Faculté de Sciences de Montpellier

Master 1 AIGLE
2016 – 2017

Projet Développement logiciel pour mobiles

Projet 3 : Cours en Ligne

Rapport de projet

Étudiants:
Olivier Montes
Sacha Weill

Sommaire

Introduction

Enoncé du sujet :

Projet 3: Cours en ligne

L'objectif de ce projet est de développer une application mobile de cours en ligne, et de permettre à des étudiants de pouvoir consulter différents types de ressources, tels que des contenus multimédias (vidéos de cours, ...), des présentations en PDF ou en PPT, des feuilles de TD/TP ou n'importe quel autre type de documents . Chaque utilisateur qui accède à cette plateforme bénéficie de droits spécifiques. Par exemple, les étudiants inscrits en L3 n'ont le droit de consulter que les documents qui ont été rendus disponibles pour les étudiants L3 par les formateurs. En plus du dépôt et de la consultation de documents, la plateforme offre la possibilité de réaliser des évaluations programmées des connaissances des étudiants. Il serait possible pour un formateur de définir, via la plate forme, des QCMs et d'inviter certains étudiants à y répondre pendant un intervalle de temps déterminé. Les réponses des étudiants seront automatiquement évaluées.

L'exemple donné (schoolmouv), est axé sur le lycée, dans le but d'aider les élèves inscrits à préparer leur bac. Nous avons choisi de donner un aspect plus universitaire à l'application, en s'inspirant de certains aspects de moodle, tout en gardant la construction général de l'exemple donné.

Notre choix s'est porté sur ce sujet car c'est le seul qui nous parlait en tant qu'étudiant, le seul que nous téléchargerions et utiliserions si nous devions choisir parmi la liste des sujets proposés.

Choix Techniques

1 Les Utilisateurs

Il fallait différencier deux types d'utilisateur : Les enseignants et les étudiants. Pour ce faire, nous avons choisit d'affilier chaque utilisateur à une année d'étude, les enseignants n'en ayant aucune. Ainsi on peut facilement savoir à quelle cours un Utilisateur a accès, puisque l'une des consignes du sujet était de pouvoir empêcher l'accès des cours en fonction du niveau d'étude. Les enseignants en revanche, ont accès à l'ensemble des cours stockés sur le site.

Pour connaître le statut d'un utilisateur, il est donc nécessaire de l'identifier, et pour ce faire, il faut mettre en place un système d'authentification des utilisateurs.

2 Les Cours

Nous avons décidé lors de l'étude technique de notre projet de s'inspirer de moodle et du fonctionnement de la FdS pour gérer les cours. Ainsi, nous avons choisi de les trier en plusieurs étapes :

- D'abord la Matière, c'est à dire Mathématiques, Informatique, Biologie, etc.
- ensuite l'Intitulé , comme pour la FdS, nous avons choisit de rassemblé les cours sous des "sous matière", telle que HLIN101, HLIN102, etc.
- enfin, sous l'intitulé, deux catégories sont proposées, QCM, ou cours.

Nous avons choisi d'affilier non pas les cours à un niveau d'étude, mais les intitulés. Nous considérons donc que tous les cours appartenant à un intitulé sont inaccessibles pour ceux qui n'ont pas accès à l'intitulé lui même.

3 Les QCMS

Les QCMs sont sérialisable en une chaîne de caractères, séquable par questions et réponses, permettant le stockage du QCM sous la forme d'une simple String.

