Rapport de stage

Mise en place d'un environnement de qualification pour la plateforme PLM

Pierric Grguric

Année 2014–2015

Stage de 2e année réalisé au LORIA

en vue de la validation de la 2e année d'études à TELECOM Nancy

Maître de stage : Martin Quinson, Gérald Oster

Encadrant universitaire : Suzanne Collin

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Grguric, Pierric

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31314760

Année universitaire : 2014–2015

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Mise en place d'un environnement de qualification pour la plateforme PLM dediée à l'apprentissage de la programmation

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Vandoeuvre-lès-Nancy, le 10 septembre 2015

Signature :

Rapport de stage

Mise en place d'un environnement de qualification pour la plateforme PLM

Pierric Grguric

Année 2014–2015

Stage de 2e année réalisé au LORIA

en vue de la validation de la 2e année d'études à TELECOM Nancy

Pierric Grguric
8, rue Jacques Callot
54500, Vandoeuvre-lès-Nancy
+33 (0)6 02 36 08 35
pierric.grguric@telecomnancy.net

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

LORIA
Campus scientifique
54506, Vandoeuvre-lès-Nancy
+33 (0)3 83 59 20 00

Maître de stage : Martin Quinson, Gérald Oster

Encadrant universitaire : Suzanne Collin

Remerciements

Je tiens à remercier toutes les personnes qui m'ont aidé tout au long de mon stage.

Plus particulièrement, je tiens à remercier mes maîtres de stage MM. Martin Quinson et Gérald Oster pour l'encadrement et les conseils qu'ils m'ont apporté pendant mon stage.

Je remercie également M. Mathieu Nicolas pour son aide, notamment pour la compréhension du code existant.

Table des matières

1 Introduction

La "Programmer's Learning Machine" (PLM) est un logiciel d'apprentissage des bases de la programmation informatique. Il est utilisé tous les ans pour former une centaine d'étudiant à TELECOM Nancy de façon intensive (en deux semaines). Jusqu'à maintenant, il n'existait qu'une version "locale" de la PLM, ce qui posait un problème en termes de possibilité de diffusion. Pour remédier à ce problème, la décision a été prise de transformer le logiciel en une application web. L'un des inconvénients de cette solution est le partage des ressources.

C'est dans le cadre de cette décision que s'inscrit mon stage. Plus particulièrement, j'ai été chargé de mettre en place un environnement de test pour déterminer la charge que peut supporter cette nouvelle version de la PLM, ainsi que des tests unitaires afin de s'assurer qu'une version stable soit prête pour la rentrée de septembre.

De prime abord, nous verrons plus en détail le contexte du stage, à savoir l'établissement d'accueil : le LORIA, ainsi que le logiciel PLM et plus particulièrement ses évolutions récentes : sa version locale et sa version web.

Ensuite, nous verrons le déroulement de mon stage, c'est-à-dire les objectifs et la manière dont je les aient complétés, ainsi que les problèmes que j'ai rencontrés et la façon dont je les aient résolus sur les trois parties de mon stage qui sont :

- la mise en place de scripts Docker afin de faciliter le lancement du serveur de PLM,
- l'implémentation de tests unitaires afin de vérifier le comportement des nouvelles fonctions,
- et l'écriture de scénarios de tests Protractor afin de simuler les actions d'un utilisateur sur l'interface web.

Pour finir, nous verrons dans quelle mesure mon travail a été intégré à la version actuelle de la PLM, ainsi que les améliorations qu'il est prévu d'apporter au projet.

2 Présentation

2.1 Présentation du LORIA

Le LORIA (laboratoire lorrain de recherche en informatique et ses applications) est une Unité Mixte de Recherche (UMR 7503), commune à plusieurs établissements : le CNRS (Centre National de la Recherche Scientifique), l'Université de Lorraine et Inria (Institut National de Recherche en Informatique et Automatique). Depuis sa création en 1977, sa mission est la recherche fondamentale et appliquée en sciences informatiques.

le LORIA emploi 450 personnes réparties en cinq départements (Figure 2.1) [1] :

- Algorithmique, calcul, image et géométrie (6 équipes)
- Méthodes formelles (6 équipes)
- Réseaux, systèmes et services (3 équipes)
- Traitement automatique des langues et des connaissances (8 équipes)
- Systèmes complexes et intelligence artificielle (5 équipes)

J'ai été accueilli dans l'équipe VERIDIS dont la mission principale est la conception de systèmes distribués et la vérification des systèmes. Elle fait partie du département Méthodes formelles.

La PLM n'est pas vraiment un projet de l'équipe VERIDIS, mais plutôt de MM. Martin Quinson et Gérald Oster qui l'ont développé pour l'enseignement à TELECOM Nancy.

