Rapport

Projet d'interface homme machine.

2014 - 2015

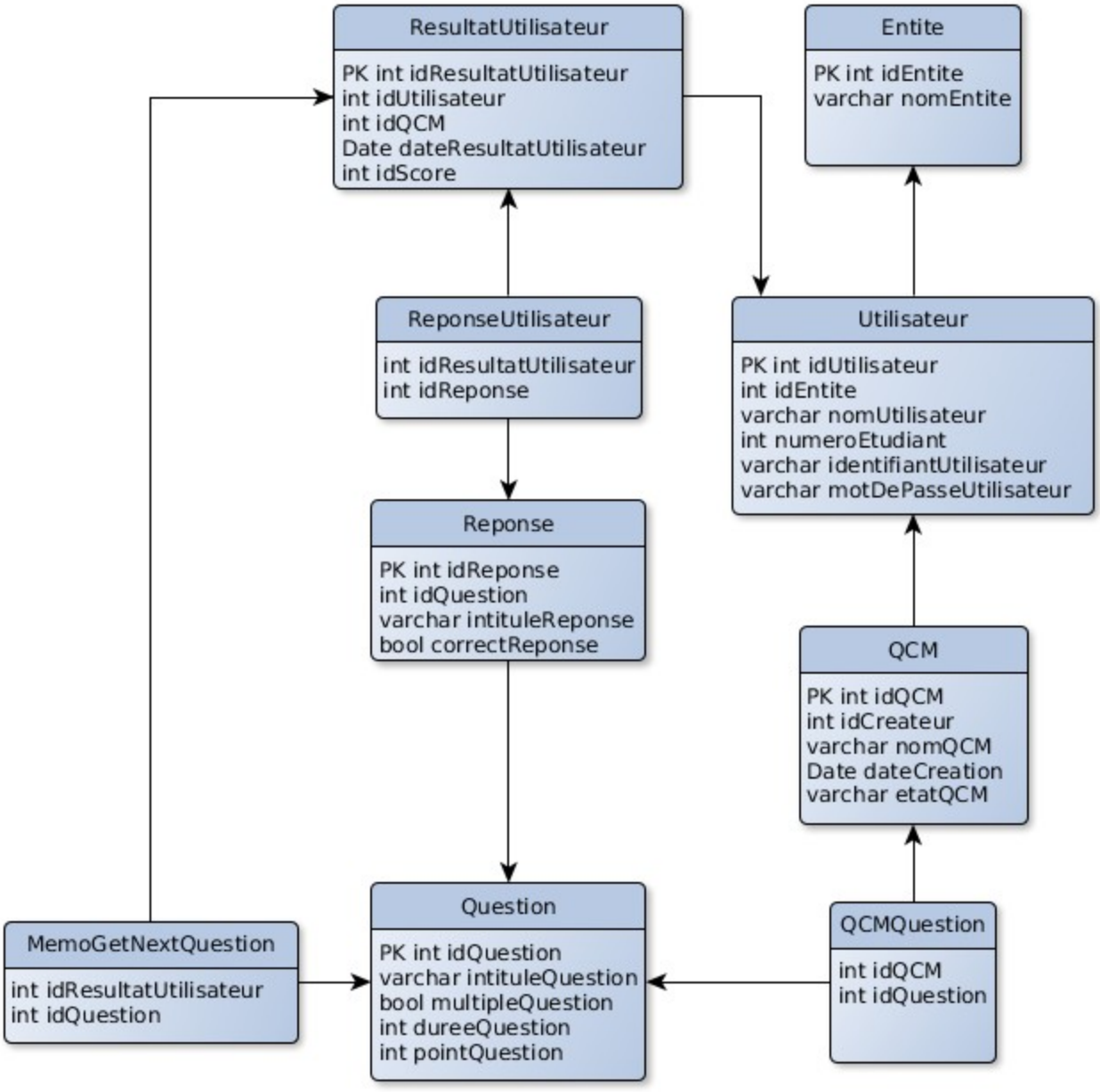
0 – Architecture générale de l'application

L'application est découpée en 3 parties :

- Modele, contenant le modèle et la couche de sauvegarde.
- Struts, contenant le site web (client léger).
- Swing, contenant le client lourd.

