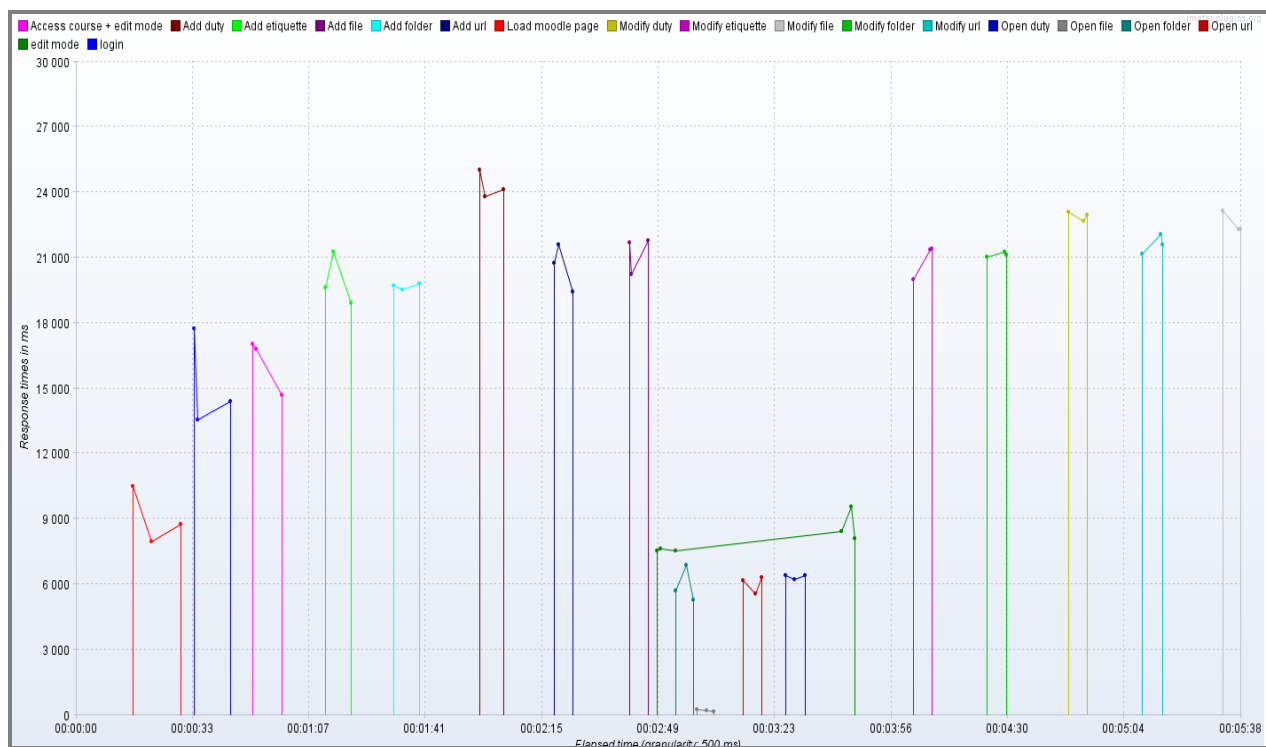


## Travail de Bachelor 2014

### Analyse de Performances



Déposé le 28.09.2014

Etudiant-e : François-Xavier Rieille

Professeur : Anne-Dominique Salamin

## 1. Résumé

La HES-SO dispose d'une plateforme d'apprentissage en ligne basée sur le système Moodle. Elle permet notamment aux professeurs de virtualiser leurs cours, et aux élèves d'y accéder.

L'applicatif Moodle est en constante évolution, de nouvelles versions sont régulièrement publiées.

Par rapport à ces changements, l'administration du système ne dispose actuellement pas d'un outil efficace pour analyser les performances des différents processus réalisés par les utilisateurs. Dans cette situation, les impacts en termes de performances des changements entrepris sur le système (mise à jour de la plateforme, développement interne, ...) ne peuvent pas être évalués avec suffisamment de pertinence.

Pour répondre à cette problématique, plusieurs logiciels d'analyse de performances ont été testés sous forme de Benchmark. L'outil JMeter a été sélectionné pour reproduire et analyser automatiquement les processus du site Moodle, de par ses capacités mises en évidence lors de ce Benchmark.

Une suite complète de tests a ensuite été créée et implémentée dans cet outil, de manière à ce qu'il puisse être déployé en production pour analyser les impacts des changements sur la plateforme.

Mots clés: performance, travail de Bachelor, analyse, JMeter, Selenium.

## 2. Avant-propos

Ce travail se positionne par rapport à l'appliquatif Moodle implémenté à la HES-SO (<http://cyberlearn.hes-so.ch/>), dont les performances ne peuvent pas aujourd'hui être appréciées valablement.

Actuellement, l'analyse des performances est réalisée manuellement; elle fournit des données moins précises et plus lentement que le ferait un logiciel.

Le présent travail a pour but d'automatiser ce processus d'analyse, pour permettre à l'administration de la plateforme de comparer les performances de différentes versions de l'appliquatif, de manière à détecter d'éventuelles variations.

Nous avons commencé notre travail par évaluer plusieurs outils de test de performance afin de trouver celui le mieux à même d'analyser une version modifiée du site Moodle.

Cette étape visait à tester les fonctions requises pour l'analyse du site Moodle, à savoir :

- la reproduction automatique des principaux processus utilisateur (connexion, ajout de ressources, ...).
- L'affichage des temps nécessaires à la réalisation des scénarios utilisateur ainsi que la synthèse de ceux-ci (graphiques, tableaux, ...).

L'outil le mieux adapté a ensuite été sélectionné pour y implémenter la suite complète de tests Moodle, de manière à fournir rapidement à la fois des informations de performances globales et des informations détaillées.

La principale difficulté rencontrée durant ce travail est liée aux nombreux codes sous-jacents relatifs aux boutons et aux autres éléments dynamiques. Certains outils testés ont échoué faute d'avoir été en mesure d'interagir avec ces éléments.

### 3. Table des matières

1. Résumé	2
2. Avant-propos	3
3. Table des matières	4
4. Liste des figures	12
5. Glossaire	17
6. Introduction	18
7. Descriptif de la plateforme Moodle	19
7.1. Description de l'applcatif	19
7.2. Implémentation HES-SO	19
7.3. Positionnement du travail	20
8. Tests d'acceptation	20
8.1. Définition	20
8.2. Scénarios	21
8.3. Suite de test	21
8.4. Définition du plan de test	21
8.5. Utilisation de ce type de test	22
9. Tests de performances	22
9.1. Objectif	22
9.2. Types de tests de performance	22
9.3. Points d'exécution	23
9.3.1. Depuis le datacenter (Load testing 1.0)	24
9.3.2. Depuis le cloud (Load testing 1.5)	24
9.3.3. Depuis la perspective de l'utilisateur (Load testing 2.0)	24
9.3.4. Sélection d'une approche de test	24
10. Navigateurs	25
11. Attentes par rapport au logiciel de test	26
11.1. Open Source	26
11.2. Navigateur	26

11.3.	Tests d'acceptation	27
11.4.	Test de performance	27
11.5.	Fiabilité des résultats	27
11.6.	Affichage des résultats	28
11.7.	Transportabilité	28
11.8.	Suite de tests	28
11.9.	Simplicité/Stabilité/rapidité	29
12.	Analyse des outils existants	29
12.1.	Selenium	30
12.2.	JMeter (Installation de base)	31
12.3.	Funkload	32
12.4.	TestMaker	32
12.5.	Gatling	33
12.6.	Fitnessse	34
13.	Sélection d'outils	35
13.1.	Selenium+ JMeter	35
13.1.1.	Description	35
13.1.2.	Motivation du choix	35
13.2.	TestMaker	36
13.2.1.	Description	36
13.2.2.	Motivation du choix	36
13.3.	FunkLoad	36
13.3.1.	Description	36
13.3.2.	Motivation du choix	36
13.4.	Gatling	37
13.4.1.	Description	37
13.4.2.	Motivation du choix	37
13.5.	JMeter + Web Driver Sampler	37
13.5.1.	Description	37
13.5.2.	Motivation du choix	38
14.	Inventaire des tests du Benchmark	38
14.1.	Analyse des parties testables	39
14.2.	Suite de test du Benchmark	40
14.2.1.	Définition	40

14.2.2.	Login	40
14.2.3.	Login	40
14.2.4.	Accéder au cours Tb_Fx	41
14.2.5.	Ajout d'un dossier dans la première section	41
14.3.	Choix d'un mode de connexion	41
14.3.1.	Contexte	41
14.3.2.	Problèmes liés au module AAI	42
14.3.3.	Choix d'un module de connexion	44
15.	Définition du Benchmark	45
15.1.	Définition de l'outil	45
15.2.	Installation de l'outil	45
15.3.	Enregistrement des tests (scénarios)	45
15.4.	Exécution des tests	45
15.5.	Création de la suite de tests	45
15.6.	Génération du rapport	46
15.7.	Evaluation de l'outil	47
16.	Benchmark PushToTest-Testmaker	47
16.1.	Modules	47
16.2.	Prérequis	48
16.3.	Installation	48
16.4.	Création des tests	49
16.5.	Exécuter un test	53
16.6.	Analyse de l'outil TestMaker	53
16.6.1.	Enregistrement des tests	54
16.6.2.	Exécution des tests	54
16.6.3.	Suite de tests	54
16.6.4.	Résultats obtenus lors du Benchmark	54
16.6.5.	Qualité des rapports	54
16.6.6.	Rapidité/Stabilité	54
17.	Selenium 2.0	55
17.1.	Modules Selenium	55
17.2.	Selenium IDE	56
17.2.1.	Domaine d'utilisation	56
17.2.2.	Installation	56

17.2.3.	Enregistrement d'un test	57
17.2.4.	Sauvegarder un test	58
17.2.5.	Limites du système	58
17.3.	Développement de tests Selenium	58
18.	Développement Selenium Web Driver	59
18.1.	Choix d'un langage	59
18.2.	Création des tests	60
18.3.	Création du projet Selenium (Eclipse)	60
18.4.	Création d'un test initial	61
18.5.	Exécution d'un test	61
18.6.	Syntaxe JUnit	62
18.6.1.	Annotations	62
18.6.2.	Syntaxe WebDriver	62
18.7.	Limites du système	63
19.	Benchmark WebDriver(Eclipse) – Jmeter	64
19.1.	Installation de base	64
19.2.	Installation des plugins Jmeter	64
19.3.	Composants Jmeter	65
19.3.1.	Thread Group	65
19.3.2.	JUnit Request	65
19.4.	Exécution des tests WebDriver (Eclipse)	66
19.5.	Création d'une suite de tests	68
19.6.	Analyse de l'outil JMeter – Web Driver(Eclipse)	68
19.6.1.	Enregistrement des tests	68
19.6.2.	Exécution des tests	69
19.6.3.	Suite de tests	69
19.6.4.	Résultats obtenus lors du Benchmark	69
19.6.5.	Rapidité/Stabilité	70
19.6.6.	Qualité des rapports	70
20.	Benchmark FunkLoad	70
20.1.	Installation de base	70
20.2.	Installer TCPWatch	71
20.3.	Installer GnuPlot	72
20.4.	Configuration du proxy avec Firefox	73

