



المدرسة العليا  
للتكنولوجيا-اسفي  
ECOLE SUPÉRIEURE  
DE TECHNOLOGIE -SAFI

---

# Compte Rendu du TP

## Gestion d'employés et de congés

( MVC DAO SWING Généricités )

---

Réalisé par  
**Aya El Alama**

Décembre 2024

# Introduction

Ce projet a pour objectif de concevoir et de développer une solution logicielle intégrée permettant de simplifier la gestion des employés et de congés tout en garantissant une interface utilisateur intuitive.

L'application repose sur l'utilisation de **Swing**, une bibliothèque Java dédiée à la création d'interfaces graphiques riches et dynamiques. En parallèle, le modèle architectural **MVC** (Modèle-Vue-Contrôleur) est utilisé pour structurer l'application, favorisant ainsi une séparation claire des responsabilités entre la gestion des données, la logique métier, et la présentation.

Afin de garantir une flexibilité et une maintenabilité accrue, le projet intègre également la notion de **généricité**, permettant de travailler avec des types paramétrés et de minimiser la duplication de code. La gestion des données repose sur le modèle **DAO** (Data Access Object), qui fournit une couche d'abstraction pour les opérations sur la base de données, assurant ainsi une interaction cohérente et sécurisée avec les données stockées.

Ce projet vise à atteindre les objectifs suivants :

- **Faciliter la gestion des employés** : inclure des fonctionnalités telles que l'ajout, la modification, la suppression et l'affichage.
- **Simplifier la gestion des congés** : permettre d'ajouter, supprimer et afficher les congés de chaque employé.
- **Offrir une interface utilisateur conviviale** : garantir une expérience utilisateur intuitive et fluide grâce à l'utilisation de Swing.
- **Assurer une modularité et une maintenabilité** : grâce à l'architecture MVC et à la généricité, le code sera facilement extensible pour répondre à des besoins futurs.

Pour mieux comprendre l'architecture de l'application, nous présenterons dans un premier temps la structure du projet. Cela inclut la base de données utilisée pour gérer les informations des employés et des congés, ainsi que l'organisation des fichiers au sein de l'environnement de développement **Eclipse**.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> <b>employee</b>	★ Parcourir Structure Rechercher Insérer Vider Supprimer	4	InnoDB	utf8mb4_general_ci	16,0 kio	-
<input type="checkbox"/> <b>holiday</b>	★ Parcourir Structure Rechercher Insérer Vider Supprimer	3	InnoDB	utf8mb4_general_ci	32,0 kio	-

Figure 1: La base de données utilisée pour l'application.

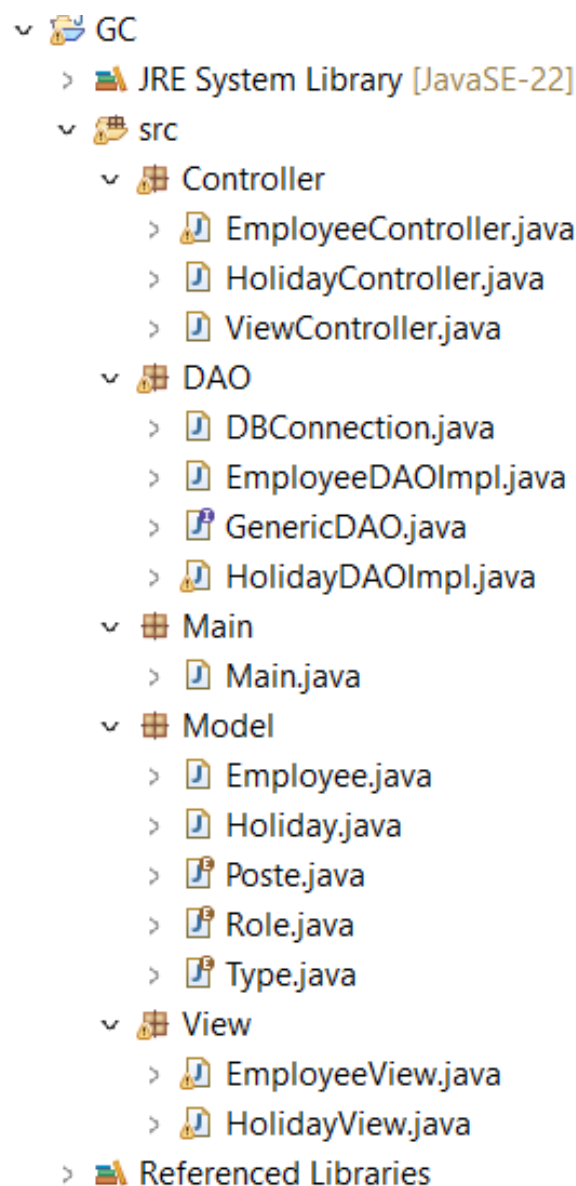


Figure 2: Organisation des fichiers du projet dans l'environnement Eclipse.

# Première Partie : Explication du Code

Dans cette première partie, nous expliquerons la structure des classes principales utilisées dans le projet, notamment celles définissant les employés (**Employee**), les rôles (**Role**) et les postes (**Poste**). Ce modèle suit une approche orientée objet en Java pour représenter les entités clés de l'application de gestion des employés et des congés.

## 1. Classe Employee

La classe **Employee** représente un employé avec les attributs suivants :

- **id** : identifiant unique de l'employé.
- **nom et prenom** : nom et prénom de l'employé.
- **email et phone** : coordonnées de l'employé.
- **salaire** : salaire de l'employé.
- **role** : rôle de l'employé (**ADMIN** ou **EMPLOYEE**).
- **poste** : poste occupé par l'employé (**INGENIEURE\_ETUDE\_ET\_DEVELOPPEMENT**, **TEAM\_LEADER**, ou **PILOTE**).

Le code de la classe est présenté ci-dessous :

```
package Model;

public class Employee {
    private int id;
    private String nom;
    private String prenom;
    private String email;
    private String phone;
    private double salaire;
    private Role role;
    private Poste poste;

    public Employee(String nom, String prenom, String email,
        String phone, double salaire, Role role, Poste poste)
    {

        this.nom = nom;
        this.prenom = prenom;
        this.email = email;
        this.phone = phone;
        this.salaire = salaire;
        this.role = role;
        this.poste = poste;
    }
}
```

```

public Employee() {

}

public int getId() { return id; }
public void setId(int id) { this.id = id; }
public String getNom() { return nom; }
public void setNom(String nom) { this.nom = nom; }
public String getPrenom() { return prenom; }
public void setPrenom(String prenom) { this.prenom =
    prenom; }
public String getEmail() { return email; }
public void setEmail(String email) { this.email = email;
}
public String getPhone() { return phone; }
public void setPhone(String phone) { this.phone = phone;
}
public double getSalaire() { return salaire; }
public void setSalaire(double salaire) { this.salaire =
    salaire; }
public Role getRole() { return role; }
public void setRole(Role role) { this.role = role; }
public Poste getPoste() { return poste; }
public void setPoste(Poste poste) { this.poste = poste; }

public void setnom(String nouveauNom) {
}
}

```

## 2. Enumération Role

Cette énumération définit les rôles possibles pour un employé :

- ADMIN : pour les administrateurs.
- EMPLOYE : pour les employés standards.

Le code correspondant est le suivant :

```

package Model;

public enum Role{
    ADMIN,
    EMPLOYE
}

```

### 3. Enumération Poste

Cette énumération décrit les postes occupés par les employés :

- `INGENIEURE_ETUDE_ET_DEVELOPPEMENT`.
- `TEAM_LEADER`.
- `PILOTE`.

Le code est présenté ci-dessous :

```
package Model;  
  
public enum Poste {  
    INGENIEURE_ETUDE_ET_DEVELOPPEMENT,  
    TEAM_LEADER,  
    PILOTE  
}
```

Ces classes et énumérations fournissent une base solide pour modéliser les données de l'application, facilitant ainsi la gestion des employés et des rôles au sein du système.

# Explication du code pour la gestion des congés

Dans cette section, nous allons expliquer le code qui gère les congés des employés. Le système utilise une classe appelée 'Holiday' et une énumération 'Type' pour organiser les informations sur les congés.

## 1. Classe Holiday

La classe 'Holiday' permet de gérer les informations relatives à un congé d'un employé, telles que l'identifiant du congé, l'employé associé, les dates de début et de fin du congé, et le type de congé.

Voici le code de la classe 'Holiday' :

```
package Model;

public class Holiday {
    private int id; // Identifiant du congé
    private int employeeId; // ID de l'employé
    private String employeeName; // Nom complet de l'employé
    private String startDate; // Date de début
    private String endDate; // Date de fin
    private Type type; // Type de congé (enum)

    // Constructeur avec employeeName pour listAll()
    public Holiday(int id, String employeeName, String
        startDate, String endDate, Type type) {
        this.id = id;
        this.employeeName = employeeName;
        this.startDate = startDate;
        this.endDate = endDate;
        this.type = type;
    }

    public Holiday(String employeeName, String startDate,
        String endDate, Type type) {
        this.employeeName = employeeName;
        this.startDate = startDate;
        this.endDate = endDate;
        this.type = type;
    }

    // Constructeur pour add() et update() (sans employeeName)
    public Holiday(int employeeId, String startDate, String
        endDate, Type type) {
        this.employeeId = employeeId;
        this.startDate = startDate;
        this.endDate = endDate;
        this.type = type;
    }
}
```

```

// Getters et Setters
public int getId() {
    return id;
}

public int getEmployeeId() {
    return employeeId;
}

public String getEmployeeName() {
    return employeeName; // Retourne le nom complet
}

public String getStartDate() {
    return startDate;
}

public String getEndDate() {
    return endDate;
}

public Type getType() {
    return type;
}

public void setEmployeeName(String employeeName) {
    this.employeeName = employeeName;
}
}

```

#### Explication :

- **Attributs** : La classe contient des attributs pour stocker l'ID du congé, l'ID de l'employé, le nom de l'employé, les dates de début et de fin du congé, et le type de congé.
- **Constructeurs** : Trois constructeurs sont définis : - Un constructeur pour initialiser tous les attributs. - Un autre pour les cas où l'ID de l'employé est donné sans le nom. - Un autre sans le nom complet de l'employé, utilisé pour les méthodes `add()` et `update()`.
- **Méthodes** : Des méthodes **getters** et **setters** sont fournies pour accéder et modifier les valeurs des attributs.