A – Base de donnée

Nous avons mis en place une basse de donnée H2, avec l'architecture suivante :

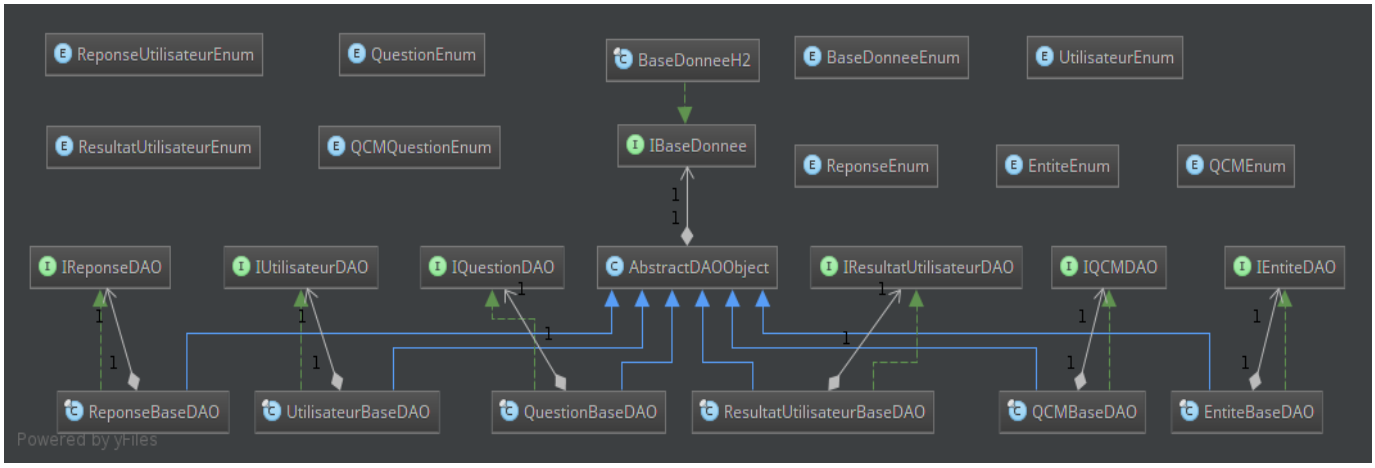


Nous avons donc, une table :

- Question.
- Reponse.
- Utilisateur.
- Entite, permettant de faire la différence entre un étudiant et un professeur.
- QCM.
- QCMQuestion, permettant de lister l'ensemble des questions d'un QCM.
- ResultatUtilisateur, la table représentant la participation d'un utilisateur à un QCM.
- ReponseUtilisateur, permettant de lister l'ensemble des réponses d'un utilisateur lors d'une participation à un QCM.
- MemoGetNextQuestion, permettant de lister l'ensemble des questions posées à l'étudiant lors d'une participation à un QCM.

