



Avenue du Ciseau 15

1348 Louvain-la-Neuve

Tel. : +3210/47.53.90

DÉVELOPPEMENT D'UN FRAMEWORK DE CORRECTION

DE CODE JAVA POUR PYTHIA

(PLATEFORME D'APPRENTISSAGE DE LA PROGRAMMATION)

Travail de fin d'études en vue de l'obtention du diplôme de bachelier en
Informatique et Systèmes : finalité Technologie de l'informatique

ETUDIANT : BISHOP GREGORY

RAPPORTEUR : LAMBEAU CHRISTIAN

ANNEE ACADEMIQUE 2013-2014

Remerciements

La découverte de Pythia et ses possibilités fut le moment clé de mon travail. Sans cela, j'aurais probablement trouvé un autre sujet beaucoup moins passionnant. En effet, ce TFE¹ fut vraiment attrayant pour moi. Malgré les longues journées de stage parfois fatigantes, je trouvais la motivation nécessaire au soir pour avancer sereinement. Je me dois donc de remercier Monsieur Combéfis pour m'avoir fait découvrir cette plate-forme et également pour tout ce qu'il a pu m'apprendre. Par ce stage, je remercie Madame Van den Schrieck pour m'avoir proposé cette période d'apprentissage et conseillé un travail en rapport avec mon objet de stage.

Je souhaite naturellement remercier Monsieur Lambeau pour ses nombreux conseils et remarques qui furent bénéfiques dans la réalisation de mon projet. De plus, il m'a fait part de nombreuses astuces et idées concernant jQuery.

Si aujourd'hui mon travail de fin d'études est pratiquement compatible avec l'EPHEC², c'est principalement grâce à l'aide généreuse que m'a proposée Madame Vroman. En effet, malgré le fait qu'elle ne soit pas mon rapporteur de TFE, elle a volontiers accepté de répondre à quelques questions concernant des détails de mon interface utilisateur. De plus, j'ai été mis en contact avec Monsieur Verhelst, notamment gestionnaire du réseau EPHEC, et Monsieur Delvigne, gestionnaire d'eCampus. Ils ont pu m'aider à mieux comprendre les aspects techniques dont aurait besoin ma plate-forme pour pouvoir se joindre au réseau de l'école. C'était donc agréable de pouvoir discuter avec d'autres enseignants prêts à m'expliquer tel ou tel aspect. Se sentir dans la bonne voie est quelque chose de sincèrement rassurant.

¹ Travail de fin d'études

² École pratique des hautes études commerciales

Table des matières

Remerciements	2
1. Introduction.....	4
2. Corps du travail.....	5
2.1 Avis des étudiants concernés.....	5
2.2 Description des outils utilisés	6
2.2.1 Pythia	6
2.2.2 Python 3.....	7
2.2.3 MySQL.....	7
2.2.4 Lightweight Directory Access Protocol (LDAP)	7
2.2.5 HTML, CSS, jQuery & JSON	8
2.3 Solution.....	8
2.3.1 Le Serveur LDAP.....	9
2.3.2 La base de données distante	10
2.3.3 La plate-forme	10
3. Difficultés rencontrées	26
4. Améliorations	27
5. Conclusion	28
6. Webographie	29
7. Annexes	30
I. Schéma relationnel.....	30
II. Diagramme d'activité : Soumission d'un étudiant.....	31

1. Introduction

Durant mes deux années à l'Université catholique de Louvain en Sciences informatiques et mes trois années à l'EPHEC en Technologie de l'informatique, j'ai souvent été confronté au même cas de figure. Les professeurs demandent à ce que l'on soumette nos travaux sur la plate-forme Claroline que ce soit eCampus ou iCampus. Dû au nombre élevé de soumissions, les professeurs n'ont pas nécessairement le temps de corriger chaque exercice pour chaque étudiant. Ainsi, nous n'avons pas constamment un feedback sur nos implémentations en langage de programmation. Nous avons bien évidemment nos compilateurs installés localement sur nos machines, mais il arrive qu'ils soient soit en anglais, soit incomplets vis-à-vis des corrections à apporter à nos codes. Approchant de la session d'examens, il était donc possible que des étudiants n'étaient pas totalement confiants quant à leur capacité à implémenter correctement. Fort de mes débuts à l'UCL³ et de mon anglais, je n'ai pas rencontré ces problèmes dans le domaine de la programmation.

Le hasard des choses ou plus sincèrement mes antécédents firent qu'un stage me fut proposé à l'EPL⁴ où j'avais passé quatre quadrimestres. Ainsi, quand j'appris que mon stage se baserait principalement sur Pythia, une plate-forme intelligente qui permet d'analyser du code soumis et de renvoyer du contenu utile sur base de l'implémentation, j'ai directement pensé à une alternative pour l'EPHEC. J'en fis donc la demande en tant que travail de fin d'études, ce qui fut accepté. Il devait se baser autour du Pascal, langage qui, jusqu'en décembre 2013, était le premier langage de programmation enseigné à l'EPHEC.

Mon framework permettrait aux enseignants d'avoir rapidement un compte rendu sur les exercices réalisés par les étudiants. Ces derniers pourront donc s'entraîner à leur guise sur des exercices basiques. Cette interface web pédagogique participerait donc à l'apprentissage des néo programmeurs.

Avant de pouvoir commencer mon travail, mon premier objectif fut d'apprendre le fonctionnement de la plate-forme Pythia. Dû à sa popularité plutôt faible et à sa jeunesse, il est fortement compliqué de trouver de la documentation principalement technique sur celle-ci. Ainsi, il me fallut découvrir une partie des fonctionnalités via mes premières semaines de stage. Je n'avais pratiquement aucune connaissance à son sujet, je ne savais rien de son système. Conscients des règles imposées par l'EPHEC, Monsieur Combéfis (mon maître de stage), Monsieur Lambeau et moi-même avons convenu que le travail effectué autour de Pythia durant cette période de stage ne concernerait pas la fonctionnalité principale qu'est l'analyse de codes mais plutôt l'amélioration de la plate-forme ainsi que l'ajout de divers modules. Bien vite, je me rendis compte que la nouvelle version de la plate-forme Pythia ne contenait plus cette "intelligence" précédemment citée. En effet, cette dernière ne permettait que de lancer des machines virtuelles contenant l'environnement adéquat à la tâche désirée. Ces fichiers systèmes étaient préalablement conçus par Pythia à l'aide de paquets Debian Squeeze. Les fonctionnalités principales de cette plate-forme sont donc la création et l'exécution de machines virtuelles Linux.

³ Université catholique de Louvain

⁴ Ecole Polytechnique de Louvain

Cependant, au cours des premières semaines de mon stage et de l'élaboration de mon TFE, je fus informé que l'EPHEC apporterait des modifications aux cours de programmation donnés au sein de l'établissement. En effet, suite au projet Marcourt, des langages tels que C ou encore Pascal étaient appelés à disparaître de l'enseignement à l'EPHEC. C'est alors qu'un débat prit place, qu'allait advenir ma plate-forme ? Il fallut donc trouver une solution rapidement afin de ne pas trop me retarder au niveau des différentes échéances. Une décision fut prise quelques jours plus tard, ma plate-forme se baserait désormais autour du langage Java, qui serait conservé pour les étudiants de deuxième année.

2. Corps du travail

2.1 Avis des étudiants concernés

La première étape de mon travail consista en l'acquisition du point de vue des étudiants face à certains langages de programmation, leurs appréhensions ou encore ce qu'ils préféraient dans ceux-ci. J'avais commencé mon travail sur le langage Pascal, j'ai donc recueilli diverses opinions vis-à-vis de celui-ci. J'ai par la suite posé des questions similaires par rapport au Java. Voici une liste d'opinions anonymes qui pourraient être utiles aux enseignants dans le futur.