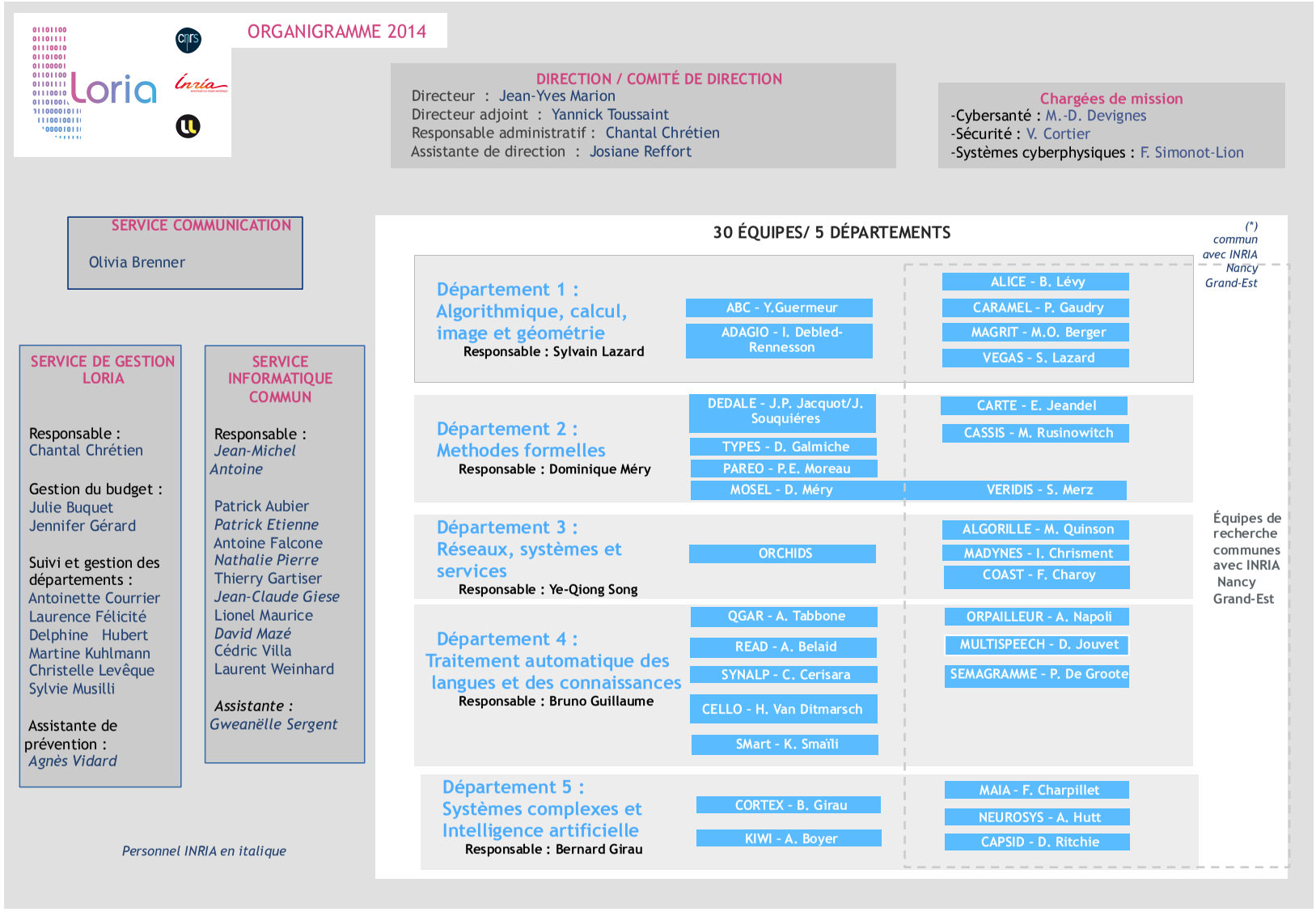

FIGURE 2.1 – Organigramme du LORIA

2.2 Présentation de la PLM

MM. Martin Quinson et Gérald Oster ont démarré le projet PLM en 2008 dans le but de faciliter les débuts dans le monde de la programmation aux élèves de TELECOM Nancy. La PLM permet d'apprendre les bases de la programmation dans plusieurs langages (Java, Python, Scala) à travers 200 exercices allant de la déclaration de variable à l'écriture de fonction récursives en passant par les algorithmes de tri.[2]

2.2.1 PLM version lourde

Une fois le logiciel PLM lancé, l'utilisateur a accès à différents univers regroupant plusieurs exercices, classés de sorte à assurer une progression logique de l'utilisateur. Le premier univers a pour objectif de faire évoluer des "buggles" (petites bêtes issu du jeu space invaders) dans différents mondes, ce qui a pour effet d'apporter un coté récréatif aux premiers niveaux, et donc l'utilisateur ne se sent pas perdu dans du "code pur".

Pour réussir un exercice, l'utilisateur doit écrire un code qui va modifier le monde de l'exercice afin qu'il devienne identique au monde objectif. Sur la Figure 2.2 on peut voir le monde objectif sur la droite. Il s'agit d'un exercice de l'univers buggle car l'on peut en apercevoir un sur la case en haut à droite. Le monde de départ correspond à la même grille mais elle n'est pas encore colorié, elle est entièrement blanche. Le code solution de l'exercice sera donc celui qui donnera au buggle les instructions nécessaires pour qu'il reproduise les motifs présent sur le monde objectif.

