Compte Rendu de Projet
Programmation Orientée Objet

Gestion des Salaires des Employés

RÉALISÉ PAR : Yassir KHARCHOUFI
ENCADRÉ PAR : Mme Ilhame KACHBAL
2025/2026

GROUPE N°2
OPTION : Intelligence Artificielle

École Supérieure de Technologies - Safi

Table des matières

1 Introduction

Ce projet a pour objectif de développer une application de gestion des salaires des employés en utilisant les principes de la Programmation Orientée Objet (POO). L'application permet de gérer les informations des employés, de calculer leurs salaires, et de persister les données dans une base de données MySQL. L'interface utilisateur est développée avec JavaFX, offrant une expérience utilisateur moderne et intuitive.

2 Architecture du Projet

2.1 Structure des Packages

L'application suit une architecture simple avec toutes les classes dans le package par défaut :

- `Main.java` : Point d'entrée de l'application
- `Controller.java` : Contrôleur de l'interface JavaFX
- `Employe.java` : Modèle de données représentant un employé
- `DBConnection.java` : Gestionnaire de connexion à la base de données
- `interface.fxml` : Définition de l'interface utilisateur

2.2 Interface Utilisateur

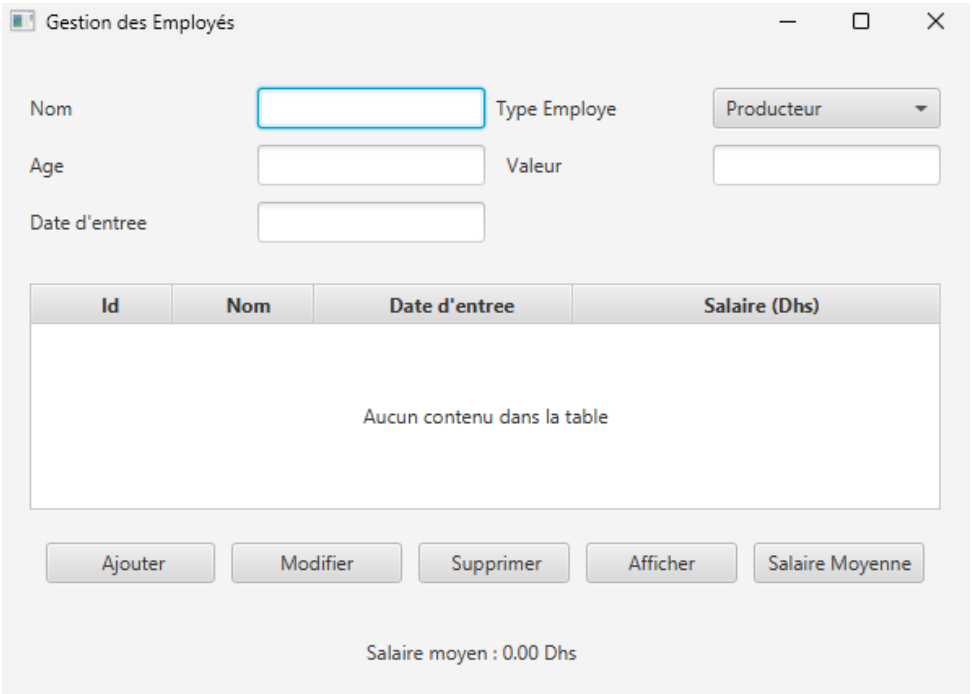


FIGURE 1 – Interface principale de l'application

2.3 Diagramme d'Architecture

Couche	Composants
Présentation (Vue)	interface.fxml, Main.java
Contrôleur	Controller.java
Modèle	Employe.java
Données	DBConnection.java, MySQL

