

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

Rapport de Tp 2

Gestion de congés

Réalisé par : Annouar hakima

Encadré par : M. Ilhame

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 Model | 8 |
| 2.2 DAO | 16 |
| 2.3 Controller | 25 |
| 2.4 View | 31 |
| 2.5 Utilities | 38 |
| 2 Resultat | 39 |
| 1 Ajouter cangé | 39 |
| 2 Modifier cangé | 40 |
| 3 Supprimer cangé | 41 |
| 4 Afficher cangé | 42 |
| 5 Gestion des employes | 43 |
| 3 Conclusion générale | 44 |
| 4 Références | 45 |

Table des figures

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

Introduction

Dans le contexte actuel, où la gestion efficace des ressources humaines joue un rôle crucial dans la réussite des entreprises, le suivi et la gestion des congés des employés représentent un défi majeur pour de nombreuses organisations. L'entreprise SEA, confrontée à des difficultés liées à l'absence d'un système centralisé de gestion des congés, illustre parfaitement cette problématique. Actuellement, les informations relatives aux congés sont dispersées, mal organisées, et génèrent des inefficacités dans les processus de demande, d'approbation et de suivi.

Afin de surmonter ces obstacles, l'entreprise SEA a décidé de développer un module de gestion des congés intégré à son application existante de gestion des employés. Ce module vise à offrir une solution innovante et centralisée pour :

Permettre aux employés de soumettre leurs demandes de congés.

Donner aux managers la possibilité d'approuver ou de rejeter ces demandes.

En s'appuyant sur le modèle MVC et la notion de généricité, ce projet a pour objectif de développer une application avec une interface utilisateur Swing, répondant aux besoins spécifiques de l'entreprise. Ce rapport détaille le processus de conception et de mise en œuvre de cette solution, ainsi que les résultats obtenus.

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
    cascade
23 );
```

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 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 1 : Implémentation du Modèle (couche Model)

Création de la classe Holiday pour représenter les congés.

Ajout d'un constructeur et des méthodes getter et setter pour manipuler les attributs.

Introduction de l'énumération HolidayType pour définir les types de congés (Congé Payé, Non Payé, Maladie)

2.1.1 Holiday

```
1 package Model;
2
3 public class Holiday {
4     private int id;
5     private int idEmployee;
6     private HolidayType type;
7     private String start;
8     private String end;
9
10    public Holiday(int id, int idEmployee, HolidayType type, String start, String end) {
11        this.id = id;
12        this.idEmployee = idEmployee;
13        this.type = type;
14        this.start = start;
15        this.end = end;
16    }
17
18    public Holiday() {
19        this.id = 0;
20        this.idEmployee = 0;
21        this.type = null;
22        this.start = null;
23        this.end = null;
24    }
25
26    public int getId() {
27        return id;
28    }
29
30    public void setId(int id) {
31        this.id = id;
32    }
33
34    public int getIdEmployee() {
35        return idEmployee;
```



```
36     }
37
38     public void setIdEmployee(int idEmployee) {
39         this.idEmployee = idEmployee;
40     }
41
42     public HolidayType getType() {
43         return type;
44     }
45
46     public void setType(HolidayType type) {
47         this.type = type;
48     }
49
50     public String getStart() {
51         return start;
52     }
53
54     public void setStart(String start) {
55         this.start = start;
56     }
57
58     public String getEnd() {
59         return end;
60     }
61
62     public void setEnd(String end) {
63         this.end = end;
64     }
65 }
```

2.1.2 HolidayModel

```
1 package Model;
2
3 import java.time.LocalDate;
4 import java.time.format.DateTimeFormatter;
5 import java.time.temporal.ChronoUnit;
6 import java.util.List;
7
8 import DAO.HolidayDAOImpl;
9 import View.HolidayView;
10
11 public class HolidayModel {
12     private HolidayDAOImpl dao;
13     public HolidayModel(HolidayDAOImpl dao) {
14         this.dao = dao;
15     }
16
17     public List<Employee> afficherEmployee() {
18         return dao.afficherEmployee();
19     }
20     public List<Holiday> afficher() {
21         return dao.afficher();
22     }
23     public void ajouterHoliday(Holiday holiday, Employee employee) {
```

```
24     int days = calculateHolidayTime(holiday.getStart(), holiday.getEnd());
25     if (startCheck(holiday.getStart())) {
26         HolidayView.fail("La date de d but doit venir avant aujourd'hui.");
27         return;
28     }
29     if (days <= 0) {
30         HolidayView.fail("La date de fin doit venir apr s la date de d but.");
31         return;
32     }
33
34     if (employee.getHolidayBalance() >= days) {
35         employee.setHolidayBalance(employee.getHolidayBalance() - days);
36         dao.ajouter(holiday);
37         dao.modifierEmployeeBalance(employee, employee.getId());
38     } else {
39         HolidayView.fail("Le nombre de jours de conge s disponibles est insuffisant
40     .");
41     }
42
43     public boolean startCheck(String startDateString) {
44         LocalDate startDate = LocalDate.parse(startDateString, DateTimeFormatter.
45 ofPattern("yyyy-MM-dd"));
46         return startDate.isBefore(LocalDate.now());
47     }
48
49     public int calculateHolidayTime(String startDateString, String endDateString) {
50         LocalDate startDate = LocalDate.parse(startDateString, DateTimeFormatter.
51 ofPattern("yyyy-MM-dd"));
52         LocalDate endDate = LocalDate.parse(endDateString, DateTimeFormatter.ofPattern("
53 yyyy-MM-dd"));
54         return (int) ChronoUnit.DAYS.between(startDate, endDate);
55     }
56
57     public Employee FindById(int EmployeeId) {
58         return dao.findById(EmployeeId);
59     }
60
61     public void ModifierHoliday(Holiday updatedHoliday, Holiday oldHoliday) {
62         int newDays = calculateHolidayTime(updatedHoliday.getStart(), updatedHoliday.
63 getEnd());
64         int oldDays = calculateHolidayTime(oldHoliday.getStart(), oldHoliday.getEnd());
65         if (startCheck(updatedHoliday.getStart())) {
66             HolidayView.fail("La date de d but doit venir avant aujourd'hui.");
67             return;
68         }
69         if (newDays <= 0) {
70             HolidayView.fail("La date de fin doit venir apr s la date de d but.");
71             return;
72         }
73         Employee newEmployee = FindById(updatedHoliday.getIdEmployee());
74         Employee oldEmployee = FindById(oldHoliday.getIdEmployee());
75
76         if (newEmployee.getHolidayBalance() >= newDays) {
77             oldEmployee.setHolidayBalance(oldEmployee.getHolidayBalance() + oldDays);
78             dao.modifierEmployeeBalance(oldEmployee, oldEmployee.getId());
79         }
```

