Projet Java - Sockets

Connect !

M1 STRI 2015/2016

Rapport Final

CHARLES BANQUET GUILLAUME BOULIC ROBIN DEGIRONDE

Table des matières

İntroduction : 4

İ) Conception 5

a. Modélisation 6

i. MCD 6

ii. MLD 7

b. Modélisation ULM 9

i. Diagramme de Classe 10

ii. Diagramme d'état 11

İİ) Bases de Données 12

a. Connection à la base de données : 12

b. Connexion à l'entité diplôme : 12

c. Connexion à l'entité compétence : 12

d. Connexion à l'entité utilisateur : 12

e. Connexion à l'entité recommander 12

f. Base de données MySQL: 12

İİİ) Rôles du Serveur TCP 13

a. Gestion du serveur 14

b. Utilisation du serveur 14

c. Codes de retour du serveur 14

d. Service Serveur Messagerie 15

İV) Rôles du Client TCP 16

a. Gestion du client 16

b. Utilisation du client 16

c. Codes d'envoi du client vers le serveur 16

i. LİSTE_USER 17

ii. DETAİL_USER 17

iii. CREER_COMPTE 17

iv. MODİF_İNFO 17

v. AJOUT_DİPLOME 17

vi. SUPP_DİPLOME 17

vii. AJOUT_COMP 18

viii. DEL_COMP 18

ix. CONNEXİON 18

x. LİST_COMP 18

xi. LİST_DİP 18

xii. ECRİRE_MAİLlidDest, Message 18
xiii. PASSER_EN_ECOUTE|0|20 19
xiv. LİST_USER_ECOUTE : 19
xv. PARLERlidUserlnum_port 19
xvi. Schéma pour la messagerie İnstantanée 20
xvii. RELEVER_MESSAGES|2 21
xviii. LİRE_MESSAGE|2lidMail 22
xix. addRecomendationlidUlidC 22
xx. delRecomendationlidUlidC 22
xxi. Diagrammes des actions possibles par le client 23
V) Conclusion 24
Vİ) Annexes 25
a. Exemples d'utilisations 25

İntroduction :

Dans le cadre de notre formation Système Télécommunication Réseaux İnformatiques nous devons réaliser un projet java.
Le but est mettre en œuvre un annuaire partagé permettant aux utilisateurs du système de connaître ses coordonnées (électroniques et téléphoniques), son année de diplomation s'il y a lieu ainsi que ses compétences. On appellera « compétence », un thème qui peut se réduire à un mot clé ("Java" ;)) ou à un ensemble de mots ("routage İSİS" ;)). Ce système suppose que chaque étudiant soit équipé d'une application lourde JAVA.

Ce projet se fera en 3 versions. Et 3 rapports seront à rendre.

Pour cette version nous avons gardé la même architecture que pour la première version avec un seul serveur. Et la même interface (interface texte).

Version 1 : Base

Ajout de contacts :

Chaque étudiant peut créer son compte et gérer ses informations attachées. Seules ses informations sont modifiables. On souhaite aussi permettre aux étudiants qui ont créé un compte de limiter la visibilité de toute ou partie de leurs informations pour toute ou partie des utilisateurs (admin, utilisateur, anonyme).

Utilisation courante :

Tous les utilisateurs du système peuvent récupérer la liste de tous les étudiants et interroger le système pour connaître les détails concernant tel ou tel étudiants. Notre application sera orientée client/serveur. Le client et le serveur auront la connaissance de protocoles d'émission et réception. Le serveur sera connecté à la base de données MySQL où la base sera actualisée à chaque requête du client vers le serveur. Notre application sera faite sous Eclipse et les diagrammes UML seront faits sur Modélio. Nous avons choisi de ne pas développer une interface graphique, l'application sera utilisée en mode commande sur un terminal.

Cette première version sera à rendre le 10 février 2016.

Version 2 : Messagerie İnstantanée

Dans cette étape, un utilisateur doit pouvoir dialoguer en privé avec un autre utilisateur du système. Deux modes de messagerie sont à prévoir :

- Conversations en direct :

Un utilisateur peut ouvrir une fenêtre de dialogue pour discuter avec un autre contact. Ces conversations privées ne devront pas transiter par le serveur. On utilise donc une communication "peer to peer".

- Messagerie

Dans ce cas de figure, un utilisateur peut laisser un message à l'attention d'un autre utilisateur. Cela peut être un cas de figure si un étudiant ne laisse pas accessible ses coordonnées. Ces conversations privées devront être hébergées sur une plateforme spécifique.

Cette première version sera à rendre le 2 mars 2016.

Version 3 : Challenge

Dans cette dernière version, on souhaite mettre en place un système de valorisation des compétences de chaque étudiant. Chaque étudiant peut à tout instant recommander un étudiant sur une compétence ou retirer sa recommandation. Ces recommandations fonctionnent sur le principe des "like" bien connu dans les réseaux sociaux.

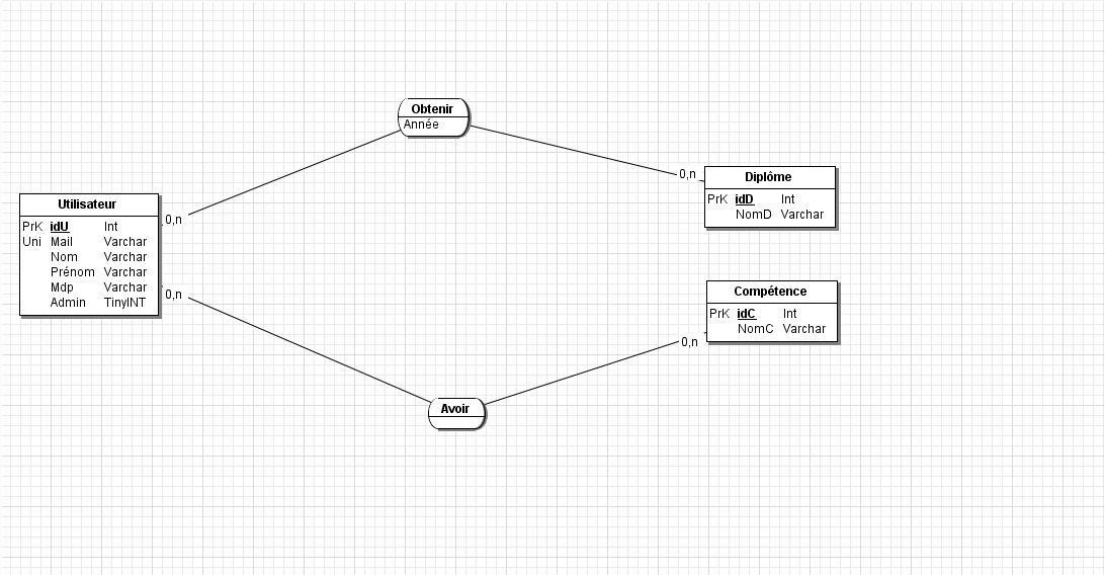
Cette dernière version est à rendre pour le 22/03/2016.