FIGURE 2 – Architecture MVC simplifiée de l'application

3 Analyse des Classes

3.1 Classe Main

```java
public class Main extends Application {

    @Override
    public void start(Stage stage) {
        try {
            Parent root = FXMLLoader.load(getClass().getResource( name: "interface.fxml"));
            if (root == null) {
                throw new RuntimeException("Impossible de charger le fichier FXML: interface.fxml non trouvé");
            }
            Scene scene = new Scene(root);
            stage.setScene(scene);
            stage.setTitle("Gestion des Employés");
            stage.show();
        } catch (Exception e) {
            System.err.println("Erreur lors du chargement du fichier FXML: " + e.getMessage());
            System.err.println("Trace complète:");
            for (StackTraceElement element : e.getStackTrace()) {
                System.err.println("  at " + element);
            }
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

FIGURE 3 – Code de la classe Main - Point d'entrée de l'application

Rôle : Cette classe est le point d'entrée de l'application JavaFX. Elle initialise l'interface graphique en chargeant le fichier FXML.

3.2 Classe Employe (Modèle)

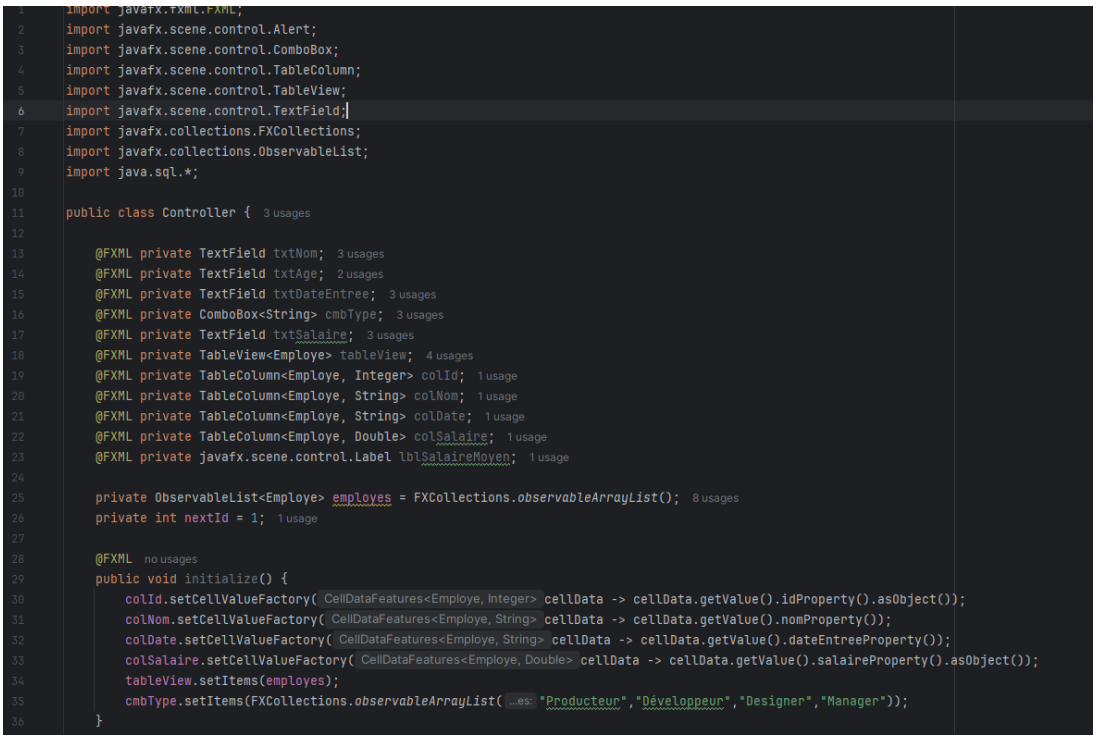
```
public class Employe {  13 usages
    private final IntegerProperty id;  5 usages
    private final StringProperty nom;  5 usages
    private final StringProperty dateEntree;  5 usages
    private final DoubleProperty salaire;  5 usages

    public Employe(int id, String nom, String dateEntree, double salaire) {  3 usages
        this.id = new SimpleIntegerProperty(id);
        this.nom = new SimpleStringProperty(nom);
        this.dateEntree = new SimpleStringProperty(dateEntree);
        this.salaire = new SimpleDoubleProperty(salaire);
    }