PASCAL

- "Ce qui plaît : la simplicité de la prise en main du langage, ce qui déplaît : les ";" aléatoires, les limites du langage."
- "Je trouve qu'il serait meilleur de se pencher sur le python pour initier les nouveaux inscrits à la programmation."
- "Le Pascal a comme grande qualité de pouvoir facilement donner un aperçu de la logique de programmation, au vue du langage de haut niveau (très proche de l'humain). Évidemment, savoir coder en Pascal se révèle être complètement inutile (le Python aurait été mieux par exemple, mais je le trouve moins facile d'accès quand on commence la programmation). En résumé, je ne donnerai pas de point négatif au Pascal simplement parce qu'il n'a plus aucune raison d'être hormis dans l'apprentissage de la logique de base."
- "Venant d'un langage plus "évolué" (Java), je déteste les ";" aléatoires. J'ai passé plus de temps à en mettre ou en supprimer des ";" qu'à coder."
- "Pourquoi ne choisis-tu pas un langage sans bugs pour ton TFE ?"
- "1) plus personne n'utilise ce langage et j'ai quand même espoir qu'avec la masse de langages existants, il y ait moyen de trouver un langage utile (Ruby, Python,...) et "facile pour débiter". 2) syntaxe lourde et parfois incompréhensible (Begin, end et ses dérivées ainsi que des randomisé ";" pour les if/else notamment). 3) IDE infâme à souhait mais cohérent avec le point 1. 4) seul point positif : quelques outils syntaxiques utiles comme le with, le switch conditionnel ou encore le in[..]. 5) ah j'ai

failli oublier le pire de tous : les tableaux qui commencent à l'indice 1. Rien de mieux pour semer le trouble lors de l'apprentissage du C..."

- "(-) Navigation dans l'IDE horrible (même un bloc-notes fait mieux) (+) Sa capacité à s'arrêter au premier faux pas du programmeur DEBUTANT car oui, ce langage est adapté aux débutants uniquement selon moi."

JAVA

- "Ce qui me plaît : langage abordable, puissance de la JVM, objectifs concrets rapidement atteints, massivement utilisé. Ce qui ne me plaît pas : gros programmes lents à la compilation, trop verbeux, peut-être un petit coup de vieux ? Remarques : bien que Java8 soit sorti, des langages fonctionnels (SCALA, HASKELL, CLOJURE etc. ...) sont maintenant plus performants pour moins de ligne de code."
- "Bien plus pratique à utiliser que le C. Pas besoin de s'embarrasser de toute l'histoire des pointeurs. Tout est fait automatiquement et dans le cadre de l'apprentissage, ça permet de se concentrer sur les algorithmes de base sans trop de problèmes. Pour tout ce qui est orienté objet, l'héritage est juste un peu complexe à assimiler. Un point positif est la portabilité du langage qui permet d'utiliser, entre autres, une interface graphique de base fonctionnant sur tous les os. La critique que j'ai le plus entendue est la relative lenteur d'exécution mais à notre niveau, il n'y a aucune différence. En bref, Java fait plein de trucs dans l'ombre et j'ai envie de dire, ce n'est pas plus mal pour un débutant !"
- "Le Java ouvre les portes sur le marché de l'emploi."
- "J'aime beaucoup les interfaces graphiques en Java."

J'en ai conclu, qu'au final, passer de Pascal à Java fut une bonne chose tant les étudiants ont tendance à préférer celui-ci. De plus, la documentation et la communauté autour de JAVA sont bien plus vastes.

2.2 Description des outils utilisés

2.2.1 Pythia

Pythia est une plate-forme dont le but premier est d'enseigner la programmation et la conception d'algorithmes. Elle exécute le code dans un environnement sécurisé afin d'éviter toute erreur préjudiciable pouvant être causée par du code étudiant malicieux. Son principal avantage est de fournir un feedback intelligent aux apprenants.



La plate-forme offre un environnement autonome qui supporte un enseignement de la programmation. Il est donc possible de proposer des exercices aux étudiants, exercices qu'ils pourront résoudre en soumettant leurs réponses. Le feedback renvoyé leur permet de

corriger leurs erreurs. Il est important de souligner le fait que Pythia n'est pas lié qu'à un seul type de langage pour les énoncés. En effet, grâce à son environnement "Bac-à-sable", la plate-forme est multi-langage.



2.2.2 Python 3

Python est un langage de programmation orienté objet, multi-paradigmes et multiplateforme utilisé pour écrire différents types d'application. Il est considéré comme un langage de haut niveau permettant de s'axer autour du problème à résoudre, permettant d'utiliser des mots usuels des langues naturelles ainsi que des symboles mathématiques. Sa syntaxe assez rigoureuse permet une lisibilité bien plus évidente que d'autres langages tels que C ou Java.

Le choix du langage m'a été imposé ou plus sincèrement suggéré par mon maître de stage. Il me permet de pouvoir réaliser mes scripts analysant les codes soumis par les étudiants. La manipulation des fichiers avec ce langage est également bien plus facile.

De plus, j'utilise le module Bottle de Python 3 pour implémenter un serveur web. Je n'ai donc pas besoin d'utiliser un serveur APACHE/PHP pour déployer mon interface utilisateur.

2.2.3 MySQL

MySQL est un système de gestion de base de données (SGBD) relationnelle faisant partie des logiciels de gestion de base de données les plus utilisés au monde. C'est donc pour cela que j'ai préféré utiliser ce système à PostgreSQL. En effet, légèrement différent de son concurrent, il m'a été suggéré d'utiliser cet autre système administrable via pgAdmin. Cependant, étant bien plus habitué et en confiance avec le premier gestionnaire cité, j'ai préféré ne pas explorer un nouveau système.



Pour se faire, j'utilise un MySQL SERVER que je gère via l'application découverte à l'EPHEC : MySQL WorkBench.

2.2.4 Lightweight Directory Access Protocol (LDAP)

Afin de pouvoir subvenir aux exigences techniques de l'EPHEC, il a été décidé d'utiliser un serveur LDAP pour l'authentification sécurisée des utilisateurs d'ores et déjà existants dans le système.

LDAP est un protocole reposant sur le Protocole Internet (TCP/IP) et permettant d'interroger et de gérer des services d'annuaire. L'arbre représente souvent la politique de la structure représentée. On utilise donc les racines ou les premières branches de l'arbre pour représenter les éléments de base de l'annuaire (le nom de la société, de l'application,...) tandis que les branches plus profondes représentent les entités (utilisateurs et groupes d'utilisateurs). Ce serveur est donc comparable à un Active Directory.

J'ai utilisé le logiciel Apache Directory Studio pour visualiser mon serveur mais j'ai préféré utiliser les commandes via un terminal pour ajouter des entités.

2.2.5 HTML, CSS, jQuery & JSON

L'HyperText Markup Language (HTML) étant le format de données par défaut pour représenter les pages web, il était certain que ce dernier serait utilisé avec du CSS dans mon projet.

À côté, jQuery, qui est une bibliothèque du JavaScript (JS), permet l'interaction entre ce dernier et l'HTML. Elle a l'avantage de simplifier l'utilisation du JS mais est plus lourde à exécuter. Néanmoins, pour ma plate-forme, ce désavantage est négligeable.