İ) Conception :

a. Modélisation:

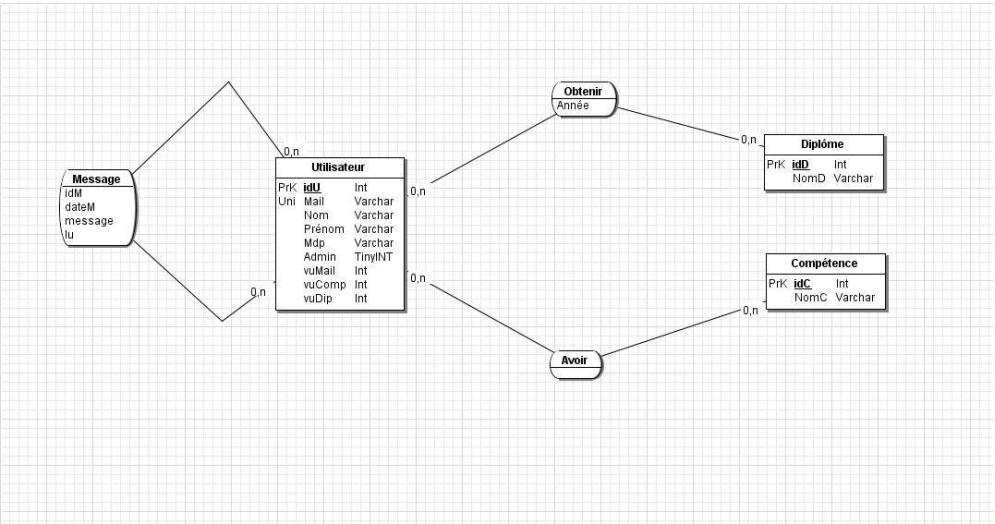
i. MCD :

Version 1 :

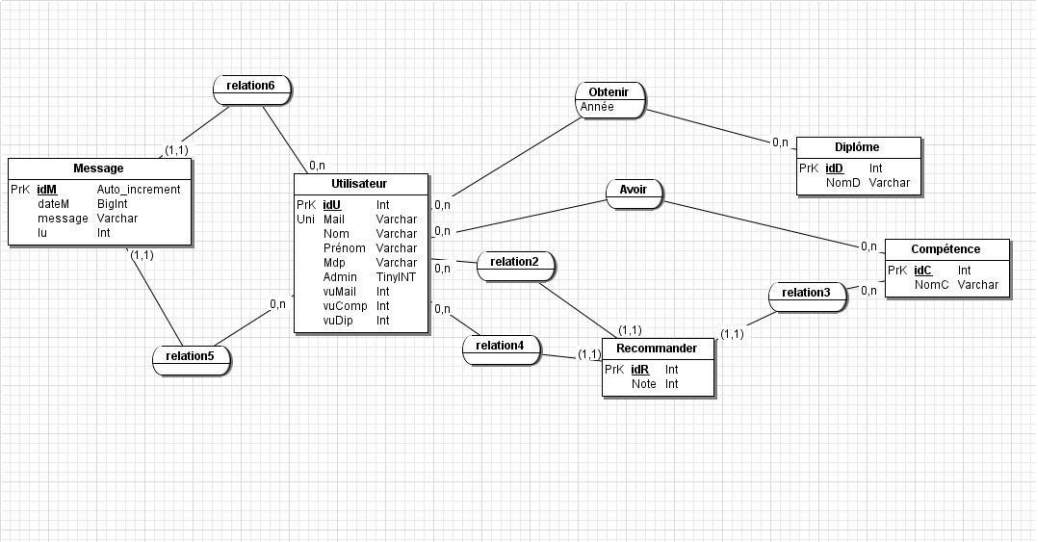


Pour cette première version nous avions besoin de 3 entités (Utilisateurs, Compétences et Diplôme). Cette modélisation est simple mais suffisante pour la version1. Ainsi un utilisateur peut obtenir un diplôme et avoir une compétence.

Version 2 :



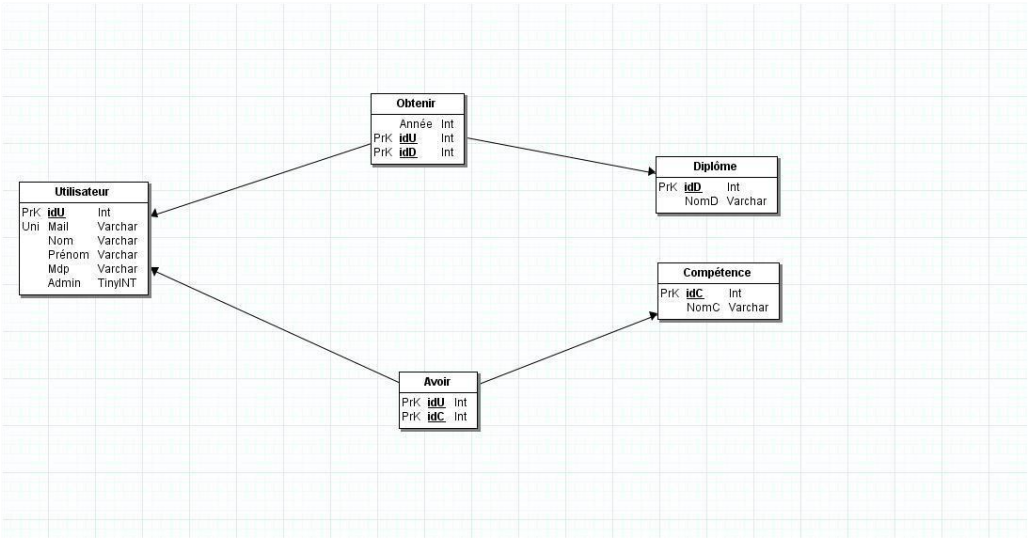
Pour la seconde version de la base de données nous avons rajouté une relation réflexive « message » qui a pour clé primaire idM et récupère les clés étrangères des Utilisateurs qui communiquent par mail (envoyeur et receveur). Ça permet à un utilisateur de pouvoir échanger par messagerie différée avec un autre utilisateur.
Nous avons aussi rajouté 3 attribues pour la l'entité Utilisateurs.
Version 3 :



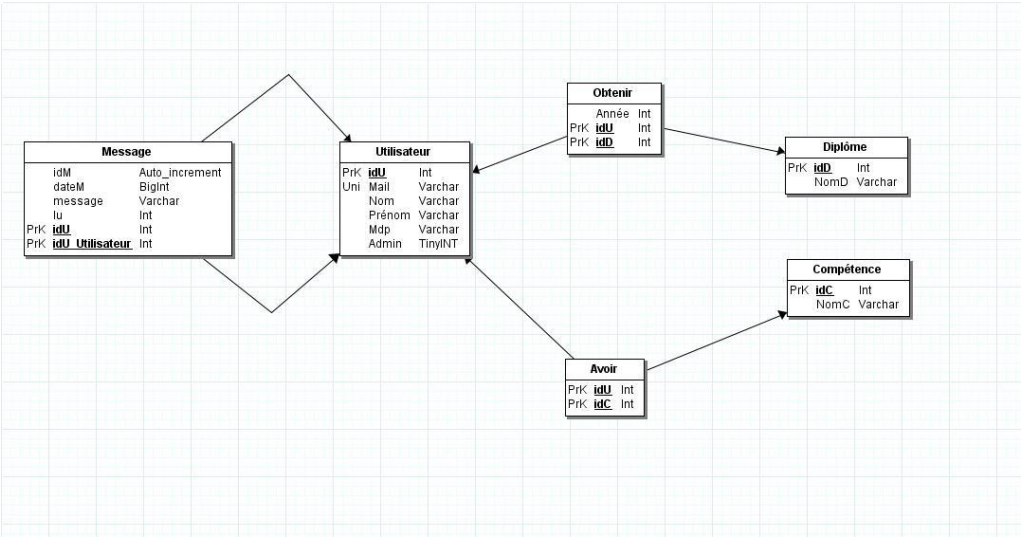
Pour cette troisième version du MCD nous avons mis en place une relation ternaire « Recommander » entre deux utilisateurs et une compétence. Ainsi pour qu'un utilisateur puisse recommander un autre utilisateur sur une compétence qu'il a. Il y a une contrainte sur cette relation c'est que l'utilisateur doit avoir la compétence en question.

