

CPES 3

Rapport de stage

Développement agile d'applications web

Auteur :
Barnabé GEFFROY

Référent :
Olivier CAILLOUX

19 juin 2020

Table des matières

Preliminaires	1
1 Gestion des presences	2
1.1 Attendance	2
1.2 JeSuisEnCours	2
2 Plaqueette-MIDO	3
2.1 L'automatisation du lancement du code	3
2.2 L'authentification	5
3 CredsRead	6
3.1 Diagramme de classe	6
4 Jugement delibere	7
 Annexes	 i
Codes Plaqueette-MIDO	ii
Codes CredsRead	iv

Introduction

Préliminaires

Lors de ce stage `Git` et `Maven` ont été utilisés sur la plupart des projets. En voici une présentation succincte.

Git

Fonctionnement

`Git` est un système de contrôle de version qui permet la collaboration entre développeurs. Le code source est conservé dans un *dépôt* distant. Il suit un modèle distribué, il n'y a pas de serveur central. Le code est donc accessible par plusieurs sources et peut être utilisé sans connexion. La connexion internet est cependant nécessaire pour envoyer ces modifications sur le dépôt distant. Ce genre de sauvegarde est appelé *commit*. Celle-ci est une version du code à instant donné. `Git` crée, avec tous les commits, une série d'instantanés qui rend la perte d'information très difficile.

`Git` gère également l'intégrité du code. Il peut y avoir des conflits, des parties identiques du code modifiées par deux utilisateurs. `Git` pointe les régions du code qui sont différentes et les utilisateurs éditent le code pour régler les zones de conflit.

Quelques commandes

- `git init` initialise un nouveau dépôt.
- `git clone` copie un dépôt `Git` déjà existant.
- `git add` ajoute les fichiers que l'on veut sauvegarder dans le commit.
- `git status` affiche l'état des fichiers (ajouté ou non).
- `git commit` crée un instantané du code en modifiant les fichiers ajoutés avec `git add`.
- `git push` envoie les commits sur le dépôt distant.

Seules les commandes `git clone` et `git push` nécessitent une connexion internet. Il est donc très aisé de travailler

GitHub

GitHub est un hébergeur de dépôts `Git`. Il offre la gestion de version distribuée, la fonctionnalité de gestion de code source de `Git`, ainsi que ses propres fonctionnalités. Il compte plus de 50 millions d'inscrits et plus de 100 millions de dépôts.

Maven

Maven est un outil de gestion de configuration de projet, en particulier de gestion des dépendances. Il permet de ne pas se soucier de l'environnement de compilation. Les dépendances sont indiquées dans un fichier nommé `pom.xml`. Grâce à ce fichier Maven configure les bibliothèque et autres dépendances. Maven propose aussi une structure du projet qui sera la même dans chacun des projets exposés.

Voici l'arborescence de base d'un projet Maven :

```
pom.xml
/src
  /main
    /java
    /resources
  /test
    /java
    /resources
```

Projet 1

Gestion des présences¹

L'offre de stage² auquel je me suis porté candidat portait initialement sur le développement d'une application gérant la présence des élèves du Master 1 MIAGE en Apprentissage. L'idée de ce projet était de digitaliser les feuilles de présences.

1.1 Attendance

Le projet Attendance avait donc pour but de développer une application web permettant au professeur de faire l'appel de sa classe et d'envoyer les données à l'administration. Dans un premier temps, j'ai conçu une application lisant un fichier JSON³ contenant une liste d'élèves et affichant celle-ci de manière à faire l'appel. Pour cela, le serveur HTTP Eclipse Jetty fournit les services web pour des applications Java. Le projet Attendance contient ainsi une classe `StudentsList` qui crée un fichier JSON contenant la liste des élèves. Une classe `RecordAttendance` récupère la liste et l'affiche dans une page HTML (voir Figure 1.1). Cette classe permet aussi de récupérer les informations entrées par l'utilisateur (les élèves absents cochés par le professeur).

```
1  [  
2      {  
3          "First Name": "Bertrand",  
4          "Last Name": "Russell"  
5      },  
6      {  
7          "First Name": "Friedrich",  
8          "Last Name": "Frege"  
9      }  
10 ]
```

Fichier JSON

The screenshot shows a web interface titled 'Liste des etudiants'. It contains a table with two columns: 'First Name' and 'Last Name'. The first row has 'Bertrand' and 'Russell' with a checked checkbox. The second row has 'Friedrich' and 'Frege' with an unchecked checkbox. Below the table is an 'Envoyer' button.