```

74         newEmployee.setHolidayBalance(newEmployee.getHolidayBalance() - newDays);
75         dao.modifierEmployeeBalance(newEmployee, newEmployee.getId());
76         dao.modifier(updatedHoliday, updatedHoliday.getId());
77     } else {
78         HolidayView.fail("Le nombre de jours de congés disponibles est insuffisant
pour le nouvel employé.");
79         return;
80     }
81 }
82
83 public void modifierEmployeeBalanceRecover(int days, int EmployeeId) {
84     Employee employee = this.findById(EmployeeId);
85     employee.setHolidayBalance(employee.getHolidayBalance() + days);
86     dao.modifierEmployeeBalance(employee, EmployeeId);
87 }
88
89 public Holiday FindHolidayById(int holidayId) {
90     return dao.FindHolidayById(holidayId);
91 }
92
93 public void supprimerHoliday(Holiday oldHoliday) {
94     int holidayId = oldHoliday.getId();
95     int oldDays = calculateHolidayTime(oldHoliday.getStart(), oldHoliday.getEnd());
96     Employee oldEmployee = findById(oldHoliday.getIdEmployee());
97     oldEmployee.setHolidayBalance(oldEmployee.getHolidayBalance() + oldDays);
98     dao.modifierEmployeeBalance(oldEmployee, oldEmployee.getId());
99     dao.supprimer(holidayId);
100 }
101 }

```

2.1.3 HolidayType

```

1 package Model;
2
3 public enum HolidayType {
4     CONGE_PAYE,
5     CONGE_NON_PAYE,
6     CONGE_MALADIE
7 }

```

2.1.4 Employee

```

1 package Model;
2
3 public class Employee {
4
5     private int id;
6     private String nom;
7     private String prenom;
8     private double salaire;
9     private String email;
10    private String phone;
11    private Role role;
12    private Poste poste;
13    private int holidayBalance;
14

```

```
15     public Employee(int id, String nom, String prenom, double salaire, String email,  
String phone, Role role, Poste poste,int holidayBalance) {  
16         this.id = id;  
17         this.nom = nom;  
18         this.prenom = prenom;  
19         this.salaire = salaire;  
20         this.email = email;  
21         this.phone = phone;  
22         this.role = role;  
23         this.poste = poste;  
24         this.holidayBalance=holidayBalance;  
25     }  
26  
27     public int getId() {  
28         return id;  
29     }  
30  
31     public void setId(int id) {  
32         this.id = id;  
33     }  
34  
35     public String getNom() {  
36         return nom;  
37     }  
38  
39     public void setNom(String nom) {  
40         this.nom = nom;  
41     }  
42  
43     public String getPrenom() {  
44         return prenom;  
45     }  
46  
47     public void setPrenom(String prenom) {  
48         this.prenom = prenom;  
49     }  
50  
51     public double getSalaire() {  
52         return salaire;  
53     }  
54  
55     public void setSalaire(double salaire) {  
56         this.salaire = salaire;  
57     }  
58  
59     public String getEmail() {  
60         return email;  
61     }  
62  
63     public void setEmail(String email) {  
64         this.email = email;  
65     }  
66  
67     public String getPhone() {  
68         return phone;
```

```
69     }
70
71     public void setPhone(String phone) {
72         this.phone = phone;
73     }
74
75     public Role getRole() {
76         return role;
77     }
78
79     public void setRole(Role role) {
80         this.role = role;
81     }
82
83     public Poste getPoste() {
84         return poste;
85     }
86
87     public void setPoste(Poste poste) {
88         this.poste = poste;
89     }
90
91     public int getHolidayBalance() {
92         return holidayBalance;
93     }
94
95     public void setHolidayBalance(int holidayBalance) {
96         this.holidayBalance = holidayBalance;
97     }
98 }
```

2.1.5 EmployeeModel

```
1 package Model;
2
3 import java.util.List;
4
5 import DAO.EmployeeDAOImpl;
6 import Utilities.Utills;
7 import View.EmployeeView;
8
9 public class EmployeeModel {
10     private EmployeeDAOImpl dao;
11     public EmployeeModel(EmployeeDAOImpl dao) {
12         this.dao = dao;
13     }
14
15     public void ajouterEmployee(String nom, String prenom, String salaire, String email,
16     String phone, Role role, Poste poste) {
17         double salaireDouble = Utills.parseDouble(salaire);
18         if(nom.trim().isEmpty() || prenom.trim().isEmpty() || email.trim().isEmpty() ||
19         phone.trim().isEmpty() || salaireDouble == 0) {
20             EmployeeView.AjouterFail("Veuillez remplir tous les champs.");
21             return;
22         }
23     }
```

```
22         if(!email.matches("^([a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,})$")) {
23             EmployeeView.AjouterFail("Veuillez entrer une adresse email valide.");
24             return;
25         }
26
27         if(!phone.matches("^0\\d{9}$")) {
28             EmployeeView.AjouterFail("Le num ro de t l phone doit contenir 10
chiffres");
29             return;
30         }
31
32         if(Utils.parseDouble(salaire) < 0 ){
33             EmployeeView.AjouterFail("Le salaire doit tre un nombre positif");
34             return;
35         }
36
37         Employee employee = new Employee(0, nom, prenom, salaireDouble, email, phone,
role, poste,25);
38         dao.ajouter(employee);
39     }
40     public List<Employee> afficherEmployee() {
41         return dao.afficher();
42     }
43     public List<Employee> findByEmail(String email) {
44         return dao.findByEmail(email);
45     }
46     public List<Employee> findByFullName(String firstname,String lastname) {
47         return dao.findByFullName(firstname,lastname);
48     }
49     public List<Employee> findByFirstName(String firstname) {
50         return dao.findByFirstName(firstname);
51     }
52     public List<Employee> findByLastName(String lastname) {
53         return dao.findByLastName(lastname);
54     }
55     public List<Employee> findByPhone(String phone) {
56         return dao.findByPhone(phone);
57     }
58     public List<Employee> findBySalaire(double salaire) {
59         return dao.findBySalaire(salaire);
60     }
61     public void supprimerEmployee(int id) {
62         if(EmployeeView.SupprimerConfirmation()){
63             dao.supprimer(id);
64         }
65         return;
66     }
67     public Employee findById(int id) {
68         return dao.findById(id);
69     }
70
71     public void updateEmployee(Employee employee,int id,String nom,String prenom,String
email,double salaire,String phone,Role role,Poste poste) {
72         if(nom.trim().isEmpty() && prenom.trim().isEmpty() && email.trim().isEmpty() &&
phone.trim().isEmpty() && salaire == 0 && role == null && poste == null) {
```

```

73     EmployeeView.ModifierFail("Veuillez remplir au moins un champ.");
74     return;
75 }
76 if(!nom.trim().isEmpty()) employee.setNom(nom);
77 if(!prenom.trim().isEmpty()) employee.setPrenom(prenom);
78 if(!email.trim().isEmpty()){
79     if(!email.matches("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$")) {
80         EmployeeView.ModifierFail("Veuillez entrer une adresse email valide.");
81         return;
82     }
83     employee.setEmail(email);
84 }
85 if(salaire != 0) {
86     if(salaire < 0 ){
87         EmployeeView.ModifierFail("Le salaire doit tre un nombre positif");
88         return;
89     }
90     employee.setSalaire(salaire);
91 };
92 if(!phone.isEmpty()){
93     if(!phone.matches("^0\d{9}$")) {
94         EmployeeView.ModifierFail("Le num ro de t l phone doit contenir 10
chiffres");
95         return;
96     }
97     employee.setPhone(phone);
98 }
99 if(role != null) employee.setRole(role);
100 if(poste != null) employee.setPoste(poste);
101 dao.modifier(employee,id);
102 }
103 }

```

2.1.6 Post