ii. MLD :

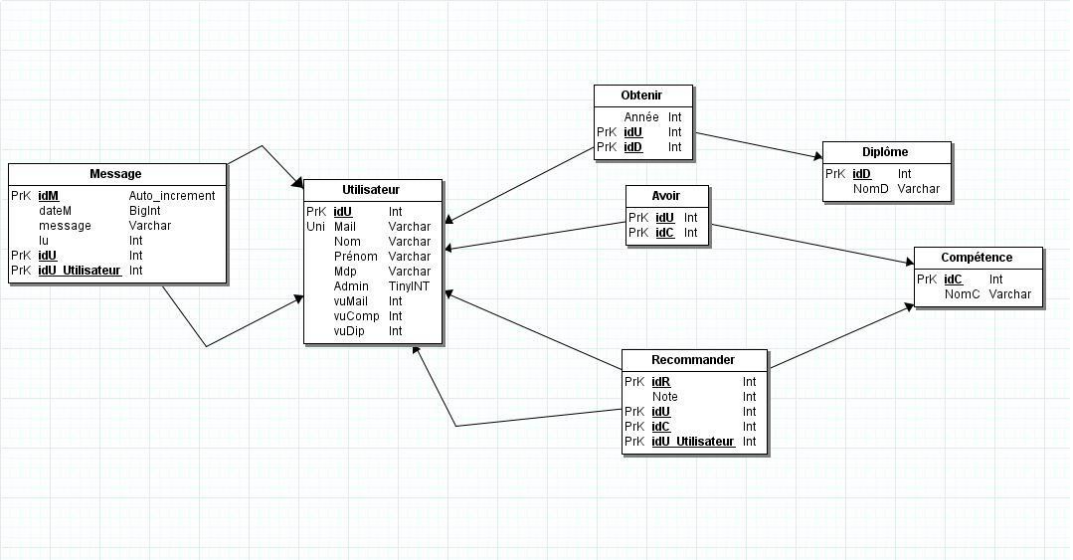
Version 1 :



Version 2 :

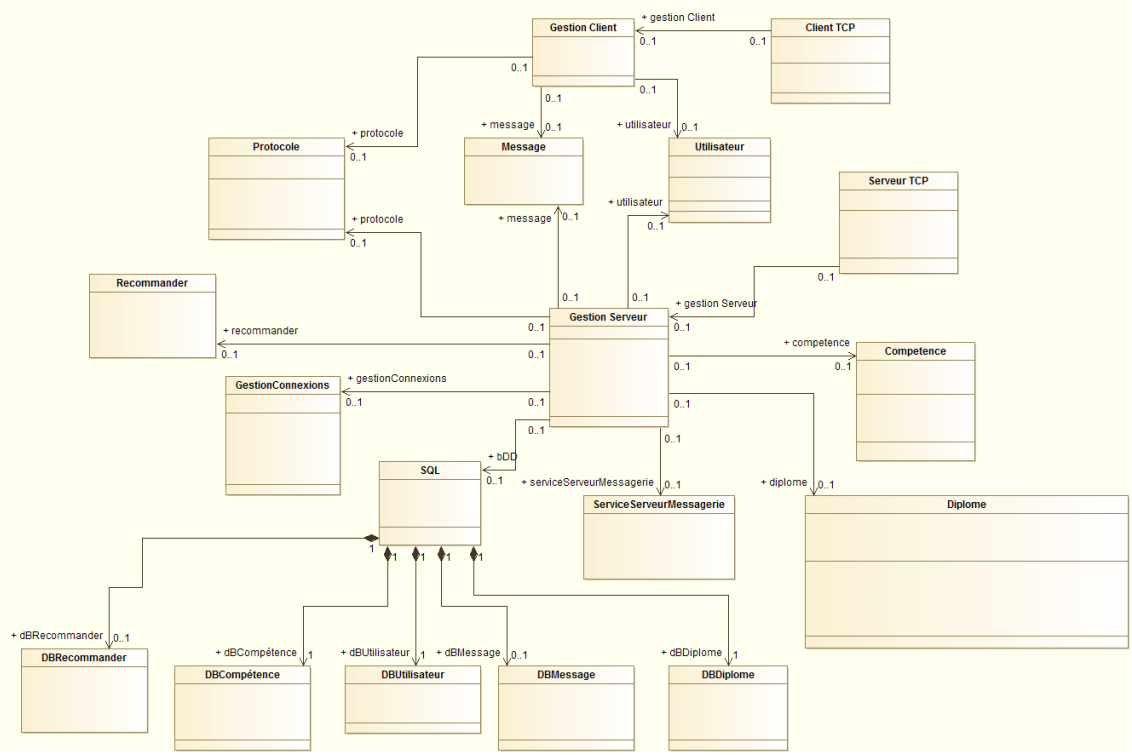


Version 3 :



b. Modélisation ULM :

i. Diagramme de Classe :



On peut voir dans le diagramme de classe, que nous avons implémenté plusieurs classes.

La classe protocole dont va se servir Gestion Serveur et Gestion Client contient toutes les actions que peut faire un client.

Les classes DBCompétences, DBUtilisateur, DBMessage, DBDiplôme et DBRecommandation sont des agrégations de la classe SQL. Ces classes permettent la gestion et la mise à jour de la base de données pour chaque entité. La classe MySQL permet la connexion à la base de données.

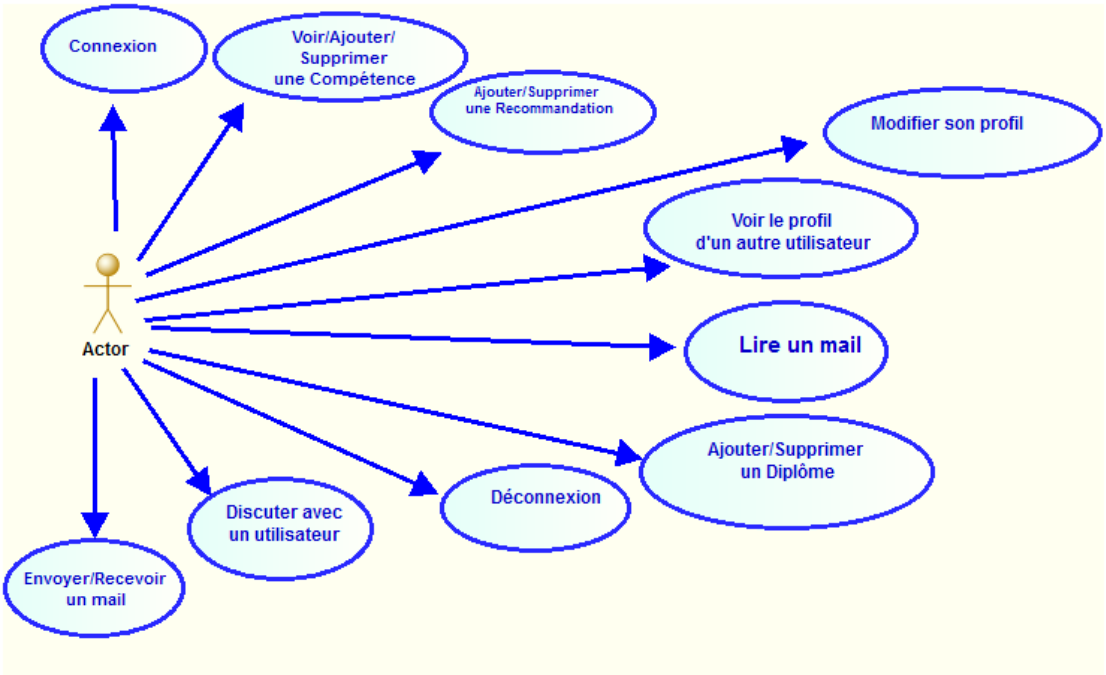
La classe gestion Serveur gère à utilise plusieurs classes dont MySQL et Protocole mais aussi Compétences, Diplôme, Recommandation, GestionConnexion, ServiceServeurMessagerie, Message et Utilisateur.

