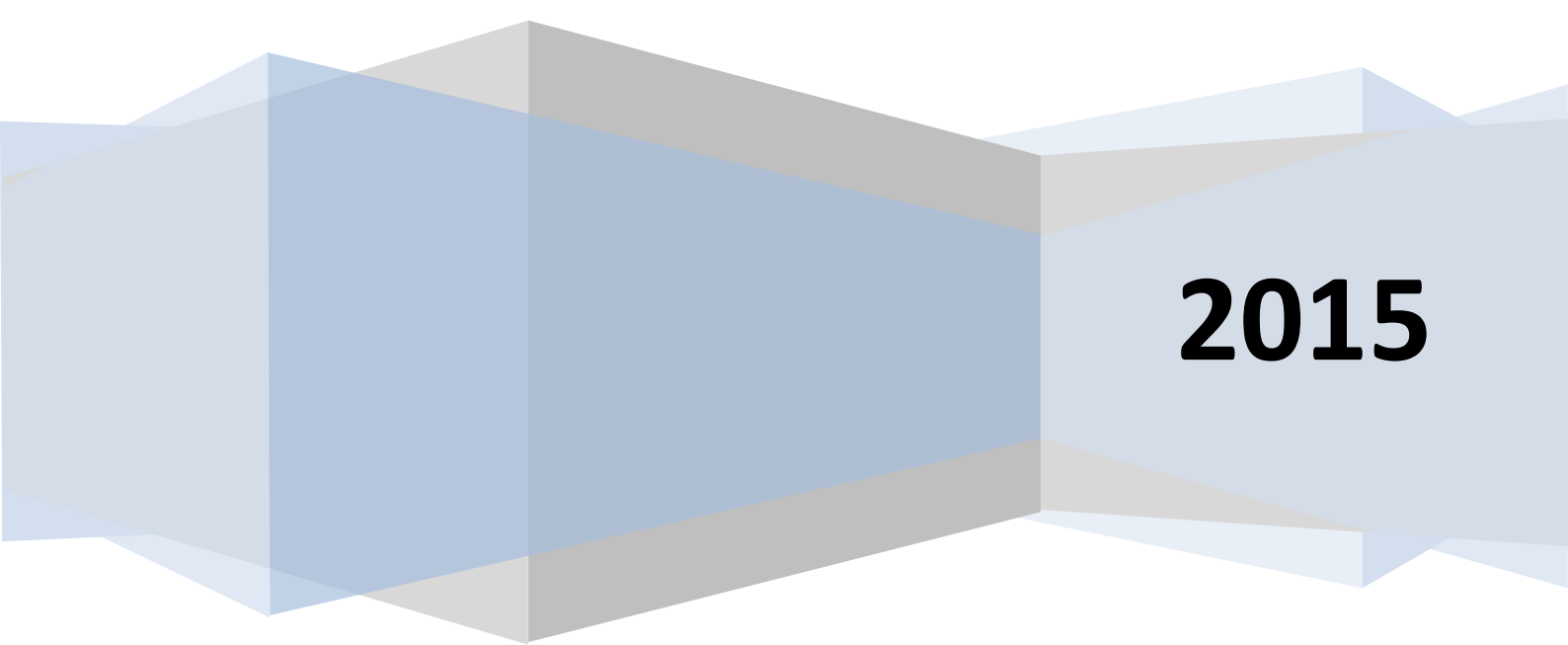


Gestion d'une école

**FABRE Maxime
FOUCHE Alexis
LEPOT Florian**



2015

Sommaire

Introduction.....	2
I. Présentation du projet	3
1- Lancement de l'application	3
Fonctionnalités réalisées.....	4
A. Le serveur	4
1 - Le réseau	4
2 - Routage d'une requête	4
3 – L'authentification.....	4
4 – L'accès à la base de données	4
5 – La gestion des logs.....	5
6 – le script de compilation Ant.....	5
7 – Les fichiers de configuration.....	5
8 – L'objet Response<T>.....	5
B. Objets Request et Response.....	6
C. Authentification.....	6
D. Le client	7
1. L'espace élève	7
2. L'espace enseignant	8
3. L'espace administration	8
II. Architecture du projet.....	9
A. Le serveur	9
B. Le client	9
C. La base de données	9
Conclusion	10
Annexe.....	0

Introduction

Pour ce deuxième projet de Java, nous devons réaliser une application complète de gestion d'une école, avec des fonctionnalités différentes pour les élèves, les professeurs et l'administration de l'école. L'application est divisée en deux parties : Le client, où les utilisateurs se connectent, et le serveur qui va effectuer toutes les infos sur la base de données.

