# Structure du Projet de Gestion des Employés

Fatim ezzahrae AKNIZ

2 janvier 2025

# Introduction

Le dossier présente les 4 parties du projets : Employees, Congées, Importer/exporter, se connecter/ajouter compte. Il est conçue avec le modèle MVC et DAO, d'où il se compose de 5 packages : Model, View, Controlleur, DAO, package par défaut(Main).

**Note :** La classe DBConnection est une classe centralisé pour les quatres parties et se situe dans le package DAO

```
public static Connection getConnection() {
    if (conn == null) { // Check if connection is null
        try {
            // Load the JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            // Establish the connection
            conn = DriverManager.getConnection(url, user, password);
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
            throw new RuntimeException("Error lors de la connexion");
        }
    }
    return conn;
}
```

Listing 1 – DAO implementation

# 1. Interface generique

Cette interface définit un ensemble de méthodes communes à l'entité Employe et Holiday, permettant ainsi de centraliser et de standardiser leur implémentation.

```
public interface GenericDAOI<T> {

    Object[][] findById(int employeeID) throws SQLException, Exception;

    void add(T employe) throws SQLException, Exception;

    void update(T employe, int emp_id) throws SQLException, Exception;

    void delete(T employe) throws SQLException, Exception;

    Object[][] findAll() throws SQLException, Exception;


}
```

Listing 2 – le corps de l'interface

# 2. Employe

## 2.1 Employe.java

Cette classe constiyue les contructeurs, les getters et les setters de l'entité Employe :

```
public Employe(String nom, String prenom, String email, String tel, int
    salaire, Role role, Poste poste) {
    this.email = email;
    this.nom = nom;
    this.prenom = prenom;
    this.salaire = salaire;
    this.tel = tel; // Updated
    this.role = role;
    this.poste = poste;
    this.holiday = 25;
}
```

Listing 3 – Constructeur de l'employe

## 2.2 EmployeDAOImpl.java

Cette classe présente la gestion des données dans la base de données : Ajout, edit, et suppression

```
@Override
public void add(Employe employe) throws Exception{

    String sql = "Insert␣Into␣Employe␣(nom,␣prenom,␣email,␣tel,␣salaire
       ,␣Poste,␣Role␣)␣values␣(?,?,?,?,?,?,?)";

    try(PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, employe.getNom());
        stmt.setString(2, employe.getPrenom());
        stmt.setString(3, employe.getEmail());
        stmt.setString(4, employe.getTel());
        stmt.setInt(5, employe.getSalaire());
        stmt.setString(6, employe.getPoste().name());
        stmt.setString(7, employe.getRole().name());
        stmt.executeUpdate();

    } catch (SQLException e) {
        throw new Exception("Error␣in␣add␣Emp:␣" + e.getMessage(), e);

    }
}
```

Listing 4 – La methodes pour ajouter un employe

## 2.3 EmployeModel.java

Cette classe encapsule les methodes de la classe EmployeDAO pour une gestion des données sécurisées, en ajoutant ainsi les regles de metiers

```
// Validate employee data
private void validateEmployee(Employe employe) throws Exception {
    if (employe == null) {
        throw new IllegalArgumentException("Employee cannot be null.");
    }
    if (employe.getNom() == null || employe.getNom().trim().isEmpty())
       {
        throw new IllegalArgumentException("Employee name cannot be
           null or empty.");
    }
    if (employe.getPrenom() == null || employe.getPrenom().trim().
       isEmpty()) {
        throw new IllegalArgumentException("Employee first name cannot
           be null or empty.");
    }
    if (employe.getEmail() == null || !employe.getEmail().matches("^[A-
       Za-z0-9+_.-]+@[A-Za-z0-9.-]+$")) {
        throw new IllegalArgumentException("Invalid email address.");
    }
    if (employe.getTel() == null ||
        !employe.getTel().matches("^0[67][0-9]{8}$")) {
        throw new IllegalArgumentException("Phone number must be 10
           digits and start with 06 or 07.");
    }
    if (employe.getSalaire() <= 0) {
        throw new IllegalArgumentException("Salary must be a positive
           number.");
    }
}


// Get an employee by ID
public Employe getEmployeeById(int emp_id) throws Exception {
    if (emp_id <= 0) {
        throw new IllegalArgumentException("Invalid employee ID.");
    }

    try {
        Employe employe = employeDAO.getEmployeById(emp_id);
        if (employe == null) {
            throw new Exception("Employee not found with ID: " + emp_id
               );
        }
        return employe;
    } catch (SQLException e) {
        throw new Exception("Error while retrieving employee: " + e.
           getMessage(), e);
    }
}
```