La classe gestion Client utilise comme on l'a vu la classe Protocole, mais aussi la classe message et utilisateur. Elle est utilisée par la classe ClientTCP.

Dans la classe gestion client nous avons rajouté les différentes

opérations pour que le client puisse gérer l'écrire la lecture et la réception de messages. Puisse voir aussi les utilisateurs connectés et parler ou se mettre en écoute. Nous avons fait de même côté serveur.
Nous avons implémenté une classe message qui aura besoin d'une nouvelle classe DBMessage qui découle de la classe MySQL.
Nous avons aussi implémenté une nouvelle classe côté serveur qui est un service qui gère la messagerie côté serveur.

ii. Diagramme d'état :



iii) Bases de Données :

a. Connection à la base de données :

La classe MySQL se connectera à la base de données en utilisant comme paramètres l'url de la base, l'Username et le mot de passe. Une fois qu'on est connecté à la base on va envoyer les requêtes pour créer la base et donc chaque table. Il y aura aussi des requêtes pour créer la base. Chaque classe aura besoin de la classe MySQL pour se connecter à la base et pouvoir mettre à jour des informations concernant des utilisateurs.

b. Connexion à l'entité diplôme :

Pour se connecter à l'entité diplôme, on se connectera avec MySQL. On aura besoin des classes diplôme et utilisateur pour mettre à jour la base de données sur chaque action qui concerne l'entité diplôme. Il y a deux méthodes pour la classe diplôme la première c'est quand l'utilisateur donne une année d'obtention de diplôme et la seconde quand il ne la donne pas dans ce cas-là on mettra 0 pour l'année, c'est-à-dire vide. Les résultats seront renvoyés dans la classe compétence.

c. Connexion à l'entité compétence :

Pour se connecter à l'entité compétence on va avoir besoin des classes utilisateur, compétence et diplôme. À chaque action qu'on fera et qui concerne l'entité compétence on se connectera en MySQL à la base. On renverra les résultats dans la classe compétence.

d. Connexion à l'entité utilisateur :

Dans un premier temps la classe de DBUtilisateur va se connecter à la base de données puis elle mettre à jour les informations concernant l'utilisateur. À chaque opération la classe va se connecter à la base et faire des opérations de lecture et écriture en fonction de la requête transmise.

e. Connexion à l'entité recommander

Pour se connecter à Recommander on a besoin des classes utilisateur et compétence. À chaque action qui se fera l'entité Recommander va se connecter à la base de donnée MySQL. Les résultats seront envoyés dans la relation recommander.

f. Base de données MySQL:

Nous avons choisi de mettre en place et d'implémenter notre sur phpMyAdmin où on peut se connecter en localhost avec le numéro de port. Le

login et le mot de passe.

Avoir (#idU, #idC) : Cette relation contient deux clés primaires étrangères qui sont l'identifiant d'un utilisateur et l'identifiant d'une compétence. Elle permet de relier l'entité compétence à l'entité Utilisateur. Et de pouvoir ajouter ou supprimer une compétence à un utilisateur.

Competences (idC, NomC) : Cette relation possède l'identifiant d'une compétence comme clé primaire et pour chaque id le nom d'une compétence est associé.

Diplome (idD, NomD) : Cette relation possède l'identifiant d'un diplôme comme clé primaire et pour chaque id le nom du diplôme est associé.

Message (idM, #idEnvoyeur, #idDestinataire, dateM, message, lu) : Cette relation possède deux clés étrangère et sa clé primaire étant l'identifiant du message, les clés étrangères viennent de l'entité Utilisateur c'est une relation réflexive. Cette relation fait qu'un utilisateur peut envoyer aucun ou plusieurs messages à un autre utilisateur (y compris lui-même).

Obtenir (#idU, #idD, Année) : Cette relation possède deux clés étrangères qui sont l'identifiant d'un utilisateur et l'identifiant d'un diplôme. Cette relation permet d'attribuer ou enlever un diplôme à un utilisateur.

Recommander (#idCompetence, #idConseilleur, #idRecommande, note) : Cette relation permet de recommander une compétence d'un utilisateur par un autre utilisateur. Elle a donc 3 clés étrangères c'est une relation ternaire. Elle va prendre idConseilleur et idRecommande sont deux clés étrangères de l'entité Utilisateur et ensuite l'idCompétence de l'entité compétence.
Il y a une vérification qui est effectuer pour qu'un utilisateur possède bien la compétence.

Utilisateur (idU, Mail, Nom, Prénom, Mdp, Admin, VuMail, VuComp, VuDip) : Cette entité est celle d'un utilisateur avec ses paramètres, sa clé primaire est un identifiant, il y a aussi la vue sur les mails, compétences et diplômes.

İİİ) Rôles du Serveur TCP:

Nous avons choisi un seul serveur qui gèrera à la fois les connections et les requêtes des clients. Ce serveur sera relié à la base de données pour avoir une gestion des données utilisateurs.

Nous avons choisi d'utiliser TCP car nous avions déjà travaillé dessus sur les TPs précédents et donc savons travailler sur TCP contrairement à UDP que nous n'avons pas pu trop manipuler. Le choix de TCP par rapport s'est par facilité pour nous. Cependant, UDP aurait peut-être été mieux adapté. Nous avons choisi d'utiliser des Thread pour faire du multi-client. Nous avons choisi d'utiliser des sockets pour pouvoir communiquer entre les clients et le serveur à l'aide d'une adresse IP et d'un port, ces sockets permettront de gérer des flux entrant et sortant afin d'assurer une communication entre les deux (le client et le serveur), soit de manière fiable à l'aide du protocole TCP/IP.
Pour la messagerie instantanée le serveur ne gère pas les transmissions de messages. Il dit à utilisateur si un tel ou un tel est en discussion ou pas. Si l'utilisateur est disponible alors on peut se connecter avec lui sur son port d'écoute. Ainsi le serveur notifiera aux autres utilisateurs que ce dernier n'est pas disponible.

a. Gestion du serveur :

Le serveur TCP interagie directement avec le client par des messages de retour lorsque le client envoi une requête. La gestion du serveur exécutera tous les messages que le client va envoyer et les analysera et à renvoyer un code de retour en fonction du message envoyé. Le serveur va d'abord analyser la requête que lui demande le client, s'il n'y en a pas il renvoi un message d'erreur. Le serveur va mettre à jour la base de données lorsqu'il reçoit une nouvelle requête utilisateur. Ou alors il ira chercher l'information.

