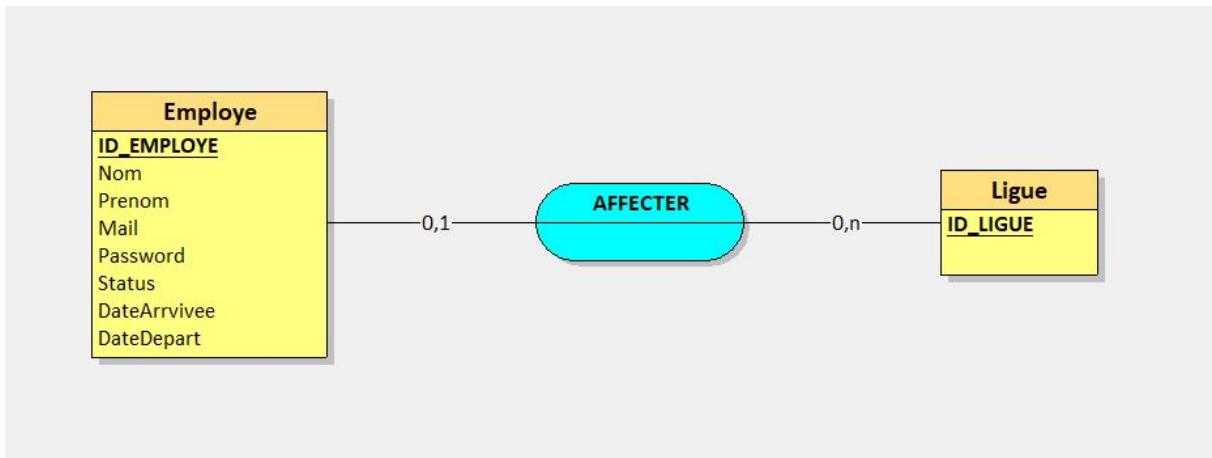


Ce projet a consisté à améliorer une application Java en y ajoutant de nouvelles fonctionnalités, telles que :

- La création d'une base de données pour lier l'application à une base de données (MCD, MLD, MPD).
 - La gestion de l'administrateur via une interface en ligne de commande.
- La gestion des dates avec prise en charge des exceptions associées.
 - L'ajout de tests unitaires pour garantir le bon fonctionnement des méthodes.
 - La gestion des employés (modifications et sélections).

1/a/ Création de la base de données : Le MCD



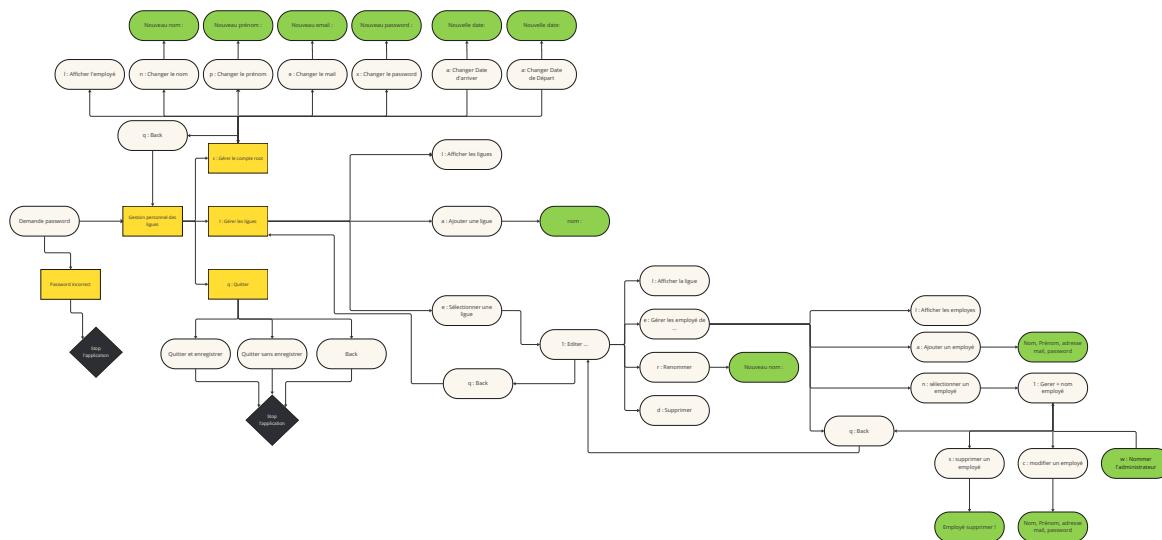
1/b/ Création de la base de données : Le script de création du MCD

```

1   CREATE TABLE Ligue(
2       id_ligue INT,
3       PRIMARY KEY(id_ligue)
4   );
5
6   CREATE TABLE Employe(
7       id_employe INT,
8       nom VARCHAR(30),
9       prenom VARCHAR(30),
10      mail VARCHAR(50),
11      password VARCHAR(30),
12      status VARCHAR(20),
13      dateArrivee DATE,
14      dateDepart DATE,
15      id_ligue INT,
16      PRIMARY KEY(id_employe),
17      FOREIGN KEY(id_ligue) REFERENCES Ligue(id_ligue)
18  );
  