Listing 5 – Exemple de quelques methodes

## 2.4 EmployeView.java

Cette classe constitue la vue que l'utulisateur travaille avec. En plus, il se compose des methodes pour communiquer les données avec le controlleur, et les faire mise à jour

```
public EmployeeView() {
    rolesCombo = new JComboBox<>(Role.values());
    postesCombo = new JComboBox<>(Poste.values());

    tableModel = new DefaultTableModel(new Object[][] {}, tableColumns)
        ;
    tbl = new JTable(tableModel);
    tbl.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
    tbl.setFillsViewportHeight(true);



    JPanel data = new JPanel();
    data.setLayout(new BorderLayout());

    JPanel entries = new JPanel();
    entries.setLayout(new GridLayout(7,2));
    entries.add(new JLabel("Nom:"));
    entries.add(nom);
    entries.add(new JLabel("Prenom:"));
    entries.add(prenom);
    entries.add(new JLabel("Email:"));
    entries.add(email);
    entries.add(new JLabel("Telephone:"));
    entries.add(tel);
    entries.add(new JLabel("Salaire:"));
    entries.add(salaire);
    entries.add(new JLabel("Role:"));
    entries.add(rolesCombo);
    entries.add(new JLabel("Poste:"));
    entries.add(postesCombo);

    data.add(entries, BorderLayout.CENTER);

    JScrollPane scroll = new JScrollPane(tbl);
    tbl.setPreferredScrollableViewportSize(new Dimension(800, 200));
        // Adjust as needed
    applyTableColumnWidths();
    JPanel table = new JPanel();
    table.add(scroll);
    data.add(table, BorderLayout.SOUTH);

    add(data, BorderLayout.CENTER);

    JPanel buttons = new JPanel();
    buttons.setLayout(new FlowLayout());
    buttons.add(addButton);
    buttons.add(updateButton);
    buttons.add(deleteButton);
    buttons.add(listButton);
    buttons.add(findIdButton);
    buttons.add(importButton);
    buttons.add(exportButton);
    buttons.add(createUserAcc);

    add(buttons, BorderLayout.SOUTH);
}
```

Listing 6 – le contructeur de la classe

## 2.5 EmployeController.java

Cette classe sert à collecter les données de la vue, et les manipuler en utilisant les methodes de EmployeModel, tout ça sous en réponse d'un evenement créé par l'utulisateur

```
view.getDeleteButton().addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (selectedEmployeeId == -1) {
            JOptionPane.showMessageDialog(view, "Please␣select␣an␣
               employee␣to␣delete." , "Error", JOptionPane.
               ERROR_MESSAGE);
            return;
        }

        int confirm = JOptionPane.showConfirmDialog(view,
            "Are␣you␣sure␣you␣want␣to␣delete␣this␣employee?",
            "Confirm␣Delete",
            JOptionPane.YES_NO_OPTION);

        if (confirm == JOptionPane.YES_OPTION) {
            try {
                Employe employe = employeModel.getEmployeeById(
                   selectedEmployeeId);
                employeModel.deleteEmployee(employe);
                selectedEmployeeId = -1;
                Object[][] data = employeModel.getAllEmployees();
                view.refreshTable(data);
                JOptionPane.showMessageDialog(view, "Employee␣deleted␣
                   successfully!", "Success", JOptionPane.
                   INFORMATION_MESSAGE);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(view, "Failed␣to␣delete␣
                   employee:␣" + ex.getMessage(), "Error", JOptionPane.
                   ERROR_MESSAGE);
            }
        }
    }
});
```

Listing 7 – action de la boutton supprimer

**Note :** La description des classes est la même pour l'entité Holiday.

# 3. Holiday

## 3.1 Holiday.java

```
public Holiday(int id, String DDebut, String DFin, HolidayType type,
    String employe) {
     this.id = id;
     this.DDebut = DDebut;
     this.DFin = DFin;
     this.type = type;
     this.employe = employe;
}

public Holiday(String DDebut, String DFin, int idType, int idEmp) {
     this.DDebut = DDebut;
     this.DFin = DFin;
     this.idType = idType;
     this.idEmp = idEmp;
}
```