Comme demandé dans le cahier des charges, et suivant les besoins du projet, nous devons également développer une interface graphique pour le client afin de rendre son utilisation plus agréable et intuitive. Nous avons utilisé Swing pour cela.

Ce projet nous a beaucoup appris sur la manière à réfléchir et visualiser la structure d'un programme avant de se lancer dans le code, ceci dans le but d'éviter d'obtenir un programme trop lourd, non optimisé, et donc, lent. L'enjeu était bien évidemment de finir de se familiariser avec le langage Java et avec la gestion de connexions entre deux entités.

I. Présentation du projet

Yagen est un gestionnaire de note adapté à tout type d'école, centralisé il regroupe l'espace élèves, professeurs et administrateurs dans la même application. L'interface graphique se veut d'être la simple possible dans le cadre d'une utilisation simple et efficace pour une personne allergique à la technique.

Le serveur Yagen est directement connecté avec une base données MySQL, les informations sont donc conservés. Le programme détient de plus des sécurisations importantes, telles que la sécurisation des requêtes SQL, la vérification du token coté serveur... Ainsi le client Yagen, même modifié par un utilisateur néfaste ne pourra accéder ou modifier les données importantes par exemple.

1- Lancement de l'application

Comme précisé dans la suite, nous utilisons un script de compilation Ant qui crée les fichiers jar et un zip. Ainsi, pour lancer l'application, il suffit d'aller dans le dossier **build**.

Aussi, étant donné que nous utilisons un serveur MySQL distant sur un serveur dédié, une connexion internet active est nécessaire. Dans le cas du réseau de l'Efrei, il suffit d'utiliser un tunnel SSH vers :

Adresse IP : 88.191.159.27

Port : 3306

Utilisateur : *yagen*

Mot de passe : *azerty*

Voici différents login pour se connecter à l'application :

- Élève :
 - **Login** : fabre **pass** : yagen
 - **Login** : lepot **pass** : motdepasse
- Professeur :
 - **Login** : marley **pass** : mdp
 - **Login** : chan **pass** : motdepasse
- Administrateur :
 - **Login** : norris **pass** : admin



Fonctionnalités réalisées

A. Le serveur

Le serveur a pour rôle de recevoir des objets **Request** et de les interpréter afin de renvoyer un objet **Response**.

1 - Le réseau

Afin de travailler plus aisément qu'avec les sockets nous avons utilisé la librairie Kryonet (ainsi que Kryo pour la sérialisation). Le fonctionnement de cette librairie est simple : elle distingue le client du serveur, et il suffit de créer un objet Listener pour obtenir les paquets.

Comme indiqué dans le sujet nous avons utilisé le protocole TCP sur le port 6789. Le port est néanmoins modifiable dans le fichier de configuration (config/server.properties).

2 - Routage d'une requête

Chaque requête est reçue par le Listener, qui crée à chaque fois un nouveau thread dédié au routage de celle-ci pour éviter la mise en tampon des requêtes des autres clients.

La classe chargée de traiter la requête s'appelle **RequestHandler**. Elle utilise la reflection (java.reflect) afin d'interpréter le nom de la classe et de la méthode et de l'invoquer si elle existe. Chaque méthode invocable doit retourner un objet Response<T> dont on étudiera sa composition plus tard.

3 - L'authentification

Afin de différencier le type de demande, nous avons créé une classe AuthRequest qui permet d'authentifier l'utilisateur. Son traitement s'effectue dans le listener qui renvoie au client un objet AuthReponse contenant le **token** que le client devra ensuite fournir à chaque requête. Ce token est une identifiant unique (généré via la classe UUID) qui identifie un utilisateur connecté.

4 - L'accès à la base de données

Afin de permettre à plusieurs clients de se connecter simultanément et afin d'éviter que la connexion au SGBD devienne un goulet d'étranglement, nous avons utilisé plusieurs connexions concurrentes.