20.5.	Enregistrer un test	75
20.6.	Développement Funkload	76
20.6.1.	Fichier de test	76
20.7.	Fichier de configuration	77
20.8.	Exécuter un test	78
20.9.	Création d'une suite de tests	79
20.10.	Génération de rapports	79
20.11.	Analyse du logiciel FunkLoad	82
20.11.1.	Enregistrement des tests	82
20.11.2.	Suite de tests	82
20.11.3.	Résultats obtenus lors du Benchmark	83
20.11.4.	Stabilité/rapidité	83
20.11.5.	Qualité des rapports	83
21.	Benchmark Gatling	83
21.1.	Installation	84
21.2.	Enregistrer un test	84
21.3.	Exécuter un test	86
21.4.	Simplifier un test généré par le Recorder	87
21.5.	Syntaxe utilisée	87
21.5.1.	Exec/pause	87
21.5.2.	Group	88
21.5.3.	Set Up	88
21.5.4.	Fichiers de test	89
21.6.	Création d'une suite de tests	89
21.6.1.	Fichier de simulation	90
21.6.2.	Création des scénarios	91
21.7.	Contrôles	92
21.8.	Rapports	93
21.9.	Analyse de l'outil Gatling	94
21.9.1.	Enregistrement des tests	94
21.9.2.	Exécution des tests	94
21.9.3.	Suite de tests	94
21.9.4.	Résultats obtenus lors du Benchmark	95
21.9.5.	Qualité des rapports	95
21.9.6.	Rapidité/Stabilité	95



21.9.7.	Problèmes liés à l'ajout d'une ressource	95
22.	Benchmark JMeter + Web Driver Sampler	98
22.1.	Installation de base	98
22.2.	Installation du Web Driver Sampler	98
22.3.	Installation des plugins JMeter	99
22.4.	Création de la structure	99
22.5.	Edition du code	102
22.5.1.	Navigation	102
22.5.2.	Analyse des performances	102
22.5.3.	Clic sur un bouton/liens	103
22.5.4.	Remplir un champ	103
22.5.5.	Valider une alerte javascript	104
22.5.6.	Navigation entre les Frames	104
22.6.	Création d'un test	105
22.7.	Exécution d'un test	105
22.8.	Création de la suite de tests	105
22.9.	Génération des rapports	105
22.10.	Analyse de l'outil JMeter + Web Driver Sampler	106
22.10.1.	Enregistrement des tests	106
22.10.2.	Exécution des tests	106
22.10.3.	Suite de tests	106
22.10.4.	Résultats obtenus lors du Benchmark	106
22.10.5.	Stabilité/rapidité	107
22.10.6.	Qualité des rapports	107
23.	Nouvelle analyse des outils	107
23.1.	Introduction	107
23.2.	Telerik Test Studio	107
23.2.1.	Description	107
23.2.2.	Versions	108
23.3.	Analyse théorique des capacités	108
24.	Benchmark Telerik Test Studio	109
24.1.	Installation	109
24.2.	Enregistrement d'un test	110
24.2.1.	Création du projet	110

24.2.2.	Création d'un test	112
24.3.	Exécution d'un test	113
24.4.	Création de la suite de tests	114
24.5.	Génération des rapports	114
24.6.	Analyse de l'outil Test Studio	116
24.6.1.	Enregistrement des tests	116
24.6.2.	Exécution des tests	116
24.6.3.	Suite de tests	116
24.6.4.	Résultats obtenus lors du Benchmark	116
24.6.5.	Stabilité/rapidité	116
24.6.6.	Qualité des rapports	117
25.	Extention FirePath	117
26.	Sélection d'un outil	118
26.1.	Rétrospective des outils testés	118
26.1.1.	PushToTest TestMaker	118
26.1.2.	JMeter + Selenium Web Driver (Eclipse)	118
26.1.3.	Funkload	118
26.1.4.	Gatling	119
26.1.5.	JMeter (Web Driver Sampler)	119
26.1.6.	Test Studio	119
26.2.	Comparaison entre JMeter et Test Studio	119
26.2.1.	Enregistrement de tests	120
26.2.2.	Suite de tests	121
26.2.3.	Rapports	121
26.2.4.	Généralités	122
26.3.	Sélection	122
27.	Définition de la suite de tests Moodle Complète	123
27.1.	Plan de test : Profil Enseignant	123
27.2.	Plan de test : Profil Etudiant	125
28.	Implémentation de la suite de tests Moodle complète	126
28.1.	Résultats	126
28.2.	Variables de script	127
28.3.	Limitations	128

28.3.1.	Contrôle des éléments	128
28.3.2.	Frame de description	129
28.4.	Génération des rapports	130
28.5.	Exportation du rapport	131
29.	Implémentation sur le serveur Moodle Old	131
30.	Conclusion	131
31.	Références	133
31.1.	Livres	133
31.2.	Sources électroniques	133
32.	Annexes	135
32.1.	Installateurs des outils	135
32.2.	Scripts de test développés	135
32.3.	Suite de test définitive	135
32.4.	Planification	135
32.5.	Rapports hebdomadaires	137
32.6.	Cahier des charges	137
33.	Déclaration	137

## 4. Liste des figures

Figure 1 Logo Moodle(moodle.org) .....	19
Figure 2 : Points d'exécution (Web Load Testing for dummies – Scott Barber) .....	23
Figure 3 Principaux navigateurs utilisés dans le monde (http://w3schools.com/) .....	25
Figure 4 Principaux navigateurs utilisés en Suisse (http://gs.statcounter.com/) .....	26
Figure 5 logo Selenium (seleniumhq.org) .....	30
Figure 6 logo JMeter (jmeter.apache.org) .....	31
Figure 7 Logo FunkLoad (funkload.nuxeo.org) .....	32
Figure 8 Logo TestMaker (pushtotest.com).....	32
Figure 9 Logo Gatling (gatling-tool.org) .....	33
Figure 10 Logo Fitnesse (fitnesse.org) .....	34
Figure 11 Performances Funkload (screenshot) .....	37
Figure 12 Tests du Benchmark (screenshot Visio) .....	40
Figure 13 Connexion AAI (screenshot Cyberlearn).....	42
Figure 14 Connexion Moodle (screenshot Cyberlean) .....	42
Figure 15 Connexion AAI (screenshot Cyberlearn).....	43
Figure 16 Cookies connexion Moodle (screenshot // CookieManager+) .....	43
Figure 17 graphique performances Moodle (screenshot // JMeter) .....	46
Figure 18 TestMaker structure (screenshoot - pushtotest.com/user guide) .....	48
Figure 19 Propriétés testmaker32.cmd (screenshot // XP sp3) .....	49
Figure 20 contenu TestMaker (screenshot // XP sp3) .....	49
Figure 21 Tools menu (screenshot - TestMaker) .....	50
Figure 22 start url Designer (screenshot // TestMaker) .....	50
Figure 23 export test (screenshot - Designer).....	51

Figure 24 Create test (screenshot - TestMaker) .....	51
Figure 25 Create load test TestMaker (screenshoot) .....	52
Figure 26 Creation d'un scénario (screenshot,TestStudio).....	53
Figure 27 scénrios TestMaker (screenshot) .....	53
Figure 28 accès Selenium IDE (screenshot - Firefox).....	56
Figure 29 Interface Selenium IDE (screenshot) .....	57
Figure 30 Projet Selenium-Eclipse(screenshot) .....	61
Figure 31 start test (screenshot - Eclipse) .....	61
Figure 32 Résultats exécution JUnit (screenshot).....	62
Figure 33 Thread group JMeter (screenshot) .....	65
Figure 34 Exportation projet Selenium(screenshot - Eclipse).....	66
Figure 35 Exportation .jar (screenshot - Eclipse) .....	67
Figure 36 Structre JMeter 1 (screenshot) .....	67
Figure 37 Load Time Sampler (screenshot - JMeter) .....	68
Figure 38 Variable d'environnement Python (screenshot - XP sp3) .....	71
Figure 39 Variable d'environnement Gnuplot (screenshot - XP sp3) .....	72
Figure 40 Interface Gnuplot (screenshot) .....	73
Figure 41 Menu Firefox v28 (screenshot) .....	74
Figure 42 Onglet réseau Firefox v28 (screenshot).....	74
Figure 43 Configuration proxy Firefox 28 (screenshot) .....	74
Figure 44 terminal enregistrement test Funkload (screenshot) .....	75
Figure 45 Fichiers Funkload (screenshot).....	76
Figure 46 test Funkload exécuté (screenshot).....	78
Figure 47 Exécution test de performance Funkload (screenshot - 1 thread) .....	80

Figure 48 Génération rapport html Funkload (screenshot).....	81
Figure 49 Chargement pages rapport Funkload (screenshot).....	81
Figure 50 Détails exécution test Funkload (screenshot) .....	81
Figure 51 Graphique rapport Funkload (screenshot) .....	82
Figure 52 Gatling - recorder window.....	85
Figure 53 requêtes chargement Moodle (screenshot - Chrome).....	86
Figure 54 Affichage des tests (screenshot - Gatling).....	86
Figure 55 requêtes annulées Gatling (screenshot) .....	87
Figure 56 Schéma suite de tests Gatling (screenshot - Visio).....	90
Figure 57 - Ajout d'un fichier sans connexion à Moodle – erreur (screenshot - Gatling).....	92
Figure 58 - Ajout d'un fichier en tant que professeur (screenshot - Gatling).....	93
Figure 59 Résultats bruts Gatling (screenshot).....	93
Figure 60 Graphique threads Gatling (screenshot) .....	93
Figure 61 Graphique temps requêtes Gatling (screenshot) .....	94
Figure 62 Boutons d'enregistrement (screenshot) .....	96
Figure 63 Requête POST vers modedit.php (screenshot) .....	96
Figure 64 2ème envoi de la requête POST (screenshot) .....	97
Figure 65 Erreur générée par modedit.php (screenshot) .....	97
Figure 66 JMeter structure (capture d'écran).....	99
Figure 67 JMeter sampler result (screenshot) .....	100
Figure 68 View result tree (screenshot - JMeter) .....	100
Figure 69 JMeter Threads over time.....	101
Figure 70 JMeter aggregated display (screenshot).....	102
Figure 71 JMeter Response time (screenshot) .....	102