    // Getters et Setters pour les propriétés
    public int getId() {  2 usages
        return id.get();
    }

    public void setId(int id) {  no usages
        this.id.set(id);
    }

    public IntegerProperty idProperty() {  1 usage
        return id;
    }
```

FIGURE 4 – Code de la classe Employe - Modèle de données

Caractéristiques :

- Utilise les **JavaFX Properties** pour le data binding
- Attributs : id, nom, date d'entrée, salaire
- Implémente les méthodes getters/setters et les propriétés pour chaque attribut
- Méthode `toString()` pour l'affichage

3.3 Classe Controller



```
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import java.sql.*;

public class Controller {  3 usages

    @FXML private TextField txtNom;  3 usages
    @FXML private TextField txtAge;  2 usages
    @FXML private TextField txtDateEntree;  3 usages
    @FXML private ComboBox<String> cmbType;  3 usages
    @FXML private TextField txtSalaire;  3 usages
    @FXML private TableView<Employe> tableView;  4 usages
    @FXML private TableColumn<Employe, Integer> colId;  1 usage
    @FXML private TableColumn<Employe, String> colNom;  1 usage
    @FXML private TableColumn<Employe, String> colDate;  1 usage
    @FXML private TableColumn<Employe, Double> colSalaire;  1 usage
    @FXML private javafx.scene.control.Label lblSalaireMoyen;  1 usage

    private ObservableList<Employe> employes = FXCollections.observableArrayList();  8 usages
    private int nextId = 1;  1 usage

    @FXML  no usages
    public void initialize() {
        colId.setCellValueFactory( CellDataFeatures<Employe, Integer> cellData -> cellData.getValue().idProperty().asObject());
        colNom.setCellValueFactory( CellDataFeatures<Employe, String> cellData -> cellData.getValue().nomProperty());
        colDate.setCellValueFactory( CellDataFeatures<Employe, String> cellData -> cellData.getValue().dateEntreeProperty());
        colSalaire.setCellValueFactory( CellDataFeatures<Employe, Double> cellData -> cellData.getValue().salaireProperty().asObject());
        tableView.setItems(employes);
        cmbType.setItems(FXCollections.observableArrayList( ...es: "Producteur","Développeur","Designer","Manager"));
    }
```

FIGURE 5 – Code de la classe Controller - Partie 1 (Déclarations et initialize)

```java
@FXML  no usages
private void ajouterAction() {
    try {
        String nom = txtNom.getText();
        Integer.parseInt(txtAge.getText());
        String dateEntree = txtDateEntree.getText();
        String type = cmbType.getValue();
        double salaire = Double.parseDouble(txtSalaire.getText());

        if (nom.isEmpty() || type == null) {
            showError( t: "Erreur",  m: "Veuillez remplir tous les champs!");
            return;
        }

        Employe employe = new Employe(nextId++, nom, dateEntree, salaire);
        employes.add(employe);
        tableView.setItems(employes);

        txtNom.clear();
        txtAge.clear();
        txtDateEntree.clear();
        cmbType.setValue(null);
        txtSalaire.clear();

        System.out.println("Tentative de connexion MySQL...");

        Connection con = DBConnection.getConnection();

        if(con == null){
            System.out.println("ECHEC CONNEXION MYSQL !");
            return;
        }
        System.out.println("Connexion réussie !");

        String sql = "INSERT INTO employe(nom, date_entree, salaire) VALUES (?, ?, ?)";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setString( parameterIndex: 1, nom);
        ps.setString( parameterIndex: 2, dateEntree);
        ps.setDouble( parameterIndex: 3, salaire);
```

FIGURE 6 – Code de la classe Controller - Partie 2 (Méthode ajouterAction)