```

2/a/ Application Java : Arbre heuristique

(Lien vers l'image :



2/b/ L'interface en ligne de commande :

```

Console X
PersonnelConsole [Java Application] /Library/In
password : toor
Gestion du personnel des ligues
c : Gérer le compte root
l : Gérer les ligues
q : Quitter

Select an option :

```

2/c/ Les tests unitaires :

Voici les tests ajoutés pour la gestion des dates :

```

68● @Test
69 void testValidDates() throws SauvegardeImpossible , ExceptionDate
70 {
71     Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
72     assertDoesNotThrow(() -> ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty" , "test" , LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 30)));
73 }
74
75● @Test
76 void testInvalidDates() throws SauvegardeImpossible , ExceptionDate
77 {
78     Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
79     Exception exception = assertThrows(ExceptionDate.class, () -> {
80         ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test" , LocalDate.of(2023, 12, 31), LocalDate.of(2023, 1, 1));
81     });
82     assertEquals("La date de départ ne peut pas être avant la date d'arrivée.", exception.getMessage());
83 }
84
85● @Test
86 void testDataArriveNull() throws SauvegardeImpossible , ExceptionDate
87 {
88     Ligue ligue = gestionPersonnel.addLigue("Football");
89     Exception exception1 = assertThrows(ExceptionDate.class, () -> {
90         ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test" , LocalDate.of(2023, 12, 31), LocalDate.of(2023, 1, 1));
91     });
92     assertEquals("La date de départ ne peut pas être avant la date d'arrivée.", exception1.getMessage());
93 }
94● @Test
95 void testDataDepartNull() throws SauvegardeImpossible , ExceptionDate
96 {
97     Ligue ligue = gestionPersonnel.addLigue("Football");
98     Exception exception1 = assertThrows(ExceptionDate.class, () -> {
99         ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test" , LocalDate.of(2023, 12, 31), LocalDate.of(2023, 1, 1));
100    });
101    assertEquals("La date de départ ne peut pas être avant la date d'arrivée.", exception1.getMessage());
102 }
103
104● @Test
105 void setDateArriveNull()throws SauvegardeImpossible, ExceptionDate
106 {
107     Ligue ligue = gestionPersonnel.addLigue("Football");
108     Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test" , LocalDate.of(2023, 12, 31), LocalDate.of(2024, 1, 1));
109     Exception exception1 = assertThrows(ExceptionDate.class, () -> {employe.setDateArrivee(null)};
110 };
111 }
112
113● void setDateDepartNull()throws SauvegardeImpossible, ExceptionDate
114 {
115     Ligue ligue = gestionPersonnel.addLigue("Football");
116     Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test" , LocalDate.of(2023, 12, 31), LocalDate.of(2023, 1, 1));
117     Exception exception1 = assertThrows(ExceptionDate.class, () -> {employe.setDateDepart(null)};
118 };
119 }
120
121● @Test
122 void testSetDateDepartInvalid() throws SauvegardeImpossible , ExceptionDate
123 {
124     Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
125     Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty" , "test" , LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
126     Exception exception = assertThrows(ExceptionDate.class, () -> employe.setDateDepart(LocalDate.of(2022, 1, 1)));
127     assertEquals("La date de départ ne peut pas être avant la date d'arrivée." , exception.getMessage());
128 }
129
130● @Test
131 void testSetDateArriveInvalid() throws SauvegardeImpossible , ExceptionDate
132 {
133     Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
134     Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test" , LocalDate.of(2023, 12, 31), LocalDate.of(2024, 1, 1));
135     Exception erreur = assertThrows(ExceptionDate.class, () -> employe.setDateArrivee(LocalDate.of(2024, 12, 2)));
136     assertEquals("La date de départ ne peut pas être avant la date d'arrivée." , erreur.getMessage());
137 }
138
139● @Test
140 void testGetDate() throws SauvegardeImpossible , ExceptionDate
141 {
142     Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
143     Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test" , LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
144     assertEquals( employe.getDateArrivee(), LocalDate.of(2023, 1, 1));
145     assertEquals( employe.getDateDepart() , LocalDate.of(2023, 12, 31));
146 }
147
148 }
```