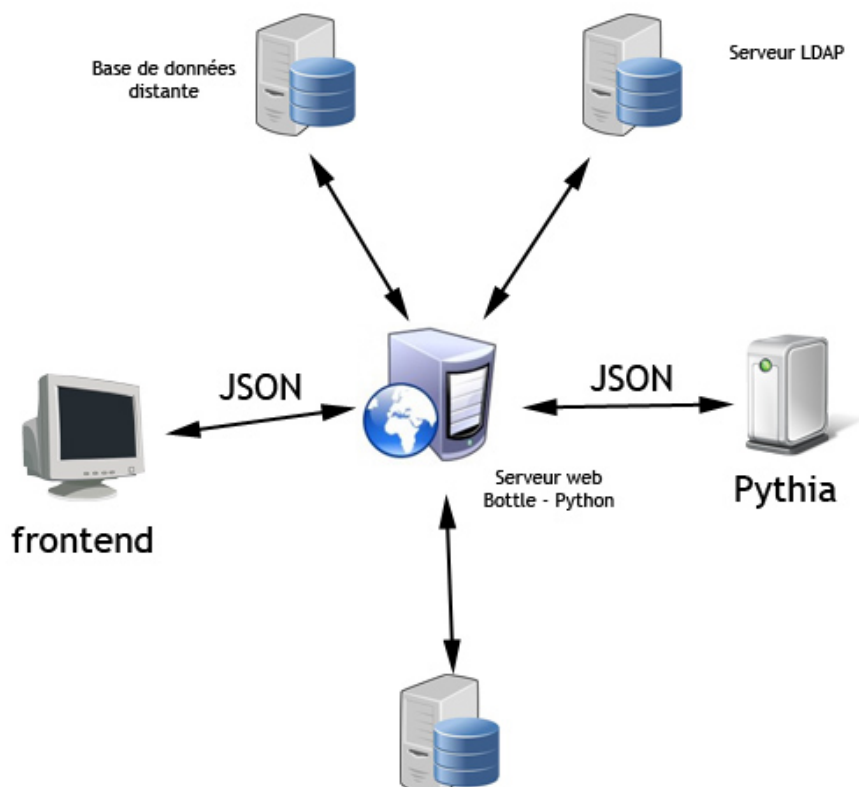
Le *JavaScript Object Notation* (JSON) permet de formater des données textuelles de la même manière que le fait le XML. Certes limité, JSON présente une structure plus agréable que ce dernier.



2.3 Solution

La solution que je propose se divise en trois parties :

- Un serveur LDAP distant permettant l'authentification des utilisateurs ;
- Une base de données MySQL distante contenant toutes les informations des utilisateurs ;
- Ma plate-forme contenant diverses entités :
 - Pythia avec un environnement pour les machines virtuelles supportant Java et Python (pour les scripts d'analyse) ;
 - Serveur web Python à l'aide du module Bottle ;
 - Une base de données MySQL ;
 - Une interface utilisateur ;



2.3.1 Le Serveur LDAP

Afin de subvenir aux exigences de l'EPHEC et de son système utilisateur, j'ai installé un serveur LDAP me servant donc de serveur d'authentification, ceci afin de simuler le probable Active Directory (AD) de l'école.

Mon service d'annuaire est probablement conçu différemment de celui de l'institut mais je pense qu'il s'en rapproche suffisamment, du moins dans la logique. J'y ai créé donc un arbre avec comme racine deux contrôleurs de domaines : "ephec" et "be".

Plus bas, nous retrouvons un groupe d'utilisateurs à savoir "students" dans lequel se retrouvent tous les étudiants inscrits. Je ne suis pas certain de comment différencier les étudiants en TI⁵, marketing ou comptabilité. En marge de cette unité d'organisation, nous retrouvons les utilisateurs "professeurs" ainsi qu'un compte admin.

J'ai choisi cette séparation en fonction des adresses mails actuellement utilisées. Un étudiant a une adresse HExxxxxx@students.ephec.be tandis qu'un professeur en a une plus simple xxxxxx@ephec.be.

Pour ajouter un groupe d'utilisateurs, il suffit d'utiliser la commande suivante :

```
ldapadd -cxWD cn=admin,dc=ephec,dc=be -f students.ldif
```

Il est ensuite demandé le mot de passe du serveur. Ce fichier "new-group.ldif" contient les lignes suivantes :

```
dn: ou=students, dc=ephec, dc=be
objectClass: top
objectClass: organizationalUnit
ou: students
```

De manière assez similaire, un fichier "student.ldif"

```
dn: uid=HE111111, ou=students, dc=ephec, dc=be
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane Doe
gn: Jane
sn: Doe
uid:HE111111
userPassword: {CRYPT}*
```

Il faut y fournir nom, prénom, matricule. Un mot de passe est à configurer par la suite à l'aide de la commande suivante :

```
ldappasswd -xWD cn=admin,dc=ephec,dc=be
-S uid=HE111111,ou=students,dc=ephec,dc=be
```

⁵ Technologie de l'Informatique

Sur ce serveur, chaque mot de passe est crypté/haché en SSHA. Je suis bien sûr conscient que tous les utilisateurs EPHEC sont déjà existants sur l'AD de l'école. Je trouve cependant important de montrer comment ajouter une unité à l'arbre.

Une fois les différents utilisateurs créés, le serveur LDAP est prêt, il attend les requêtes provenant de mon serveur web Python.

2.3.2 La base de données distante

Également exigée par l'EPHEC, une base de données externe à celle de ma plate-forme est nécessaire. Les étudiants et enseignants ne devant pas créer un nouveau compte, il faut pouvoir facilement aller chercher leurs informations de base à l'aide de leur identifiant.

Ces informations sont assez simples : prénom, nom, email et dans le cas d'un étudiant, le groupe auquel il appartient.

Comme pour l'Active Directory de l'EPHEC, je n'ai pas pu avoir connaissance de la structure de la base de données. Celle-ci contient donc selon moi deux tables.

La première "USER" est la principale avec tous les utilisateurs et leurs informations respectives citées précédemment avec leur matricule ou identifiant comme clé primaire afin de respecter l'unicité.

La seconde, ne contenant uniquement qu'une colonne, me paraît moins probable mais pas impossible. Elle conserve l'ensemble des groupes ou classes de l'école, ceci afin de pouvoir les récupérer assez facilement.

2.3.3 La plate-forme

2.3.3.1 La base de données locale

Ayant commencé mon projet dans l'idée de devoir créer des nouveaux utilisateurs pour chaque étudiant ou enseignant, j'ai dû modifier mes plans par la suite. En effet, Madame Vroman souhaite pouvoir intégrer cette plate-forme au système de l'EPHEC. Ainsi, il faut aller chercher les données sur un serveur distant tel que décrit au point précédent. Je ne prends donc plus en compte une table USER locale, les matricules viennent s'ajouter via les variables sessions des connexions. Les identifiants utilisateurs dans ma base de données ne sont pas des clés étrangères, en effet, il n'y a pas de table les stockant concrètement.

Mon système se divise en cours, eux-mêmes divisés en exercices. Chaque tentative pour un exercice spécifique de la part d'un utilisateur correspond à une soumission. Un exercice peut donc contenir un nombre élevé de soumissions.

La première table que je peux brièvement décrire est donc la table COURSE, reprenant l'ensemble des cours. J'ai choisi les termes anglais car je trouve cela plus agréable et plus intuitif. Elle contient principalement le titre, la brève description, l'identifiant du professeur principal du cours, l'identifiant du langage de programmation choisi ou encore l'état actuel du cours.

Il en va de même pour les exercices à la différence près que les descriptions sont stockées sur le serveur même et non en base de données. De plus, il y a l'identifiant du cours auquel l'exercice se rapporte.

Chaque soumission se retrouve dans la table SUBMISSION. L'état et la date de soumission sont bien évidemment les attributs clés. L'utilisateur et l'exercice auxquels elle se rapporte sont également spécifiés.

Je peux ensuite vous parler des autres tables telles que FOLLOWS ou MODERATES. La première reprend les étudiants étant inscrits à un cours en particulier. Un étudiant peut être inscrit manuellement, via son identifiant, ou via son groupe, lui-même inscrit. La seconde table reprend cette fois-ci les enseignants ayant des droits sur certains cours.

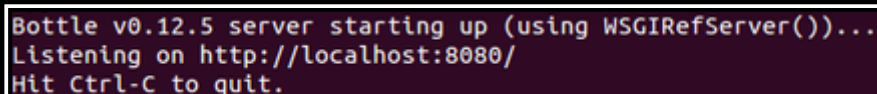
Pour terminer, des tables moins importantes telles que LOGIN, LANGUAGE ou encore ROLE reprennent respectivement la date et l'heure de la dernière connexion de l'utilisateur, les langages de programmation disponibles sur la plate-forme et les noms des rôles des utilisateurs (admin, professeur, étudiant).

