DOCUMENTATION PERSONNEL :

Gestion du personnel des ligues*:*

Ce projet a été réalisé en groupe de 3 personnes lors de ma 2ème année de BTS SIO durant lequel nous avons dû modifié et programmé en Java.

Un des responsables de la M2L, utilise une application pour gérer les employés des ligues. Cette application, très simple, n'existe qu'en ligne de commande et est mono-utilisateur. Nous souhaiterions désigner un administrateur par ligue et lui confier la tâche de recenser les employés de sa ligue.

Le besoin attendu :

Les niveaux d'habilitation des utilisateurs sont les suivants :

- Un simple employé de ligue peut ouvrir l'application et s'en servir comme un annuaire, mais il ne dispose d'aucun droit d'écriture.
- Un employé par ligue est administrateur et dispose de droits d'écriture peut gérer la liste des employés de sa propre ligue avec une application bureau.
- Le super-administrateur a accès en écriture à tous les employés des ligues. Il peut aussi gérer les comptes des administrateurs des ligues avec une application accessible en ligne de commande.

- L'application doit être rendue multi-utilisateurs grace à l'utilisation d'une base de données.
- Les trois niveaux d'habilitation ci-dessus doivent être mis en place.

Nous devions :

- Conserver java pour l'application.
- Utiliser le versionnement avec git
- Utiliser la bibliothèque de dialogue en ligne de commande fournie.

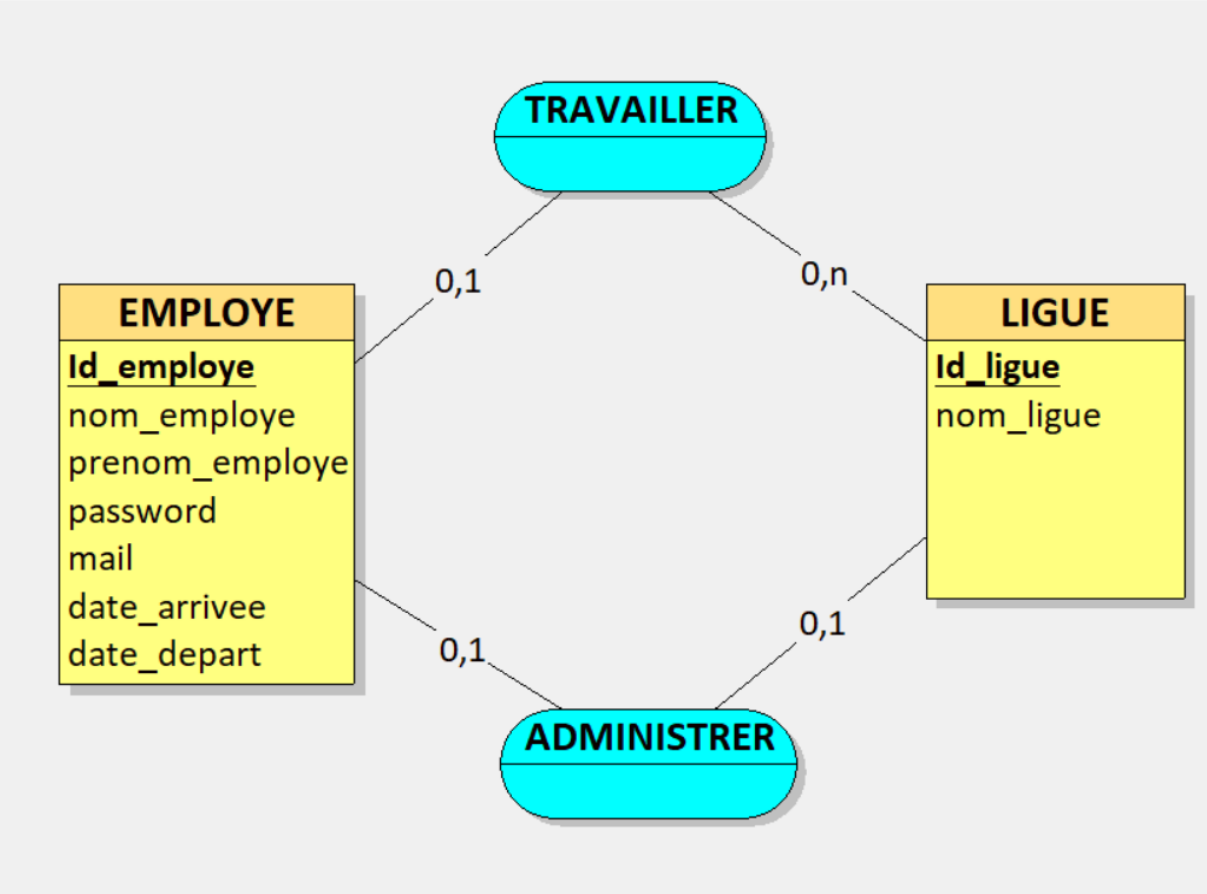
Caractéristiques à mettre en place sur le projet :