B – DAO

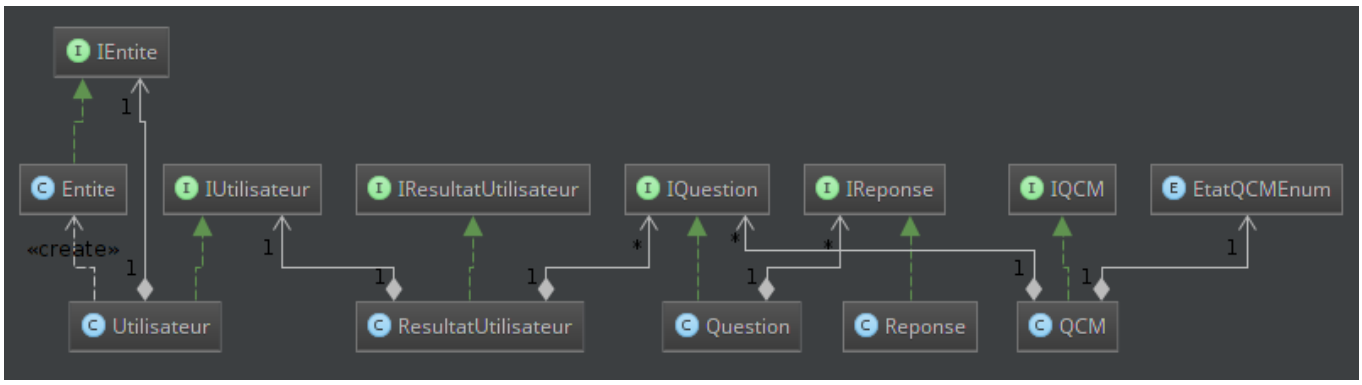
Afin d'utiliser notre base de donnée, une couche d'abstraction à été mis en place avec l'architecture suivante :



Nous avons créé un ensemble d'énumération afin d'avoir accès rapidement aux champs d'une table et à la liste des tables présent en base.

Chaque classe concernant une table, implémente une interface qui lui est propre, afin de pouvoir mettre en place plusieurs implémentation en fonction du type de couche de sauvegarde, sans avoir à changer l'utilisation de ces classes. Pour la même raison, chaque classe étend de AbstractDAOObject qui permet d'abstraire la connexion à n'importe quel type de couche de sauvegarde.
Enfin, la connexion à la base de donnée H2 est implémenté dans la classe BaseDonneH2.