4 La synchronisation des données

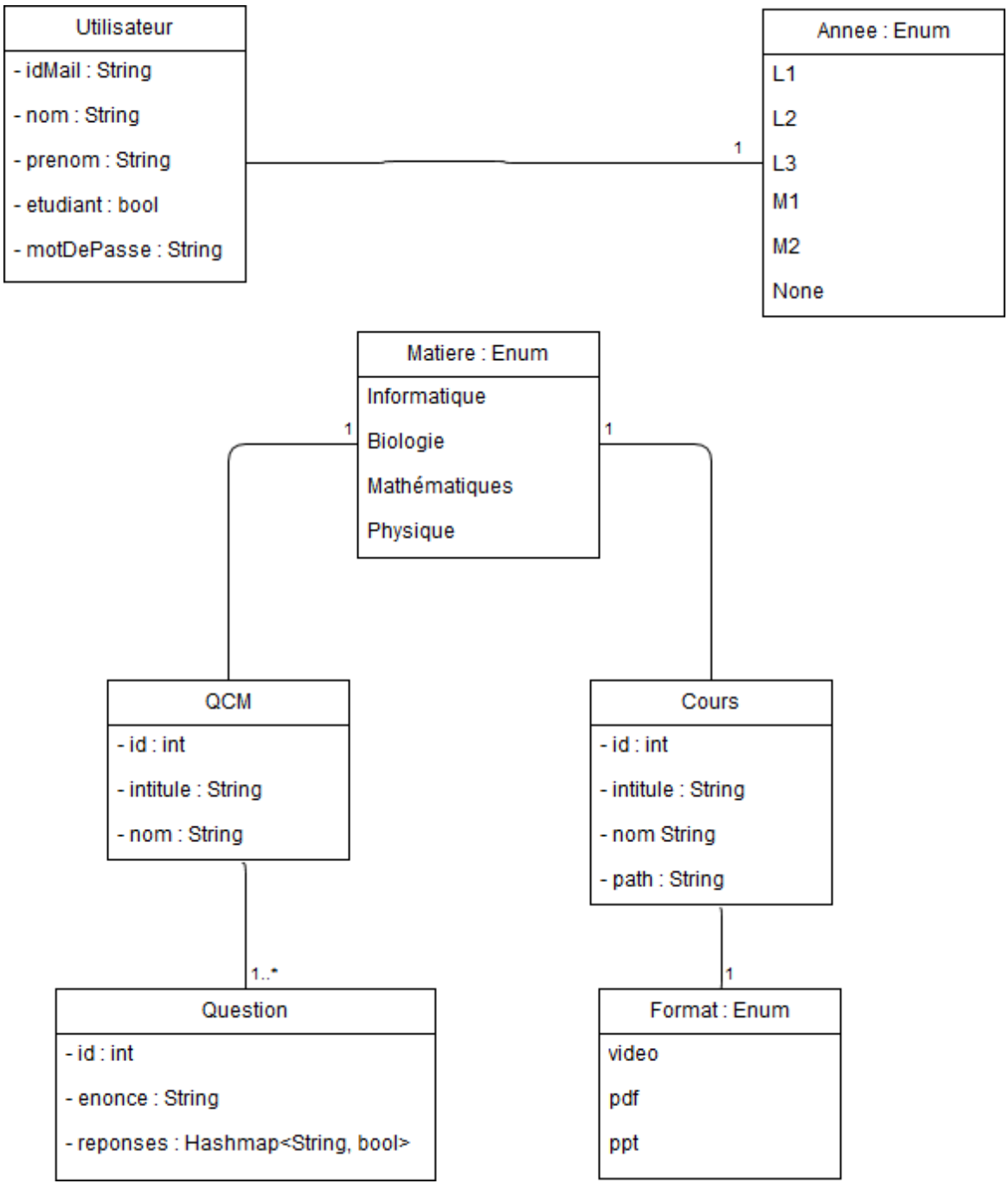
Puisque le but est le partage de document, il fallait décider du moyen de les partager à tous les utilisateurs. Nous avons décider de construire notre application sous la forme Client/Serveur

Ainsi, le client est représenté par l'application mobile et est développée sous androidStudio.

Tandis que le serveur doit être fixe et en permanence connecté au réseau, et doit donc être sur un poste fixe, c'est pourquoi nous l'avons développé en java simple (sans bibliothèques Android). Il doit également disposer d'une base de données pour pouvoir stocker les différentes informations nécessaires au fonctionnement de l'application(Cours, données des utilisateurs, etc.).

5 Diagramme de Classes

Lors du commencement de notre projet, nous avons élaboré le diagramme UML, représentant les différents objets devant être manipulés par les clients et le serveur.