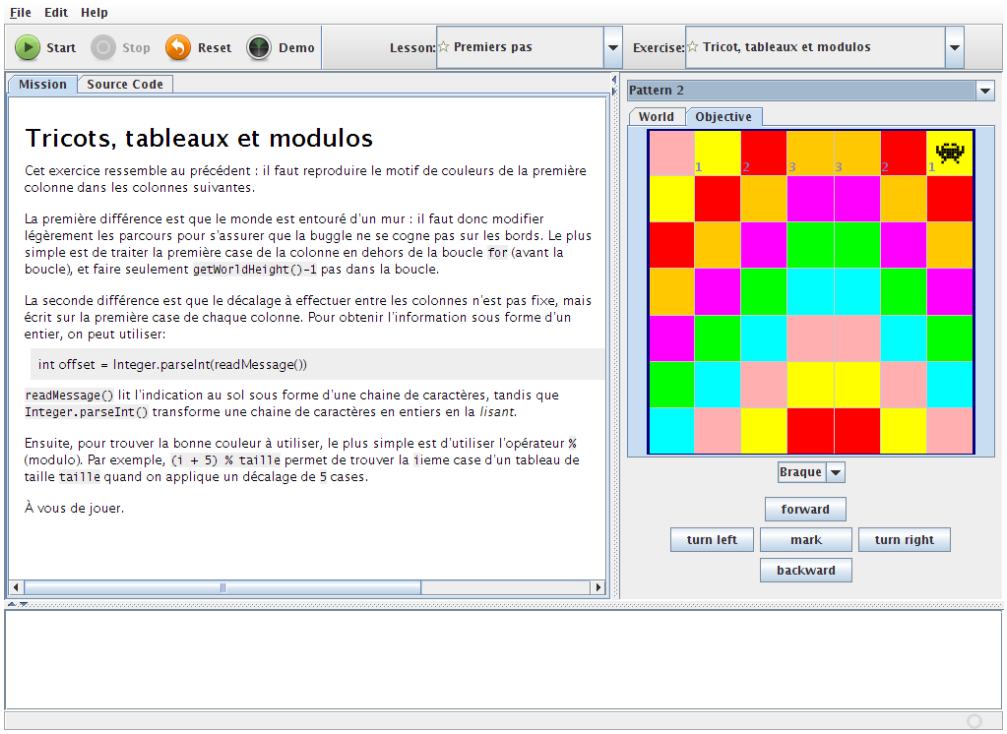

FIGURE 2.2 – Aperçu de la PLM version lourde.

Afin d'aider l'utilisateur dans l'écriture du bon code, et pour lui apprendre les commandes correspondantes lorsque c'est la première fois qu'il les rencontrent, il y a un texte descriptif de l'exercice sur la gauche de l'écran. Une fois que l'utilisateur a lu ces instructions, il peut changer d'onglet afin que cette partie de l'écran soit remplacé par un éditeur de texte où il va pouvoir y écrire son

code. Une fois que l'utilisateur estime avoir le bon code, il va lancer l'exécution et un message va apparaître dans la partie basse de l'écran. Ce message peut être de trois types : soit le code contient une erreur et donc le message décrit cette erreur, soit le code s'exécute correctement mais ne résout pas l'exercice et donc le message liste les différences entre le monde d'arrivée et le monde objectif, soit le code est solution de l'exercice et donc le texte est un message de félicitations pour confirmer à l'utilisateur qu'il a bien réussi l'exercice.

À l'origine, la PLM a été développée en tant qu'application Java avec interface Swing, ce qui implique que l'utilisateur doit lancer le programme sur son ordinateur, après avoir préalablement téléchargé l'archive jar de la PLM. L'un des inconvénient est que la progression de l'utilisateur doit être exportée manuellement lorsque l'on souhaitait changer de machine.

2.2.2 PLM version web

Pour la rentrée scolaire 2015, l'objectif du projet PLM était d'abandonner le client lourd au profit d'une version web, ceci afin de supprimer l'étape de téléchargement, d'avoir la possibilité de continuer son code facilement même si l'on change de machine, et de pouvoir diffuser la PLM à une plus large échelle, puisqu'il suffira d'accéder à une page web pour commencer à utiliser la PLM.

Cette nouvelle version a entraîné un changement de technologie, puisque le rendu graphique, auparavant réalisé en Swing, est maintenant assuré par AngularJS[3], ainsi que canvas pour le rendu des mondes. Un serveur web tournant sous Play Framework a également dû être mis en place, afin d'exécuter le code des utilisateurs. La connexion entre les utilisateurs (clients) et le serveur se fait à l'aide de WebSockets, ce qui permet un échange constant. La version obtenue n'ayant été testée que par une dizaine d'utilisateurs, il était crucial que des tests certifient le bon fonctionnement de l'application avant la mise en production à la rentrée de septembre.