First Name	Last Name	
Bertrand	Russell	<input checked="" type="checkbox"/>
Friedrich	Frege	<input type="checkbox"/>

Envoyer

FIGURE 1.1 – Page HTML généré par `RecordAttendance`

1.2 JeSuisEnCours

Après quelques semaines, nous nous sommes rendus compte que la direction du projet était incompatible avec les exigences administratives. En effet, l'application prévoyait un simple appel du professeur, or l'étudiant doit personnellement attester de sa présence en émargeant un document. Il donc été décidé d'abandonner le projet intiale Attendance pour se tourner vers une application JeSuisEnCours, spécialisée dans la digitalisation les feuilles de présences.

Le but principal du projet est donc devenu la connexion de l'application JeSuisEnCours aux données de l'université, d'une part, pour accéder aux données (annuaires, emplois du temps,...), d'autre part pour gérer les éléments renvoyés par l'application (absence, justificatif,...).

Malheureusement, la crise sanitaire a fortement ralenti les contacts avec l'équipe de JeSuisEnCours et le projet a finalement été abandonné.

1. <https://github.com/barnabegeffroy/Attendance>

2. [Offre de stage: développement agile d'applications web et de bibliothèques open-source](#)

3. JSON est un format représentant les données de manière structurée.

Projet 2

Plaquette-MIDO⁴

Ce projet a pour but d'implémenter un code générant un fichier PDF détaillant les différents enseignements du Master 1 MIAGE en apprentissage à partir de la base de données de Dauphine. La finalité est d'automatiser le lancement du code de sorte que quotidiennement le fichier PDF soit mis à jour et publié en ligne.

À mon arrivée, un code permettait déjà la génération du fichier PDF. Certains passages du code étaient cependant à revoir pour améliorer l'esthétique du fichier PDF. De plus, le code initial était très peu généralisé à d'autres utilisateurs et plusieurs changements dans le code étaient nécessaires pour qu'un autre utilisateur puisse lancer Plaquette-MIDO. Il fallait donc davantage généraliser le code de façon à rendre accessible le code à d'autres utilisateurs. Le principal aspect à généraliser était l'authentification à l'API de Dauphine⁵, indispensable pour avoir accès aux données de l'université.

2.1 L'automatisation du lancement du code

La finalité du projet Plaquette-MIDO est de lancer la construction du fichier PDF quotidiennement et de le publier de manière à ce qu'il soit accessible sur le site de Dauphine. Pour réaliser cette automatisation, j'ai utilisé l'outil Travis-CI.

2.1.1 Travis-CI

Travis-CI est logiciel d'intégration continue qui permet de compiler, tester et déployer le code de dépôts GitHub. Il est configuré à partir d'un fichier nommé `.travis.yml` présent dans la racine du répertoire. Celui-ci est lu à chaque nouveau commit par Travis-CI qui exécute son contenu sur une machine virtuelle. L'exécution peut alors réussir dans le cas où aucune erreur n'a été signalée ou échouer si une erreur est survenue. Travis-CI peut donc s'assurer de la bonne compilation d'un projet. Il peut également effectuer des déploiements. En effet, il est possible d'insérer du script que Travis-CI exécute pour déployer des fichiers. Par exemple, ajouter un script avec les commandes git adéquat pour pousser des fichiers vers un dépôt. Travis-CI peut donc exécuter le code source permettant la création du fichier PDF et ensuite déployer ce fichier vers un dépôt GitHub.

