Rapport Projet Génie Logiciel

Pôle projet

27 mai 2022

MASTAIN Vincent

n° étudiant : 22014664

CY Tech - Ingénieur Informatique 1 - Année 2021-2022

Table des matières

1 Introduction **2**

2 Conception **3**

2.1 Modèle conceptuel de données . . . 3

2.2 UML . . . 4

2.2.1 Diagramme de cas d'utilisation . . . 4

2.2.2 Diagramme de classes . . . 5

2.2.3 Diagramme d'activité . . . 7

2.2.4 Diagramme de séquence . . . 8

3 Justification des choix **9**

3.1 Justification des choix de conception UML . . . 9

3.2 Justification des choix de programmation . . . 9

3.2.1 Utilisation des Exceptions . . . 9

3.2.2 Utilisation d'un package Utils . . . 9

3.3 Justification des choix de conception IHM . . . 10

4 Problèmes rencontrés **10**

5 Améliorations possibles **11**

6 Conclusion **11**

1 Introduction

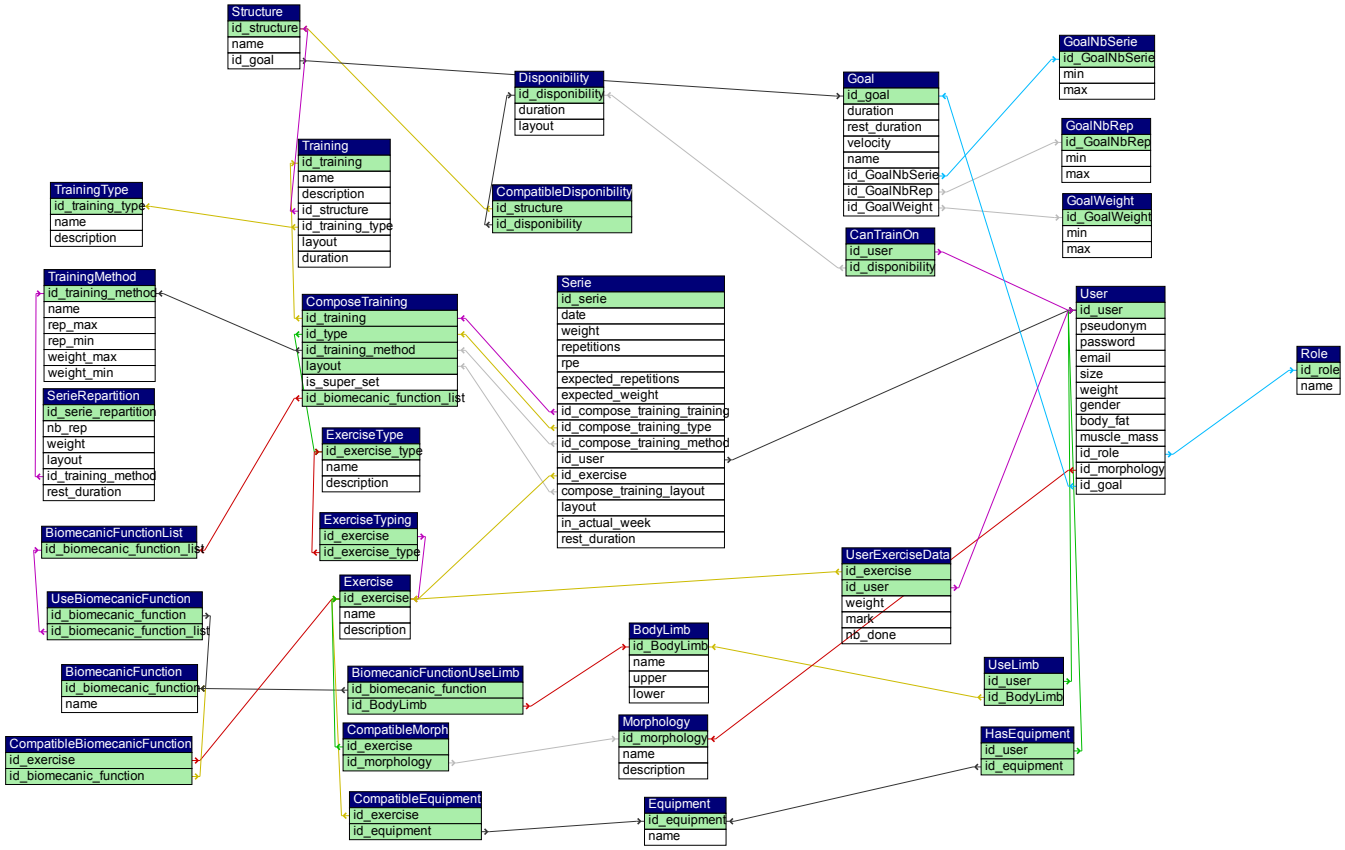
À l'occasion du projet génie logiciel proposé par l'école, j'ai eu la chance de travailler sur mon projet de création d'application de coaching de musculation. Dans le cadre du projet de l'école, l'objectif est de réaliser un concepteur d'entraînement de musculation personnalisé ainsi qu'un adaptateur d'entraînement au fil de l'activité du pratiquant. Créer son programme d'entraînement, faire ses séances de sport, voir son historique d'entraînement, etc.