2.3.3.2 *Le serveur Web*

J'ai implémenté un serveur web avec le langage Python pour deux raisons. La première est personnelle et subjective. En effet, je n'avais jamais entendu parler de serveur web autre qu'Apache et PHP, j'en ai donc saisi l'occasion. La seconde, quant à elle, vient de mon maître de stage Monsieur Combéfis. Il m'a conseillé Python pour la facilité d'implémentation des sockets et donc notamment la connexion plus aisée avec Pythia.

Pour se faire, j'ai utilisé le module Bottle qui, après avoir été importé, ne nécessite qu'une simple commande pour lancer le serveur web en mode écoute.

```
run(host='localhost', port=8080, debug=True, reloader=True, app=app)
```

A terminal window with a dark background and light-colored text. The text shows the Bottle server starting up, listening on http://localhost:8080, and提示ing to hit Ctrl-C to quit.

```
Bottle v0.12.5 server starting up (using WSGIRefServer())...  
Listening on http://localhost:8080/  
Hit Ctrl-C to quit.
```

Les paramètres de la fonction **run()** sont assez simples. "Reloader" signifie qu'en cas de modifications dans les fichiers, le serveur redémarre automatiquement, en les prenant en compte.

L'ensemble des fonctionnalités de la plate-forme sont globalement liées mais réparties dans des fichiers Python afin de garder le code clair et cohérent. Je ne décrirai ici qu'une partie de chaque fichier.

1. Fichier **server.py**

Le serveur Bottle est exécuté grâce au fichier **server.py**, toutes les requêtes POST ou GET venant de l'interface utilisateur sont redirigées vers leur fichier respectif. Par exemple, la requête d'une création d'un exercice ou d'un cours est envoyée vers le fichier **create.py**. C'est également via ce fichier serveur que du contenu est envoyé à Pythia et par conséquent, la réponse y est également réceptionnée. Pour cette connexion, j'utilise un socket se connectant au port 9000, celui où Pythia est par défaut à l'écoute.

Pour rediriger une requête vers la bonne fonction du fichier qui, à son tour, redirigera vers le bon fichier, il faut vérifier la route empruntée par la requête à l'aide de cette commande :

```
@route('/create/exercice/<course>',method="POST")
def create_exercice(course):
```

Les deux premières parties du lien forment une route statique alors que la seconde partie est dynamique et la valeur s'y trouvant peut être alors utilisée comme argument pour la fonction liée à la route.

Avant de rediriger une création, ce fichier vérifie également la validité des paramètres envoyés. En effet, il analyse chaque élément envoyé en JSON et les compare aux valeurs attendues. En cas d'erreur, le processus s'arrête et renvoie un message à l'interface utilisateur via du JSON.

Il permet également le téléchargement des fichiers des soumissions, soit chaque fichier individuellement soit, après compression, l'ensemble des soumissions sous forme d'une archive ZIP.

Lors de la connexion de l'utilisateur au serveur web, il y a l'initialisation d'une session qui permettra de lier des variables à l'utilisateur sur le frontend.

```
# Création de la session
session_opts = {
    'session.type': 'file',
    'session.data_dir': './session/',
    'session.auto': True,
    'session.timeout':1800
}
```

Une session est programmée pour durer trente minutes maximum en inactivité. Le contenu de la session est stocké dans un dossier **session** sur le serveur.

2. Fichier **function.py**

L'ensemble des fonctions utilisées dans les divers fichiers Python pour faire interagir la plate-forme se trouvent dans le fichier **function.py**.

Il est important de noter qu'au début de chaque fichier autre que **server.py**, la session actuellement initiée par ce dernier est transférée afin de pouvoir constamment prendre en compte l'utilisateur actuellement connecté à la plate-forme.

Les fonctions principales sont bien évidemment les fonctions de connexion aux différentes bases de données. Il faut en effet pouvoir se connecter pour en retirer ce dont nous avons besoin.

- **Connect()** : se connecte à la base de données locale
- **EPHECconnect()** : se connecte à la base de données distante

Lors d'une connexion via l'interface web, on vérifie le login entré. Si celui-ci contient «HE» suivi d'une série de chiffres, il sera d'ores et déjà considéré comme un étudiant. Le login « admin », qui reste modifiable, sera reconnu en tant qu'administrateur à plein pouvoir. Pour terminer, si aucun des deux cas précédents ne s'applique, l'utilisateur sera considéré en tant que professeur.

Les fonctions telles que **isAdmin()**, **isTeacher()**, **isConnected()** parlent d'elles-mêmes.

Pour le reste, il y a majoritairement des « getters » tels que **getExercices()** qui renvoie tous les exercices et leurs attributs, **getAllTeachers()**, **getLastLogin()**, **getUser()** et bien d'autres encore.

La fonction **getStudentsFor()** est particulière, elle va premièrement chercher tous les étudiants existants pour ensuite les comparer avec les étudiants inscrits manuellement à un cours (et non via leur groupe). Dans le cas où un étudiant est effectivement inscrit, elle l'ajoute à une liste. Dans le cas contraire, elle vérifie ensuite s'il n'est pas inscrit via les groupes inscrits au cours. La fonction est implémentée de sorte que l'étudiant ne sera pas compté deux fois dans le groupe des inscriptions.

3. Fichier **htmlgenerator.py**

Le fichier **htmlgenerator.py** s'occupe de générer l'html à renvoyer via JSON à l'interface utilisateur. Ceci sera davantage expliqué au point concernant le frontend. En effet, l'ensemble de ce fichier traitant l'aspect visuel, il n'est pas dans mon intérêt de le décrire ici même.

En cas d'inactivité trop longue ou de déconnexion volontaire, toutes les navigations jusqu'alors autorisées seront obligatoirement redirigées vers la page de connexion, ceci afin d'éviter l'utilisation d'une session ne nous appartenant pas.

4. Create.py

Après chaque validation par **server.py** lors d'une création, le contenu du JSON est traité dans **create.py**. Il y a alors deux fonctions : la création d'un cours ou celle d'un exercice.

Dans le cas d'un cours, c'est assez simple. La totalité des paramètres encodés vont en base de données. Un UID⁶ de douze caractères est généré et placé en tant que clé primaire du cours à condition que celui-ci n'existe pas déjà. Dans le cas contraire, une autre clé sera générée. Les modérateurs, groupes et étudiants sélectionnés sont également sauvegardés.

Dans le cas d'un exercice, le processus est bien plus long et plus compliqué. En effet, créer un exercice implique de créer une tâche sur Pythia, d'y créer divers fichiers scripts Python et de lancer un MAKE sur Pythia afin que la tâche soit compilée par le système pour enfin supprimer tous les fichiers utiles lors de la création mais désormais inutiles après celle-ci.

Il est important de souligner que le contexte d'un exercice (son énoncé en somme) sera non pas stocké en base de données, mais bien dans un fichier sur le serveur.

Lors du début du processus, le script s'assure que l'exercice n'existe pas déjà en vérifiant son chemin absolu sur le serveur.

Le chemin du fichier contexte est comme suit :

`./CONTEXT/ID_COURSE/ID_TEACHER/ID_EXERCISE/ID_EXERCISE.desc`

Dans ce même dossier se trouve **answer.txt** contenant le code complet produit par le professeur.

Les divers scripts ajoutés dans le dossier de la tâche seront décrits dans le chapitre décrivant la création d'une tâche. Ceux-ci sont, tout comme dans **htmlgenerator.py**, construits en fonction des paramètres reçus afin de faire correspondre l'exercice aux spécifications encodées.

Si Pythia devait rencontrer une erreur lors de la compilation et création de l'exercice, un message d'erreur serait renvoyé et le processus serait interrompu.

En cas de réussite, à la fin du processus, il y aura à la fois des fichiers stockés côté serveur (description, paramètres pour la machine virtuelle, réponse,...) et à la fois des informations enregistrées en base de données.