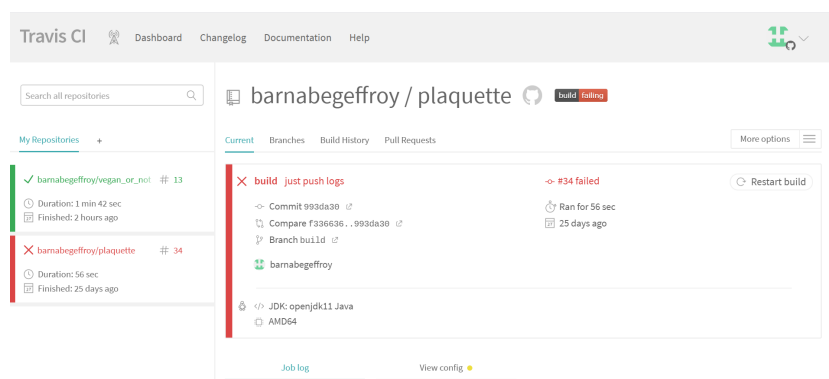


FIGURE 2.1 – Capture d'écran de l'interface de Travis-CI

4. <https://github.com/Dauphine-MIDO/plaquette-MIDO>

5. Une API est une interface de programmation d'application qui permet d'accéder à un ensemble de classes, méthodes, fonctions et autres données. Dans le cas de Dauphine, son API donne accès aux fonctions informatiques permettant de manipuler le programme des cours des différentes formations

La figure 2.1 montre que la construction du commit `just push logs` a échoué pour le dépôt `plaquette`, tandis que celle du dépôt `vegan_or_not` a réussi.

Travis-CI permet également de référencer des variables d'environnement sécurisées (clefs d'accès, identifiants, ...).

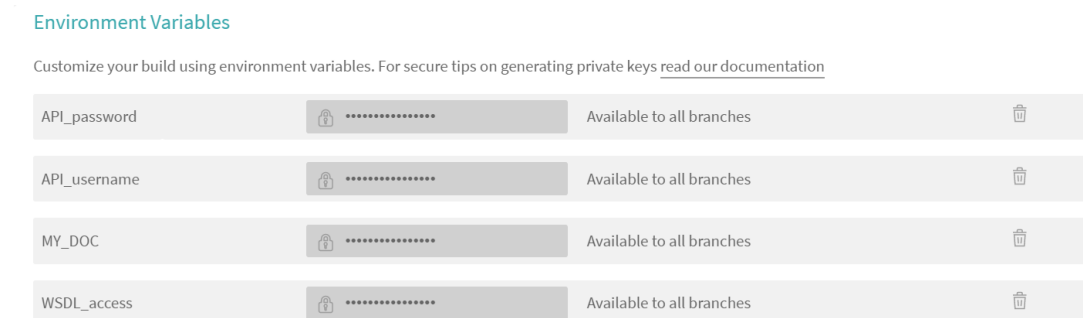


FIGURE 2.2 – Différentes variables d'environnement entrées pour la construction Travis-CI d'un dépôt

2.1.2 L'exécution du code

Les dépendances

Travis-CI identifie automatiquement un projet Maven et installe les dépendances indiquées dans le `pom.xml`. Dans le projet `Plaquette-MIDO`, la dépendance qui importe le code source de l'API de Dauphine nécessite un fichier texte nommé `WSDL_Login.txt` contenant un URL spécifique des identifiants de l'utilisateur. Ce fichier ne peut pas être dans le dépôt car il contient des informations personnelles. Il faut donc un script qui crée le fichier sur la machine virtuelle de Travis-CI. Avant de lancer l'installation des dépendances le fichier doit donc être créé. Dans le `.travis.yml`, on peut ajouter un script qui génère ce fichier.

Voici ci-dessous un extrait du script permettant la création d'un tel fichier. On y retrouve les variables d'environnement présentées dans la figure 2.2.

```
1 API_username = $(urlencode "${API_username}")
2 API_password = $(urlencode "${API_password}")
3
4 echo https://${API_username}:${API_password}@* > WSDL_login.txt
```

Création du fichier PDF et déploiement

Une fois que toutes les dépendances du projet Maven sont installées, il faut exécuter le code source qui génère le fichier PDF. La classe qui permet cette génération est `M1ApprBuilder`. Le script de Travis-CI va donc exécuter cette classe, il faudra ensuite déployer vers un dépôt d'arrivée le fichier PDF ainsi que le logs de la construction. La construction de Travis-CI doit échouer si le fichier PDF n'est pas généré. Néanmoins, dans tous les cas les logs doivent être poussés vers le dépôt. Ainsi, si le fichier PDF est généré, il est poussé avec les logs vers le dépôt d'arrivée. Si ce n'est pas le cas, la construction échoue, le dernier PDF déployé reste disponible. Nous sommes avertis immédiatement par e-mail de cet échec. Les logs du code source de `Plaquette-MIDO` ainsi que ceux de Travis-CI permettront alors rendre la compréhension de l'échec.

