#### Université Cadi Ayyad

#### École Supérieure De Technologie-Safi Département : Informatique

#### Filière : génie informatique

**Gérer des employés**

Rapport du TP java avance 1

**Réalisé par :** LOUIASS salma

**Encadré par :** Mme. ELKOURCHI asmaa

**Année Universitaire** : 2024/2025

**Table des matières**

[Inroduction](#_bookmark0) 4

[Outils & environnement de travail](#_bookmark1) 5

1. [Environnement de travail](#_bookmark2) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2. [Outils de travail](#_bookmark4) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
3. [Language de Programmation](#_bookmark8) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
4. [Réalisation](#_bookmark10) 7
   1. [Création de la base de donnée](#_bookmark11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
      1. [Script base de donnée](#_bookmark12) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   2. [Architecture MVC (Model-View-Controller)](#_bookmark13) . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
      1. [Implémentation du modèle (couche Model)](#_bookmark14) . . . . . . . . . . . . . . . . . . . . . . . 8
      2. [Gestion des données (DAO)](#_bookmark15) 11
      3. [Contrôleur (couche Controller)](#_bookmark16) 13
      4. [Interface graphique (couche View)](#_bookmark17) 15
      5. [Main](#_bookmark18) 20
5. [Résultats](#_bookmark19) 21
   1. [Ajout](#_bookmark20) 21
   2. [Modification](#_bookmark23) 22
   3. [Suppression](#_bookmark24) 22

**Table des figures**

##### [intellij idea logo](#_bookmark3) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

##### [MySQL Workbench logo](#_bookmark5) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

##### [xampp logo](#_bookmark6) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

##### [java developpement kit logo](#_bookmark7) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

##### [java logo](#_bookmark9) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

##### [Interface Utilisateur](#_bookmark21) 21

##### [Affichage de l’Ajout](#_bookmark22) 21

##### [Affichage de modification](#_bookmark25) 22

##### [Affichage de suppression](#_bookmark26) 22

# Inroduction

##### Ce travail pratique (TP) vise le développement d’une application Java de gestion des employés en adop- tant l’architecture MVC (Modèle-Vue-Contrôleur), tout en mettant en œuvre des interfaces graphiques à l’aide de la bibliothèque Swing.

##### Ce projet s’inscrit dans une démarche pédagogique qui combine l’apprentissage des bases de la programma- tion orientée objet (POO) et des bonnes pratiques de conception logicielle. L’objectif est de concevoir une application intuitive et modulaire, permettant de gérer efficacement les données des employés à travers des fonctionnalités telles que l’ajout, la modification, la suppression et l’affichage des informations.

##### Grâce à une structure bien définie et à une séparation claire des responsabilités, ce TP offre une opportunité d’approfondir les notions de maintenabilité et d’évolutivité, tout en préparant les étudiants à des développe- ments logiciels plus avancés.

# Outils & environnement de travail

## Environnement de travail



##### Figure 1 – intellij idea logo

’

##### **IntelliJ IDEA** : est un IDE développé par JetBrains, conçu principalement pour Java, mais prenant en charge d’autres langages comme Kotlin, Scala et Python. Il se distingue par ses outils intelligents, sa complétion de code avancée, son analyse en temps réel et son intégration avec des outils comme Maven, Gradle et Git, offrant une productivité accrue aux développeurs.

## Outils de travail



##### Figure 2 – MySQL Workbench logo

##### **MySQL Workbench** : est un outil graphique développé pour simplifier la conception, l’administra- tion et la gestion des bases de données MySQL. Il offre une interface intuitive qui permet de manipu- ler les bases de données visuellement, sans dépendre uniquement des commandes en ligne, rendant les tâches plus accessibles et efficaces.

##### SI



##### Figure 3 – xampp logo

##### **XAMPP** : est une solution open-source qui regroupe Apache, MySQL, PHP, et Perl pour faciliter le développement et le déploiement d’applications web. Il est conçu pour créer un environnement de ser- veur local rapide et simple, adapté aux développeurs souhaitant tester ou gérer des projets web sur leur machine avant le déploiement.



##### Figure 4 – java developpement kit logo

##### **java developpement kit** : est un ensemble d’outils logiciels indispensables pour le développement d’applications Java. Il comprend des composants essentiels tels qu’un compilateur, une machine vir- tuelle Java (JVM), des bibliothèques standard, et des outils de débogage, permettant de coder, compi- ler, exécuter et tester des programmes Java efficacement.

## Language de Programmation



##### Figure 5 – java logo

##### **Java** : un langage de programmation orienté objet et une plateforme largement utilisée pour le déve- loppement d’applications logicielles. Il a été créé par Sun Microsystems (maintenant propriété d’Oracle) en 1995 et reste l’un des langages les plus populaires au monde, notamment pour les applications d’entreprise, le développement mobile (Android) et les applications web.