Tests unitaires ajoutés pour tester la suppression des ligues et des employés et l'ajout d'employés

```

77● @Test
78     void deleteLigue() throws SauvegardeImpossible {
79         Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
80         ligue.remove();
81         assertFalse(gestionPersonnel.getLigues().contains(ligue));
82     }
83
84
85
86 }

```

```

22● @Test
23     void addEmploye() throws SauvegardeImpossible, ExceptionDate {
24         Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
25         Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test", LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 30));
26         assertEquals(employe, ligue.getEmployes().first());
27     }
28
29
30● @Test
31     void setEmploye() throws SauvegardeImpossible, ExceptionDate {
32         Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
33         Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test", LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 30));
34         employe.setNom("test");
35         assertEquals("test", employe.getNom());
36     }
37

```

Test unitaires ajoutés pour tester les guetteurs et setteurs des employés

```

37
38● @Test
39     void getNom() throws SauvegardeImpossible {
40         Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
41         assertEquals("Fléchettes", ligue.getNom());
42     }
43
44● @Test
45     void setNom() throws SauvegardeImpossible {
46         Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
47         ligue.setNom("Bowling");
48         assertEquals("Bowling", ligue.getNom());
49     }
50

```

Tests unitaires ajoutés pour tester le changement d'administrateur

```
51    @Test
52    void getAdministrator() throws SauvegardeImpossible, ExceptionDate {
53        Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
54        Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test", LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 30));
55        ligue.setAdministrateur(employe);
56        assertEquals(employe, ligue.getAdministrateur());
57    }
58}
59    @Test
60    void deleteAndChangeAdmin() throws SauvegardeImpossible, ExceptionDate{
61        Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
62        Employe employe;
63
64        employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", "test", LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 30));
65
66        ligue.setAdministrateur(employe);
67        assertEquals(employe, ligue.getAdministrateur());
68
69        employe.remove();
70        assertFalse(ligue.getEmployees().contains(employe));
71
72        assertFalse(ligue.getEmployees().contains(employe));
73        assertEquals(gestionPersonnel.getRoot(), ligue.getAdministrateur());
74    }
75}
```

3/a/ Modification en ligne de commande :

Changer l'administrateur

```
84●    public void setAdministrateur(Employe administrateur)
85    {
86        Employe root = gestionPersonnel.getRoot();
87        if (administrateur != root && administrateur.getLigue() != this)
88            throw new DroitsInsuffisants();
89        this.administrateur = administrateur;
90    }
91
92●    private Option changerAdmin(final Employe employe) {
93    return new Option("Nommer l'administrateur", "w", () -> employe.getLigue().setAdministrateur(employe));
94}
95
96
97●    private Menu editerEmployer(Employe employe) {
98    Menu menu = new Menu("Gerer :" + employe.getNom());
99    menu.add(modifierEmploye(employe));
100   menu.add(supprimerEmploye(employe));
101   menu.add(changerAdmin(employe));
102
103   menu.addBack("q");
104   return menu;
105}
106
```