## 2. Énumération Type

L'énumération `Type` définit les trois types de congé qui peuvent être associés à un employé : maladie, payé et non payé.

Voici le code de l'énumération `Type` :

```

package Model;

public enum Type {

```



```
    CONGE_MALADIE ,  
    CONGE_PAYE ,  
    CONGE_NON_PAYE  
}
```

L'énumération **Type** permet de définir les trois types de congé disponibles dans le système : - 'CONGE MALADIE' pour les congés maladie. - 'CONGE PAYE' pour les congés payés. - 'CONGE NON PAYE' pour les congés non payés.

Ces deux morceaux de code permettent à l'application de gérer efficacement les congés des employés en utilisant une structure de données bien définie et des types distincts pour les congés.

# Explication du code pour la gestion de la connexion à la base de données

Dans cette section, nous allons expliquer le code de la classe `DBConnection`, qui gère la connexion à la base de données. Cette classe permet d'établir une connexion avec une base de données MySQL, dans le cadre de notre application de gestion des congés et des employés.

## Classe `DBConnection`

La classe `DBConnection` contient la logique nécessaire pour se connecter à une base de données MySQL. Elle inclut une méthode `getConnection()` qui permet d'obtenir une connexion active à la base de données.

Voici le code de la classe `DBConnection` :

```
package DAO;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/TP7";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    public static Connection getConnection() throws
        SQLException {
        return DriverManager.getConnection(URL, USER,
            PASSWORD);
    }
}
```

### Explication :

- **Attributs** : - **URL** : Contient l'URL de la base de données MySQL, y compris l'adresse du serveur (`localhost`) et le nom de la base de données (`TP7`). - **USER** : Le nom d'utilisateur utilisé pour se connecter à la base de données. Dans ce cas, l'utilisateur est `root`. - **PASSWORD** : Le mot de passe associé à l'utilisateur. Ici, il est vide (`""`), ce qui est souvent le cas dans un environnement de développement local.

- **Méthode `getConnection()`** : - Cette méthode est **static**, ce qui signifie qu'elle peut être appelée sans créer d'instance de la classe `DBConnection`. - La méthode utilise `DriverManager.getConnection(URL, USER, PASSWORD)` pour obtenir une connexion à la base de données MySQL. - Si la connexion échoue, une exception `SQLException` est lancée. Cette exception doit être gérée par le code appelant.

## Explication du code pour la gestion des employés

Dans cette section, nous allons expliquer le code de la classe `EmployeeDAOImpl`, qui implémente les méthodes de gestion des employés. Cette classe fournit des opérations CRUD (Create, Read, Update, Delete) pour interagir avec la base de données MySQL.

### Classe `EmployeeDAOImpl`

La classe `EmployeeDAOImpl` implémente l'interface `GenericDAO<Employee>`, ce qui signifie qu'elle définit des méthodes génériques pour la gestion des entités `Employee`. Ces méthodes incluent l'ajout, la suppression, la mise à jour, et la récupération des employés.

Voici le code de la classe `EmployeeDAOImpl` :

```
package DAO;

import Model.Employee;
import Model.Poste;
import Model.Role;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class EmployeeDAOImpl implements GenericDAO<Employee>
{
    @Override
    public void add(Employee employee) {
        String sql = "INSERT INTO Employee (nom, prenom, email, phone, salaire, role, poste) VALUES (?, ?, ?, ?, ?, ?, ?)";
        try (Connection conn = DBConnection.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setString(1, employee.getNom());
            stmt.setString(2, employee.getPrenom());
            stmt.setString(3, employee.getEmail());
            stmt.setString(4, employee.getPhone());
            stmt.setDouble(5, employee.getSalaire());
            stmt.setString(6, employee.getRole().name());
            stmt.setString(7, employee.getPoste().name());
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void delete(int id) {
```

```

        String sql = "DELETE FROM Employe WHERE id = ?";
        try (Connection conn = DBConnection.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public List<Employee> listAll() {
        List<Employee> employees = new ArrayList<>();
        String sql = "SELECT * FROM Employe";
        try (Connection conn = DBConnection.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                String roleStr = rs.getString("role").toUpperCase();
                String posteStr = rs.getString("poste").toUpperCase();

                Role role = null;
                Poste poste = null;
                try {
                    role = Role.valueOf(roleStr);
                } catch (IllegalArgumentException e) {
                    System.out.println("Role non valide : " + roleStr);
                    role = Role.EMPLOYE; // Valeur par d faut
                }

                try {
                    poste = Poste.valueOf(posteStr);
                } catch (IllegalArgumentException e) {
                    System.out.println("Poste non valide : " + posteStr);
                    poste = Poste.INGENIEURE_ETUDE_ET_DEVELOPPEMENT; // Valeur par d faut
                }

                Employee employee = new Employee(
                    rs.getString("nom"),
                    rs.getString("prenom"),

```

```

        rs.getString("email"),
        rs.getString("phone"),
        rs.getDouble("salaire"),
        role,
        poste
    );
    employee.setId(rs.getInt("id"));
    employees.add(employee);
}
} catch (SQLException e) {
    e.printStackTrace();
}
return employees;
}

@Override
public Employee findById(int id) {
    String sql = "SELECT * FROM Employe WHERE id = ?";
    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            return new Employee(
                rs.getString("nom"),
                rs.getString("prenom"),
                rs.getString("email"),
                rs.getString("phone"),
                rs.getDouble("salaire"),
                Role.valueOf(rs.getString("role")),
                Poste.valueOf(rs.getString("poste"))
            );
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}

@Override
public void update(Employee employee, int id) {
    String sql = "UPDATE Employe SET nom = ?, prenom = ?,
        email = ?, phone = ?, salaire = ?, role = ?,
        poste = ? WHERE id = ?";

    try (Connection conn = DBConnection.getConnection();

```

```

        PreparedStatement stmt = conn.prepareStatement(sql
    )) {
        stmt.setString(1, employee.getNom());
        stmt.setString(2, employee.getPrenom());
        stmt.setString(3, employee.getEmail());
        stmt.setString(4, employee.getPhone());
        stmt.setDouble(5, employee.getSalaire());
        stmt.setString(6, employee.getRole().name()); //
            Envoi du rôle en tant que chaîne (avec la
            méthode .name())
        stmt.setString(7, employee.getPoste().name()); //
            Idem pour le poste
        stmt.setInt(8, id); // L'ID de l'employé
            mettre à jour
        int rowsUpdated = stmt.executeUpdate();

        if (rowsUpdated > 0) {
            System.out.println("L'employé a été mis
                jour avec succès.");
        } else {
            System.out.println("Aucun employé trouvé
                avec cet ID.");
        }

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

```

## Explication du code

**Méthode add(Employee employee) :** - Cette méthode permet d'ajouter un employé à la base de données. - Elle utilise une requête SQL `INSERT INTO Employe` pour insérer les informations de l'employé dans la table `Employe`. - Les valeurs des différents champs (nom, prénom, email, etc.) sont extraites de l'objet `employee` et insérées dans la requête SQL via des `PreparedStatement`.

**Méthode delete(int id) :** - Cette méthode permet de supprimer un employé de la base de données en fonction de son `id`. - La requête SQL `DELETE FROM Employe WHERE id = ?` est exécutée avec l'ID de l'employé à supprimer.

**Méthode listAll() :** - Cette méthode récupère tous les employés de la base de données et les retourne sous forme de liste `List<Employee>`. - Elle exécute une requête SQL `SELECT * FROM Employe` et traite le `ResultSet` pour créer un objet `Employee` pour chaque ligne récupérée. - Les rôles et postes sont récupérés sous forme de chaînes de caractères et convertis en valeurs d'énumération `Role` et `Poste`.

**Méthode findById(int id) :** - Cette méthode permet de trouver un employé en particulier à partir de son `id`. - Elle exécute une requête SQL `SELECT * FROM Employe`

WHERE id = ? et retourne un objet `Employee` avec les informations de l'employé.

**Méthode** `update(Employee employee, int id)` : - Cette méthode met à jour les informations d'un employé dans la base de données. - La requête SQL `UPDATE Employe SET nom = ?, prenom = ?, email = ?, phone = ?, salaire = ?, role = ?, poste = ? WHERE id = ?` est utilisée. - Les informations de l'employé à mettre à jour sont passées dans la requête via des `PreparedStatement`.

