {
185         Prenom = prenomField;
186     }
187
188     public JTextField getSalaireField() {
189         return Salaire;
190     }
191
192     public void setSalaireField(JTextField salaireField) {
193         Salaire = salaireField;
194     }
195
196     public JTextField getEmailField() {
197         return Email;
198     }
199
200     public void setEmailField(JTextField emailField) {
201         Email = emailField;
202     }
203
204     public JTextField getPhoneField() {
205         return Telephone;
```

```

206     }
207
208     public void setPhoneField(JTextField phoneField) {
209         Telephone = phoneField;
210     }
211
212     public JComboBox<Role> getRoleComboBox() {
213         return RoleComboBox;
214     }
215
216     public void setRoleComboBox(JComboBox<Role> roleComboBox) {
217         RoleComboBox = roleComboBox;
218     }
219
220     public JComboBox<Poste> getPosteComboBox() {
221         return PosteComboBox;
222     }
223
224     public void setPosteComboBox(JComboBox<Poste> posteComboBox) {
225         PosteComboBox = posteComboBox;
226     }
227     public static EmployeeView getInstance() {
228         return INSTANCE;
229     }
230     public void afficherMessageErreur(String message) {
231         JOptionPane.showMessageDialog(this, message, "Erreur", JOptionPane.ERROR_MESSAGE
232     );
233     }
234     public void afficherMessageSucces(String message) {
235         JOptionPane.showMessageDialog(this, message, "Succ s", JOptionPane.
236     INFORMATION_MESSAGE);
237     }
238 }

```

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 java.util.List;
4 import javax.swing.table.DefaultTableModel;
5 import Model.Employee;
6 import Model.EmployeeModel;
7 import Model.Poste;
8 import Model.Role;
9 import Utilities.Utills;
10 import View.EmployeeView;

```

```

11
12 public class EmployeeController {
13     protected EmployeeModel employeeModel;
14     protected static EmployeeView employeeView;
15     public EmployeeController(EmployeeModel employeeModel, EmployeeView employeeView) {
16         this.employeeModel = employeeModel;
17         EmployeeController.employeeView = employeeView;
18         EmployeeController.employeeView.getAjouterButton().addActionListener(e -> this.
ajouterEmployee());
19         EmployeeController.employeeView.getAfficherButton().addActionListener(e -> {
20             if (employeeView.getNomField().getText().isEmpty() && employeeView.
getPrenomField().getText().isEmpty() && employeeView.getSalaireField().getText().
isEmpty() && employeeView.getEmailField().getText().isEmpty() && employeeView.
getPhoneField().getText().isEmpty()) {
21                 this.afficherEmployee();
22             }
23             if (!employeeView.getNomField().getText().isEmpty() && !employeeView.
getPrenomField().getText().isEmpty()) {
24                 String firstname = employeeView.getNomField().getText();
25                 String lastname = employeeView.getPrenomField().getText();
26                 this.findByFullName(firstname, lastname);
27             }
28             if (employeeView.getNomField().getText().isEmpty() || employeeView.
getPrenomField().getText().isEmpty()) {
29                 if (!employeeView.getNomField().getText().isEmpty()) {
30                     String lastname = employeeView.getNomField().getText();
31                     this.findByLastName(lastname);
32                 }
33                 if (!employeeView.getPrenomField().getText().isEmpty()) {
34                     String firstname = employeeView.getPrenomField().getText();
35                     this.findByFirstName(firstname);
36                 }
37             }
38             if (!employeeView.getPhoneField().getText().isEmpty()) {
39                 String phone = employeeView.getPhoneField().getText();
40                 this.findByPhone(phone);
41             }
42             if (!employeeView.getEmailField().getText().isEmpty()) {
43                 String email = employeeView.getEmailField().getText();
44                 this.findByEmail(email);
45             }
46             if (!employeeView.getSalaireField().getText().isEmpty()) {
47                 String salaireString = employeeView.getSalaireField().getText();
48                 double salaire = Double.parseDouble(salaireString);
49                 this.findBySalaire(salaire);
50             }
51         });
52         EmployeeController.employeeView.getSupprimerButton().addActionListener(e -> this.
supprimerEmployee());
53         EmployeeController.employeeView.getModifierButton().addActionListener(e -> this.
updateEmployee());
54         this.afficherEmployee();
55     }
56     public void ajouterEmployee() {
57         String nom = employeeView.getNomField().getText();