```java
@FXML no usages
private void modifierAction() {
    Employe e = tableView.getSelectionModel().getSelectedItem();
    if (e == null) return;

    try {
        Connection con = DBConnection.getConnection();
        String sql = "UPDATE employe SET nom=?, date_entree=?, salaire=? WHERE id=?";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setString( parameterIndex: 1, txtNom.getText());
        ps.setString( parameterIndex: 2, txtDateEntree.getText());
        ps.setDouble( parameterIndex: 3, Double.parseDouble(txtSalaire.getText()));
        ps.setInt( parameterIndex: 4, e.getId());
        ps.executeUpdate();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

@FXML no usages
private void supprimerAction() {
    Employe e = tableView.getSelectionModel().getSelectedItem();
    if (e == null) return;

    try {
        Connection con = DBConnection.getConnection();
        String sql = "DELETE FROM employe WHERE id=?";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setInt( parameterIndex: 1, e.getId());
        ps.executeUpdate();
        con.close();
        employes.remove(e);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

@FXML no usages
```

FIGURE 7 – Code de la classe Controller - Partie 3 (Autres méthodes CRUD)

Fonctionnalités implémentées :

- **Ajouter** : Ajoute un nouvel employé avec validation
- **Modifier** : Met à jour les informations d'un employé
- **Supprimer** : Supprime un employé sélectionné
- **Afficher** : Récupère tous les employés depuis la BD
- **Salaire Moyen** : Calcule et affiche le salaire moyen

3.4 Classe DBConnection

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

public class DBConnection {  4 usages

    private static final String URL = "jdbc:mysql://localhost:3306/BDTest";  1 usage
    private static final String USER = "root";  1 usage
    private static final String PASSWORD = "";  1 usage

    public static Connection getConnection() {  4 usages
        try {
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

FIGURE 8 – Code de la classe DBConnection - Gestion de la connexion MySQL

Configuration :

- Base de données : MySQL sur localhost :3306
- Nom de la BD : `BDTest`
- Utilisateur : `root`
- Mot de passe : vide
- Driver : MySQL Connector/J

3.5 Code FXML (Extrait)

```
<?import javafx.scene.control.ComboBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.layout.VBox?>

<VBox maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" spacing="20.0" xmlns="http://javafx.com/javafx/25" xmlns:fx="http://javafx.com/fxml/1" fx:co
    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
    </padding>
    <children>
        <GridPane prefHeight="107.0" prefWidth="560.0">
            <columnConstraints>
                <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
            </columnConstraints>
            <rowConstraints>
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
            </rowConstraints>
            <children>
                <Label prefHeight="23.0" prefWidth="141.0" text="Nom *" />
                <TextField fx:id="txtNom" GridPane.columnIndex="1" />
                <Label prefHeight="23.0" prefWidth="141.0" text="Age" GridPane.rowIndex="1" />
                <Label prefHeight="23.0" prefWidth="141.0" text="Date d'entree" GridPane.rowIndex="2" />
                <TextField fx:id="txtAge" GridPane.columnIndex="1" GridPane.rowIndex="1" />
                <TextField fx:id="txtDateEntree" prefHeight="12.0" prefWidth="140.0" GridPane.columnIndex="1" GridPane.rowIndex="2" />
                <Label prefHeight="23.0" prefWidth="141.0" text="  Type Employe" GridPane.columnIndex="2" />
                <ComboBox fx:id="cmbType" prefWidth="150.0" promptText="Producteur" GridPane.columnIndex="3" />
                <Label prefHeight="23.0" prefWidth="141.0" text="    Valeur" GridPane.columnIndex="2" GridPane.rowIndex="1" />
                <TextField fx:id="txtSalaire" GridPane.columnIndex="3" GridPane.rowIndex="1" />
            </children>
        </GridPane>
```

FIGURE 9 – Code FXML - Partie 1 (Structure générale)