Pour réaliser ce projet, je me suis déjà penché sur la conception UML, afin d'anticiper l'implémentation des différentes fonctionnalités. Ensuite, j'ai utilisé le langage JAVA, orienté objet, facilitant l'implémentation de la conception UML. Afin de connecter l'application à la base de données j'ai utilisé le package JDBC ainsi que le driver mysql associé.

Je vais d'abord présenter le modèle conceptuel de données, puis la conception UML, l'implémentation en Java et enfin, la justification des choix entrepris et les problèmes rencontrés.

2 Conception

2.1 Modèle conceptuel de données

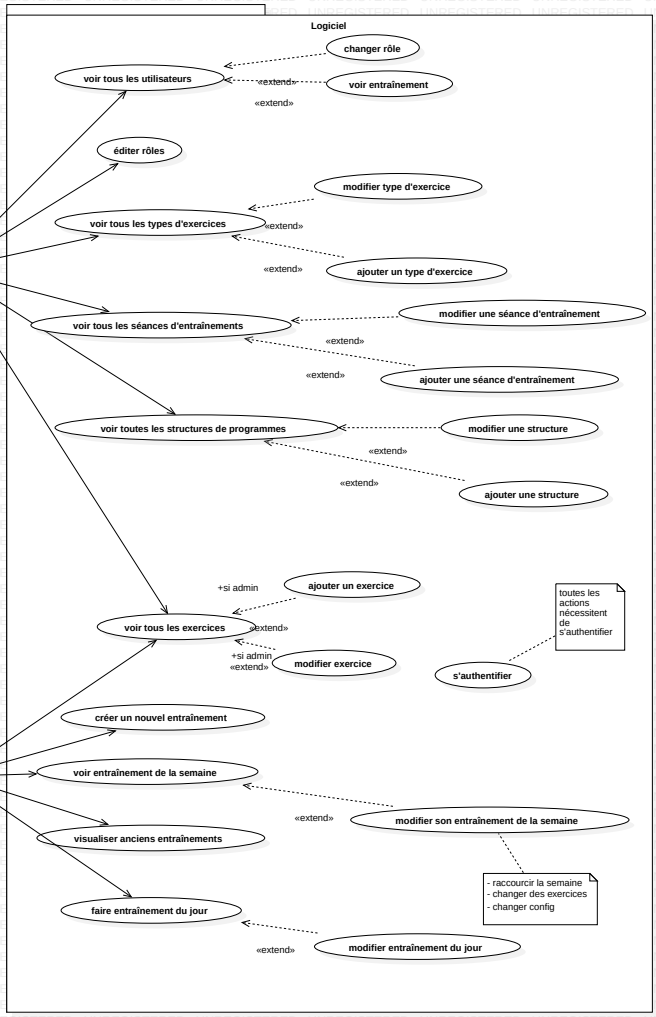


Afin d'anticiper la conception logique du projet, j'ai d'abord pensé à la partie donnée du projet. En effet, cela permet d'avoir une idée de ce qu'il sera nécessaire à faire, quelles classes créer ensuite, quelles méthodes....