**Réalisation**

## Création de la base de donnée

### Script base de donnée

1

create database DBperson; use DBperson;

CREATE TABLE employers (

id INT AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(100), last\_name VARCHAR(100), email VARCHAR(50),

phone INT(10), salary NUMERIC, role VARCHAR(50),

poste VARCHAR(50)

);

CREATE TABLE Role (

name varchar(30) NOT NULL

);

INSERT INTO Role (name) VALUES ('Admin'),('Employee');

INSERT INTO Poste (name) VALUES ('Ingenieur'),('Team\_Leader'),('Pilote');

CREATE TABLE Poste (

name varchar(30) NOT NULL

);

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

##### Listing 1.1 – Script SQL de la base de données

•

## Architecture MVC (Model-View-Controller)

##### L’architecture MVC est un modèle de conception qui sépare les responsabilités au sein d’une application, faci- litant ainsi la gestion et la maintenance du code. Elle repose sur trois composants principaux :

### Implémentation du modèle (couche Model)

#### Employer

1

package model; import enums.\*;

public class Employer {

private int id;

private String firstName; private String lastName; private String email; private int phoneNumber; private double salary; private Role role; private Poste poste;

public Employer(int id, String firstName, String lastName, String email, int phoneNumber, double salary, Role role, Poste poste) {

this.id = id; this.firstName = firstName; this.lastName = lastName; this.email = email;

this.phoneNumber = phoneNumber; this.salary = salary;

this.role = role; this.poste = poste;

}

public int getId() { return id;

}

public String getFirstName() { return firstName;

}

public String getLastName() { return lastName;

}

public String getEmail() { return email;

}

public int getPhoneNumber() {

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

return phoneNumber;

}

public double getSalary() { return salary;

}

public Role getRole() { return role;

}

public Poste getPoste() { return poste;

}

}

45

46

47

48

49

50

51

52

53

54

55

56

57

58

#### EmployerLogic

1

package model;

import enums.\*;

import dao.EmployerDAO; import java.util.List;

public class EmployerLogic { private EmployerDAO dao;

public EmployerLogic (EmployerDAO dao) { this.dao = dao;

}

public boolean addEmployer(int id, String firstName, String lastName, String email, int phoneNumber, double salary, Role role, Poste poste) {

// Vérifie si l'email est valide avant de mettre à jour l'employé.

if ( isValidEmail(email) ) {

// Création de l'objet Employer avec les informations fournies.

return dao.addEmployer( new Employer( id,

firstName, lastName, email, phoneNumber, salary, role,

poste

));

// Appel à la méthode DAO pour mettre à jour les informations dans la base de données.

}

// Retourne faux si l'email n'est pas valide.

return false;

}

public boolean updateEmployer(int id, String firstName, String lastName, String email, int phoneNumber, double salary, Role role, Poste poste) {

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 36 | // Vérifie que l'email contient "@gmail.com". Simpliste mais fonctionnel. | | | |
| 37 |  | | if ( isValidEmail(email) ) { | |
| 38 |  | |  | |
| 39 |  | | Employer employer = new Employer( | |
| 40 |  | | id, | |
| 41 |  | | firstName, | |
| 42 |  | | lastName, | |
| 43 |  | | email, | |
| 44 |  | | phoneNumber, | |
| 45 |  | | salary, | |
| 46 |  | | role, | |
| 47 |  | | poste | |
| 48 |  | | ); | |
| 49 |  | |  | |
| 50 |  | | return dao.updateEmployer(employer); | |
| 51 |  | | } | |
| 52 |  | |  | |
| 53 |  | | return false; | |
| 54 | } | |  | |
| 55 |  | |  | |
| 56 | | private boolean isValidEmail(String email) { | |  |
| 57 | | return email.contains("@gmail.com") ? true | | : false; |
| 58 | | } | |  |
| 59 | |  | |  |
| 60 | |  | |  |
| 61 | | public boolean deleteEmployer(int id) { | |  |
| 62 | | return dao.deleteEmployer(id); | |  |
| 63 | | } | |  |
| 64 | |  | |  |
| 65 | | public List<Employer> getAllEmployers() { | |  |
| 66 | | return dao.getAllEmployers(); | |  |
| 67 | | } | |  |
| 68 | |  | |  |
| 69 } | |  | |  |

package dao;

import java.sql.\*;

import java.util.ArrayList; import java.util.List; import model.Employer; import enums.Role;

import enums.Poste;

public class EmployerDAO implements EmployerInterface { private Connection connection;

public EmployerDAO() { try {

connection = DBConnection.getConnection();

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

}

}

@Override

public boolean addEmployer(Employer employer) {

try (PreparedStatement addStatement = connection.prepareStatement( "INSERT INTO employers (first\_name, last\_name, email, phone,

salary, role, poste) VALUES (?, ?, ?, ?, ?, ?, ?)")) {

addStatement.setString(1, employer.getFirstName()); addStatement.setString(2, employer.getLastName()); addStatement.setString(3, employer.getEmail());

