Haute Ecole Léonard de Vinci

INSTITUT PAUL LAMBIN

Associé à l'Université Catholique de Louvain

Département d'informatique

Design et développement d'un système de monitoring et d'un studio de création de tâches pour Pythia, une plateforme d'évaluation de codes

Mémoire présenté par

Maton, Anthony

en vue de l'obtention du grade
de bachelier en informatique
de gestion

Année académique 2016-2017

Table des matières

1 Remerciements

Je tiens à remercier les personnes suivantes pour leurs contributions significatives à la réalisation de ce travail.

1.1 Dr Sébastien Combéfis

Pour m'avoir accueilli dans son équipe et permis de travailler sur un projet enrichissant dans un environnement agréable et ce avec tout le soutien dont j'ai eu besoin.

1.2 Grégory Seront

Pour m'avoir soutenu et guidé dans la réalisation de mon travail et veiller au bon déroulement de mon stage.

1.3 Ian Childress

Pour m'avoir prodigué de nombreux conseils sur le fonctionnement du langage Go et la philosophie derrière ce langage.

2 Introduction

2.1 Contexte

Dans le cadre de ma troisième année d'études à l'Institut Paul Lambin, dans le cadre du Bachelier en Informatique de Gestion, j'ai eu le privilège de réaliser un stage de seize semaines dans une entreprise afin d'y mettre en pratique les compétences acquises à l'Institut.

Pour choisir l'entreprise en question, j'ai axé ma recherche sur la volonté de travailler sur un projet aux sources ouvertes et de préférence écrit dans un langage différent du Java afin de découvrir un nouveau langage et une autre facette du métier.

Je suis donc allé voir le Dr Sébastien Combéfis, professeur à l'École Centrale des Arts et Métiers que j'avais eu l'occasion de rencontrer à de multiples reprises lors des conférences qu'il a données à l'Institut l'an passé conférences au cours desquelles, nous avions l'occasion de découvrir de nouvelles choses sous des angles différents et qui m'avait donc donné envie de travailler avec lui.

Après une conversation avec lui, il me présenta son projet et la perspective de travailler sur une application à visée pédagogique me séduit immédiatement et c'est ainsi que je trouvai l'endroit où passer mes seize semaines de stage.

2.2 Présentation de l'Entreprise

L'École Centrale des Arts et Métiers est une école d'ingénieurs certifiée par la Commission des Titres d'Ingénieurs et part de la Haute Ecole Léonard de Vinci. Elle propose des cursus de Bachelier et de Master en Ingénerie industrielle.

J'ai pour ma part travaillé au sein du département Génie Eléctrique et informatique et plus précisément au sein du Bureau Master Informatique.

3 Objectifs

Depuis quelques années au sein du département Informatique de l'ECAM, il y'a une volonté de proposer une introduction à la programmation la plus interactive possible en se basant sur une méthode pédagogique adaptée et le recours aux technologies de l'information.

C'est pour cela qu'a été développée la plateforme Pythia qui permet aux professeurs de créer des cours de programmation interactifs et aux étudiants de suivre ces cours et de réaliser des exercices interactifs tout au long du cours et d'obtenir un retour clair sur leur prestation instantanément.

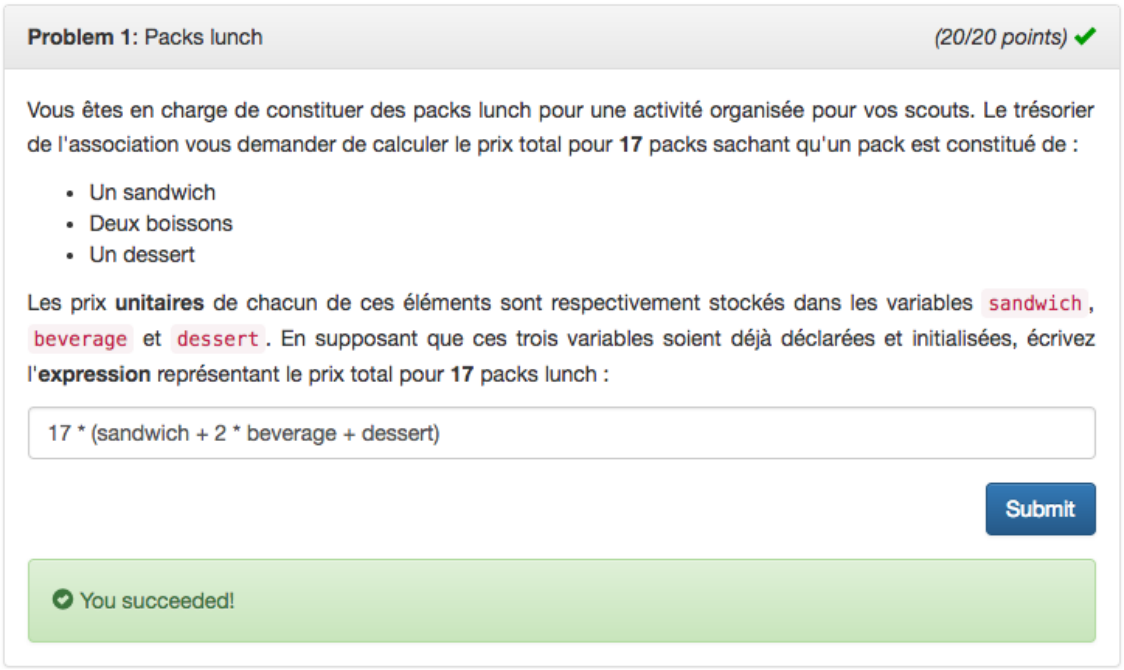

FIGURE 1 – Exemple de question interactives.

L'objectif qui a été défini avec le Dr Combéfis était de réaliser diverses tâches sur Pythia dans un but d'amélioration de la qualité et d'itération continue sur le logiciel. Et ce principalement, pour rendre le logiciel plus résistant à la charge et aux pannes ainsi qu'à faciliter l'ajout de nouveaux modules de cours par les professeurs.

3.1 Objectifs spécifiques

En complément de tendre vers, une amélioration continue dans le but d'atteindre l'objectif global de la plateforme ; plusieurs objectifs concrets m'ont été confiés.

3.1.1 Analyse, documentation et prise en main du processus de création d'environnements

Le processus de création d'environnements est actuellement non documenté et compliqué à mettre en oeuvre.

Dans le cadre de cet objectif, mon travail consiste en l'analyse du processus et en la réalisation d'une documentation sur la création d'environnements, ainsi qu'en la préparation du terrain, pour la réalisation de l'objectif spécifique "Création d'un outil de gestion d'environnements et de tâches".

Les produits cet objectif sont une documentation permettant la création d'un environnement, afin de vérifier la validité et l'exhaustivité de cette documentation, j'ai créée un environnement pour un nouveau langage, le langage Octave en suivant pas à pas les instructions de cette documentation.

3.1.2 Modernisation de l'environnement de développement

L'environnement de développement réclame une machine sous Linux avec une série de dépendances ainsi qu'une version spécifique de make, cela rend la contribution au projet compliquée et nécessite l'installation d'un environnement spécifique au risque de casser le fonctionnement d'autres projets en installant la version spécifique de make par exemple.

Dans le cadre de cet objectif, mon travail consiste en la recherche d'une solution permettant la contribution depuis un maximum de plateformes et dans un environnement le moins dépendant du système possible.

En effet, la plateforme ayant recours à "User Mode Linux" [1], il faut donc une chaîne de compilation permettant la compilation d'un noyau Linux et les utilitaires liés à cet environnement ainsi qu'à la création d'un système de fichiers en lecture seule.

Les produits de cet objectif sont une solution permettant le développement depuis n'importe quel système d'exploitation ainsi qu'une documentation permettant l'usage de ce toute nouvelle documentation.

3.1.3 Création d'un outil de supervision de processus

La Queue et les Pools [2] sont les deux éléments principaux nécessaires au fonctionnement de l'application. En effet, la Queue sert à répartir les tâches sur les différents Pools qui sont quant à elles chargées de superviser l'exécution des tâches sur les machines virtuelles. Ces deux composants sont donc critiques dans le cadre de notre application.

Or le fonctionnement de la Queue et des Pools n'est actuellement pas supervisé, il est donc nécessaire de réfléchir à la mise en place d'un système de supervision, et ce afin d'éviter des interruptions de services ou une absence de contrôle sur la montée en charge du système.

Dans le cadre de cet objectif, je dois réaliser des recherches concernant les possibilités d'analyse du fonctionnement de la Queue et des Pools en temps réel.

1. "User Mode Linux est une façon sécurisée de lancer une variété de versions de Linux et de processus sans aucun risque pour l'environnement Linux principal." Traduit depuis `http://user-mode-linux.sourceforge.net/`
2. Le fonctionnement de la Queue et de la Pool sont détaillés dans le point Pythia Core

Les produits de cet objectif sont une amélioration du code existant afin d'ajouter la possibilité de collecter l'état de l'application Pythia Core, ainsi que la réalisation d'une solution de monitoring et de lancement des processus sur base d'un fichier de configuration.

3.1.4 Création d'un outil de gestion d'environnements et de tâches

Les tâches[3] sont les exercices proposés aux étudiants, elles sont composées de multiples fichiers et d'une architecture particulière à suivre.

L'ajout et la suppression de tâches sont donc actuellement deux processus fastidieux, il est donc opportun afin de permettre un large usage de la plateforme par un public d'horizon divers de simplifier ce processus au maximum.

Dans le cadre de cet objectif, je vais réaliser une phase exploratoire dans le but de déterminer les possibilités de simplification et travailler en vue de leurs mises en oeuvre.

Les produits de cet objectif sont la réalisation d'un modèle de données afin de représenter les tâches et les environnements ainsi qu'un outil de gestion de tâches et d'environnements.

3. Le fonctionnement et l'architecture des tâches est détaillé dans le point Pythia Core

4 Contexte de travail

L'environnement de travail tant technique que social dans lequel j'ai eu l'occasion d'évoluer était particulièrement dynamique et agréable. J'ai pu bénéficier de leurs expériences tout au long du stage tout en ayant l'opportunité d'expérimenter et de découvrir de nouvelles choses, avec une vraie volonté de la part de l'équipe de compléter mon apprentissage et de m'inciter à la créativité et à la recherche de solutions nouvelles.

4.1 Le bureau Master Informatique

Le bureau Master Informatique de l'ECAM est le lieu où le coeur du développement de Pythia se produit et le lieu où j'ai passé ces seize semaines de stage.

4.1.1 Sébastien Combéfis

Le Dr Sébastien Combéfis a été mon responsable au sein de l'entreprise et principal développeur de l'application Pythia. C'est donc avec lui qu'ont été définis les tâches à accomplir, leur ordre de priorité ainsi que la façon de procéder. Il m'a aussi fortement aidé à prendre la plateforme en main ainsi qu'à comprendre pleinement son architecture et les technologies utilisées.

4.1.2 Quentin Lurkin

Quentin Lurkin est l'un des quatre autres enseignants de l'équipe informatique. Il est également responsable du réseau et des comptes utilisateurs au sein de l'ECAM. Il m'a aidé à comprendre comment Pythia était utilisé au sein de l'ECAM et à mieux cerner les apports possibles à la plateforme.

4.1.3 Damien Vanhove

Damien Vanhove est un étudiant travaillant sur un autre projet interne à l'ECAM dans le cadre de son travail de fin d'études. Il était régulièrement présent dans le bureau et m'a été d'une grande aide pour comprendre certaines subtilités liées à Angular.

4.1.4 Sabine Lebon

Sabine Lebon est la coordinatrice des services informatiques pour l'ECAM et l'Institut Libre Marie Haps, elle travaille dans le bureau Master Informatique.

4.2 L'environnement informatique

Le choix de l'environnement de travail m'a été laissé libre, j'ai donc travaillé avec un MacBook et OS X.

En effet, le Dr Combéfis a accordé une grande importance au fait que je puisse utiliser les outils avec laquelle j'étais déjà familier.

Concernant les langages de programmation utilisés, ceux-ci ont été nombreux et variés en fonction des parties de la plateforme sur lesquelles j'ai opérés.

Sur le core de Pythia et le Watchdog, j'ai travaillé avec le langage Go. Pour la plateforme Pythia Studio, j'ai travaillé avec Node.JS et Angular2 et MongoDb avant de basculer le développement sur du Python3 et le cadre applicatif avec une base de données Sqlite. La partie gestion des machines virtuelles et de création d'environnements est composée quant à elle de scripts shell.

Enfin, le code produit était versionné à l'aide de l'outil de versionnement de code Git et hébergé sur la plateforme Github.

5 L'application Pythia

L'application Pythia est en réalité un ensemble d'applications qui interagissent les unes avec les autres pour former la plateforme Pythia.

Nous y trouvons un système de gestions de tâches, une plateforme pour créer et suivre les cours, un dépôt de machines virtuelles nécessaires à l'exécution des réponses des étudiants, des librairies spécifiques à chaque langage pour exécuter les tests et enfin un studio de création de tâches et d'environnements.

La plateforme Pythia repose sous l'ombrelle de l'organisation Github Pythia Project (`https://github.com/pythia-project`), la page web officielle est disponible à l'adresse `http://www.pythia-project.org/` (celle-ci sera bientôt mise à jour).

5.1 Pythia Core

Pythia Core est comme son nom l'indique l'application au coeur de la plateforme. Il s'agit de l'application chargée de l'exécution des tâches. Cette application est composée de trois composants clés servant à assurer une répartition équilibrée de la charge entre plusieurs machines et une grande vitesse de réaction.

Le travail réalisé sur Pythia Core rentre dans le cadre de la réalisation des objectifs spécifiques "Modernisation de l'environnement de développement" et de "Création d'un outil de supervision de processus".

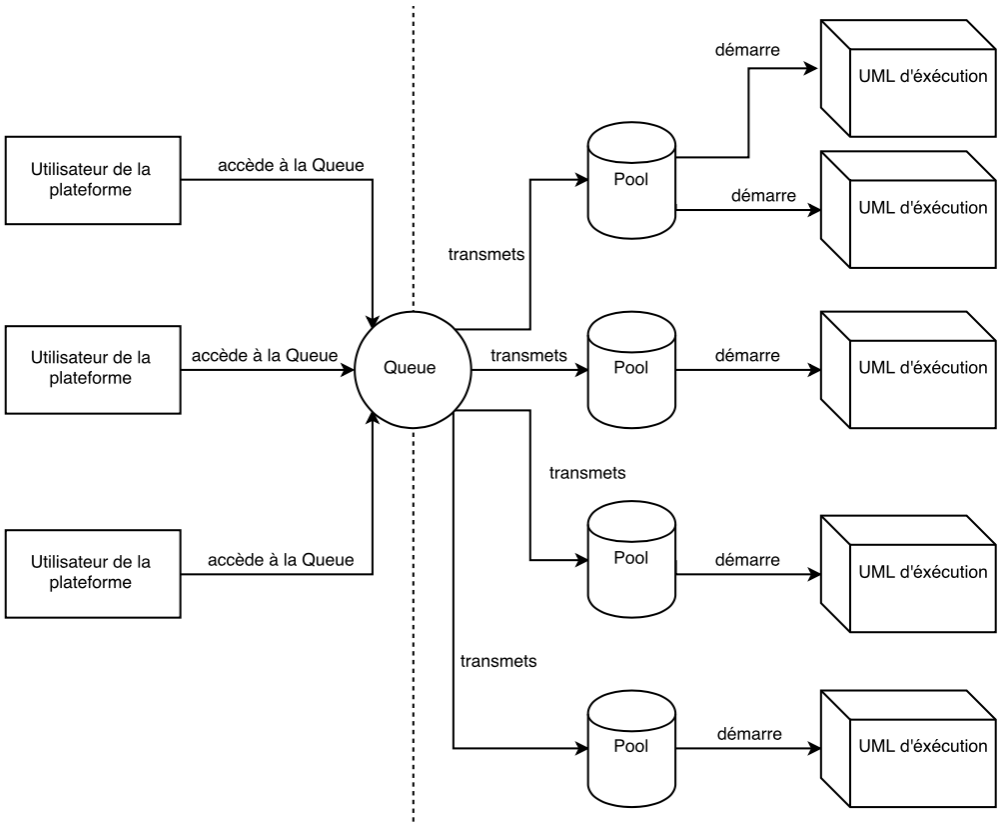

FIGURE 2 – Architecture haut niveau du coeur de la plateforme Pythia

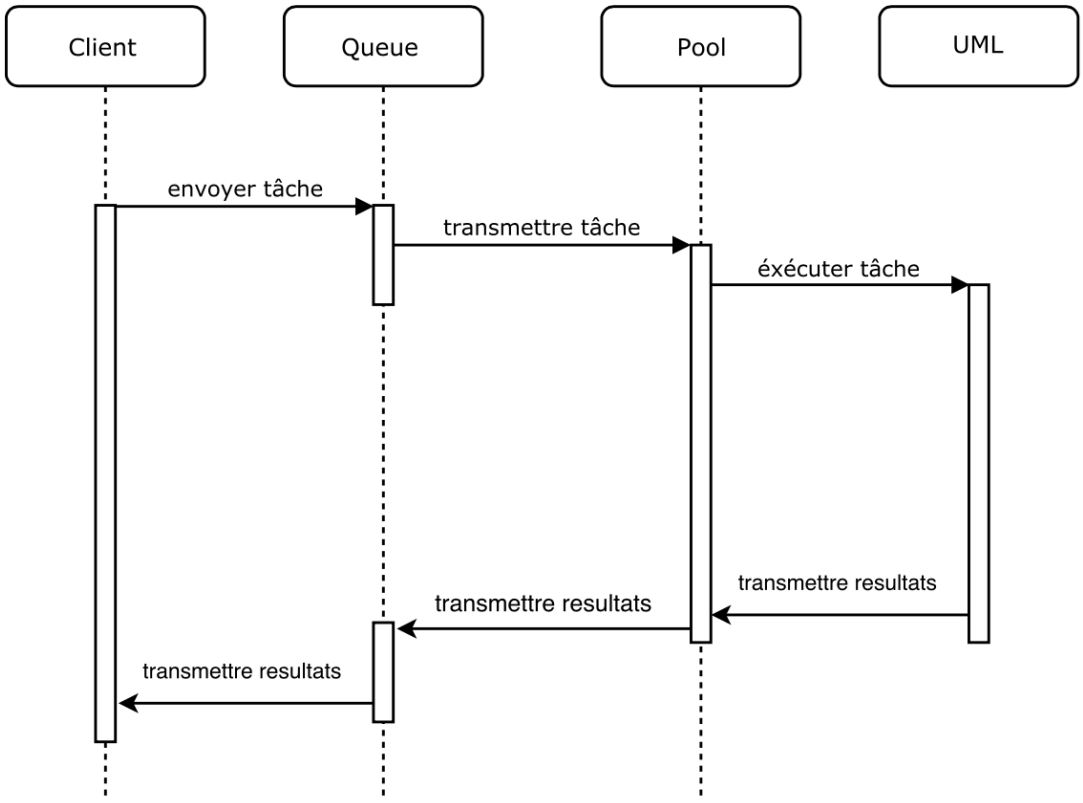

FIGURE 3 – Diagramme de séquence du traitement d'une tâche

L'on peut voir sur le diagramme de séquence ci-dessus que l'exécution d'une tâche est transversale et passe par chaque élément de Pythia Core. Chacun de ces éléments est interconnecté à l'aide de sockets TCP afin d'assurer la fiabilité de la connexion et des données transmises, à l'exception de l'UML d'exécution[4] qui est crée directement par le Pool[5] Et dont les résultats sont récupérés par le Pool.

L'on voit aussi que les UML d'exécution n'ont pas de ligne de vie. En effet, les UML d'exécution n'existent que le temps de l'exécution de la tâche et sont détruits immédiatement après dans un souci d'économie des ressources de la machine.

4. Les UML d'exécution sont décrits en détail au point UML d'exécution, de manière simplifiée, il s'agit de la machine virtuelle exécutant le code de l'étudiant

5. Les Pools sont décrits en détail au point Pool, en quelques mots, un Pool est le composant chargé de lancer les machines virtuelles exécutant les tâches et de transmettre les résultats à la Queue.

5.1.1 La Queue

La Queue est la partie principale du Core du Pythia, elle reçoit les demandes de lancement de tâches via un socket et est chargée de répartir ces tâches entre les différents pools de manière équitable afin d'assurer une répartition de charge équilibrée.

Dans son fonctionnement, la Queue fonctionne sur un modèle client-serveur avec souscription de la part des clients. Les clients (les pools) s'enregistrent auprès de la Queue et la Queue leur transmet ensuite les tâches au fur et à mesure qu'elles arrivent selon un principe de round robin.

Ce modèle basé sur la souscription permet l'ajout et le retrait de Pool dynamiquement sans affecter le bon fonctionnement de l'application.

Ajouter ici un exemple de tâches.

La Queue est donc interdépendante avec les Pools.

La Queue est écrite en Golang et prend la forme après compilation d'un binaire compatible avec de multiples plateformes (Linux, OS X, Windows).

1. Travail réalisé

   Dans le cadre de l'objectif spécifique "Création d'un outil de supervision de processus", j'ai été amené à ajouter un point d'accès permettant d'obtenir le statut de la Queue.

   La réalisation de cette tâche a pris un temps certain à cause de ma méconnaissance du langage Go et c'est déroulé en plusieurs étapes.

   — Il a fallu créer la structure du message et comprendre comment réaliser une sérialisation de ce message en JSON. Il s'avère que les structures du Go sont directement inspirées de celles utilisées en C et qu'il m'a donc suffi d'avoir recours à une structure. Quant à la sérialisation vers le JSON, celle-ci repose sur une librairie standard 'encoding/json' qui s'appuie sur des tags, un peu à la manière des annotations que l'on peut trouver en Java ou en C#.

```
type QueueStatus struct {
  Capacity     int            `json:"capacity"`
  Available    int            `json:"available"`
  Clients      []*queueClient `json:"clients, omitempty"`
  Jobs         []*queueJob    `json:"jobs, omitempty"`
  Waiting      *list.List     `json:"waiting"`
  CreationDate time.Time      `json:"creation_date"`
}
```

Listing 1: Structure QueueStatus

   — Une fois la structure attendue connue, il a fallu écrire un test unitaire qui a été peaufiné tout au long de la suite de ce processus afin de tester la fonctionnalité et de s'assurer de son bon fonctionnement.

   — Dans un troisième temps, j'ai découvert qu'il n'était possible d'exporter via 'encoding/json' que les éléments ayant une visibilité "exported", cette visibilité est symbolisée par une lettre majuscule comme première lettre du nom de la variable. Il m'a donc fallu refactorer le code afin de changer ceci.

   — Dans un quatrième temps, nous avons constaté que le langage Go ne sérialisait pas les maps dans la forme attendue et que la forme obtenue n'avait pas le sens sémantique espéré. Nous avons donc décide d'écrire une méthode transformant ces deux maps en slice, et ce afin d'obtenir un json sémantiquement correct.

   J'ai tenté dans un premier temps d'écrire une méthode générique de conversion de map en slices en ayant recours à la réflexivité, mais bien que cela soit intéressant d'un point de

vue académique, cela s'avère compliqué et beaucoup moins performant que d'écrire deux méthodes spécifiques pour réaliser ces tâches pour les deux maps que nous avions.
En effet, la réflexivité en Go est extrêmement coûteuse en termes de ressources et le coût en termes de maintenance et de duplication de code est donc largement compensé par le gain de performance.

```go
func convertMapToSlice(myMap interface{}) (err Error, mySlice []interface{}) {
    reflectedMap := reflect.ValueOf(myMap)
    mapContentType := reflect.TypeOf(myMap).Elem()
    if reflectedMap.Kind() == reflect.Map {
        slice := reflect.MakeSlice(reflect.SliceOf(mapContentType), 0, 0).Interface()
        for index, element := range reflectedMap {
            append(slice, element)
        }
        return nil, slice
    } else {
        return errors.New("The parameter must be of type Map")
    }
}
```

Listing 2: Tentative d'écriture d'une méthode générique de conversion

```go
func convertClientsToSlice(clients map[int]*queueClient) (clientsSlice []*queueClient) {
  clientsSlice = make([]*queueClient, 0)
  for _, element := range clients {
    clientsSlice = append(clientsSlice, element)
  }
  return clientsSlice
}
```

Listing 3: Méthode de conversion

— Enfin, il a fallu ajouter un nouveau message pour la demande de status et une réponse adaptée à cette demande.

5.1.2 Les Pools

Les Pools sont une part importante du Core de Pythia, elles reçoivent les tâches venant du Core et sont chargée de démarrer des UML d'exécutions[6] Afin d'exécuter ces tâches. Elles s'assurent ensuite de transmettre le résultat de la tâche à la Queue qui le retransmettra ensuite au client.

Elles sont un élément crucial dans la robustesse du système, le but de ces pools étant de pouvoir être réparties sur plusieurs serveurs afin de pouvoir tenir la charge en période d'affluence et ainsi de privilégier une évolutivité horizontale plutôt qu'une évolutivité verticale plus coûteuse et limitée par des contraintes physiques.

Les Pools font partie du même binaire que la Queue, mais sont appelés à l'aide d'options de lancement différentes.

5.1.3 Les UML d'éxécution

Les UML d'exécution sont des machines virtuelles extrêmement optimisées contenant le minimum vital nécessaire à l'exécution des tâches.

Elles sont basées sur un noyau Linux auxquels on ajoute l'interpréteur ou le compilateur du langage et les éventuelles librairies utilisées par le cours.

La distribution Linux utilisée est basée sur Busybox et non sur les outils GNU à des fins de performance. En effet, Busybox est une réimplementation de très nombreux utilitaires standard Unix avec une orientation sur la légèreté et l'efficacité.

Cette légèreté est rendue possible par la compilation de ces utilitaires en un seul binaire et la réécriture de ceux-ci en y intégrant diverses optimisations visant principalement le monde de l'embarqué.

Les librairies installées sont obtenues depuis les dépôts Debian, mais sans recourir à un gestionnaire de paquets qui alourdirait la machine virtuelle, les librairies sont ajoutées à la machine virtuelle lors de la création de celle-ci.

En effet, le système de fichiers utilisés sur la machine est de type SquashFS (Squash Filesystem), il s'agit d'un système de fichier en lecture seule, ce qui nous permet d'assurer une facilité de déploiement ainsi qu'une sécurité et une résilience à la triche accrue de par l'impossibilité d'y réaliser des modifications.

Ces machines sont donc précompilées à l'avance et lors de l'exécution d'une tâche, les fichiers propres à la tâche sont montés dans la machine virtuelle à son lancement, et ce afin de préserver les performances les plus importantes possible.

L'objectif étant d'obtenir des machines virtuelles d'un poids inférieur à 50Mb (éventuellement 100Mb pour les plus grosses) et un temps de démarrage inférieur à 5 secondes (éventuellement 10 secondes pour les plus lentes), et ce afin de garantir un résultat rapide et de préserver l'interactivité avec le soumissionnaire.

1. Travail réalisé

   Afin de prendre en main le fonctionnement des UMLs d'exécution, ma première tâche sur ceux-ci fut d'en créer un. J'ai donc créé un environnement d'exécution pour le langage Octave, j'ai aussi documenté la procédure de réalisation d'un environnement. Ce travail rentre donc dans le cadre de l'objectif spécifique "Analyse, documentation et prise en main du processus de création d'environnements".

   La procédure de création d'un environnement consiste en l'écriture des fichiers nécessaires à la construction d'une image Linux permettant de compiler ou interpréter le langage choisi.

6. Le fonctionnement des UML d'exécution est détaillé au point suivant.

La première étape consiste donc à créer un Makefile permettant de créer l'image sur base du script de configuration. Il faut ensuite écrire le script de configuration qui contient le noyau de base à utiliser (actuellement busybox) ainsi que les paquets à installer pour compiler ou interpréter le langage et enfin si nécessaire des commandes à exécuter lors de la création de la machine dans celle-ci. Enfin, il faut ajouter la compilation de cet environnement dans le Makefile général.

Ma seconde tâche relative aux UML d'exécutions a été de réaliser une analyse de sécurité basique afin de vérifier qu'il n'est pas possible d'importer la solution du professeur ou de faire réussir le test automatiquement.

Cette micro-analyse a été réalisée en plusieurs parties, une analyse du fonctionnement théorique du processus a d'abord été menée. J'ai ensuite analysé le code existant mettant en application ce processus. Enfin, j'ai prouvé à l'aide d'une tâche de test que le processus fonctionnait bien comme attendu.

2. Analyse du fonctionnement théorique

   L'intégralité du code soumis par l'étudiant est exécuté dans un environnement virtualisé, ce code est exécuté par un user "slave" non privilégié. La suite de test du professeur est-elle exécutée via un user "master" privilégié.

   L'exécution d'un fichier par master ou par slave est définie à l'aide d'un fichier de contrôle. Ce fichier de contrôle prend la forme suivante

```
1 /task/scripts/preprocess.py
2 /task/scripts/generate.py
3 !/task/scripts/execute.py
4 /task/scripts/feedback.py
```

   Les instructions précéder d'un " !" sont exécutées par master, les autres sont exécutées par slave.

   Les fichiers précédés d'un " !" sont la propriété de "slave" et sont lisibles par slave et master. Les autres fichiers ne sont lisibles que par "master".

   Cette sécurité est assurée à l'aide du système de permissions de Linux.

3. Analyse du fonctionnement existant

   L'analyse du fonctionnement existant a été réalisé sur Pythia-Core tel qu'il était le 3 mars au commit ff873b3 [7]

   Afin de réaliser cette analyse, une procédure de test a été écrite afin de déterminer si les protections mises en places étaient bien fonctionnelles. Cette procédure consiste en la création d'une tâche tentant d'accéder à des fichiers auxquels elle n'a normalement pas accès.

   Après lancement de cette tâche de test, si le résultat est négatif, les fichiers sont bel et bien sécurisés et non accessibles depuis le programme de l'étudiant, dans le cas contraire, une réécriture du code incriminé aurait dû être réalisée afin de s'assurer de la protection des fichiers.

5.1.4 L'environnement de développement

1. Migration vers Docker

7. Le commit complet est disponible à l'adresse suivante : `https://github.com/pythia-project/pythia-core/commit/ff873b3da4c3cea3f0e7b28abe1a1dc57591de18`

```python
import os, sys

def getSolutionAttempt():
    ''' I check permissions on each scripts using os.access '''
    teacherFiles = [
        '/task/scripts/preprocess.py',
        '/task/scripts/generate.py',
        '/task/scripts/feedback.py'
    ]

    for file in teacherFiles:
        if os.access(file, os.R_OK) or os.access(file, os.X_OK):
            return False

    return True
```

Listing 4: Code tentant d'accèder aux solutions.

```python
def getSolutionAttempt():
    ''' This is a placeholder to check against the hack attempt
        It\'s always returning True if the student code returns false
        then there is a security breach
    '''
    return True
```

Listing 5: Code vérifiant que l'accès ait bien echoué

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-
# Course: Pentesting
# Problem: Access solution file
# Execution script

import sys

sys.path.append('/task/static')
from lib import pythia

# Redirect stdout and stderr
sys.stdout = open('/tmp/work/output/stdout', 'w')
sys.stderr = open('/tmp/work/output/stderr', 'w')

# Try to import student's code
sys.path.append('/tmp/work')
try:
  import test1
except Exception as e:
  print(e, file=sys.stderr)
  sys.exit(0)

class TaskTestSuite(pythia.TestSuite):
  def __init__(self):
    pythia.TestSuite.__init__(self, '/tmp/work/input/data.csv')

  def studentCode(self, data):
    return test1.getSolutionAttempt()

  def parseTestData(self, data):
    return data[0]

TaskTestSuite().run('/tmp/work/output', 'data.res')
```

Listing 6: Code servant à l'éxécution de la tâche

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-
# Course: Apprendre Python
# Problem: Pentesting
# Feedback script

import ast
import csv
import json
import os
import sys

sys.path.append('/task/static')
from lib import pythia

import math

def getSolutionAttempt():
  return True

class TaskFeedbackSuite(pythia.FeedbackSuite):
  def __init__(self, config):
    pythia.FeedbackSuite.__init__(self, '/tmp/work/output/stderr', None,
      '/tmp/work/input/data.csv', '/tmp/work/output/data.res', config)

  def teacherCode(self, data):
    return getSolutionAttempt()

  def parseTestData(self, data):
    return data[0]

# Retrieve task id
with open('/tmp/work/tid', 'r', encoding='utf-8') as file:
tid = file.read()
output = {'tid': tid, 'status': 'failed', 'feedback': {'score': 0}}

# Read test configuration
config = []
with open('/task/config/test.json', 'r', encoding='utf-8') as file:
content = file.read()
config = json.loads(content)
config = config['predefined']
(verdict, feedback) = TaskFeedbackSuite(config).generate()
output['feedback'] = feedback
output['status'] = 'success' if verdict else 'failed'

print(json.dumps(output))
```

Listing 7: Code servant à fournir un retour à l'étudiant

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-
# Course: Pentesting
# Problem: Access solution file
# Preprocessing script

import json
import sys

sys.path.append('/task/static')
from lib import pythia

# Setup working directory
pythia.setupWorkingDirectory('/tmp/work')

# Read input data and fill skeleton files
data = sys.stdin.read().rstrip('\0')
input = json.loads(data)
pythia.fillSkeletons('/task/skeleton', '/tmp/work', input['fields'])

# Save task id
with open('/tmp/work/tid', 'w', encoding='utf-8') as file:
    file.write(input['tid'])
```

Listing 8: Code nécessaire à la préparation du test

```
root@f5c15812a789:/home/pythia/out: ./pythia execute -input="input.txt" \
                                        -task="tasks/pentest.task"
Status: success
Output: execve: Permission denied
```

Listing 9: Le message confirmant que les fichiers sont bel et bien securisés

L'environnement de développement de Pythia Core était initialement extrêmement exigeant en termes de dépendances. Il fallait impérativement un système Linux avec diverses dépendances afin de pouvoir compiler les UMLs d'exécution et lancer ces UMLs.

Afin de s'ouvrir à une communauté de contributeurs la plus large possible, il a été décidé de consacrer du temps à la création d'un environnement de développement portable sur un maximum de plateformes.

Nous avons choisi d'utiliser Docker, et j'ai réalisé une première itération sur base d'un premier essai réalisé par Mme Van den Schrieck, une contributrice au projet.

Cette itération bien que fonctionnelle et permettant de travailler correctement nous a semblé insuffisante en comparaison de l'éventail de possibilités offert par Docker.

```
FROM ubuntu

MAINTAINER Virginie Van den Schrieck, virginie.vandenschrieck@pythia-project.org

# Build utilities
RUN cat /etc/resolv.conf
RUN apt-get update  \
    && apt-get install -y gcc libc6-dev make curl wget xz-utils\
                ca-certificates bzip2 --no-install-recommends \
    && rm -rf /var/lib/apt/lists/*


#Install Go
ENV GOLANG_GOOS linux
ENV GOLANG_GOARCH amd64
ENV GOLANG_VERSION 1.5

RUN curl -k -sSL https://golang.org/dl/go$GOLANG_VERSION.$GOLANG_GOOS-$GOLANG_GOARCH.tar.gz \
    | tar -v -C /usr/local -xz

ENV GOPATH /go
ENV PATH $GOPATH/bin:/usr/local/go/bin:$PATH

RUN mkdir -p "$GOPATH/src" "$GOPATH/bin" \
    && chmod -R 777 "$GOPATH"

WORKDIR $GOPATH

#COPY go-wrapper /usr/local/bin/

#Install Git

RUN apt-get update \
  &&apt-get install -y git

#Install other utilities

RUN apt-get update \
  && apt-get install -y fakeroot squashfs-tools libc6-dev-i386 bc

#Install Pythia

WORKDIR /home
RUN git clone https://github.com/pythia-project/pythia-core.git pythia
RUN ls && pwd
WORKDIR /home/pythia/
RUN git submodule update --init --recursive && make


#Change fstab to have shm in no-exec mode for UML
RUN echo "tmpfs /dev/shm tmpfs defaults,nosuid,nodev 0 0" >> /etc/fstab && echo """>>/etc/fstab

```

```
53  #TODO manually when running in privileged mode : mount /dev/shm
```

Nous avons donc décidé de réaliser une seconde itération afin de permettre d'utiliser docker pour déployer l'application en production aussi.

Pour cela, j'ai travaillé avec l'extension à Docker, docker-compose qui permet de spécifier des environnements afin de lancer un réseau de containers connecté les uns aux autres tout en étant séparé du reste du réseau local.

Ceci ayant pour objectif à terme d'envisager un déploiement facile dans une infrastructure sur le cloud à l'aide d'un cluster Docker Swarm, qui permettrait une gestion de la montée en charge optimale et automatisable.

```
1  version: '3'
2  services:
3    pool:
4      privileged: true
5      security_opt:
6        - seccomp:unconfined
7      build: .
8      volumes:
9        - .:/home/pythia
10     links:
11       - queue
12     entrypoint: ./out/pythia -queue "queue:9000" pool & mount -av
13   queue:
14     privileged: true
15     security_opt:
16       - seccomp:unconfined
17     build: .
18     ports:
19       - "9000:9000"
20     volumes:
21       - .:/home/pythia
22     entrypoint: ./out/pythia -queue "0.0.0.0:9000" queue
23   server:
24     privileged: true
25     security_opt:
26       - seccomp:unconfined
27     build: .
28     ports:
29       - "8080:8080"
30     volumes:
31       - ./:/home/pythia
32     links:
33       - queue
34     entrypoint: ./out/pythia -queue "queue:9000" server
```

Listing 10: Configuration de docker-compose

C'est à ce moment-là qu'est apparu une contrainte particulière, l'exécution des UML nécessite l'accès à des ressources spécifiques du système protégé par Docker par défaut et plus précisément à un système de fichiers partagés temporaire (/dev/shm).

L'accès à ce système de fichier est en effet requis pour procéder à l'exécution des UML le système de fichiers des UML étant monté en mémoire temporaire. Hors ce principe faisant directement appel aux ressources du système hôte, il est interdit par défaut afin de préserver le principe de sandbox [8] du container.

8. "La sandbox ou bac à sable est définie par la capacité de diminuer les privilèges d'un processus programmatiquement et sans autorité sur la machine, et ce sans que le code dans la sandbox n'en soit conscient." Source : Security In-Depth for Linux Software - Preventing and Mitigating Security Bugs by Julien Tinnes & Chris Evans, Google Inc.

Ces accès étant incontournables, peu importe l'environnement utilisé, nous avons choisi de diminuer les paramètres de sécurité de Docker afin d'autoriser l'accès à ces ressources réservées en tenant compte du fait qu'une sortie du container afin d'écrire sur le système hôte serait facilité, mais nécessiterait toujours un haut niveau de connaissances.

En effet, si la sécurité est réduite dans la machine virtuelle, celle du système hôte ne l'est pas et le noyau Linux ne permet pas d'accéder à des ressources sans les permissions appropriées.

Nous avons aussi tenu compte du facteur temps, la durée de vie de la machine virtuelle étant limité, une tentative de sortie de la sandbox nécessiterait l'usage d'une faille de sécurité permettant de très rapidement utiliser le système de fichiers temporaire pour infecter l'hôte, ce qui nous semble compliqué en raison du long temps d'exécution habituellement nécessaire à l'exécution des failles de sécurité reposant sur l'usage de la mémoire.

Enfin, une fois l'environnement de développement reposant sur Docker fonctionnel, la documentation en a été écrit afin qu'il puisse être utilisé facilement.

2. Réparation de l'Intégration Continue

   Lors du développement, nous avons constaté une défaillance dans l'exécution du processus d'intégration continue, il s'avère qu'un des utilitaires utilisés pour configurer l'environnement nécessaire à l'exécution de la suite de tests était défaillant à cause de la fin du support de l'utilitaire.

   Il m'a donc fallu réaliser quelques recherches afin de trouver quel était l'outil recommandé pour la gestion des versions du compilateur Golang dans l'environnement de Travis CI.

   C'est l'utilitaire 'gimme' développé par Travis CI qui a été choisi.

```
17  17      - sudo apt-get update
18  18
19  19  install:
20      -   # Install gvm (go version manager)
21      -   - bash < <(curl -s -S -L https://raw.githubusercontent.com/moovweb/gvm/master/binscripts/gvm-installer)
22      -   - source /home/travis/.gvm/scripts/gvm
23  20      # TODO: to remove when GNU make >= 4.0 in travis CI
24  21      - wget http://ftp.us.debian.org/debian/pool/main/m/make-dfsg/make_4.0-8.1_amd64.deb
25  22      - sudo dpkg -i make_4.0-8.1_amd64.deb
26      -   # Install various version of golang
27      -   - gvm install go1.2 -B
28      -   - gvm install go1.4 -B
29      -   # https://github.com/moovweb/gvm#a-note-on-compiling-go-15
30      -   - gvm use go1.4
31      -   - gvm install go1.6 -B
32  23
33  24  script:
34  25      # Build everything and test with go1.2
35      -   - gvm use go1.2 && make all check
        26  +   - eval "$(gimme 1.2)" && make all check
36  27      # Test with go1.4
37      -   - rm out/pythia && rm -r go/pkg go/bin && gvm use go1.4 && make go check
        28  +   - rm out/pythia && rm -r go/pkg go/bin && eval "$(gimme 1.4)" && make go check
38  29      # Test with go1.6
39      -   - rm out/pythia && rm -r go/pkg go/bin && gvm use go1.6 && make go check
        30  +   - rm out/pythia && rm -r go/pkg go/bin && eval "$(gimme 1.6)" && make go check
        31  +   # Test with go1.7
        32  +   - rm out/pythia && rm -r go/pkg go/bin && eval "$(gimme 1.7)" && make go check
        33  +   # Test with go1.8
        34  +   - rm out/pythia && rm -r go/pkg go/bin && eval "$(gimme 1.8)" && make go check
40  35
41  36  addons:
42  37      apt:
```

FIGURE 4 – Patch ayant servi à refaire fonctionner Travis CI

5.1.5 Potentiel d'amélioration

Le Core de Pythia dispose de quelques améliorations potentielles qui pourraient avoir un impact significatif sur le fonctionnement de la plateforme.

— Le raffinement de l'algorithme de répartition des tâches Raffiner l'algorithme de répartitions des tâches en y ajoutant par exemple un indice de performance des Pools qui y sont associés permettraient de maintenir de meilleures performances dans certaines situations, en se concentrant sur l'usage des Pools les plus efficaces dans l'exécution des tâches.

— Modulariser le fonctionnement des Pools Modulariser le fonctionnement des Pools permettrait d'imaginer à terme la création de modules exécutant les tâches sur d'autres types d'infrastructures notamment de type FaaS (Fonction en tant que Service). Les infrastructures FaaS telles qu'AWS Lambda par Amazon permettent d'exécuter une fonction dans un environnement cloisonné et d'en obtenir le résultat.

5.2 Watchdog

Le watchdog est une application de surveillance de processus locaux et distants reposant sur les signaux standards POSIX ainsi que sur le protocole SSH.

Cette application n'existait pas à mon arrivée et son développement rentre dans le cadre de l'objectif spécifique "Création d'un outil de supervision de processus".

5.2.1 Analyse de la problématique

Actuellement, l'état de la plateforme Pythia n'est pas supervisé, ce qui peut entraîner en cas de plantage des impossibilités pour les étudiants de soumettre leurs tâches ou travailler.

Il est donc nécessaire d'avoir les outils nécessaires à la surveillance de la plateforme afin de pouvoir permettre une réaction en cas d'incident, mais aussi afin de pouvoir permettre une évolutivité automatisée de la plateforme en fonction de la charge afin d'assurer une continuité de service.

5.2.2 Choix techniques

Sur le plan technique, les contraintes étaient d'utiliser un logiciel performant, idéalement binaire et facile à installer et déployer.

J'ai donc commencé par effectuer des recherches afin de trouver un logiciel existant permettant de remplir ces critères. Et bien qu'il existe de très nombreux systèmes de monitoring, la majorité d'entre eux sont complexes à déployer et pas exactement orienté performance.

De ces recherches, deux outils sont ressortis, netadata [9] et Monit [10], ils présentaient tous deux d'intéressantes caractéristiques.

— netdata : L'outil est extrêmement puissant, mais est prévu pour le monitoring d'une architecture entière et non de processus précis. Il propose toutefois un système d'alarme sophistiqué qui pourrait permettre une relance automatique en cas d'incident.

— Monit : Monit est beaucoup plus léger que ses concurrents, mais il ne teste pas réellement le fonctionnement, il teste uniquement la réponse à un appel réseau ni n'offre un réel monitoring de l'usage en termes de ressources.

Aucun de ces deux outils ne nous amenait pleine satisfaction, nous avons donc exploré deux autres solutions, le recours aux solutions de monitoring déjà en usage à l'ECAM ainsi que le développement de notre propre outil de surveillance.

L'usage des outils de l'ECAM n'était pas approprié à cause de l'élan que nous souhaitons donner à la plateforme. Nous souhaitons permettre un déploiement facile de la plateforme par toute organisation intéressée, il nous semblait donc vital d'offrir l'outil de surveillance dans la plateforme et de ne pas recourir à un outil indépendant.

Sur le plan technique, une des contraintes étant dans la mesure du possible de rester sur un logiciel exécutable sous forme de binaire et d'être performant.

Ainsi que de rester dans la logique d'utiliser les technologies déjà en usage au sein de Pythia à des fins de maintanibilité et de pérennité du logiciel, nous avons décidé de coder l'application en Golang tout comme Pythia Core.

Notre choix a de plus été conforté par la présence dans la librairie standard de fonctionnalités telles que la gestion des signaux et appels système et dans une libraire supportée officiellement par les mainteneurs du langage le support des connexions SSH.

9. `https://github.com/firehol/netdata`
10. `https://github.com/monit/#home`

5.2.3 Développement de l'application

L'application a été développée en trois temps, avec dans un premier temps la création d'une librairie de manipulation de processus locaux et distants, et dans un second temps la création du programme de supervision en lui-même. Enfin, le troisième temps a été consacré à l'écriture des tests unitaires de la libraire de manipulation de processus.

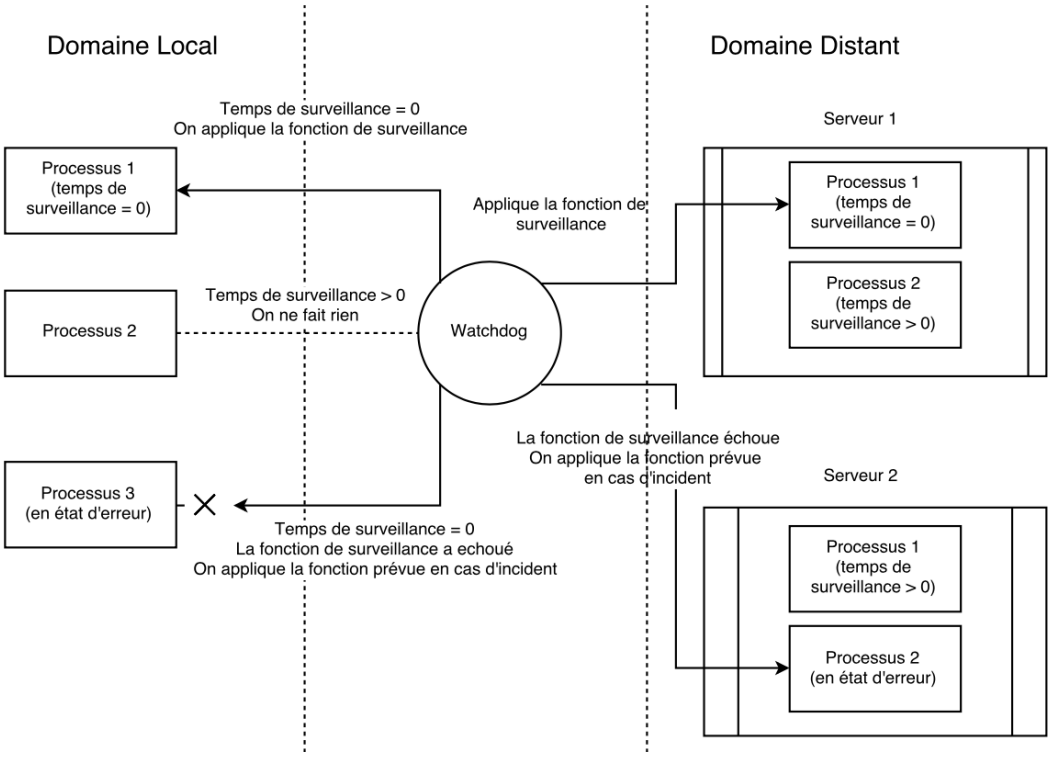

FIGURE 5 – Schéma de fonctionnement du Watchdog

1. Librairie de manipulation de processus
   Cette librairie a été écrite en Golang et dispose des fonctionnalités suivantes :
   — Lancement d'un processus local ou distant
   Le lancement d'un processus se fait soit en local, soit à distance via le protocole SSH. Les entrées et sorties du processus sont redirigées dans des fichiers de sortie et la connexion SSH n'est pas maintenue une fois le processus lancé.
   — Envoi d'un signal à un processus local ou distant
   L'envoi d'un signal à un processus se fait sur base de son PID, si le processus est distant, une connexion SSH est émise et la commande kill est utilisée. Dans le cas où le processus est local, il s'agit toujours de la commande kill qui est utilisée.
   — Appel à une fonction passée en paramètre à intervalle régulier avec une fonction à appeler en cas d'erreur de l'application
   Cette fonctionnalité sert à vérifier à intervalles réguliers que le processus est bien dans l'état attendu et non pas dans un état d'erreur. Elle pourrait toutefois être utilisée à d'autre fin telle que par exemple, lancer un nettoyage des fichiers de log tous les soirs.
   Elle repose sur l'exécution d'une fonction selon l'intervalle spécifié et en cas d'erreur l'exécution d'une fonction d'erreur spécifiée en paramètre.

```go
// Run a Process on a remote server
func (runtime Process) RunRemoteProcess(server Target) (*StartedProcess, error) {
  session, err := createSSHSession(server)
  if err != nil {
    return nil, errors.New("Failed to obtain an SSH session")
  }

  var buffer bytes.Buffer
  session.Stdout = &buffer

  // Create the command string
  command := createCommand(runtime.Executable, runtime.Arguments, runtime.Logs)

  err = session.Run(command)
  if err != nil {
    return nil, errors.New("Command : " + command + " : failed")
  }

  output := buffer.String()
  pid, err := strconv.Atoi(output)
  if err != nil {
    return nil, errors.New("Unexpected output")
  }

  return &StartedProcess{
    Executable: runtime.Executable,
    Server: server,
    Pid: pid,
    Logs: Logs {
      Stdout: runtime.Logs.Stdout,
      Stderr: runtime.Logs.Stderr,
    },
    Name: runtime.Name,
  }, nil

}
```

Listing 11: Fonction permettant le lancement d'un processus distant

```go
// Execute the function passed in parameter at the define frequency (in millisecond) on the given process
// Go count in nanosecond but we multiply by time.Millisecond
func (process StartedProcess) Watch(frequency int, onTick func(StartedProcess) (string, error),
  onCrash func(*StartedProcess) error) error {

  if frequency <= 0 {
    return errors.New("frequency must be greater than 0")
  }

  ticker := time.NewTicker(time.Duration(frequency) * time.Millisecond)
  quit := make(chan(struct{}))
  go func() {
    for {
      select {
      case <- ticker.C:
        _, err := onTick(process)
        if err != nil {
          onCrash(&process)
        }
      case <- quit:
        ticker.Stop()
        return
      }
    }
  }()
  return nil
}
```

Listing 12: Fonction permettant de lancer une fonction à intervalle régulier et de réagir en cas d'erreur

Cette librairie a été testée à l'aide de tests unitaires et a obtenu une couverture de code de 77%. Elle fait aussi l'objet d'une documentation réalisé selon les codes en vigueur dans la communauté Golang. À savoir une ligne présentant sommairement le but de la fonction ou de la variable.

2. Application de surveillance des processus

   L'application de surveillance et de gestion des processus repose sur la librairie process.go décrite au point précédent.

   Elle permet de lancer des processus locaux et distants et de réagir en cas d'incident.

   L'ensemble des informations nécessaires à l'exécution des processus sur les hôtes distants est fourni à l'application à l'aide d'un fichier JSON.

   Les processus sont lancés indépendamment les uns des autres en parallèle à l'aide de goroutines [11] afin d'éviter tout ralentissement ou problème de concurrence que pourrait entraîner un processus mettant du temps à démarrer. Il en va de même pour les fonctions de surveillance de processus, celle-ci doivent impérativement s'exécuter en parallèle et sont donc exécutés dans des goroutines séparées.

3. Tests unitaires de la libraire de manipulation de processus

   La librairie de manipulation des processus est unitairement testée à l'aide du framework de test unitaire fourni dans la libraire standard du langage.

11. "Une goroutine est un fil d'exécution tournant en parallèle avec d'autres goroutines à l'intérieur du même espace d'adressage (le processus). Les goroutines permettent de paralléliser des tâches en les répartissant sur plusieurs coeurs ou processeurs. Le mot clé go permet de lancer un appel de fonction en une goroutine, et de ne pas attendre le résultat." Source : Wikibooks, Programmation en Go

Cette librairie repose sur l'exécution des méthodes commençant par le mot clé Test et prenant en paramètre la structure testing.T fournie par le langage.

```go
// Try to start a SSH Session on a non-existing server using passworded authentication
func TestCreateSSHSessionPassword(t *testing.T) {
  dummyTarget := Target{
    Auth: Auth{
      Password: "password",
      PrivateKey: "",
    },
    Hostname: "Idonotexist",
    Name: "I-do-not-exist",
    Port: 9999,
    Username: "root",
  }

  _, err := createSSHSession(dummyTarget);
  if err == nil {
    t.Errorf("Received nil expected error")
  }
}
```

Listing 13: Exemple de test unitaire en Go

5.2.4 Rétrospective

Avec le recul et bien que cette application soit fonctionnelle, je l'aurais développée de la même manière, avec les mêmes contraintes. Néanmoins, si les contraintes avaient été plus flexibles, j'aurai sans aucun doute proposé une proposition basée sur un langage interprété telle que le Python afin de permettre une plus grande modularité.

En effet, le désavantage majeur de l'application actuellement est que s'il s'agit d'un simple binaire, les fonctions de réactions sur incident et de surveillance sont elles aussi compilées directement avec le binaire rendant leur modification compliquée sans interruption de services, là où un langage de script aurait pu être conçu pour régulièrement charger de nouveaux scripts.

Quant au Python, j'aurai proposé celui-ci pour deux raisons, il est présent sur la majorité des systèmes Linux et il fait partie des langages fortement utilisés dans l'entreprise.

Mais s'il n'y avait pas eu le souhait de proposer le Watchdog en tant que morceau de la plateforme, j'aurais eu recours à netdata et utilisé le temps consacré à l'écriture de ce Watchdog pour écrire une configuration optimale de netdata ainsi qu'un guide de configuration de netdata pour la plateforme.

5.2.5 Potentiel d'amélioration

Le Watchdog dispose de quelques améliorations potentielles qui pourraient avoir un impact significatif sur son fonctionnement.

— Rapatrier les logs.
Cela permettrait d'obtenir une vision claire sur le fonctionnement du logiciel. Actuellement, les logs sont stockés sur la machine exécutant le processus, il serait intéressant de pouvoir les récupérer sur la machine qui surveille afin de les analyser et de gérer plus finement les incidents.

— Une meilleure gestion des incidents.
Une meilleure gestion de la réaction sur incident serait particulièrement bénéfique au logiciel. Ce n'est toutefois pas réaliste pour le moment à cause du langage choisi, le recours à un langage compilé empêche d'ajouter une fonction sans devoir recompiler tout le logiciel, ce qui rend l'ajout de nouvelles réactions compliquées et nécessite une chaîne de compilation Go fonctionnelle.

— Un ajustement automatique des composants de la plateforme.
Un ajustement automatique des composants de la plateforme, par rapport à une configuration désirée. Si par exemple, l'on souhaite une Queue et trois Pools et que l'on s'attache à des processus existants et qu'il manque un Pool, il faudrait pouvoir démarrer un Pool supplémentaire.

— Récupérer les tâches dans la Queue en cas de crash.
Cette amélioration serait transversale au Core de Pythia en plus du Watchdog, mais les modifications les plus substantielles seraient à réaliser dans le Watchdog.
L'idée est de permettre la reprise des tâches existantes dans les composants en cas de plantage plutôt que de renvoyer une erreur pour toutes les tâches. Il faudrait pour mettre ceci en oeuvre, modifier les Queues et les Pools pour que les tâches y soient stockées dans des fichiers et que les Queues et les Pools puissent être démarrées avec un fichier de tâche à effectuer. Ainsi qu'avec les sockets à rouvrir avec leurs clients.

5.3 Pythia Learning Management System

Le Pythia Learning Management System est l'élément visible au public de la plateforme. Il s'agit de l'espace où les étudiants peuvent suivre le cours interactif et soumettre leur solution dans les champs présents tout au long du cours prévu à cet effet. Cet environnement est écrit en JavaScript (AngularJS) avec de l'HTML5 et du CSS3 pour la partie client, la partie serveur est quant à elle écrite en JavaScript et a recours à une base de données NoSQL orienté document, MongoDb.

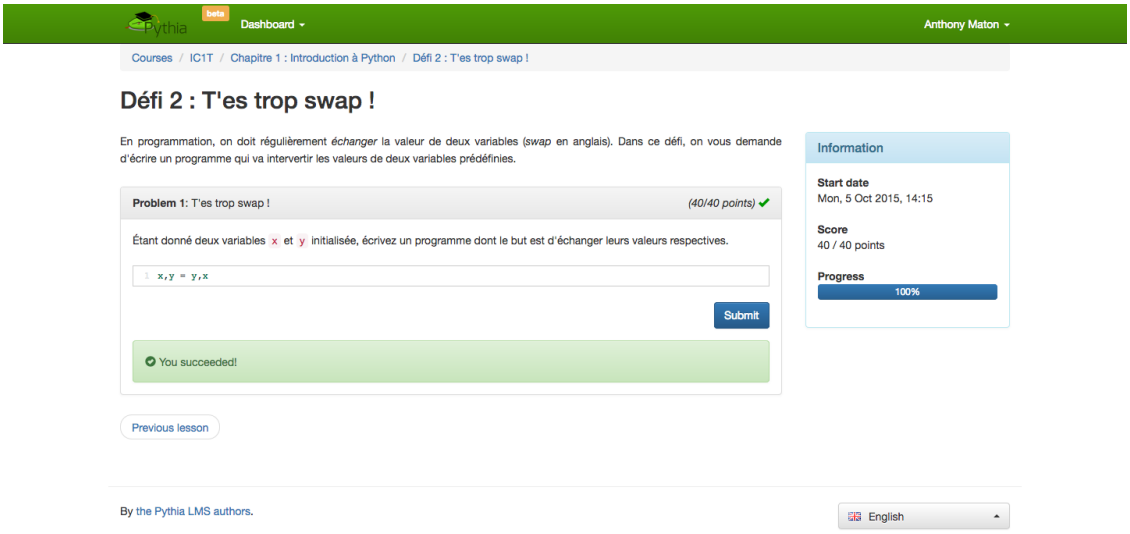

FIGURE 6 – Une leçon dans Pythia Learning Management System (partie étudiant)

5.3.1 Potentiel d'améliorations

Le Pythia Learning Management System a un fort potentiel d'évolution, notamment dû à l'incertitude liée à l'avenir du cadre applicatif sur laquelle il repose.

— Le cadre applicatif utilisé repose sur le projet MEAN [12] a actuellement un avenir des plus incertains.
En effet, le cadre applicatif en lui-même est actuellement en recherche d'un mainteneur pour poursuivre la maintenance du projet sur le long terme, cette recherche étant en cours depuis maintenant près de 6 mois sans succès, la pérennité du projet s'en retrouve donc mise en danger.
De plus, AngularJS [13] arrive doucement en fin de vie et est désormais en maintenance corrective uniquement. Google recommande d'ailleurs le passage à Angular 4. Or il s'agit de l'élément clé autour duquel la partie cliente est structurée.

— L'ajout de nouveaux types de questions, tels que par exemple des questions à choix multiples ou des questions plus théoriques style textes à trous.

12. Il existe de nombreux cadres applicatifs appelés MEAN, le cadre applicatif en usage dans l'application est le suivant, `http://meanjs.org/`

13. AngulrJS est un cadre applicatif pour développer des applications web dynamiques à l'aide de JavaScript, `https://angularjs.org/`

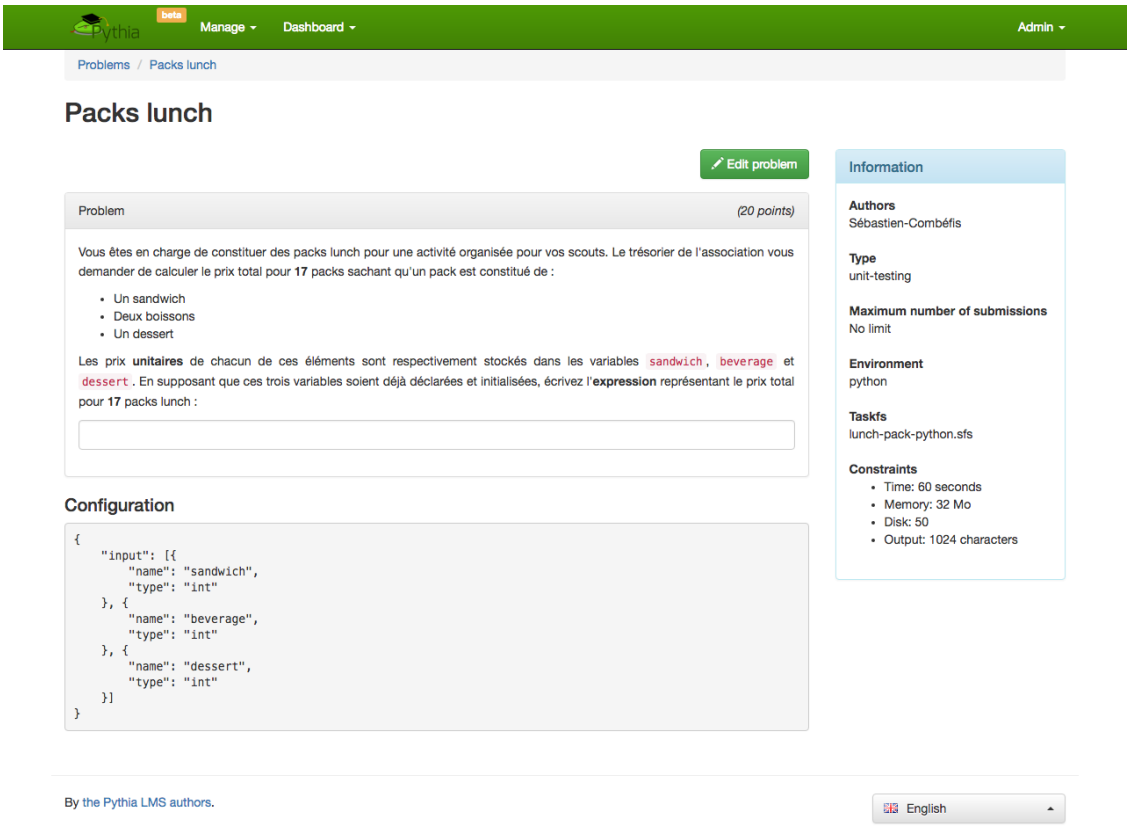

FIGURE 7 – Vue professeur d'un problème

5.4 Pythia Studio

Le studio est une application réalisée après une analyse du processus d'ajout d'un environnement et d'une tâche. Ce processus est relativement long et complexe, il nous a donc semblé important de simplifier ce processus afin de permettre un plus large usage de la plateforme et c'est ainsi que le studio est né.

Le studio est une application web écrite en Python et qui s'appuie sur une base de données relationnelle de type SQL, SQLite. Elle sert à simplifier la procédure d'ajout d'un environnement ou d'une tâche par l'intermédiaire d'une procédure basée sur le remplissage d'un formulaire plutôt que par l'écriture de scripts shell et de fichiers de configurations.

FIGURE 8 – Prototype de page d'accueil du studio

5.4.1 Analyse - Recherches et Développement

Le studio sert à la simplification de deux processus que sont l'ajout d'un environnement et l'ajout d'une tâche. L'application révolutionne donc autour de ces deux éléments.

Dans les tableaux ci-dessous, vous pouvez voir les données utilisées par notre application vue sous un angle métier. Ces tableaux correspondent à l'état de l'application que nous souhaitons obtenir à l'issue de la phase de Recherches et Développements.

TABLE 1 – Définition d'un environnement

*	Nom	Type	Spécificités
1	Nom	String	unique
1	Makefile	String	
1	Script	String	
0 à N	Processus	Processus	FK
0 à N	Dépendences	Dépendence	FK

TABLE 2 – Définition d'un processus

*	Nom	Type	Spécificités
1	Nom	String	unique
0-1	Description	String	
1	Executable	String	
1	Sortie	Sortie	FK
0 à N	Arguments	Argument	FK

TABLE 3 – Définition d'une tâche

*	Nom	Type	Spécificités
1	Nom	String	unique
1	Environnement	Environnement	FK
0 à N	Processus d'éxécution	Processus d'éxécution	FK
0 à N	Ressources	Ressource	FK
0 à N	Codes	Code	FK

TABLE 4 – Définition d'un processus d'éxécution

*	Nom	Type	Spécificités
1	Processus	Processus	FK
1	Position	String	
1	Erreur Fatale	Boolean	
1	Tâche	Tâche	FK

TABLE 5 – Définition d'une ressource

*	Nom	Type	Spécificités
1	Fichier	File	

TABLE 6 – Définition d'un code

*	Nom	Type	Spécificités
1	Nom	String	unique
1	Protected	Boolean	
1	Fichier	File	

TABLE 7 – Définition d'une dépendence

*	Nom	Type	Spécificités
1	Dépendence	String	unique

TABLE 8 – Définition d'une sortie

*	Nom	Type	Spécificités
1	Out	String	
1	Err	String	

5.4.2 Recherches et Développement

Lors de cette phase, nous avons tenté de réaliser l'application en nous basant sur une architecture et des technologies proches de celles utilisées pour le Learning Module System, à savoir un stack Node.JS / JS / MongoDb. À l'exception de l'usage d'AngularJS qui est progressivement supplanté par Angular4.

Un temps important a donc été consacré à la réalisation d'un prototype fonctionnel basé sur cette architecture.

1. Développement de la partie cliente

   La partie cliente en Angular 2 est écrite en TypeScript et repose sur une architecture orientée objet et des microservices spécialisés pour les liens clients-serveur.

   Les principaux types d'objets que l'on trouve sont les Components et les Services.

   (a) Les Components

   Les Components servent à l'affichage des informations, ils sont composés de 3 éléments. Une partie en Typescript qui contient la logique métier de l'élément avec ses propriétés, c'est aussi à ce niveau que se déroule les interactions avec les Services. La seconde partie est quant à elle écrite en HTML et contient le code HTML qui sera affiché à l'écran, avec la possibilité d'y effectuer des couplages avec les données venant de la partie TypeScript. Enfin, la dernière partie consiste en un fichier écrit en SASS afin de pouvoir styliser l'affichage du Component.

   (b) Les Services

   Les Services sont des éléments spécialisés dans la communication qu'elle soit inter-component ou client-serveur. Dans le cas qui nous occupe, nous les utilisons majoritairement pour de la communication inter-serveur. Ces Services sont injectés dynamiquement dans les Components qui les utilisent afin de ne les instancier qu'une seule fois et de pouvoir éventuellement réutiliser le résultat de requêtes précédentes.

   Par exemple, afin d'exécuter des appels sécurisés vers le serveur de manière aisée et efficace, j'ai eu recours à un wrapper autour du service de requêtes HTTP fourni par Angular afin d'y ajouter automatiquement les en-têtes nécessaires à l'authentification des requêtes et donc au lieu d'injecter le service HTTP fourni par Angular mes services sont injectés avec le wrapper HTTP personnalisé pour l'occasion.

   Le code employé dans la partie client est transpilé de Typescript vers JavaScript (ES5) à l'aide de webpack afin d'assurer une compatibilité maximale inter-navigateurs. La partie stylistique écrite en SASS est quant à elle compilée vers CSS.

```typescript
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';
import { Headers, RequestOptions } from '@angular/http';

import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/map';

import { AuthHttpService } from '../services/auth-http.service';
import { Environment } from 'models/Environment';
import { Process } from 'models/Process';
import { Config } from 'config';


@Injectable()
export class EnvironmentService {
  private url: string = `${Config.URL}/Environnements`

  constructor(private http: AuthHttpService) { }

  getEnvironments(): Observable<Environment[]> {
    return this.http.get(this.url)
      .map(this.extractData)
      .catch(this.handleError);
  }

  addEnvironment(environment: Environment): Observable<Environment> {
    return this.http.post(this.url, environment.removeId())
                    .map(this.extractData)
                    .catch(this.handleError);
  }

  addProcessToEnvironment(environment: Environment, process: Process): Observable<Process> {
    return this.http.post(`${this.url}/${environment.id}/processes`, process.removeId())
      .map(this.extractData)
      .catch(this.handleError);
  }

  removeProcessFromEnvironment(environment: Environment, process: Process): Observable<Environment> {
    return this.http.delete(`${this.url}/${environment.id}/processes/${process.id}`)
      .map(this.extractData)
      .catch(this.handleError);
  }

  private extractData(res: Response) {
    let body = res.json();
    return body || { };
  }

  private handleError (error: Response | any) {
    /* A standard error handler */
  }
}
```

Listing 14: Exemple de service servant à récupérer des environnements

```
@Injectable()
export class AuthHttpService extends Http {
  constructor (backend: XHRBackend, options: RequestOptions) {
    let token = JSON.parse(localStorage.getItem('authentication')).id;
    options.headers.set('Authorization', `Bearer ${token}`);
    options.headers.set('Content-Type', 'application/json');
    super(backend, options);
  }

  request(url: string|Request, options?: RequestOptionsArgs): Observable<Response> {
    let token = JSON.parse(localStorage.getItem('authentication')).id;
    if (typeof url === 'string') {
      if (!options) {
        options = {headers: new Headers()};
      }
      options.headers.set('Authorization', `${token}`);
      options.headers.set('Content-Type', 'application/json');
    } else {
    // we have to add the token to the url object
      url.headers.set('Authorization', `${token}`);
    }
    return super.request(url, options).catch(this.catchAuthError(this));
  }

  private catchAuthError (self: AuthHttpService) {
    // we have to pass HttpService's own instance here as `self`
    return (res: Response) => {
      console.log(res);
      if (res.status === 401 || res.status === 403) {
        localStorage.removeItem("authentication");
        console.log(res);
      }
      return Observable.throw(res);
    };
  }
}
```

Listing 15: Extension du module HTTP d'Angular 2

Les Components affichés par l'application sont chargés dynamiquement par l'application en fonction de l'adresse et du statut de l'utilisateur à l'aide d'un Component de routage. Nous avons donc affaire ici à une application de type Single Page Application.

2. Développment la partie serveur

   La partie serveur quant à elle repose sur la plateforme JavaScript Loopback avec une base de données NoSQL MongoDb.

   Loopback est un cadre applicatif de développement en JavaScript spécialisé dans la création d'API REST. Ce cadre applicatif a été utilisé afin de ne pas réinventer la roue et de s'assurer de l'usage d'une technologie pérenne, tout en respectant le critère d'usage d'une technologie aux sources ouvertes. Les deux autres raisons nous ayant poussés au choix de ce cadre applicatif sont la possibilité d'exporter les modèles de l'API dans un format interopérable, la norme Swagger ; ainsi que la très grande clarté et les possibilités de personnalisation des messages d'erreurs.

   Ce cadre applicatif a permis de très rapidement obtenir une API fonctionnelle et securisée à l'aide d'une Access Control List et d'une gestion des utilisateurs. Loopback permet par son utilitaire Apiconnect de générer très facilement des modèles et ainsi d'économiser un temps précieux. Néanmoins, ces modèles générés ne sont pas assez détaillés et l'utilitaire ne permet pas une configuration assez fine de ceux-ci. Ils nécessitent donc un grand nombre de retouches et de configuration afin de les adapter à nos besoins, notamment en termes de sécurité via la paramétrisation des Access Control List et en termes de relations entre les modèles.

   Ce modèle peut par exemple être appelé en HTTP à l'aide la requête HTTP suivante

   L'on peut voir clairement dans cette requête que l'objet "output" et le tableau d'"arguments" ont été correctement encapsulés dans la réponse, afin de fournir un objet le plus complet possible. J'ai privilégié l'usage d'objets complet au détriment du recours à une seconde requête pour obtenir les relations car la quantité de relations reste relativement faible et qu'elles n'ont dans la majorité des cas aucun sens à être séparées du reste. Dans les cas, où cela fait sens, j'ai utilisé une autre configuration afin que celle-ci ne soit pas incluse par défaut.

   Les modèles élaborés sont accessibles via les verbes HTTP existants (GET, POST, PUT, PATCH, DELETE, HEAD) et sont retournés en JSON ou en XML selon la valeur transmise à l'aide du Header Accept. L'API peut aussi être utilisé en XML ou en JSON à l'aide du Header Content-Type.

   Les scripts de création de tâches et d'environnements pourront ensuite être générés facilement une fois approuvés.

```
{
  "name": "Process",
  "plural": "Processes",
  "base": "PersistedModel",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "name": {
      "type": "string",
      "required": true
    },
    "description": {
      "type": "string",
      "required": true
    },
    "executable": {
      "type": "string",
      "required": true
    }
  },
  "validations": [],
  "relations": {
    "outputs": {
      "type": "embedsOne",
      "model": "Output",
      "scope": {
        "include": "linked"
      }
    },
    "arguments": {
      "type": "embedsMany",
      "model": "Argument",
      "scope": {
        "include": "linked"
      }
    }
  },
  "acls": [
    {
      "accessType": "*",
      "principalType": "ROLE",
      "principalId": "$unauthenticated",
      "permission": "DENY"
    }
  ],
  "methods": {}
}
```

Listing 16: Exemple de modèle Loopback avec relations et Access Control List

```
GET /api/Processes HTTP/1.1
Host: localhost:3000
User-Agent: curl/7.51.0
Accept: application/json
```

Listing 17: Requête HTTP servant à obtenir un Process

```
[
  {
    "name": "string",
    "description": "string",
    "executable": "string",
    "id": 0,
    "output": {
      "stdout": "string",
      "stderr": "string",
      "id": 0
    },
    "_arguments": [
      {
        "short": true,
        "argument": "string",
        "description": "string",
        "type": "string",
        "id": 0,
        "argumentValueId": 0
      }
    ]
  }
]
```

Listing 18: Réponse obtenue après la requête HTTP ci-dessus

3. Problématiques rencontrées

   Au fur et à mesure de cette phase de recherches et développements, nous avons constaté certains défauts majeurs liés à l'usage d'une solution telle que celle utilisée.

   — L'usage d'une base de données orientée Document s'est avéré compliqué à utiliser correctement dans un contexte orienté objet. Et ce particulièrement dû à la nature de ces systèmes n'étant pas prévu pour réaliser du stockage d'éléments ayant un couplage fort, ainsi que l'absence de support des transactions rendant très compliqué de certifier l'intégrité des données.

   — Le cadre applicatif utilisé côté serveur manquait de fonctionnalités importantes et qu'il n'était pas réaliste de se lancer dans le développement de celle-ci au vu des délais de livraisons.

      — L'absence de support de l'effacement en cascade sur les bases de données orienté objet rendait encore une fois impossible de certifier l'intégrité des données lors des suppressions.

      — L'impossibilité d'ajouter des objets complets (incluant leurs relations) en une seule requête HTTP, sans devoir écrire une route spécifique et donc non-respectueuse du modèle utilisé nous a mis devant un dilemme entre l'usage de routes personnalisées ou la perte de la certitude de l'intégrité des données.

4. Conclusion

   Par conséquent, nous avons mis fin à l'exploration de cette piste au profit d'un retour à une application traditionnelle en Python avec le Framework Django supportée par une base de données relationnelle de type SQLite. L'usage des développements réalisés pour la partie cliente nous semble utilisable dans un futur proche moyennant des adaptations, mais la nécessité de réaliser une partie serveur fonctionnelle nous pousse à repousser ceci plus bas dans la liste des priorités.

   Ce temps n'a toutefois pas été perdu entièrement, il m'aura ouvert les yeux sur le choix extrêmement stratégique que représente le système de stockage de données, mais aussi à quel point les systèmes de stockage alternatifs tels que les bases de données orientées document ou colonnes sont spécifique à des problématiques et la décision d'avoir recours à ces systèmes ne doit pas se faire à la légère, étant donné la complexité voir l'impossibilité de revenir sur cette décision sans engendrer des coûts colossaux.

   Ainsi que, sur le manque de maturité de l'écosystème Node.JS. En effet, le Node.JS bénéficie d'un fort attrait par les développeurs de par sa nouveauté et la légèreté dans l'écriture du code. Néanmoins, les outils et frameworks existants sont souvent incomplets ou peu maintenus rendant l'écosystème chaotique et peu propices à un usage en production dans un environnement critique.

5.4.3 Développement de l'application

À l'issue de cette phase exploratoire, il a été décidé de consacrer le temps restant à la réalisation d'une partie des fonctionnalités et plus précisément de la partie serveur. Ce développement se fait selon un principe itératif avec pour but une application fonctionnelle minimale et prête à être itérer pour en étendre les fonctionnalités.

Cela se traduit notamment par une simplification du modèle de données utilisées. Il n'est désormais plus prévu de pouvoir spécialiser les processus exécutés tâche par tâche dans cette itération. Ces processus sont maintenant configurés entièrement de manière non modulable et la création d'un processus spécialisé nécessite de dupliquer celui-ci.

Le nouveau modèle de données utilisées pour cette itération est le suivant :

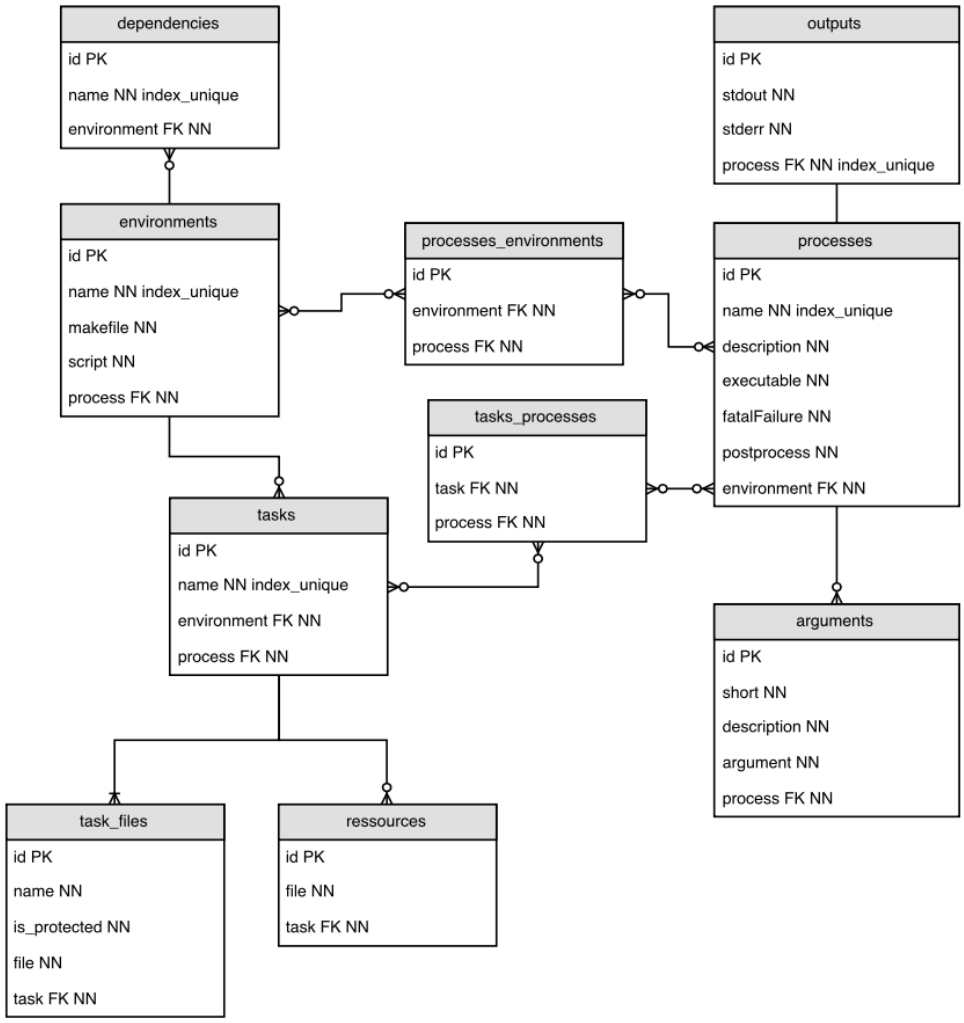

FIGURE 9 – Modèle de données simplifiés destinés à la phase 1 du développement de Pythia Studio

Lors de la prochaine itération, il est prévu de permettre la spécialisation de ce processus pour chaque tâche indépendamment, des modifications du schéma de données pourront être effectuées facilement grâce au système de migration proposé par Django. Ce système permet de modifier le modèle d'une base de données sans perte de données et en adaptant le schéma au fur et à mesure des changements.

1. Choix des technologies

   Suite à nos essais avec Node.JS, nous avons décidé de ne pas reitérer l'expérience dans l'immédiat et avons décidé de recourir au framework Django et au langage Python (dans sa version 3).

   Ce choix a été réalisé sur base des principes suivants :
   — L'ECAM utilise énormément de Python et a donc les ressources humaines et techniques pour assurer la pérénnité de l'application.
   — Le framework Django est extrêmement connu et reconnu dans la communauté des développeurs Python et jouit d'une très grande documentation et d'un support élevé.

   Nous avons aussi décidé de ne pas poursuivre nos essais avec une base de données Orienté Document MongoDb.

   Le paradigme de stockages de données orienté Document, nous semblant incompatible avec la nature de nos données. En effet, si la nature des bases de données orientées Document présente des avantages dans les environnements peu relationnels.

   Notre environnement est lui extrêmement relationnel et il n'est pas concevable de modifier l'intégralité des tâches manuellement à chaque fois qu'un environnement est modifié.

   Nous avons donc choisi d'avoir recours à une base de données relationnelle qui nous permet de couvrir nos besoins et avons pour cela choisi SQLite tout en tenant compte du fait que le changement de moteurs de bases de données en Django se modifie très facilement dans la configuration et que SQLite étant très respectueux du standard SQL, nous pourrons importer nos données dans PostgreSQL ou MySQL facilement.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Listing 19: settings.py configuré pour l'usage de SQLite

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'mydatabaseuser',
        'PASSWORD': 'mypassword',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

Listing 20: settings.py configuré pour l'usage de PostgreSQL

Le code ci-dessus relatif à l'usage de PostgreSQL est issue de la documentation officielle Django (`https://docs.djangoproject.com/en/1.11/ref/settings/#databases`)

2. Environnement de développement L'environnement de développement utilisé est un environnement de développement Python standard avec un environnement virtuel afin de s'assurer que les librairies et utilitaires installées soient correct et permettre à n'importe qui d'installer l'application facilement.

```python
class Environment(models.Model):
    name = models.CharField(max_length=128)
    makefile = models.TextField()
    script = models.TextField()
    process = models.ManyToManyField('Process', through='ProcessEnvironment',
                                     related_name='process_environment')

    def __str__(self):
        return self.name
```

Listing 21: Modélisation d'un Environnement dans Django

3. Création des modèles de données Les modèles de nos donnéees sont transposés sous forme de classe afin d'être comprise l'Object Relational Mapping de Django.

   Ces modèles sont aussi enregistrés dans l'interface d'administration de Django afin de pouvoir être modifié via celle-ci.

```python
class DependencyInline(admin.TabularInline):
    model = Dependency

@admin.register(Environment)
class EnvironmentAdmin(admin.ModelAdmin):
    inlines = [DependencyInline]
    pass
```

Listing 22: Enregistrement d'un modèle dans l'interface d'administration de Django

Une fois les modèles enregistrés dans la base de données, ils sont modifiable depuis l'interface d'administration de Django.

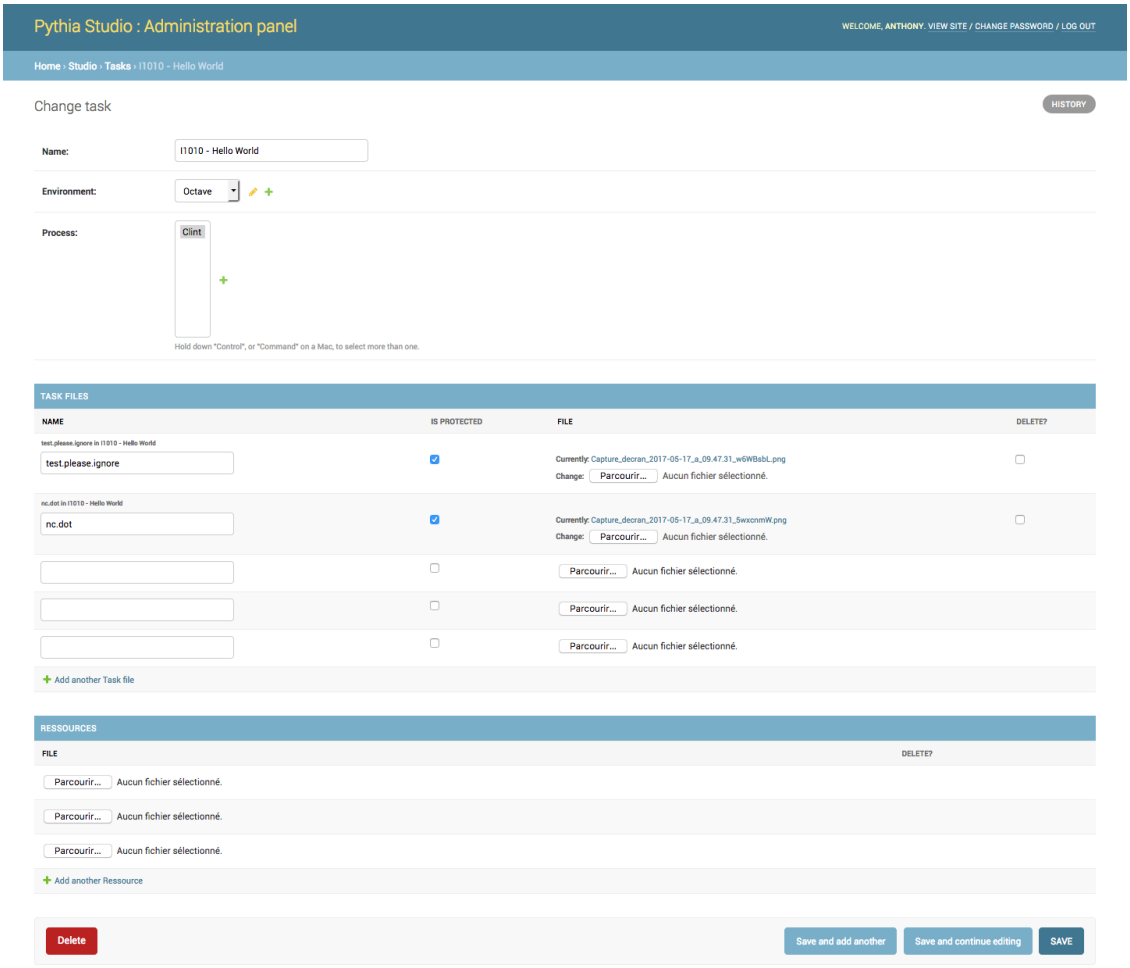

FIGURE 10 – Interface de modification d'une tâche

4. Création des scripts de génération d'environnements

   À partir des données stockées dans notre base de données nous constituons des scripts permettant de créer un environnement et nous fournissons ceux-ci sous la forme d'une archive tar (tape archiver) afin que ceux-ci puissent être intégrés dans une politique de déploiement automatisé le plus simplement et efficacement possible.

   Ces scripts sont basés sur des modèles qui sont remplis à l'aide des champs issus de la base de données. Un soin particulier a été accordé au fait que ces scripts restent lisibles par un être humain.

```
rootfs_config = '''{}

# Install busybox
install_busybox

# Install environment {}
{}

{}
        '''.format(self.license, environment.name,
                   self.__create_dependency_list(environment),
                   self.__create_script(environment)).rstrip()
```

Listing 23: Modèle servant à créer le script rootfs-config.sh

```
def __create_environment(self, environment):
        archive = tarfile.open('{}.tar'.format(environment.name),'x')
        with tempfile.TemporaryDirectory() as tmp_dir:
            try:
                self.__create_makefile(environment, tmp_dir)
                self.__create_rootfs_config(environment, tmp_dir)

                archive.add(tmp_dir, arcname=environment.name)
                archive.close()
            except:
                traceback.print_exc()
```

Listing 24: Méthode assurant la création de l'archive dans un environnement isolé pour éviter les problématiques de concurrence

Ces scripts sont ensuite appelés depuis des liens paramétrisés afin de générer les archives souhaitées.

— /generator
  — /tasks
    Génére une archive contenant l'intégralité des tâches existantes.
  — /task/{id}
    Génére une archive contenant la tâche correspondante.
  — /environments
    Génére une archive contant l'intégralité des environnements existants.
  — /environment/{id}
    Génére une archive contenant l'environnement correspondant.

Le lien entre l'adresse paramétrisée et le script de génération se fait à l'aide d'une vue. Toute interaction avec un utilisateur se fait à l'aide de vues en Django. Une vue est une fonction prenant une requête en paramètre et les paramètres de l'adresse paramétrisée.

```python
def generate_environment(request, identifier):
    try:
        EnvironmentGenerator().generate(identifier)
        return HttpResponse()
    except EnvironmentNotFound:
        return HttpResponse(status=404)
    except:
        return HttpResponse(status=500)
```

Listing 25: Vue servant à lancer la génération d'un environnement

5.4.4 Rétrospective

Fort de l'expérience acquise durant ce développement, si les choses étaient à refaire, il y'a certaines choses que j'aurai aimé faire différemment.

En particulier, lors du choix de la technologie, je suis tombé dans le piège de la "raison historique" et j'ai tenté de me conformer aux technologies en usage malgré que celle-ci ne soit pas la plus adaptée à notre besoin.

Cela s'est notamment ressenti par l'usage sans aucune réflexion d'une base de données NoSQL pour stocker des données profondément relationnelles en se basant sur la volonté de rester sur les technologies existantes dans la plateforme.

Mais aussi dans l'implémentation d'une interface de programmation REST complète là où une application client-serveur traditionnelle aurait suffi amplement. Cette interface de programmation ayant lourdement complexifié le développement d'un client capable de l'utiliser.

Si les choses étaient à refaire fort de cette expérience, j'aurais prioritisé l'usage d'une base de données relationnelle qui correspondent à nos besoins, et ce même si cela impliquait la mise en place d'un tel système.

J'aurais aussi privilégié une approche minimaliste du développement, en cherchant à créer une application qui fasse ce qu'on attend d'elle, ni plus ni moins, en allant droit au but.

6 Carnet de bord

Ce carnet de bord retrace le déroulement de mon stage semaine par semaine de manière succinte, l'essentiel ayant été dit précedemment.

6.1 Semaine 1 : Introduction à la plateforme

Lors de la première semaine de stage, j'ai découvert l'ECAM ainsi que le projet sur lequel j'ai travaillé.

Je me suis attelé lors de cette première semaine à obtenir un environnement de développement fonctionnel et à résoudre les soucis liés à Docker.

J'ai aussi débuté ma première tâche sur le code, en ajoutant la possibilité d'obtenir le statut de la Queue à l'aide d'un message TCP spécifique.

6.2 Semaine 2

Lors de la seconde semaine de stage, j'ai finalisé la tâche qui m'a été confiée la semaine précédente.

J'ai ensuite corrigé le fonctionnement défaillant de l'utilitaire utilisé pour l'intégration continue. J'ai aussi documenté le processus d'ajout d'un environnement pour le langage Octave, ce qui m'a permis de documenter le processus d'ajout d'un environnement.

Enfin, j'ai réalisé une analyse des possibilités d'évolutivité et de surveillance en cas de montée en charge ou de plantage.

6.3 Semaine 3

Lors de la troisième semaine de stage, j'ai poursuivi la réflexion sur les possibilités d'évolutivité et de surveillance de la plateforme.

J'ai aussi investigué deux bugs particulièrement tenaces, hélas il ne m'a pas été possible de reproduire ces deux bugs, ceux-ci n'existent peut-être pas.

6.4 Semaine 4

Lors de la quatrième semaine de stage, j'ai entamé le développement d'un système de surveillance des processus afin de pouvoir monitorer le fonctionnement de la plateforme.

6.5 Semaine 5

Lors de la cinquième semaine de stage, j'ai achevé le développement du système de surveillance des processus.

6.6 Semaine 6

Les développements réalisés en semaine cinq ont fait l'objet de la rédaction d'une suite de tests unitaires, cela s'est avéré particulièrement intéressant de tester les interactions avec le réseau.

6.7 Semaine 7

Lors de la septième semaine de stage, j'ai réalisé une analyse de la sécurité du processus d'exécution d'une tâche sur la plateforme. J'ai aussi commencé mon apprentissage du cadre applicatif AngularJS dans le cadre de futur développement du Pythia Learning Management System. Enfin,

j'ai entamé l'amélioration des processus de développement existant par l'ajout de docker-compose en plus de Docker.

6.8 Semaine 8

Le travail réalisé lors de la huitième de stage a consisté en l'ajout d'une page de statut dans le Pythia Learning Management System à l'aide d'AngularJS.

6.9 Semaine 9

Le travail réalisé lors de la neuvième semaine a été dans la continuité de celui de la semaine précédente. En effet, la documentation sur le cadre applicatif utilisé étant relativement pauvre, j'ai été confronté à de nombreuses difficultés.

Cela a finalement tourné à la réalisation de l'absence de support du cadre applicatif sur le plus long terme. Cette tâche a ensuite été annulée au profit de la recherche et développement sur l'utilitaire d'ajout et de suppression de processus et d'environnements.

6.10 Semaine 10

Suite au constat dressé en lors de la neuvième semaine cette tâche a été annulée et le début du processus de réfléxion autour de la création d'un outil d'ajout et de suppression d'environnements a débuté.

6.11 Semaine 11

Lors de cette onzième semaine, j'ai continué la réflexion autour d'un outil d'ajout de tâches et d'environnements. J'ai réalisé des interfaces hommes-machines afin d'offrir un aperçu de ce que je concevais comme potentiel pour l'application.

6.12 Semaine 12

Lors de cette douzième de stages, j'ai travaillé à la recherches et aux développements relatif à la réalisation d'une partie serveur afin de réaliser une démonstration de l'application d'ajout d'environnements et de processus.

6.13 Semaine 13

Lors de cette treizième semaine, j'ai travaillé à la communication entre la partie serveur réalisée la semaine précédente et la partie cliente. C'est ici que les ennuis ont commencés, certains choix techniques inadaptés ayant menés à une certain frustration à la fin de la semaine.

6.14 Semaine 14

Lors de cette antépénultième semaine, j'ai oeuvré à la résolution des soucis rencontrés, cela ne s'est hélas pas avéré concluant et a conduit à la course finale des semaines quinze et seize.

6.15 Semaine 15

Lors de cette pénultième semaine, la décision a été prise de mettre fin à la phase de recherches et développements non concluantes et d'entamer une phase de développement rapide en utilisant un cadre applicatif reputé et pérenne afin de livrer une application avec le plus de fonctionnalités possibles.

6.16 Semaine 16

Il s'agit de la dernière semaine de mon stage, le travail sur l'application se termine avec la possibilité d'ajouter des environnements et des tâches ainsi que de les modifier et de générer des archives contenant les fichiers nécessaire à l'éxécution des tâches.

7 Conclusion

Le projet a bien avancé par rapport aux objectifs définis au départ. Le Core de Pythia est désormais doté d'une application de surveillance de processus là où il n'y avait rien avant. L'environnement de développement de la plateforme est désormais multi-plate-forme au lieu de nécessiter une machine Linux impérativement. Le processus de création d'un environnement est désormais documenté et réalisable depuis une application web, il en va de même pour le processus d'ajout d'une tâche.

Mon seul regret par rapport à ces objectifs est l'impossibilité de raffiner l'application web qui permet l'ajout d'environnement et de tâches à cause d'une perte de temps liée à un mauvais choix technologique de ma part.

J'ai eu l'occasion de rencontrer de nouveaux problèmes auxquels je n'avais encore jamais été confronté.

En effet, j'ai dû apprendre un nouveau langage le Go à l'aide de la documentation et du code existant. Mais aussi faire des choix de technologies, tels que l'usage d'une base de données NoSQL plutôt que SQL, cette difficulté a été surmontée en me trompant de bases de données et en devant recommencer.

Le stage a été pour moi le paroxysme de ma formation à l'Institut Paul Lambin, j'ai eu l'occasion d'y apprendre énormément de nouvelles choses dans un nouvel environnement.

J'y ai découvert un aspect très différent de ce qui est inculqué à l'Institut. Là où l'Institut se concentre sur l'apprentissage des métiers de la création de logiciels en partant de la phase de conception, jusqu'à la livraison du produit, j'ai ici eu l'opportunité de découvrir la phase de maintenance et d'amélioration de la qualité de l'application qui se déroule après le développement initial.

Cela m'a permis de comprendre l'aspect crucial du métier qui est de se confronter au travail existant, le comprendre et l'améliorer. Mais aussi l'importance de la documentation pour la personne qui va reprendre le développement comme pour moi-même quelques mois après avoir écrit le code.

J'ai aussi pu découvrir à ma grande surprise, les conséquences catastrophiques d'une analyse incomplète ou incorrecte et de son impact sur le projet ainsi que le coût colossal que cela peut représenter pour une entreprise de devoir changer de cap ou de technologies en cours de développement.

Mais aussi que l'analyse conceptuelle n'est pas la seule analyse à réaliser, qu'il y'a aussi une analyse profonde des technologies à utiliser à réaliser. Le choix d'un cadre applicatif et d'un ensemble technologique pérenne et adapté aux besoins du développement s'avère extrêmement compliqué entre l'usage de technologies récentes et potentiellement plus performantes ou des technologies plus stables et à la stabilité reconnue, mais avec des performances parfois moindres.

Les deux enseignements principaux que je retire de cette période de stage, sont qu'on ne construit pas le futur en oubliant le passé et que le métier du développement informatique sont des métiers qui réclame une connaissance de la passée, une volonté de construire l'avenir et surtout une flexibilité et un dynamisme constant dans la façon d'aborder les problèmes.

Les limites et améliorations potentielles du projet ont été détaillées point par point dans leurs intitulés respectifs, d'un point de vue global, la plateforme Pythia a un potentiel pédagogique certain et un potentiel d'évolution afin de le rendre plus accessible à de nouvelles institutions. L'amélioration la plus intéressante à mon sens à ce jour serait de poursuivre la simplification du processus d'ajout d'un énoncé.

La plateforme Pythia peut selon moi être un formidable outil d'apprentissage et si l'on peut vouloir critiquer le choix de l'usage de technologies récentes tel que le Go, celui-ci s'avère particulièrement stable et performant, comme l'on peut s'y attendre d'un produit venant de chez Google.

Et même si l'on peut a posteriori regretter le choix d'utiliser un cadre applicatif à l'avenir

incertain, ce choix fut en son temps le bon choix, et le produit est amené à évoluer dans le temps, avec cette fois-ci probablement un choix plus pérenne.

8 Annexes

Vous trouverez à la fin de ce travail, une pochette contenant un CD-Rom, ce CD contient les annexes mentionnées ci-après.

- Un fichier readMe reprenant les instructions de cette liste.
- Un dossier "travail écrit" reprenant les sources permettant la génération de ce document.
- Un dossier "code" sous-divisé en sous dossiers
  - Un dossier "environments" contenant les environments liés à Pythia
  - Un dossier "pythia-lms" contenant le code du Pythia Learning Management System
  - Un dossier "pythia-studio" contenant le code source issu de la recherche et du développement de la partie client de l'outil Pythia Studio.
  - Un dossier "pythia-studio-backend" contenant le code source issu de la recherche et du développement de la partie serveur de l'outil Pythia Studio.
  - Un dossier "python" content un unique sous-dossier "pythia-studio" qui contient le code relatif à la phase de développement de l'outil Pythia Studio lors du passage au cadre applicatif Django.
  - Un dossier "task-reprocessing-test" contenant une démonstration du fonctionnement du test basé sur les propriétés à des fins de comparaisons avec le processus de test actuel utilisé par Pythia.
  - Un dossier "watchdog" contenant le code source du logiciel utilisé pour surveiller les processus liés à Pythia ainsi que dans le sous-dossier "process" la librairie écrite afin de manipuler les processus.
  - Un dossier "pythia-core" contenant le code lié aux composants côté serveur de Pythia (Queues, Pools)
- Un dossier "documentations" reprenant les documents produits au long du stage
  - Proccesus d'ajout d'un environnement, reprend la procédure d'ajout d'un environnement à Pythia
  - Analyse de sécurité relatif aux accès aux solutions
  - Analyse partielle afin de comprendre le fonctionnement de Pythia lib
  - Comparatif entre le processus de test actuel et les tests unitaires
  - Préanalyse du Watchdog
  - Première analyse du Watchdog
  - Première réfléxion lié à Pythia Studio
  - Processus d'ajout d'un environnement (version propre)

9 Bibliographie

Cette bibliographie reprend les ouvrages de référence m'ayant été utiles durant la période de stage. Les ouvrages de référence cités directement dans le document présent ont eux fait l'objet d'une bibliographie en notes de bas de page tel que recommandé dans le guide de rédaction de TFE de l'Institut Paul lambin édition 2017.

Références

[1] Various authors. Django documentation. `https://docs.djangoproject.com/en/1.11/`, 2017.

[2] Various authors mainly from Uber Inc. Blazing fast, structured, leveled logging in go. `https://godoc.org/go.uber.org/zap`, 2017.

[3] Google Inc. Angular js api docs. `https://docs.angularjs.org/api`, 2017.

[4] Google Inc. Golang documentation. `http://golang.org/doc/`, 2017.

[5] Google Inc. A quick look at angular basics. `https://angular.io/docs/ts/latest/quickstart.html`, 2017.

[6] StrongLoop Inc. Loopback documentation. `http://loopback.io/doc/en/lb3/`, 2017.

[7] Vianney le Clément de Saint-Marcq Sébastien Combéfis. Teaching programming and algorithm design with pythia, a web-based learning platform. *Olympiads in Informatics*, 6 :31–34, 2012.