Figure 72 Exemple de Iframe Moodle .....	104
Figure 73 Bouton de lancement JMeter (screenshot) .....	105
Figure 74 Logo TestStudio (www.telerik.com ) .....	107
Figure 75 Différentes versions de Test Studio.....	108
Figure 76 Installation Test Studio (screenshot) .....	110
Figure 77 Sélection de la version Test Studio (screenshot).....	110
Figure 78 menu projet Test Studio (screenshot).....	110
Figure 79 Création d'un projet Test Studio (screenshot) .....	111
Figure 80 Structure des scénarios Test Studio (screenshot) .....	112
Figure 81 Enregistrement d'un test Test Studio (screenshot) .....	112
Figure 82 Requête test (screenshot - TestStudio) .....	112
Figure 83 Propriétés d'une requête (screenshot - TestStudio) .....	113
Figure 84 Lancement test (screenshot - TestStudio) .....	113
Figure 85 Test d'acceptation Test Studio.....	114
Figure 86 Scénarios TestStudio (screenshot) .....	114
Figure 87 Bandeau Test Studio.....	115
Figure 88 Rapport Test Studio .....	115
Figure 89 Instances Test Studio.....	115
Figure 90 Comparaison résultats Test Studio.....	115
Figure 91 Interface Firepath (addons.mozilla.com) .....	117
Figure 92 Schéma plan de test - professeur .....	124
Figure 93 Schéma plan de test - Etudiant.....	125
Figure 94 Liste des tests JMeter - def (screenshot) .....	127
Figure 95 Variables JMeter thread-id (screenshot) .....	128

Figure 96 Editeur de texte Moodle (screenshot) .....	129
Figure 97 IFrame relative à l'éditeur de code source (screenshot - Firepath).....	129
Figure 98 Erreur liée à un mauvais id (screenshot) .....	129
Figure 99 Zeroing lines (JMeter).....	130
Figure 100 Exportation rapport JMeter (screenshot).....	131



## 5. Glossaire

**Header (en-tête) :** Comme cela est décrit sur le site w3.org (<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>), le header permet de définir les paramètres d'une transaction http. Par exemple, il permet de définir l'encodage, le langage ou les types de fichier acceptés pour les échanges entre le client et le serveur.

**Scénario de test :** D'après le site <http://pic.dhe.ibm.com>, un scénario de test est composé de requêtes regroupées sous la forme d'un script de test, et répond à la question « Que vais-je tester ? ».

**Suite de tests :** D'après le site <http://pic.dhe.ibm.com>, une suite de tests correspond au regroupement de plusieurs scénarios utilisateurs, placés dans un ordre logique (connexion, achat d'un produit, paiement, déconnexion).

**Script de test :** D'après <http://softwaretestingfundamentals.com>, un script de test est un fichier contenant les instructions qui seront exécutées sur le système testé. Nous pouvons distinguer à ce niveau deux types de tests :

- Les tests manuels qui sont exécutés manuellement par l'administrateur.
- Les tests automatisés qui sont réalisés au moyen d'un logiciel spécialisé.

**Thread/utilisateur virtuel :** D'après <http://fr.openclassrooms.com/>, un thread est un processus pouvant être exécuté en parallèle (multitâche). Dans le cadre de notre travail, il va nous permettre de tester l'accès à la plateforme Moodle via plusieurs utilisateurs connectés en parallèle de manière à comparer les performances de chacun d'entre eux.

**Unit testing :** Un test unitaire permet de tester une fonction logique, et de contrôler que la valeur retournée correspond à ce qui est attendu. Cette description est tirée du site suivant : <http://artofunittesting.com/definition-of-a-unit-test/>

**Functional testing :** D'après <http://stackoverflow.com/questions/3370334/difference-between-acceptance-test-and-functional-test>, les tests fonctionnels ont pour but de contrôler que les fonctionnalités d'une application sont conformes à celles qui ont été prévues. Le résultat de ce genre de tests est binaire (la fonctionnalité est-elle conforme ou pas).

**Acceptance testing :** Consiste à tester une application dans son contexte, à simuler les interactions humaines de manière à déterminer si l'application répond à la problématique de départ. Pour ce faire, les tests d'acceptation doivent être relatifs à un scénario proche de la réalité de l'utilisateur du système à tester. Contrairement aux tests unitaires, les tests

d'acceptance sont utilisés à un niveau supérieur, pour tester les fonctionnalités métier du système. Ces informations sont tirées du site suivant : <http://stackoverflow.com/questions/3370334/difference-between-acceptance-test-and-functional-test>

Pour plus d'informations à ce sujet, se référer au chapitre 8.

**IDE (Integrated development environment):** Un IDE est un logiciel proposant un aide au développement pour les programmeurs. Un IDE est composé de plusieurs modules, comme un éditeur de source ou un débogueur (<http://searchsoftwarequality.techtarget.com>).

**Performance testing :** Se référer au chapitre 9.

## 6. Introduction

La HES-SO dispose actuellement d'un système d'apprentissage en ligne, basé sur l'appliquatif gratuit Moodle. Rien que pour l'année 2013, la plateforme a recensé pas moins d'un million de visites d'après le rapport d'activités.

Vu la fréquentation importante de ce service, les performances de celui-ci s'avèrent capitales. C'est dans ce but que chaque modification de la plateforme (nouvelle version de l'appliquatif ou développement interne) doit être précédée par une analyse des impacts sur les performances du point de vue de l'utilisateur.

L'administration de la plateforme Moodle ne disposant pas d'un outil d'analyse automatique, le but de ce travail vise à trouver un outil capable de reproduire les processus utilisateur de la plateforme, et d'en tirer des informations de performance. L'administration sera alors en mesure d'analyser de manière rapide et pertinente les effets d'un changement sur l'appliquatif avant de passer en mode production.

Ce travail a commencé par l'analyse des outils existants et la détermination de leurs capacités théoriques.

Les outils les plus intéressants ont ensuite été testés sous forme de Benchmark. En fonction des résultats de celui-ci, nous avons été en mesure de comparer les différents outils, et de ressortir le logiciel qui correspond le mieux aux attentes.

Finalement, une suite de tests a été créée sur la plateforme de développement avec l'outil sélectionné, et les performances de cette version ont pu être dégagées.

## 7. Descriptif de la plateforme Moodle

Dans ce chapitre, nous allons décrire la plateforme Moodle implémentée à la HES-SO. Nous commencerons par définir le système Moodle en général, avant de parler des parties constitutives de la gestion du site au niveau de la HES-SO. Finalement, nous positionnerons le présent travail par rapport à cette structure.

### 7.1. Description de l'appliquatif



Figure 1 Logo Moodle(moodle.org)

Moodle est une plateforme d'apprentissage en ligne, diffusée sous licence libre. D'après leur site officiel (<http://docs.moodle.org/27/en/History>), la première version de la plateforme a été lancée en 2002. Depuis, de nombreuses améliorations ont été apportées (multi-langue, thèmes, ...).

Implémenté au niveau de la HES-SO, cet applicatif permet notamment aux professeurs d'ajouter des ressources à leur cours virtuels (devoirs, documents, quizz), et aux élèves, d'y accéder. D'après le rapport d'activité publié en 2013, la plateforme a du faire face à plus d'un million de visites, pour 3651 Go de bande passante. Au 16 décembre 2013, 26'473 utilisateurs différents étaient inscrits sur le site Moodle de la HES-SO.

Nous remarquons à ce niveau, l'importance que relève l'analyse des performances de la plateforme, afin de garantir aux utilisateurs un accès rapide aux fonctionnalités.

### 7.2. Implémentation HES-SO

La gestion du Site Moodle est séparée sur trois zones géographiques, qui correspondent à trois différentes couches:

- **Besoins utilisateurs** : L'analyse et la gestion des besoins utilisateur sont réalisés à Sierre, tout comme l'évaluation des nouvelles versions du site Moodle.
- **Couche applicative** : La gestion de l'appliquatif Moodle est réalisée à Yverdon. Les mises à jour du système sont effectuées par les employés de cette structure.

- **Hébergement physique** : L'hébergement physique des serveurs se passe à Fribourg. Les analyses de performances des machines sont réalisées depuis cet emplacement.

### 7.3. Positionnement du travail

Le but de ce travail vise à aider les administrateurs de la plateforme Moodle à évaluer les performances des nouvelles versions de l'applicatif, dans le but de déterminer les impacts d'une éventuelle migration du point de vue de l'utilisateur.

Les tests réalisés dans le cadre de ce travail devront donc être relatifs aux performances des processus utilisateurs, et non pas aux performances des serveurs Moodle car ceux-ci sont déjà analysés par une équipe située à Fribourg.

L'analyse des performances liée aux changements apportés à la plateforme Moodle est réalisée par l'équipe de Sierre, c'est donc dans cet environnement que notre travail se positionne.

## 8. Tests d'acceptation

Le présent travail devra fournir un outil capable d'exécuter automatiquement différents scénarios sur la plateforme Moodle. L'objectif de ce chapitre est de comprendre le type de test qui sera utilisé pour répondre à cette contrainte.

### 8.1. Définition

D'après <http://www.develop.com/useracceptancetests>, les tests d'acceptation (UAT) permettent de mesurer la qualité d'une application, tout en contrôlant que les scénarios utilisateurs fonctionnent correctement. Le résultat d'un test d'acceptation est binaire : Il indique si le scénario en cours de test est valide ou pas.

Dans le cas d'un test d'acceptation automatisé, un outil devra être en mesure de reproduire automatiquement un test d'acceptation(UAT), de manière à ce que l'application puisse être testée rapidement et à tout moment.

## 8.2. Scénarios

Les scénarios sont composés de plusieurs tests d'acceptation simples, par exemple :

- Login
- Ouverture d'un cours
- Récupération d'une ressource
- Fermeture de la ressource
- Logout

## 8.3. Suite de test

Une suite de test comprend plusieurs scénarios, et constitue une stratégie de test complète pour une application.

- Scénario pour les ressources
- Scénario pour le blog
- Scénario admin
- ....

## 8.4. Définition du plan de test

La définition des tests d'acceptation passe par la procédure suivante d'après l'ouvrage de Scott Barber « Web Load Testing for Dummies » :

- Analyser les besoins de l'application
- Identifier les scénarios
- Définir le plan de test
- Implémenter les tests dans le logiciel
- Exécuter les tests
- Enregistrer les résultats
- Confirmer que les objectifs sont atteints

Cette procédure devra être utilisée lors de la définition du plan de test complet de la plateforme Moodle.

### 8.5. Utilisation de ce type de test

Les tests d'acceptation seront utilisés pour reproduire les scénarios utilisateurs et les grouper sous la forme d'un plan de test. Nous parlons ici d'une exécution des scénarios totalement automatique, l'outil sera donc en mesure d'effectuer les actions de l'utilisateur sans intervention de sa part.

## 9. Tests de performances

Nous évoquons ici un autre type de test, qui sera utilisé cette fois-ci pour évaluer les performances des scénarios utilisateurs de la plateforme Moodle. Les informations présentées sont tirées du livre de Scott Barber « Web Load Testing for Dummies ».

### 9.1. Objectif

L'analyse du site Moodle devra permettre de ressortir les temps d'exécution des scénarios utilisateurs, qui pourront être utilisés pour comparer les performances de différentes versions de l'appli. Ces informations pourront aussi être utilisées pour justifier ou désapprouver une migration vers une nouvelle version de la plateforme.