Listing 8 – Constructeurs de Holiday

## 3.2 HolidayDAOImpl.java

```
public void update(Holiday employe, int emp_id) throws SQLException {
    String sql = "SELECT id from typeholiday where type = ?";

    try(PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, employe.getType().toString());
        try(ResultSet rs = stmt.executeQuery()) {
            if(rs.next()) {
                idType = rs.getInt("id");
            } else {
                throw new SQLException("Type not found: " + employe.
                    getType().toString());
            }
        }

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

    String sql1 = "UPDATE holiday SET idEmp = ?, idType = ?, DDebut = ?
        , DFin = ? WHERE id = ?";
    try(PreparedStatement stmt = conn.prepareStatement(sql1)) {

        stmt.setInt(1, getIdEmp(employe.getEmploye()));
        stmt.setInt(2, idType);
        stmt.setString(3, employe.getDDebut());
        stmt.setString(4, employe.getDFin());
        stmt.setInt(5, emp_id);
        stmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
}
```

Listing 9 – methode pour editer une holiday

## 3.3 HolidayModel.java

```
public int calculateHoliday(String DDate, String FDate, int idEmp)
    throws Exception {
     int EmpHoliday = dao.getEmployeeHoliday(idEmp);
     if (EmpHoliday == -1) {
          throw new Exception("Failed␣to␣retrieve␣remaining␣holiday␣
              balance.");
     }

     int updatedHoliday = HolidayPeriod(DDate, FDate, EmpHoliday);
     if (updatedHoliday < 0) {
          throw new Exception("Holiday␣period␣exceeds␣the␣permitted␣limit
              .");
     }
     return updatedHoliday;
}

public int getEmployeeHoliday(int idEmp) throws Exception {
     int EmpHoliday = dao.getEmployeeHoliday(idEmp);
     if (EmpHoliday == -1) {
          throw new Exception("Failed␣to␣retrieve␣remaining␣holiday␣
              balance.");
     }
     return EmpHoliday;
}
```

Listing 10 – Exemple de methodes avec regles de métier

## 3.4 HolidayView.java

```
public HolidayView() {
     ArrayList<String> employes = dao.getEmployees();
     employees = new JComboBox<>(employes.toArray(new String[0]));

     ArrayList<HolidayType> types = dao.getHolidayType();
     typeHoliday = new JComboBox<>(types.toArray(new HolidayType[0]));

     JPanel data = new JPanel();
     data.setLayout(new BorderLayout());

     JPanel entries = new JPanel();
     entries.setLayout(new GridLayout(4,2));

     entries.add(new JLabel("Nom␣de␣l'employe:"));
     entries.add(employees);
     entries.add(new JLabel("Type:"));
     entries.add(typeHoliday);
     entries.add(new JLabel("Date␣de␣debut:"));
     entries.add(dateDebut);
```

```
    entries.add(new JLabel("Date de fin:"));
    entries.add(dateFin);

    data.add(entries, BorderLayout.CENTER);


    tableModel = new DefaultTableModel(dao.getHoliday(), tableColumns);
    tbl = new JTable(tableModel);
    tbl.setFillsViewportHeight(true);

    //hide the idEmp row
    hideEmpColumn();

    JScrollPane scroll = new JScrollPane(tbl);
    tbl.setPreferredScrollableViewportSize(new Dimension(700, 150));
       // Adjust as needed
    JPanel table = new JPanel();
    table.add(scroll);

    data.add(table, BorderLayout.SOUTH);

    add(data, BorderLayout.CENTER);

    JPanel buttons = new JPanel();
    buttons.setLayout(new FlowLayout());
    buttons.add(addButton);
    buttons.add(updateButton);
    buttons.add(deleteButton);
    buttons.add(importButton);
    buttons.add(exportButton);

    add(buttons, BorderLayout.SOUTH);

}
```

Listing 11 – La vue pour Holiday

## 3.5 HolidayController.java

```
//add a new employee
view.getAddButton().addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {

        if (!view.validateFields()) return;

        Employe employe = extractEmployeeFromFields();

        try {
            employeModel.addEmployee(employe);
            Object[][] data = employeModel.getAllEmployees();
            view.refreshTable(data);
            JOptionPane.showMessageDialog(view, "Employee added
               successfully!", "Success", JOptionPane.
               INFORMATION_MESSAGE);
        } catch (Exception ex) {
```

```
            JOptionPane.showMessageDialog(view, "Failed to add employee
                : " + ex.getMessage(), "Error", JOptionPane.
                ERROR_MESSAGE);
        }
    }
});
```