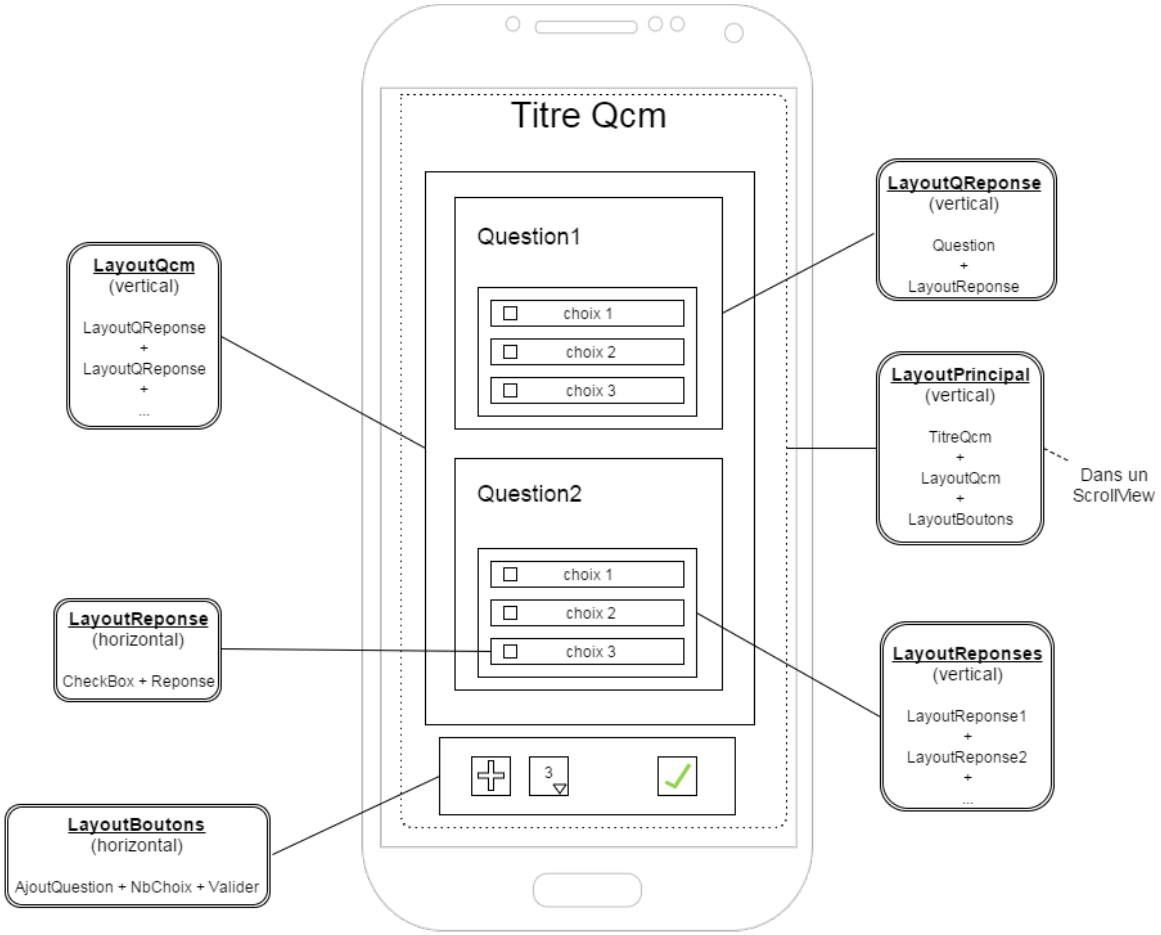
Ces classes ont vocations à être communes aux clients et au serveur, même si elles sont utilisées différemment par les deux partis. Par exemple le mot de passe d'un utilisateur, pour des raisons de sécurité évidentes, ne sera jamais stocké par l'utilisateur.

Il est a noté que les Intitulés ne sont pas représenté dans ce diagramme. Ayant vocation à être plus ou moins souvent modifié, ajouté, ou supprimé, ils ne pouvaient pas être contenus dans une enum, et sont par conséquent gérés par la base de données du serveur qui s'occupe de l'intégrité de leur nom.

Coté Client

Le rôle du client n'est donc que l'affichage et l'interprétation des informations reçues par le serveur. Le client ne stocke aucune information lui-même à l'exception de l'utilisateur qui joue le rôle de session courante, et de quelques informations minimes dont le but étaient d'améliorer la persistance des données entre les différentes activités.