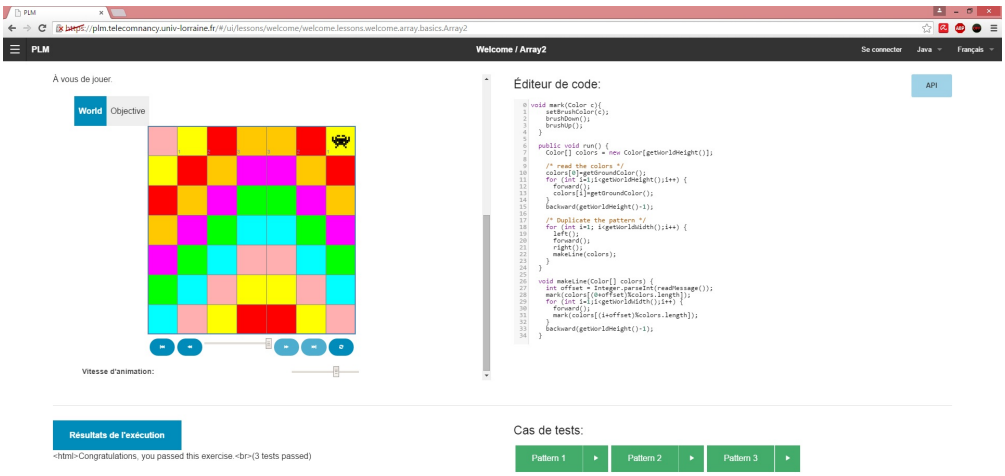

FIGURE 2.3 – Aperçu de la PLM version web

Sur la Figure 2.3 on peut voir un aperçu de la PLM dans cette nouvelle version. on y retrouve l'éditeur de texte sur la droite, et les instructions ainsi que le monde de l'exercice sur la gauche. La zone de texte donnant le résultat de l'exécution du code est toujours présente en bas de l'écran. On observe que l'on n'a plus une fenêtre d'application java, mais celle d'un navigateur web puisque la PLM est maintenant accessible depuis un site web.

Le développement de cette version web a été assuré par M. Mathieu Nicolas, un ingénieur diplomé de TELECOM Nancy qui travaille sur le projet PLM depuis un an. Le projet a été accéléré par la contribution d'élèves de l'IUT Nancy Charlemagne en stage sur le projet PLM sur la période mai/juin. C'est durant cette période que l'interface coté client a été développée, mais elle n'avait pas encore pu être testée complètement.

Durant mon stage, il y avait également trois autres élèves qui effectuait leurs stage sur la PLM :

- M. Bühler s'occupait de créer un réseau de neurones afin de détecter les élèves en difficulté d'après les codes soumis,
- M. Carpentier avait pour objectif de détecter les erreurs les plus fréquentes afin de proposer une aide personnalisée aux utilisateurs qui rencontrent ces erreurs,
- M. Gloaguen était chargé de développer une infrastructure distribuable coté serveur capable de supporter une centaine d'utilisateurs.

3 Déroulement du stage

C'est dans ce contexte de changement de version que mon stage c'est déroulé. En effet, les technologies utilisées n'étant plus les mêmes, il était nécessaire que certaines parties de la PLM soient recodées dans un autre langage. Afin de s'assurer que ces changements ne modifient pas le comportement de la PLM, il a été nécessaire de mettre en place deux types de test : des tests de correction ("Est ce que l'application fonctionne ?") et des tests de performance ("Est ce que X utilisateurs peuvent utiliser l'application en même temps ?"). C'est là l'objectif de mon stage.

Il s'est découpé en trois grandes parties à savoir :
- établir un script de lancement afin de déployer rapidement un serveur,
- écrire des tests unitaires afin de s'assurer que les composantes de la PLM se comportent comme attendu,
- écrire des tests E2E afin de simuler le parcours d'un utilisateur.

3.1 Infrastructure Docker

La première partie du stage était d'avoir un script de lancement de la PLM. En effet, affin d'effectuer les tests sur un environnement séparé, il était nécessaire d'avoir un script permettant de lancer un serveur de la PLM rapidement.

Pour ce faire, j'ai choisi d'utiliser la technologie Docker qui permet de générer une image équivalente à une machine virtuelle possédant uniquement les composantes souhaitées. Une image docker se créée de la façon suivante : on écrit un dockerfile qui liste l'ensemble des commandes nécessaires à l'obtention de l'état dans lequel on souhaite avoir notre image docker. Ainsi la "compilation" du dockerfile génère une image docker qui est dans l'état que l'on souhaite, prête à l'emploi. L'avantage de cette technologie est qu'une fois une image docker obtenue, elle peut être utilisée facilement pour déployer l'application.

Un des avantages qui m'a poussé à choisir cette technologie est que l'on peut facilement mettre à jour les composantes de l'image puisqu'il suffit de recompiler le dockerfile. De même, si l'on souhaite légèrement changer la configuration de l'image, la seule modification à faire est de changer la ligne correspondante du Dockerfile. Ensuite on recompile le Dockerfile et l'image est mise à jour.

Un autre avantage de la technologie Docker est que les dockerfiles peuvent hériter d'autres dockerfiles, ainsi on peut repartir d'une image docker déjà existante et exécuter quelques lignes de commandes à l'intérieur de cette image afin de créer une image docker adaptée à nos besoins. J'ai donc profité de cette fonctionnalité en découpant le script en deux parties :

- La première partant d'une image sous ubuntu avec un jdk où je rajoute le package Play Framework. C'est la partie "stable" du script puisqu'elle n'est pas modifiée lorsque l'on change le code de la PLM. Il n'y a donc pas besoin de régénérer l'image correspondante lors d'une modification de la PLM. J'ai choisi d'utiliser une image sous ubuntu comme base, car parmi les distributions sur lesquelles Play Framework est stable, c'est l'image de plus petite taille.
- La deuxième partie repart de cette image docker pour y télécharger la dernière version de PLM et la lancer. Ainsi en cas de modification de la PLM, il n'y a que la compilation du deuxième dockerfile à relancer.