Nous observons dans un premier temps, que la série d'exercices est au coeur du mcd. Il permet de faire le lien entre la partie gauche qui concerne l'ensemble de données métiers propres à la préparation physique comme les types d'exercices, d'entraînements, etc. Et la partie droite, qui concerne l'utilisateur et ses attributs, objectif, disponibilités et équipements.

2.2 UML

2.2.1 Diagramme de cas d'utilisation



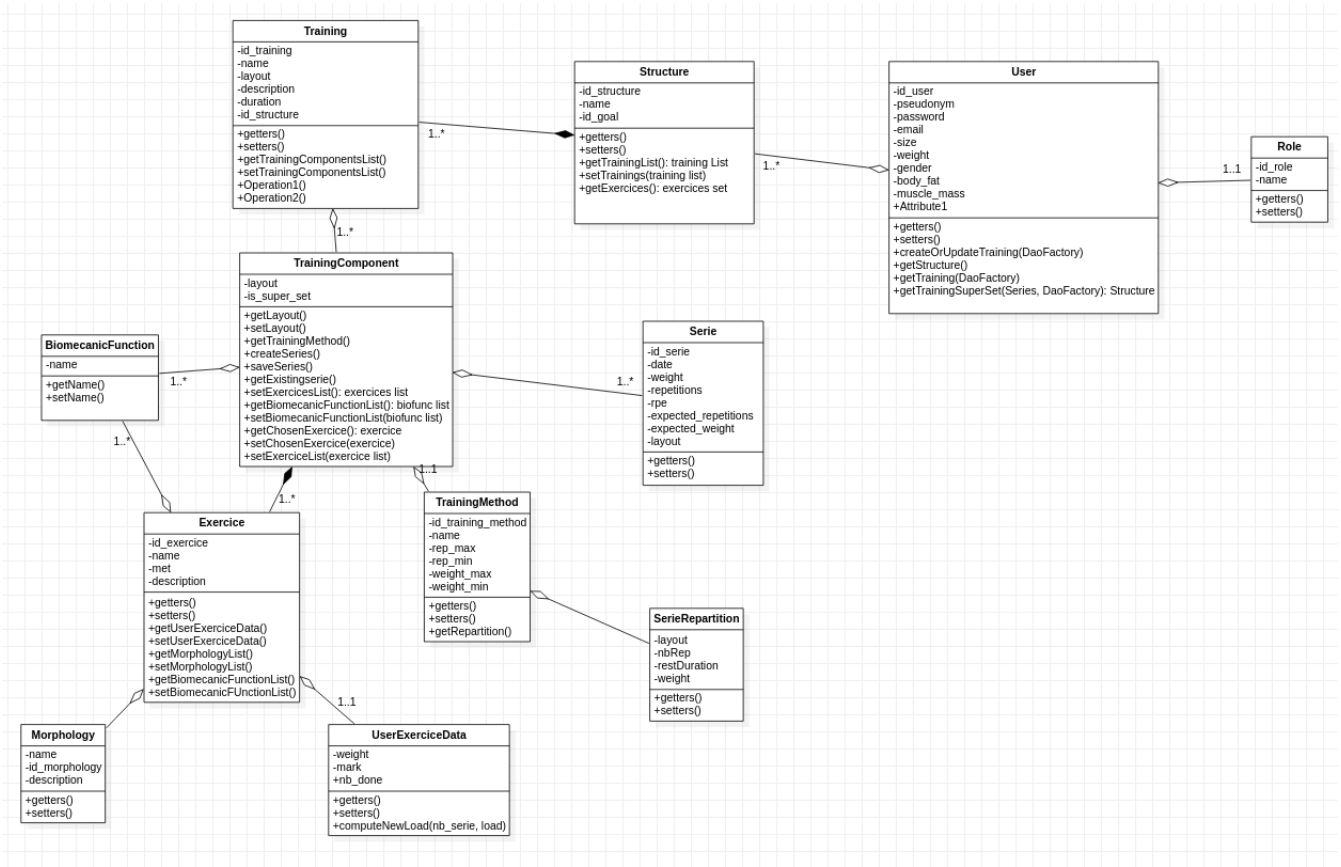
Afin de comprendre les fonctionnalités à mettre en place, j'ai ensuite fait le diagramme de cas d'utilisation