```
        <TableView fx:id="tableView" minWidth="0.0" prefHeight="200.0" prefWidth="200.0">
            <columns>
                <TableColumn fx:id="colId" prefWidth="87.0" text="   Id" />
                <TableColumn fx:id="colNom" prefWidth="87.0" text="  Nom" />
                <TableColumn fx:id="colDate" prefWidth="159.0" text="   Date d'entree" />
                <TableColumn fx:id="colSalaire" minWidth="0.0" prefWidth="226.0" text="  Salaire (Dhs)" />
            </columns>
            <VBox.margin>
                <Insets />
            </VBox.margin>
        </TableView>
        <HBox prefHeight="100.0" prefWidth="200.0">
            <children>
                <Button mnemonicParsing="false" onAction="#ajouterAction" prefHeight="25.0" prefWidth="104.0" text=" Ajouter" translateX="10.0" />
                <Button mnemonicParsing="false" onAction="#modifierAction" prefHeight="25.0" prefWidth="105.0" text="Modifier" translateX="20.0" />
                <Button mnemonicParsing="false" onAction="#supprimerAction" prefHeight="25.0" prefWidth="93.0" text=" Supprimer" translateX="30.0" />
                <Button mnemonicParsing="false" onAction="#afficherAction" prefHeight="25.0" prefWidth="93.0" text="Afficher" translateX="40.0" />
                <Button mnemonicParsing="false" onAction="#salaireMoyenAction" prefHeight="25.0" prefWidth="105.0" text="Salaire Moyenne" translateX="50.0" />
            </children>
        </HBox>
        <HBox prefHeight="70.0" prefWidth="560.0">
            <children>
                <!-- fx:id ajouté pour que Controller puisse mettre à jour le label -->
                <Label fx:id="lblSalaireMoyen" prefHeight="19.0" prefWidth="546.0" text="                                                                                Salaire moyen : 0.
            </children>
        </HBox>
    </children>
</VBox>
```

FIGURE 10 – Code FXML - Partie 2 (TableView et boutons)

4 Interface Utilisateur - Composants

4.1 Champs du Formulaire

Champ	Type	Description
Nom	TextField	Nom complet de l'employé
Âge	TextField	Âge de l'employé
Date d'entrée	TextField	Date d'embauche (format texte)
Type Employé	ComboBox	Sélection : Producteur, Développeur, Designer, Manager
Valeur	TextField	Salaire de l'employé

TABLE 1 – Champs du formulaire de saisie

4.2 Boutons d'Action

Bouton	Fonction
Ajouter	Insère un nouvel employé dans la BD
Modifier	Met à jour l'employé sélectionné
Supprimer	Supprime l'employé sélectionné
Afficher	Récupère tous les employés depuis la BD
Salaire Moyenne	Calcule et affiche le salaire moyen

TABLE 2 – Fonctions des boutons d'action

5 Base de Données

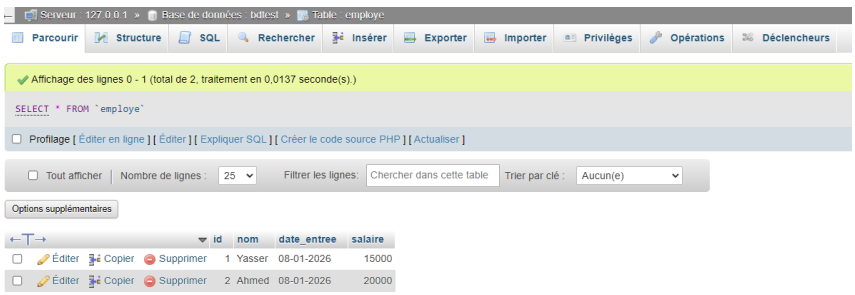


FIGURE 11 – Structure de la table 'employe' dans MySQL

5.1 Structure de la Table Employe