### 9.2. Types de tests de performance

Les tests de performance peuvent être réalisés de différentes manières, dont voici les principales :

- **Tests de performance simples** : Ces tests permettent de mesurer les temps d'exécution d'une page ou d'un scénario utilisateur, en général avec un seul utilisateur connecté depuis le logiciel. Cette méthode est l'équivalent automatisé de la mesure manuelle du chargement d'une ressource, un chronomètre à la main.
- **Tests de charge** : Les tests de charge consistent à accéder à une ressource avec plusieurs utilisateurs connectés en parallèle. Cela permet de simuler une certaine charge sur le système, et de constater les variations de performance qui en découlent.

- **Stress test** : Il consiste à déterminer les conditions de charge dans lesquelles l'application ne fonctionne plus correctement ou n'est tout simplement plus accessible.
- **Tests d'endurance** : Qui consiste à exécuter un test de charge sur une durée importante (50 heures par exemple).

Par rapport à cette liste, les tests qui nous intéressent sont surtout les tests de performance simples. En effet, le but est avant tout de constater les temps de chargement des scénarios utilisateur, sans forcément simuler une charge importante sur les serveurs Moodle.

Il va de soi qu'une mesure de performance individuelle est moins intéressante qu'une moyenne obtenue à partir de plusieurs échantillons. On peut donc imaginer l'exécution de plusieurs tests de performance simples en parallèle (threads), de manière à obtenir un lot de valeurs représenté sous forme de moyenne/graphique.

### 9.3. Points d'exécution

Nous allons maintenant analyser les différents emplacements à partir desquels les tests de performance peuvent être réalisés. Nous allons ensuite déterminer le point d'exécution le plus adapté pour répondre aux besoins de ce travail.

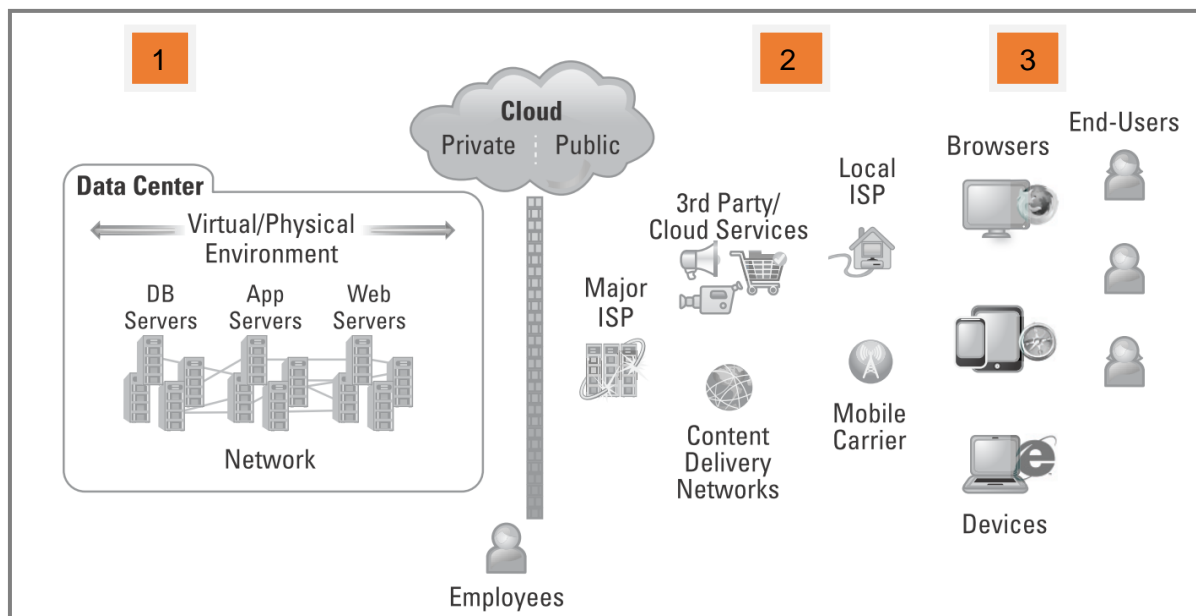


Figure 2 : Points d'exécution (Web Load Testing for dummies – Scott Barber)

### 9.3.1. Depuis le datacenter (Load testing 1.0)

Dans cette première situation, les tests sont réalisés à l'intérieur du datacenter (centre de traitement de données), où se trouve hébergé le site Moodle (devant le firewall). Cette approche peut être utilisée pour tester les performances des composants physiques, de manière à atteindre un certain résultat en charge. Mais ce résultat ne sera pas représentatif des utilisateurs finaux, car un certain nombre de paramètres n'ont pas été pris en compte dont notamment :

- Les éléments entre le datacenter et le réseau internet.
- La connexion entre l'utilisateur et notre réseau.
- La machine et le navigateur de l'utilisateur.

### 9.3.2. Depuis le cloud (Load testing 1.5)

Les tests sont maintenant réalisés en dehors de notre réseau interne, à savoir derrière le firewall. Mais comme cela peut être remarqué sur le schéma présenté à la page précédente, de nombreux éléments comme le fournisseur internet du client (ISP) n'ont pas été pris en compte. Les résultats d'une analyse réalisée à ce niveau ne sont toujours pas comparables à l'expérience utilisateur réelle.

### 9.3.3. Depuis la perspective de l'utilisateur (Load testing 2.0)

Qualifié d'approche **Inside-Out**, le Load Testing 2.0 se place du côté de l'utilisateur, ce qui permet de se mettre à la place de celui-ci et de constater si les attentes au niveau des performances sont atteintes. De plus, cette approche est en mesure de déterminer d'éventuels ralentissements provoqués par l'environnement du client (connexion internet, version et type de navigateur, emplacement géographique, ...).

### 9.3.4. Sélection d'une approche de test

En fonction des informations présentées ci-dessus et dans le but de réaliser des tests de performances proches de la réalité de l'utilisateur, une approche de type **Inside Out** sera adoptée pour l'analyse de la plateforme Moodle. Les tests seront donc exécutés en dehors du parc des serveurs Moodle (situé à Fribourg) et depuis un emplacement proche de celui utilisé par les étudiants et professeurs (Suisse romande).



## 10. Navigateurs

Un logiciel d'analyse de performance positionné du côté client devra permettre de simuler au moins un navigateur internet. Si cela est possible, l'outil devra simuler un navigateur représentatif sur le marché Suisse, de manière à ce que les résultats obtenus puissent être comparés à ceux des utilisateurs réels de la plateforme.

Les parts de marché des différents navigateurs ont considérablement varié avec le temps. Le graphique suivant (source: <http://w3schools.com/>) nous donne une idée sur les tendances actuelles au niveau du monde entier :

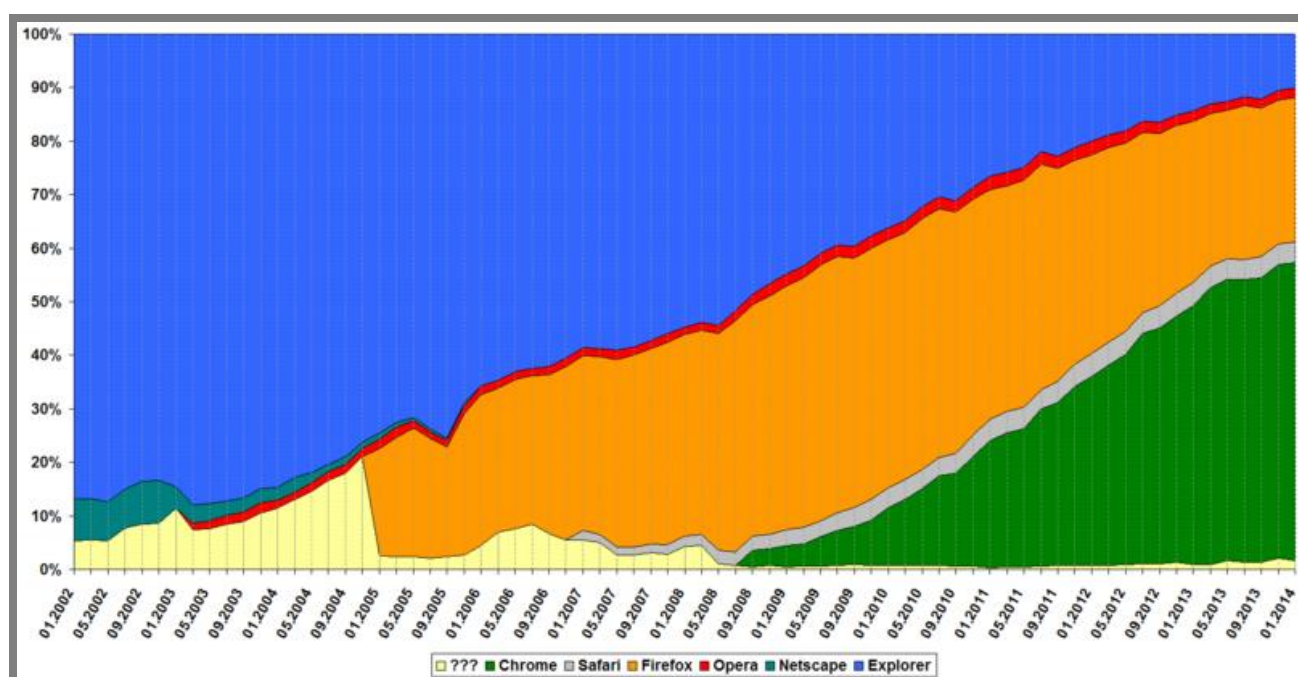


Figure 3 Principaux navigateurs utilisés dans le monde (<http://w3schools.com/>)

D'après un graphique de StatCounter (<http://gs.statcounter.com/>) présenté à la page suivante, et mis à jour en mars 2014, Internet Explorer et Safari sont particulièrement présents en Suisse, presque au même rang que Chrome ou Firefox.

Nous pouvons donc en déduire que les de navigateurs susceptibles d'être utilisés par les clients de la plateforme Moodle sont respectivement : Internet Explorer, Google Chrome, Mozilla Firefox et Safari.