Pour cela, nous utilisons la classe **ComboPooledDataSource** qui permet de créer un ensemble de connexions. Au lancement du programme seule une connexion est instanciée, mais si jamais elle venait à être trop sollicitée alors 2 nouvelles seraient créées (le temps d'acquisition d'une connexion est relativement long, donc on préfère en créer 2 en même temps). Le nombre maximal de connexions simultanées est déterminé par le fichier de configuration.

Aussi, pour des fins de débogage et surtout d'optimisation, à chaque connexion demandée, un thread du type **ScheduledExecutorService** est créé afin de vérifier que la connexion a bien été fermée dans les 60 secondes. En effet, une connexion non fermée est une ressource à la fois non disponible et non utilisée.

Enfin, par question de simplicité et de propreté, la classe DatabaseFactory est un **singleton**, c'est-à-dire que n'importe où dans notre programme il est possible de récupérer son instance avec `DatabaseFactory.getInstance()`.

5 – La gestion des logs

Pour toutes les sorties textes du programme nous avons utilisé la classe Logger incluse dans java (`java.util.logging.Logger`). Elle permet de définir différents niveaux de sortie (INFO, WARNING, etc).

La configuration de cet outil est disponible dans le fichier `logging.properties` dans le dossier `config`. Cependant, il n'est pas lu par défaut si le programme est lancé depuis Eclipse et non depuis le script batch dans le dossier `build`. Du coup, il faut préciser l'option suivante dans les paramètres de lancement de la VM Java :

`-Djava.util.logging.config.file=config/logging.properties`

6 – le script de compilation Ant

Les fichiers du serveur et du client sont contenus dans un seul projet, et afin de générer 2 fichiers jars différents, et plus globalement des dossiers client et serveur propres, nous avons utilisé un script de compilation Ant (le fichier `build.xml`).

Celui-ci crée le dossier `build` qui contient les librairies utilisées, les fichiers de configuration, les JAR, ainsi qu'un zip du dossier `build`.

7 – Les fichiers de configuration

Au lieu de recompiler notre serveur à chaque fois, nous avons utilisé des fichiers de configuration de type `Properties` (`java.util.Properties`). Le fichier `server.properties` est lu et interprété à chaque démarrage du serveur.

8 – L'objet `Response<T>`

Afin de permettre au serveur de renvoyer plusieurs types de données dans les réponses, nous avons utilisé un type paramétré. Cependant, pour chaque type utilisé nous devons l'ajouter à la classe de sérialisation du serveur et du client.

B. Objets Request et Response

Pour que le client puisse envoyer une requête au serveur afin d'obtenir des informations, enregistrer des données ou autres, nous avons créé un objet Request. Cet objet Request va contenir le Controller à appeler, la méthode à exécuter dans ce Controller, les arguments à passer à la méthode, un token qui sert à identifier l'utilisateur envoyant cette requête et enfin un id pour que nous puissions la récupérer facilement et l'identifier correctement.

Comme nous l'avons précisé précédemment, chaque requête est traitée par un Thread différent. Grâce à cela, nous pouvons ordonner les requêtes pour les traiter dans le bon ordre. L'id de la requête est attribué par le module gérant le réseau, ce qui permet de toujours savoir combien de requêtes nous avons en cours, et dans quel ordre elles doivent être traitées. Le token quant à lui est généré lors de l'authentification de l'utilisateur à sa connexion, un token est généré aléatoirement et sera le même pour cet utilisateur tant qu'il ne quittera pas l'application.

Cet objet Request permet donc au client de demander au serveur d'exécuter des instructions. Certaines de ces instructions nécessitent un retour avec des données, par exemple pour récupérer la liste des élèves.

Nous avons donc créé un objet Response qui servira à cette fin.

Notre objet Response est composé de l'id de la requête pour que nous puissions récupérer les bonnes informations si plusieurs clients font une requête en simultané. Il dispose aussi d'un champ isSuccess, qui nous permet de savoir si la requête n'a pas retourné d'erreur. Et pour finir, nous avons utilisé le principe de la généricité pour un champ data. Grâce à cela, nous pouvons retourner nos données de n'importe quel type (Une liste, un tableau, un entier, etc..).

C. Authentification

L'identification est la première étape de notre programme, lorsqu'on lance l'application, l'utilisateur doit se connecter avec son login et mot de passe pour accéder à l'espace qui lui est réservé.

Yagen contient trois espaces distincts.