du projet. On aperçoit deux acteurs principaux, l'administrateur et l'utilisateur lambda. L'administrateur ne sera pas disponible dans la version finale du projet par manque de temps. Cependant, toutes les classes et méthodes nécessaires à son utilisation sont disponibles dans modèle logique de données.

L'utilisateur quant à lui peut interagir avec son programme de sport, il peut le créer, visualiser les anciens et modifier l'actuel en indiquant des changements dans ses disponibilités par exemple.

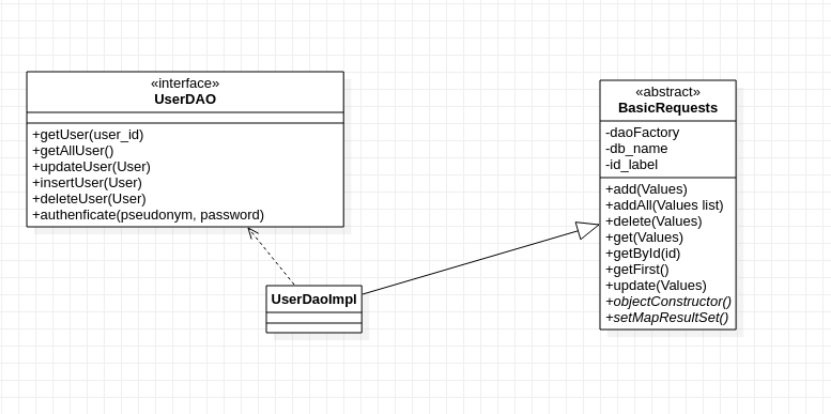
2.2.2 Diagramme de classes

Le diagramme de classes est découpé en deux parties différentes, la première représente la partie logique du projet et l'autre la connexion avec la base de données.



Le diagramme de classes comporte une classe principale user, qui permettra la création d'entraînement.

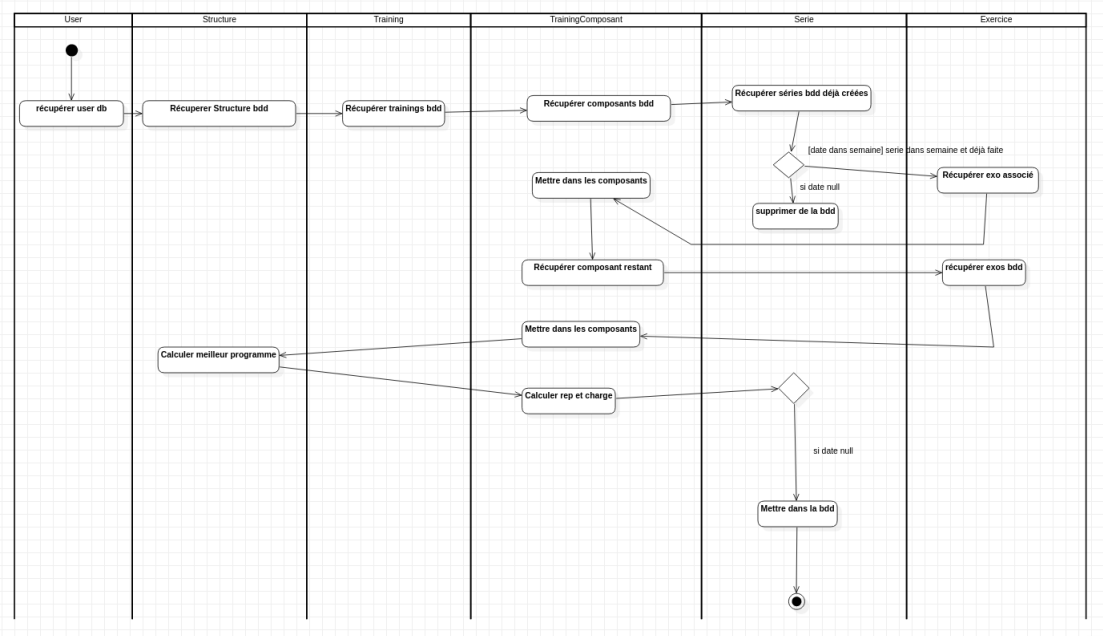
La classe structure représente une semaine d'entraînement (une modification future sera de changer le nom de cette classe pour une meilleure compréhension).
La classe TrainingComponent représente un ensemble d'exercice accompagné d'une manière de s'entraîner.



Chaque objet du modèle dispose d'une interface permettant de faire le lien avec la base de données. Chacune des interfaces dispose d'une implémentation utilisant le driver du SGBD mysql. De plus ces implémentations étendent une classe abstraite permettant de faire les requêtes de base CRUD.

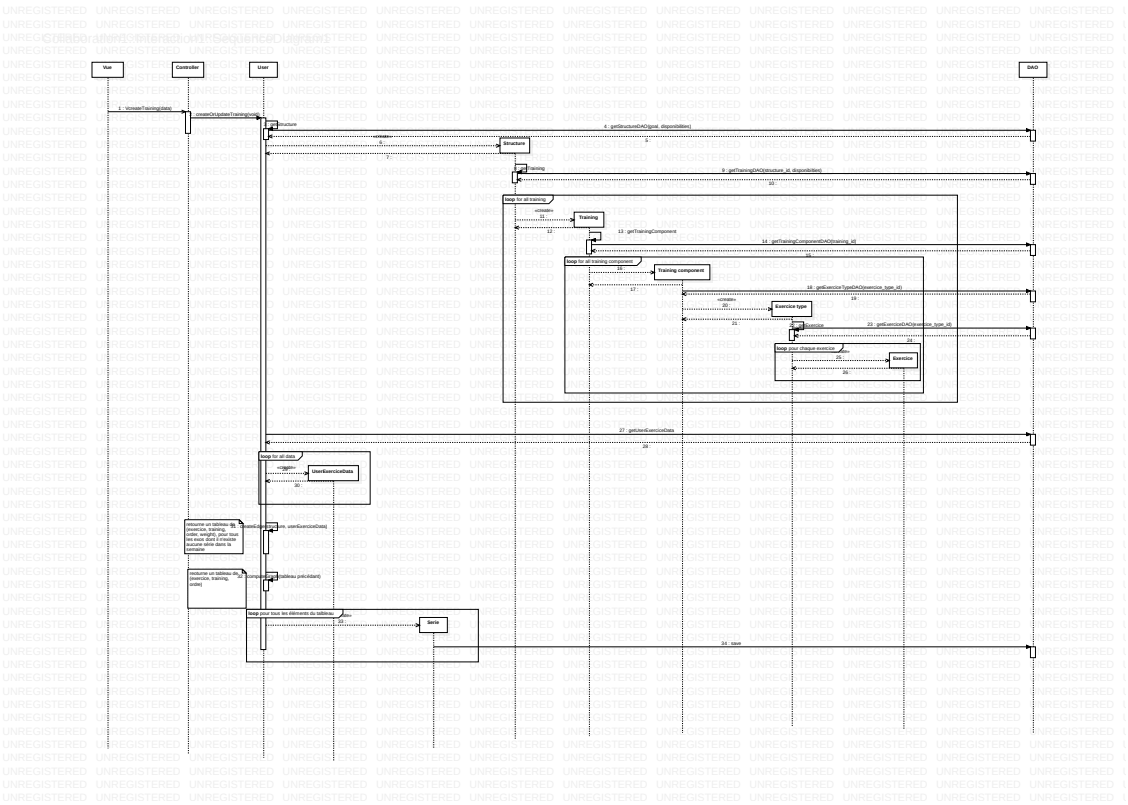
Ce n'est pas montré dans le diagramme mais j'ai utilisé le design pattern singleton et factory afin de ne faire qu'une seule connexion à la base de données et limiter la création d'instance dans le programme. Pour ce faire, il existe une classe singleton DaoFactory renvoyant des singletons des implémentations de chacune des interfaces DAO.