Le serveur reçoit une requête d'un client pour écrire un mail à un autre utilisateur, lire un mail qu'il a reçu d'un autre utilisateur et relever ses messages. Le serveur reçoit le socket de l'utilisateur et va se connecter à la base de données utilisateur.

Le serveur reçoit une requête pour qu'un client puisse discuter avec un autre utilisateur il va regarder les utilisateurs qui sont en écoute et les lister.

b. Utilisation du serveur :

Le serveur TCP a trois phases une partie où il va se connecter et créer un socket. Le serveur sera alors en attente de connexion avec un socket d'écoute. Ensuite une autre où il va recevoir les connexions de clients en utilisant sa partie de gestion pour analyser les requêtes entrantes avec un socket de service jusqu'à qu'il soit fermé.

c. Codes de retour du serveur :

Le serveur possède 2 requêtes qu'il envoi au client.
Les requêtes sont faites de la façon suivante :

- 200|OK :
signifie que la requête envoyer est bonne.
Le client lui de son côté ne verra que OK.

- 400|Erreur Protocole
signifie que le client s'est trompé de requêtes, que la syntaxe n'existe pas, que le client a fait une erreur de syntaxe, que la requête DB s'est mal passé, que le client est déjà connecté ailleurs ou alors qu'il n'y a pas le bon nombre de paramètres. Le serveur ira d'abord se connecter à l'entité en question sur la base de données et entrera la commande MySQL en question pour savoir si l'utilisateur s'est trompé ou pas. Ainsi la base de données lui retournera le type d'erreur en fonction de l'entité où le serveur à fait la requête.

- 200|ip|port :
signifie que la requête s'est bien passé et que le serveur renvoi à un utilisateur le port pour pouvoir discuter avec un autre utilisateur.

d. Service Serveur Messagerie :

Cette nouvelle classe récupère un socket limité à une instanciation pour ne pas avoir plusieurs utilisateurs connectés sur un même utilisateur et le livre à un client qui sera en écoute prêt à être demander en conversation directe.
Un utilisateur a son socket d'écoute fermé lorsqu'il en train de parler avec un autre utilisateur. Ainsi lorsqu'on demande la liste des utilisateurs connectés ceux qui sont en train de parler ont leur socket d'écoute fermé et donc le serveur ne notifie pas aux autres utilisateurs leur socket.
C'est une version mis à jour du ServiceServeur de la première qui livre le socket au client.

IV) Rôles du Client TCP :

a. Gestion du client :

Le client va se connecter au serveur sur la machine sur le port 12345. Il y a création d'un socket, création d'un flux d'entrée et flux de sorti où les messages, les requêtes et les réponses seront envoyés/reçus.

Pour la messagerie différée, le client va écrire et envoyer un mail à un autre utilisateur. Pour ça il lui faudra seulement l'id de l'utilisateur et le corps du message. Cette requête sera envoyée au serveur. Qui mettra à jour la base de données.
Un identifiant mail sera créer ce qui permettra à l'utilisateur qui a reçu le message de retrouver le mail et le lire.
Pour la messagerie instantanée, le client peut voir qui est connecté. Lorsque le client est connecté avec un autre utilisateur, les autres utilisateurs ne peuvent entrer en communication directe avec eux. Les sockets de ces clients passent en écoute et ils peuvent échanger. Pour quitter la conversation les utilisateurs devront faire « q » (quit), les ports des utilisateurs sont alors supprimés.

b. Utilisation du client :

Le classe client TCP va servir de la classe GestionClientTCP et utilisateur pour pouvoir envoyer des requêtes. Elle aura besoin de la classe protocole pour connaitre toutes les requêtes qu'il peut envoyer au serveur
Pour la messagerie instantanée le client une fois connecter avec un autre utilisateur en ouvrant son socket d'écoute il sera en constante discussion tant qu'il n'a pas fait « q » pour quitter la conversation. Il pourra envoyer et recevoir les messages avec l'utilisateur qui est connecté avec lui.

c. Codes d'envoi du client vers le serveur :

Le client possède des requêtes qu'il enverra vers le serveur TCP pour

se connecter avec un autre utilisateur ou pour lui envoyer un mail. Toutes les requêtes seront disposées de la façon suivante :
COMMANDE | LİSTE_PARAM

İl y a donc minimum 2 paramètres le premier étant la commande que va effectuer le client et la seconde est le paramètre dont il a besoin pour effectuer la requête. Chaque paramètre est séparé par un pipe. Si un de ces paramètres est incorrect ou n'existe pas le serveur renverra alors un message d'erreur.
Voici la liste des commandes utilisé par le client avec ses paramètres :

Pour connaître la liste les commandes on tape : « ? ».

i. LİSTE_USER :
Permet d'obtenir la liste des utilisateurs présent. Pour cette commande on aura besoin comme paramètre pour cette commande de l'İD utilisateur.

ii. DETAİL_USER :
Permet d'obtenir le détail d'un utilisateur. Nous avons donc besoin pour cette commande d'avoir 2 paramètres, l'identifiant du client qui demande et l'identifiant du client cible.

iii. CREER_COMPTE :
Cette commande est faite pour créer un nouveau compte utilisateur. Cette commande a besoin de tous les paramètres obligatoires qui ne peuvent pas être nuls pour que le serveur puisse les insérer et mettre à jour la base de données. Les paramètres obligatoires sont l'id de l'utilisateur, son nom, son prénom, le mot de passe et le mail.

iv. MODİF_İNFO :
Cette commande permet à l'utilisateur de modifier ses informations. La requête envoyée n'est valable que pour un paramètre. 1 requête envoyée = 1 paramètre modifié. İl faut donc le nom du paramètre qu'on veut modifier et la nouvelle valeur que l'utilisateur souhaite mettre. Le serveur mettra alors à jour la base de données.

v. AJOUT_DİPLOME :
Permet à l'utilisateur d'ajouter un diplôme qu'il a obtenu en spécifiant l'année de l'obtention du diplôme. İl y aura donc 2 paramètres. Le serveur mettra alors à jour la base de données.

vi. SUPP_DİPLOME :
Permet à l'utilisateur de supprimer un diplôme. İl n'y a qu'un seul paramètre qui est l'identifiant du diplôme. Le serveur mettra alors à jour

la base de données.

vii. AJOUT_COMP :
Permet à l'utilisateur d'ajouter une ou plusieurs compétences il y a donc minimum un paramètre. Chaque une compétence = un paramètre. Le serveur mettra alors à jour la base de données.