```

```

58     String prenom = employeeView.getPrenomField().getText();
59     String salaire = employeeView.getSalaireField().getText();
60     String email = employeeView.getEmailField().getText();
61     String phone = employeeView.getPhoneField().getText();
62     Role role = (Role) employeeView.getRoleComboBox().getSelectedItem();
63     Poste poste = (Poste) employeeView.getPosteComboBox().getSelectedItem();
64     employeeModel.ajouterEmployee(nom, prenom, salaire, email, phone, role , poste);
65 }
66 public void afficherEmployee() {
67     List<Employee> employees = employeeModel.afficherEmployee();
68     DefaultTableModel tableModel = (DefaultTableModel) employeeView.getTable().
getModel();
69     tableModel.setRowCount(0);
70     for(Employee e : employees) {
71         tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.
getEmail(), e.getSalaire(), e.getPhone(), e.getRole(), e.getPoste(),e.
getHolidayBalance()});
72     }
73 }
74 public void findByEmail(String email) {
75     email = employeeView.getEmailField().getText();
76     List<Employee> employees = employeeModel.findByEmail(email);
77     DefaultTableModel tableModel = (DefaultTableModel) employeeView.getTable().
getModel();
78     tableModel.setRowCount(0);
79     for(Employee e : employees) {
80         tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.
getEmail(), e.getSalaire(), e.getPhone(), e.getRole(), e.getPoste(),e.
getHolidayBalance()});
81     }
82 }
83 public void findByFullName(String firstname, String lastname) {
84     firstname = employeeView.getPrenomField().getText();
85     lastname = employeeView.getNomField().getText();
86     List<Employee> employees = employeeModel.findByFullName(firstname,lastname);
87     DefaultTableModel tableModel = (DefaultTableModel) employeeView.getTable().
getModel();
88     tableModel.setRowCount(0);
89     for(Employee e : employees) {
90         tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.
getEmail(), e.getSalaire(), e.getPhone(), e.getRole(), e.getPoste(),e.
getHolidayBalance()});
91     }
92 }
93 public void findByFirstName(String firstname) {
94     firstname = employeeView.getPrenomField().getText();
95     List<Employee> employees = employeeModel.findByFirstName(firstname);
96     DefaultTableModel tableModel = (DefaultTableModel) employeeView.getTable().
getModel();
97     tableModel.setRowCount(0);
98     for(Employee e : employees) {
99         tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.
getEmail(), e.getSalaire(), e.getPhone(), e.getRole(), e.getPoste(),e.
getHolidayBalance()});
100    }

```



```

101     }
102     public void findByLastName(String lastname) {
103         lastname = employeeView.getNomField().getText();
104         List<Employee> employees = employeeModel.findByLastName(lastname);
105         DefaultTableModel tableModel = (DefaultTableModel) employeeView.getTable().
getModel();
106         tableModel.setRowCount(0);
107         for(Employee e : employees) {
108             tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.
getEmail(), e.getSalaire(), e.getPhone(), e.getRole(), e.getPoste(), e.
getHolidayBalance()});
109         }
110     }
111     public void findByPhone(String phone) {
112         List<Employee> employees = employeeModel.findByPhone(phone);
113         DefaultTableModel tableModel = (DefaultTableModel) employeeView.getTable().
getModel();
114         tableModel.setRowCount(0);
115         for(Employee e : employees) {
116             tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.
getEmail(), e.getSalaire(), e.getPhone(), e.getRole(), e.getPoste(), e.
getHolidayBalance()});
117         }
118     }
119     public void findBySalaire(double salaire) {
120         List<Employee> employees = employeeModel.findBySalaire(salaire);
121         DefaultTableModel tableModel = (DefaultTableModel) employeeView.getTable().
getModel();
122         tableModel.setRowCount(0);
123         for(Employee e : employees) {
124             tableModel.addRow(new Object[]{e.getId(), e.getNom(), e.getPrenom(), e.
getEmail(), e.getSalaire(), e.getPhone(), e.getRole(), e.getPoste(), e.
getHolidayBalance()});
125         }
126     }
127     public void supprimerEmployee() {
128         int selectedRow = employeeView.getTable().getSelectedRow();
129         if (selectedRow != -1) {
130             try {
131                 int id = Integer.parseInt(employeeView.getTable().getModel().getValueAt (
selectedRow, 0).toString());
132                 employeeModel.supprimerEmployee(id);
133             } catch (NumberFormatException e) {
134                 System.out.println("Invalid ID format.");
135             }
136         } else {
137             EmployeeView.SupprimerFail("Veuillez choisir un employé.");
138         }
139         this.afficherEmployee();
140     }
141     public void updateEmployee() {
142         int selectedRow = employeeView.getTable().getSelectedRow();
143         if (selectedRow != -1) {
144             try {
145                 int id = Integer.parseInt(employeeView.getTable().getModel().getValueAt (