## 1 Interface GenericDAO

L'interface `GenericDAO` définit un ensemble de méthodes génériques pour effectuer des opérations de gestion de données (CRUD - Créer, Lire, Mettre à jour, Supprimer) sur des entités de types différents. Cette interface utilise des génériques, ce qui permet de réutiliser les méthodes avec différents types d'entités. Voici une explication détaillée de chaque méthode définie dans cette interface :

```
package DAO;

import java.util.List;

public interface GenericDAO<T> {
    void add(T entity); // Ajouter un objet
    void delete(int id); // Supprimer un objet par ID
    List<T> listAll(); // Lister tous les objets
    T findById(int id); // Trouver un objet par ID
    void update(T entity, int id); // Mettre à jour un objet
}
```

### 1.1 Méthodes de l'interface

- **`add(T entity)`** : Cette méthode permet d'ajouter une nouvelle entité dans la base de données. Elle prend en paramètre un objet de type `T`, qui est le type générique de l'entité (par exemple, `Employee`, `Holiday`, etc.). Elle ne renvoie rien (`void`).
- **`delete(int id)`** : Cette méthode supprime une entité de la base de données en fonction de son identifiant (`id`). Elle prend un `id` de type `int` comme paramètre et ne renvoie rien (`void`).
- **`listAll()`** : Cette méthode retourne une liste de toutes les entités de type `T` dans la base de données. Elle renvoie une liste (`List<T>`) contenant toutes les entités correspondantes.
- **`findById(int id)`** : Cette méthode permet de rechercher une entité en fonction de son identifiant (`id`). Elle retourne l'entité trouvée de type `T`, ou `null` si aucune entité n'a été trouvée avec l'ID spécifié.
- **`update(T entity, int id)`** : Cette méthode met à jour une entité existante dans la base de données. Elle prend en paramètre l'entité (`T entity`) à mettre à jour et l'identifiant (`id`) de l'entité à mettre à jour. La méthode ne renvoie rien (`void`).

## 2 Implémentation de l'interface HolidayDAOImpl

La classe `HolidayDAOImpl` implémente l'interface `GenericDAO<Holiday>` et fournit des méthodes pour gérer les données des congés des employés dans une base de données relationnelle. Cette classe utilise JDBC pour exécuter des requêtes SQL pour effectuer des opérations de création, lecture, mise à jour et suppression (CRUD) sur les entités de type `Holiday`. Voici une explication détaillée de chaque méthode dans cette implémentation.

```
package DAO;

import Model.Holiday;
import Model.Type;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class HolidayDAOImpl implements GenericDAO<Holiday> {
    private static final String INSERT_HOLIDAY_SQL = "INSERT
        INTO holiday (employeeId, startDate, endDate, type)
        VALUES (?, ?, ?, ?)";
    private static final String DELETE_HOLIDAY_SQL = "DELETE
        FROM holiday WHERE id = ?";
    private static final String SELECT_ALL_HOLIDAY_SQL = "
        SELECT h.id, CONCAT(e.nom, ' ', e.prenom) AS
        employeeName, h.startDate, h.endDate, h.type FROM
        holiday h JOIN employe e ON h.employeeId = e.id";
    private static final String SELECT_HOLIDAY_BY_ID_SQL = "
        SELECT h.id, CONCAT(e.nom, ' ', e.prenom) AS
        employeeName, h.startDate, h.endDate, h.type FROM
        holiday h JOIN employe e ON h.employeeId = e.id WHERE
        h.id = ?";
    private static final String
        SELECT_EMPLOYEE_ID_BY_NAME_SQL = "SELECT id FROM
        employe WHERE CONCAT(nom, ' ', prenom) = ?";

    @Override
    public void add(Holiday holiday) {
        // Ajoute un congé à la base de données
        try (Connection conn = DBConnection.getConnection();
            PreparedStatement stmt = conn.prepareStatement(
                INSERT_HOLIDAY_SQL)) {
            int employeeId = getEmployeeIdByName(holiday.
                getEmployeeName());
            if (employeeId == -1) return;
            stmt.setInt(1, employeeId);
            stmt.setString(2, holiday.getStartDate());
            stmt.setString(3, holiday.getEndDate());
            stmt.setString(4, holiday.getType().name());
        }
```



```

        stmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Override
public void delete(int id) {
    // Supprime un congé de la base de données
    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(
            DELETE_HOLIDAY_SQL)) {
        stmt.setInt(1, id);
        stmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Override
public List<Holiday> listAll() {
    // Retourne la liste de tous les congés
    List<Holiday> holidays = new ArrayList<>();
    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(
            SELECT_ALL_HOLIDAY_SQL); ResultSet rs = stmt.
        executeQuery()) {
        while (rs.next()) {
            holidays.add(new Holiday(
                rs.getInt("id"),
                rs.getString("employeeName"),
                rs.getString("startDate"),
                rs.getString("endDate"),
                Type.valueOf(rs.getString("type"))
            ));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return holidays;
}

@Override
public Holiday findById(int id) {
    // Trouve un congé en fonction de son ID
    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(
            SELECT_HOLIDAY_BY_ID_SQL)) {

```

```

        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            return new Holiday(
                rs.getInt("id"),
                rs.getString("employeeName"),
                rs.getString("startDate"),
                rs.getString("endDate"),
                Type.valueOf(rs.getString("type"))
            );
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}

@Override
public void update(Holiday holiday, int id) {
    // Met jour un cong existant
    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement("
UPDATE holiday SET employeeId = ?, startDate = ?,
endDate = ?, type = ? WHERE id = ?")) {
        int employeeId = getEmployeeIdByName(holiday.
            getEmployeeName());
        if (employeeId == -1) return;
        stmt.setInt(1, employeeId);
        stmt.setString(2, holiday.getStartDate());
        stmt.setString(3, holiday.getEndDate());
        stmt.setString(4, holiday.getType().name());
        stmt.setInt(5, id);
        stmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public int getEmployeeIdByName(String employeeName) {
    // Retourne l'ID d'un employé en fonction de son nom
    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(
            SELECT_EMPLOYEE_ID_BY_NAME_SQL)) {
        stmt.setString(1, employeeName);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) return rs.getInt("id");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

```

    }
    return -1;
}

public int getTotalDaysTakenThisYear(String employeeName,
    int year) {
    // Calcule le nombre total de jours de cong pris
    // par un employé dans une année donnée
    int totalDays = 0;
    String query = ""
        SELECT SUM(DATEDIFF(endDate, startDate) + 1) AS
            totalDays
        FROM holiday
        WHERE employeeId = ?
        AND YEAR(startDate) = ?
        "";

    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(
            query)) {
        int employeeId = getEmployeeIdByName(employeeName);
        if (employeeId == -1) return 0;

        stmt.setInt(1, employeeId);
        stmt.setInt(2, year);

        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            totalDays = rs.getInt("totalDays");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return totalDays;
}

public List<String> getAllEmployeeNames() {
    // Retourne la liste des noms de tous les employés
    List<String> employeeNames = new ArrayList<>();
    String query = "SELECT CONCAT(nom, ' ', prenom) AS
        fullName FROM employe";
    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(
            query); ResultSet rs = stmt.executeQuery()) {
        while (rs.next()) employeeNames.add(rs.getString(
            "fullName"));
    } catch (SQLException e) {

```

```

        e.printStackTrace();
    }
    return employeeNames;
}

public List<Type> getAllHolidayTypes() {
    // Retourne tous les types de cong
    List<Type> holidayTypes = new ArrayList<>();
    for (Type type : Type.values()) holidayTypes.add(type);
    return holidayTypes;
}

public boolean hasOverlappingHoliday(String employeeName,
    String startDate, String endDate) {
    // V rifie si un employ e a un cong qui se
    chevauche avec les dates donn es
    String query = ""
        SELECT COUNT(*) AS overlapCount
        FROM holiday
        WHERE employeeId = ?
        AND (
            (startDate <= ? AND endDate >= ?) OR
            (startDate <= ? AND endDate >= ?) OR
            (startDate >= ? AND endDate <= ?)
        )
    "";

    try (Connection conn = DBConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(
            query)) {
        int employeeId = getEmployeeIdByName(employeeName);
        if (employeeId == -1) return false;

        stmt.setInt(1, employeeId);
        stmt.setString(2, startDate);
        stmt.setString(3, startDate);
        stmt.setString(4, endDate);
        stmt.setString(5, endDate);
        stmt.setString(6, startDate);
        stmt.setString(7, endDate);

        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            return rs.getInt("overlapCount") > 0;
        }
    } catch (SQLException e) {

```

```

        e.printStackTrace();
    }
    return false;
}
}

```

## 2.1 Méthodes de la classe

- **add(Holiday holiday)** : Cette méthode permet d'ajouter un congé pour un employé à la base de données. Elle récupère l'ID de l'employé à partir de son nom, puis insère un nouveau congé.
- **delete(int id)** : Cette méthode supprime un congé spécifique de la base de données en utilisant son identifiant unique.
- **listAll()** : Cette méthode retourne une liste de tous les congés, avec les informations sur l'employé, la date de début, la date de fin et le type de congé.
- **findById(int id)** : Cette méthode recherche un congé par son identifiant unique et retourne un objet `Holiday`.
- **update(Holiday holiday, int id)** : Cette méthode permet de mettre à jour les informations d'un congé existant, en modifiant les détails associés.
- **getEmployeeIdByName(String employeeName)** : Cette méthode permet d'obtenir l'ID d'un employé en fonction de son nom complet.
- **getTotalDaysTakenThisYear(String employeeName, int year)** : Cette méthode calcule le nombre total de jours de congé pris par un employé durant une année spécifique.
- **getAllEmployeeNames()** : Cette méthode retourne une liste des noms de tous les employés présents dans la base de données.
- **getAllHolidayTypes()** : Cette méthode retourne tous les types de congés définis dans l'enum `Type`.
- **hasOverlappingHoliday(String employeeName, String startDate, String endDate)** : Cette méthode vérifie si un employé a un congé qui se chevauche avec les dates données.

Cette classe permet de gérer efficacement les informations relatives aux congés des employés, en assurant la gestion des données dans la base de données à l'aide de méthodes génériques et adaptées aux besoins spécifiques de l'application.

## 3 Contrôleur des employés

Le contrôleur des employés (`EmployeeController`) est une classe centrale de l'application, qui gère l'interaction entre la vue (*View*) et le modèle (*Model*). Ce contrôleur permet d'effectuer les opérations principales telles que l'ajout, la suppression, la modification et la gestion des employés, tout en coordonnant les vues de gestion des employés et des congés.

### 3.1 Code du contrôleur des employés

Voici le code de la classe:

```
package Controller;

import DAO.GenericDAO;
import DAO.EmployeeDAOImpl;
import Model.Employee;
import Model.Poste;
import Model.Role;
import View.EmployeeView;
import View.HolidayView;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

public class EmployeeController {
    private final EmployeeView view;
    private final GenericDAO<Employee> dao;
    private final HolidayView holidayView;

    public EmployeeController(EmployeeView view, HolidayView
        holidayView) {
        this.view = view;
        this.dao = new EmployeeDAOImpl();
        this.holidayView = holidayView;

        // couteur pour le bouton Ajouter
        view.addButton.addActionListener(e -> addEmployee());

        // couteur pour le bouton Lister
        view.listButton.addActionListener(e -> listEmployees
            ());

        // couteur pour le bouton Supprimer
        view.deleteButton.addActionListener(e ->
            deleteEmployee());

        // couteur pour le bouton Modifier
        view.modifyButton.addActionListener(e ->
            modifyEmployee());

        // ActionListener pour le bouton "Gérer les Congés"
        view.switchViewButton.addActionListener(e -> {
            view.setVisible(false); // Cacher la vue des
```

```

        employ s
        holidayView.setVisible(true); // Afficher la vue
        des cong s
    });

    // Ajouter un couteur de s lection sur la table
    des employ s
    view.employeeTable.getSelectionModel().
    addListSelectionListener(e -> {
        if (!e.getValueIsAdjusting()) {
            int selectedRow = view.employeeTable.
            getSelectedRow();
            if (selectedRow != -1) {
                // R cup rer les donn es de la ligne
                s lectionn e
                int id = (int) view.employeeTable.
                getValueAt(selectedRow, 0);
                String nom = (String) view.employeeTable.
                getValueAt(selectedRow, 1);
                String prenom = (String) view.
                employeeTable.getValueAt(selectedRow,
                2);
                String email = (String) view.
                employeeTable.getValueAt(selectedRow,
                3);
                String phone = (String) view.
                employeeTable.getValueAt(selectedRow,
                4);
                double salaire = (double) view.
                employeeTable.getValueAt(selectedRow,
                5);
                Role role = Role.valueOf(view.
                employeeTable.getValueAt(selectedRow,
                6).toString());
                Poste poste = Poste.valueOf(view.
                employeeTable.getValueAt(selectedRow,
                7).toString());

                // Remplir les champs de modification
                avec les valeurs de la ligne
                s lectionn e
                view.nameField.setText(nom);
                view.surnameField.setText(prenom);
                view.emailField.setText(email);
                view.phoneField.setText(phone);
                view.salaryField.setText(String.valueOf(
                salaire));
                view.roleCombo.setSelectedItem(role.

```

```

        toString());
        view.posteCombo.setSelectedItem(poste.
            toString());

        // Sauvegarder l'ID de l'employé dans le
            bouton de modification pour une mise
                jour
        view.modifyButton.setActionCommand(String
            .valueOf(id));
    }
}

// Charger la liste des employés au démarrage
listEmployees();
}

// Méthode pour ajouter un employé avec validation
private void addEmployee() {
    try {
        String nom = view.nameField.getText().trim();
        String prenom = view.surnameField.getText().trim
            ();
        String email = view.emailField.getText().trim();
        String phone = view.phoneField.getText().trim();
        String salaireText = view.salaryField.getText().
            trim();

        // Validation des champs
        if (nom.isEmpty() || prenom.isEmpty() || email.
            isEmpty() || phone.isEmpty() || salaireText.
            isEmpty()) {
            JOptionPane.showMessageDialog(view, "Tous les
                champs sont obligatoires.");
            return;
        }
        if (!email.contains("@")) {
            JOptionPane.showMessageDialog(view, "Veuillez
                entrer une adresse email valide.");
            return;
        }
        double salaire = Double.parseDouble(salaireText);

        Role role = Role.valueOf(view.roleCombo.
            getSelectedItem().toString().toUpperCase());
        Poste poste = Poste.valueOf(view.posteCombo.
            getSelectedItem().toString().toUpperCase());
    }
}

```



```

        Employee employee = new Employee(nom, prenom,
            email, phone, salaire, role, poste);
        dao.add(employee);
        JOptionPane.showMessageDialog(view, "Employ
            ajout avec succ s.");
        listEmployees(); // Rafra chir la liste
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(view, "Salaire
            invalide.");
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(view, "Erreur: " +
            ex.getMessage());
    }
}

private void listEmployees() {
    List<Employee> employees = dao.listAll();
    String[] columnNames = {"ID", "Nom", "Pr nom", "
        Email", "T l phone", "Salaire", "R le", "Poste
        "};
    DefaultTableModel model = new DefaultTableModel(
        columnNames, 0);

    for (Employee emp : employees) {
        Object[] row = {emp.getId(), emp.getNom(), emp.
            getPrenom(), emp.getEmail(), emp.getPhone(),
            emp.getSalaire(), emp.getRole(), emp.getPoste
            ()};
        model.addRow(row);
    }

    view.employeeTable.setModel(model);
}

private void deleteEmployee() {
    try {
        // V rifier si une ligne est s lectionn e dans
        la table
        int selectedRow = view.employeeTable.
            getSelectedRow();
        if (selectedRow == -1) {
            JOptionPane.showMessageDialog(view, "Veuillez
                s lectionner un employ supprimer.");
            ;
            return;
        }

        // R cup rer l'ID de l'employ partir du

```

```

        mod le de table
int id = (int) view.employeeTable.getValueAt(
    selectedRow, 0); // Supposons que la colonne 0
    contient l'ID

// Demander confirmation avant de supprimer
int confirm = JOptionPane.showConfirmDialog(view,
    " tes -vous s r de vouloir supprimer l'
        employ avec l'ID " + id + " ?",
    "Confirmation de suppression",
    JOptionPane.YES_NO_OPTION);

if (confirm == JOptionPane.YES_OPTION) {
    // Supprimer l'employ via le DAO
    dao.delete(id);

    JOptionPane.showMessageDialog(view, "Employ
        supprim avec succ s.");
    listEmployees(); // Rafra chir la liste des
        employ s
} else {
    JOptionPane.showMessageDialog(view, "
        Suppression annul e.");
}
} catch (Exception ex) {
    JOptionPane.showMessageDialog(view, "Erreur : " +
        ex.getMessage());
}
}

private void modifyEmployee() {
    try {
        // R cup rer l'ID de l'employ partir du
            bouton de modification
        String actionCommand = view.modifyButton.
            getActionCommand();
        if (actionCommand != null && !actionCommand.trim
            ().isEmpty()) {
            int id = Integer.parseInt(actionCommand.trim
                ());

            String nom = view.nameField.getText().trim();
            String prenom = view.surnameField.getText().
                trim();
            String email = view.emailField.getText().trim
                ();
            String phone = view.phoneField.getText().trim
                ();
            String salaireText = view.salaryField.getText

```

```

        ().trim());

// Validation des champs
if (nom.isEmpty() || prenom.isEmpty() ||
    email.isEmpty() || phone.isEmpty() ||
    salaireText.isEmpty()) {
    JOptionPane.showMessageDialog(view, "Tous
        les champs sont obligatoires.");
    return;
}
if (!email.contains("@")) {
    JOptionPane.showMessageDialog(view, "
        Veuillez entrer une adresse email
        valide.");
    return;
}
double salaire = Double.parseDouble(
    salaireText);

Role role = Role.valueOf(view.roleCombo.
    getSelectedItem().toString().toUpperCase()
    );
Poste poste = Poste.valueOf(view.posteCombo.
    getSelectedItem().toString().toUpperCase()
    );

Employee updatedEmployee = new Employee(nom,
    prenom, email, phone, salaire, role, poste
    );
dao.update(updatedEmployee, id);