```

1 package Model;
2
3 public enum Poste {
4     INGENIEUR_ETUDE_ET_DEVELOPPEMENT,
5     TEAM_LEADER,
6     PILOTE
7 }

```

2.1.7 Role

```

1 package Model;
2
3 public enum Role {
4     ADMIN,
5     MANAGER,
6     EMPLOYEE
7 }

```

2.2 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). **2.2.1 DBConnection**

```
1 package DAO;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class DBConnection {
8     private static final String URL = "jdbc:mysql://localhost:3306/gestion_des_employs";
9     private static final String USER = "root";
10    private static final String PASSWORD = "";
11    private static Connection connection;
12
13    public static Connection getConnection() {
14        if (connection == null) {
15            try {
16                Class.forName("com.mysql.cj.jdbc.Driver");
17                connection = DriverManager.getConnection(URL, USER, PASSWORD);
18            } catch (ClassNotFoundException | SQLException e) {
19                e.printStackTrace();
20                throw new RuntimeException("Error lors de la connexion");
21            }
22        }
23        return connection;
24    }
25 }
```

2.2.2 EmployeeDAOI

```
1 package DAO;
2
3 import java.util.List;
4 import Model.Employee;
5
6 public interface EmployeeDAOI {
7     public List<Employee> findByFullName(String firstname, String lastname);
8     public List<Employee> findByEmail(String email);
9     public List<Employee> findByFirstName(String firstname);
10    public List<Employee> findByLastName(String lastname);
11    public List<Employee> findByPhone(String phone);
12    public List<Employee> findBySalaire(double salaire);
13 }
```

2.2.3 EmployeeDAOImpl

```
1 package DAO;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
```



```

5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.util.*;
8
9 import Controller.EmployeeController;
10 import Model.Employee;
11 import Model.Poste;
12 import Model.Role;
13 import View.EmployeeView;
14
15 public class EmployeeDAOImpl implements EmployeeDAOI , GeneriqueDAOI<Employee>{
16     private Connection connection;
17
18     public EmployeeDAOImpl() {
19         connection = DBConnection.getConnection();
20     }
21
22     @Override
23     public List<Employee> afficher() {
24         String SQL = "SELECT * FROM employee";
25         EmployeeController.viderLesChamps();
26         List<Employee> employees = new ArrayList<>();
27         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
28             try (ResultSet rset = stmt.executeQuery()) {
29                 while (rset.next()) {
30                     int id = rset.getInt("id");
31                     String nom = rset.getString("nom");
32                     String prenom = rset.getString("prenom");
33                     double salaire = rset.getDouble("salaire");
34                     String email = rset.getString("email");
35                     String phone = rset.getString("phone");
36                     String role = rset.getString("role");
37                     String poste = rset.getString("poste");
38                     int holidayBalance = rset.getInt("holidayBalance");
39                     employees.add(new Employee(id, nom, prenom, salaire, email, phone,
40 Role.valueOf(role), Poste.valueOf(poste), holidayBalance));
41                 }
42             } catch (SQLException e) {
43                 e.printStackTrace();
44             }
45             if (employees.isEmpty()) {
46                 EmployeeView.AfficherFail("Aucun employé trouvé.");
47             }
48             return employees;
49         }
50         @Override
51         public void ajouter(Employee employee) {
52             String SQL = "INSERT INTO employee (nom, prenom, salaire, email, phone, role,
53 poste, holidayBalance) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
54             EmployeeController.viderLesChamps();
55             try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
56                 stmt.setString(1, employee.getNom());
57                 stmt.setString(2, employee.getPrenom());
58                 stmt.setDouble(3, employee.getSalaire());

```

```

58         stmt.setString(4, employee.getEmail());
59         stmt.setString(5, employee.getPhone());
60         stmt.setString(6, employee.getRole().name());
61         stmt.setString(7, employee.getPoste().name());
62         stmt.setInt(8, employee.getHolidayBalance());
63         stmt.executeUpdate();
64         EmployeeView.AjouterSuccess(employee);
65     } catch (SQLException e) {
66         e.printStackTrace();
67     }
68 }
69 @Override
70 public List<Employee> findByEmail(String email) {
71     String SQL = "SELECT * FROM employee WHERE email = ?";
72     EmployeeController.viderLesChamps();
73     List<Employee> employees = new ArrayList<Employee>();
74     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
75         stmt.setString(1, email);
76         try (ResultSet rset = stmt.executeQuery()) {
77             while(rset.next()) {
78                 employees.add(new Employee(rset.getInt("id"), rset.getString("nom"),
79                     rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
80                     getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
81                     getString("poste")), rset.getInt("holidayBalance")));
82             }
83         } catch (SQLException e) {
84             e.printStackTrace();
85         }
86         if (employees.isEmpty()) {
87             EmployeeView.AfficherFail("Aucun employé trouvé avec cet adresse
88             Email.");
89         }
90         return employees;
91     }
92 }
93 @Override
94 public List<Employee> findByFullName(String firstname, String lastname) {
95     String SQL = "SELECT * FROM employee WHERE nom = ? AND prenom = ?";
96     EmployeeController.viderLesChamps();
97     List<Employee> employees = new ArrayList<Employee>();
98     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
99         stmt.setString(1, lastname);
100         stmt.setString(2, firstname);
101         try (ResultSet rset = stmt.executeQuery()) {
102             while(rset.next()) {
103                 employees.add(new Employee(rset.getInt("id"), rset.getString("nom"),
104                     rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
105                     getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.

```

```

106         EmployeeView.AfficherFail("Aucun employé trouvé avec ce nom et
prenom.");
107     }
108     return employees;
109 }
110 @Override
111 public List<Employee> findByFirstName(String firstname) {
112     String SQL = "SELECT * FROM employee WHERE prenom = ?";
113     List<Employee> employees = new ArrayList<Employee>();
114     EmployeeController.viderLesChamps();
115     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
116         stmt.setString(1, firstname);
117         try (ResultSet rset = stmt.executeQuery()) {
118             while(rset.next()) {
119                 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")));
120             }
121         }
122     } catch (SQLException e) {
123         e.printStackTrace();
124     }
125     if (employees.isEmpty()) {
126         EmployeeView.AfficherFail("Aucun employé trouvé avec ce prenom.");
127     }
128     return employees;
129 }
130 @Override
131 public List<Employee> findByLastName(String lastname) {
132     String SQL = "SELECT * FROM employee WHERE nom = ?";
133     List<Employee> employees = new ArrayList<Employee>();
134     EmployeeController.viderLesChamps();
135     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
136         stmt.setString(1, lastname);
137         try (ResultSet rset = stmt.executeQuery()) {
138             while(rset.next()) {
139                 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")));
140             }
141         }
142     } catch (SQLException e) {
143         e.printStackTrace();
144     }
145     if (employees.isEmpty()) {
146         EmployeeView.AfficherFail("Aucun employé trouvé avec ce nom.");
147     }
148     return employees;
149 }
150 @Override
151 public List<Employee> findByPhone(String phone) {
152     String SQL = "SELECT * FROM employee WHERE phone = ?";
153     List<Employee> employees = new ArrayList<Employee>();

```