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/DBperson"; private static final String USERNAME = "root";

private static final String PASSWORD = "";

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

}

}

|  |  |  |
| --- | --- | --- |
| **2.2** |  | **Gestion des données (DAO)** |
|  |  | **DBConnection** |
|  | 1 |  |
|  | 2 |  |
|  | 3 |  |
|  | 4 |  |
|  | 5 |  |
|  | 6 |  |
|  | 7 |  |
|  | 8 |  |
|  | 9 |  |
|  | 10 |  |
|  | 11 |  |
|  | 12 |  |
|  | 13 |  |
|  | 14 |  |
|  | 15 |  |
|  | 16 |  |
|  |  | **EmployerDAO** |
|  | 1 |  |
|  | 2 |  |
|  | 3 |  |
|  | 4 |  |
|  | 5 |  |
|  | 6 |  |
|  | 7 |  |
|  | 8 |  |
|  | 9 |  |
|  | 10 |  |
|  | 11 |  |
|  | 12 |  |
|  | 13 |  |
|  | 14 |  |
|  | 15 |  |
|  | 16 |  |
|  | 17 |  |
|  | 18 |  |
|  | 19 |  |
|  | 20 |  |
|  | 21 |  |
|  | 22 |  |
|  | 23 |  |
|  | 24 |  |
|  | 25 |  |
|  | 26 |  |
|  | 27 |  |
|  | 28 |  |
|  | 29 |  |
|  | 30 |  |

31

32

33

34

35

36

37

38

39

40

41

42 }

43

44

addStatement.setInt(4, employer.getPhoneNumber()); addStatement.setDouble(5, employer.getSalary()); addStatement.setString(6, employer.getRole().name()); addStatement.setString(7, employer.getPoste().name());

return addStatement.executeUpdate() > 0;

} catch (SQLException addException) { addException.printStackTrace(); return false;

}

45 @Override

46 public boolean updateEmployer(Employer employer) {

47 try (PreparedStatement updateStatement = connection.prepareStatement(" UPDATE employers SET first\_name = ?, last\_name = ?, email = ?, phone = ?, salary = ?, role = ?, poste = ? WHERE id = ?")) {

48

49 updateStatement.setString(1, employer.getFirstName());

50 updateStatement.setString(2, employer.getLastName());

51 updateStatement.setString(3, employer.getEmail());

52 updateStatement.setInt(4, employer.getPhoneNumber());

53 updateStatement.setDouble(5, employer.getSalary());

54 updateStatement.setString(6, employer.getRole().name());

55 updateStatement.setString(7, employer.getPoste().name());

56 updateStatement.setInt(8, employer.getId());

57

58 return updateStatement.executeUpdate() > 0;

59

60 } catch (SQLException updateException) {

61 updateException.printStackTrace();

62 return false;

63 }

64 }

65

66 @Override

67 public boolean deleteEmployer(int id) {

68 try (PreparedStatement deleteStatement = connection.prepareStatement(" DELETE FROM employers WHERE id = ?")) {

69

70 deleteStatement.setInt(1, id);

71 return deleteStatement.executeUpdate() > 0;

72

73 } catch (SQLException deleteException) {

74 return false;

75 }

76 }

77

78 @Override

79 public List<Employer> getAllEmployers() {

80 List<Employer> employers = new ArrayList<>();

81 try (ResultSet getResult = connection.prepareStatement("SELECT \* FROM employers").executeQuery()) {

82

while (getResult.next()) { employers.add(new Employer(

getResult.getInt("id"), getResult.getString("first\_name"), getResult.getString("last\_name"), getResult.getString("email"), getResult.getInt("phone"), getResult.getDouble("salary"), Role.valueOf(getResult.getString("role")), Poste.valueOf(getResult.getString("poste"))

));

}

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

}

return employers;

}

}

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

### Contrôleur (couche Controller)

#### EmployerController

1

package controller;

import view.\*; import dao.\*; import model.\*; import enums.\*;

import java.util.List;

import javax.swing.JOptionPane; public class EmployerController {

private FormFrame frame;

private EmployerLogic employerLogic;

public EmployerController(FormFrame frame, EmployerLogic employerLogic) { this.frame = frame;

this.employerLogic = employerLogic;

frame.getBtnPanel().getAddBtn().addActionListener(addEvent -> addEmployer

());

frame.getBtnPanel().getUpdateBtn().addActionListener(updateEvent -> updateEmployer());

frame.getBtnPanel().getRemoveBtn().addActionListener(deleteEvent -> deleteEmployer());

loadEmployers();