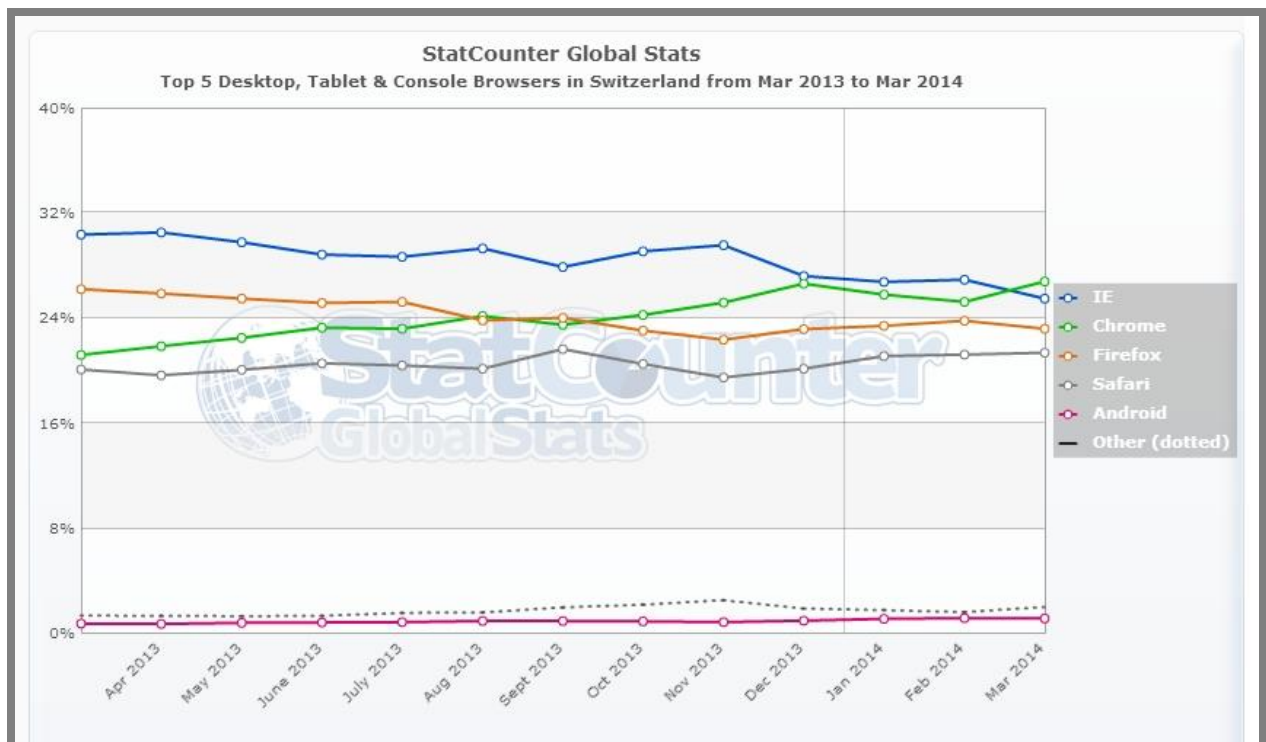


Figure 4 Principaux navigateurs utilisés en Suisse (<http://gs.statcounter.com/>)

## 11. Attentes par rapport au logiciel de test

L'ouvrage « Web Load Testing For Dummies » de Scott Barber nous donne de nombreuses informations concernant les attentes que l'on pourrait avoir face à un outil de test de performance. En combinaison avec les informations transmises par les mandataires de ce travail, nous allons dégager les principales fonctionnalités que les outils sélectionnés devront présenter.

### 11.1. Open Source

L'outil choisi devra être gratuit (freeware), à savoir qu'il devra être « distribué gratuitement sans pour autant conférer à l'utilisateur toutes les libertés d'usage ». (<http://www.dmoz.org/World/Fran%C3%A7ais/Informatique/Logiciels/Freeware/R%C3%A9pertoires> ).

### 11.2. Navigateur

De manière à se rapprocher au maximum de la réalité de l'utilisateur, l'outil choisi devra implémenter si possible un des quatre navigateurs principalement utilisés en Suisse :

- Internet Explorer

- Google Chrome
- Mozilla Firefox
- Safari

Dans tous les cas, le même navigateur devra être utilisé lors de la comparaison de deux tests sur la plateforme Moodle, pour éviter de fausser les mesures (les performances peuvent différer entre les navigateurs).

Du fait que les performances peuvent varier en fonction de la position géographique, il faudra que les tests soient réalisés depuis la position courante des étudiants (Suisse romande) et non pas depuis un emplacement étranger (en utilisant un outil externalisé par exemple).

### **11.3. Tests d'acceptation**

Les différentes requêtes devront être regroupées en scénarios, sous forme de processus utilisateurs automatisés. Les différentes actions réalisées habituellement sur un navigateur devront être effectuées automatiquement depuis un logiciel.

### **11.4. Test de performance**

Le logiciel devra retourner les temps d'exécution de chaque scénario utilisateur et si possible, des requêtes qui le compose.

Par rapport aux simples tests d'acceptation, ce sont ici les temps requis pour que les processus aboutissent qui sont évalués, et non le bon fonctionnement du site.

Les résultats découlant de l'analyse de performance devront permettre de déterminer les variations de performances entre différentes instances d'exécution des scénarios (sous forme de graphique par exemple).

### **11.5. Fiabilité des résultats**

L'outil devra restituer de manière fiable les résultats des analyses, à savoir que les mesures découlant du logiciel devront correspondre à une situation en production semblable. Par exemple, les performances d'un site analysé avec un navigateur donné et un chronomètre devront être comparables à celles retournées par l'outil choisi.

## 11.6. Affichage des résultats

De nombreuses variations peuvent être observées lorsque des tests sont exécutés du côté client, et cela pour les raisons suivantes (Web Load Testing For Dummies – Scott Barber) :

- La connexion du client entre Internet et les serveurs Moodle.
- La charge actuelle du serveur Moodle.
- Le navigateur/système utilisé par le client.

Pour les raisons identifiées ci-dessus, la tendance des résultats est plus intéressante qu'un résultat individuel. L'outil devra donc pouvoir analyser les résultats de manière compréhensible, en proposant par exemple une synthèse des résultats sous forme de graphique.

## 11.7. Transportabilité

Les tests développés devront être réutilisables sur une autre machine, et non pas assignés à une seule instance du logiciel de manière à garantir une certaine flexibilité et une possibilité de sauvegarde.

## 11.8. Suite de tests

Il devra être possible d'organiser les tests de manière hiérarchique, pour qu'une suite de régression entièrement automatisée et modulable puisse être créée.

De plus, les rapports générés devront permettre de visualiser les performances globales de la suite de tests, celles des scénarios utilisateur et les détails des sous-requêtes comme cela est présenté ci-dessous :

Temps global (45 secondes)

- Scénario 1 (15 secondes)
  - Requête 1.1 (5 secondes)
  - Requête 1.2 (5 secondes)
  - Requête 1.3 (5 secondes)
- Scénario 2 (30 secondes)
  - Requête 2.1 (10 secondes)

- Requête 2.2 (20 secondes)

### 11.9. Simplicité/Stabilité/rapidité

Les tests devront si possible être développés de manière simple et intuitive. L'interface du logiciel devra être aboutie et l'accès aux fonctions optimale. Nous pensons par exemple à la désactivation d'une partie de la suite de tests, qui devrait être une manipulation facile à réaliser. De plus, l'outil devra être le plus stable possible (pas de plantages, interface rapide, modules fonctionnels à 100%).

## 12. Analyse des outils existants

Nous allons maintenant analyser plusieurs outils en fonction des critères définis au chapitre précédent. Nous déterminerons si les outils sont capables d'automatiser un processus utilisateur, puis s'ils sont en mesure d'en extraire des informations de performance. Voici les différents points qui seront abordés dans l'analyse :

- **Functional testing** : Est-t-il possible de réaliser des tests fonctionnels avec cet outil ?
- **Performance testing** : L'outil est-t-il en mesure de générer des informations de performance à partir de tests créés?
- **Suite de test** : L'outil permet-t-il de regrouper et d'exécuter les tests sous la forme d'une suite de test ?
- **Rapports** : Est-t-il possible de générer des rapports en fonction des données de performance ?
- **Licence** : Quelle est la licence fournie avec cet outil ?
- **Dernière mise à jour** : De quand date la dernière mise à jour de l'outil ?
- **Site internet** : Quelle est le site internet de l'outil ?

Pour répondre à ces différentes questions, nous nous baserons sur la documentation officielle des outils. Nous validerons ces informations théoriques durant le Benchmark qui sera réalisé plus tard.

## 12.1. Selenium

Selenium est un logiciel créé à la base pour automatiser un navigateur Web. Il est en mesure d'automatiser des scénarios utilisateurs, mais pas d'en analyser les performances. Selenium peut être contrôlé via de nombreux langages de programmation, comme Java ou Ruby.



Figure 5 logo Selenium (seleniumhq.org)

Functional testing	Oui. Possibilité de créer et de tester des processus utilisateurs avec Firefox – Chrome – Internet Explorer.
Performance testing	Non. Selenium permet simplement de déterminer si les processus fonctionnent ou pas.
Suite de tests	Selenium est capable de créer des suites de test, contenant plusieurs scénarios. Cela est réalisable depuis le langage de programmation utilisé pour interagir avec Selenium.
Rapports	Les rapports Selenium sont orientés sur la réussite ou l'échec des tests (pas de graphiques de performance).
Licence	OpenSource
Dernière mise à jour	Janvier 2014
Site internet	<a href="http://docs.seleniumhq.org/">http://docs.seleniumhq.org/</a>

## 12.2. JMeter (Installation de base)

L'outil JMeter est capable d'analyser les performances de différents serveurs/protocoles comme http, ftp ou ldap. La reproduction de scénarios utilisateur est possible via un add-on (Web Driver Sampler).



Figure 6 logo JMeter (jmeter.apache.org)

Functional testing	Jmeter ne permet pas de reproduire les actions de l'utilisateur depuis l'installation de base, mais Il est possible de passer par un add-on (Web Driver Sampler).
Performance testing	Oui. JMeter est capable d'analyser les performances d'un site.
Suite de tests	Oui. JMeter est capable de créer une suite de tests à partir de composants JMeter (Sampler).
Rapports	Oui. JMeter permet de créer des rapports sous forme de tableaux/graphiques.
Licence	OpenSource
Dernière mise à jour	2013
Site internet	<a href="http://jmeter.apache.org/">http://jmeter.apache.org/</a>

### 12.3. Funkload

Funkload est un outil développé pour la création de tests d'acceptation couplés à des tests de performance. La description de l'outil ne fournit pas d'informations concernant la création d'une suite de tests.



Figure 7 Logo FunkLoad  
([funkload.nuxeo.org](http://funkload.nuxeo.org))

Functional testing	Oui. Il est possible d'enregistrer et de reproduire les actions du navigateur automatiquement (utilise un navigateur propre à cet outil).
Performance testing	Oui. FunkLoad analyse les performances des scénarios réalisés avec ce même logiciel.
Suite de tests	Pas d'information à ce sujet sur le site officiel.
Rapports	Oui. Les performances sont exportées sous forme d'un rapport HTML.
Licence	OpenSource
Dernière mise à jour	2012
Site internet	<a href="http://funkload.nuxeo.org/">http://funkload.nuxeo.org/</a>

### 12.4. TestMaker

TestMaker est un outil basé sur le logiciel Open Source Selenium. Il permet d'y ajouter une fonctionnalité notable, qui permet d'analyser les performances des tests créés.