- Espace élèves
- Espace professeur
- Espace administrateur

Chaque utilisateur possède un chiffre d'accès selon le grade qu'il dispose sur le programme ; (1 : élève, 2 : professeur, 3 : administrateur)

L'identification se divise en trois parties

- L'envoi d'une requête (AuthRequest) : constitue la requête de demande d'identification
- Le traitement de la requête en relation avec la base de données
- La réception de la réponse (AuthResponse) : contient la réponse de l'identification

AuthRequest :

Chaque AuthRequest contient le login et le mot de passe que l'utilisateur vient de rentrer dans le formulaire de connexion. Coté serveur la requête est récupérée et traitée. Le serveurinstanciera un objet AuthResponse contenant les informations suivant

1. isSuccess : **true** ou **false** si l'identification est réussie
2. Le token unique généré et renvoyé à l'utilisateur
3. La valeur isAdmin à **true** ou **false**
4. La valeur isTeacher à **true** ou **false**
5. Id : qui contient l'id de l'utilisateur contenu dans la table users de notre BDD qui vient de se connecter. Cette valeur est essentielle pour la suite de nos requêtes.

Si les deux booléens sont à **false**, l'utilisateur a donc le statut d'élève.

AuthResponse :

La réponse sera reçue côté client dans le fichier Network. En fait une fois la requête d'identification envoyée, nous sommes obligés d'attendre la réponse et de gérer les temps de latence entraînés par le réseau. Ainsi à la fin de l'envoi on sort de la class Listener de notre bouton valider, pour attendre la réponse dans notre fichier Network.

Une fois la réponse reçue on vérifie dans un premier temps la valeur du booléen isSuccess, voir si l'identification est réussie ou non. Si elle ne l'est pas on affiche un message d'erreur et renvoie l'utilisateur à la fenêtre de connexion. Si l'identification est réussie, on renvoie l'utilisateur dans l'espace attribué à son grade.

D. Le client

1. L'espace élève

L'espace étudiant se divise en deux parties :

- L'inscription à une majeure :

L'étudiant ne peut s'inscrire qu'une fois à une majeure, une fois choisi aucune modification ne sera acceptée (l'élève ne pourra pas changer de majeure quand il le désire). Si l'élève n'est pas inscrit, il pourra choisir sa majeure entre plusieurs choix proposés.

Si l'élève est déjà inscrit à une majeure et essaye de s'inscrire, un message d'erreur s'affichera. Toute la partie de l'inscription de la majeure est totalement automatisée. Si une modification est faite dans la base de données, le client gèrera cette modification.

- La consultation des notes :

Dans cette partie, l'élève peut consulter ses notes. Ses notes sont récupérées de la base de données et affichées dans un jTable.

2. L'espace enseignant

Lorsque l'enseignant se connecte, il arrive sur une page dédiée dans laquelle il dispose de trois fonctionnalités :

- Ajouter une note
- Modifier une note
- Vérifier le tutorat

S'il va dans « Ajouter une note », il devra choisir une matière et un élève. La liste des matières ne contient que les matières dans lesquels il enseigne. Il doit ensuite valider, entrer le type d'examen (CE, DE, Oral, etc..) et la note.

Si l'enregistrement réussit une fenêtre de validation l'informe, sinon c'est une fenêtre d'erreur qui apparaît.

Pour modifier une note, il doit sélectionner un élève et une matière. Là encore seules les matières où il enseigne apparaissent. Une fois fait, il arrive sur une autre fenêtre avec les notes de l'élève dans la matière en question. Elle se présente sous forme d'un tableau. Il peut alors modifier le type de la note en double cliquant simplement sur la cellule et y rentrer la nouvelle valeur, idem pour la note.

Dans l'espace tutorat, chaque professeur peut être tuteur d'un groupe de td relatif à une promo. Dans la partie tutorat, le professeur sera directement alerté du nom, prénom, groupe et promo de chaque élève ayant au moins trois notes en dessous de la moyenne. Si aucun élève n'est en difficulté, le jTable n'affichera rien.

3. L'espace administration

L'espace d'administration permet d'ajouter et de supprimer des étudiants, des professeurs, et des matières. Il permet aussi de lier un professeur à une matière.