}

private void addEmployer() {

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

|  |  |  |  |
| --- | --- | --- | --- |
| 26 | try { | | |
| 27 |  | | |
| 28 | if (employerLogic.addEmployer( | | |
| 29 | 1, | | |
| 30 | frame.getInPanel().getFirstNameField().getText(), | | |
| 31 | frame.getInPanel().getLastNameField().getText(), | | |
| 32 | frame.getInPanel().getEmailField().getText(), | | |
| 33  34 | Integer.parseInt(frame.getInPanel().getTelephoneNumberField().  getText()),  Double.parseDouble(frame.getInPanel().getSalaryField().getText | | |
| 35 | ()),  Role.valueOf(frame.getInPanel().getSelectedRole().toString()), | | |
| 36  37 | Poste.valueOf(frame.getInPanel().getSelectedPoste().toString()  )  )) | | |
| 38 | { | | |
| 39  40 | JOptionPane.showMessageDialog(frame, "Employer added successfully  .");  loadEmployers(); | | |
| 41 | } else { | | |
| 42 | JOptionPane.showMessageDialog(frame, "Failed to add employer."); | | |
| 43 | } | | |
| 44 | } catch (Exception e) { | | |
| 45  46 | JOptionPane.showMessageDialog(frame, "Invalid input: " + e.getMessage  ());  } | | |
| 47 | } | | |
| 48 | // Méthode pour mettre à jour un employé via l'interface utilisateur. | | |
| 49 | private void updateEmployer() { | | |
| 50 | try {  // Récupère les données des champs de saisie et les envoie à la couche logique. | | |
| 51 | if (employerLogic.updateEmployer( | | |
| 52 | frame.getListPanel().getSelectedRowId(), | | |
| 53 | frame.getInPanel().getFirstNameField().getText(), | | |
| 54 | frame.getInPanel().getLastNameField().getText(), | | |
| 55 | frame.getInPanel().getEmailField().getText(), | | |
| 56  57 | Integer.parseInt(frame.getInPanel().getTelephoneNumberField().  getText()),  Double.parseDouble(frame.getInPanel().getSalaryField().getText()) | | |
|  | , | | |
| 58 |  |  | Role.valueOf(frame.getInPanel().getSelectedRole().toString()), |
| 59 |  |  | Poste.valueOf(frame.getInPanel().getSelectedPoste().toString()) |
| 60 |  | )) |  |
| 61 | { |  |  |
| 62  63  64  65  66  67  68  69  70  71  72 | // Affiche un message de succès et recharge la liste des employeurs.  JOptionPane.showMessageDialog(frame, "Employer updated successfully  .");  loadEmployers();  } else {  // Affiche un message d'échec.  JOptionPane.showMessageDialog(frame, "Failed to update employer.");  }  } catch (Exception e) {  // Gère les erreurs d'entrée utilisateur.  JOptionPane.showMessageDialog(frame, "Invalid input: " + e.getMessage());  }  }  private void deleteEmployer() { | | |

|  |  |  |
| --- | --- | --- |
| 73  74  75 | | try {  if (employerLogic.deleteEmployer(frame.getListPanel().getSelectedRowId())  ) {  JOptionPane.showMessageDialog(frame, "Employer deleted successfully  ."); |
|  | 76 | loadEmployers(); |
|  | 77 | } else { |
|  | 78 | JOptionPane.showMessageDialog(frame, "Failed to delete employer."); |
|  | 79 | } |
|  | 80 | } catch (Exception e) { |
|  | 81 | JOptionPane.showMessageDialog(frame, "Invalid input: " + e.getMessage()); |
|  | 82 | } |
|  | 83 | } |
|  | 84 |  |
|  | 85 | private void loadEmployers() { |
|  | 86 | frame.getListPanel().updateEmployerList(employerLogic.getAllEmployers()); |
|  | 87 | } |
|  | 88 | } |
| **2.4** |  | **Interface graphique (couche View)** |
|  |  | **BtnPanel** |
|  | 1 | package view; |
|  | 2 | // Importation des classes nécessaires pour créer une interface graphique Swing. |
|  | 3 | import javax.swing.\*; |
|  | 4 | import java.awt.\*; |
|  | 5 | // Déclaration de la classe BtnPanel qui hérite de JPanel. |
|  | 6 | public class BtnPanel extends JPanel { |
|  | 7 | // Déclaration des boutons qui seront ajoutés au panneau. |
|  | 8 | private JButton addBtn, removeBtn , updateBtn;  // Constructeur de la classe BtnPanel. |
|  | 9  10 | public BtnPanel() {  // Définit la disposition des composants dans le panneau comme FlowLayout (disposition en ligne horizontale). |
|  | 11 | setLayout(new FlowLayout()); |
|  | 12 |  |
|  | 13 | addBtn = new JButton("Add"); |
|  | 14 | removeBtn = new JButton("Remove"); |
|  | 15 | updateBtn = new JButton("Update"); |
|  | 16 |  |
|  | 17 | add(addBtn); |
|  | 18 | add(removeBtn); |
|  | 19 | add(updateBtn); |
|  | 20 | } |
|  | 21 |  |
|  | 22 | public JButton getAddBtn() { |
|  | 23 | return addBtn; |
|  | 24 | } |
|  | 25 |  |
|  | 26 | public JButton getRemoveBtn() { |
|  | 27 | return removeBtn; |
|  | 28 | } |
|  | 29 |  |
|  | 30 | public JButton getUpdateBtn() { |