```

154     EmployeeController.viderLesChamps();
155     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
156         stmt.setString(1, phone);
157         try (ResultSet rset = stmt.executeQuery()) {
158             while(rset.next()) {
159                 employees.add(new Employee(rset.getInt("id"), rset.getString("nom"),
160                     rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
161                     getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
162                     getString("poste")), rset.getInt("holidayBalance")));
163             }
164         } catch (SQLException e) {
165             e.printStackTrace();
166         }
167         if (employees.isEmpty()) {
168             EmployeeView.AfficherFail("Aucun employé a été trouvé avec ce numéro de
169             téléphone.");
170         }
171         return employees;
172     }
173     @Override
174     public List<Employee> findBySalaire(double salaire) {
175         String SQL = "SELECT * FROM employee WHERE salaire = ?";
176         List<Employee> employees = new ArrayList<Employee>();
177         EmployeeController.viderLesChamps();
178         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
179             stmt.setDouble(1, salaire);
180             try (ResultSet rset = stmt.executeQuery()) {
181                 while(rset.next()) {
182                     employees.add(new Employee(rset.getInt("id"), rset.getString("nom"),
183                         rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.
184                         getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
185                         getString("poste")), rset.getInt("holidayBalance")));
186                 }
187             } catch (SQLException e) {
188                 e.printStackTrace();
189             }
190             if (employees.isEmpty()) {
191                 EmployeeView.AfficherFail("Aucun employé a été trouvé avec ce salaire.");
192             }
193         }
194         return employees;
195     }
196     @Override
197     public Employee findById(int EmployeeId) {
198         String SQL = "SELECT * FROM employee WHERE id = ?";
199         Employee employee = null;
200         EmployeeController.viderLesChamps();
201         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
202             stmt.setInt(1, EmployeeId);
203             try (ResultSet rset = stmt.executeQuery()) {
204                 if(rset.next()) {
205                     employee = new Employee(rset.getInt("id"), rset.getString("nom"),
206                         rset.getString("prenom"), rset.getDouble("salaire"), rset.getString("email"), rset.

```

```

200     getString("phone"), Role.valueOf(rset.getString("role")), Poste.valueOf(rset.
201     getString("poste")), rset.getInt("holidayBalance"));
202     }
203     }catch(SQLException e) {
204         e.printStackTrace();
205     }
206     }catch(SQLException e) {
207         e.printStackTrace();
208     }
209     return employee;
210 }
211 @Override
212 public void modifier(Employee employee, int EmployeeId) {
213     String SQL = "UPDATE employee SET nom = ?, prenom = ?, salaire = ?, email = ?,
214     phone = ?, role = ?, poste = ? WHERE id = ?";
215     EmployeeController.viderLesChamps();
216     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
217         stmt.setString(1, employee.getNom());
218         stmt.setString(2, employee.getPrenom());
219         stmt.setDouble(3, employee.getSalaire());
220         stmt.setString(4, employee.getEmail());
221         stmt.setString(5, employee.getPhone());
222         stmt.setString(6, employee.getRole().name());
223         stmt.setString(7, employee.getPoste().name());
224         stmt.setInt(8, EmployeeId);
225         stmt.executeUpdate();
226         EmployeeView.ModifierSuccess();
227     } catch (SQLException e) {
228         e.printStackTrace();
229     }
230 }
231 @Override
232 public void supprimer(int EmployeeId) {
233     String SQL = "DELETE FROM employee WHERE id = ?";
234     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
235         EmployeeController.viderLesChamps();
236         stmt.setInt(1, EmployeeId);
237         stmt.executeUpdate();
238         EmployeeView.SupprimerSuccess();
239     } catch (SQLException e) {
240         e.printStackTrace();
241     }
242 }

```

2.2.4 GeneriqueDAOI

```

1 package DAO;
2
3 import java.util.List;
4
5 import Model.Employee;
6
7 public interface GeneriqueDAOI<T> {
8     public List<T> afficher();

```

```

9     public void ajouter(T t);
10    public void modifier(T t,int id);
11    public void supprimer(int id);
12    public Employee findById(int EmployeeId);
13 }

```

2.2.5 HolidayDAOImpl

```

1 package DAO;
2
3 import java.sql.*;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 import javax.management.relation.Role;
8
9 import Controller.EmployeeController;
10 import Model.Employee;
11 import Model.Holiday;
12 import Model.HolidayModel;
13 import Model.HolidayType;
14 import Model.Poste;
15 import View.EmployeeView;
16 import View.HolidayView;
17
18 public class HolidayDAOImpl implements GeneriqueDAOI<Holiday> {
19     private Connection connection;
20     public HolidayDAOImpl() {
21         connection = DBConnection.getConnection();
22     }
23     public List<Employee> afficherEmployee() {
24         List<Employee> employees = new ArrayList<>();
25         String query = "SELECT * FROM employee";
26
27         try (PreparedStatement statement = connection.prepareStatement(query);
28             ResultSet resultSet = statement.executeQuery()) {
29
30             while (resultSet.next()) {
31                 int id = resultSet.getInt("id");
32                 String nom = resultSet.getString("nom");
33                 String prenom = resultSet.getString("prenom");
34                 double salaire = resultSet.getDouble("salaire");
35                 String email = resultSet.getString("email");
36                 String phone = resultSet.getString("phone");
37                 Model.Role role = Model.Role.valueOf(resultSet.getString("role"));
38                 Model.Poste poste = Poste.valueOf(resultSet.getString("poste"));
39                 int holidayBalance = resultSet.getInt("holidayBalance");
40                 Employee employee = new Employee(id, nom, prenom, salaire, email, phone,
41 role, poste, holidayBalance);
42                 employees.add(employee);
43             }
44         } catch (SQLException e) {
45             e.printStackTrace();
46         }
47
48         return employees;

```

```

48     }
49     public List<Holiday> afficher(){
50         List<Holiday> holidays = new ArrayList<>();
51         String query = "SELECT * FROM holiday";
52
53         try (PreparedStatement statement = connection.prepareStatement(query);
54             ResultSet resultSet = statement.executeQuery()) {
55
56             while (resultSet.next()) {
57                 int id = resultSet.getInt("id");
58                 int employeeId = resultSet.getInt("employee_id");
59                 HolidayType type = HolidayType.valueOf(resultSet.getString("type"));
60                 String startDate = resultSet.getString("start");
61                 String endDate = resultSet.getString("end");
62                 Holiday holiday = new Holiday(id,employeeId,type, startDate, endDate);
63                 holidays.add(holiday);
64             }
65         } catch (SQLException e) {
66             e.printStackTrace();
67         }
68
69         return holidays;
70     }
71     @Override
72     public void ajouter(Holiday holiday) {
73         String SQL = "INSERT INTO holiday (employee_id, type, start, end) VALUES (?, ?,
74         ?, ?)";
75         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
76             stmt.setInt(1, holiday.getIdEmployee());
77             stmt.setString(2, holiday.getType().toString());
78             stmt.setString(3, holiday.getStart());
79             stmt.setString(4, holiday.getEnd());
80             stmt.executeUpdate();
81             HolidayView.success("Cong ajout avec succ ss !");
82         } catch (SQLException e) {
83             e.printStackTrace();
84         }
85
86     @Override
87     public void modifier(Holiday holiday,int holidayId) {
88         String SQL = "UPDATE holiday SET employee_id = ?, type = ?, start = ?, end = ?
89         WHERE id = ?";
90         try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
91             stmt.setInt(1, holiday.getIdEmployee());
92             stmt.setString(2, holiday.getType().toString());
93             stmt.setString(3, holiday.getStart());
94             stmt.setString(4, holiday.getEnd());
95             stmt.setInt(5, holidayId);
96             stmt.executeUpdate();
97             HolidayView.success("Cong modifi avec succ ss !");
98         } catch (SQLException e) {
99             e.printStackTrace();
100     }

```

```

101 public void modifierEmployeeBalance (Employee employee, int EmployeeId) {
102     String SQL = "UPDATE employee SET holidayBalance = ? WHERE id = ?";
103     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
104         stmt.setInt(1, employee.getHolidayBalance());
105         stmt.setInt(2, EmployeeId);
106         stmt.executeUpdate();
107     } catch (SQLException e) {
108         e.printStackTrace();
109     }
110 }
111 @Override
112 public void supprimer(int holidayId) {
113     String SQL = "DELETE FROM holiday WHERE id = ?";
114     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
115         stmt.setInt(1, holidayId);
116         stmt.executeUpdate();
117         HolidayView.success("Conge supprimé avec succès !");
118     } catch (SQLException e) {
119         e.printStackTrace();
120     }
121 }
122 @Override
123 public Employee findById(int EmployeeId) {
124     String SQL = "SELECT * FROM employee WHERE id = ?";
125     Employee employee = null;
126     EmployeeController.viderLesChamps();
127     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
128         stmt.setInt(1, EmployeeId);
129         try (ResultSet rset = stmt.executeQuery()) {
130             if(rset.next()) {
131                 int id = rset.getInt("id");
132                 String nom = rset.getString("nom");
133                 String prenom = rset.getString("prenom");
134                 double salaire = rset.getDouble("salaire");
135                 String email = rset.getString("email");
136                 String phone = rset.getString("phone");
137                 Model.Role role = Model.Role.valueOf(rset.getString("role"));
138                 Model.Poste poste = Poste.valueOf(rset.getString("poste"));
139                 int holidayBalance = rset.getInt("holidayBalance");
140                 employee = new Employee(id, nom, prenom, salaire, email, phone, role
141 , poste, holidayBalance);
142             }
143             }catch(SQLException e) {
144                 e.printStackTrace();
145             }
146         }catch(SQLException e) {
147             e.printStackTrace();
148         }
149     }
150     return employee;
151 }
152 public Holiday FindHolidayById(int holidayId) {
153     String SQL = "SELECT * FROM holiday WHERE id = ?";
154     Holiday holiday = null;
155     try (PreparedStatement stmt = connection.prepareStatement(SQL)) {
156         stmt.setInt(1, holidayId);

```



```

155         try (ResultSet rset = stmt.executeQuery()) {
156             if (rset.next()) {
157                 int id = rset.getInt("id");
158                 int idEmployee = rset.getInt("employee_id");
159                 Model.HolidayType type = Model.HolidayType.valueOf(rset.getString("
type"));
160                 String start = rset.getString("start");
161                 String end = rset.getString("end");
162                 holiday = new Holiday(id, idEmployee, type, start, end);
163             }
164         } catch (SQLException e) {
165             e.printStackTrace();
166         }
167     } catch (SQLException e) {
168         e.printStackTrace();
169     }
170     return holiday;
171 }
172 }

```

2.3 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.Utils;
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 employe.");
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 (
selectedRow, 0).toString());
146             String nom = employeeView.getNomField().getText();
147             String prenom = employeeView.getPrenomField().getText();
148             String email = employeeView.getEmailField().getText();
149             double salaire = Utils.parseDouble(employeeView.getSalaireField().
getText());
150             String phone = employeeView.getPhoneField().getText();
151             Role role = (Role) (employeeView.getRoleComboBox().getSelectedItem());
152             Poste poste = (Poste) employeeView.getPosteComboBox().getSelectedItem();
153             Employee employeeToUpdate = employeeModel.findById(id);
154             if (employeeToUpdate != null) {
155                 employeeModel.updateEmployee(employeeToUpdate,id, nom, prenom, email
, salaire, phone, role, poste);
156             } else {

```

```

157         EmployeeView.ModifierFail("L'employé avec l'ID spécifié n'existe
pas.");
158     }
159     } catch (NumberFormatException e) {
160         EmployeeView.ModifierFail("Erreur lors de la mise à jour de l'employé
.");
161     }
162     }else{
163         EmployeeView.ModifierFail("Veuillez choisir un employé.");
164     }
165 }
166 public static int getId(){
167     int selectedRow = employeeView.getTable().getSelectedRow();
168     int id=-1;
169     if (selectedRow != -1) {
170         try {
171             id = Integer.parseInt(employeeView.getTable().getModel().getValueAt (
selectedRow, 0).toString());
172         } catch (NumberFormatException e) {
173             System.out.println("Invalid ID format.");
174         }
175     }
176     return id;
177 }
178 public static void viderLesChamps(){
179
180     EmployeeView employeeView = EmployeeView.getInstance();
181     employeeView.getNomField().setText("");
182     employeeView.getPrenomField().setText("");
183     employeeView.getSalaireField().setText("");
184     employeeView.getEmailField().setText("");
185     employeeView.getPhoneField().setText("");
186     employeeView.getRoleComboBox().setSelectedIndex(-1);
187     employeeView.getPosteComboBox().setSelectedIndex(-1);
188     return;
189 }
190 }

```

2.2.6 HolidayController

```

1 package Controller;
2
3 import java.util.List;
4
5 import javax.swing.DefaultComboBoxModel;
6 import javax.swing.JComboBox;
7 import javax.swing.table.DefaultTableModel;
8
9 import Model.Employee;
10 import Model.Holiday;
11 import Model.HolidayModel;
12 import Model.HolidayType;
13 import View.HolidayView;
14
15 public class HolidayController {
16     private HolidayModel holidayModel;