Vous trouverez le script exécuté par Travis-CI dans l'annexe à . Le script exécute plusieurs fonctions :

- `clean` (l.20 à 24), supprime les fichiers déjà existants.
- `get_current_deploy` (l.26 à 30), copie sur la machine virtuelle de Travis-CI le dépôt dans lequel les fichiers vont être déployés. Pour éviter de cloner un dépôt `Git` dans un autre dépôt `Git`, celui est cloné dans le dossier parent de `Plaquette-MIDO` de la machine virtuelle `"../$REPO"`.
- `build_doc` (l.31 à 41), essaie de générer le document en exécutant `M1ApprBuilder`. Elle met aussi à jour la variable `BUILT_EXIT_CODE` qui prend la valeur 0 si le code est correctement exécuté et que le fichier PDF est généré, ou prend la valeur 1 sinon.
- `deploy` (l.43 à 60), déplace les logs et le fichier PDF (s'il y en a un) vers le dépôt d'arrivée (`"../$REPO"`). Les différentes commandes `Git` présentées ultérieurement sont alors exécutées pour déployer les fichiers déplacés.

La dernière ligne `exit $BUILT_EXIT_CODE` renvoie à Travis-CI la valeur mis à jour dans `build_doc`. Si la valeur est 0, la construction continue et s'achève par un succès. Sinon la construction échoue et une notification est envoyée pour prévenir les développeurs.

Automatisation

Travis-CI lance initialement la construction du code du dépôt à chaque nouveau commit. Il est cependant possible de configurer des *Cron Jobs*. Ceux-ci vont renouveler la construction du code de manière quotidienne, hebdomadaire ou mensuelle.

Cron Jobs

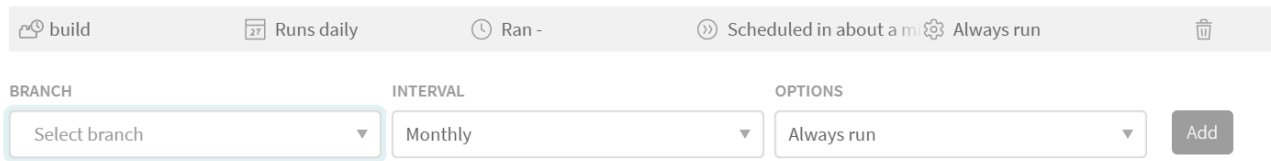


FIGURE 2.3 – Les *Cron Jobs* de Travis-CI, ici la construction est programmé quotidiennement

En programmant sur *Daily*, Travis-CI va relancer la construction du code tous les jours et ainsi renouveler le fichier PDF et les logs si des changements sont effectués. Si une erreur se produit et que le fichier PDF n'est pas généré, une e-mail nous prévient de la non-construction du code et le fichier PDF le plus récent sera toujours disponible sur le site.

2.2 L'authentification

Pour se connecter à l'API de Dauphine, un nom d'utilisateur et un mot de passe sont nécessaires. Le code initial prévoyait trois manières différentes de fournir ces informations afin de se connecter à l'API :

- les propriétés du système
- les variables d'environnement
- un fichier texte contenant les informations nécessaires

Seulement, ce code ne lisait initialement que le mot de passe et le nom d'utilisateur était une valeur par défaut. Un nouvel utilisateur devait donc modifier le code pour pouvoir utiliser ses identifiants. L'idée d'une valeur par défaut pour le nom d'utilisateur a donc abandonné pour rendre le programme plus accessible. La valeur du nom d'utilisateur serait lue de la même manière que celle du mot de passe.