31

return updateBtn;

}

}

32

33

#### FormFrame

1

package view;

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

public class FormFrame extends JFrame { private InputPanel inPanel;

private BtnPanel btnPanel;

private ListPanel listPanel;

public FormFrame() { setTitle("Person Form"); setSize(800, 600);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); setLayout(new BorderLayout());

inPanel = new InputPanel(); btnPanel = new BtnPanel(); listPanel = new ListPanel();

add(inPanel, BorderLayout.NORTH); add(btnPanel, BorderLayout.SOUTH);

JScrollPane scrollPane = new JScrollPane(listPanel); add(scrollPane, BorderLayout.CENTER);

setVisible(true);

}

public InputPanel getInPanel() { return inPanel;

}

public BtnPanel getBtnPanel() { return btnPanel;

}

public ListPanel getListPanel() { return listPanel;

}

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

#### InputPanel

1 package view;

2

3 import javax.swing.\*;

4 import java.awt.\*;

5 import enums.Role;

6 import enums.Poste;

7

8 public class InputPanel extends JPanel {

9

10 JTextField firstNameField, lastNameField, emailField, telephoneNumberField, salaryField;

11 JComboBox<Role> roleField;

12 JComboBox<Poste> posteField;

13

14 public InputPanel() {

15 setLayout(new GridLayout(7, 2, 5, 5));

16 setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

17

18 firstNameField = new JTextField(15);

19 lastNameField = new JTextField(15);

20 emailField = new JTextField(15);

21 telephoneNumberField = new JTextField(15);

22 salaryField = new JTextField(15);

23 roleField = new JComboBox<>(Role.values());

24 posteField = new JComboBox<>(Poste.values());

25

26 add(new JLabel("First Name"));

27 add(firstNameField);

28

29 add(new JLabel("Last Name"));

30 add(lastNameField);

31

32 add(new JLabel("Email"));

33 add(emailField);

34

35 add(new JLabel("Telephone Number"));

36 add(telephoneNumberField);

37

38 add(new JLabel("Salary"));

39 add(salaryField);

40

41 add(new JLabel("Role"));

42 add(roleField);

43

44 add(new JLabel("Poste"));

45 add(posteField);

46 }

47

48 public JTextField getFirstNameField() {

49 return firstNameField;

50 }

51

52 public JTextField getLastNameField() {

53 return lastNameField;

54 }

55

56

public JTextField getEmailField() { return emailField;

}

public JTextField getTelephoneNumberField() { return telephoneNumberField;

}

public JTextField getSalaryField() { return salaryField;

}

public Role getSelectedRole() {

return (Role) roleField.getSelectedItem();

}

public Poste getSelectedPoste() {

return (Poste) posteField.getSelectedItem();

}

}

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

#### ListPanel

1

package view;

// Importation des bibliothèques nécessaires pour créer une interface graphique Swing.

import javax.swing.\*;

import javax.swing.border.LineBorder; import java.awt.\*;

import java.util.ArrayList;

import java.util.List; import model.Employer;

import java.awt.event.MouseAdapter; import java.awt.event.MouseEvent;

// Déclaration de la classe ListPanel qui hérite de JPanel.

public class ListPanel extends JPanel {

// Déclaration des variables pour contenir les données et gérer les interactions.

private JPanel contentPanel; private int selectedRowId = -1;

private List<JPanel> rowPanels = new ArrayList<>();

// Constructeur de la classe ListPanel.

public ListPanel() {

setLayout(new BorderLayout());

// Définit la disposition principale du panneau comme BorderLayout.

JPanel titlePanel = new JPanel(); titlePanel.setLayout(new GridLayout(1, 5)); titlePanel.add(new JLabel("Id", SwingConstants.CENTER)); titlePanel.add(new JLabel("Nom", SwingConstants.CENTER));

titlePanel.add(new JLabel("Prenom", SwingConstants.CENTER)); titlePanel.add(new JLabel("Email", SwingConstants.CENTER)); titlePanel.add(new JLabel("Salaire", SwingConstants.CENTER)); titlePanel.setBorder(new LineBorder(Color.BLACK)); add(titlePanel, BorderLayout.NORTH);

contentPanel = new JPanel();

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

|  |  |  |
| --- | --- | --- |
| 32 |  | contentPanel.setLayout(new BoxLayout(contentPanel, BoxLayout.Y\_AXIS)); |
| 33 |  | JScrollPane scrollPane = new JScrollPane(contentPanel); |
| 34 |  | add(scrollPane, BorderLayout.CENTER); |
| 35 | } |  |
| 36 |  |  |

