Projet Application Java

Présentation du projet

L'application développée pour la M2L permet de gérer les employés des ligues via une interface en ligne de commande. Elle fonctionne selon une architecture 3-tiers et est mono- utilisateur dans sa forme actuelle. L'application est mise à disposition avec le code source sur GitHub, une documentation et une bibliothèque logicielle pour la gestion des dialogues en ligne de commande.

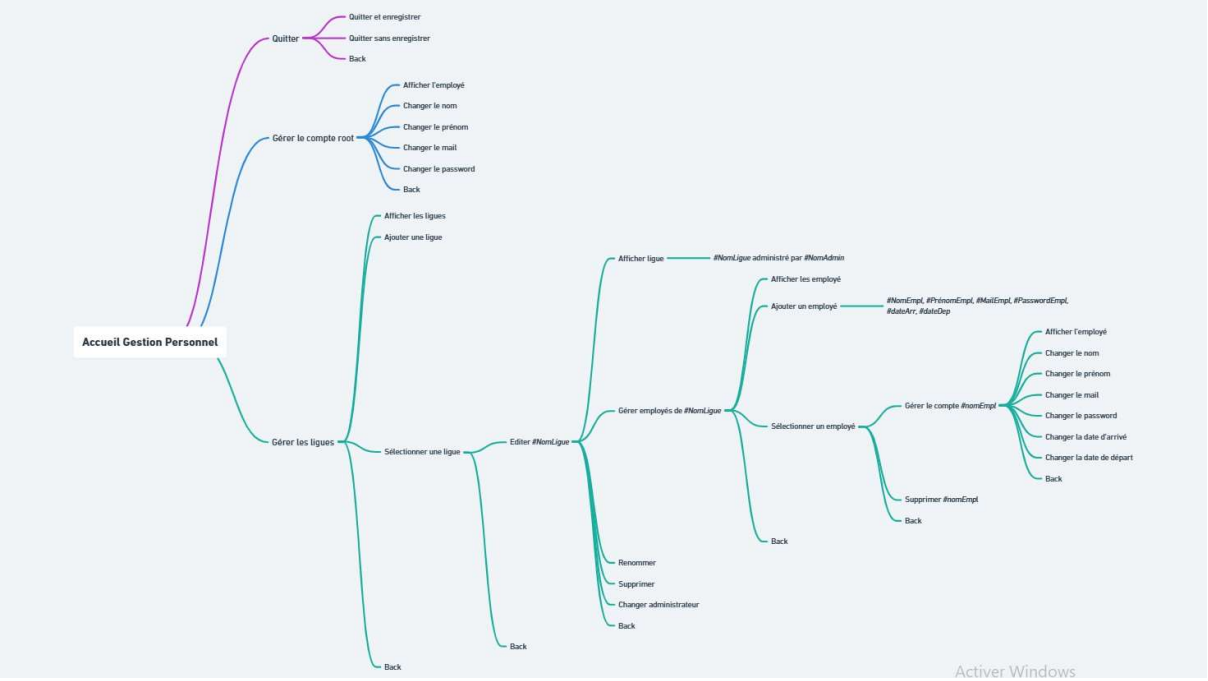
L'objectif est de permettre la gestion multi-utilisateur de l'application en y ajoutant des niveaux d'habilitation et en l'intégrant à une base de données. Les différents rôles d'utilisateurs sont les suivants :

1. **Employé de ligue** : Peut consulter l'annuaire des employés, mais ne dispose d'aucun droit d'écriture.
2. **Administrateur de ligue** : Dispose de droits d'écriture pour gérer les employés de sa propre ligue à l'aide de l'application de bureau.
3. **Super-administrateur** : A un accès en écriture à tous les employés des ligues et peut gérer les comptes des administrateurs des ligues via une interface en ligne de commande.