- Modélisation d'une base de données avec un MCD.
- Vérification du fonctionnement correct de l'application grâce à des tests unitaires.
- Gestion de la date de départ et de celle d'arrivée de chaque employé (couche métier + tests unitaires).
- Représentation des menus du dialogue en ligne de commande avec un arbre heuristique (Utilisez un logiciel de type Freemind).

- Création de la base de données et production du script de création de tables.
- Gestion des dates dans le dialogue en ligne de commande.
- Dans le dialogue en ligne de commande, un employé doit être selectionné avant que l'on puisse choisir de modifier ou de supprimer.
- Possibilité de changer l'administrateur d'une ligue en ligne de commande.

- Création d'une classe fille de Passerelle permettant de gérer le dialogue avec la base de données avec JDBC (ou avec Hibernate si vous le souhaitez).
- Utilisation de la base de données pour réaliser les opérations d'ajout, de modification, de suppression des ligues et des employés.
- Modélisation de l'interface graphique avec des maquettes.

MCD Réalisé pour cette application :



Tests unitaires :

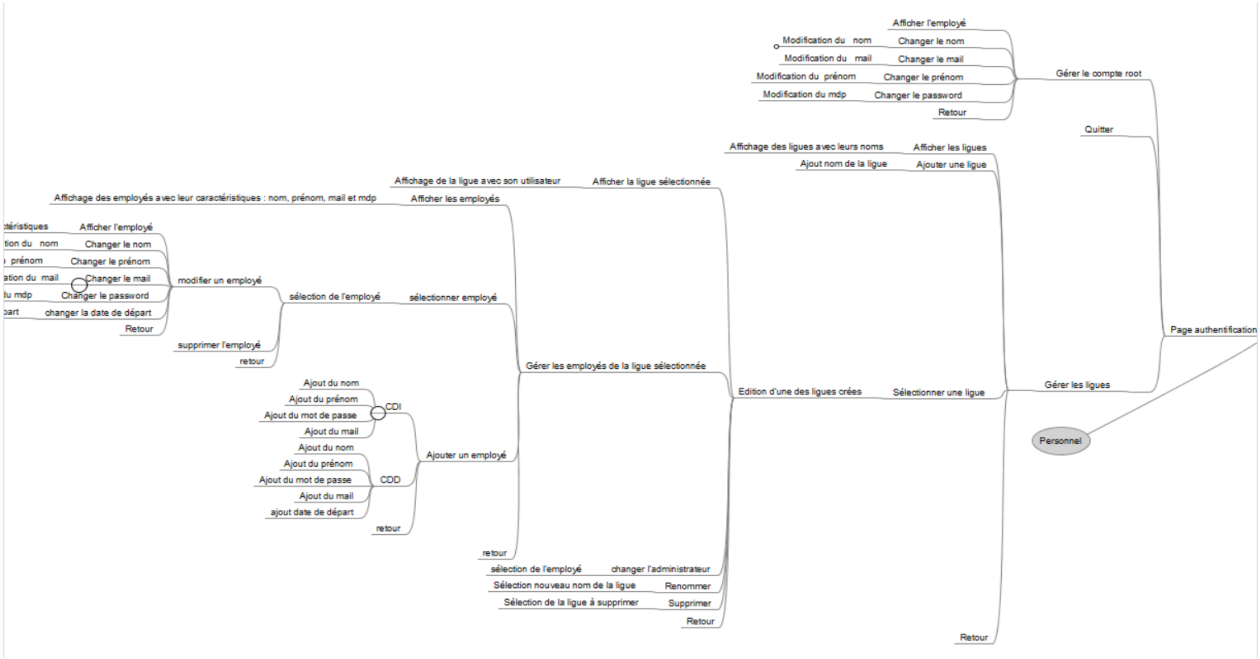
Afin de vérifier le fonctionnement correct de l'application nous avons du effectuer des test unitaires.

```java
void remove(Employe employe)
{
    employes.remove(employe);
}

/**
 * Supprime la ligue, entraîne la suppression de tous les employés
 * de la ligue.
 */

public void remove()
{
    gestionPersonnel.remove(this);
}
```