37 public void updateEmployerList(List<Employer> employers) {

38 contentPanel.removeAll();

39 rowPanels.clear();

40

41 for (Employer employer : employers) {

42 JPanel rowPanel = new JPanel(new GridLayout(1, 5));

43 rowPanel.setBorder(new LineBorder(Color.GRAY));

44 rowPanel.setMaximumSize(new Dimension(Integer.MAX\_VALUE, 30));

45

46 JLabel idLabel = new JLabel(String.valueOf(employer.getId()), SwingConstants.CENTER);

47 JLabel lastNameLabel = new JLabel(employer.getLastName(), SwingConstants.CENTER);

48 JLabel firstNameLabel = new JLabel(employer.getFirstName(), SwingConstants.CENTER);

49 JLabel emailLabel = new JLabel(employer.getEmail(), SwingConstants. CENTER);

50 JLabel salaryLabel = new JLabel(String.valueOf(employer.getSalary()), SwingConstants.CENTER);

51

52 rowPanel.addMouseListener(new MouseAdapter() {

53 @Override

54 public void mouseClicked(MouseEvent e) {

55 highlightRow(rowPanel, employer.getId());

56 }

57 });

58

59 rowPanel.add(idLabel);

60 rowPanel.add(lastNameLabel);

61 rowPanel.add(firstNameLabel);

62 rowPanel.add(emailLabel);

63 rowPanel.add(salaryLabel);

64

65 contentPanel.add(rowPanel);

66 rowPanels.add(rowPanel);

67 }

68

69 contentPanel.revalidate();

70 contentPanel.repaint();

71 }

72

73 private void highlightRow(JPanel selectedRow, int employerId) {

74 for (JPanel row : rowPanels) {

75 row.setBackground(Color.WHITE);

76 }

77

78 selectedRow.setBackground(Color.LIGHT\_GRAAY);

79 selectedRowId = employerId;

80 }

81

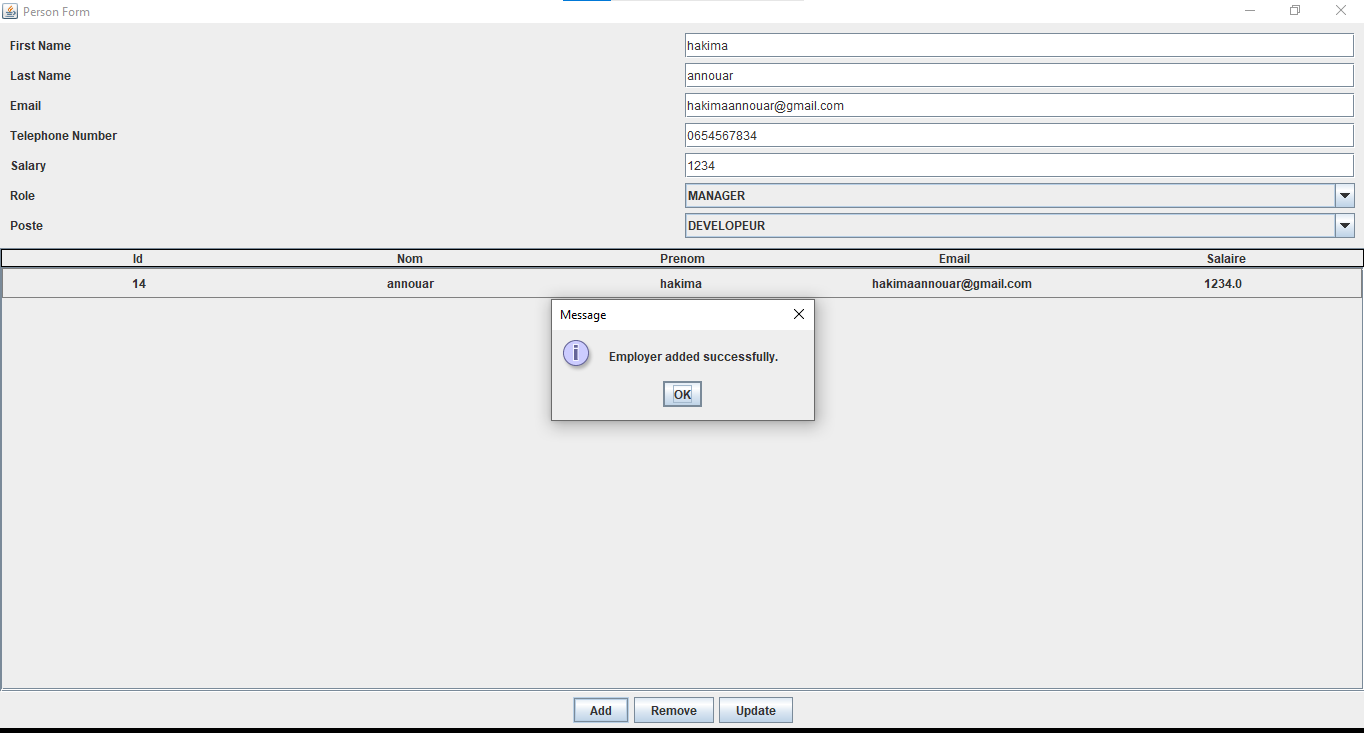
|  |  |  |
| --- | --- | --- |
| 82  83  84 | | public int getSelectedRowId() { return selectedRowId;  } |
|  | 85 | } |
| **2.5** |  | **Main** |
|  | 1 | import view.\*; |
|  | 2 | import dao.\*; |
|  | 3 | import model.\*; |
|  | 4 | import controller.EmployerController; |
|  | 5 | // Déclaration de la classe Main, le point d'entrée de l'application. |
|  | 6 | public class Main {  // La méthode main est le point de démarrage de l'application |
|  | 7 | public static void main(String[] args) {  // Création et initialisation du contrôleur principal. // Ce contrôleur relie la vue (FormFrame) à la logique métier (EmployerLogic) // et à la couche d'accès aux données (EmployerDAO). |
|  | 8 | new EmployerController( new FormFrame(), new EmployerLogic(new EmployerDAO |
|  |  | ())); |
|  | 9 | } |
|  | 10 | } |