Figure 8 Logo TestMaker  
([pushtotest.com](http://pushtotest.com))

Functional testing	Oui. TestMaker permet de créer des scénarios utilisateurs en se basant sur Selenium.
Performance testing	Présenté comme une surcouche Selenium, TestMaker permet d'analyser les performances des tests.



Suite de tests	D'après la documentation officielle, TestMaker permet de créer des suites de test à partir des scénarios créés.
Rapports	Oui. TestMaker crée et exporte des rapports de performances.
Licence	Gratuit (version payante Appvance).
Dernière mise à jour	2011
Site internet	<a href="http://www.pushtotest.com/products.html">http://www.pushtotest.com/products.html</a>

## 12.5. Gatling

D'après le site de Gatling, cet outil se voudrait particulièrement performant (plus efficace que JMeter). Spécialisé dans les tests de performance, ce logiciel est aussi capable d'exécuter des scripts de test (scénarios).



Figure 9 Logo Gatling (gatling-tool.org)

Functional testing	Gatling est capable de créer des scripts de test, et donc d'automatiser les processus utilisateurs (utilise pour cela un navigateur propre à cet outil).
Performance testing	Oui. Gatling est capable d'analyser les performances d'un site avec un nombre d'utilisateurs virtuel défini.
Suite de tests	Gatling nous permet de lier les scénarios entre eux sous forme de suite de tests, et de générer des informations de performances par rapport aux scénarios/à la suite de tests  <a href="http://www.osaxis.fr/blog/gatling-nouvelle-solution-libre-pour-les-tests-de-montee-en-charge/">http://www.osaxis.fr/blog/gatling-nouvelle-solution-libre-pour-les-tests-de-montee-en-charge/</a>

Rapports	Les résultats des performances sont exportés sous forme de rapport HTML avec graphiques.
Licence	OpenSource
Dernière mise à jour	Octobre 2013
Site internet	<a href="http://gatling-tool.org/">http://gatling-tool.org/</a>

## 12.6. Fitnessse

Développé sous la forme d'un serveur Wiki, Fitnessse permet d'exécuter des scénarios utilisateur depuis une plateforme Web. Ce système ne permet pas la réalisation de tests de performance et l'exportation de rapports.



Figure 10 Logo Fitnessse (fitnessse.org)

Functional testing	Oui. Fitnessse automatise les tests d'acceptation et les centralise sur un serveur web.
Performance testing	Non. Fitnessse n'est pas utilisé pour les tests de performance.
Rapports	Les rapports Fitnessse sont relatifs aux tests fonctionnels et pas aux performances.

Licence	OpenSource
Dernière mise à jour	2013
Site internet	<a href="http://www.fitnessse.org/">http://www.fitnessse.org/</a>

## 13. Sélection d'outils

Maintenant que les capacités théoriques des différents outils ont été appréciées, nous allons sélectionner ceux qui seront testés lors du Benchmark.

### 13.1. Selenium+ JMeter

#### 13.1.1. Description

Cette première solution est composée de deux outils complémentaires :

- Selenium, qui est avant tout un logiciel conçu pour automatiser un navigateur. En utilisant un script créé depuis un des langages de programmation supportés, il permet d'automatiser un test d'acceptation utilisateur (UAT).
- Jmeter est orienté vers les tests de performance. L'idée serait donc d'exporter les tests fonctionnels créés sur Selenium vers Jmeter, pour pouvoir obtenir les performances des tests.

Cette approche permet de palier au principal défaut de Selenium, à savoir qu'il n'est pas conçu pour réaliser des tests de performances.

#### 13.1.2. Motivation du choix

Avec ces deux outils, nous disposons donc en théorie d'une solution contenant les fonctionnalités requises pour mener à bien ce travail.

- Tests de performances couplés à des scénarios utilisateur.
- Création d'une suite de tests.
- Simulation des principaux navigateurs (Firefox, Internet Explorer, Chrome).
- Création de rapports.

## **13.2. TestMaker**

### **13.2.1. Description**

TestMaker est une solution gratuite développée par la société PushToTest. Remplacée aujourd'hui par une version payante (AppVance), la version Open Source est toujours disponible. Son but premier était d'ajouter les fonctions de tests de performance et de tests modulaires (chaque test est composé de blocs réutilisable), à l'outil Selenium.

### **13.2.2. Motivation du choix**

Nous avons opté pour l'outil TestMaker car celui-ci présente toutes les fonctionnalités requises pour mener à bien ce travail :

- Tests de performance couplés à des scénarios utilisateur
- Création d'une suite de tests
- Simulation des principaux navigateurs (Firefox, IE, Chrome)
- Création de rapports.

## **13.3. FunkLoad**

### **13.3.1. Description**

FunkLoad est un outil spécialisé dans le Web Testing, analysant les performances de requêtes GET/POST exécutées sur un serveur web. L'outil est capable d'enregistrer les actions d'un navigateur Web avant de les exporter sous forme de script exécutable avec FunkLoad.

### **13.3.2. Motivation du choix**

D'après la description du logiciel, cet outil serait capable d'analyser les performances du site Moodle. Du fait que cet outil n'est pas basé sur l'automatisation d'un navigateur, il n'est pas capable de simuler un navigateur connu.

- Tests de performances couplés à des scénarios utilisateur automatisés.
- Création d'une suite de tests.

- Rapport exhaustifs et exportés sous le format HTML. .

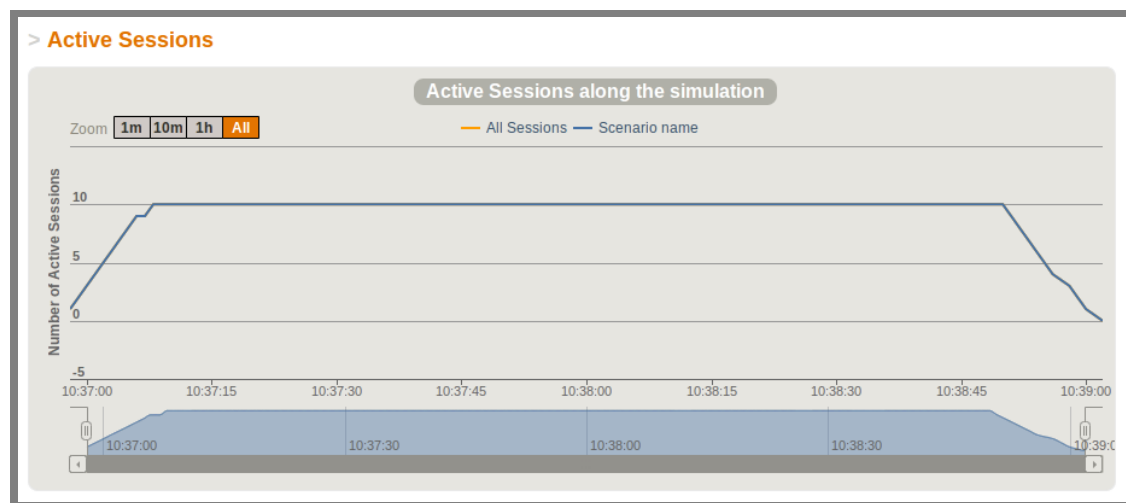


Figure 11 Performances Funkload (screenshot)

## 13.4. Gatling

### 13.4.1. Description

Gatling est un outil de test similaire au logiciel Funkload, qui envoie directement des requêtes GET/POST sans automatiser un navigateur. La différence entre ces deux logiciels se situera au niveau de la suite de tests (qui n'est pas présentée sur le site de Funkload), de la stabilité et de la qualité des rapports.

### 13.4.2. Motivation du choix

Les capacités de cet outil sont théoriquement les mêmes que celles de l'outil Funkload. Nous verrons lors du Benchmark si l'un ou l'autre prédomine.

## 13.5. JMeter + Web Driver Sampler

### 13.5.1. Description

Cette combinaison se compose de l'outil JMeter et du plugin Web Driver Sampler. Celui-ci nous permet d'enregistrer des tests Selenium directement dans l'outil JMeter, sans passer par un langage externe. Nous verrons si cette solution est plus intéressante que la liaison Selenium (IDE) + JMeter.

### 13.5.2. Motivation du choix

Les motivations qui nous poussent à tester cette solution sont relatives à la combinaison Selenium(IDE)+JMeter. En effet, les capacités des deux solutions sont théoriquement identiques du fait qu'elles sont toutes les deux basées sur l'outil Selenium. Nous relèverons durant le Benchmark les différences entre les deux solutions.

## 14. Inventaire des tests du Benchmark

Nous allons maintenant définir les tests qui seront implémentés lors du Benchmark des outils. Le Benchmark nous permettra de déterminer si un outil est à même d'analyser les performances du site Moodle.

Les tests implémentés pour le Benchmark des outils devront être réalisés directement sur la plateforme Moodle, de manière à ce que les résultats obtenus puissent être comparés à la suite de tests complète qui sera implémenté à la fin de ce travail.

Nous commencerons par analyser les parties testables du site Moodle, puis nous définirons les différents scénarios qui seront implémentés sur les outils choisis au chapitre précédent.

Le Benchmark sera réalisé sur le serveur de développement de la plateforme Moodle (<http://cyberlearn-dev.hes-so.ch/>).

Nous disposons d'un compte (**tb\_fx\_user**) ayant un accès de type professeur sur un cours appelé **tb\_fx**. C'est à partir de ce compte que nous allons réaliser les différents processus nécessitant une connexion préalable.

## 14.1. Analyse des parties testables

Nous allons maintenant établir une liste des principaux processus utilisateur de la plateforme Moodle. Nous choisirons plus tard quels éléments seront implémentés lors du Benchmark ou lors de la création de la suite de tests complète (sf. glossaire). Cette liste est basée sur des tests manuels réalisés précédemment par l'administration du système Moodle.

- Connexion sur le site Moodle
  - Depuis le module AAI.
  - Depuis le module de connexion Moodle par défaut.
- Accéder au cours Tb\_Fx
  - Depuis un utilisateur connecté.
  - Depuis un visiteur non-connecté (anonyme).
- Ajouter une ressource
  - Dossier
  - Fichier
  - Devoir
  - Etiquette
  - Url
- Accéder à une ressource
  - Déterminer le temps nécessaire pour accéder aux ressources créées.
- Modifier une ressource
  - Déterminer le temps nécessaire pour modifier une ressource.
- Supprimer une ressource
  - Temps requis pour supprimer une ressource.

## 14.2. Suite de test du Benchmark

Nous allons maintenant sélectionner et détailler les processus utilisateur qui seront adoptés pour le Benchmark.

### 14.2.1. Définition

La suite de tests définie pour le Benchmark se compose d'une première connexion sur le site Moodle Dev, suivie de l'ouverture du cours **tb\_fx** et de l'ajout d'une ressource de type **répertoire** dans ce dernier.

Cette sélection nous permettra de tester les parties les plus importantes de la plateforme Moodle, et donc de nous faire une première idée des capacités des outils.