```
1 CREATE TABLE employe (
2     id INT PRIMARY KEY AUTO_INCREMENT,
3     nom VARCHAR(100) NOT NULL,
```

```
    date_entree VARCHAR(20),
    salaire DECIMAL(10, 2)
);
```

5.2 Opérations CRUD Implémentées

— **Create** : INSERT INTO employe(nom, date_entree, salaire) VALUES (?, ?, ?)
— **Read** : SELECT * FROM employe
— **Update** : UPDATE employe SET nom=?, date_entree=?, salaire=? WHERE id=?
— **Delete** : DELETE FROM employe WHERE id=?

6 Résultats d'Exécution

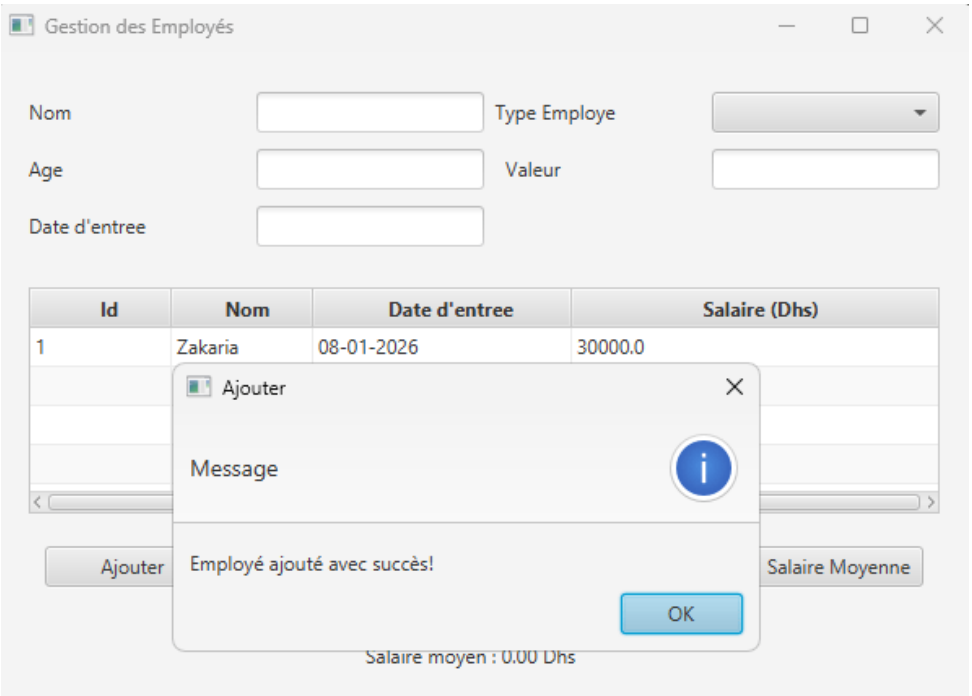


FIGURE 12 – Application en cours d'exécution avec données

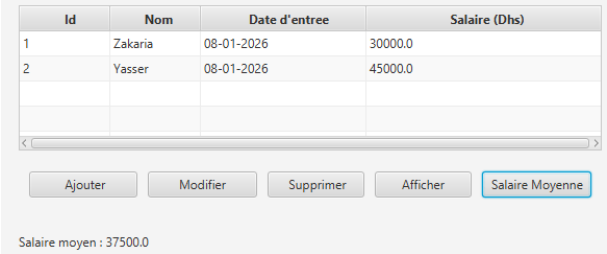


FIGURE 13 – Calcul du salaire moyen

7 Concepts POO Utilisés

7.1 Encapsulation

— Les attributs de la classe `Employe` sont privés
— Accès via des getters/setters
— Utilisation de JavaFX Properties pour le data binding

7.2 Data Binding avec JavaFX

— Liaison entre les propriétés JavaFX et l'interface
— Mise à jour automatique de l'interface lors des changements
— Utilisation de `CellValueFactory` pour la TableView

7.3 Gestion des Événements

— Annotations `@FXML` pour lier les composants
— Méthodes de gestion d'événements (ex : `#ajouterAction`)
— Validation des entrées utilisateur

8 Fonctionnalités Principales

8.1 Gestion Complète des Employés

— Ajout avec validation des champs
— Modification des informations existantes
— Suppression avec confirmation implicite
— Affichage en temps réel dans la TableView

8.2 Calculs et Statistiques

— Calcul du salaire moyen des employés
— Affichage formaté dans un Label
— Mise à jour dynamique lors des modifications

8.3 Persistance des Données

— Sauvegarde automatique dans MySQL

— Récupération au démarrage
— Transactions CRUD complètes

9 Difficultés Rencontrées et Solutions

9.1 Connexion à la Base de Données

— **Problème** : Échec de connexion MySQL
— **Solution** : Vérification des paramètres de connexion, ajout de logs détaillés, gestion des exceptions

9.2 Data Binding JavaFX

— **Problème** : TableView non mise à jour automatiquement
— **Solution** : Utilisation d'`ObservableList` et des JavaFX Properties

9.3 Gestion des Types d'Employés

— **Problème** : ComboBox non peuplée correctement
— **Solution** : Initialisation dans la méthode `initialize()` du contrôleur

9.4 Validation des Données

— **Problème** : Entrées utilisateur non validées