**Résultats**

## Ajout

**•**

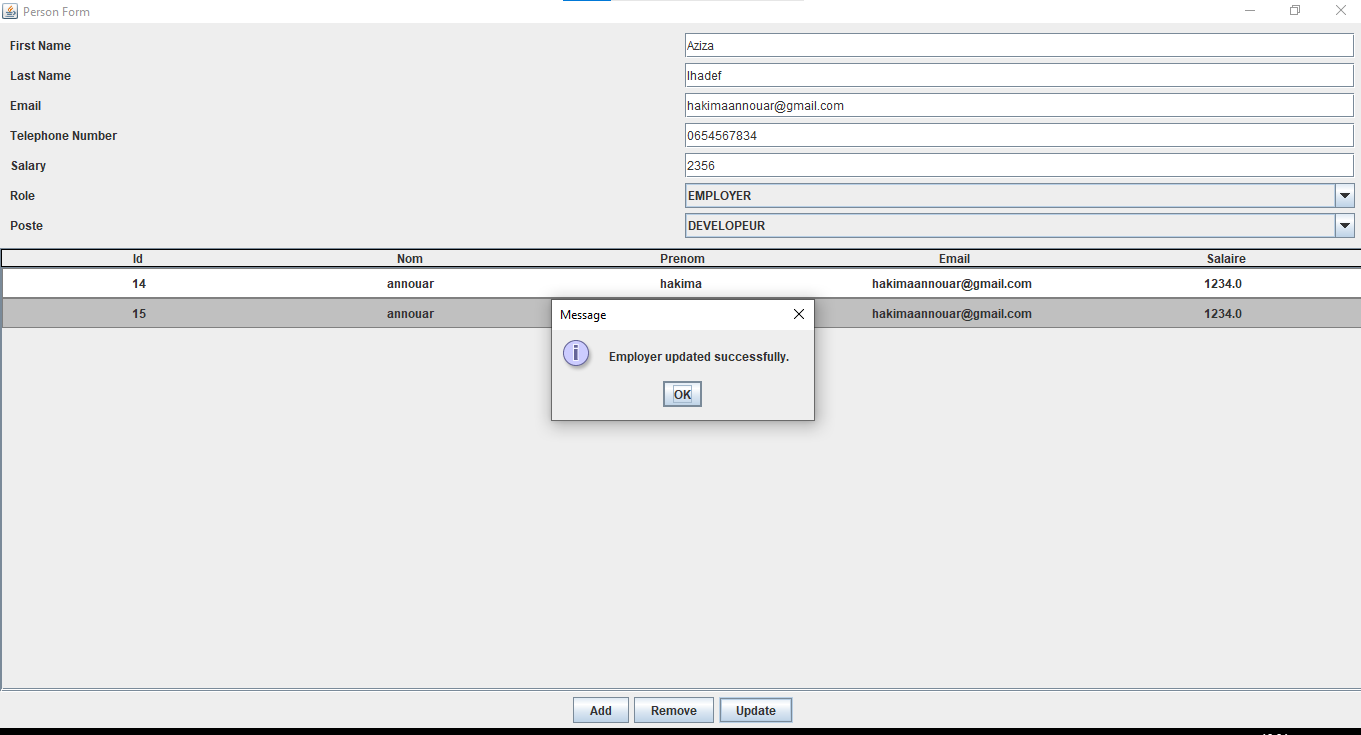
##### Figure 2.1 – Interface Utilisateur