JOptionPane.showMessageDialog(view, "Employ
    mis jour avec succès.");
listEmployees(); // Rafraichir la liste
} else {
    JOptionPane.showMessageDialog(view, "Veuillez
        sélectionner un employé à modifier.");
}
} catch (NumberFormatException ex) {
    JOptionPane.showMessageDialog(view, "Salaire
        invalide.");
} catch (Exception ex) {
    JOptionPane.showMessageDialog(view, "Erreur: " +
        ex.getMessage());
}
}
}

```

## 3.2 Explication du code

La classe `EmployeeController` réalise plusieurs fonctions importantes :

- **Gestion des boutons d'actions** : Le contrôleur utilise des *ActionListener* pour écouter les interactions utilisateur via les boutons d'ajout, de suppression, de modification et de gestion des congés. Par exemple, un clic sur le bouton `addButton` appelle la méthode `addEmployee()`.
- **Connexion entre la vue et le modèle** : Le contrôleur récupère les données depuis la `EmployeeView` et les traite en utilisant le DAO (`GenericDAO<Employee>`).
- **Validation des champs** : Lors de l'ajout ou de la modification d'un employé, le contrôleur vérifie que tous les champs sont bien remplis et qu'ils respectent certaines contraintes (par exemple, une adresse e-mail valide).
- **Rafraîchissement de la table des employés** : Après chaque opération (ajout, suppression, modification), la méthode `listEmployees()` met à jour l'affichage de la liste des employés dans la vue.
- **Gestion des congés** : La méthode `switchViewButton` permet de basculer entre la gestion des employés et la gestion des congés.
- **Écoute des sélections dans la table** : L'utilisateur peut sélectionner une ligne dans la table pour pré-remplir les champs, ce qui facilite la modification d'un employé.

## 4 Contrôleur des congés

Le code suivant présente la classe `HolidayController`, qui sert de contrôleur dans une application de gestion des congés. Cette classe implémente les fonctionnalités principales telles que l'ajout, la modification et la suppression de congés. Elle interagit avec la vue (`HolidayView`) et le modèle (`HolidayDAOImpl`) pour maintenir l'interface utilisateur à jour. Voici le code :

### 4.1 Code du contrôleur des congés

Voici le code de la classe:

```
package Controller;

import DAO.HolidayDAOImpl;
import Model.Holiday;
import Model.Type;
import View.HolidayView;

import javax.swing.*;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.List;
```

```

public class HolidayController {
    private final HolidayView view;
    private final HolidayDAOImpl dao;

    public HolidayController(HolidayView view) {
        this.view = view;
        this.dao = new HolidayDAOImpl();

        loadEmployeeNames();
        refreshHolidayTable();

        view.addButton.addActionListener(e -> addHoliday());
        view.deleteButton.addActionListener(e ->
            deleteHoliday());
        view.modifyButton.addActionListener(e ->
            modifyHoliday());

        // Ajout d'un couteur de s lection sur la table
        view.holidayTable.getSelectionModel().
            addListSelectionListener(e -> {
                if (!e.getValueIsAdjusting()) { // V rifie si la
                    s lection est termin e
                        int selectedRow = view.holidayTable.
                            getSelectedRow();
                        if (selectedRow != -1) {
                            // R cup rer l'ID et les autres
                                informations de la ligne
                                    s lectionn e
                            int id = (int) view.holidayTable.
                                getValueAt(selectedRow, 0); //
                                R cup re l'ID depuis la colonne 0
                            String employeeName = (String) view.
                                holidayTable.getValueAt(selectedRow,
                                    1);
                            String startDate = (String) view.
                                holidayTable.getValueAt(selectedRow,
                                    2);
                            String endDate = (String) view.
                                holidayTable.getValueAt(selectedRow,
                                    3);
                            Type type = Type.valueOf(view.
                                holidayTable.getValueAt(selectedRow,
                                    4).toString());

                            // Remplir les champs de modification
                                avec ces valeurs
                            view.employeeNameComboBox.setSelectedItem
                                (employeeName);

```

```

        view.startDateField.setText(startDate);
        view.endDateField.setText(endDate);
        view.typeCombo.setSelectedItem(type.
            toString());

        // Modifier l'action du bouton de
        // modification pour utiliser l'ID de la
        // ligne slectionn e
        view.modifyButton.setActionCommand(String
            .valueOf(id));
    }
}

});
}

private void loadEmployeeNames() {
    view.employeeNameComboBox.removeAllItems();
    List<String> names = dao.getAllEmployeeNames();

    if (names.isEmpty()) {
        System.out.println("Aucun employ  trouv .");
    } else {
        System.out.println("Noms des employ s charg s :
            " + names);
        for (String name : names) {
            view.employeeNameComboBox.addItem(name);
        }
    }
}

private void refreshHolidayTable() {
    List<Holiday> holidays = dao.listAll();
    String[] columnNames = {"ID", "Employ ", "Date
        D but", "Date Fin", "Type"};
    Object[][] data = new Object[holidays.size()][5];

    for (int i = 0; i < holidays.size(); i++) {
        Holiday h = holidays.get(i);
        data[i] = new Object[]{h.getId(), h.
            getEmployeeName(), h.getStartDate(), h.
            getEndDate(), h.getType()};
    }

    view.holidayTable.setModel(new javax.swing.table.
        DefaultTableModel(data, columnNames));
}

private boolean isValidDate(String date) {

```

```

        try {
            DateTimeFormatter formatter = DateTimeFormatter.
                ISO_LOCAL_DATE;
            LocalDate.parse(date, formatter);
            return true;
        } catch (Exception e) {
            return false;
        }
    }

private boolean isEndDateAfterStartDate(String startDate,
String endDate) {
    DateTimeFormatter formatter = DateTimeFormatter.
        ISO_LOCAL_DATE;
    LocalDate start = LocalDate.parse(startDate,
        formatter);
    LocalDate end = LocalDate.parse(endDate, formatter);

    return end.isAfter(start);
}

private void addHoliday() {
    try {
        String employeeName = (String) view.
            employeeNameComboBox.getSelectedItem();
        String startDate = view.startDateField.getText();
        String endDate = view.endDateField.getText();
        Type type = Type.valueOf(view.typeCombo.
            getSelectedItem().toString().toUpperCase());

        if (!isValidDate(startDate) || !isValidDate(
            endDate)) {
            throw new IllegalArgumentException("Les dates
                doivent tre au format YYYY-MM-DD.");
        }

        if (!isEndDateAfterStartDate(startDate, endDate))
        {
            throw new IllegalArgumentException("La date
                de fin doit tre sup rieure la date
                de d but.");
        }

        DateTimeFormatter formatter = DateTimeFormatter.
            ISO_LOCAL_DATE;
        LocalDate start = LocalDate.parse(startDate,
            formatter);
        LocalDate end = LocalDate.parse(endDate,
            formatter);
    }
}

```

```

// Calcul de la dur e en jours
long daysBetween = java.time.temporal.ChronoUnit.
    DAYS.between(start, end) + 1; // +1 pour
    inclure le jour de d but

if (daysBetween > 25) {
    throw new IllegalArgumentException("La dur e
        du cong ne peut pas d passer 25 jours
        .");
}
if (dao.hasOverlappingHoliday(employeeName,
    startDate, endDate)) {
    throw new IllegalArgumentException("L'
        employ a d j un cong durant cette
        p riode.");
}

// V rifier le total des jours de cong pris
    cette ann e
int year = start.getYear();
long existingDays = dao.getTotalDaysTakenThisYear
    (employeeName, year);
if (existingDays + daysBetween > 25) {
    throw new IllegalArgumentException(
        "L'employ a d j pris " +
            existingDays + " jours de cong cette
            ann e. " +
            "Le total ne peut pas d passer 25 jours
            ."
    );
}

// Ajout du cong si toutes les validations sont
    pass es
Holiday holiday = new Holiday(employeeName,
    startDate, endDate, type);
dao.add(holiday);
loadEmployeeNames();
refreshHolidayTable();
JOptionPane.showMessageDialog(view, "Cong
    ajout avec succ s.");
} catch (Exception ex) {
    JOptionPane.showMessageDialog(view, "Erreur : " +
        ex.getMessage());
}
}

```



```

private void modifyHoliday() {
    try {
        // Récupérer l'ID du congé à partir de l'
        // action du bouton
        String actionCommand = view.modifyButton.
            getActionCommand();
        if (actionCommand != null && !actionCommand.trim()
            .isEmpty()) {
            int id = Integer.parseInt(actionCommand.trim()
                ());

            String employeeName = (String) view.
                employeeNameComboBox.getSelectedItemAt();
            String startDate = view.startDateField.
                getText();
            String endDate = view.endDateField.getText();
            Type type = Type.valueOf(view.typeCombo.
                getSelectedItem().toString().toUpperCase()
                ());

            if (!isValidDate(startDate) || !isValidDate(
                endDate)) {
                throw new IllegalArgumentException("Les
                    dates doivent être au format YYYY-MM-
                    DD.");
            }

            if (!isEndDateAfterStartDate(startDate,
                endDate)) {
                throw new IllegalArgumentException("La
                    date de fin doit être supérieure
                    à la date de début.");
            }

            if (dao.hasOverlappingHoliday(employeeName,
                startDate, endDate)) {
                throw new IllegalArgumentException("L'
                    employé a déjà un congé durant
                    cette période.");
            }

            Holiday holiday = new Holiday(employeeName,
                startDate, endDate, type);
            dao.update(holiday, id);
            loadEmployeeNames();
            refreshHolidayTable();
            JOptionPane.showMessageDialog(view, "Congé
                modifié avec succès.");
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(view, "Erreur lors de la
            modification du congé : " + e.getMessage());
    }
}

```

```

        } else {
            JOptionPane.showMessageDialog(view, "Veuillez
                slectionner un cong        modifier.");
        }
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(view, "Erreur : " +
            ex.getMessage());
    }
}

private void deleteHoliday() {
    try {
        // V rifier si une ligne est slectionn e dans
            la table
        int selectedRow = view.holidayTable.
            getSelectedRow();
        if (selectedRow == -1) {
            JOptionPane.showMessageDialog(view, "Veuillez
                slectionner un cong        supprimer.");
            return;
        }