Exemple de test pour vérifier la suppression d'un employé. :

```java
void removeemploye() throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye ("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty");
    Employe employe2 = ligue.addEmploye ("Michel", "Jean", "j.michel@gmail.com", "azerty");
    employe.remove();
    assertEquals(employe,ligue.getEmployes());
}
@Test
void removeligue() throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye ("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty");
    ligue.remove();
    assertNotEquals(ligue, gestionPersonnel.getLigues());
}
```

Voici l'arbre heuristique représentant les menus du dialogue en ligne de commande :



Ensuite nous avons crée le script de la base de données et production du script de création de tables

Voici le script :

```
DROP DATABASE IF EXISTS personnel;
CREATE DATABASE personnel ;
USE personnel;


DROP TABLE IF EXISTS Employe;
DROP TABLE IF EXISTS Ligue;

create table Employe (
id_employe BIGINT(8) ,
nomEmploye varchar(25),
prenomEmploye varchar(25) ,
password varchar(25) not null,
mail varchar(70) not null,
DateArrivee DATE,
DateDepart DATE,
id_ligue BIGINT,

constraint PK_EMP primary key(id_employe)
)ENGINE=INNODB;
DESC Employe;

create table Ligue(
id_ligue BIGINT(4),
nom varchar(255) not null,
administrateur varchar(50) not null,


constraint PK_LIG primary key(id_ligue)
)ENGINE=INNODB;
DESC Ligue;

ALTER TABLE `Employe` ADD FOREIGN KEY (`id_ligue`) REFERENCES `ligue`
(`id_ligue`);
```

ALTER TABLE `Ligue` ADD FOREIGN KEY (`id_ligue`) REFERENCES `Employe` (`id_employe`);