II. Architecture du projet

A. Le serveur

Diagramme en Annexe

B. Le client

Diagramme en Annexe

C. La base de données

Diagramme en Annexe

Conclusion

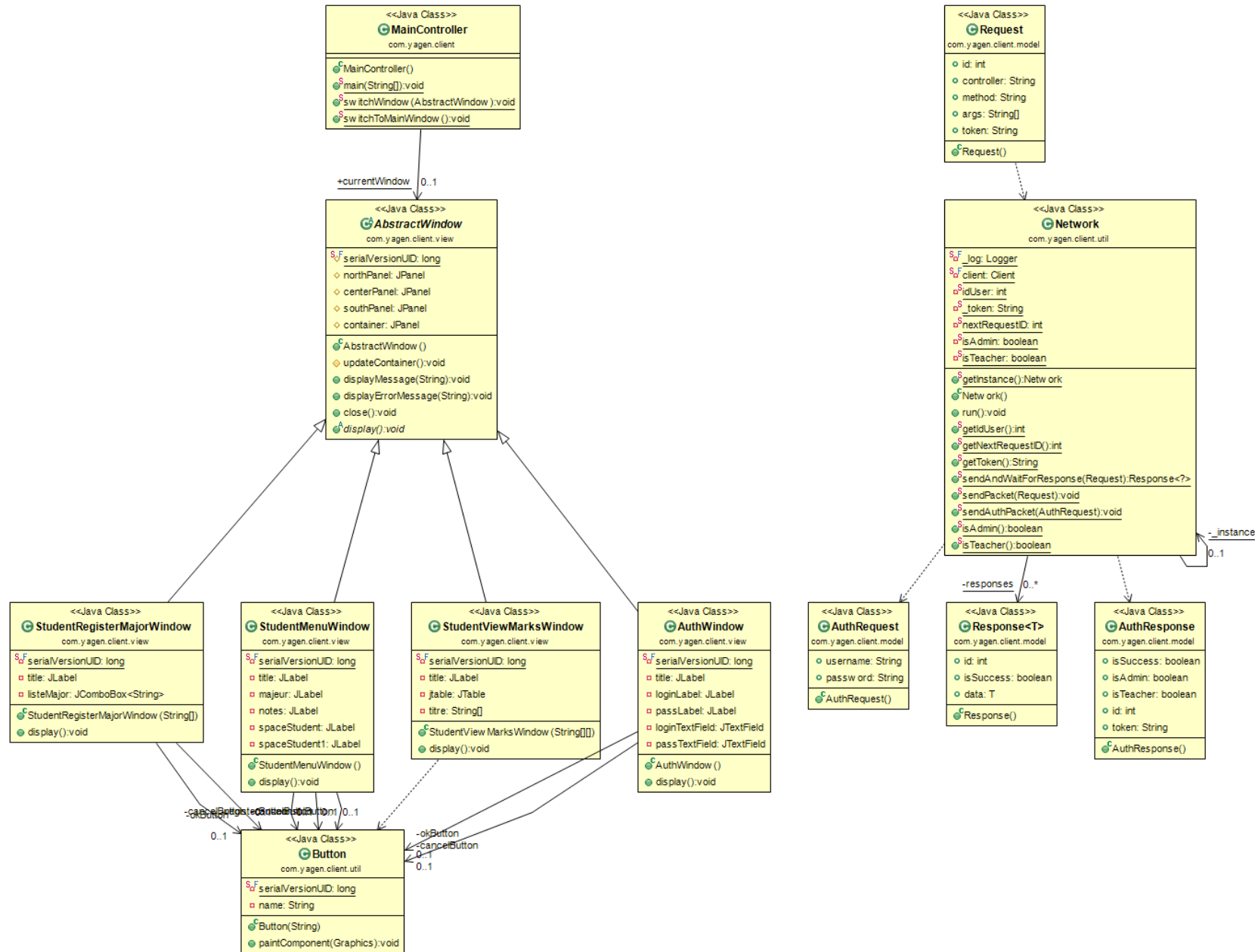
Pour finir sur ce projet, nous avons essayé de respecter au maximum le cahier des charges. En plus d'avoir réalisé nos objectifs de départ nous avons implémenté des améliorations au niveau de la performance. Notre application supporte en effet un grand nombre de connexions simultanées au serveur, et sait gérer une file d'attente pour les requêtes à la base de données afin de ne pas la surcharger.