Listing 12 – Action de la boutton Ajouter

# 4. JTabbedPane

En ce qui concerne cette classe, il est ajouté pour combiner les deux vues (Employe, Holiday) dans une seule fenêtre avec une navigation simple et facile

```
public class MainView extends JFrame{

        private JTabbedPane tabbedPane;

        public MainView(EmployeeView viewEmp, HolidayView viewHol) {

                setTitle("Application de gestion");
                setSize(850, 550);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setLocationRelativeTo(null);

                tabbedPane = new JTabbedPane();

                tabbedPane.addTab("Employees", viewEmp);
                tabbedPane.addTab("Congees", viewHol);

                add(tabbedPane);

                setVisible(true);

        }

}
```

Listing 13 – le corps de la classe

# 5. Importer/Exporter

Pour cette option deux bouttons sont ajoutées à la vue de l'employée pour importer ou exporter ses données. En plus, d'autres deux bouttons sont ajoutées à la vue des congées pour importer ou exporter les informations des employées ainsi que ses congées déclarés. La gestion de ces boutton est faite dans les classes de l'employer et holiday.

### 0.0.1 5.1 Interface générique

Une interface générique est conçue pour ces deux fonctionnalitées et implémentée par Employe et Holiday pour centraliser les methodes à utiliser

```
public interface DataImportExport<T> {

    void importData(String fileName) throws IOException;

    void exportData(String fileName, List<T> data) throws IOException;

}
```

Listing 14 – Le corps de l'interface

### 0.0.2 5.2 Données employées

```
public void exportData(String filePath, List<Employe> data) throws
   IOException {

    try(BufferedWriter writer = new BufferedWriter(new FileWriter(
       filePath))) {

        writer.write("Nom,␣Prenom,␣Email,␣Tel,␣Salaire,␣Poste,␣Role");
        writer.newLine();
        for (Employe employe : data) {
            String line = String.format("%s,␣%s,␣%s,␣%s,␣%d,␣%s,␣%s",
                    employe.getNom(),
                    employe.getPrenom(),
                    employe.getEmail(),
                    employe.getTel(),
                    employe.getSalaire(),
                    employe.getPosteSt(),
                    employe.getRoleSt()
                    );
            writer.write(line);
            writer.newLine();
        }

    }

}
```

Listing 15 – le methode pour exporter les données des employées

### 0.0.3 5.3 Données employées avec congées

```
public void importData(String filePath) throws IOException {
    String sql = "Insert␣Into␣Employe␣(nom,␣prenom,␣email,␣tel,␣salaire
       ,␣Poste,␣Role␣)␣values␣(?,?,?,?,?,?,?)";

    try(BufferedReader reader = new BufferedReader(new FileReader(
       filePath)); PreparedStatement pstmt = conn.prepareStatement(sql)
       ) {
        String line;
        while((line = reader.readLine()) != null) {
            System.out.println("First␣Line:␣" + line);
            if (line == null) {
                System.out.println("The␣file␣is␣empty.");
            }
            String[] data = line.split(",");
            System.out.println("Data␣length:␣" + data.length);
            if(data.length == 11) {
                System.out.println("Processing␣line:␣" + line);
                pstmt.setString(1, data[0].trim());
                pstmt.setString(2, data[1].trim());
                pstmt.setString(3, data[2].trim());
                pstmt.setString(4, data[3].trim());
                pstmt.setString(5, data[4].trim());
                pstmt.setString(6, data[5].trim());
                pstmt.setString(7, data[6].trim());
                pstmt.addBatch();
                Holiday holiday = new Holiday(
                        data[7].trim(),
                        data[8].trim(),
                        HolidayType.valueOf(data[9].trim()),
                        data[10].trim()
                        );
                add(holiday);
            } else {
                System.out.println("Skipping␣line␣due␣to␣incorrect␣
                   number␣of␣data␣fields");
            }
        }
        pstmt.executeBatch();
        System.out.println("Holiday␣imported␣successfully!");
    } catch (IOException | SQLException e) {
        e.printStackTrace();
    }
}
```