— **Solution** : Ajout de vérifications et messages d'erreur

10 Améliorations Possibles

10.1 Court Terme

— Ajouter la validation des dates avec `DatePicker`
— Implémenter la recherche/filtrage des employés
— Ajouter l'exportation des données en PDF/Excel
— Améliorer la gestion des erreurs avec des messages utilisateur

10.2 Moyen Terme

— Implémenter l'authentification des utilisateurs
— Ajouter des rôles et permissions
— Créer des rapports statistiques avancés
— Internationalisation (multi-langues)

10.3 Long Terme

— Architecture multi-couches avec DAO pattern
— Tests unitaires avec JUnit
— Déploiement en ligne avec Java Web Start
— Interface responsive pour mobile

11 Conclusion

Ce projet a permis de mettre en pratique les concepts fondamentaux de la Programmation Orientée Objet avec Java, JavaFX et MySQL. L'application fonctionnelle démontre la maîtrise des compétences suivantes :

— Développement d'interfaces graphiques avec JavaFX
— Conception de modèles de données avec encapsulation
— Persistance des données avec JDBC et MySQL
— Implémentation du pattern MVC
— Gestion des événements utilisateur
— Validation et gestion des erreurs

L'application "Gestion des Salaires" constitue une base solide pour des projets plus complexes et démontre une compréhension approfondie du développement d'applications desktop avec Java.

12 Annexes

12.1 Code Source Complet

Le code source complet est disponible dans les fichiers fournis :
— `Main.java` - Point d'entrée
— `Controller.java` - Logique de contrôle
— `Employe.java` - Modèle de données
— `DBConnection.java` - Connexion BD
— `interface.fxml` - Interface utilisateur

12.2 Prérequis Techniques

— Java JDK 8+
— JavaFX SDK
— MySQL Server 5.7+
— MySQL Connector/J
— IDE Eclipse ou IntelliJ avec plugin JavaFX

12.3 Instructions d'Installation

1. Installer MySQL et créer la base `BDTest`
2. Exécuter le script SQL pour créer la table `employe`
3. Configurer le projet avec les bibliothèques JavaFX
4. Ajouter le driver MySQL Connector/J au classpath
5. Exécuter la classe `Main`

12.4 Références

— Documentation JavaFX : `https://openjfx.io`
— Tutoriel JDBC Oracle : `https://docs.oracle.com/javase/tutorial/jdbc/`
— MySQL Documentation : `https://dev.mysql.com/doc/`
— JavaFX Properties : `https://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm`