PROJET LO21: UT'PROFILER

1. Présentation de l'architecture

Pour développer UT'Profiler, nous nous sommes rapidement décidés à sauvegarder toutes les données en base de données relationnelle SQLite (bdd.sqlite) qui est très léger et suffisamment performant pour notre utilisation. En effet, une fois la connexion réalisée, il est très facile d'interroger cette base pour récupérer nos données. De plus, le langage SQL est presque totalement géré par la table.
Nos tables sont les suivantes :

- Branche : Elle contient toutes les informations d'un cursus Branche donné (Nom/Description/Nombre de crédits dans chaque catégorie).
- BranchePourUv :Fait le lien entre les Uvs d'une branche et cette dernière. On peut alors savoir si elle est obligatoire, est de type PCB etc. . .
- CategorieUv:Permet de connaître la catégorie d'une UV en question avec son nombre de crédit.(Voir Évolutivité de l'architecture).
- Choix : Cette table relie le choix d'un étudiant pour une UV donnée de son cursus. Elle peut être voulue, neutre ou rejetée. (Une UV obligatoire peut être rejeté mais la base de donnée est faite pour que l'étudiant soit contraint à la faire quand même).
- Dossier:Contient toutes les informations propres à un étudiant identifié par son id. (Le Nombre de crédits validés, l'année,le semestre actuel etc. . .).
- Filière :Du même principe que Branche, on a accès à toutes les informations d'une filière pour un Cursus de Type Branche donné.
- FilièrePourUV :Elle possède les mêmes fonctionnalités que BranchePourUv mais pour les filières.
- Inscription:Cette table permet de vérifier les notes d'un étudiant pour un semestre donnée. On peut aussi avoir les UVS auxquelles un étudiant est Inscrit(En prévision) pour les semestres à venir.
- Mineur :De même que Filière, cette table est composé des informations d'un mineur.
- ListeUV:En relation avec Mineur, elle indique le nombre d'UVS à valider pour un mineur donné.
- MineurPourUV,TC,TCPourUv ; Du même principe que les tables précédentes.
- Uvs : Contient les informations liés aux UVS(Sauf leurs catégories et leurs nombres crédits qui se trouvent dans CategorieUv).

Étant donné que SQLite ne gère pas les accès concurrents (enfin, le logiciel les traite un par un en laissant en attente les autres), nous avons décidé de créer une classe Singleton, nommé DBManager, qui se chargera d'interroger, d'ajouter, de modifier ou de supprimer des données. De plus, puisque cette classe renvoie à chaque fois des booléens, des QList, des QString, elle permet de faire le pont entre les classes et la BDD. Ainsi, nous avons réalisé le design pattern Bridge qui permet une indépendance entre l'abstraction et l'implémentation.
DBManager ne vérifie ni la validité des arguments ni des méthodes qui n'utilise pas la BDD. Ceci est réalisé par les autres managers (CursusManager, UVManager, ETUManager). Ces classes sont aussi instanciable une unique fois (design pattern Singleton) pour être facilement accessible dans les différentes classes qui s'occupent de l'interface graphique.
Pour ne pas répéter le code, nous avons crée la template class HandlerSingleton qui détruit / créé les classes Singleton.

La classe prévision permet de proposer des UV pour chaque semestre restant du cursus courant de l'étudiant. Elle utilise le design pattern Strategy pour choisir l'algorithme de complétion de cursus en fonction du type de cursus. En fonction du type de Cursus, la classe prévision va appliquer un algorithme spécifique. En effet, pour un cursus Branche, il faut prendre en compte les crédits PCB et ceux de Filières. L'algorithme devient alors plus complexe, c'est pourquoi nous avons décider de le séparer en deux algorithmes distincts. Pressés par le temps et devant la difficulté occasionnée, nous n'avons malheureusement fait que l'algorithme pour un cursus TC. Vous pouvez retrouver en annexe l'algorithme de complétion.
Pour simplifier l'algorithme de prévision, nous avons réalisé une classe cursus qui compose la classe Prevision (elle n'est instancié que dans cette classe).
Cette classe Cursus et les classes filles regroupent toutes les informations propres à un cursus donné. Comme il y a de nombreuses combinaisons de Cursus, nous avons décidé de mettre en place le design pattern Decorator. Ainsi, nous pouvons « décorer », ajouter des contraintes et/ou des attributs dynamiquement à Cursus. Dans notre cas, nous avons ajouté des contraintes pour pouvoir modéliser les Branche, TC et Hutech (qui est très semblable à un cursus TC).
Pour fabriquer la bonne classe cursus (s'il y a des UV obligatoires ou non, s'il y a un nombre de crédit minimum à avoir dans une catégorie, ….) nous utilisons le design pattern Factory, dans CursusFactory. En fonction du nom du cursus ou de l'id de l'étudiant, cette classe construit le Cursus adéquat.

Comme il y a des attributs propres à des classes filles (comme nbCreditCS), nous avons dû utiliser le design pattern Visitor pour avoir accès à ces donnés à partir d'un pointeur vers Cursus. Dans notre cas, nous avons 6 classes Visitor pour connaître nbCreditLibre, nbCreditCS, nbCreditTM, nbCreditTSH, nbCreditPCB, nbCreditPSF. Chaque classe hérite de la classe abstraite VisitorCursus.

Nous avons également une classe ErrorManager qui reçoit toutes les erreurs envoyer par les différentes classes que nous avons codées. Cette classe enregistre les erreurs dans un fichier log.txt sous le format :

Error : NomDeLaCLasse : Texte de description de l'erreur : heure et date de l'erreur

Ainsi, il est beaucoup plus facile de résoudre les différents problèmes rencontrés. Cette classe est instanciable une unique fois (design pattern Singleton) car elles doit être accessible de manière connue et que le fichier log.txt ne gère pas les accès concurrents.