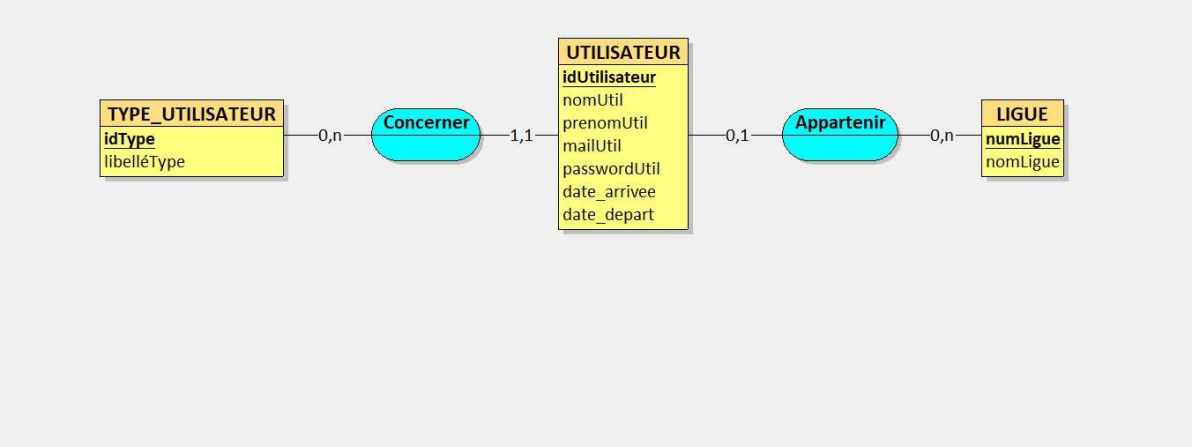
L'application doit être modifiée pour rendre ces rôles fonctionnels et permettre une gestion multi-utilisateur via une base de données.

1ère étape – création d'un arbre heuristique grâce à un outil internet nommé « Whimsical Diagrams »

Cet arbre représente chaque menu de l'application dont 3 principaux : Ligue, Compte root et Quitter



2e étape – création d'un MCD (Modèle Conceptuel de Données) qui regroupe les données de l'application regroupé en tables.



3e étape – script de création des tables du mcd ci-dessous

```
alter table UTILISATEUR
drop foreign key fk_idType;
alter table UTILISATEUR
drop foreign key fk_numLigue;


DROP TABLE IF EXISTS UTILISATEUR;
DROP TABLE IF EXISTS TYPE_UTILISATEUR;
DROP TABLE IF EXISTS LIGUE;

CREATE TABLE UTILISATEUR (
    idUtilisateur INT PRIMARY KEY NOT NULL,
    nomUtil VARCHAR(50),
    prenomUtil VARCHAR(50),
    mailUtil VARCHAR(250),
    passwordUtil VARCHAR(50),
    date_arrivee date,
    date_depart date,
    idType INT,
    numLigue INT null
);

CREATE TABLE TYPE_UTILISATEUR (
    idType INT PRIMARY KEY NOT NULL,
    libelleType VARCHAR(50)
);

CREATE TABLE LIGUE (
    numLigue INT PRIMARY KEY NOT NULL,
    nomLigue VARCHAR(50)
);
```

```
alter table UTILISATEUR
add constraint fk_idType
foreign key(idType) references TYPE_UTILISATEUR(idType);

alter table UTILISATEUR
add constraint fk_numLigue
foreign key(numLigue) references LIGUE(numLigue);
```

4e étape – Modifications apportées sur le code source de certaines couches de l'application

```java
package personnel;

public class DateIncoherente extends Exception {
        /**
         *
         */
        private static final long serialVersionUID = 1L;

        public String DateIncoherente() {
                return "";
        }

}
```

5e étape –Constructeur de la classe Employe avec gestion des dates et validation des incohérences

```java
public Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, LocalDate dateArrivee, LocalDate dateDepart)
throws DateIncoherente
{
        this.gestionPersonnel = gestionPersonnel;
        this.nom = nom;
        this.prenom = prenom;
        this.password = password;
        this.mail = mail;
        this.ligue = ligue;
        if (dateDepart.isBefore(dateArrivee) || dateArrivee == null || dateDepart == null){
                throw new DateIncoherente();
        }
        this.dateArrivee = dateArrivee;
        this.dateDepart = dateDepart;
}
```