1 Exemple de conception d'une activité - Création QCM



Il y a bien sûr trop d'activité pour pouvoir tout définir précisément et beaucoup sont des activités très simples, mais la création de QCM possède deux attraits sur lesquels il peut être intéressant de s'attarder : l'interface dynamique avec les interactions des boutons, et le stockage particulier des layouts pour pouvoir les mettre à jour.

Problème initial Le bouton ajouter (représenté par un "+" dans le schéma ci-dessus, qui est censé ajouter un ensemble de question+choix), met à jour l'interface des qcms. Il faut donc pouvoir récupérer les références des layouts afin de les modifier lors d'un clic du bouton ajouter. Une première approche naïve avait été de stocker le layout principal tel quel, et de le parcourir grâce aux fonctions *viewGroup.getChildCount()* et *viewGroup.getChildAt(i)*, en castant (puisque le child est un simple View) à chaque itération la view en ce qu'elle est censée être logiquement. Cette approche mena à un débuggage difficile et à un code difficilement relisible. C'est pourquoi dans un but de simplification de la gestion des layouts, nous avons introduit les classes ci-dessous (représentées par les doubles carrés à bords arrondis dans le schéma).
L'interface des qcms se compose donc de 6 types de layouts différents entremêlés :

- *LayoutReponse*, qui est donc une classe personnalisée héritant de *LinearLayout* et qui est composée d'une CheckBox et d'un TextView représentant la réponse.
- *LayoutReponses* qui hérite de LinearLayout aussi et est composé d'un Vector de LayoutReponse. C'est donc le layout représentant la liste des choix pour une question.
- *LayoutQReponse* encore une fois qui est un LinearLayout composé d'un TextView, où la question est stockée, et d'un LayoutReponses.
- *LayoutQcm* enfin est composée d'une liste (Vector) de LayoutQReponse.

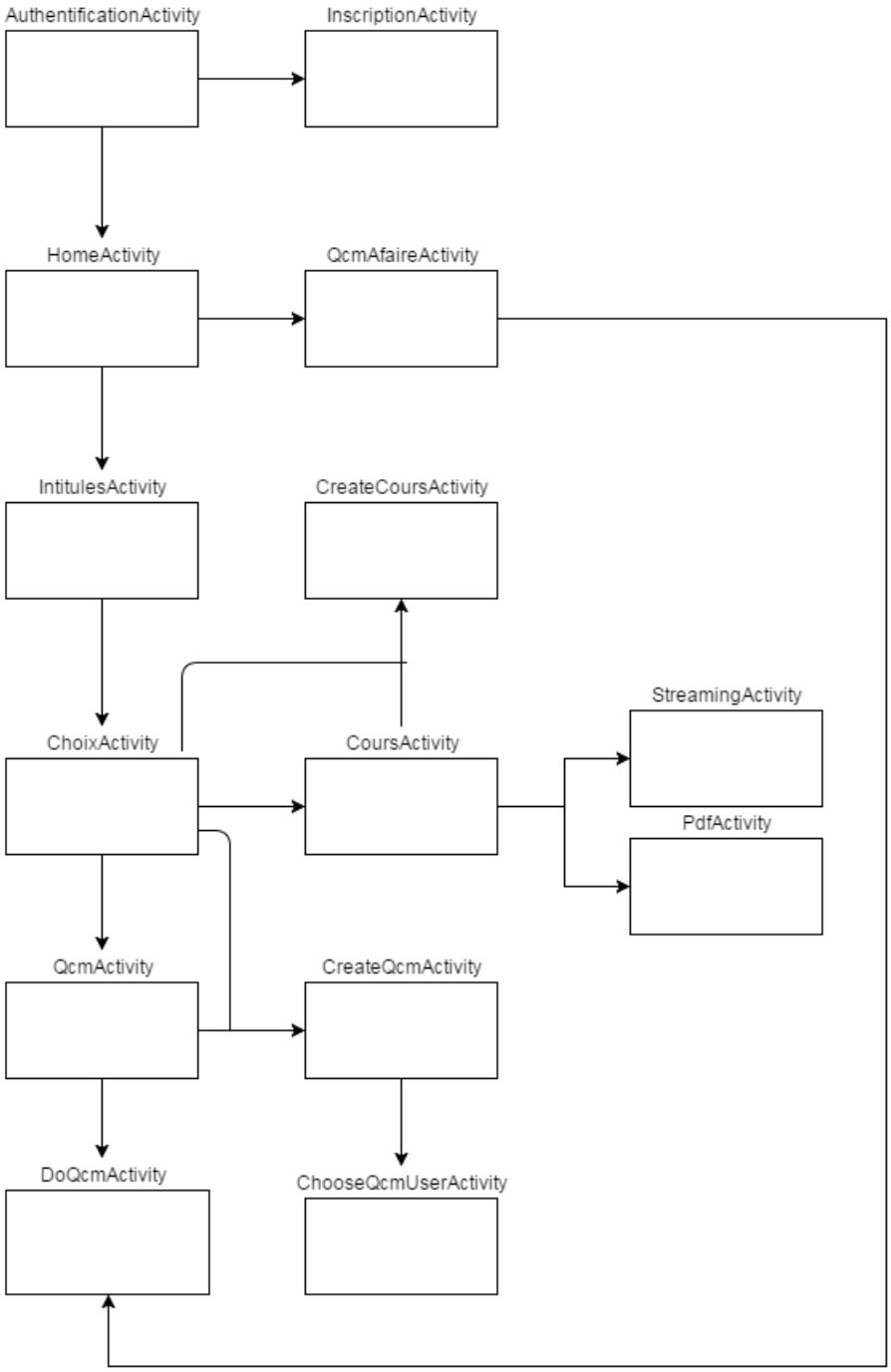
Le layout des boutons n'a ni besoin d'être stocké dans un layout particulier, ni n'est dynamique, il est donc représenté par un simple LinearLayout, il en va de même pour le layout principal (composé du titre, du layoutQcm et du layoutBoutons), qui est ensuite mis dans un ScrollView et est finalement passé en paramètre de la fonction *setContentView(view)*.
Le layout que l'on stocke pour mettre à jour lors de l'appuie du bouton ajouter est donc le *LayoutQcm*, où on récupérera la valeur du Spinner représentant le nombre de questions à ajouter (limité à 10). Le bouton valider ensuite enclenche un parcours de l'objet LayoutQcm pour récupérer les valeurs des CheckBox et des TextView des questions et des réponses où aucun cast n'a besoin d'être fait cette fois, permettant une programmation bien plus aisée et moins dangereuse.