La raison qui m'a poussé à séparer le script en deux parties est que l'installation de Play Framework requiert le téléchargement de nombreuses petites librairies. Or le téléchargement n'étant pas "direct" puisque le container est isolé du système d'exploitation de la machine hôte, il arrive que certaines dépendance soit "perdues" et donc la création de l'image docker échoue une fois sur trois. Ainsi la séparation en deux Dockerfile a permis d'installer Play Framework une fois pour toutes et par la suite cette image n'a pas était recréée.

Voici le script de la deuxième partie :

```
FROM pierricgrguric/docker-play-framework

RUN apt-get -y update && apt-get -y install git
RUN git clone https://github.com/MatthieuNICOLAS/webPLM.git --depth 1
WORKDIR webPLM
RUN activator compile
RUN activator clean stage

CMD ["./target/universal/stage/bin/web-plm"]
```

La première ligne correspond à l'héritage : on indique que l'image doit être construite à partir d'une image préexistante, en l'occurrence celle que j'ai auparavant construite et qui contient donc un environnement ubuntu, avec une jdk ainsi que le package Play Framework.
Les deux lignes suivantes correspondent à l'installation de git, puis au téléchargement de la dernière branche du dépot de PLM.
Ensuite, on se place à l'intérieur du dossier de PLM, puis on compile le projet afin de créer un binaire exécutable.
Enfin la dernière ligne correspond à la commande qui sera exécutée lors du lancement de l'image Docker. Elle aura pour effet de lancer le serveur PLM à l'intérieur du container Docker.

3.2 Tests unitaires

La deuxième partie du stage consistait en l'écriture de tests unitaires, car avec le passage en version web, les actions effectuées sur un monde sont calculées d'après l'exécution du code de l'utilisateur coté serveur, et sont rejouées coté client. Il était donc important de s'assurer que le monde visible coté client était identique à celui coté serveur.

L'ensemble des exercices de la PLM est divisé en plusieurs univers : l'univers baseball, l'univers bat, l'univers buggle, l'univers dutchFlag, l'univers hanoi, l'univers pancake et l'univers sort.
L'univers bat ne nécessite pas de tests car il n'y a pas de "monde", dans cet univers l'utilisateur doit créer des fonctions qui manipulent des chaînes de caractères, il n'y a donc pas d'image à afficher. Les exercices de cet univers sont issus du site codingbat[4].
Il existait déjà des tests sur l'univers buggle. Ces tests sont écrits en JavaScript avec le framework de test Jasmine[5]. Après une étape de compréhension des tests existants, j'ai donc implémenté les tests manquant sur cet univers, à savoir vérifier que l'application d'une opération puis son inversion sur un monde ne change pas le monde en question.

Le framework de test Jasmine permet d'écrire des scénarios de tests de façon structurée. Chaque test se présente alors en plusieurs parties :

- Une première section appelée "BeforeEach" qui correspond aux actions effectuée avant chaque test (avant chaque bloc "it")
- Une seconde section appelée "AfterEach" qui sera exécutée après chaque test (après chaque bloc "it")
- Une succession de bloc "it" qui correspondent chacun à un test différent. Chaque bloc "it" est constitué de commandes qui font fonctionner le code à tester et de conditions attendues. Si toutes les conditions d'un tests sont validées, alors le test est passé correctement. Sinon si l'une des conditions n'est pas respectée alors le test échoue et l'on passe au bloc "it" suivant.

Voici un exemple de test unitaire (un bloc "it") qui vérifie que l'application successive des méthodes apply puis reverse ne change pas le monde sur lequel on les applique. Il s'agit de l'opération flipOperation sur un monde de l'univers pancake. Le fichier complet est présenté en annexe.

```
it('should_not_change_currentWorld_when_applied_then_reversed',
    function () {
     var current = {
          pancakeStack: currentWorld.pancakeStack.slice(),
          moveCount: moveCount,
          numberFlip: numberFlip
     };
     flipOperation.apply(currentWorld);
     flipOperation.reverse(currentWorld);
     expect(currentWorld).toEqual(current);
});
```

La première ligne correspond à la déclaration du bloc "it". Elle sert également à décrire ce qui sera testé, ce sera le message qui sera affiché lors de l'exécution et qui permettra de repérer quels sont les tests qui ont échoués.
Ensuite, on créé une copie du monde courant que l'on stocke dans la variable current. En effet, l'on souhaite vérifier que le monde n'as pas subit de modification au final, il faut donc sauvegarder son état initial. Le bloc "it" est extrait d'un test du monde des pancakes, le monde contient donc

la pile de pancake et deux entiers correspondant au compteur du nombre d'opération ainsi qu'à l'indice du dernier pancake retourné.
Ensuite on applique l'opération testée (ici il s'agit de flipOperation) au monde, puis on l'inverse. Enfin, on vérifie que le monde est identique à sa version originale que l'on avait sauvegardé dans current.