Puis nous avons géré les dates en ligne de commande avec :

```
//  }
    private Option ajouterEmploye(final Ligue ligue)
    {
        Menu menu = new Menu("ajouter un employe ", "a" );
        menu.add(CDI(ligue));
        menu.add(CDD(ligue));
        menu.addBack("q");
        return menu;
    }

    private List<Employe> changerAdministrateur(final Ligue ligue)
    {
        return new List<>("changer l'administrateur", "a", () -> new ArrayList<>(ligue.getEmployes()),(index,element) -> {ligue.setAdministrateur(element)
    }

    private List<Employe> selectionnerEmploye(final Ligue ligue)
    {
        return new List<>("Sélectionner un employé", "e",
                () -> new ArrayList<>(ligue.getEmployes()),
                employeConsole.editerEmploye()
                );
    }

    private Option supprimer(Ligue ligue)
    {
        return new Option("Supprimer", "d", () -> {ligue.remove();});
    }
    private Option CDI(Ligue ligue)
    {
        return new Option("CDI", "i",
                () ->
                {
                    ligue.addEmploye(getString("nom : "),
                        getString("prenom : "), getString("mail : "),
                        getString("password : "));
                }
        );
    }
    private Option CDD(Ligue ligue)
    {
        return new Option("CDD", "d",
                () ->
```

```
public class Employe implements Serializable, Comparable<Employe>
{
    private static final long serialVersionUID = 4795721718037994734L;
    private String nom, prenom, password, mail;
    private Ligue ligue;
    private GestionPersonnel gestionPersonnel;
    private LocalDate dateArrivee, dateDepart;


    /*Employé en CDI*/
    Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password)
    {
        this.gestionPersonnel = gestionPersonnel;
        this.nom = nom;
        this.prenom = prenom;
        this.password = password;
        this.mail = mail;
        this.ligue = ligue;
        /*Date de départ fixée*/
        this.dateArrivee = LocalDate.now();
        this.dateDepart = null;
    }

    /*Employé en CDD*/
    Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, LocalDate dateDepart)
    {
        this.gestionPersonnel = gestionPersonnel;
        this.nom = nom;
        this.prenom = prenom;
        this.password = password;
        this.mail = mail;
        this.ligue = ligue;
        /*Date de départ fixée*/
        this.dateArrivee = LocalDate.now();
        this.dateDepart = dateDepart;
```

Il y a 2 catégories d'employés CDI et CDD afin d'avoir une date d'arrivée et de fin pour celui en CDD. L'employé n'a pas de date de fin prédéfinie.

Voici les méthodes permettant de gérer les dates d'arrivée et les dates de départ.

```java
public LocalDate getDateArrivee()
{
    return this.dateArrivee;
}
public void setDateDepart(String dateDepart)
{
    this.dateDepart = LocalDate.parse(dateDepart);
}
public LocalDate getDateDepart()
{
     return this.dateDepart;
}
```

Ajout de dateDepart dans le fichier Ligue.java pour actualiser la date de départ d'un employé en CDD

```java
public Employe addEmploye(String nom, String prenom, String mail, String password)
{
    Employe employe = new Employe(this.gestionPersonnel, this, nom, prenom, mail, password);
    employes.add(employe);
    return employe;
}
public Employe addEmploye(String nom, String prenom, String mail, String password, LocalDate dateDepart)
{
    Employe employe = new Employe(this.gestionPersonnel, this, nom, prenom, mail, password, dateDepart);
    employes.add(employe);
    return employe;
}
```

Voici les test unitaires correspondants :

```java
GestionPersonnel gestionPersonnel = GestionPersonnel.getGestionPersonnel();

@Test
void getDateArrivee() throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty");
    assertEquals(LocalDate.now(), employe.getDateArrivee());
}
@Test
void getDateDepart() throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2022, 12, 20));
    assertEquals(LocalDate.of(2022, 12, 20), employe.getDateDepart());
}
@Test
void setDateDepart() throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2022, 12, 20));
    employe.setDateDepart("2016-08-16");
    assertEquals(LocalDate.of(2016, 8, 16), employe.getDateDepart());
}
```

Afin de gérer le permettant de gérer le dialogue avec la base de données avec JDBC :

```java
public class Credentials
{
    private static String driver = "mysql";
    private static String driverClassName = "com.mysql.cj.jdbc.Driver";
    private static String host = "localhost";
    private static String port = "3306";
    private static String database = "personnel";
    private static String user = "root";
    private static String password = "toor";

    static String getUrl()
    {
        return "jdbc:" + driver + "://" + host + ":" + port + "/" + database ;
    }

    static String getDriverClassName()
    {
        return driverClassName;
    }

    static String getUser()
    {
        return user;
    }

    static String getPassword()
    {
        return password;
    }
}
```

Mise en place du fichier Credentials.java afin de relier à la base de données

Mise en place du fichier JBDC.java afin de relier à la base de données

Puis mise en place des insert et update