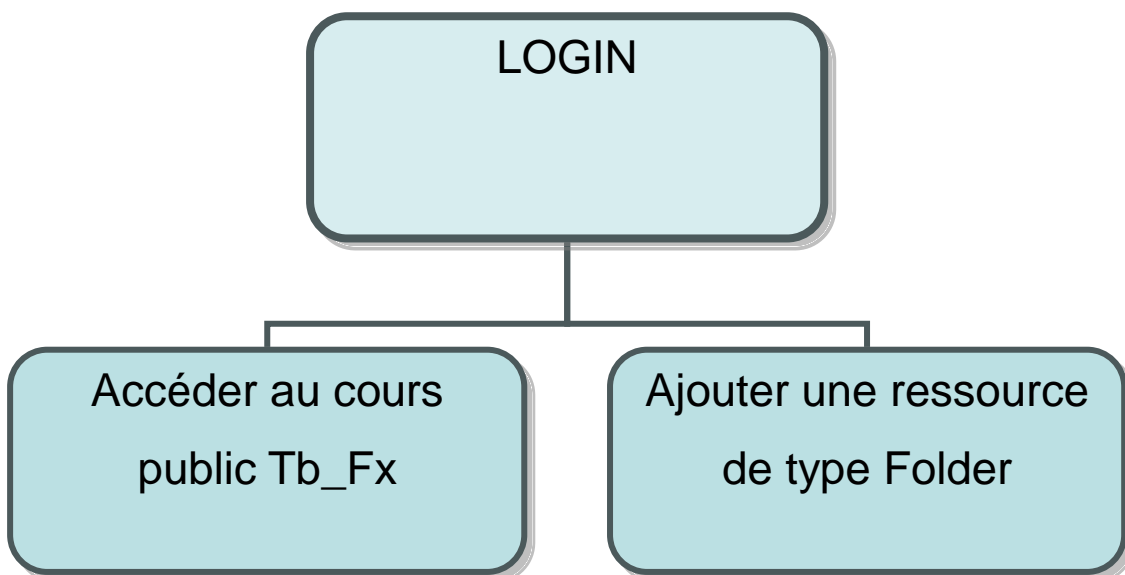


Figure 12 Tests du Benchmark (screenshot Visio)

### 14.2.3. Login

Nous allons maintenant détailler les requêtes relatives aux différents processus.

1. Navigation sur le site Moodle Dev.
  - a. <http://cyberlearn-dev.hes-so.ch/>
1. Accéder à la page de connexion.
  - a. <http://cyberlearn-dev.hes-so.ch/login/index.php>
2. Entrer les informations de connexion.



- a. Utilisateur : **tb\_fx\_user**
  - b. Mot de passe : **password**
3. Redirection automatique vers la page Moodle de l'utilisateur.
    - a. <http://cyberlearn-dev.hes-so.ch/my/>

#### 14.2.4. Accéder au cours Tb\_Fx

1. Accès au cours Tb\_Fx via le menu de gauche de l'utilisateur connecté.
  - b. <http://cyberlearn-dev.hes-so.ch/course/view.php?id=4861>

#### 14.2.5. Ajout d'un dossier dans la première section

1. Accès au cours Tb\_Fx via le menu de gauche de l'utilisateur connecté.
  - a. <http://cyberlearn-dev.hes-so.ch/course/view.php?id=4861>
2. Activation du mode modification.
  - a. <http://cyberlearn-dev.hes-so.ch/course/view.php?id=4861&notifyeditingon=1>
3. Clic sur **ajouter une activité ou une ressource**, et sélection de la ressource **fichier**.
  - a. <http://cyberlearn-dev.hes-so.ch/course/modedit.php?add=folder&type=&course=4861&section=1&return=0&sr=0>
4. Validation du formulaire (requête POST).

### 14.3. Choix d'un mode de connexion

Comme cela est décrit plus haut (chapitre 14.1), nous disposons de deux moyens différents pour nous connecter sur la plateforme Moodle. Nous allons maintenant choisir quel moyen sera utilisé pour la connexion, en sachant que deux des outils sélectionnés (Gatling et Funkload) ne sont pas capables de gérer le clic sur le bouton de connexion, mais uniquement d'envoyer des requêtes http (GET/POST).

#### 14.3.1. Contexte

Les cours présents sur la plateforme Moodle peuvent être accessibles par les visiteurs, ou bien par les utilisateurs connectés.

Par contre, la modification du contenu du cours n'est pas réalisable sans une connexion préalable sur le site Moodle, comme cela est décrit sur le site officiel ([http://docs.moodle.org/19/fr/Visiteur\\_anonyme](http://docs.moodle.org/19/fr/Visiteur_anonyme)).

L'ajout de ressource passe donc forcément par une connexion sur le site, réalisable de deux manières différentes :

- Via le module AAI

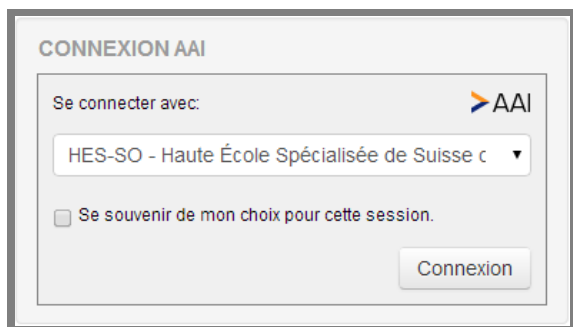


Figure 13 Connexion AAI (screenshot Cyberlearn)

- Via le module de connexion par défaut de Moodle.

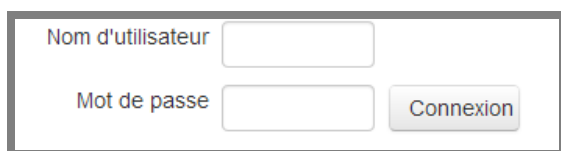


Figure 14 Connexion Moodle (screenshot Cyberlean)

### 14.3.2. Problèmes liés au module AAI

La connexion à la plateforme Moodle via le module AAI introduit la présence de cookies, créés lors du clic sur le bouton **Connexion** (méthode **onClick()**). Le fait de cliquer sur ce bouton appelle une fonction JavaScript (submitForm). Cette fonction va ensuite générer plusieurs cookies, permettant à la fois la connexion au site Moodle et le maintien de ladite connexion (durant le temps de la session utilisateur).

Cette situation explique pourquoi une simple requête GET vers la page de connexion AAI ne fonctionne pas, les cookies n'ayant pas été créés par la classe JavaScript **submitForm()**. Le seul moyen d'utiliser la méthode étant de cliquer physiquement sur le bouton **Connexion**.

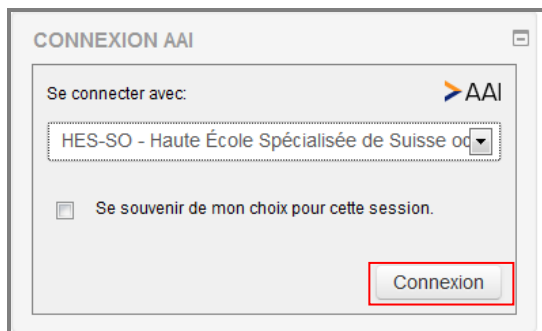


Figure 15 Connexion AAI (screenshot Cyberlearn)

#### Analyse des cookies

Parmi les cookies créés par la méthode SubmitForm, seul le cookie **MoodleSession** est utilisé pour conserver son accès sur Moodle d'une fois que la connexion est établie. Si celui-ci est supprimé, la connexion au site Moodle expirera aussitôt. En modifiant la date d'expiration du cookie, la connexion restera effective pendant environ 24 heures, mais il finira par expirer même si la date du cookie spécifie une durée supérieure .

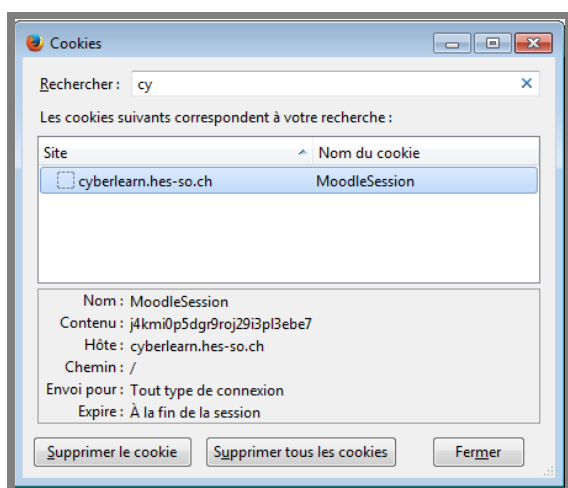


Figure 16 Cookies connexion Moodle (screenshot // CookieManager+)

Le fait de modifier le cookie MoodleSession avec l'extension Firefox gratuite **CookieManager+** et une nouvelle date d'expiration (22 mai 2025), permet de s'authentifier sur la plateforme Moodle sans passer par la méthode Javascript **SubmitForm** pour une durée limitée à 24 heures.

```
#Created by Cookies Manager+ v1.5.2 on Wed May 21 2014 15:55:34 GMT+0200
```

```
cyberlearn.hes-so.ch FALSE / FALSE 1747921309 MoodleSession  
j4kmi0p5dgr9roj29i3pl3ebe7
```

### 14.3.3. Choix d'un module de connexion

Du fait de la problématique liée aux cookies, qui finissent par expirer après un certain laps de temps, les tests seront effectués sur des cours ouverts (pas de modifications possible), ou alors en utilisant le **module de connexion Moodle par défaut** qui ne contient pas de fonction javascript derrière le bouton de connexion : <http://cyberlearn-dev.hes-so.ch/login/index.php>

## 15. Définition du Benchmark

Dans ce chapitre, nous définirons la procédure qui sera appliquée pour le Benchmark des différents outils. Nous détaillerons toutes les étapes nécessaires pour tester et analyser les performances des logiciels.

### 15.1. Définition de l'outil

Nous commencerons par détailler la structure du logiciel choisi selon les critères suivants :

- Les différents modules de l'outil.
- Le format des résultats.
- Le langage ou l'interface utilisée pour la création des scénarios.

### 15.2. Installation de l'outil

Dans un deuxième temps, nous présenterons l'installation du logiciel et de ses éventuels modules. Précisons que tous les logiciels seront installés dans un environnement Windows, pour des questions de simplicité.

### 15.3. Enregistrement des tests (scénarios)

Durant cette étape, nous implémenterons les différents scénarios du Benchmark dans l'outil en cours de test. Nous détaillerons la méthode utilisée par le logiciel pour reproduire les actions de l'utilisateur :

- Automatisation d'un navigateur.
- Requêtes get/post depuis le logiciel.

### 15.4. Exécution des tests

Nous détaillerons ensuite la procédure visant à exécuter les tests créés au point précédent, et à générer des rapports de performances.

### 15.5. Création de la suite de tests

Une fois que les scénarios auront été créés, ils devront être intégrés dans une suite de tests, de manière à ce qu'ils se lancent automatiquement à partir d'un point de départ. Pour le Benchmark, la suite de tests permettra donc de tester les performances des scénarios définis au chapitre précédent de manière modulable et automatique.