```

```

146     selectedRow, 0).toString());
147     String nom = employeeView.getNomField().getText();
148     String prenom = employeeView.getPrenomField().getText();
149     String email = employeeView.getEmailField().getText();
150     double salaire = Utils.parseDouble(employeeView.getSalaireField().
151     getText());
152     String phone = employeeView.getPhoneField().getText();
153     Role role = (Role) (employeeView.getRoleComboBox().getSelectedItem());
154     Poste poste = (Poste) employeeView.getPosteComboBox().getSelectedItem();
155     Employee employeeToUpdate = employeeModel.findById(id);
156     if (employeeToUpdate != null) {
157         employeeModel.updateEmployee(employeeToUpdate, id, nom, prenom, email
158         , salaire, phone, role, poste);
159     } else {
160         EmployeeView.ModifierFail("L'employé avec l'ID spécifié n'existe
161         pas.");
162     }
163     } catch (NumberFormatException e) {
164         EmployeeView.ModifierFail("Erreur lors de la mise à jour de l'employ
165         é.");
166     }
167     }else{
168         EmployeeView.ModifierFail("Veuillez choisir un employé.");
169     }
170 }
171 public static int getId(){
172     int selectedRow = employeeView.getTable().getSelectedRow();
173     int id=-1;
174     if (selectedRow != -1) {
175         try {
176             id = Integer.parseInt(employeeView.getTable().getModel().getValueAt (
177             selectedRow, 0).toString());
178         } catch (NumberFormatException e) {
179             System.out.println("Invalid ID format.");
180         }
181     }
182     return id;
183 }
184 public static void viderLesChamps(){
185     EmployeeView employeeView = EmployeeView.getInstance();
186     employeeView.getNomField().setText("");
187     employeeView.getPrenomField().setText("");
188     employeeView.getSalaireField().setText("");
189     employeeView.getEmailField().setText("");
190     employeeView.getPhoneField().setText("");
191     employeeView.getRoleComboBox().setSelectedIndex(-1);
192     employeeView.getPosteComboBox().setSelectedIndex(-1);
193     return;
194 }
195 }

```

2.5.1 Main

```

1 import Controller.EmployeeController;
2 import Controller.HolidayController;

```

```
3 import DAO.EmployeeDAOImpl;
4 import DAO.HolidayDAOImpl;
5 import Model.EmployeeModel;
6 import Model.HolidayModel;
7 import View.PanelsView;
8 import View.EmployeeView;
9 import View.HolidayView;
10
11 public class Main {
12
13     public static void main(String[] args) {
14         EmployeeController employeeController = new EmployeeController(new
EmployeeModel(new EmployeeDAOImpl()), EmployeeView.getInstance());
15         HolidayController holidayController = new HolidayController(new HolidayModel
(new HolidayDAOImpl()), HolidayView.getInstance());
16         PanelsView.getInstance();
17     }
18 }
```

Resultat

1 L'exportation la liste des employés

Admin Dashboard - Gestion des Employés et Congés

Nom

Prénom

Email

Téléphone

Salaire

Role

Poste

Id	Nom	Prenom	Role	Poste	Holiday Balance
1	salma	salma	MANAGER	INGENIEUR_ET...	25
2	wafa	waa	MANAGER	PILOTE	25
3	adam	adam	MANAGER	TEAM_LEADER	25
4	nizar	nizar	MANAGER	TEAM_LEADER	25
5	nosair	nosair	ADMIN	INGENIEUR_ET...	25

Succès

Exportation réussie !