```

```

17 private HolidayView holidayView;
18
19 public HolidayController(HolidayModel model, HolidayView view) {
20     this.holidayModel = model;
21     this.holidayView = view;
22     setEmployeesInComboBox();
23     holidayView.getAjouterButton().addActionListener(e -> this.ajouterHoliday());
24     holidayView.getAfficherButton().addActionListener(e -> this.afficherHoliday());
25     holidayView.getModifierButton().addActionListener(e -> this.ModifierHoliday());
26     holidayView.getSupprimerButton().addActionListener(e -> this.supprimerHoliday());
27
28 ;
29     this.afficherHoliday();
30 }
31
32 public void ajouterHoliday() {
33     JComboBox<String> nom = holidayView.getNomEmployeComboBox();
34     int Employeeid = Integer.parseInt(nom.getSelectedItem().toString().split(" - ")[0]);
35     HolidayType type = (HolidayType) holidayView.getTypeComboBox().getSelectedItem();
36
37 ;
38     String dateDebut = holidayView.getDateDebut();
39     String dateFin = holidayView.getDateFin();
40     Holiday holiday = new Holiday(1, Employeeid, type, dateDebut, dateFin);
41     Employee employee = holidayModel.FindById(Employeeid);
42     holidayModel.ajouterHoliday(holiday, employee);
43     this.afficherHoliday();
44 }
45
46 public void afficherHoliday() {
47     DefaultTableModel model = (DefaultTableModel) holidayView.getHolidayTable().
48 getModel();
49     Employee employee;
50     model.setRowCount(0);
51     List<Holiday> holidays = holidayModel.afficher();
52     for (Holiday holiday : holidays) {
53         employee = holidayModel.FindById(holiday.getIdEmployee());
54         model.addRow(new Object[]{holiday.getId(), employee.getNom() + " " +
55 employee.getPrenom(), holiday.getType(), holiday.getStart(), holiday.getEnd()});
56     }
57 }
58
59 public void ModifierHoliday() {
60     int selectedRow = holidayView.getTable().getSelectedRow();
61
62     if (selectedRow == -1) {
63         HolidayView.fail("Veuillez slectionner une ligne.");
64         return;
65     }
66
67     int idHoliday = Integer.parseInt(holidayView.getTable().getModel().getValueAt (
68 selectedRow, 0).toString());
69     Holiday oldHoliday = holidayModel.FindHolidayById(idHoliday);
70     Holiday updatedHoliday = new Holiday();
71     updatedHoliday.setId(idHoliday);
72     updatedHoliday.setIdEmployee(Integer.parseInt(holidayView.getNomEmployeComboBox
73 ().getSelectedItem().toString().split(" - ")[0]));
74     updatedHoliday.setType((HolidayType) holidayView.getTypeComboBox().

```

```

65     getSelectedItem());
66     updatedHoliday.setStart(holidayView.getDateDebut());
67     updatedHoliday.setEnd(holidayView.getDateFin());
68     holidayModel.ModifierHoliday(updatedHoliday, oldHoliday);
69     this.afficherHoliday();
70 }
71 public void supprimerHoliday() {
72     int selectedRow = holidayView.getTable().getSelectedRow();
73     if(selectedRow == -1) {
74         HolidayView.fail("Veuillez Slectionner une ligne.");
75         return;
76     }else{
77         int idHoliday = Integer.parseInt(holidayView.getTable().getModel().
78     getValueAt(selectedRow, 0).toString());
79         Holiday oldHoliday = holidayModel.FindHolidayById(idHoliday);
80         holidayModel.supprimerHoliday(oldHoliday);
81     }
82     this.afficherHoliday();
83 }
84 public void setEmployeesInComboBox() {
85     List<Employee> employees = holidayModel.afficherEmployee();
86     DefaultComboBoxModel<String> comboBoxModel = new DefaultComboBoxModel<>();
87     for (Employee e : employees) {
88         comboBoxModel.addElement(e.getId() + " - " + e.getNom() + " " + e.getPrenom
89     ());
90     }
91     holidayView.getNomEmployeComboBox().setModel(comboBoxModel);
92 }
93 }

```

2.4 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.

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         @Override
19         public boolean isCellEditable(int row, int column) {
20             return false;
21         }
22     };
23     protected JTable Tableau = new JTable(tableModel);
24     protected JButton Ajouter = new JButton("Ajouter");
25     protected JButton Modifier = new JButton("Modifier");
26     protected JButton Supprimer = new JButton("Supprimer");
27     protected JButton Afficher = new JButton("Afficher");
28     protected JLabel NomLabel;
29     protected JTextField Nom;
30     protected JLabel PrenomLabel;
31     protected JTextField Prenom;
32     protected JLabel EmailLabel;
33     protected JTextField Email;
34     protected JLabel TelephoneLabel;
35     protected JTextField Telephone;
36     protected JLabel SalaireLabel;
37     protected JTextField Salaire;
38     protected JLabel RoleLabel;
39     protected JComboBox<Role> RoleComboBox;
40     protected JLabel PosteLabel;
41     protected JComboBox<Poste> PosteComboBox;
42
43     public EmployeeView() {
44         setTitle("Gestion des employe s");
45         setDefaultCloseOperation(EXIT_ON_CLOSE);
46         setSize(930, 520);
47         setLocationRelativeTo(null);
48         add(General);
49         General.setLayout(new BorderLayout());
50         General.add(GeneralUp, BorderLayout.NORTH);
51         General.add(GeneralDown, BorderLayout.CENTER);
52         GeneralUp.setLayout(new GridLayout(7, 2));
53         GeneralUp.setBorder(BorderFactory.createEmptyBorder(10, 18, 10, 18));
54         NomLabel = new JLabel("Nom");
55         Nom = new JTextField();
56         GeneralUp.add(NomLabel);
57         GeneralUp.add(Nom);
58         PrenomLabel = new JLabel("Pre nom");
59         Prenom = new JTextField();
60         GeneralUp.add(PrenomLabel);
61         GeneralUp.add(Prenom);
62         EmailLabel = new JLabel("Email");
63         Email = new JTextField();
64         GeneralUp.add(EmailLabel);
65         GeneralUp.add(Email);
66         TelephoneLabel = new JLabel("Te le phone");

```