Le fichier enumeration.h contient tous les types énumérés qui sont utiliser dans UT'Profiler : le type CategorieUV pour les différentes catégories d'uv

2. Évolutivité de l'architecture

Nous avons pensés notre Architecture de tels sorte qu'elle soit le plus flexible possible dans la plupart des domaines. Ainsi, si l'université décide de rajouter des nouvelles catégories pour les UVS, ou des catégories mixtes(Par exemple, LO21 devient une CS/TM). Il suffit de l'intégrer à la table Catégorie UVS. Le nom de l'UV et la catégorie étant la clé primaire de la table, on aurait alors dans cette dernière ('LO21','CS','4') et ('LO21','TM','6').
De par l'utilisation du design pattern Bridge, il est possible de changer l'implémentation (de passe de l'utilisation d'une base de données à l'utilisation de fichier XML par exemple) en ne modifiant que la classe DBManager.
Notre class cursus suit le design pattern decorator. Ainsi, on peut ajouter de nouvelles contraintes assez facilement.
Nous avons décider de séparer les cursus de type Branche et type TC pour pouvoir adapter notre code en fonction.
Notre class Prevision utilisant le design Pattern Strategy, il est possible d'ajouter des algorithmes pour proposer de nouvelles prévision au cursus.

3. Limite de notre architecture

Cependant, notre architecture présente certaines limites. Tout d'abord, nous ne prenons pas en compte les stages dans l'architecture, ce qui représente une perte d'information. Malgré cela, nous en tenons compte dans notre prévision puisque si le type du cursus de l'étudiant est un type Branche, la prévision est faite sur 4 semestres au minimum, 8 au maximum.
Notre architecture a prévu les Mineurs, c'est à dire qu'un étudiant peut s'inscrire à un mineur. Cependant, nous ne tenons pas en compte du Mineur choisi par l'étudiant dans la complétion. En partant du principe que l'étudiant veut valider son mineur, il va alors mettre les UVS TSH qu'il doit valider en voulue.
Nous ne prenons pas en compte la grille de validation des TSHS durant notre complétion, ni même dans le dossier étudiant. Une solution pour remédier à cela serait de rajouter un colonne dans Catégorie UV, et de test si l'UV est TSH et d'obtenir son « type ».
Pour le design Pattern Strategy, l'algorithme pour la complétion d'un Cursus Branche n'a pas été faits. Malheureusement, par manque de temps, nous avons préféré nous concentrer sur plus importants.

4. Limite de notre interface graphique

Par manque de temps, certaines fonctionnalités de notre programme n'ont pas pu être ajouté graphiquement.
En effet, toutes la partie concernant les Mineurs (Ajouts d'un Mineur notamment) n'a pas été implémentée graphiquement.
De même pour la prévision, qui a été fait au niveau du code, mais n'a pas été implémentée au niveau graphique.
Afin de montrer au maximum notre architecture, certaines applications graphiques ne sont présentes que pour certains Cursus.

4. Annexe

Début de l'explication de l'Algorithme prévision :

```
void Prevision::prevision(QString cursus,int id){
   QList<QString> UValide= new QList<QString>; // La fonction va travailler sur une liste D'uvs qui sera rangée dans l'ordre suivant : UV Obligatoire/UV Voulue/UV Neutre/UV Rejeté
Vu qu'il s'agit d'une prévision sur les semestres à venir, on prends le semestre actuel de l'étudiant que l'on permute. (Automne devient Printemps et inversement).
   while(i<UV.size() && NombreCreditActuelCursus < NombreCreditCursus)
   {// Tant que la liste d'uvs formées n'est pas finites et que tous les crédits du Cursus ne sont pas validés, la boucle while continue //
      if(semetreActuel<6) // On part du principe que la personne peut faire 2 semestre en plus, à partir de là, elle sera renvoyée//
      {
         if(NombreUVSemestre<7 && NombreCreditCeSemestre<31) // Pour faire l'alternance des semestres. Cependant, il pourra avoir plus de 31 ce semestre.
         {
            if( benefique(UV[i],NbCreditCS,NbCreditTM,NbCreditTSH,cur) && isEnseigne(UV,semestre)) // Si l'uv améliore notre profil et qu'elle est enseignée ce semestre, On rajoute alors l'UV à la list des UVS valide.
            {
               UValide.push_back(UV[i]); //

                                    Compte des Crédits de L'UV Validée // Rajout des crédits de l'UV dans les crédits Totaux du semestre et dans la catégorie de celle-ci.

               NombreUVSemestre++; // On incrémente le nombre d'UV Pour ce semestre de prévision
            }
         }
         else
         {
   // On Inscrit les UV Validées par l'étudiant.
               for(j=0;j<NbUVSemestre;j++)
               {
               InscriptionUValide(id,UVValide[j],annee,semestre);// On met les Notes des UVS à INSCRIT et le semestre et l'annee ou il compte le faire
               }
               if(semestre=='Automne') // On fait l'alternance des semestres
                 semestre='Primtemps';
               else
               {
                 semestre='Automne';
                 annee++;
               }
               UV=cursus::NewList(cursus); // on fait la nouvelle liste d'uv qui prends en compte les UVS déjà choisies.
               i=0; // On remet le " compteur de la liste uv " à 0;
               NombreUVSemestre=0; // On remet le compteur UV et Credit à 0 car il s'agit d'un nouveau semestre.
               NombreCreditCeSemestre=0;
               Compte des crédits de l'UV // On compte les crédits de L'uvs.
               NombreUVSemestre++;
               }
            }
         }
   }
      i++; // On incrémente le nombre d'uv parcourue dans la liste d'uv
   }
}
```