2 Navigation des activités

De nombreuses activités composent l'application. En tout il y a 14 activités distinctes qui sont celles-ci :

Authentification	Activité permettant de s'authentifier
Inscription	Activité permettant de s'inscrire
Home	Activité donnant la liste des matières
Intitules	Activité donnant la liste des intitulés d'une matière donné pour un utilisateur donné
Choix	Activité où l'utilisateur choisi si il veut consulter les QCMs ou les Cours d'un intitulé donné
Qcm	Liste des Qcms d'un intitulé donné
DoQcm	Activité où un utilisateur pourra faire un Qcm et le valider
CreateQcm	Activité pour les professeurs où ils pourront créer un qcm
ChooseQcmUser	Liste des étudiants inscrit à l'intitulé pour le qcm en cours de création
Cours	Liste des cours d'un intitulé donné
CreateCours	Activité de création d'un cours pour un professeur
Streaming	Activité Youtube permettant le streaming vidéo des vidéos du site
Pdf	Activité permettant de visionner un pdf par une WebView
QcmAfaire	Liste des QCMs qu'un étudiant n'a pas encore validé et dont les professeurs ont notifié le qcm au dit élève

La navigation de ces activités démarre par l'écran d'authentification et se compose de la manière ci :



On notera qu'il est possible de revenir à l'écran d'authentification ou Home par des boutons présent dans la quasi totalité des activités dans le menu (les liens ne sont pas représentés afin de ne pas trop alourdir le schéma).
Dans le cadre du projet nous avons aussi ajouté une Activité qui vient se placer avant l'authentification, où l'utilisateur peut choisir l'adresse IP du serveur et essayer de communiquer avec avant de se lancer dans l'application elle-même par le biais d'un bouton "Skip" [1].