2.2.1 La classe Authentication

Une nouvelle classe, **Authentication**, a donc été créée pour permettre la généralisation lecture du code et améliorer sa lisibilité. Celle-ci permet de créer un objet contenant un nom d'utilisateur et un mot de passe de type **Optional**. Ce type permet d'instancier aussi bien la valeur d'une chaîne de caractère que l'absence d'une information. Cette classe **Authentication** est lu dans une autre classe, **QueriesHelper**, qui permet de renvoyer les informations nécessaires pour se connecter à l'API. L'introduction de la classe **Authentication** permet ainsi de détecter si le nom d'utilisateur ou le mot de passe manquent et alors jeté une exception appropriée faisant échouer l'exécution du code.

2.2.2 Le projet CredsRead

La généralisation du code permettant l'authentification nous a poussé à le séparer dans un projet bien distinct de Plaquette-MIDO, le projet CredsRead. En effet, les méthodes de **QueriesHelper** ainsi que la classe **Authentication** n'avait, en grande partie, aucun lien spécifique avec plaquette-MIDO et pouvait ainsi être totalement publié dans un autre projet pour pouvoir être réutilisé plus facilement ce code dans d'autres projets. Le projet Creds-Read a ensuite été intégré au code source de Plaquette-MIDO.

Projet 3

CredsRead⁶

CredsRead, pour Credentials Read, gère comme son nom l'indique la lecture des identifiants d'un utilisateur.

3.1 Diagramme de classe

Le diagramme de classe permet de avoir une idée précise du code que l'on veut écrire. Papyrus est un outil permettant de réaliser ce genre de diagramme. Son interface intuitive facilite la rédaction d'un diagramme UML(le langage standard des diagrammes de classe) lisible et rigoureux.