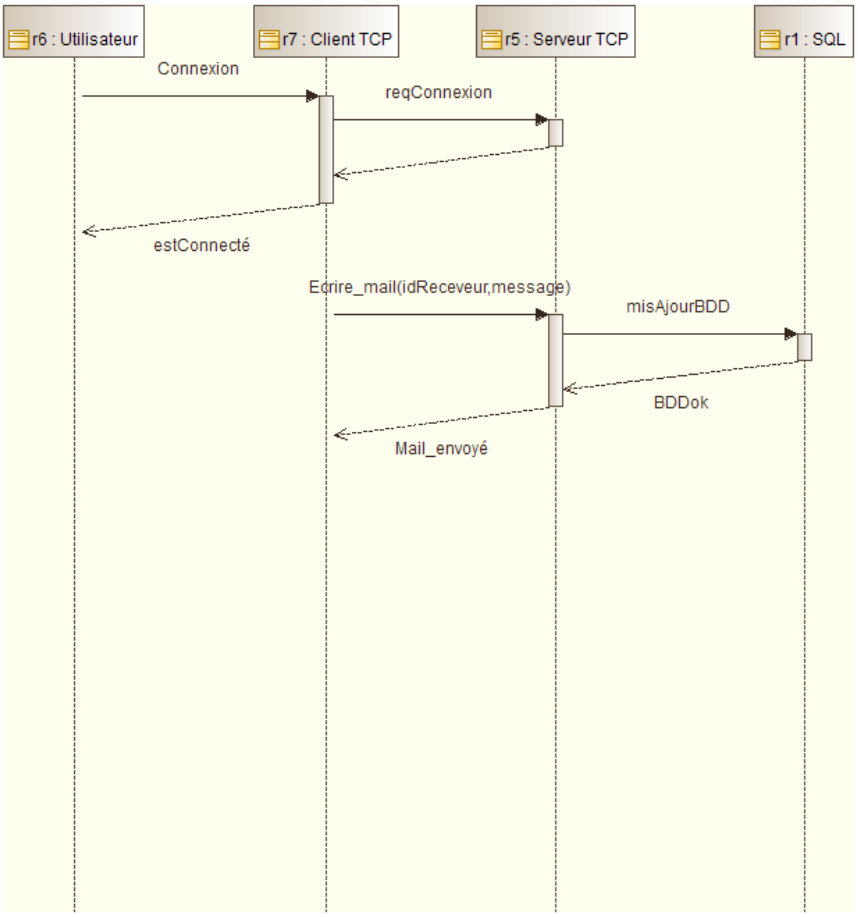
viii. DEL_COMP :
Permet à l'utilisateur de supprimer une compétence. Cette requête contient donc 1 paramètre. Avec l'identifiant de la compétence. Le serveur mettra alors à jour la base de données.

ix. CONNEXION :
L'utilisateur a besoin de 2 paramètre pour cette requête que sont son identifiant et son mot de passe.

x. LIST_COMP :
Permet à l'utilisateur d'afficher la liste des compétences disponibles.

xi. LIST_DIP :
Permet à l'utilisateur d'afficher la liste des diplômes disponible.

xii. ECRIRE_MAILIidDest, Message :
Le serveur va recevoir cette requête pour qu'un utilisateur puisse envoyer un mail à un autre utilisateur. Le serveur mettra à jour la base de données pour que l'utilisateur qui a reçu le mail puisse le voir et le lire.

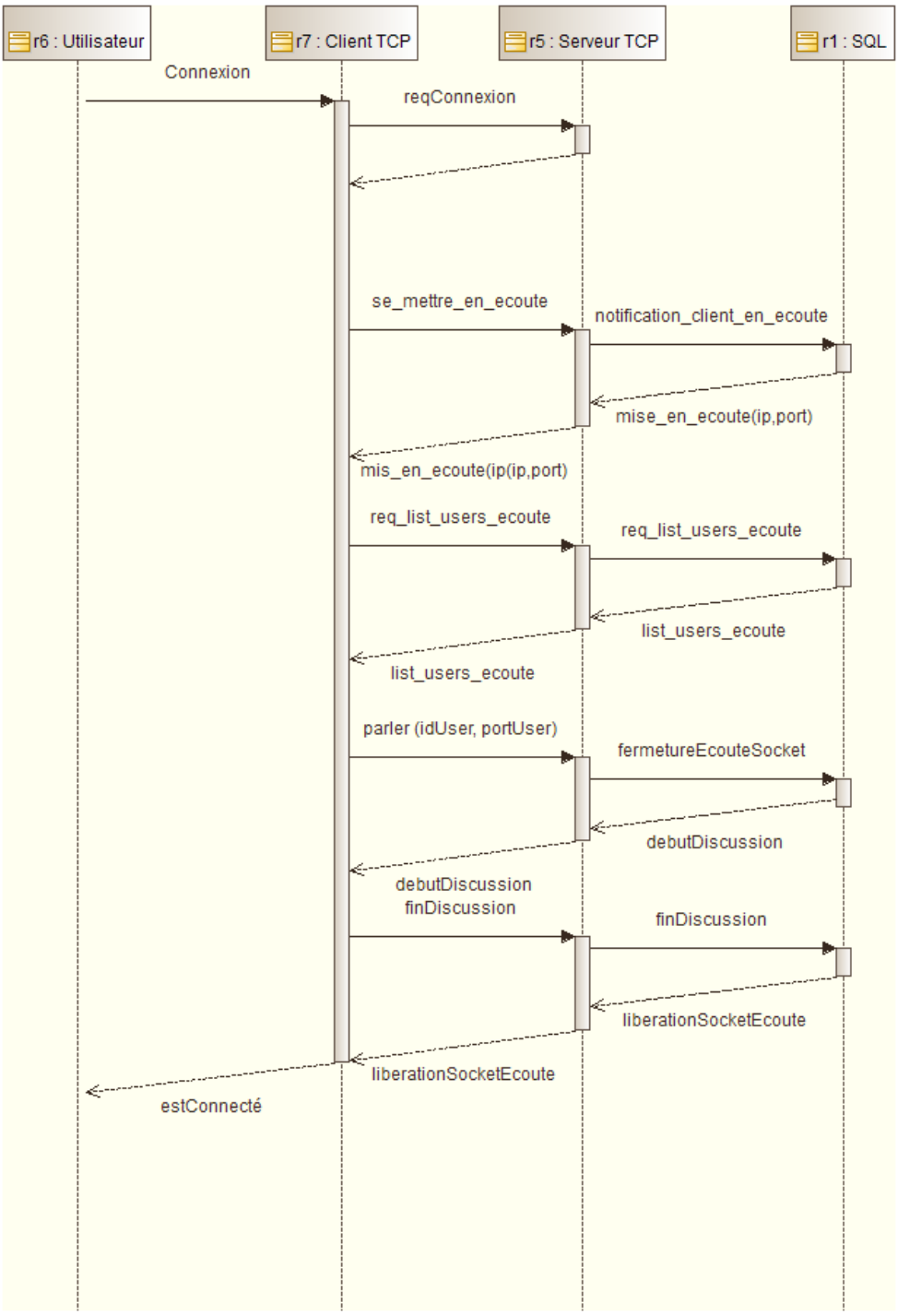


xiii. PASSER_EN_ECOUTE|0|20 :
Cette requête permet à un utilisateur de notifier au serveur qu'il a activé son socket d'écoute.

xiv. LIST_USER_ECOUTE :
Cette requête envoyée au serveur permet de savoir quels sont les utilisateurs connectés qui sont en écoute