Ce test en particulier sert à compléter les tests existant puisqu'il existe déjà un bloc "it" pour tester la fonction apply, et un autre pour la fonction reverse, mais l'utilité des tests étant de détecter des comportements qui normalement ne devrait pas se produire, il a été décidé d'implémenter ce test ainsi que le test concernant reverse/apply pour rendre l'ensemble des tests plus exhaustif.

Le but global de ces tests est de s'assurer que les fonctions qui manipulent les différents objets présents à l'intérieur de PLM modifient correctement ces objets lorsqu'elles sont appelées. Ces tests servent également à assurer que lors du lancement en septembre, les élèves ne se retrouveront pas devant une page blanche. L'écriture de ces tests m'a notamment permis de détecter une erreur d'affichage dans l'univers des pancakes, que j'ai pu corriger.

3.3 Tests EndToEnd

La troisième et dernière partie du stage était consacrée à l'écriture de tests EndToEnd (E2E). Le but de ce type de test est de simuler le comportement d'un utilisateur navigant à travers l'application. Pour cela, il faut envoyer des commandes au navigateur qui correspondent aux action qu'un utilisateur effectuerait (cliquer sur un bouton, écrire du texte dans certaines zones, etc.)

La PLM utilise Angular.js comme librairie principale pour l'interface graphique. Par conséquent, j'ai décidé d'utiliser Protractor qui est un framework de test E2E pour les applications utilisant angular.js. L'avantage de Protractor est qu'il se synchronise avec la page actuellement testée, ce qui fait que l'on n'a en théorie pas besoin de gérer les latences dû au serveur ou au navigateur. Malheureusement la PLM utilise une websocket pour réaliser les échanges entre le client et le serveur, ce qui fait que Protractor ne peut plus se synchroniser (à cause des requêtes permanentes). J'ai donc du gérer manuellement la synchronisation entre Protractor et la page web qui était testée. Pour cela, avant chaque action que je souhaitais effectuer sur un élément, j'ai ajouté un test qui attend que l'élément concerné soit visible. Ceci afin de m'assurer que la partie de la page contenant l'élément avec lequel je voulais interagir soit bien chargée.

Protractor n'étant pas parfaitement adapté à la synchronisation manuelle, il en résulte que les tests sont instable : neuf fois sur dix le test va se passer correctement, et une fois sur dix Protractor va détecter que l'élément est visible avant qu'il soit complètement chargé, ce qui va poser problème lorsque l'on va tenter d'exécuter la ligne suivante. Dans le meilleur des cas, la ligne est ignoré, mais du coup les actions suivantes ne s'effectueront pas sur une page dans l'état souhaité et le test échoue, sinon l'exécution de la ligne renvoie une erreur et le test s'arrête.
Actuellement, les fonctions permettant d'effectuer de la synchronisation manuelle sont en cours d'amélioration, ainsi les tests gagneront peut-être en stabilité lorsque l'optimisation de ces fonctions sera terminé. Une solution alternative au problème est d'ajouter une temporisation entre chaque action en plus de la synchronisation manuelle, ce qui augmente la stabilité des tests mais augmente énormément la durée des tests.

Un autre avantage de Protractor est que les tests écrits peuvent facilement être mis à l'échelle. En effet, chaque test Protractor utilisant sa propre page du navigateur (chrome par défaut), il est possible d'en lancer autant que l'on souhaite en parallèle, la seule limitation étant la capacité de la machine. Ainsi, les tests écrits pourront être réutilisés lors de la conception de tests de charges. Il était prévu que j'en établisse dans le cadre de mon stage, mais par manque de temps je n'ai pas pu en implémenter.

Voici un exemple de bloc "it" de test Protractor (Le fichier complet est présenté en annexe), il simule un utilisateur qui réussit un exercice :

```
it('should_congrats_the_user_for_passing_the_exercise', function() {
    browser.executeScript("window.editor.setValue(\"avance();\");");//
       on ecrit le code de l'exercice
    browser.wait(until.visibilityOf(button), 5000, "Button_unclickable"
       );
    button.click();//on clique sur executer
    browser.wait(until.visibilityOf(congratsW), 5000, "Congrats_pop-up_
       isn't_here");
    browser.wait(until.textToBePresentInElement(congratsTitle,'Exercice
       _reussi'), 5000, "Congrats_pop-up_isn't_here");
    expect(congratsTitle.getText()).toEqual('Exercice_reussi');//on s'
       attend a ce que la fenetre de validation de l'exercice
       apparaisse
    congratsCross.click();
    browser.wait(until.visibilityOf(button), 500, "Pop-up_still_here");
});
```

