Documentation Projet

Pour ce projet, nous avons récupéré depuis Discord une application Java permettant de gérer l'organisation de ligues et d'employés en ligne de commande.

Actuellement dans l'itération 2, nous avons eu pour objectif d'ajouter les améliorations suivantes :

- Un MCD
- La création des tables SQL
- Un arbre heuristique
- Le changement de l'administrateur en ligne de commande
- La saisie des dates en ligne de commande
- Le portfolio

Pour ce faire, nous avons utilisé GitHub comme environnement de travail ainsi qu'Eclipse, Looping, Canva et l'IA comme outils.

Partie 1 : Base de données

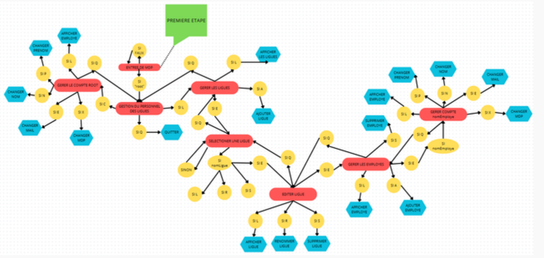
Dans un premier temps, nous avons commencé par imaginer la structure de la base de données, ce qui nous a permis de créer le MCD :

```sql
ALTER TABLE Employe
ADD COLUMN ID_Ligue INT,
ADD COLUMN ID_Role INT;


ALTER TABLE Employe
ADD CONSTRAINT FK_ID_Ligue
FOREIGN KEY (ID_Ligue) REFERENCES Ligue(ID_Ligue);

ALTER TABLE Employe
ADD CONSTRAINT FK_ID_Role
FOREIGN KEY (ID_Role) REFERENCES Role(ID_Role);
```

Une fois cela fait, nous avons travaillé sur les scripts de création des tables et des contraintes :

Partie 2 : Arbre heuristique

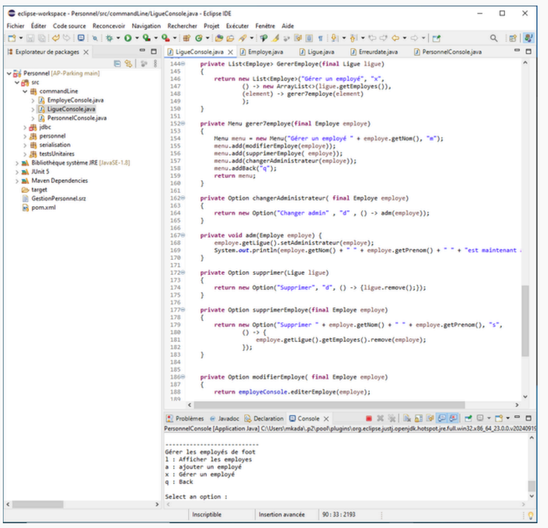
Pour la deuxième partie de la structure de notre projet, nous avons cloné l'application depuis GitHub et l'avons lancée pour nous familiariser avec celle-ci. Par la suite, nous avons réalisé un arbre heuristique retraçant toutes les fonctionnalités de l'application :



Partie 3 : Améliorations Java

Enfin, après avoir établi une structure au projet, nous avons utilisé Eclipse afin d'améliorer l'application.

Nous avons ajouté un sous-menu **Gérer les employés** au menu **Gérer Ligue** d'origine, permettant une meilleure clarté. Nous avons également profité de ce nouveau sous-menu pour inclure une option permettant de changer l'administrateur de la ligue en lien avec un employé donné.



Ensuite, nous avons décidé d'améliorer la gestion des erreurs, notamment concernant les dates d'arrivée et de départ, en intégrant un contrôle rigoureux du format des dates.

```java
private Option ajouterEmploye(final Ligue ligue)
{
    return new Option("ajouter un employé", "a",
            () ->
            {
                // Saisie en chaine - demande toutes les informations d'abord
                String nom = getString("nom : ");
                String prenom = getString("prenom : ");
                String mail = getString("mail : ");
                String password = getString("password : ");
                String dateArrStr = getString("date arrivée (AAAA-MM-JJ) : ");
                String dateDepStr = getString("date départ (AAAA-MM-JJ) : ");

                try {
                    // Exception Format - vérifie le format des dates
                    LocalDate dateArrivee = LocalDate.parse(dateArrStr);
                    LocalDate dateDepart = LocalDate.parse(dateDepStr);

                    // Exception ordre - vérifie l'ordre chronologique
                    if (dateDepart.isBefore(dateArrivee)) {
                        throw new Erreurdate();
                    }

                    // Si tout est valide, crée l'employé
                    ligue.addEmploye(nom, prenom, mail, password, dateArrivee, dateDepart);
                    System.out.println("Employé ajouté avec succès");

                } catch (DateTimeParseException e) {
                    // Gestion de l'exception de format
                    System.out.println("Erreur : Format de date invalide. Utilisez le format AAAA-MM-JJ");
                } catch (Erreurdate e) {
                    // Gestion de l'exception d'ordre chronologique
                    System.out.println(e.getMessage());
                }
            }
    );
}
```