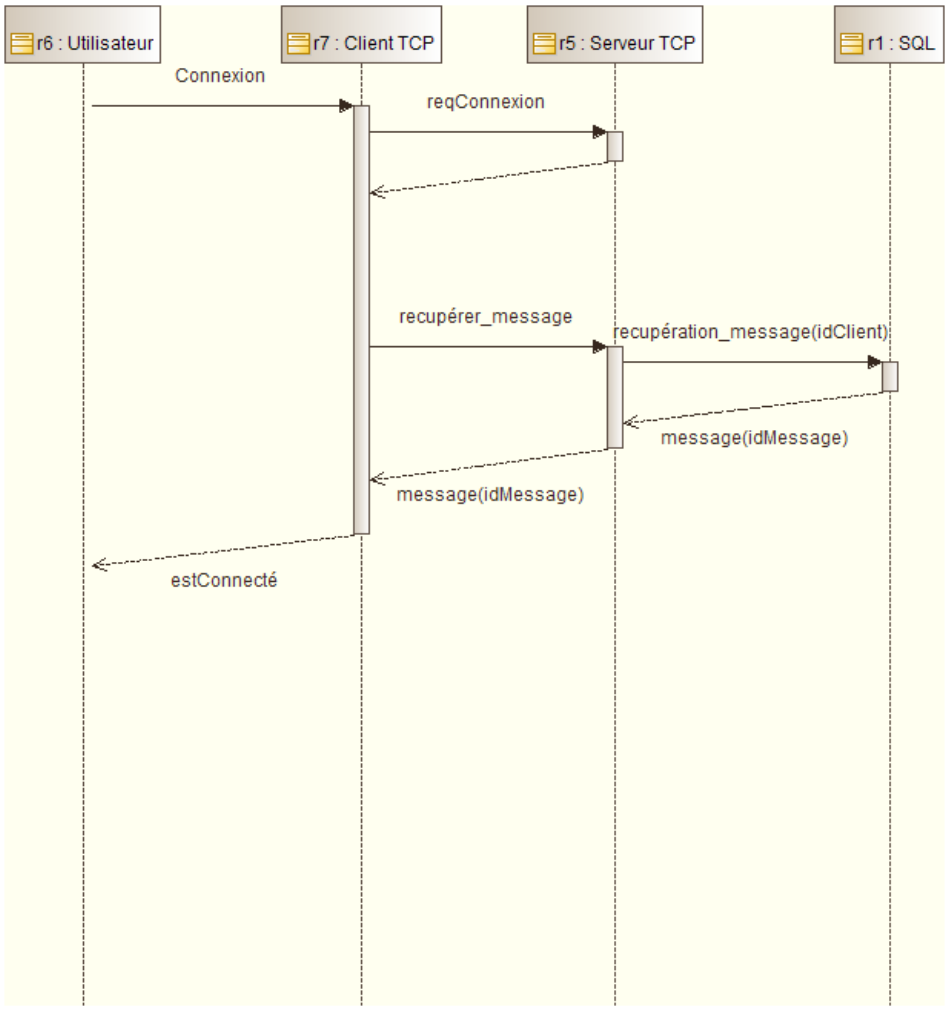
xv. PARLER|idUser|num_port :
Une fois qu'un utilisateur à notifier à l'utilisateur qu'il est en écoute et qu'il a eu la liste des users en écoute alors l'utilisateur enverra cette requête et sera en discussion avec l'utilisateur choisi.

xvi. Schéma pour la messagerie Instantanée :



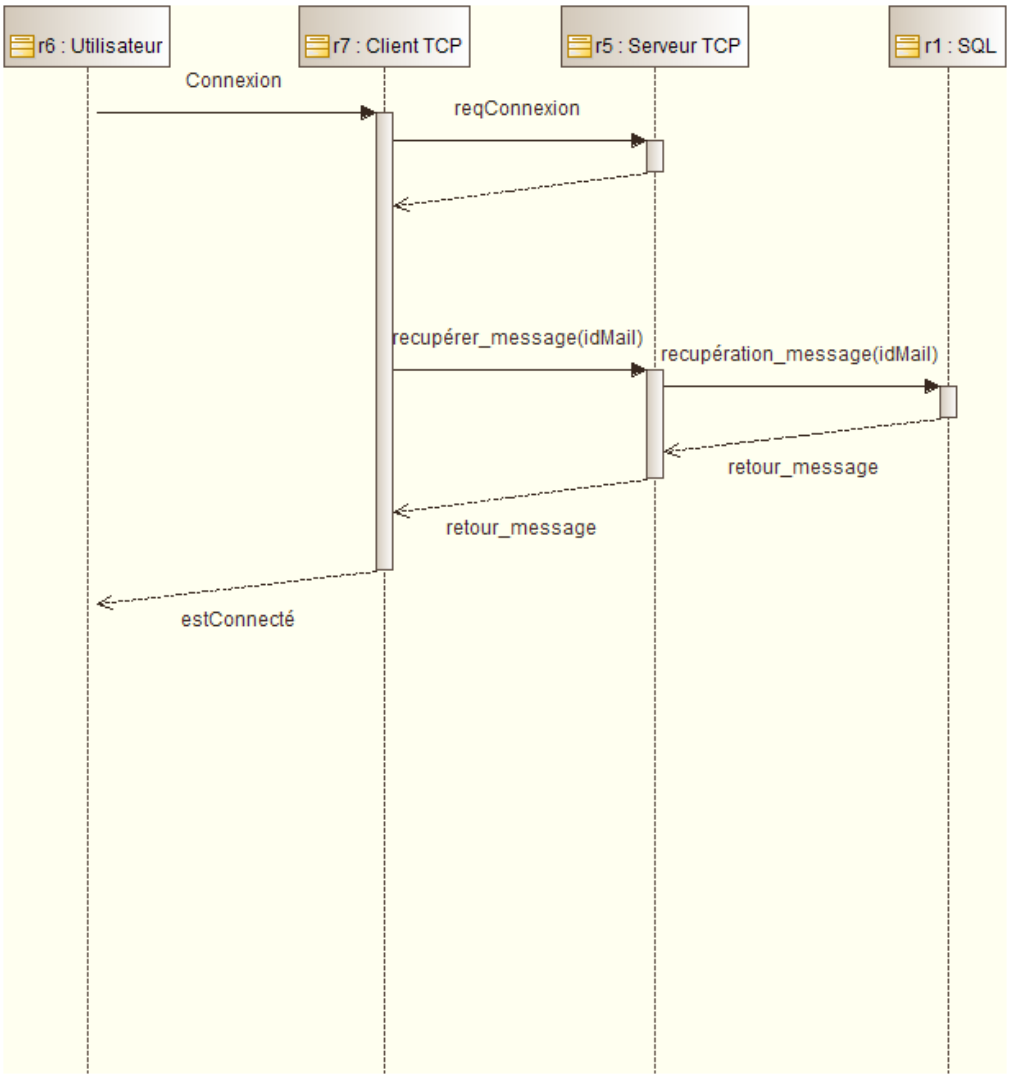
xvii. RELEVER_MESSAGESI2 :

Cette requête envoyée au serveur permet de voir les mails qu'un utilisateur a pu recevoir.



xviii. LIRE_MESSAGEI2IidMail :

Cette requête envoyer au serveur permet de lire les mails que l'utilisateur a reçu. Le client envoi une requête au serveur avec l'identifiant du mail. Le serveur va donc relever l'identifiant du mail et envoyer le mail en question que l'utilisateur a reçu.