        // R cup rer l'ID du cong        partir du
            mod le de table
        int id = (int) view.holidayTable.getValueAt(
            selectedRow, 0); // Supposons que la colonne 0
            contient l'ID

        // Demander confirmation avant de supprimer
        int confirm = JOptionPane.showConfirmDialog(view,
            " tes -vous s r de vouloir supprimer le
                cong avec l'ID " + id + " ?",
            "Confirmation de suppression",
            JOptionPane.YES_NO_OPTION);

        if (confirm == JOptionPane.YES_OPTION) {
            // Supprimer le cong via le DAO
            dao.delete(id);

            // Rafra chir les donn es dans la vue
            loadEmployeeNames();
            refreshHolidayTable();

            JOptionPane.showMessageDialog(view, " Cong
                supprim avec succ s.");
        } else {
            JOptionPane.showMessageDialog(view, "
                Suppression annul e.");
        }
    }
}

```

```

        }
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(view, "Erreur : " +
            ex.getMessage());
    }
}
}

```

**Explication :** La classe `HolidayController` agit comme un pont entre la couche d'interface utilisateur (vue) et la couche de gestion des données (DAO). Elle inclut des méthodes pour charger les données des employés, gérer les congés, valider les entrées, et mettre à jour la vue.

- **Constructeur :** Le constructeur initialise les composants `HolidayView` et `HolidayDAOImpl`. Il configure également les écouteurs pour les boutons (ajout, suppression, modification) et gère la sélection dans le tableau des congés.
- **Méthodes principales :**
  - `loadEmployeeNames()` : Charge les noms des employés depuis la base de données et met à jour la liste déroulante dans la vue.
  - `refreshHolidayTable()` : Rafraîchit les données du tableau pour refléter les congés actuels dans la base de données.
  - `isValidDate()` : Vérifie si une date est valide au format ISO (YYYY-MM-DD).
  - `isEndDateAfterStartDate()` : S'assure que la date de fin est postérieure à la date de début.
- **Ajout de congés (`addHoliday()`) :**
  - Valide les dates et le type de congé avant de l'ajouter.
  - Vérifie si la durée totale de congés ne dépasse pas 25 jours par an.
  - Vérifie les chevauchements de congés existants pour un employé.
- **Modification de congés (`modifyHoliday()`) :**
  - Met à jour les informations d'un congé sélectionné, après validation.
  - Identifie le congé via l'ID obtenu depuis l'action associée au bouton de modification.
- **Suppression de congés (`deleteHoliday()`) :**
  - Supprime un congé sélectionné après confirmation par l'utilisateur.
  - Rafraîchit les données affichées dans la vue.

## 5 Contrôleur des vues

Le code suivant présente la classe `ViewController`, qui permet de gérer la navigation entre deux vues d'une application Java Swing : `EmployeeView` et `HolidayView`. Cette classe agit comme un contrôleur, reliant les vues pour une navigation fluide et intuitive. L'objectif principal est de fournir un mécanisme simple pour basculer entre les deux interfaces utilisateur.

### 5.1 Code du contrôleur des vues

Voici le code de la classe:

```
package Controller;

import View.EmployeeView;
import View.HolidayView;

public class ViewController {
    private final EmployeeView employeeView;
    private final HolidayView holidayView;

    public ViewController() {
        this.employeeView = new EmployeeView();
        this.holidayView = new HolidayView();

        // Connect views for navigation
        employeeView.switchViewButton.addActionListener(e ->
            showHolidayView());
        holidayView.switchViewButton.addActionListener(e ->
            showEmployeeView());
    }

    public void showEmployeeView() {
        holidayView.setVisible(false);
        employeeView.setVisible(true);
    }

    public void showHolidayView() {
        employeeView.setVisible(false);
        holidayView.setVisible(true);
    }

    public static void main(String[] args) {
        new ViewController().showEmployeeView();
    }
}
```

## Explication

La classe `ViewController` assure les fonctionnalités suivantes :

- Elle initialise deux vues principales : `EmployeeView` et `HolidayView`.
- Elle configure des écouteurs d'événements (`ActionListener`) pour les boutons de navigation dans chaque vue.
- Les méthodes `showEmployeeView` et `showHolidayView` permettent de basculer entre les deux vues en ajustant leur visibilité.
- La méthode `main` lance l'application en affichant la vue `EmployeeView` par défaut.

Ce design suit le modèle MVC (Modèle-Vue-Contrôleur), où le contrôleur gère l'interaction entre les vues. Cela permet de maintenir une structure claire et d'améliorer la maintenabilité du code.

## Présentation des vues dans l'application

Cette application Java Swing est composée de deux principales interfaces : `EmployeeView` pour la gestion des employés et `HolidayView` pour la gestion des congés. Chaque interface offre des fonctionnalités spécifiques avec des boutons d'action et des champs de saisie, tout en permettant une navigation fluide entre elles.

### EmployeeView - Gestion des employés

La classe `EmployeeView` représente l'interface utilisateur pour gérer les employés. Elle comprend des champs pour saisir les informations des employés, une table pour afficher les données, et des boutons pour effectuer des actions (ajout, affichage, suppression, modification). Un bouton de navigation permet de passer à la vue `HolidayView`.

### Code source

```
package View;

import javax.swing.*.*;
import java.awt.*.*;

public class EmployeeView extends JFrame {
    public JTable employeeTable;
    public JButton addButton, listButton, deleteButton,
        modifyButton, switchViewButton;
    public JTextField nameField, surnameField, emailField,
        phoneField, salaryField;
    public JComboBox<String> roleCombo, posteCombo;

    public EmployeeView() {
        setTitle("Gestion des Employés");
        setSize(800, 600);
    }
}
```

```

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLayout(new BorderLayout());

JPanel inputPanel = new JPanel(new GridLayout(0, 2,
    10, 10)); // 0 lignes et 2 colonnes

inputPanel.add(new JLabel("Nom:"));
nameField = new JTextField();
inputPanel.add(nameField);

inputPanel.add(new JLabel("Pr nom:"));
surnameField = new JTextField();
inputPanel.add(surnameField);

inputPanel.add(new JLabel("Email:"));
emailField = new JTextField();
inputPanel.add(emailField);

inputPanel.add(new JLabel("T l phone:"));
phoneField = new JTextField();
inputPanel.add(phoneField);

inputPanel.add(new JLabel("Salaire:"));
salaryField = new JTextField();
inputPanel.add(salaryField);

inputPanel.add(new JLabel("R le:"));
roleCombo = new JComboBox<>(new String[]{"Admin", "
    Employe"});
inputPanel.add(roleCombo);

inputPanel.add(new JLabel("Poste:"));
posteCombo = new JComboBox<>(new String[]{"
    INGENIEURE_ETUDE_ET_DEVELOPPEMENT", "TEAM_LEADER",
    "PILOTE"});
inputPanel.add(posteCombo);

add(inputPanel, BorderLayout.NORTH);

employeeTable = new JTable();
add(new JScrollPane(employeeTable), BorderLayout.
    CENTER);

JPanel buttonPanel = new JPanel();
addButton = new JButton("Ajouter");
buttonPanel.add(addButton);
listButton = new JButton("Afficher");
buttonPanel.add(listButton);

```

```

        deleteButton = new JButton("Supprimer");
        buttonPanel.add(deleteButton);
        modifyButton = new JButton("Modifier");
        buttonPanel.add(modifyButton);

        // Adding the navigation button to switch to the
        Holiday view
        switchViewButton = new JButton("Gérer les Congés");
        buttonPanel.add(switchViewButton);

        add(buttonPanel, BorderLayout.SOUTH);
    }
}

```

## HolidayView - Gestion des congés

La classe `HolidayView` fournit une interface pour gérer les congés des employés. Elle inclut des champs de saisie pour les informations de congé (nom de l'employé, dates, type), une table pour afficher les congés, et des boutons pour effectuer des actions. Un bouton de navigation permet de revenir à la vue `EmployeeView`.

### Code source

```

package View;

import javax.swing.*.*;
import java.awt.*.*;

public class HolidayView extends JFrame {
    public JTable holidayTable;
    public JButton addButton, listButton, deleteButton,
        modifyButton, switchViewButton;
    public JComboBox<String> employeeNameComboBox;
    public JTextField startDateField, endDateField;
    public JComboBox<String> typeCombo;

    public HolidayView() {
        setTitle("Gestion des Congés");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Panneau de saisie des champs
        JPanel inputPanel = new JPanel(new GridLayout(0, 2,
            10, 10));
        inputPanel.add(new JLabel("Employé Nom Complet:"));
        employeeNameComboBox = new JComboBox<>();
        inputPanel.add(employeeNameComboBox);
    }
}

```

```

inputPanel.add(new JLabel("Date D but:"));
startDateField = new JTextField();
inputPanel.add(startDateField);

inputPanel.add(new JLabel("Date Fin:"));
endDateField = new JTextField();
inputPanel.add(endDateField);

inputPanel.add(new JLabel("Type:"));
typeCombo = new JComboBox<>(new String[]{"CONGE_PAYE", "CONGE_MALADIE", "CONGE_NON_PAYE"});
inputPanel.add(typeCombo);

add(inputPanel, BorderLayout.NORTH);

// Table des congés
holidayTable = new JTable();
add(new JScrollPane(holidayTable), BorderLayout.CENTER);