##### Figure 2.2 – Affichage de l’Ajout

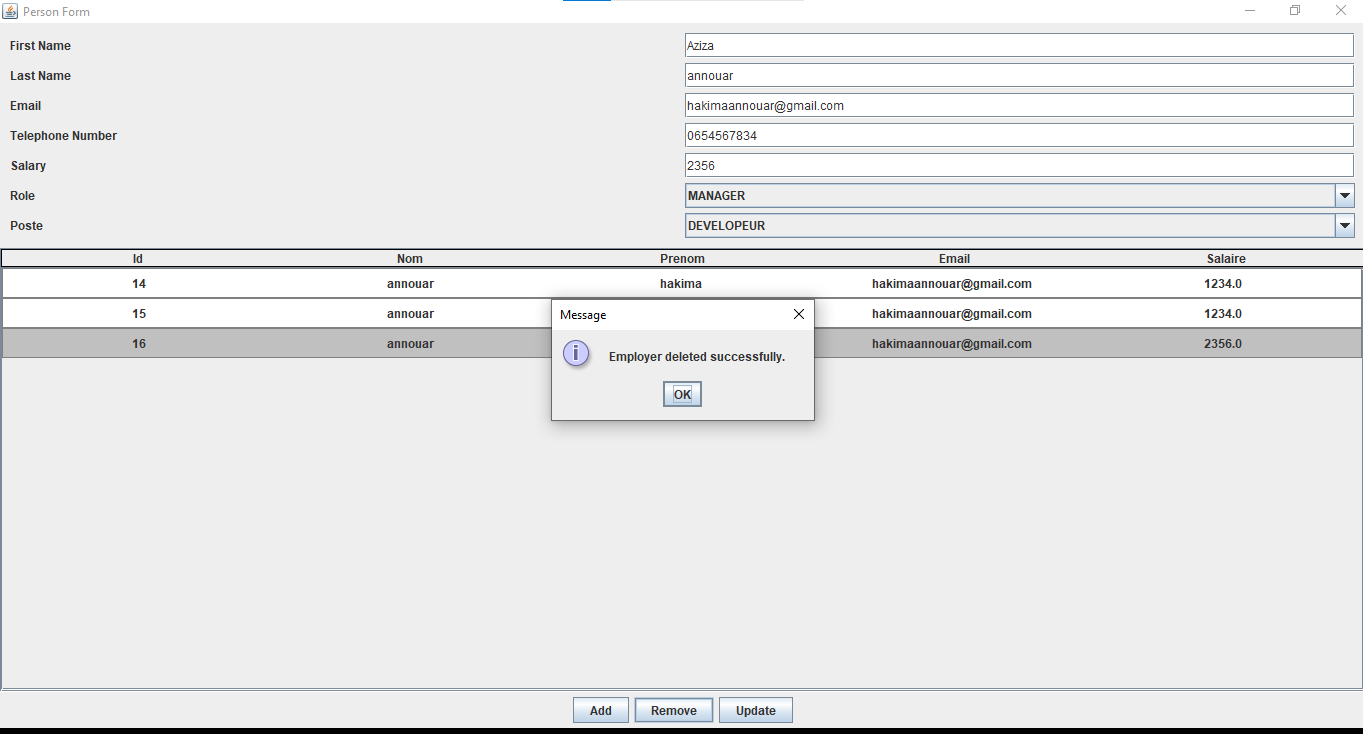
##### SI

## Modification



## Suppression

##### Figure 2.3 – Affichage de modification



##### Figure 2.4 – Affichage de suppression

**Dans cette séance de travaux pratiques, nous avons travaillé sur la gestion des employés d'une entreprise à travers une application Java complète composée de plusieurs couches : logique métier, DAO, contrôleur, et interface utilisateur. Voici les tâches principales effectuées :**

1. **Avec mon coéquipier :**
   * Nous avons configuré le projet Java et défini les classes principales.
   * Nous avons codé les méthodes de gestion des employés dans la couche DAO, notamment pour l'ajout, la mise à jour, et la suppression.
   * Nous avons testé et corrigé les requêtes SQL utilisées dans la gestion des données.
   * Nous avons intégré les interactions entre la couche logique et le contrôleur.
2. **Individuellement :**
   * J'ai travaillé sur l'interface graphique pour créer des panneaux interactifs permettant d'entrer et d'afficher les informations des employés.
   * J'ai implémenté des validations pour les champs de saisie dans l'interface.
   * J'ai testé l'application en simulant différentes actions : ajout, modification, suppression et consultation des employés.
3. **Tests et validation :**
   * Nous avons vérifié la cohérence des données entre l'interface et la base.
   * Nous avons identifié des bugs (comme des erreurs de format dans les champs) et proposé des corrections.

**Cette séance a permis de renforcer nos compétences en programmation Java tout en appliquant des concepts de programmation en couches et d'interactions avec une base de données.**