Protractor utilise également le framework de test Jasmine, j'ai donc pu réutiliser les connaissances acquises lors de l'écriture des tests unitaires. On retrouve donc la déclaration du bloc "it" en première ligne avec la description du test.
Le bloc "BeforeEach" correspondant nous a amené sur la page du première exercice et l'on a déjà attendu qu'elle soit complètement chargé. Il n'y a donc pas à effectuer de synchronisation manuelle. On peut donc directement effectuer la première action qui consiste à écrire le code que l'on veut soumettre au serveur PLM dans la zone prévue. Le bloc "it" est extrait du test du premier exercice de la PLM, le code nécessaire pour réussir l'exercice est donc : "avance();".
Les lignes commençant par "browser.wait" sont les lignes de synchronisation manuelle. On peut voir qu'il y en a au minimum une avant chaque action qui attend un élément nécessaire à l'action suivante. Ces lignes disposent d'une durée au delà de laquelle Protractor considère que la condition ne sera pas remplie, et donc il passe à la suite en affichant le message associé.
Ensuite, on clique sur le bouton qui envoie le code au serveur. Si tout se passe bien, le serveur répond avec une liste d'instructions et le monde de l'exercice est modifié en conséquence.
La ligne débutant par "expect" correspond à la condition qui va déterminer si le test a été passé avec succès ou non. En l'occurrence on vérifie qu'il y a bien une pop-up avec écrit "Exercice reussi" qui apparaît.
Enfin, on ferme cette pop-up afin de retrouver une page "normale" (sans pop-up) pour pouvoir effectuer la suite du test (le bloc "it" suivant).

4 Bilan

Les tests écrits pendant mon stage permettent actuellement d'assurer un déploiement rapide du serveur, et garantissent une fiabilité dans le comportement interne de l'application, ainsi que le comportement global de la PLM vis à vis de l'utilisateur. Ces tests sont intégrés dans la version en production afin de garantir la stabilité de l'application.

L'une des améliorations possibles de la partie test de la PLM est la mise en place de tests pouvant généré la charge représentant un nombre d'utilisateurs donné. Ainsi, il suffira de se placer du coté du serveur de l'application pour déterminer les ressources nécessaires pour supporter tant d'utilisateurs.

Actuellement, la PLM n'est que très peu utilisée (moins de 200 utilisateurs par an). Le passage à la version web est une première étape dans la diffusion de la PLM à plus grande échelle, il est prévu de l'intégrer à un MOOC d'INRIA.

5 Conclusion

Lors de ce stage, j'ai pris conscience de l'importance et de l'ampleur que peuvent avoir les tests. Avoir à en implémenter sur un projet de l'envergure du projet PLM m'a amené à établir une méthodologie pour s'assurer de l'exhaustivité des tests.

J'ai aussi pu renforcer mes connaissances sur les scripts et les tests ainsi que découvrir les outils Docker et Protractor. Ce stage m'a également confronté à la difficulté de plonger dans un projet déjà existant et de comprendre son fonctionnement afin de le tester.

Ce stage c'est déroulé dans un laboratoire de recherche, ce qui m'a permis d'avoir un aperçu du monde de la recherche. J'ai eu l'occasion de contribuer à un projet en cours ce qui a été l'occasion pour moi d'apprendre à travailler en équipe. Ainsi, ce stage fût pour moi une expérience enrichissante aussi bien sur le plan personnel que professionnel.

Le projet PLM vient de faire un énorme bond en avant dans son développement avec son passage en version web, car à présent il n'y a plus aucun pré requis à son utilisation, si ce n'est le connaître, ce qui pourrait amener le projet à être utilisé mondialement.

Bibliographie / Webographie

[1] http://www.loria.fr/le-loria-1/organisation/organigramme. 3
[2] http://www.loria.fr/~quinson/Teaching/PLM/. 5
[3] https://angularjs.org/. 6
[4] http://codingbat.com/. 11
[5] http://jasmine.github.io/2.0/introduction.html. 11

Liste des illustrations

Glossaire

- E2E : EndToEnd
- jdk : java development kit
- LORIA : laboratoire lorrain de recherche en informatique et ses applications
- PLM : Programmer's Learning Machine