// Boutons d'action
JPanel buttonPanel = new JPanel();
addButton = new JButton("Ajouter");
buttonPanel.add(addButton);
listButton = new JButton("Afficher");
buttonPanel.add(listButton);
deleteButton = new JButton("Supprimer");
buttonPanel.add(deleteButton);
modifyButton = new JButton("Modifier");
buttonPanel.add(modifyButton);

switchViewButton = new JButton("Gerer les Employés");
buttonPanel.add(switchViewButton);

add(buttonPanel, BorderLayout.SOUTH);
}
}

```

## Code Principal : Classe Main

La classe **Main** sert de point d'entrée à l'application. Elle initialise les vues et les contrôleurs, puis gère la navigation entre les interfaces.

### Code source



```
package Main;

import Controller.EmployeeController;
import Controller.HolidayController;
import View.EmployeeView;
import View.HolidayView;

public class Main {
    public static void main(String[] args) {

        EmployeeView employeeView = new EmployeeView();
        HolidayView holidayView = new HolidayView();

        new EmployeeController(employeeView, holidayView);
        new HolidayController(holidayView);

        employeeView.setVisible(true);

        employeeView.switchViewButton.addActionListener(e -> {
            {
                employeeView.setVisible(false);
                holidayView.setVisible(true);
            }
        });