OK

Ajouter Modifier Supprimer Afficher import export

1.1 gestion des fichiers au forme txt

exfile.txt - Bloc-notes

Fichier Edition Format Affichage Aide

```

prenom,nom,email,phone,rolr,poste,salaire
salma,salma,salma@gmail.com,0678765434,MANAGER,INGENIEUR_ETUDE_ET_DEVELOPPEMENT,34455,00
waa,wafa,aaaa@gmail.com,0678987654,MANAGER,PILOTE,11111,00
adam,adam,adam@gmail.com,0567656754,MANAGER,TEAM_LEADER,123333,00
nizar,nizar,nizar@gmail.com,0645567654,MANAGER,TEAM_LEADER,12222,00
nosair,nosair,nosair@gmail.com,0745565456,ADMIN,INGENIEUR_ETUDE_ET_DEVELOPPEMENT,20000000,00

```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

2 L'importation

Admin Dashboard - Gestion des Employés et Congés

Nom

Prénom

Email

Téléphone

Salaire

Role

Poste

Id	Nom	Prenom	Role	Poste	Holiday Balance
1	salma	salma	MANAGER	INGENIEUR_ET...	25
2	wafa	waa	MANAGER	PILOTE	25
3	adam	adam	MANAGER	TEAM_LEADER	25
4	nizar	nizar	MANAGER	TEAM_LEADER	25
5	nosair	nosair	ADMIN	INGENIEUR_ET...	25

Succès

Importation réussie !

OK

Ajouter Modifier Supprimer Afficher import export

Admin Dashboard - Gestion des Employés et Congés

Gestion des Employés Gestion des Congés

Nom
 Prénom
 Email
 Téléphone
 Salaire
 Role
 Poste

Id	Nom	Prenom	Email	Salaire	Phone	Role	Poste	Holiday Balance
1	salma	salma	salma@gmail.co...	34455.0	0678765434	MANAGER	INGENIEUR_ET...	25
2	wafa	waa	aaaa@gmail.com	11111.0	0678987654	MANAGER	PILOTE	25
3	adam	adam	adam@gmail.co...	123333.0	0567656754	MANAGER	TEAM_LEADER	25
4	nizar	nizar	nizar@gmail.com	12222.0	0645567654	MANAGER	TEAM_LEADER	25
5	nosair	nosair	nosair@gmail.co...	2.0E7	0745565456	ADMIN	INGENIEUR_ET...	25
6	hakima	hakima	hakima@gmail.c...	100000.0	0621507547	ADMIN	PILOTE	25
7	amine	amine	amine@gmail.co...	1234.0	0610201020	ADMIN	TEAM_LEADER	25

Ajouter Modifier Supprimer Afficher import export

Conclusion générale

Ce TP nous a permis de découvrir et d'appliquer les principes fondamentaux de la manipulation des fichiers dans un contexte applicatif. À travers les opérations d'importation et d'exportation, nous avons appris à gérer efficacement des données externes, que ce soit en les intégrant dans notre système ou en les exportant vers des formats standardisés pour faciliter leur réutilisation.

Nous avons également mis en œuvre des concepts essentiels tels que la vérification des droits d'accès, le contrôle du format des fichiers et la gestion des exceptions, afin de garantir la robustesse et la fiabilité du système. Ces étapes sont cruciales pour offrir une expérience utilisateur optimale et assurer une interopérabilité harmonieuse entre différents systèmes.

En conclusion, ce TP a renforcé nos compétences en programmation orientée objet, notamment dans l'utilisation du modèle MVC et des DAO, tout en mettant en évidence l'importance d'une gestion rigoureuse des données. Ces acquis sont indispensables pour concevoir des applications performantes, évolutives et adaptées 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/>