2.2.3 Diagramme d'activité



Ce diagramme d'activité montre le fonctionnement de la création d'entraînement. Le but était, pour chaque composant de l'entraînement, d'aller chercher la liste d'exercice associée. Enfin le programme va déterminer l'exercice optimal pour chaque composant et enregistrer dans la base de données le programme d'entraînement résultat.

2.2.4 Diagramme de séquence



Le diagramme de séquence montre plus en détail la méthode utilisée pour construire le programme d'entraînement.

1. Pour chaque composant : aller chercher la liste d'exercice dans la base de données
2. Récupérer les données utilisateurs pour chaque exercice.
3. Faire fonctionner l'algorithme de Kuhn afin d'optimiser le problème.
4. Récupérer le résultat de l'algorithme et enregistrer dans la base de données l'entraînement correspondant.

3 Justification des choix

3.1 Justification des choix de conception UML

Concernant la partie data Access du projet, j'ai choisi d'utiliser les design patterns factory et singleton afin de limiter l'accès à la base de données. En effet, utiliser un singleton pour la connexion permet d'avoir un accès unique à la base de données et donc d'accélérer les requêtes. De la même manière n'avoir qu'une instance DAO pour chaque objet concret du modèle permet de limiter le nombre d'instance créer dans le système et donc d'accélérer le processus.
De plus, j'ai choisi de créer pour chaque objet une interface DAO et une implémentation correspondant, permettant d'accéder à la base de données. Ce choix permet d'avoir une abstraction entre ce que je veux faire dans la base de données et la manière dont je le fais. En effet, la base de données pourrait utiliser un tout autre SGBD ou même un autre type de base de données. L'interface permet de séparer la partie accès à la base de données de la partie logique métier du modèle de données.

Pour le diagramme de classes de la partie logique du projet. L'encapsulation de chaque classe dans la classe User permet une création facile et rapide de la méthode createOrUpdateTraining(), cependant l'accès à chaque exercice pourra s'avérer long dans le futur s'il y a un grand nombre d'exercice à aller chercher.

3.2 Justification des choix de programmation

3.2.1 Utilisation des Exceptions

Les exceptions m'ont beaucoup servi tout au long du projet. Principalement pour la liaison à la base de données. J'ai créé des exceptions comme EmptyResultsQueryException indiquant qu'une requête SQL ne retournait aucun résultat. Celle-ci, par exemple, indique si l'utilisateur a rentré un mauvais mot de passe et prévient des erreurs lors de la création de l'entraînement si par exemple aucun exercice n'est associé à un composant.

3.2.2 Utilisation d'un package Utils

Afin de correctement organiser le code, j'ai créé un package Utils regroupant toutes les classes et méthodes aidant le programme principal sans y être directement associé. On peut y retrouver une classe DateGroup utilisé pour stocker des dates dans une composition de map, ceci est utilisé dans la gestion de l'historique de l'utilisateur. De même il y a une méthode permettant de crypter le mot de passe de l'utilisateur en utilisant MD5.

3.3 Justification des choix de conception IHM

Pour l'IHM, j'ai opté pour SceneBuilder afin de faciliter la création des composants.
J'ai choisi aussi de ne pas directement connecter les méthodes d'événement mais de configurer dans le contrôleur de fenêtre. Ceci m'a permis de mettre en place le design pattern Observer facilement et de manière générique pour chaque composant.