        holidayView.switchViewButton.addActionListener(e -> {
            holidayView.setVisible(false);
            employeeView.setVisible(true);
        });
    }
}
```

# Résultat de l'application

Dans cette section, nous allons illustrer les résultats obtenus après avoir exécuté le projet.

## Captures d'écran des Résultats

Gestion des Employés

Nom:

Prénom:

Email:

Téléphone:

Salaire:

Rôle:

Admin

Poste:

INGENIEURE\_ETUDE\_ET\_DEVELOPPEMENT

ID	Nom	Prénom	Email	Téléphone	Salaire	Rôle	Poste
2	AMRANI	Sara	nwlam@email.c...	0623456789	7000.0	EMPLOYE	INGENIEURE
6	Imrani	Ahlam	ahlam@gmail.co...	0685421789	5000.0	EMPLOYE	INGENIEURE
7	SQUALLI	Ali	ali@gmail.com	0612457456	9000.0	EMPLOYE	TEAM_LEAD
9	AMMOUN	Ilham	ilham@gmail.com	0674512896	9000.0	ADMIN	TEAM_LEAD

Ajouter

Afficher

Supprimer

Modifier

Gérer les Congés

Figure 3: Le Résultat Final

Gestion des Employés

Nom:

EL HATIMI

Prénom:

Amal

Email:

amalelha@gmail.com

Téléphone:

0674859123

Salaire:

5000

Rôle:

Employe

Poste:

ET\_DEVELOPPEMENT

ID	Nom	Prénom	Salaire	Rôle	Poste
2	AMRANI	Sara		EMPLOYE	INGENIEUR
6	Imrani	Ahlam		EMPLOYE	INGENIEUR
7	SQUALLI	Ali		EMPLOYE	TEAM_LEAD
9	AMMOUN	Ilham		ADMIN	TEAM_LEAD

Ajouter

Afficher


Supprimer

Modifier

Gérer les Congés

Message

×



Employé ajouté avec succès.

OK

Figure 4: L'ajout d'un employé

Gestion des Employés

Nom:

EL HATIMI

Prénom:

Amal

Email:

amalelha@gmail.com

Téléphone:

0674859123

Salaire:

5000

Rôle:

Employe

Poste:

INGENIEURE\_ETUDE\_ET\_DEVELOPPEMENT

ID	Nom	Prénom	Email	Téléphone	Salaire	Rôle	Poste
2	AMRANI	Sara	nwlam@email.c...	0623456789	7000.0	EMPLOYE	INGENIEUR
3	Imrani	Ahlam	ahlam@gmail.co...	0685421789	5000.0	EMPLOYE	INGENIEUR
7	SQUALLI	Ali	ali@gmail.com	0612457456	9000.0	EMPLOYE	TEAM_LEAD
9	AMMOUN	Ilham	ilham@gmail.com	0674512896	9000.0	ADMIN	TEAM_LEAD
10	EL HATIMI	Amal	amalelha@gmail...	0674859123	5000.0	EMPLOYE	INGENIEUR

Ajouter

Afficher

Supprimer

Modifier

Gérer les Congés

Figure 5: Affichage de la liste des employés

Gestion des Employés

Nom: EL HATIMI

Prénom: Amal

Email: amalelha@gmail.com

Téléphone: 0674859123

Salaire: 6000.0

Rôle: Employe

Poste: INGENIEURE\_ETUDE\_ET\_DEVELOPPEMENT

ID	Nom	Prénom	Salaire	Rôle	Poste
2	AMRANI	Sara		EMPLOYE	INGENIEUR
6	Imrani	Ahlam		EMPLOYE	INGENIEUR
7	SQUALLI	Ali		EMPLOYE	TEAM_LEAD
9	AMMOUN	Ilham		ADMIN	TEAM_LEAD
10	EL HATIMI	Amal		EMPLOYE	INGENIEUR

Message

Employé mis à jour avec succès.

OK

Ajouter Afficher Supprimer Modifier Gérer les Congés

Figure 6: Modification de salaire

ID	Nom	Prénom	Email	Téléphone	Salaire	Rôle	Poste
2	AMRANI	Sara	nwlam@email.c...	0623456789	7000.0	EMPLOYE	INGENIEURE
6	Imrani	Ahlam	ahlam@gmail.co...	0685421789	5000.0	EMPLOYE	INGENIEURE
7	SQUALLI	Ali	ali@gmail.com	0612457456	9000.0	EMPLOYE	TEAM_LEAD
9	AMMOUN	Ilham	ilham@gmail.com	0674512896	9000.0	ADMIN	TEAM_LEAD
10	EL HATIMI	Amal	amalelha@gmail...	0674859123	6000.0	EMPLOYE	INGENIEURE

Figure 7: La modification des données réussie!

Gestion des Employés

Nom:

EL HATIMI

Prénom:

Amal

Email:

amalelha@gmail.com

Téléphone:

0674859123

Salaire:

6000.0

Rôle:

Employe

Poste:

INGENIEURE\_ETUDE\_ET\_DEVELOPPEMENT

ID	Nom	Prénom	Email	Téléphone	Salaire	Rôle	Poste
2	AMRANI	Sara	nwlam@email.c...	0623456789	7000.0	EMPLOYE	INGENIEUR
6	Imrani	Ahlam	ahlam@gmail.co...	0685421789	5000.0	EMPLOYE	INGENIEUR
7	SQUALLI	Ali	ali@gmail.com	0612457456	9000.0	EMPLOYE	TEAM_LEA
9	AMMOUN	Ilham	ilham@gmail.com	0674512896	9000.0	ADMIN	TEAM_LEA
10	EL HATIMI	Amal	amalelha@gmail...	0674859123	6000.0	EMPLOYE	INGENIEUR

Confirmation de suppression

?

Êtes-vous sûr de vouloir supprimer l'employé avec l'ID 10 ?

Yes

No

Ajouter

Afficher

Supprimer

Modifier

Gérer les Congés

Figure 8: Suppression de l'employé

Gestion des Employés

Nom:

EL HATIMI

Prénom:

Amal

Email:

amalelha@gmail.com

Téléphone:

0674859123

Salaire:

6000.0

Rôle:

Employee

Poste:

ET\_DEVELOPPEMENT

ID	Nom	Prénom	Salaire	Rôle	Poste
2	AMRANI	Sara		EMPLOYE	INGENIEURE
6	Imrani	Ahlam		EMPLOYE	INGENIEURE
7	SQUALLI	Ali		EMPLOYE	TEAM_LEAD
9	AMMOUN	Ilham		ADMIN	TEAM_LEAD
10	EL HATIMI	Amal	amalelha@gmail.com	0674859123	6000.0

Ajouter

Afficher

Supprimer

Modifier

Gérer les Congés

Message


×

i

Employé supprimé avec succès.

OK

Figure 9: Suppression réussie


**Gestion des Congés**

**Employé Nom Complet:**

AMRANI Sara

**Date Début:**

2025-01-14

**Date Fin:**

2025-01-19

**Type:**

CONGE\_PAYE

ID	Employé	Date Début	Date Fin	Type
12	AMRANI Sara	2024-12-25	2024-12-30	CONGE_PAYE
15	Imrani Ahlam	2025-01-02	2025-01-10	CONGE_MALADIE
16	SQUALLI Ali	2025-01-14	2025-01-19	CONGE_PAYE
14	AMMOUN Ilham	2025-01-01	2025-01-10	CONGE_PAYE

Ajouter

Afficher

Supprimer

Modifier

Gerer les Employés

Figure 10: Gestion de congés



Gestion des Congés

Employé Nom Complet:

AMRANI Sara

Date Début:

2025-01-01

Date Fin:


2025-01-10

Type:

CONGE\_PAYE

ID	Employé	Date Début	Date Fin	Type
2	AMRANI Sara	2024-12-25	2024-12-30	CONGE_PAYE
5	Imrani Ahlam	2025-01-02	2025-01-10	CONGE_MALADIE
3	SQUALLI Ali	2024-12-26	2024-12-29	CONGE_PAYE
4	AMMOUN Ilham	2025-01-01	2025-01-10	CONGE_PAYE

Message



Congé modifié avec succès.

OK

Ajouter

Afficher

Supprimer

Modifier

Gerer les Employés

Figure 11: Modification de la date de congé

Gestion des Congés

Employé Nom Complet:

SQUALLI Ali

Date Début:

2024-12-26

Date Fin:

2024-12-29

Type:

CONGE\_PAYE

ID	Employé	Date Début	Date Fin	Type
12	AMRANI Sara	2024-12-25	2024-12-30	CONGE_PAYE
15	Imrani Ahlam	2025-01-02	2025-01-10	CONGE_MALADIE
13	SQUALLI Ali	2024-12-26	2024-12-29	CONGE_PAYE
14	AMMOUN Ilham	2025-01-01	2025-01-10	CONGE_PAYE

Confirmation de suppression

?

Êtes-vous sûr de vouloir supprimer le congé avec l'ID 13 ?

Yes

No

Ajouter

Afficher

Supprimer

Modifier

Gerer les Employés

Figure 12: Suppression du congé

Gestion des Congés

Employé Nom Complet:

AMRANI Sara

Date Début:

2024-12-26

Date Fin:

2024-12-29

Type:

CONGE\_PAYE

ID	Employé	Date Début	Date Fin	Type
12	AMRANI Sara	2024-12-25	2024-12-30	CONGE_PAYE
15	Imrani Ahlam	2025-01-02	2025-01-10	CONGE_MALADIE
14	AMMOUN Ilham	2025-01-01	2025-01-10	CONGE_PAYE

Message

×

i

Congé supprimé avec succès.

OK

Ajouter

Afficher

Supprimer

Modifier

Gerer les Employés

Figure 13: Suppression réussie

Gestion des Congés

Employé Nom Complet:

AMMOUN Ilham

Date Début:

2025-03-28

Date Fin:


2025-04-15

Type:

CONGE\_PAYE

ID	Employé	Date Début	Date Fin	Type
12	AMRANI Sara	2024-12-25	2024-12-30	CONGE_PAYE
15	Imrani Ahlam	2025-01-02	2025-01-10	CONGE_MALADIE
14	AMMOUN Ilham	2025-01-01	2025-01-10	CONGE_PAYE

Message



Erreur : L'employé a déjà pris 10 jours de congé cette année. Le total ne peut pas dépasser 25 jours.

OK

Ajouter

Afficher

Supprimer

Modifier

Gérer les Employés

Figure 14: Erreur lors du dépassement de la durée de congé qui est 25 jours

51

Gestion des Congés

Employé Nom Complet:

AMMOUN Ilham

Date Début:

2025-01-03

Date Fin:

2025-01-12

Type:

CONGE\_PAYE

ID	Employé	Date Début	Date Fin	Type
12	AMRANI Sara	2024-12-25	2024-12-30	CONGE_PAYE
15	Imrani Ahlam	2025-01-02	2025-01-10	CONGE_MALADIE
16	SQUALLI Ali	2025-01-14	2025-01-19	CONGE_PAYE
14	AMMOUN Ilham	2025-01-01	2025-01-10	CONGE_PAYE

Message

Erreur : L'employé a déjà un congé durant cette période.

OK

Ajouter

Afficher

Supprimer

Modifier

Gerer les Employés

Figure 15: L'employé a déjà un congé dans cette durée là donc il ne doit pas prendre un autre congé dans la meme période

Gestion des Congés

Employé Nom Complet:

AMRANI Sara

Date Début:

2025-01-14

Date Fin:


2025-01-19

Type:

CONGE\_PAYE

ID	Employé	Date Début	Date Fin	Type
12	AMRANI Sara	2024-12-25	2024-12-30	CONGE_PAYE
15	Imrani Ahlam	2025-01-02	2025-01-10	CONGE_MALADIE
16	SQUALLI Ali	2025-01-14	2025-01-19	CONGE_PAYE
14	AMMOUN Ilham	2025-01-01	2025-01-10	CONGE_PAYE

Message



Congé ajouté avec succès.

OK

Ajouter

Afficher

Supprimer

Modifier

Gerer les Employés

Figure 16: Ajout réussi

Gestion des Employés

Nom:

ESSALMI

Prénom:

Imrane

Email:

img

Téléphone:

0674859123

Salaire:

6000.0

Rôle:

Employe

Poste:

INGENIEURE\_ETUDE\_ET\_DEVELOPPEMENT

ID	Nom	Prénom	Email	Téléphone	Salaire	Rôle	Poste
2	AMRANI	Sara	nwlam@email.c...	0623456789	7000.0	EMPLOYE	INGENIEURE
6	Imrani	Ahlam	ahlam@gmail.co...	0685421789	5000.0	EMPLOYE	INGENIEURE
7	SQUALLI	Ali	ali@gmail.com	0612457456	9000.0	EMPLOYE	TEAM_LEAD
9	AMMOUN	Ilham	ilham@gmail.com	0674512896	9000.0	ADMIN	TEAM_LEAD

Ajouter

Afficher

Supprimer

Modifier

Gérer les Congés

Message

Veuillez entrer une adresse email val

OK

Figure 17: Il faut entrer une adresse mail valide!

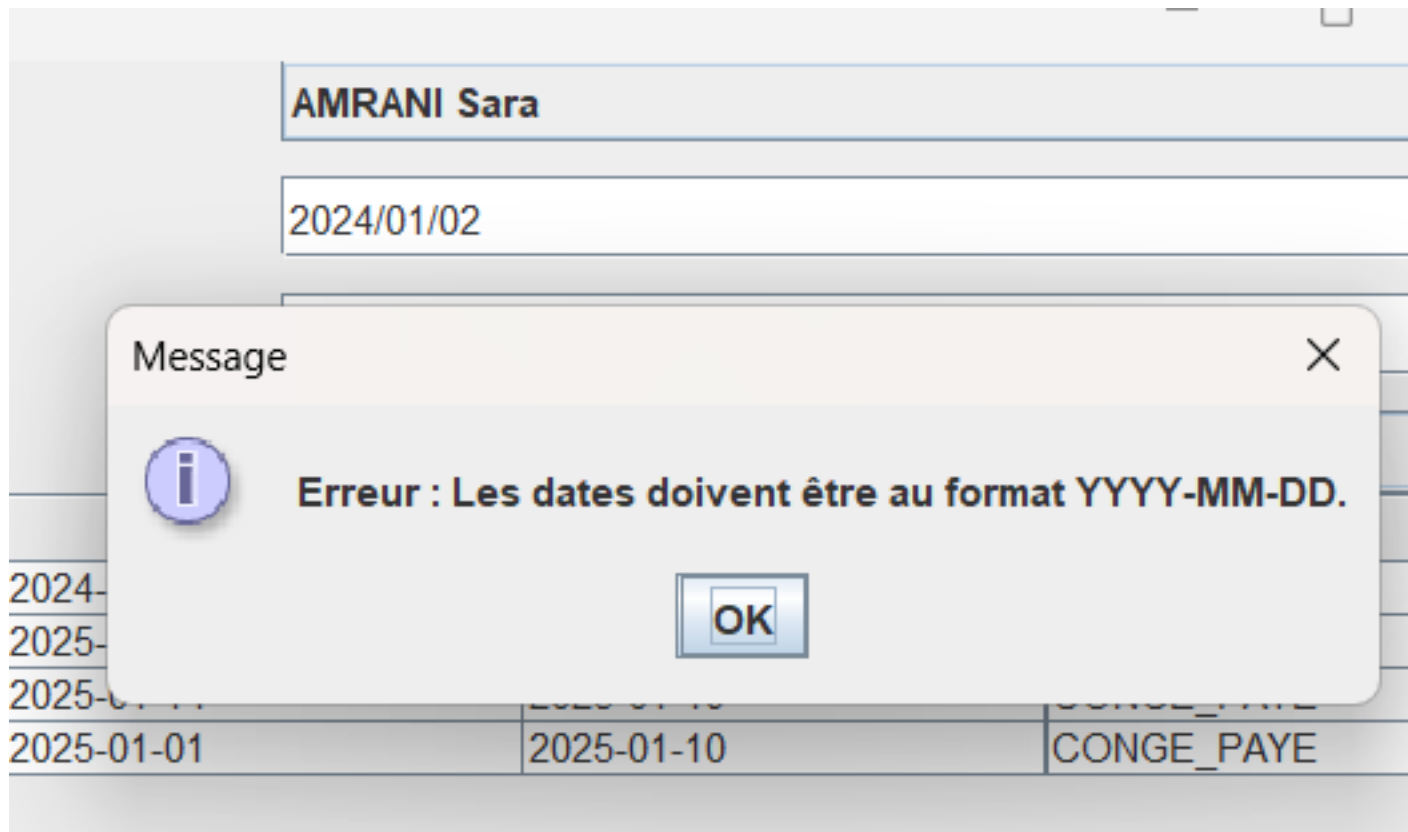


Figure 18: La date doit etre au format YYYY-MM-DD

## Conclusion

Ce travail pratique m'a permis de me familiariser avec les principes de l'architecture MVC, en séparant clairement les différentes responsabilités dans l'application. J'ai également appris à implémenter le DAO (Data Access Object) pour gérer la persistance des données, ce qui m'a permis de séparer la logique d'accès à la base de données de l'application elle-même. De plus, j'ai exploré l'utilisation de la généricité en Java pour créer des classes réutilisables et flexibles, et j'ai utilisé Swing pour concevoir l'interface graphique, garantissant une interaction utilisateur fluide et efficace.

Ce projet m'a ainsi permis d'acquérir une meilleure compréhension des concepts clés du développement logiciel moderne, tout en appliquant des pratiques recommandées pour la gestion des données et la conception d'applications modulaires et évolutives.