Annexes

A Exemple de test unitaire

```
(function(){
    'use strict';

    describe('FlipOperation', function() {
        var _FlipOperation;

        var currentWorld;
        var flipOperation;
        var number;
        var pancakeStack;
        var moveCount;
        var numberFlip;

        beforeEach(module('PLMApp'));

        beforeEach(inject(function(FlipOperation) {
            _FlipOperation = FlipOperation;
        }));

        beforeEach(function() {
            var i;
            var nbPancake;
            var memory;

            pancakeStack = [];
            nbPancake = getRandomInt(20) + 2;
            for(i = 0; i < nbPancake; i++) {
                pancakeStack.push({radius: i,upsideDown: true});
            }


            number = getRandomInt(nbPancake);
            moveCount = getRandomInt(42);
            numberFlip = getRandomInt(42);

            currentWorld = {
                pancakeStack: pancakeStack,
                moveCount: moveCount,
                numberFlip: numberFlip
            };

            var dataOperation = {
```

```
            number: number,
            oldNumber: numberFlip
        };

        flipOperation = new _FlipOperation(dataOperation);
    });

    it('should be initialized correctly by its constructor',
       function () {
        expect(flipOperation.number).toEqual(number);
        expect(flipOperation.oldNumber).toEqual(numberFlip);
    });

    it('should flip pancake between number and the end when applied
        ', function () {
        var length = currentWorld.pancakeStack.length;
        flipOperation.apply(currentWorld);
        for (var i=1;i<=number;i++){
            expect(currentWorld.pancakeStack[length-i].radius).
               toEqual(length-number+i-1);
            expect(currentWorld.pancakeStack[length-i].upsideDown).
               toEqual(false);
        }
        for (var i=number+1;i<=length;i++){
            expect(currentWorld.pancakeStack[length-i].radius).
               toEqual(length-i);
            expect(currentWorld.pancakeStack[length-i].upsideDown).
               toEqual(true);
        }
        expect(currentWorld.moveCount).toEqual(moveCount+1);
        expect(currentWorld.numberFlip).toEqual(number);
    });

    it('should reflip pancake between number and the end when
       reversed', function () {
        var length = currentWorld.pancakeStack.length;
        flipOperation.reverse(currentWorld);
        for (var i=1;i<=number;i++){
            expect(currentWorld.pancakeStack[length-i].radius).
               toEqual(length-number+i-1);
            expect(currentWorld.pancakeStack[length-i].upsideDown).
               toEqual(false);
        }
        for (var i=number+1;i<=length;i++){
            expect(currentWorld.pancakeStack[length-i].radius).
               toEqual(length-i);
            expect(currentWorld.pancakeStack[length-i].upsideDown).
               toEqual(true);
        }
        expect(currentWorld.moveCount).toEqual(moveCount-1);
        expect(currentWorld.numberFlip).toEqual(numberFlip);
    });
```

```
        it('should not change currentWorld when applied then reversed',
            function () {
            var current = {
                pancakeStack: currentWorld.pancakeStack.slice(),
                moveCount: moveCount,
                numberFlip: numberFlip
            };
            flipOperation.apply(currentWorld);
            flipOperation.reverse(currentWorld);
            expect(currentWorld).toEqual(current);
        });

        it('should not change currentWorld when reversed then applied',
            function () {
            var current = {
                pancakeStack: currentWorld.pancakeStack.slice(),
                moveCount: moveCount,
                numberFlip: number
            };
            flipOperation.reverse(currentWorld);
            flipOperation.apply(currentWorld);
            expect(currentWorld).toEqual(current);
        });
    });
})();
```

B Exemple de test Protractor

```
describe('Exo1', function() {

    var until = protractor.ExpectedConditions;
    var cross = element(by.css('.close-reveal-modal[ng-click]'));
    var button = element(by.css('.button-with-icon'));
    var redButton = element(by.css('ul:first-child li .button-with-icon
       .alert'));
    var congratsW = element(by.css('#successModal[style]'));
    var congratsTitle = element(by.css('#successModal h2 span'));
    var congratsCross = element.all(by.css('.close-reveal-modal')).get
       (2);

    beforeEach(function() {
        browser.get('https://plm.telecomnancy.univ-lorraine.fr/#/ui/
           lessons/welcome/');
        browser.ignoreSynchronization=true;
        browser.wait(until.or(until.visibilityOf(cross),until.
           visibilityOf(button)), 5000, "Pop-up isn't here");
        cross.isDisplayed().then(function(isVisible){
            if (isVisible)
                browser.actions().click(cross).perform();
        });
    });

    afterEach(function() {
        browser.executeScript('window.sessionStorage.clear();');
        browser.executeScript('window.localStorage.clear();');
    });

    it('should warn the user that world are unequal if no code is
       submitted', function() {
        browser.wait(until.visibilityOf(button), 500, "Button
           unclickable");
        button.click();
        browser.wait(until.visibilityOf(redButton), 5000, "Button isn't
           red");
        expect(protractor.ExpectedConditions.visibilityOf(redButton)())
           .toEqual(true);
    });

    it('should congrats the user for passing the exercise', function()
       {
```

```
        browser.executeScript("window.editor.setValue(\"avance();\");_"
            );
        browser.wait(until.visibilityOf(button), 5000, "Button_
            unclickable");
        button.click();
        browser.wait(until.visibilityOf(congratsW), 5000, "Congrats_pop
            -up_isn't_here");
        browser.wait(until.textToBePresentInElement(congratsTitle,'
            Exercice_reussi'), 5000, "Congrats_pop-up_isn't_here");
        expect(congratsTitle.getText()).toEqual('Exercice_reussi');
        congratsCross.click();
        browser.wait(until.visibilityOf(button), 500, "Pop-up_still_
            here");
    });
});
```

Résumé

L'objectif du projet PLM (Programmer's Learning Machine) est de proposer un outil permettant d'apprendre les bases de la programmation. Dans le cadre de son portage en application web, il a été nécessaire d'implémenter des tests unitaires (pour le comportement interne) ainsi que des tests de bouts en bouts (simulation d'un utilisateur). Une automatisation du déploiement de l'application a aussi été réalisée.

Mots-clés : Docker, test unitaire, test E2E

Abstract

The PLM (Programmer's Learning Machine) project's goal is to create a software which help beginners to learn to program. For the development of the web version, it has been mandatory to do unit testing (for intern behaviour) and EndToEnd testing (simulation of an user). An automation of the application deployment has been also realised.

Keywords : Docker, unitary testing, E2E testing