```

67     Telephone = new JTextField();
68     GeneralUp.add(TelephoneLabel);
69     GeneralUp.add(Telephone);
70     SalaireLabel = new JLabel("Salaire");
71     Salaire = new JTextField();
72     GeneralUp.add(SalaireLabel);
73     GeneralUp.add(Salaire);
74     RoleLabel = new JLabel("Role");
75     RoleComboBox = new JComboBox<>(Role.values());
76     GeneralUp.add(RoleLabel);
77     GeneralUp.add(RoleComboBox);
78     PosteLabel = new JLabel("Poste");
79     PosteComboBox = new JComboBox<>(Poste.values());
80     GeneralUp.add(PosteLabel);
81     GeneralUp.add(PosteComboBox);
82     GeneralDown.setLayout(new BorderLayout());
83     GeneralDown.add(ListContainer, BorderLayout.CENTER);
84     ListContainer.setLayout(new FlowLayout());
85     Dimension preferredSize = new Dimension(EmployeeView.this.getWidth() - 50, 500);
86     Tableau.setPreferredScrollableViewportSize(preferredSize);
87     Tableau.setFillViewportHeight(true);
88     ListContainer.add(new JScrollPane(Tableau));
89     GeneralDown.add(ButtonsContainer, BorderLayout.SOUTH);
90     ButtonsContainer.setLayout(new FlowLayout());
91     ButtonsContainer.add(Ajouter);
92     ButtonsContainer.add(Modifier);
93     ButtonsContainer.add(Supprimer);
94     ButtonsContainer.add(Afficher);
95     setVisible(true);
96 }
97 public static void AjouterSuccess(Employee employee){
98     JOptionPane.showMessageDialog(null, "L'employe a e te ajoute avec
succes");
99 }
100 public static void AjouterFail(String message){
101     JOptionPane.showMessageDialog(null, "L'employe n'a pas e te ajoute . " +
message);
102 }
103 public static void AfficherFail(String message){
104     JOptionPane.showMessageDialog(null, message);
105 }
106 public static void SupprimerSuccess(){
107     JOptionPane.showMessageDialog(null, "L'employe a bien te supprim.");
108 }
109 public static void SupprimerFail(String message){
110     JOptionPane.showMessageDialog(null, message);
111 }
112 public static void ModifierSuccess(){
113     JOptionPane.showMessageDialog(null, "L'employe a bien t modifi.");
114 }
115 public static void ModifierFail(String message){
116     JOptionPane.showMessageDialog(null, message);
117 }
118 protected void CacherColumn(int index){
119     Tableau.getColumnModel().getColumn(index).setMinWidth(0);

```

```
120     Tableau.getColumnModel().getColumn(index).setMaxWidth(0);
121     Tableau.getColumnModel().getColumn(index).setWidth(0);
122 }
123 public static boolean SupprimerConfirmation(){
124     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");
125     return choice == JOptionPane.YES_OPTION;
126 }
127 public JTable getTable() {
128     return Tableau;
129 }
130 public JButton getAjouterButton() {
131     return Ajouter;
132 }
133
134 public JButton getModifierButton() {
135     return Modifier;
136 }
137
138 public JButton getSupprimerButton() {
139     return Supprimer;
140 }
141
142 public JButton getAfficherButton() {
143     return Afficher;
144 }
145
146 public JTextField getNomField() {
147     return Nom;
148 }
149
150 public void setNomField(JTextField nomField) {
151     Nom = nomField;
152 }
153
154 public JTextField getPrenomField() {
155     return Prenom;
156 }
157
158 public void setPrenomField(JTextField prenomField) {
159     Prenom = prenomField;
160 }
161
162 public JTextField getSalaireField() {
163     return Salaire;
164 }
165
166 public void setSalaireField(JTextField salaireField) {
167     Salaire = salaireField;
168 }
169
170 public JTextField getEmailField() {
171     return Email;
172 }
```

```

173
174     public void setEmailField(JTextField emailField) {
175         Email = emailField;
176     }
177
178     public JTextField getPhoneField() {
179         return Telephone;
180     }
181
182     public void setPhoneField(JTextField phoneField) {
183         Telephone = phoneField;
184     }
185
186     public JComboBox<Role> getRoleComboBox() {
187         return RoleComboBox;
188     }
189
190     public void setRoleComboBox(JComboBox<Role> roleComboBox) {
191         RoleComboBox = roleComboBox;
192     }
193
194     public JComboBox<Poste> getPosteComboBox() {
195         return PosteComboBox;
196     }
197
198     public void setPosteComboBox(JComboBox<Poste> posteComboBox) {
199         PosteComboBox = posteComboBox;
200     }
201     public static EmployeeView getInstance() {
202         return INSTANCE;
203     }
204 }

```

2.3.2 HolidayView

```

1 package View;
2
3 import javax.swing.*;
4 import javax.swing.table.DefaultTableModel;
5 import Model.Employee;
6 import Model.HolidayType;
7 import Model.Role;
8
9 import java.awt.*;
10
11 public class HolidayView extends JFrame {
12     private static final HolidayView INSTANCE = new HolidayView();
13     private JPanel generalPanel = new JPanel();
14     private JLabel nomEmployeLabel = new JLabel("Nom de l'employé");
15     private JComboBox<String> nomEmployeComboBox = new JComboBox<>();
16     private JLabel typeLabel = new JLabel("Type");
17     private JComboBox<HolidayType> typeComboBox = new JComboBox<>(HolidayType.values());
18     private JLabel dateDebutLabel = new JLabel("Date de début");
19     private JTextField dateDebut = new JTextField("YYYY-MM-DD");
20     private JLabel dateFinLabel = new JLabel("Date de fin");
21     private JTextField dateFin = new JTextField("YYYY-MM-DD");

```

```
22     private DefaultTableModel tableModel = new DefaultTableModel(new String[]{"Id", "
Employ ", "Type", "Date d but ", "Date fin"}, 0){
23         @Override
24         public boolean isCellEditable(int row, int column) {
25             return false;
26         }
27     };
28     protected JTable holidayTable = new JTable(tableModel);
29     private JScrollPane tableScrollPane = new JScrollPane(holidayTable);
30     private JButton ajouterButton = new JButton("Ajouter");
31     private JButton modifierButton = new JButton("Modifier");
32     private JButton supprimerButton = new JButton("Supprimer");
33     private JButton afficherButton = new JButton("Afficher");
34     private JPanel inputPanel = new JPanel();
35     private JPanel buttonPanel = new JPanel();
36
37     public HolidayView() {
38         setTitle("Gestion des holidays");
39         setDefaultCloseOperation(EXIT_ON_CLOSE);
40         setSize(930, 520);
41         setLocationRelativeTo(null);
42         setVisible(true);
43
44         generalPanel.setLayout(new BorderLayout());
45         inputPanel.setLayout(new GridLayout(5, 2, 10, 10));
46         inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 0, 10));
47
48         inputPanel.add(nomEmployeLabel);
49         inputPanel.add(nomEmployeComboBox);
50         inputPanel.add(typeLabel);
51         inputPanel.add(typeComboBox);
52         inputPanel.add(dateDebutLabel);
53         inputPanel.add(dateDebut);
54         inputPanel.add(dateFinLabel);
55         inputPanel.add(dateFin);
56         generalPanel.add(inputPanel, BorderLayout.NORTH);
57
58         tableScrollPane.setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 10));
59         generalPanel.add(tableScrollPane, BorderLayout.CENTER);
60
61         buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 10));
62         buttonPanel.add(ajouterButton);
63         buttonPanel.add(modifierButton);
64         buttonPanel.add(supprimerButton);
65         buttonPanel.add(afficherButton);
66         generalPanel.add(buttonPanel, BorderLayout.SOUTH);
67
68         add(generalPanel);
69
70         holidayTable.setFillsViewportHeight(true);
71         holidayTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
72     }
73     public JComboBox<String> getNomEmployeComboBox() {
74         return nomEmployeComboBox;
75     }
```