C - Modèle



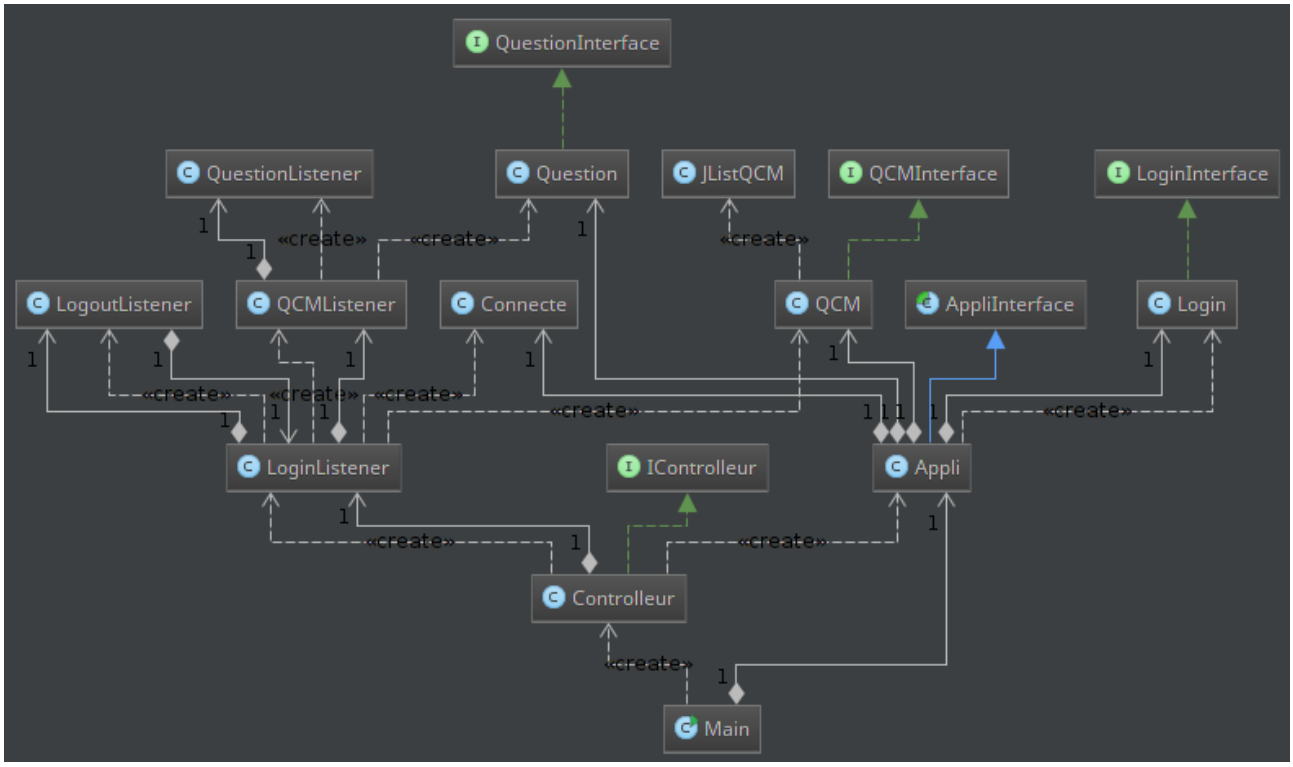
Architecture du modèle

De manière logique, on retrouve :

- La classe Utilisateur contenant une Entite.
- La classe Question contenant une liste de Reponse.
- La classe ResultatUtilisateur contenant une liste de Question et un Utilisateur.
- La classe QCM contenant une liste de Question.