4 Problèmes rencontrés

Le principal problème a été le temps de calcul du programme. En effet, le logiciel pouvait prendre jusqu'à 5 secondes pour créer ou mettre à jour le programme d'entraînement ce qui est extrêmement long.
Pour résoudre ce problème j'ai d'abord déterminé quelle méthode prenait autant de temps, bien entendu c'était createorupdatetraining(). J'ai alors coupé la méthode en plusieurs parties, en séparant les appels à la base de données et l'algorithme d'optimisation. J'ai donc découvert que le problème venait des appels à la base, j'ai donc compris que les boucles imbriquées ralentissaient le programme, pour corriger cela j'ai réuni au maximum les requêtes Sql.
Je suis donc passé de 5 secondes à entre 1 et 2 secondes, ce qui était toujours trop long. C'est en analysant la classe DAOFactory que j'ai compris d'où venait le problème.
Je refaisais une connexion à la base à chaque requête, en effet, mon singleton était mal fait ce qui instanciait une nouvelle connexion pour chaque appel.

L'IHM ne m'a pas posée de problèmes, en effet. J'ai pris beaucoup de temps au début afin d'anticiper son implémentation. Je me suis donc renseigné et c'est là que j'ai découvert l'architecture MVC. J'ai opté pour cette architecture dès le début et même pour la partie ligne de commande afin de gagner un maximum de temps pour la suite.
Le problème ici, a donc été de correctement visualiser l'ensemble du projet et de correctement placer la partie logique du logiciel du reste. Il y avait cependant toujours un souci : Ou mettre createorupdatetraining() ? C'est après les premiers tests que j'ai compris que cela ne concernait que l'utilisateur et c'est donc pourquoi c'est cette classe qui porte cette méthode (cela a permis de finaliser le diagramme de classes).

Enfin le dernier problème que j'ai rencontré a été de comprendre l'aspect métier du projet. Mon frère n'étant pas dans le domaine de l'informatique, il a fallu beaucoup échanger, afin de lui faire comprendre ce qui est possible de faire, et pour moi de comprendre l'entièreté du raisonnement d'un préparateur physique afin de modéliser le problème. Le modèle conceptuel de données est la preuve du nombre de changements que nous avions dû faire, il a fallu faire beaucoup d'ajustement notamment comment on associe un composant de programme et les exercices possibles. Bien entendu le résultat final n'est pas fixe et nous pourrons le modifier si nous voulons rajouter des contraintes par exemple.

5 Améliorations possibles

Comment dit précédemment, la partie administrateur n'est pas disponible, c'est le point faible du projet. Il a été difficile de faire des tests avec seulement une vingtaine d'exercices. C'est pourquoi une amélioration serait d'ajouter la possibilité de mettre des exercices, des entraînements, etc. en plus dans la base de données.

Une autre amélioration aurait été de modifier l'architecture DAO utilisé en effet, il y a une assez forte redondance de code même en ayant factorisé les méthodes principales dans BasicRequestsDAO. Chaque implémentions d'interface doit redéfinir le nom de la table, la clé primaire et comment on construit un objet à partir d'un résultat de requête sql.

Par rapport à la logique métier du projet. Nous n'avons pas pu prendre en compte les blessures du pratiquant. Il est important de prendre cela en compte lors de la création d'un entraînement sportif. Le problème était que ce sont des facteurs qui relèvent de la santé et donc pour lesquels il faut prendre plus de précautions.

6 Conclusion

Réaliser ce projet a déjà était une grande chance pour moi, pouvoir travailler sur mon projet personnel pour un travail d'école m'a permis de consacrer beaucoup de temps dessus et donc d'en gagner. J'ai utilisé cette chance au maximum ce qui m'a permis d'apprendre énormément de choses. Déjà, sur le Java, faire un projet complet nous fait appliquer toutes les notions vues en cours et permet de les fixer dans la mémoire.
J'ai aussi pus beaucoup apprendre sur l'architecture d'un projet avec IHM, réfléchir à l'organisation des classes nous fait prendre du recul et nous contraint à visualiser en détail les différentes fonctionnalités. C'est pourquoi la partie conception a été très importante, réfléchir à la manière dont notre diagramme de classes va être utilisé dans Java nous permet d'être plus efficace lors d'autres conceptions.