```

76     public JComboBox<HolidayType> getTypeComboBox() {
77         return typeComboBox;
78     }
79     public String getDateDebut() {
80         return dateDebut.getText();
81     }
82     public String getDateFin() {
83         return dateFin.getText();
84     }
85     public JButton getAfficherButton() {
86         return afficherButton;
87     }
88     public JButton getAjouterButton() {
89         return ajouterButton;
90     }
91
92     public JButton getModifierButton() {
93         return modifierButton;
94     }
95
96     public JButton getSupprimerButton() {
97         return supprimerButton;
98     }
99     public JTable getHolidayTable() {
100         return holidayTable;
101     }
102     public static HolidayView getInstance() {
103         return INSTANCE;
104     }
105     public static void success(String message) {
106         JOptionPane.showMessageDialog(null, message, "Success", JOptionPane.
107         INFORMATION_MESSAGE);
108     }
109     public static void fail(String message) {
110         JOptionPane.showMessageDialog(null, message, "Error", JOptionPane.ERROR_MESSAGE)
111     };
112     public JTable getTable() {
113         return holidayTable;
114     }
115 }

```

2.3.3 PanelsView

```

1 package View;
2
3 import javax.swing.*;
4
5 public class PanelsView extends JFrame {
6     private static PanelsView INSTANCE = new PanelsView();
7     private JTabbedPane tabbedPane = new JTabbedPane();
8     private EmployeeView employeeView = EmployeeView.getInstance();
9     private HolidayView holidayView = HolidayView.getInstance();
10
11     public PanelsView() {

```

```
12     setTitle("Admin Dashboard - Gestion des Employ s et Cong s");
13     setDefaultCloseOperation(EXIT_ON_CLOSE);
14     setSize(930, 520);
15     setLocationRelativeTo(null);
16     employeeView.dispose();
17     holidayView.dispose();
18     tabbedPane.addTab("Gestion des Employ s", employeeView.getContentPane());
19     tabbedPane.addTab("Gestion des Cong s", holidayView.getContentPane());
20     add(tabbedPane);
21     setVisible(true);
22 }
23 public static PanelsView getInstance() {
24     return INSTANCE;
25 }
26 }
```

2.5 Utilities

2.4.1 Utils

```
1 package Utilities;
2
3 public class Utils {
4     public static double parseDouble(String s, double defaultValue) {
5         try {
6             return Double.parseDouble(s);
7         } catch (NumberFormatException e) {
8             return defaultValue;
9         }
10    }
11    public static double parseDouble(String s) {
12        return parseDouble(s, 0);
13    }
14 }
```

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 }
```

```
17     }  
18 }
```

Resultat

1 Ajouter cangé

Admin Dashboard - Gestion des Employés et Congés

Gestion des Employés

Gestion des Congés

Nom de l'employé

1 - annouar hakima

Type

CONGE_PAYE

Date de début

2024-12-29

Date de fin

2024-12-30

| Id | Employé | Date début | Date fin |
|----|----------------|------------|------------|
| 3 | laila elfanani | 2024-12-27 | 2024-12-30 |
| 4 | salma wafaa | 2024-12-29 | 2024-12-30 |

Success

i

Congé ajouté avec succès !

OK

Ajouter

Modifier

Supprimer

Afficher

2 Modifier cangé

Admin Dashboard - Gestion des Employés et Congés

Gestion des Employés Gestion des Congés

Nom de l'employé 1 - annouar hakima

Type CONGE_PAYE

Date de début 2024-12-29

Date de fin 2025-01-01

| Id | Employé | Date début | Date fin |
|----|----------------|------------|------------|
| 3 | laila elfanani | 2-27 | 2024-12-30 |
| 4 | salma wafaa | 2-29 | 2024-12-30 |
| 8 | annouar hakima | 2-29 | 2024-12-30 |

Success

Congé modifié avec succès !

OK

Ajouter Modifier Supprimer Afficher

3 Supprimer cangé

Admin Dashboard - Gestion des Employés et Congés

Gestion des Employés Gestion des Congés

Nom de l'employé 1 - annouar hakima

Type CONGE_PAYE

Date de début 2024-12-29

Date de fin 2025-01-01

| Id | Employé | Date début | Date fin |
|----|----------------|------------|------------|
| 3 | laila elfanani | 2-27 | 2024-12-30 |
| 4 | salma wafaa | 2-29 | 2024-12-30 |
| 8 | annouar hakima | 2-29 | 2025-01-01 |

Success

Congé supprimé avec succès !

OK

Ajouter Modifier Supprimer Afficher

4 Afficher cangé

Admin Dashboard - Gestion des Employés et Congés

Gestion des Employés

Gestion des Congés

Nom de l'employé

1 - annouar hakima

Type

CONGE_PAYE

Date de début

2024-12-29

Date de fin

2025-01-01

| Id | Employé | Type | Date début | Date fin |
|----|----------------|------------|------------|------------|
| 4 | salma wafaa | CONGE_PAYE | 2024-12-29 | 2024-12-30 |
| 8 | annouar hakima | CONGE_PAYE | 2024-12-29 | 2025-01-01 |

Ajouter

Modifier

Supprimer

Afficher

5 Gestion des employes

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 | annouar | hakima | hakima@gmail... | 1235.0 | 0654342189 | ADMIN | INGENIEUR_E... | 14 |
| 2 | laila | elfanani | layla@gmail.com | 1236.0 | 0789653423 | MANAGER | TEAM_LEADER | 25 |
| 4 | salma | wafaa | wafaa@gmail.c... | 3245.0 | 0612345678 | MANAGER | TEAM_LEADER | 19 |

Ajouter

Modifier

Supprimer

Afficher

Conclusion générale

Ce projet vise à résoudre les problèmes de gestion des congés rencontrés par l'entreprise SEA. En développant une application intégrée à leur système existant, l'objectif est d'améliorer l'efficacité et la transparence des processus liés aux congés des employés.

Les principales étapes du projet sont :

1. L'implémentation du modèle de données avec la classe Holiday, qui permettra de représenter les différents types de congés.
2. La gestion des données avec la couche DAO, notamment l'interface GenericDAO et l'implémentation HolidayDAOImpl.
3. La logique métier, avec la classe HolidayModel, pour gérer les règles de gestion des congés.
4. L'interface graphique, développée avec JTabbedPane, offrant une expérience utilisateur intuitive.
5. Le contrôleur, pour gérer les événements de l'interface.
6. L'initialisation de l'application dans la classe Main.

En suivant une architecture MVC et DAO, ce projet permettra à l'entreprise SEA de centraliser la gestion des congés, d'automatiser les processus et d'offrir une meilleure visibilité aux managers et aux ressources humaines. Cela contribuera à une gestion plus équitable et transparente des demandes de congés de la part des employés.

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/>