6e étape-Tests unitaires pour la gestion des ligues et des employés

```java
package testsUnitaires;

import static org.junit.jupiter.api.Assertions.*;

import java.time.LocalDate;

import org.junit.jupiter.api.Test;

import personnel.*;

class testLigue {
	GestionPersonnel gestionPersonnel = GestionPersonnel.getGestionPersonnel();
//
	@Test
	void createLigue() throws SauvegardeImpossible {
		Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
		assertEquals("Fléchettes", ligue.getNom());
	}

	@Test
	void addEmploye() throws SauvegardeImpossible, DateIncoherente {
		Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
		Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty",
				LocalDate.of(2023, 12, 01), LocalDate.of(2024, 12, 01));
		assertEquals(employe, ligue.getEmployes().first());
	}
```

```java
@Test // test lorsqu'on supprimer un employé (ne devrait plus être présent après le
		// remove() donc assertFalse)
void removeEmploye() throws SauvegardeImpossible, DateIncoherente {
	Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
	Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty",
			LocalDate.of(2023, 12, 01), LocalDate.of(2024, 12, 01));

	assertTrue(ligue.getEmployes().contains(employe));

	ligue.remove(employe);

	assertFalse(ligue.getEmployes().contains(employe));
}

@Test // test pour supprimer une ligue
void removeLigue() throws SauvegardeImpossible, DateIncoherente {
	Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
	Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2023, 12, 01), LocalDate.of(2024, 12, 01));

	assertTrue(ligue.getEmployes().contains(employe));

	ligue.remove();

	assertFalse(gestionPersonnel.getLigues().contains(ligue));
```

7e étape-Gestion des employés et des administrateurs au sein d'une ligue via des menus interactifs

```
private Menu gererEmployes(Ligue ligue) {
        Menu menu = new Menu("Gérer les employés de " + ligue.getNom(), "e");
        menu.add(afficherEmployes(ligue));
        menu.add(ajouterEmploye(ligue));
        menu.add(selectionnerEmploye(ligue));
        menu.addBack("q");
        return menu;
}

// méthode permettant de choisir l'employé pour ensuite le gérer
private List<Employe> selectionnerEmploye(final Ligue ligue) {
        return new List<>("Sélectionner un employé", "s",
                        () -> new ArrayList<>(ligue.getEmployes()),
                        this::menuEmploye);
}

//menu employé pour intégrer la suppression et l'édition de l'employé
private Menu menuEmploye(Employe employe) {
        Menu menu = new Menu("Gérer " + employe.getNom() + " " + employe.getPrenom(), "g");
        menu.add(employeConsole.editerEmploye(employe));
        menu.add(supprimerEmploye(employe));
        menu.addBack("q");
        return menu;
}
```

```
// option changer d'admin
private List<Employe> changerAdministrateur(final Ligue ligue) {
        return new List<>("Changer d'administrateur", "c", () -> new ArrayList<>(ligue.getEmployes()),
                        (index, element) -> {ligue.setAdministrateur(element); system.out.println(element + " est le nouvel administrateur");});}


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

private Option supprimer(Ligue ligue) {
        return new Option("Supprimer", "d", () -> {
                ligue.remove();
        });
}
```

8e étape – Connexion à la base de données

L'application a été conçu pour interagir avec la base de données créée précédemment. Cette base de données (ici nommée « ligue ») devra être accessible par un utilisateur spécifique créé à partir de MySQL. Pour cela, une copie de Credentials.java a été ajouté au projet qui spécifie l'environnement de base de données utilisé sur la machine locale, elle répertorie en plus les détails de l'utilisateur de la base de données. En voici un exemple :

```java
package jdbc;

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 = "ligue";
    private static String user = "tojo";
    private static String password = "123456";

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

    static String getDriverClassName()
    {
        return driverClassName;
    }

    static String getUser()
    {
        return user;
    }

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

9e étape – Insertion des éléments saisis en ligne de commande de l'application dans la base de données et (inversement) lecture à partir de la base de données

Plusieurs actions peuvent être effectuées depuis le menu en ligne de commande de l'application (ajouter une ligue, supprimer une ligue, ajouter un employé, modifier un employé, etc…).

Exemple d'étapes pour insérer un employé ajouté en ligne de commande dans la base de données :

- Constructeur employé dans la classe Employe