Listing 16 – La methode pour importer les données des employées aves ses congées

# 6. Se connecter/Ajouter compte

### 0.0.4 6.1 Se connecter

Cette partie est traité independamment avec tous les classes appropriée(Model, view, cntrolleur...), et est conçue d'apparaître en premier et alors si les identifiants sont correctes passer à la fenêtre de gestion des employées et congées avec les droits spécifiés en fonction de son role sinon un message d'erreur apparaîtra(ceci est traité dans le main).

```
view.getLoginButton().addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String user = view.getUser().getText();
        String pass = view.getPass().getText();

        try {
            // Validate user and get the role
            userRole = model.validateUserAndGetRole(user, pass);

            if (userRole != null) {
                // Pass the role to EmployeeView and HolidayView
                employeeView.setUserRole(userRole);
                holidayView.setUserRole(userRole);

                // Notify login success
                if (loginSuccessListener != null) {
                    loginSuccessListener.run(); // Notify that login is
                        successful
                }
            } else {
                JOptionPane.showMessageDialog(view, "Invalid␣login␣
                    credentials!", "Error", JOptionPane.ERROR_MESSAGE);
            }

        } catch (Exception ex) {
            JOptionPane.showMessageDialog(view, "Login␣failed:␣" + ex.
                getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
});
```

Listing 17 – Le processus declenché après la validation des identifiants par l'utilisateur

### 0.0.5 6.2 Ajouter Compte utilisateur

Cette fonction est dédiée au role du Manager seulement, il le permet de choisir depuis un JComboBox l'employe auquel il veut créer un compte puis entrer le nom d'utilisateur, et le most de passe associé à ce compte. Ceci permet de créer une nouvelle ligne dans la table login dans la base de données avec un username et un mot de passe hashé pour des raisons des sécurit.

```
view.getAddAccButton().addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        ArrayList<Employe> emps;
```

```
        try {
            emps = employeModel.ListEmployes_NomPre();
            // Pass a callback to handle the data submitted by the user
            view.createUserAcc(emps, (data) -> {
                // This is the callback that will be invoked when the
                    user submits the form
                String employee = data.get("employee");
                String username = data.get("username");
                String password = data.get("password");

                int idEmp = employeModel.getEmpId(employee);
                String hashedPassword = BCrypt.hashpw(password, BCrypt.
                    gensalt());
                // Call the model method to add the user to the
                    database
                boolean success = employeModel.addUser(idEmp, username,
                     hashedPassword);

                if (success) {
                     JOptionPane.showMessageDialog(view, "User␣added␣
                        successfully!");
                } else {
                     JOptionPane.showMessageDialog(view, "Failed␣to␣add␣
                        user.");
                }

                // Here you can call methods on the model to save the
                    data
                // For example: employeModel.addUserAccount(employee,
                    username, password);
            });
        } catch (Exception e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }
});
```

Listing 18 – Action de la boutton créer compte

# 7. Main

La classe Main joue un rôle central dans l'application. Elle est responsable de créer les instances nécessaires et d'appeler les constructeurs appropriés. De plus, elle gère l'enchaînement logique entre l'affichage de la fenêtre de connexion et celle de gestion, assurant ainsi une expérience utilisateur fluide.

```
public class Main {
    public static void main(String[] args) {
        // Initialize login components
        loginDAOImpl dao = new loginDAOImpl();
        loginModel loginModel = new loginModel(dao);
        loginView loginView = new loginView();

        // Initialize the DAO for Employee
        EmployeeDAOImpl employeeDAO = new EmployeeDAOImpl();
        EmployeModel employeModel = new EmployeModel(employeeDAO);
        EmployeeView employeeView = new EmployeeView(); // Create
            employee view

        // Initialize the DAO for Holiday
        HolidayDAOImpl holidayDAO = new HolidayDAOImpl();
        HolidayModel holidayModel = new HolidayModel(holidayDAO);
        HolidayView holidayView = new HolidayView(); // Create holiday
            view

        // Initialize the controllers
        loginController loginController = new loginController(
            loginModel, loginView, employeeView, holidayView);
        EmployeeController employeeController = new EmployeeController(
            employeModel, employeeView);
        HolidayController holidayController = new HolidayController(
            holidayModel, holidayView);

        // Show the login view first
        loginView.setVisible(true);

        // Initialize the MainView after successful login
        // Here, you'll switch to MainView after login success
        loginController.setLoginSuccessListener(new Runnable() {
            @Override
            public void run() {
                // Hide the login view
                loginView.setVisible(false);

                // Show the MainView with EmployeeView and HolidayView
                    in tabs
                new MainView(employeeView, holidayView).setVisible(true
                    );
            }
        });
    }
}
```

Listing 19 – Main