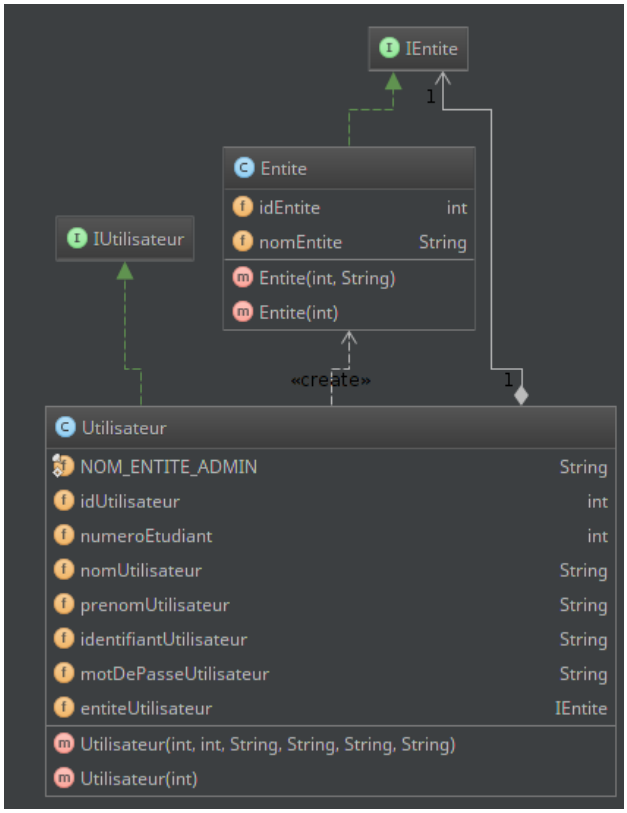
D – Struts

E – Swing



1 – Objets Métiers

A – Entite et Utilisateur

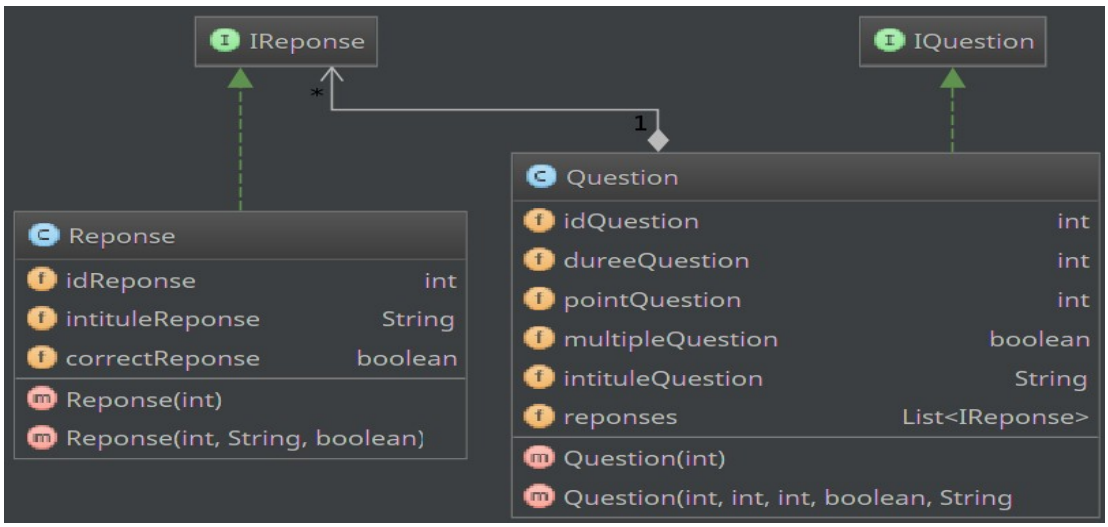


Un utilisateur est défini par son nom, prénom et s'il est étudiant d'un numéro d'étudiant, c'est pourquoi un utilisateur contient une instance d'Entite, qui permet de savoir s'il est étudiant ou professeur.

Nous choisi de représenter une entité avec un idEntite et un nomEntite pour bénéficier d'une plus grande évolutivité, par exemple il serait possible de mettre des entités hiérarchisées en base de données (Etudiant > Master > Informatique > Première année).

Enfin, un utilisateur doit également avoir un identifiant et un mot de passe afin de se connecter à l'application.

B – Question et Reponse

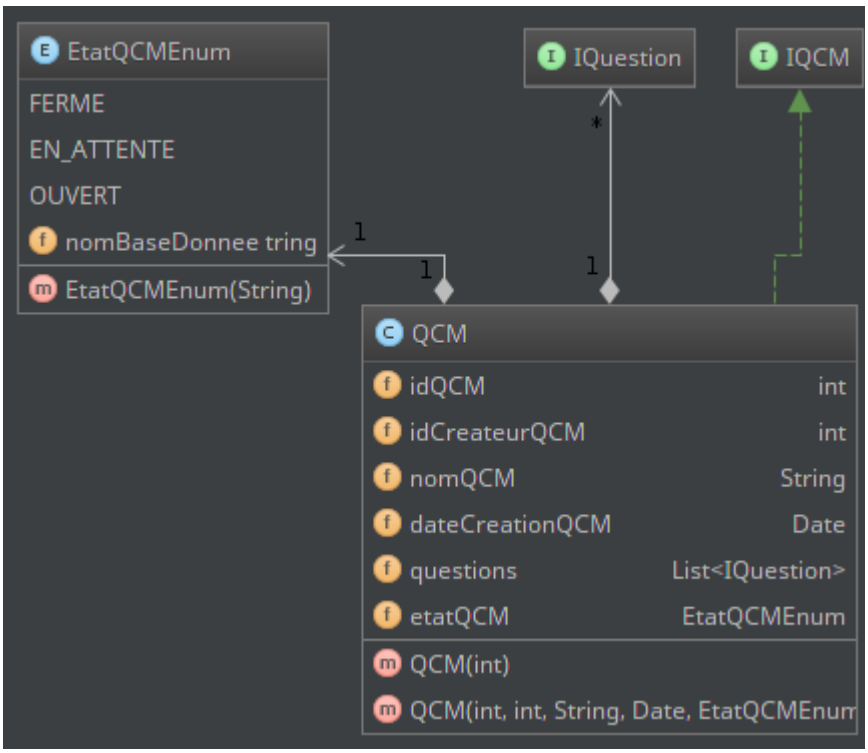


Une réponse est déterminée par un idReponse pour l'intéraction avec la base de donnée, un intitulé et un booléen afin de savoir si la réponse est correct.