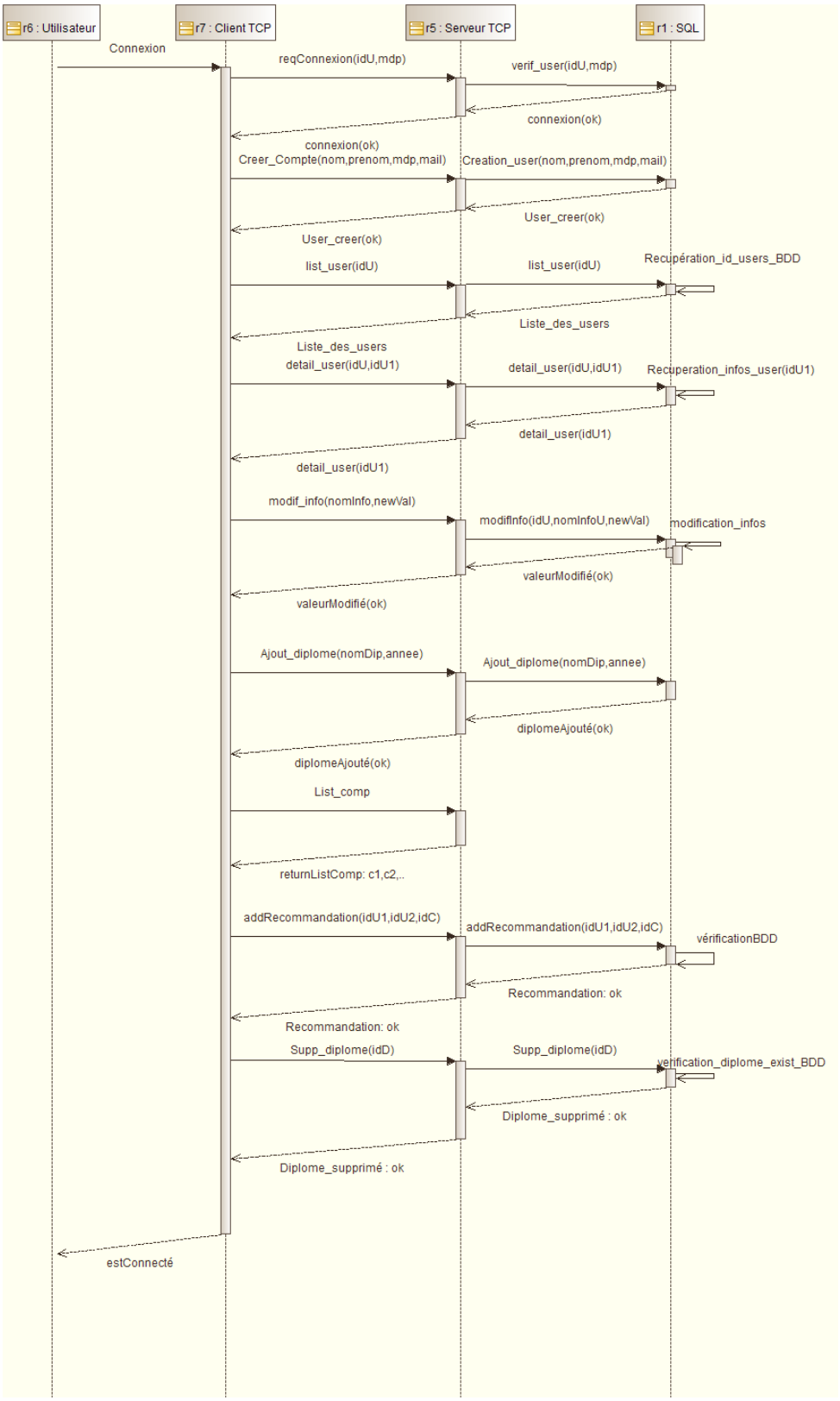
xix. addRecomendationIidUIidC :

Cette requête envoyée au serveur permet de recommander une compétence d'un utilisateur, où on met son id en paramètre, ainsi que l'id de sa compétence.

xx. delRecomendationIidUIidC :

À l'inverse de la requête précédente celle-ci supprime la recommandation que l'on a fait sur un utilisateur, elle a besoin des mêmes paramètres que addRecommandation.

xxi. Diagrammes des actions possibles par le client :



V) Conclusion :

La première partie du projet Java nous a permis de travailler en équipe, d'utiliser nos connaissances et de se répartir les tâches. Ce projet est dans la continuité de ce que nous faisions en TP.
Lors de la seconde version nous avons continué à nous partager le travail comme pour la première version. Nous sommes mis d'accord sur la conception de la version 2. Nous avons cette fois-ci tenu le cahier des charges sans rajouté d'éléments en plus, nous avons donc pu bien prendre en compte les besoins de cette version, ce qui nous a permis de pas être en retard sur l'avancement de la version.
La troisième version nous nous sommes mis d'accord sur comment implémenter les recommandations. Ensuite nous l'avons mis en place. Cette partie n'a pas pris beaucoup de temps à implémenter.
En définitive pour ce projet effectué en trinôme, nous avons compris l'importance de respecter le cahier des charges, l'importance de communiquer et de se voir régulièrement pour faire le point sur l'avancement de l'avancement du projet. Nous avons su nous répartir les tâches.
Ce projet nous a permis de solidifier nos connaissances en JAVA et en acquérir de nouvelles.

VI) Annexes :

a. Exemples d'utilisations :

```
mainClient [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (22 mars 2016 13:19:25)
Connecte sur :Socket[addr=/127.0.0.1,port=12345,localport=53301]
faites ? pour de l'aide
Commande: ?
LIST_USERS récuperer la liste des utilisateurs
DETAIL_USER récupérer les détails d'un utilisateur
CREER_COMPTE créer un compte
MODIF_INFO modifier vos informations
AJOUT_DIPLOME Ajouter un diplome
SUPP_DIPLOME suprimer un diplome
AJOUT_COMP Ajouter une compétence
DEL_COMP Suprimer une compétence
CONNECTION Se connecter
LIST_COMP lister toutes les compétences dispo
LIST_DIP lister l'ensemble des diplomes dispo
ECRIRE_MAIL Envoyer un mail à un utilisateur
RELEVER_MESSAGES Relever les messages non lus
LIRE_MESSAGE Lire un message en detail
LIST_USER_ECOUTE Lister les utilisateurs qui sont connectés
PASSER_EN_ECOUTE Passer en mode ecoute
PARLER Parler avec un utilisateur connecté
faites ? pour de l'aide
Commande:
```

```
Commande: ?
LIST_USERS récuperer la liste des utilisateurs
DETAIL_USER récupérer les détails d'un utilisateur
CREER_COMPTE créer un compte
MODIF_INFO modifier vos informations
AJOUT_DIPLOME Ajouter un diplome
SUPP_DIPLOME suprimer un diplome
AJOUT_COMP Ajouter une compétence
DEL_COMP Suprimer une compétence
CONNECTION Se connecter
LIST_COMP lister toutes les compétences dispo
LIST_DIP lister l'ensemble des diplomes dispo
faites ? pour de l'aide
Commande: CREER_COMPTE
Nom:nom
Prenom:prenom
Mailmailmail
Mot de passemdp
Niveau visibilite mail:0
Niveau visibilite competences:0
Niveau visibiité diplomes0
200
OK
faites ? pour de l'aide
Commande: CONNECTION
pseudo:mailmail
mdp:mdp
Connexion OK
Bonjour NOM prenom

faites ? pour de l'aide
Commande: LIST_USERS
200
1 dumbuldore albus ad@baguette.com
2 nom prenom hp@sorcier.com
3 nomm prenomm mail
4 n1 p1 mail1
5 nom prenom mailmail
faites ? pour de l'aide
Commande: DETAIL_USER
Id user:4
Id: 4, Nom: N1, Prenom:p1, Mail:Caché
Diplomes: 0 Caché
Competences: 0 Caché
faites ? pour de l'aide
Commande: DETAIL_USER
Id user:5
Id: 5, Nom: NOM, Prenom:prenom, Mail:mailmail
faites ? pour de l'aide
Commande: MODIF_INFO
id:5
Nom:nom
Prenom:pronom
Mot de passe:mdpmdpmdpmdpmdp
Niveau visibilite mail:0
```