⁶ Unique ID

5. Modify.py

Modify.py ne permet pas de modifier l'entièreté d'un exercice. En effet, un exercice étant compilé par Pythia, les fichiers sont alors dans le système. Seules les valeurs présentes en base de données et l'énoncé sont modifiables. Modifier le code de l'énoncé pourrait engendrer des incohérences. Quand un étudiant réussit un exercice, si ensuite le professeur le modifie, l'étudiant a-t-il toujours réussi l'exercice ? Sa soumission indique oui alors qu'actuellement, son implémentation n'est peut-être plus correcte.

Le déroulement, quant à lui, est assez similaire à la création d'un cours. Les vérifications sont identiques. Il y a également la possibilité de changer l'état d'un cours. Pouvoir passer le cours de 'public' à 'caché' ou encore 'supprimé'.

6. Submissions.py

Après avoir reçu la réponse de Pythia concernant une implémentation pour un exercice donné, **server.py** délègue les tâches à **submissions.py**.

La fonction **checking()** va permettre à la fois de regarder si effectivement l'étudiant a réussi l'exercice mais également d'automatiser une sauvegarde, ceci afin de pouvoir toujours retrouver la dernière soumission faite. La réponse de Pythia sera décrite plus loin. Cette fonction-ci renverra un message adéquat à la réussite ou non de l'apprenant.

Après cela, la fonction **submitted()** est appelée, elle se charge de stocker en base de données l'identifiant unique de la soumission, l'utilisateur de la soumission, la date ou encore l'état. Le contenu de la tentative, lui, est stocké côté serveur de la même manière que la description de l'énoncé.

Le chemin du fichier de la soumission unique est comme suit :

`./SUBMISSIONS/ID_COURSE/ID_TEACHER/ID_EXERCICE/ID_USER/ID_SUBMISSION.txt`

Dans ce même dossier se trouve le fichier **saved** comprenant la dernière sauvegarde de l'utilisateur.

2.3.3.3 Pythia

Le principal composant de mon produit final est bien évidemment la plate-forme Pythia. Celle-ci permet de créer des machines virtuelles avec un environnement spécifique afin de pouvoir ensuite les exécuter de pair avec les tâches, qui correspondent aux exercices.

1. Création d'un environnement

La tâche fastidieuse qu'est la conception d'un système de fichiers simple (Simple File System) implique de trouver les paquets nécessaires et toutes les dépendances pour l'environnement.

Dans le cas de l'environnement adéquat pour ma plate-forme, il me fallut trouver les paquets pour installer Python et Java. Les UML⁷ de base sur Pythia, sont d'origine pratiquement vides, ne contiennent que le strict minimum du noyau Linux, ceci afin d'avoir des machines les plus légères possible, permettant notamment de les lancer très rapidement.

Lors du MAKE, les différents paquets ajoutés dans le fichier de configuration seront installés sur le noyau Linux. Après cela, notre file system est opérationnel ! Il est donc possible d'exécuter des commandes Python ou Java.

```
1  # Copyright 2013 The Pythia Authors.
2  # This file is part of Pythia.
3  #
4  # Pythia is free software: you can redistribute it and/or modify
5  # it under the terms of the GNU Affero General Public License as published by
6  # the Free Software Foundation, version 3 of the License.
7  #
8  # Pythia is distributed in the hope that it will be useful,
9  # but WITHOUT ANY WARRANTY; without even the implied warranty of
10 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
11 # GNU Affero General Public License for more details.
12 #
13 # You should have received a copy of the GNU Affero General Public License
14 # along with Pythia. If not, see <http://www.gnu.org/licenses/>.
15
16 # JAVA JDK 6
17 install_debs openjdk-6-jdk openjdk-6-jre-lib default-jre
18
19 # Base libraries
20 install_debs libc6 libx11-6 libc-bin libgcc1 zlib1g binutils
21 install_debs libasound2 libxext6 libxi6 libxrender1 libxtst6 openjdk-6-jre-headless
22
23 # Python 3.1
24 install_debs python3.1 python3-minimal python3.1-minimal
25
26 # Additional libraries
27 install_debs libexpat1 libssl0.9.8 zlib1g libbz2-1.0 libdb4.8 libncursesw5 \
28             libreadline6 libncurses5 readline-common libsqlite3-0 mime-support
29
30 # busybox
31 install_busybox
```

⁷ User Mode Linux

2. Création d'une tâche

Le dossier d'une tâche contient une multitude de fichiers que je vais vous décrire et qui sont également conçus par **create.py**, précédemment expliqué. L'entièreté de ces fichiers se trouve, après le MAKE, dans le file system de la tâche. Il y a donc bien deux files system : un pour l'environnement et un pour la tâche.

- *control*

Premier fichier de la tâche, celui-ci contient l'unique ligne permettant l'exécution de la tâche. En effet, par un simple chemin, il appelle le fichier Shell contenant toutes les instructions Unix à exécuter à la chaîne.

Par exemple pour mes tâches génériques : **/ephec.sh**

- *ephec.sh*

Le fichier central de la tâche, il exécutera chaque fichier script ou fichier Java en fonction de l'existence de l'un ou l'autre fichier. J'explique ceci à l'aide d'une capture d'écran.

```
1  #!/bin/sh
2  export PATH=$PATH:/usr/lib/jvm/java-6-openjdk/bin/
3
4  mkdir /tmp/work
5  cp /task/files.py /tmp/work/files.py
6  cp /task/student.py /tmp/work/student.py
7  cp /task/errors.py /tmp/work/errors.py
8  cp /task/run-errors.py /tmp/work/run-errors.py
9  cp /task/input.rules /tmp/work/input.rules
10 cp /task/student.txt /tmp/work/student.txt
11 cp /task/Teacher.class /tmp/work/Teacher.class
12 cp /task/Problem.class /tmp/work/Problem.class
13 cd /tmp/work
14
15
16 python3 student.py
17
18 if [ -s warnings.txt ]; then
19     cat warnings.txt
20     echo "hello"
21 else :
22     python3 files.py
23     javac MyClass.java 2> errors.txt
24     if [ -s errors.txt ]; then
25         python3 errors.py
26         cat translated.txt
27     else :
28         java Problem 2> errors.txt
29         if [ -s errors.txt ]; then
30             python3 run-errors.py
31             cat translated.txt
32         fi
33     fi
34 fi
```

Les premières lignes sont simples. L'UML ne trouvant pas automatiquement JAVA dans son répertoire, je rajoute un chemin d'accès à celui déjà existant.

En ce qui concerne les droits d'exécution, la machine virtuelle ne peut pas exécuter les commandes n'importe où. Elle doit donc créer un dossier temporaire, y déplacer tous les fichiers. Ensuite seulement, viennent les étapes de la tâche.

- *Student.py*

Premier fichier script de la tâche, ce script est un choix de ma part. J'ai opté pour une pré-vérification de compilation en vérifiant si l'étudiant n'avait pas placé des identifiants interdits ou inutiles tels que `System.out.println` ou encore des zones de commentaires.

Je vérifie également que l'utilisateur a bien commencé à implémenter la classe ou la fonction si cela est demandé. La pré-vérification permet d'éviter de lancer le compilateur JAVA qui, lui, est bien plus lent. En cas d'erreurs détectées, elles sont placées dans le fichier **warnings.txt**

- *Warnings.txt*

Si ce fichier est inexistant, le processus passe à la suite. S'il existe, le contenu (écrit sous forme JSON) va être renvoyé par Pythia au serveur web, qui à son tour le renverra au frontend de l'utilisateur. Pour que du contenu soit renvoyé, il suffit de l'envoyer sur l'output, c'est pour cela que j'utilise la commande `cat`.

- *Files.py*

Second script de la tâche. Sa principale fonction est de placer le contenu de l'étudiant dans l'éventuel contenu que l'enseignant avait défini. La chaîne de caractères : « `@@@xxx@@@` » permet d'indiquer où le code de l'étudiant sera inséré.

```
1  class MyClass {
2
3  static int myFunction(int x){
4
5
6  @@@xxx@@@
7
8
9
10 }
11 }
12
13
14 }
```

Implémentation professeur



```
int sum = 0;
for (int i = 0; i<x;i++)
{
    sum += i;
}

return sum;
}
```

Soumission envoyé par l'étudiant



```
1  class MyClass {
2
3  static int myFunction(int x){
4  int sum = 0;
5  for (int i = 0; i<x;i++)
6  {
7      sum += i;
8  }
9
10 return sum;
11 }
12
13
14 }
```

Soumission assemblée, envoyée à la compilation

- *Errors.txt*

Les éventuelles erreurs engendrées lors de la compilation du nouveau fichier Java sont stockées à leur tour. En cas d'erreur, c'est le fichier **errors.py** qui est appelé alors que dans le cas contraire, on exécute le fichier **class** créé lors de la compilation réussie.

- *Errors.py*

Ce fichier script se charge d'analyser chaque ligne d'erreur générée lors de la compilation. Si possible, il les retranscrit en français et donne des petites astuces de temps à autre. De la même manière que pour les erreurs de pré-compilations, Pythia les renvoie sous format JSON.

Vu le grand nombre d'erreurs possibles, je n'ai pas eu le temps d'ajouter chaque erreur à mes scripts. Les principales y sont, il est cependant possible que d'autres soient ajoutées prochainement.

- *Run-errors.py*

À l'instar du fichier précédent, **run-errors.py** analyse ligne par ligne les erreurs générées cette fois-ci lors de l'exécution. Il est donc également probable que toutes les erreurs ne soient pas traitées.

Rappelons le fait que ce script n'est appelé que si des erreurs d'exécutions ont eu lieu. Dans le cas contraire, le processus arrive à la fin du fichier **ephec.sh** et Pythia renvoie alors uniquement l'output renvoyé par le fichier JAVA qui stipule si oui ou non, les bonnes réponses ont été trouvées.

Le code étudiant et le code professeur sont soumis à des jeux de tests adéquats en fonction des types des arguments. En cas de réponses identiques, la soumission est déclarée réussie.

3. Fichier de configuration

À chaque tâche est associé un fichier de configuration devant être lu par Pythia. Ce fichier est sous format JSON.

```
{
  "environment": "java",
  "taskfs": "04546266456c/h.paul/86cdc5ec2db/86cdc5ec2db.sfs",
  "limits": {
    "time": 10,
    "memory": 32,
    "disk": 50,
    "output": 1024
  }
}
```

Il contient l'environnement à exécuter ainsi que la localisation du file system de la tâche. Les autres paramètres sont ceux qui configurent le côté technique de la machine virtuelle (temps limite d'exécution autorisé, mémoire vive allouée, mémoire physique allouée et le nombre de caractères maximum qu'elle peut renvoyer).

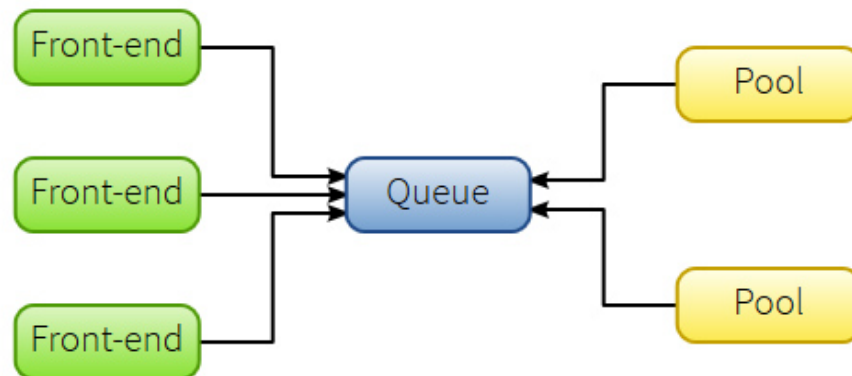
Chaque fichier de configuration associé respectivement à son propre exercice se trouve sur le serveur dans le même dossier que son file system.

Pour bien faire la distinction :

- Le file system de la tâche se trouve côté serveur dans un dossier en dehors de Pythia.
- Le fichier de configuration lié à la tâche est également à cet endroit.
- Le file system de l'environnement Java + Python 3 se trouve, lui, au sein de Pythia.

4. Exécuter Pythia

La plate-forme open-source écoute, par défaut, sur le port 9000. Pour se faire, il faut lancer une queue (une file) et un nombre adéquat de pools, programmes capables de lancer plusieurs machines virtuelles.



Sur cette illustration, le frontend représente les différents utilisateurs passant via le serveur web Bottle. Le serveur se charge d'envoyer le contenu via socket vers Pythia et sa file de tâches.

Chaque tâche est alors distribuée à un pool qui n'est pas occupé. Dans le cas où aucun n'est libre, la tâche patiente jusqu'à pouvoir être transférée. Après traitement par la machine virtuelle, le contenu repasse par la queue qui le renvoie à son tour au serveur.

Lancer ces différents composants est assez simple.

- 1) Se positionner dans le dossier out de Pythia ;
- 2) Lancer la commande pour démarrer la queue, qui écoute par défaut sur le port 9000
`./pythia queue`
- 3) Lancer chaque machine avec la commande suivante :
`./pythia pool --tasksdir : «/SERVER/tasks/ »`

Par défaut, les fichiers de configuration sont au sein de Pythia, or, dans ma solution, je les place sur le serveur en-dehors du dossier principal, tout comme les exercices. En cas de mise à jour de Pythia, les exercices ne seraient donc pas supprimés.

2.3.3.4 Interface Utilisateur – frontend

Ce dernier point concernant ma plate-forme va décrire l'expérience vécue par les différents utilisateurs. Je vais expliquer la quasi-entière des fonctionnalités actuellement disponibles.

Écoutant sur le port 8080, nous commençons notre navigation par la page connexion à l'adresse suivante : 127.0.0.1 :8080 ou localhost:8080. La présentation se fera si possible sur serveur distant.

- *Connexion*

Nous avons alors face à nous le bouton nous connectant avec le serveur LDAP. En fonction des identifiants rentrés et de leur exactitude, nous serons redirigés vers le tableau de bord. En cas d'identifiants erronés, un message pertinent est affiché.

- *Tableau de bord*

La première interface apparaît alors dynamiquement sans chargement d'une nouvelle page HTML. En effet, tout se base sur une unique page dont le contenu change à l'aide de jQuery.

Une barre de navigation est également apparue, celle-ci restera présente tant que l'utilisateur reste connecté.

- *Cours*

Le lien vers les cours présents dans cette barre de navigation permet de pouvoir retourner à tout moment sur la liste des cours disponibles.

Le contenu principal est un tableau avec tri reprenant les cours, stipulant leur titre, leur(s) professeur(s), leur niveau de difficulté. Pour des utilisateurs avec des droits plus élevés, le statut du cours est également visible.

- Un étudiant ne peut voir que les cours auxquels il est inscrit et dont le statut est public ;
- Un professeur peut ajouter un cours et visionner les cours dont il est modérateur. Cependant, les cours supprimés lui seront masqués. Les cours sont modifiables et uniquement supprimables en tant que créateur ;
- Un administrateur peut voir tous les cours, en ajouter, les modifier ou les supprimer.

- *Ajouter un cours*

Cette page n'est visible que pour les enseignants ou les administrateurs.

Le créateur ayant plein droit sur son cours, il doit fournir :

- Le titre : maximum cinquante caractères ;
- Le langage n'est pas modifiable, un langage ne peut s'ajouter que manuellement via le système Pythia et non par ma plate-forme. J'ai donc placé le select dans le but de prévoir un éventuel futur avec d'autres langages sur celle-ci ;
- Une description, elle doit pouvoir décrire en quelques mots le cours, les exercices qu'il contient ;
- Le statut du cours (Public ou Caché) ;
- Ajouter un étudiant manuellement via son matricule, son nom ou son prénom ;
- Un tableau reprenant les étudiants d'ores et déjà ajoutés est affiché (tri autorisé) ;
- Deux tableaux contenant les groupes inscrits et les groupes non-inscrits. Les groupes sont interchangeables entre ceux-ci ;
- Deux tableaux contenant les professeurs modérateurs et les non-sélectionnés. Les enseignants sont interchangeables entre ceux-ci.

En cas d'erreur, un message adéquat apparaît sur la page. Il appartient donc à l'utilisateur de modifier les champs nécessaires.

- *Modifier un cours*

Cette page n'est visible que pour les enseignants modérateurs du cours concerné ou les administrateurs.

Elle est identique à la page de création, si ce n'est qu'elle possède une contrainte.

- Un professeur n'étant pas le créateur du cours ne peut pas modifier la liste des modérateurs.

- *Supprimer un cours et le réafficher*

Cette option retranscrite sous la forme d'un bouton n'est disponible que sur la page des cours et n'est visible que pour les créateurs d'un cours et tous les administrateurs.

Un message de confirmation est demandé pour éviter toute manipulation involontaire.

En cas de suppression effective, le cours est toujours disponible en base de données et sur l'interface de l'administrateur. Dans le cas où l'enseignant souhaite réafficher ce cours, il devra contacter par lui-même l'admin. Ce dernier pourra le faire soit via la page de modification en changeant le statut du cours soit via un bouton. Le cours repasse alors en statut caché.

- *Exercices*

Cet affichage contient tous les exercices disponibles pour un cours spécifique.

Tout comme les cours, ils sont affichés dans un tableau similaire avec une option de tri.

Le contenu du tableau est cependant différent sur un point. Les étudiants ne voyant pas le statut de l'exercice, il est remplacé par le statut de leur éventuelle dernière soumission pour cet exercice-là. Si l'étudiant l'a déjà réussi mais qu'il commet tout de même une erreur lors d'une tentative postérieure, l'exercice sera considéré comme non-réussi. Il faut souligner qu'il n'y a qu'un seul professeur par exercice, bien que l'ensemble des enseignants modérant le cours y ait tous tout de même accès.

- Un étudiant ne pourra voir que les exercices qui sont publics ;
- Les exercices ne sont pas entièrement modifiables (cf. **Modify.py**) ;
- Les exercices sont supprimables. Le processus est identique à celui d'un cours ;
- Un professeur modérant ce cours peut visionner tous les exercices, sauf ceux supprimés. Il peut en ajouter et voit toutes les soumissions du cours ;
- Un administrateur a bien évidemment plein pouvoir.

- *Ajouter un exercice*

À l'instar d'un cours, l'enseignant doit fournir du contenu. Cependant, celui-ci est bien plus précis et spécifique. Un message d'erreur apparaît si nécessaire.

- Un intitulé : cinquante caractères maximum ;
- La classe principale et la fonction principale de l'énoncé ;
Pour ces deux derniers paramètres, le professeur doit décider si oui ou non l'étudiant devra les fournir ou si l'exercice se déroulera à l'intérieur de ceux-ci (cf. les schémas de **Files.py**). Il doit donc cocher ou non les cases en-dessous de chaque option.
- La liste des éventuels arguments de la fonction principale (uniquement fournir le type de l'argument) ;
- Il faut ensuite fournir la configuration pour la machine virtuelle (cf. Fichier de configuration) ;
- Le statut public ou caché ;
- Le niveau de difficulté (de 0 à 5, le pas est de 0.5) ;
- Le contexte qui correspond à l'énoncé de l'exercice. Le textarea est muni de TINYMCE pour pouvoir donner un aspect plus agréable au contexte ;
- Le code complet avec comme nom de classe « Teacher » afin de pouvoir faire la distinction avec la classe de l'étudiant ;
- La partie de ce même code que l'étudiant ne doit pas fournir (cf. les schémas de **Files.py**) ainsi que la chaîne de caractères où le code de l'étudiant doit s'insérer.

- *Soumissions*

Disponible pour tous les professeurs modérant le cours et les administrateurs, cette page permet de visionner toutes les soumissions de chaque étudiant pour l'exercice choisi.

La date de dernière connexion ainsi que les dates de chaque soumission et leur réussite ou non sont affichées.

Le professeur peut alors choisir de télécharger individuellement une tentative de l'étudiant ou plutôt toutes les télécharger sous format ZIP.

Le tableau possède les options de tri et de recherche.

- *Statistiques*

Il est possible, pour les professeurs ayant accès, de visionner quelques statistiques concernant un exercice ou un cours :

- Le nombre d'étudiants inscrits ;
- Le nombre de soumissions pour un cours ou un exercice ;
- Le pourcentage de réussite vis-à-vis des étudiants ;
- Le pourcentage de réussite vis-à-vis des soumissions ;
- ...

- *Exercice*

La page la plus importante est celle où les soumissions vont pouvoir être effectuées.

- Il y a la possibilité de masquer l'énoncé afin d'éviter d'avoir l'affichage trop encombré ;
- L'option « sauvegarder » est disponible ;
- Chaque envoi engendre une sauvegarde ;
- Un professeur peut charger les soumissions d'un étudiant donné ;
- On peut changer à tout moment le thème de son éditeur. Le choix est stocké dans un cookie valable sept jours ;
- Zone de messages pour les erreurs, les astuces et les conseils.

3. Difficultés rencontrées

- La première difficulté rencontrée fut tout simplement l'environnement. J'étais dans l'incapacité d'installer une machine virtuelle ou un dual boot sur ma machine. Mon ordinateur bloquant les démarrages autres que sur l'OS actuel et la lenteur considérable de la virtualisation m'empêchèrent de commencer assez rapidement mon travail. Pendant ce temps, je fis des recherches de documentations qui me permirent de mieux débiter une fois paré. En effet, après plusieurs journées de tentatives, je finis par réussir à débloquent l'UEFI⁸ pour enfin installer un Linux me permettant de débiter mon travail.
- J'ai été pendant de longues semaines dans le doute concernant les utilisateurs de ma plate-forme. Fallait-il des nouveaux comptes ? Pourrais-je accéder à l'Active Directory ? Des incertitudes assez déconcertantes sur le long terme.
- Tous les modules ne sont pas optimaux sur Python 3, par exemple, le module LDAP est assez brouillon et ne permet pas de faire ce que d'autres modules permettent. Or, j'ai cherché pendant des heures avant de trouver comment faire. La faute à quoi ? Trop peu de documentations. J'ai donc contacté moi-même le créateur du module, mail auquel il m'a répondu avec beaucoup de sympathie.
- Pythia étant assez peu connu, les erreurs rencontrées et les messages d'erreurs inconnus au web furent difficiles à déchiffrer.
- Je n'ai utilisé un gestionnaire de version que très tardivement. De ce fait, j'ai pris plus de temps à réécrire quelque chose de perdu alors qu'avec cet outil, cela aurait été bien plus rapide.

⁸ **U**nified **E**xtensible **F**irmware **I**nterface définit un logiciel intermédiaire entre le micro logiciel (firmware) et le système d'exploitation (OS) d'un ordinateur. Cette interface succède sur certaines cartes-mères au BIOS.

4. Améliorations

- Actuellement, la création est générique. Elle ne permet donc que des exercices simples. Il est cependant possible de créer manuellement des exercices plus complexes en écrivant les scripts soi-même. Une amélioration à apporter serait donc d'élargir cette genericité.
- La modification technique des exercices.
- Ajouter plus tard d'autres langages à la plate-forme serait pour moi une réelle réussite.
- Plus de statistiques pertinentes.
- Un module anti-plagiat pour des exercices plus importants.
- Pouvoir importer des librairies.
- Pouvoir importer d'autres classes.
- Mettre des contraintes temporelles ou des contraintes sur le nombre de soumissions.
- Rendre les exercices un peu plus ludiques, plus attrayants.