Une question est déterminée par un idQuestion pour la base de donnée, un intitulé, le nombre de points que la question rapporte, la durée (en secondes) correspondant au temps durant lequel on peut répondre à la question, et un booléen afin de savoir si la question est multiple, c'est-à-dire si on peut sélectionner une seule ou plusieurs réponses.

Et enfin, une question est constituée d'une liste de réponses, nous avons choisi une liste pour un accès rapide, également car le nombre de réponses n'est pas connu à l'avance et enfin pour conserver l'ordre.

C – QCM

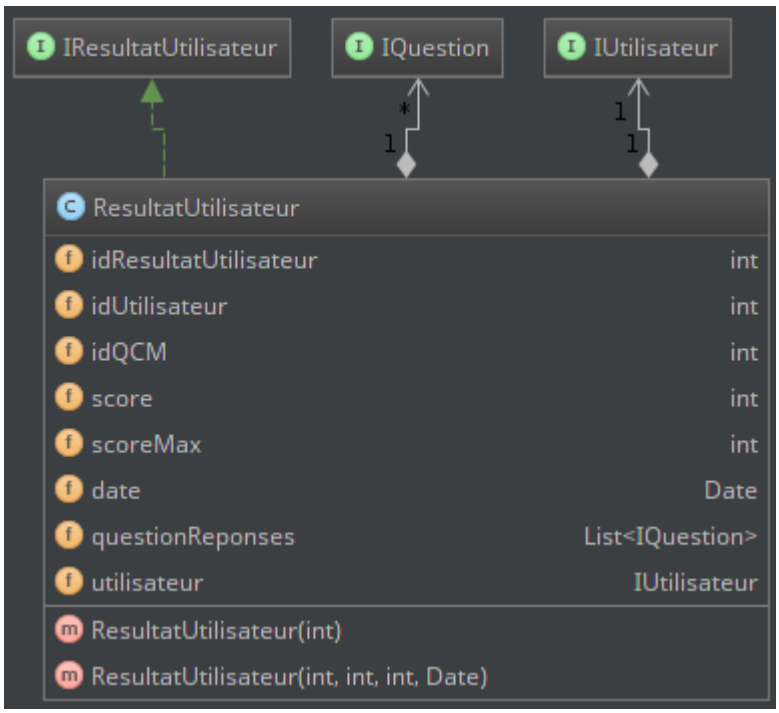


Un QCM est défini par son identifiant, l'identifiant de son créateur, son nom, sa date de création et son état (EtatQCMEnum) qui peut avoir trois valeurs : ouvert, fermé et en attente.

Lorsqu'un QCM est ouvert les utilisateurs peuvent participer, et lorsqu'il est fermée ou en attente, il ne peuvent plus participer, même si un utilisateur était en cours d'une participation.

Un QCM possède aussi une liste de question, nous avons choisi une liste pour un accès rapide, également car le nombre de questions n'est pas connu à l'avance et enfin pour conserver l'ordre.

D – Résultat utilisateur



Un résultat utilisateur est définie par son identifiant, l'identifiant de son utilisateur, l'identifiant du QCM, le score, le scoreMax du QCM (pour des raisons pratiques lors de certaines requêtes DAO) et sa date de participation.

Il possède aussi une instance d'utilisateur, pour des raisons pratiques lors de certaines requêtes JSON.

Enfin, il contient une liste de questions, et chaque question contient la liste des réponses de l'utilisateur.

2 – Couche de sauvegarde

Comme vu précédemment, chaque classe concernant une table, implémente une interface qui lui est propre, en plus d'hériter de AbstractDAOObject.

Les classes concernant une table, ainsi que la classe BaseDonneH2 sont des classes singleton, c'est-à-dire qui n'ont qu'une seule instance, puisqu'il est inutile d'avoir plusieurs instance de ces classe.