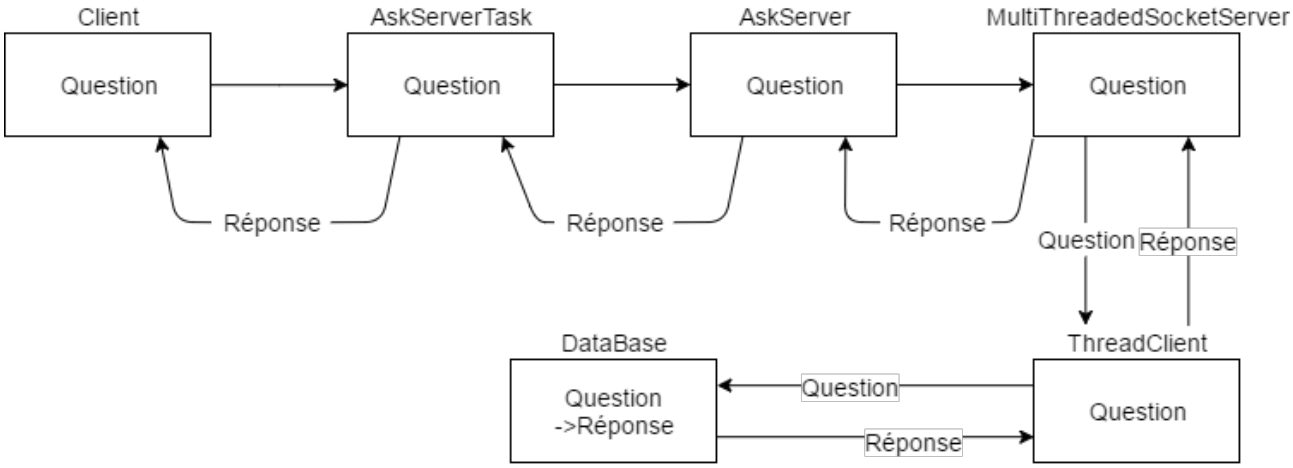
[1] Dans le malheureux cas où la connection avec le serveur est impossible pour des raisons de portforwarding ou tout autre raison empêchant le mobile de communiquer avec le serveur, nous avons mis en place un mode de navigation où le client reçoit des informations fixes pour permettre au moins de voir à quoi ressemble l'application à défaut de pouvoir réellement interagir avec

Coté Serveur

1 Protocole

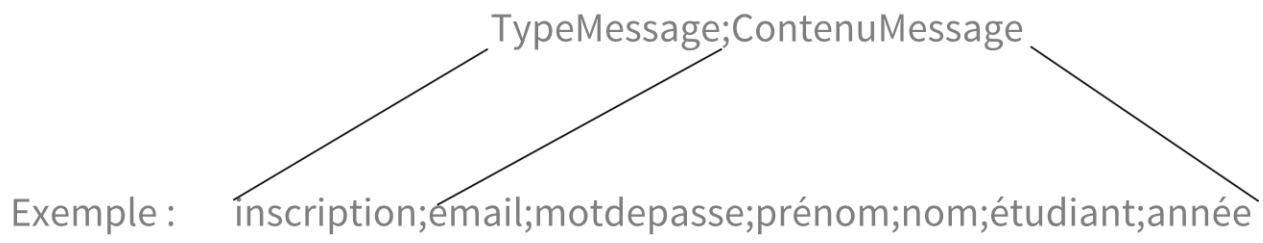
Le serveur a été codé de manière assez simple avec les sockets en TCP en Java. Le serveur est multi-client, il attend une connection avec un client, lance le client dans un thread où il traite la requête, renvoi la réponse et enfin ferme la connection avec le client ainsi que son thread. La fermeture de la connection et du thread client n'est pas obligatoire, mais nous avons choisi de le laisser tel quel dans un souci d'éviter certains possibles problèmes de communication avec une connectione serveur/client prolongé. On pourra admettre que le threadage est peut-être peu mis en avant puisque les connections sont courtes.

Itinéraire des requête :



2 Gestion des message

Les clients et le serveur communique par l'intermédiaire de message, prenant la forme d'une string.



Pour chaque requête différente est associé un identificateur (ici dans l'exemple, inscription), qui est séparé du reste et évalué par un switch. Le switch s'occupe ensuite d'effectuer l'action voulu en fonction du type du message.