```java
Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, LocalDate dateArrivee, Lo
    this(gestionPersonnel, ligue, -1, nom, prenom, mail, password, dateArrivee, dateDepart);
    this.id = gestionPersonnel.insert(this);
}
```

- Ajout de la méthode insert(Employe employe) dans la passerelle

```java
package personnel;

public interface Passerelle
{
    public GestionPersonnel getGestionPersonnel();
    public void sauvegarderGestionPersonnel(GestionPersonnel gestionPersonnel)  throws SauvegardeImpossible;
    public int insert(Ligue ligue) throws SauvegardeImpossible;
    public int insert(Employe employe) throws SauvegardeImpossible;
    public void update(Ligue ligue) throws SauvegardeImpossible;
    public void update(Employe employe) throws SauvegardeImpossible;
    public void delete(Employe employe) throws SauvegardeImpossible;
    public void delete(Ligue ligue) throws SauvegardeImpossible;
}
```

- Ajout de la méthode insert(Employe employe) dans la classe GestionPersonnel

```java
int insert(Employe employe) throws SauvegardeImpossible {
    return passerelle.insert(employe);
}
```

- Préparation de la requête d'insertion dans la table « utilisateur » depuis JDBC

```java
@Override
public int insert(Employe employe) throws SauvegardeImpossible
{
    try
    {
        PreparedStatement instruction = connection.prepareStatement(
            "INSERT INTO utilisateur (nomUtil, prenomUtil, mailUtil, passwordUtil, date_arrivee, date_depart, numLigue, admin) VALUES (?,
            Statement.RETURN_GENERATED_KEYS);
        instruction.setString(1, employe.getNom());
        instruction.setString(2, employe.getPrenom());
        instruction.setString(3, employe.getMail());
        instruction.setString(4, employe.getPassword());
        instruction.setDate(5, employe.getDateArrivee() != null ? java.sql.Date.valueOf(employe.getDateArrivee()) : null);
        instruction.setDate(6, employe.getDateDepart() != null ? java.sql.Date.valueOf(employe.getDateDepart()) : null);
        instruction.setInt(7, employe.getLigue().getIdLigue());
        instruction.setBoolean(8, employe.estAdmin(employe.getLigue()));
        instruction.executeUpdate();
        ResultSet id = instruction.getGeneratedKeys();
        id.next();
        return id.getInt(1);
    }
    catch (SQLException exception)
    {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}
```

- Compléter le code dans Serialization.java

```java
@Override
public int insert(Employe employe) throws SauvegardeImpossible {
    // TODO Auto-generated method stub
    return 0;
}
```

Exemple du contenu de la table utilisateur :

```
mysql> select * from utilisateur;
+---------------+-----------+------------+-----------------+--------------+--------------+--------------+----------+-------+
| idUtilisateur | nomUtil   | prenomUtil | mailUtil        | passwordUtil | date_arrivee | date_depart  | numLigue | admin |
+---------------+-----------+------------+-----------------+--------------+--------------+--------------+----------+-------+
|             1 | tojo      | tojo       | tojo@example.fr |              | 1999-12-31   | 2000-12-31   |        1 |     0 |
|             4 | michel    |            |                 |              | 1999-12-31   | 2000-12-31   |        1 |     0 |
|             5 | root      | proot      | NULL            | toor         | NULL         | NULL         |     NULL |     1 |
|             6 | tojoAdmin | tojo       | NULL            | NULL         | 2024-01-01   | 2025-01-01   |        1 |     1 |
|             7 | kylian    | mbappé     |                 |              | 2018-12-31   | 2019-12-31   |        3 |     0 |
+---------------+-----------+------------+-----------------+--------------+--------------+--------------+----------+-------+
```