Sélectionner un employé avant de le modifier ou de le supprimer

```
157  
158     private Option selectEmploye(Ligue ligue)  
159     {  
160         return new List<>("Selectionner un employe", "n",  
161                             () -> new ArrayList<>(ligue.getEmployes()),  
162                             (nb) -> editerEmployer(nb));}  
163     ;  
164 }  
165
```

```

127
128     private Menu editerEmploye(Employe employe) {
129         Menu menu = new Menu("Gérer : " + employe.getNom());
130         menu.add(modifierEmploye(employe));
131         menu.add(supprimerEmploye(employe));
132         menu.add(changerAdmin(employe));
133
134         menu.addBack("q");
135         return menu;
136     }
137

```

Dates d'arrivées et de départ

```

21     Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, String status, LocalDate dateArrivee, LocalDate dateDepart)
22     throws ExceptionDate
23     {
24         this.gestionPersonnel = gestionPersonnel;
25         this.nom = nom;
26         this.prenom = prenom;
27         this.mail = mail;
28         this.ligue = ligue;
29         this.password = password;
30         this.setStatus(status);
31         if (dateDepart.isBefore(dateArrivee)) throw new ExceptionDate();
32         this.dateArrivee = dateArrivee;
33         this.dateDepart = dateDepart;
34     }
35

```

```

25
26     Option editerEmploye(Employe employe)
27     {
28         Menu menu = new Menu("Gérer le compte " + employe.getNom(), "c");
29         menu.add(afficher(employe));
30         menu.add(changerNom(employe));
31         menu.add(changerPrenom(employe));
32         menu.add(changerMail(employe));
33         menu.add(changerPassword(employe));
34         menu.add(modifierDateArrivee(employe));
35         menu.add(modifierDateDepart(employe));
36         menu.addBack("q");
37         return menu;
38     }
39
40     private Option modifierDateArrivee(Employe employe) {
41         return new Option("Changer date d'arriver", "a",
42             () ->
43                 {
44                     try {
45                         employe.setDateArrivee(LocalDate.parse(getString("Nouvelle date")));
46                     } catch (ExceptionDate e) {
47                         System.out.println("Les dates ne sont pas cohérentes : la date de départ ne peut pas être antérieure à la date d'arrivée.");
48                     }
49                     catch (DateTimeParseException s) {
50                         System.out.println("Merci de fournir la date dans le format suivant : AAAA-MM-JJ.");
51                     }
52                 });
53     }
54     private Option modifierDateDepart(Employe employe) {
55         return new Option("Changer date Depart", "d",
56             () ->
57                 {
58                     try {
59                         employe.setDateDepart(LocalDate.parse(getString("Nouvelle date")));
60                     } catch (ExceptionDate e) {
61                         System.out.println("Les dates ne sont pas cohérentes : la date de départ ne peut pas être antérieure à la date d'arrivée.");
62                     }
63                     catch (DateTimeParseException s) {
64                         System.out.println("Merci de fournir la date dans le format suivant : AAAA-MM-JJ.");
65                     }
66                 });
67

```

3/b/ Exceptions dates

Création exception :

```

1 package personnel;
2
3 public class ExceptionDate extends Exception {
4     public String getMessage(){
5         return "La date de départ ne peut pas être avant la date d'arrivée.";
6     }
7 }
8

```

Dans le constructeur :

```

22     Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, String status, LocalDate dateArrivee, LocalDate dateDepart)
23     throws ExceptionDate
24     {
25         this.gestionPersonnel = gestionPersonnel;
26         this.nom = nom;
27         this.prenom = prenom;
28         this.mail = mail;
29         this.ligue = ligue;
30         this.password = password;
31         this.setStatus(status);
32         if (dateDepart.isBefore(dateArrivee)) throw new ExceptionDate();
33         this.dateArrived = dateArrivee;
34         this.dateDepart = dateDepart;
35     }

```

Dans les setteurs :

```

198     public void setDateArrivee(LocalDate dateArrivee)
199     throws ExceptionDate
200     {
201         if (dateArrivee == null || dateDepart.isBefore(dateArrivee)) {
202             throw new ExceptionDate();
203         }
204         else {
205             this.dateArrivee = dateArrivee;}
206     }
207
208
209     public void setDateDepart(LocalDate dateDepart)
210     throws ExceptionDate
211     {
212         if (dateDepart == null || dateDepart.isBefore(dateArrivee)) {
213             throw new ExceptionDate();
214         }
215         else {
216             this.dateDepart = dateDepart;}
217     }
218
219
220

```