Liste des différents messages servant à communiquer entre le client et le serveur :

Message du client vers le serveur (question)	Message du serveur vers le client (réponse)
inscription;email;mdp;prenom; nom;boolEtudiant;annee	inscription;boolResultat
authentification;email;mdp	inscription;prenom;nom;boolEtud;annee
matiere;nomMatiere;anneeEtudiant	matiere;intitule1;intitule2...
intitulecours;nom	intituleCours;nomCours1|format1|id1; nomCours2|format2|id2...
intituleqcm;nom	intituleqcm;intituleqcm;nomQcm1|id1;nomQcm2|id2...
qcm;idQcm	qcm;serialQcm
validerqcm;idQcm	validerqcm;boolResultat
qcmafaire;idUtilisateur	qcmafaire;nomQcm1|id1;nomQcm2|id2...
creationQcm;nomQcm;intituleQcm; serialQcm;idUser1|idUser2...	creationQcm;boolResultat
CreationCours;nom;intitule;Format;Path	creationcours;boolResultat
getusers;intitule	getusers;idUser1|NomUser1|Prenomuser1; idUser2|Nomuser2|nomuser2...

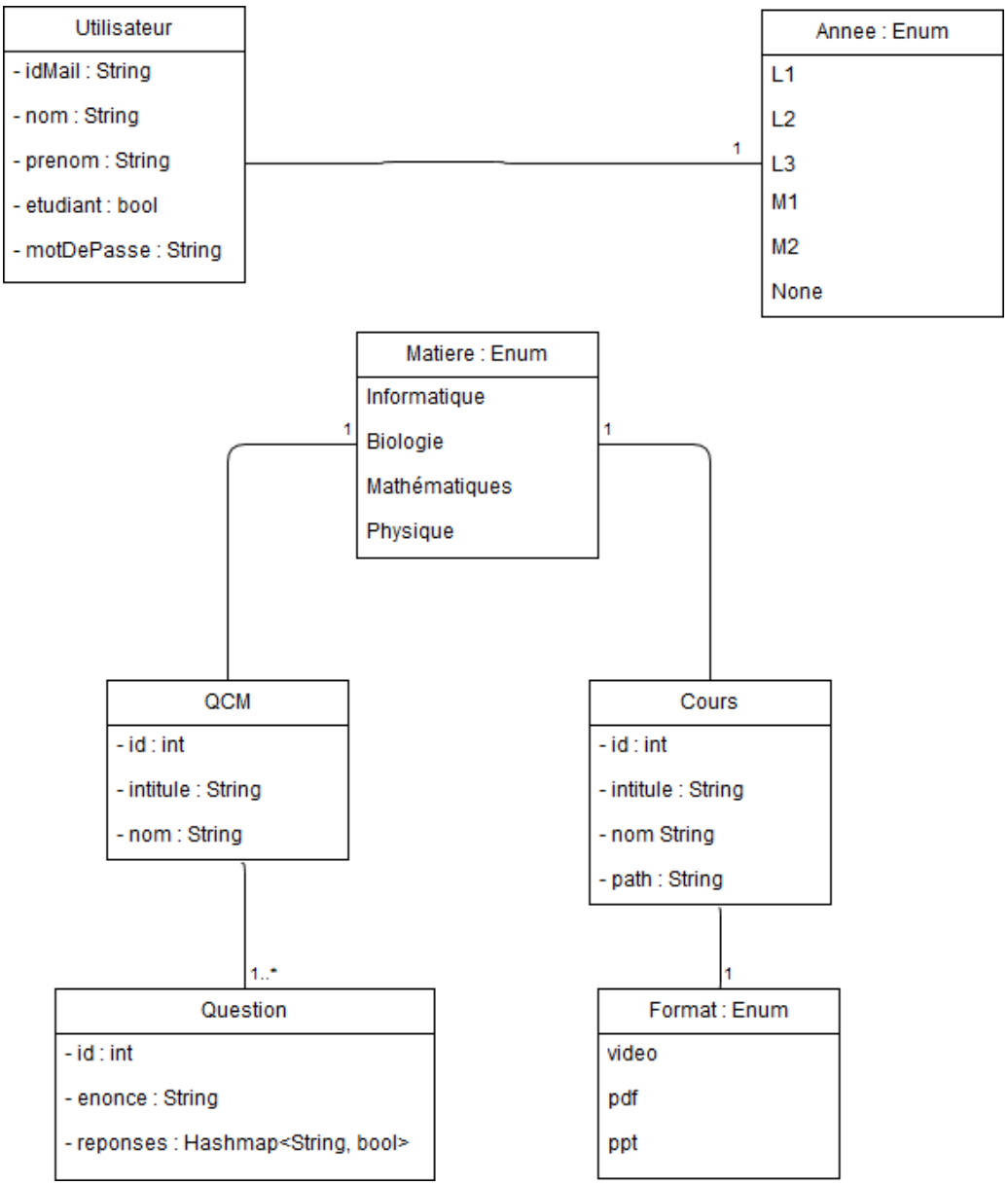
3 Problèmes rencontrés

Le serveur, bien que simple et fonctionnelle, nous a confronté à de nombreux problèmes concernant la communication avec un portable. Nous n'avons malheureusement pas pu communiquer à distance de l'android vers la machine windows (il y avait eu aussi des soucis de portforwarding, ainsi il fallait être connecté au même wifi pour pouvoir communiquer).
Au final, le serveur en socket n'était pas une idée judicieuse, mais par manque de temps nous n'avons ni pu perfectionner cet aspect, ni changer méthode de serveur (mais l'architecture du projet permet au moins de changer le serveur assez facilement pour de futures perspectives grâce à une séparation de la gestion des messages en plusieurs couches successives).

Base de Données

1 UML de la base

Voici l'UML de notre base de données



Les QCMs sont donc stockés sous la forme d'un code de sérialisation. Ainsi on voit une différence

avec l'UML de classe, car il n'y a pas de table pour les questions et les réponses. Une table dédiée aux intitulés est en revanche présente, permettant de garantir l'intégrité de leur nom puisqu'ils sont récupérés par la base et ne sont jamais modifiés.

2 Choix de la Technologie

La base de données présentée ci-dessus ne possédant pas d'objets complexes, une base de données relationnelle est amplement suffisante pour les besoins du projet. Nous avons donc choisi une base MySQL, car simple à déployer sur les différents OS, et utilisable avec l'outil phpMyAdmin, également utilisé lors de ce projet.