Chaque classe qui concerne une table, implémente l'ensemble des fonctions de sélection, de modification et de suppression des données en base de donnée, ainsi que parfois des fonctions plus spécifiques comme (liste non exhaustive) :

- getNextQuestionQCM(int idQCM, int idResultatUtilisateur)) dans IQCMDAO, qui permet d'obtenir la prochaine question à un QCM (en prenant la question avec le moins de participation à l'instant t) et insère l'identifiant de la question posée dans la base de donnée afin de ne pas la reproposer à nouveau.
- calculerScore(int idResultatUtilisateur) dans IresultatUtilisateurDAO, qui permet de calculer et de sauvegarder le score d'un résultat utilisateur.

Enfin, l'ensemble des fonctions sont disponibles via un service fourni par le serveur RMI qui se lance lors du lancement de l'application web.

De plus, lors des tests unitaires, une base de données spécifique à cet usage est utilisée.

Afin d'éviter les injections SQL, nous avons utilisé des requêtes paramétrés, qui permettent d'éviter les injections SQL et de demander à la base de donnée de préparé la requête avant de l'utiliser. Par la suite, si l'on souhaite exécuter plusieurs cette requête avec des valeurs différentes, il est possible de relancer la requête sans avoir à repasser par la phase de préparation de la requête par la base de donnée, puisque cette phase à déjà été effectuée, gagnant ainsi en performances lors de multiples utilisations consécutives.

3 – Struts

4 – Swing

Afin d'appliquer la conception MVC, les classes « écouteurs » ont été séparé des JPanels de la vue. Ainsi quand, par exemple, on clique sur un bouton, c'est la classe écouteur correspondante qui effectue les différentes modifications à faire et met à jour la vue.

Au lancement de l'application, le modèle et un contrôleur sont créent, le contrôleur prenant en paramètre le modèle. Ensuite, le contrôleur initialise la vue, qui prend aussi en paramètre le modèle. La vue est initialisée avec un premier JPanel permettant de se connecter. Un contrôleur est lié à ce JPanel et permet de changer de JPanel (quand les conditions le permettent) afin d'avoir les bonnes informations dans la vue.

Chaque JPanel à un contrôleur sous forme de classe écouteurs. Ces contrôleurs créent de nouveaux JPanels s'il n'ont pas été créé auparavant dans la vue, sinon ils rendent invisibles les JPanels qu'ils faut enlever de la vue et rendent visibles les JPanels qu'il faut remettre.

Les événements utilisés sont des ActionLiteners, les classes « écouteurs » correspondantes sont *LoginListener*, *LogoutListener*, *QCMListener* et *QuestionListener*.

La vue est représentée par *Appli*, et regroupe les différents JPanels *Login, Connecte, QCM, Question*.

Les layout utilisés sont des BorderLayout (*Connecte* est placé au Nord, les autres JPanels sont placés au Centre.

Les communications clients/serveurs sont faites via le service (*ModeleService*) du RMI :

La classe *LoginListener* utilise :

- *getUtilisateurByIdentifiant* pour tester si l'utilisateur est dans la base de donnée
- *validerMotDePasseUtilisateur* pour tester si le mot de passe rentré est bon pour l'utilisateur

donnée

La classe *QCMListener* utilise :

- *getQCM(idqcm).isOpened()* pour obtenir les qcm ouverts.

La classe Question*Listener* utilise :

- *getNextQuestionQCM* pour avoir la question suivante du qcm

La classe *Question* utilise :

- *creerResultatUtilisateur* pour créer de nouveau résultats pour l'utilisateur
- *getQCMWithQuestionList pour avoir le qcm avec ses questions*
- *getFirstQuestionQCM pour avoir la question du qcm*

La classe *QCM* utilise :

- *getListQCMDispo* pour obtenir la liste des qcm disponibles

La classe *JListQCM* utilise :

- getListQCMDispo().size() pour avoir la taille de la liste des qcm disponibles
- getListQCMDispo().get(index) pour avoir le qcm au rang index

5 – Répartition du travail

Éléonore GÉDÉON : Swing

Alexis LAVIE : Struts

Sullivan PERRIN : Modele, DAO et Struts