5. Conclusion

Après de nombreuses heures de programmation, je suis toujours aussi content de ma proposition de TFE. J'aime en effet énormément l'idée d'avoir la possibilité de fournir un outil à l'école où j'ai appris pas mal de choses dans certains domaines.

Dû au fait que leur produit final ne sera peut-être jamais utilisé, plusieurs étudiants de mon année trouvent moins de motivation à réaliser leur travail. Pour ma part, savoir que des étudiants auront peut-être la possibilité d'utiliser ma plate-forme est une motivation supplémentaire. Cela me donne envie d'apporter de nombreuses possibilités à mon produit.

Après la remise finale de mon travail, j'espère garder contact avec les enseignants étant intéressés par mon travail. J'espère que ceux-ci me contacteront afin de pouvoir apporter telle ou telle correction. Il est en effet possible que mon produit ne soit pas optimal pour tous les exercices qu'ils souhaitent proposer à leurs étudiants. Voir mon travail être intégré à l'enseignement de la programmation à l'EPHEC serait pour moi une réelle réussite.

En ce qui concerne les acquis, j'ai beaucoup appris notamment dans l'utilisation de Linux et ses terminaux Unix. Python étant un nouveau langage pour moi, je peux dire que ce langage est désormais mon préféré, celui que je prônerais dans le cas de conceptions d'applications. Malgré la découverte d'un nouveau type de serveur web, je maintiens ma préférence pour APACHE et le PHP en tant que langage de programmation.

Pythia reste par-dessus tout un projet d'avenir. En effet, suite à mon grand dévouement lors de mon stage concernant son amélioration et ma motivation à domicile pour mener à bien mon travail de fin d'études, Monsieur Combéfis souhaite garder contact avec moi afin de pouvoir, plus tard, établir une future collaboration dans le but de développer davantage Pythia et ses composants.

Un autre véritable acquis est tout simplement la gestion du temps et le respect d'un planning. J'ai pris soin de bien respecter mes échéances personnelles concernant des parties plus ou moins importantes de mon projet, quitte à travailler bien plus tard certains jours. Pour la première fois, je n'ai pas senti un réel manque de temps ou du stress pour la remise d'un travail.

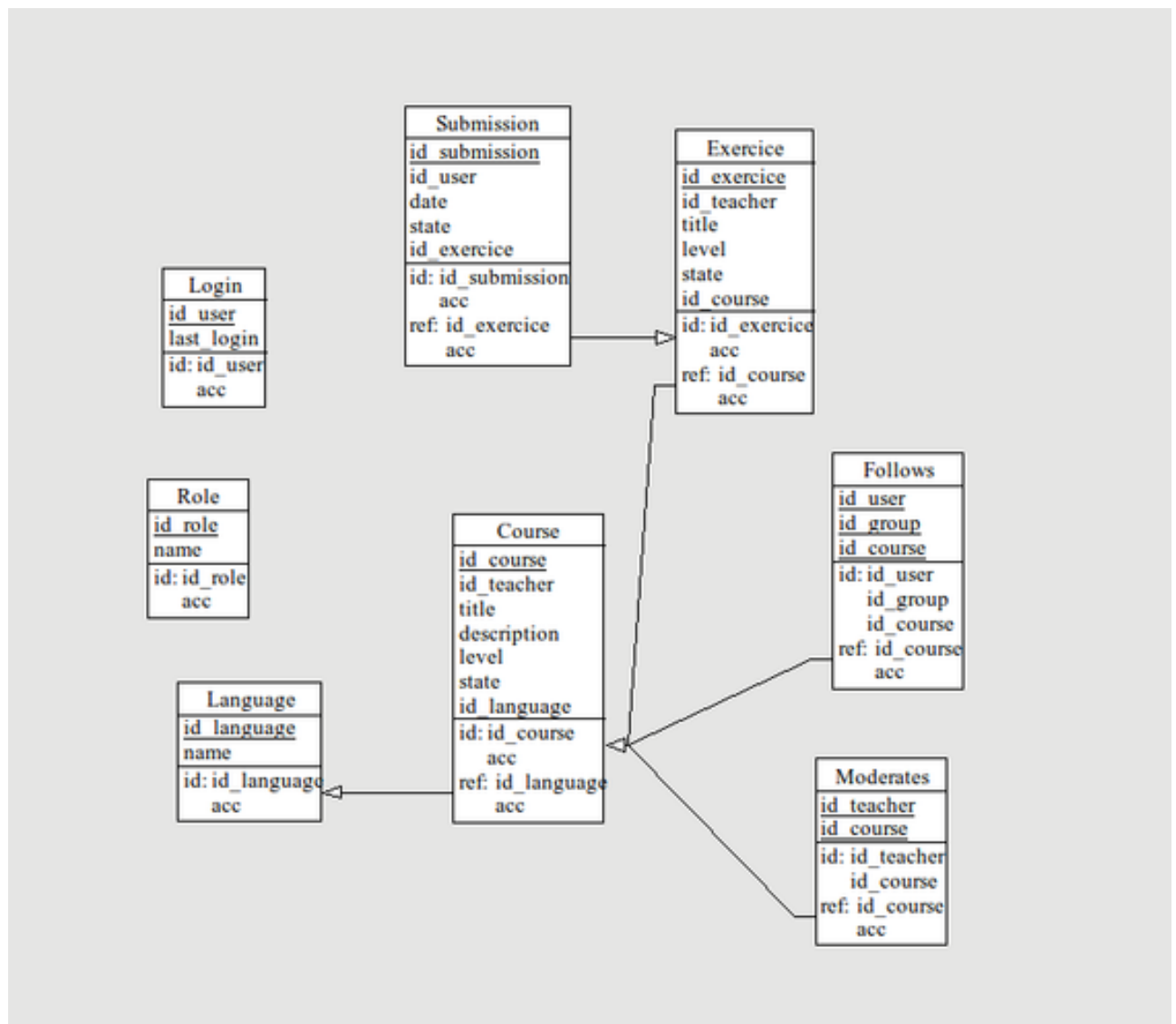
Pour terminer, après ce stage fructueux et ce travail de fin d'études captivant, je suis d'ores et déjà certain de vouloir faire de la programmation mon futur métier. Je possède une véritable passion pour la conception de programmes, d'algorithmes ou encore la résolution de problématiques.

6. Webographie

- ❖ The Pythia Project – www.pythia-project.org (consulté en février 2014)
- ❖ Building an Address Book with OpenLDAP
www.onlamp.com/pub/a/onlamp/2003/03/27/ldap_ab.html
(consulté en mai 2014)
- ❖ How to create a new user in OpenLDAP
mindref.blogspot.be/2010/12/openldap-create-user.html
(consulté en mai 2014)
- ❖ Bottle: Python Web Framework – bottlepy.org/docs/dev/index.html
(consulté en 2014)
- ❖ davispuh/MySQL-for-Python-3 - <https://github.com/davispuh/MySQL-for-Python-3>
(consulté en mars 2014)
- ❖ Welcome to Python.org - <https://pypi.python.org/> (consulté en 2014)
- ❖ Stack Overflow - stackoverflow.com (consulté en 2014)
- ❖ MySQL Documentation - dev.mysql.com/doc/ (consulté en mars 2014)
- ❖ jQuery UI - jqueryui.com (consulté en mai 2014)
- ❖ jQuery API Documentation - api.jquery.com (consulté en mai 2014)
- ❖ W3Schools Online Web Tutorials - w3schools.com (consulté en mai 2014)
- ❖ Mottie/tablesorter - <https://github.com/Mottie/tablesorter> (consulté en mai 2014)
- ❖ CodeMirror - codemirror.net (consulté en février 2014)
- ❖ Bootstrap - getbootstrap.com (consulté en mars 2014)
- ❖ RegExr: Learn, Build, & Test RegEx - regexr.com (consulté en 2014)

7. Annexes

I. Schéma relationnel



II. Diagramme d'activité : Soumission d'un étudiant