Les requêtes sur la base sont effectuées par le serveur à l'aide du package JDBC. Nous avons choisi de créer des méthodes générales permettant d'écrire la plupart des requêtes, c'est à dire toutes les requêtes de la forme :

Select * from Table where Attribut1 = AttributValue1 and ...;

Ainsi la méthode prend par exemple, pour l'utilisateur, la forme :

GetUtilisateursByAttributes(Vector<String> attribute, Vector<String> attributeValue) et renvoie la liste d'utilisateur tel que tous ses attributs cités dans le vector attribute soit égal a la valeur correspondante dans le vector attributeValue.

Les informations que le serveur doit envoyer sont ensuite extraites à l'aide des accesseurs en lecture et envoyées au client.

Conclusion

En conclusion, même si nous avons été pris par le temps et n'avons pas pu implémenter quelques fonctionnalités demandées, comme la date butoir pour les QCM, ainsi que la possibilité pour l'enseignant de connaître les résultats des élèves a qui il a pu envoyer un QCM, notre application implémente tout de même la plupart des fonctionnalités demandées de façon fonctionnelle.

Nous avons choisi des méthodes simples pour gérer les différentes fonctionnalités pour ainsi ne pas trop surcharger l'application. Même si pour certaines cela aurait peut-être mérité un peu plus de réflexion avant de se lancer dans l'implémentation.

En perspective, il serait possible pour de futures améliorations d'améliorer un peu le graphique, pour donner une personnalité propre à notre application, ainsi qu'implémenter les fonctionnalités restantes.
Les Cours vidéo et pdf étant gérer par lien Internet, on pourrait aussi imaginer une fonctionnalité permettant d'uploader son cours sur internet par l'application avant de le rendre disponible aux élèves.

Enfin, il était intéressant de s'essayer à la réalisation d'une application de A à Z, chose que nous n'avions jamais fait pour la plateforme mobile. Nous avons ainsi pu voir les soucis et demandes qu'entraînait ce support, et même si nos choix d'utilisation de socket n'étaient pas forcément les plus justifiés, nous avons pu au moins apprendre de cette erreur que des alternatives sont à considérer.