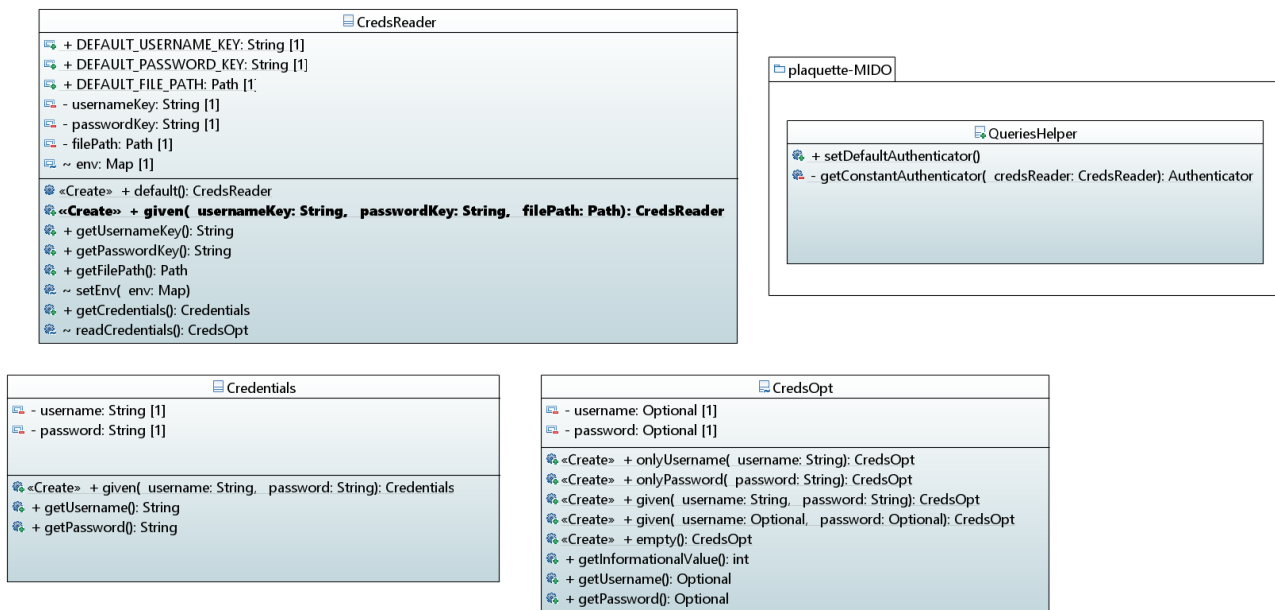


FIGURE 3.1 – Diagramme de classe du projet CredsRead

6. <https://github.com/oliviercailloux/creds-read>

Projet 4

Jugement délibéré⁷

7. https://github.com/barnabegeffroy/vegan_or_not

Conclusion

Annexes

Codes Plaque-MIDO

Scripts pour Travis-CI

writeWSDL.sh

```
1  #code from https://gist.github.com/cdown/1163649
2  urlencode() {
3      old_lc_collate=$LC_COLLATE
4      LC_COLLATE=C
5
6      local length="${#1}"
7      for (( i = 0; i < length; i++ )); do
8          local c="${1:i:1}"
9          case $c in
10             [a-zA-Z0-9.~_-]) printf "$c" ;;
11             *) printf '%%%02X' "'$c" ;;
12          esac
13      done
14
15      LC_COLLATE=$old_lc_collate
16  }
17
18  API_username = $(urlencode "${API_username}")
19  API_password = $(urlencode "${API_password}")
20
21  echo https://${API_username}:${API_password}@* > WSDL_login.txt
```

cibuild.sh

```
1  ##${ACCESS_TOKEN} is a personal access token from GitHub with the repo
   autorisations
2  set -e
3  REPO="receive_plaquette"
4  FILE_LOG="out.log"
5  FILE_PDF="out.pdf"
6  DEPLOY_REPO="https://${ACCESS_TOKEN}@github.com/barnabegeffroy/${REPO}.git"
7  BUILT_EXIT_CODE=1
8
9  function main {
10     clean
11     get_current_deploy
12     build_doc
13     if [ -z "${TRAVIS_PULL_REQUEST}" ]; then
14         echo "except don't publish doc for pull requests"
15     else
16         deploy
17     fi
18 }
```

```

19
20 function clean {
21     echo "Cleaning docs."
22     if [ -f "${FILE_LOG}" ]; then rm -f "${FILE_LOG}"; fi
23     if [ -f "${FILE_PDF}" ]; then rm -f "${FILE_PDF}"; fi
24 }
25
26 function get_current_deploy {
27     echo "Getting latest target deployment repository."
28     git clone --depth 1 ${DEPLOY_REPO} "../${REPO}"
29 }
30
31 function build_doc {
32     echo "Trying to generate document."
33     mvn dependency:build-classpath -Dmdep.outputFile=.classpath
34     if java -cp "target/classes:$(cat .classpath)" "io.github.oliviercailloux.
        plaquelette_mido_soap.M1ApprBuilder"; then
35         echo "Document generation succeeded."
36         BUILT_EXIT_CODE=0
37     else
38         echo "Document generation failed."
39         BUILT_EXIT_CODE=1
40     fi
41 }
42
43 function deploy {
44     echo "Deploying changes."
45     mv -f "${FILE_LOG}" "../${REPO}"
46     if test -f "${FILE_PDF}"; then
47         mv -f "${FILE_PDF}" "../${REPO}"
48     fi
49     cd "../${REPO}"
50     git config user.name "Travis CI"
51     git config user.email barnabe.geffroy@psl.eu
52     git add "${FILE_LOG}"
53     if test -f "${FILE_PDF}"; then
54         git add "${FILE_PDF}"
55     fi
56     git status
57     git commit -m "Lastest doc built on travis build $TRAVIS_BUILD_NUMBER auto-
        pushed to github (exit code ${BUILT_EXIT_CODE})"
58     git push ${DEPLOY_REPO} master
59     exit ${BUILT_EXIT_CODE}
60 }
61
62 main

```

Codes CredsRead

CredsReader

Bibliographie

- [1] Bash reference manual.
- [2] Jekyll theme for documentation. page 190.
- [3] Stéphane Airiau. Introduction à la programmation en java - cours 11. page 46.
- [4] Olivier Cailloux. Git.
- [5] Olivier Cailloux. Maven.
- [6] Olivier Cailloux and Yves Meinard. A formal framework for deliberated judgment. 88(2) :269–295.
- [7] Olivier Cailloux, Yves Meinard, and Nicolas Salliou. Deliberated diet.