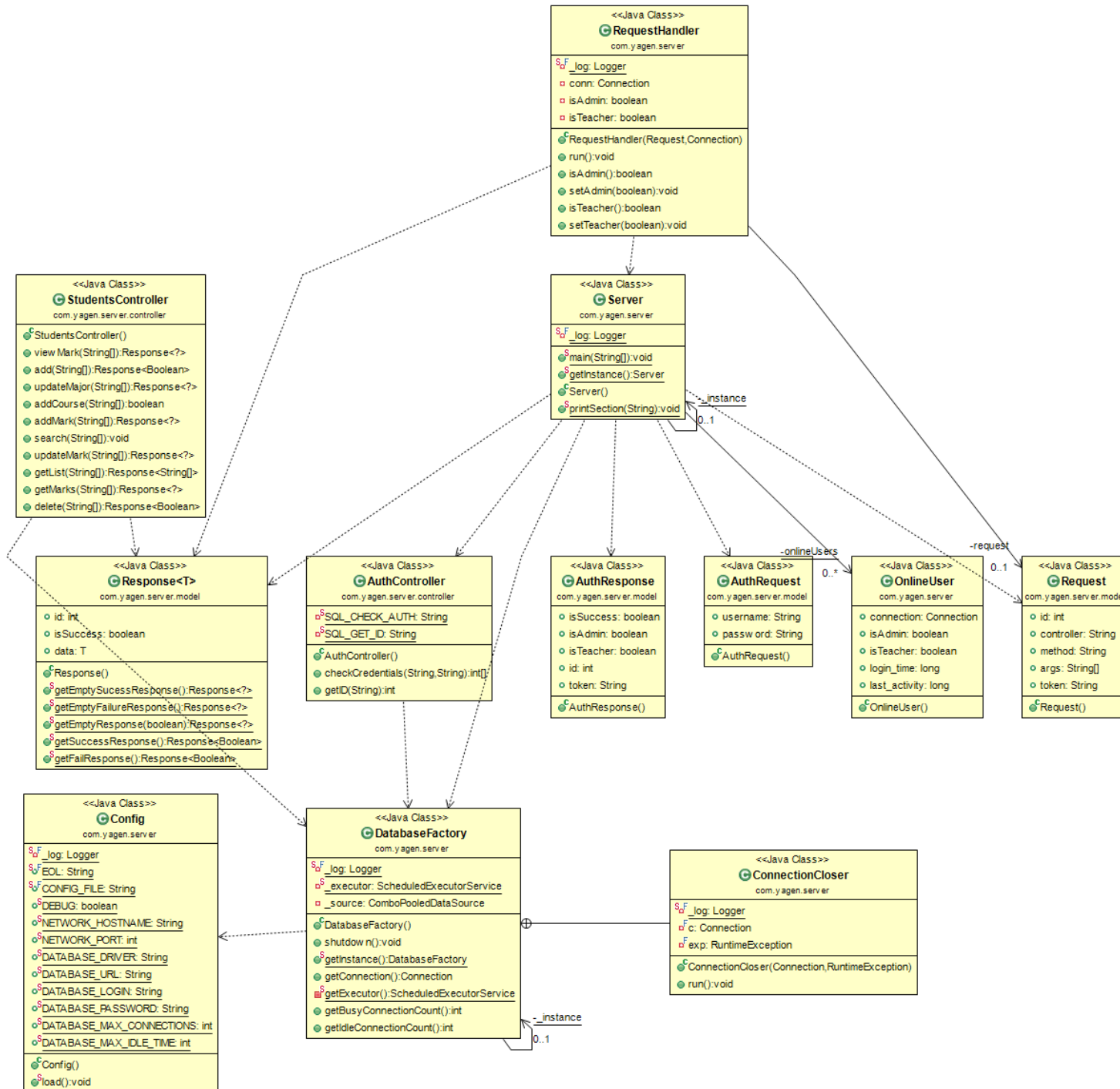
Nous avons aussi beaucoup appris quant à l'utilisation du langage Java, la création d'un environnement liant serveur et client, et cela nous a montré que ce langage avait d'énormes possibilités.

Ce projet nous a également permis d'apprendre à utiliser Swing pour la création d'interface graphique pour nos éventuels futurs travaux sur Java.

Annexe

Client





Base de données