## 15.6. Génération du rapport

L'exécution de la suite de tests devra aboutir sur un rapport généré par le logiciel, qui devrait nous renseigner sur les éléments suivants :

- Les performances globales de la suite de tests.
  - Temps d'exécution global
  - Utilisateurs virtuels (threads)
- Les performances de chaque scénario.
  - Temps d'exécution de chaque scénario
- Les détails de tous les temps d'exécution des sous-requêtes.

Les différentes analyses de performance devront être présentées sous forme de graphique, ou du moins avec une information exprimant la valeur moyenne, pour obtenir des résultats crédibles.

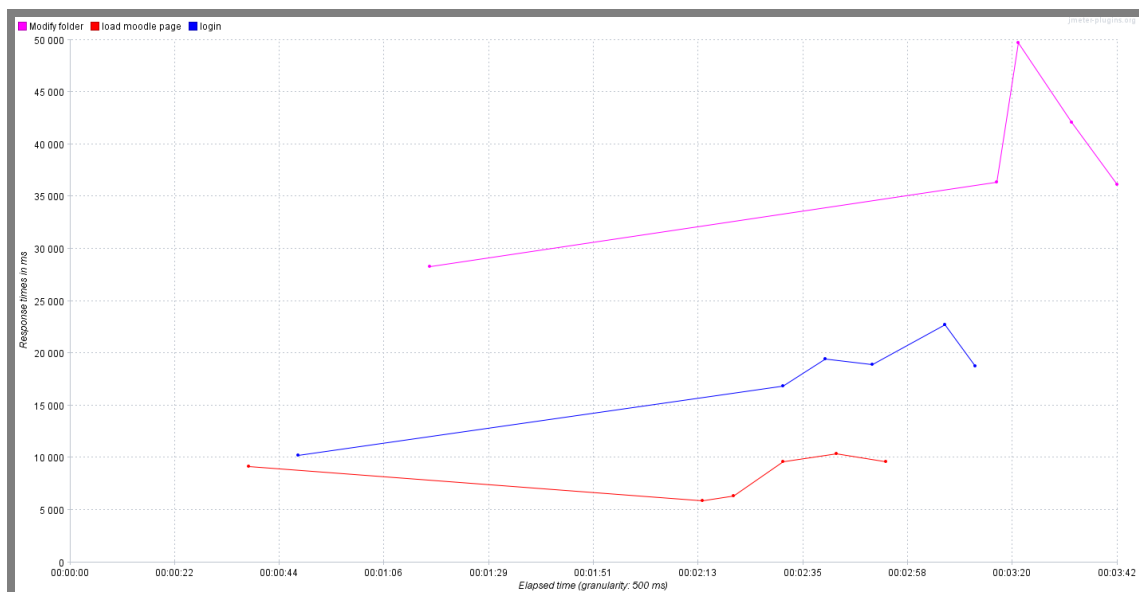


Figure 17 graphique performances Moodle (screenshot // JMeter)

## 15.7. Evaluation de l'outil

Les étapes précédentes nous permettent de nous familiariser avec l'outil. Nous devons maintenant évaluer les points testés de manière à déterminer les capacités réelles du logiciel. Pour se faire, nous nous baserons sur éléments suivants :

- Enregistrement d'un test
  - L'enregistrement des tests est-t-il fastidieux ?
  - Existe-t-il un système de création automatique des tests (enregistrement des actions du navigateur) ?
- Exécution d'un test
  - Problèmes rencontrés lors de l'exécution des tests créés ?
- Création de la suite de tests
  - A-t-il été possible de créer une suite de tests ?
  - La suite de tests permet-t-elle d'exécuter automatiquement tous les scénarios ?
- Résultats du Benchmark
  - Les tests du Benchmark ont-t-ils été implémentés/sont-t-ils fonctionnels ?
- Rapports
  - Les rapports présentent-t-ils les performances globales, les performances des scénarios et des sous-requêtes ?
- Rapidité/Stabilité
  - Le logiciel plante-t-il souvent ?
  - Le logiciel est-t-il particulièrement lent pour l'utilisateur ?

## 16. Benchmark PushToTest-Testmaker

### 16.1. Modules

Développé par la société PushToTest, Testmaker est un logiciel gratuit qui se présente comme une surcouche de l'outil Selenium. Il se compose de trois unités distinctes :

- La **console TestMaker** utilisée pour créer une première ébauche des tests via un navigateur, de les séparer en modules, et de les assigner à des utilisateurs virtuels.
- Le **TestNode** permet d'exécuter des tests créés précédemment, en passant par un navigateur.
- L'application testée (**Application Under Test**) correspond, dans le cadre de ce travail, au site Moodle qui sera testé avec cet outil.

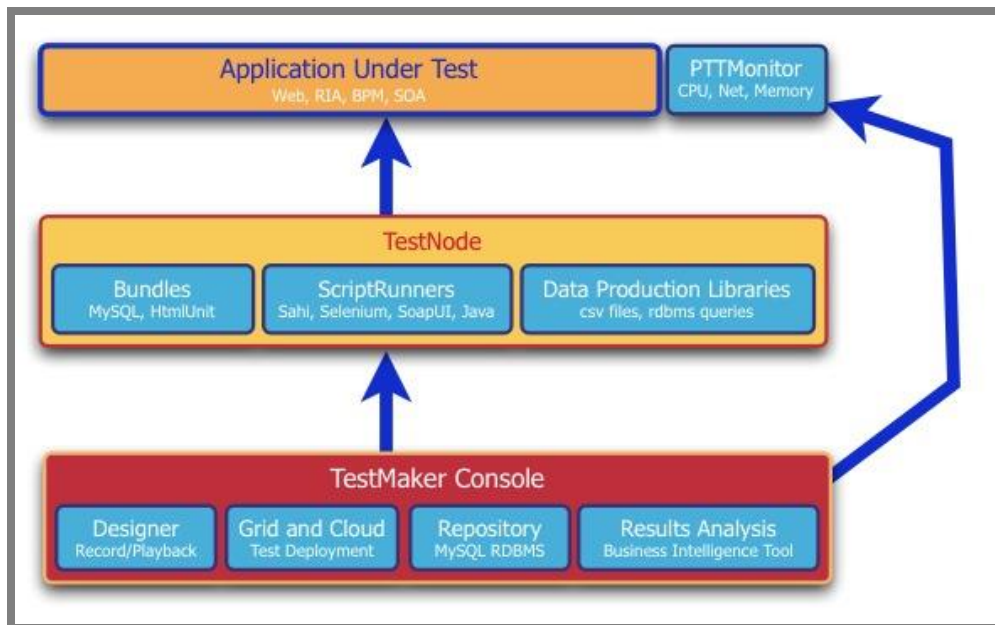


Figure 18 TestMaker structure (screenshot - pushtotest.com/user guide)

## 16.2. Prérequis

- L'installation de l'outil testMaker nécessite une version Java (1.6 minimum), comme cela est décrit sur le site officiel : <http://www.pushtotest.com/installation-configuration-deployment-testmaker-6>. Java peut être téléchargé depuis ici : <https://www.java.com/fr/download/>

## 16.3. Installation

Télécharger l'outil depuis le site de la communauté Test Maker.

- <http://www.pushtotest.com/testmaker-6-community-download>

Extraire l'archive téléchargée.



Ouvrir les propriétés du fichier **setup\_testmaker32.cmd** et configurer le mode de compatibilité pour Windows XP SP3.

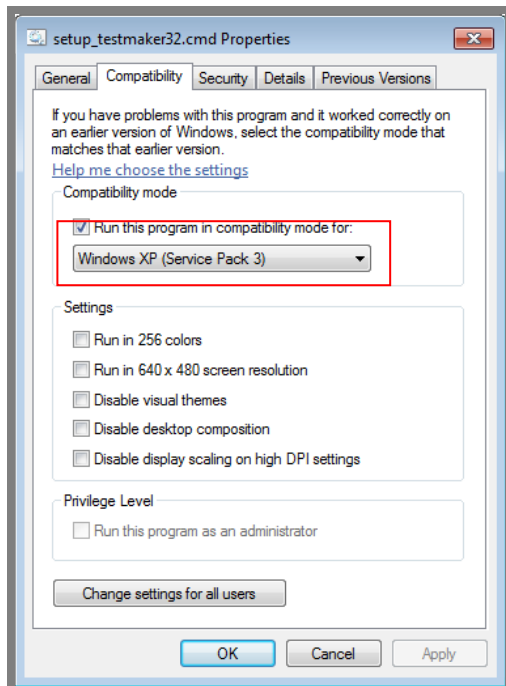


Figure 19 Propriétés testmaker32.cmd (screenshot // XP sp3)

Lancer maintenant le fichier **setup\_testmaker32.cmd**

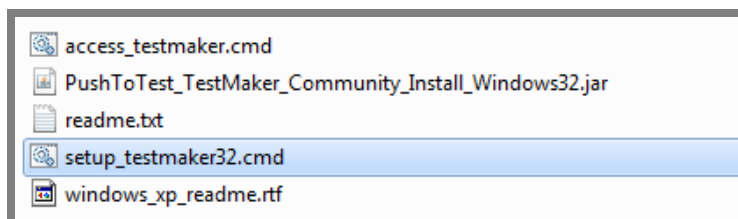


Figure 20 contenu TestMaker (screenshot // XP sp3)

L'installation va alors démarrer (accepter le contrat de licence, sélectionner le répertoire de destination). Au besoin, il peut être nécessaire de désactiver le UAC (User Account Control) sur Windows 7. Une description de cette procédure est disponible ici : <http://www.mydigitallife.info/how-to-disable-and-turn-off-uac-in-windows-7/>

## 16.4. Création des tests

Commencer par lancer TestMaker via l'exécutable **TestMaker.exe** situé dans le répertoire de TestMaker.

Lancer ensuite l'utilitaire d'enregistrement de tests (onglet Tools, Designer).

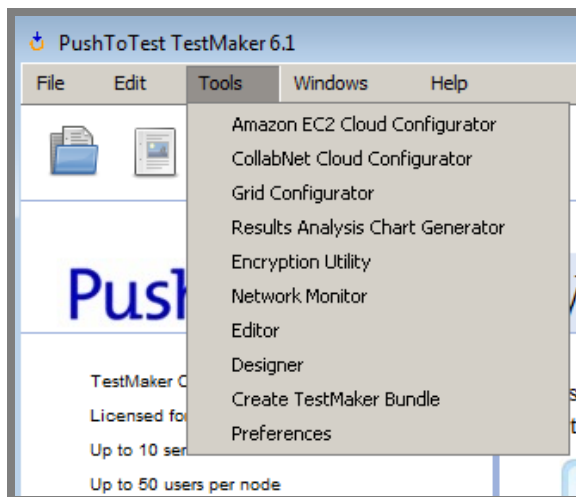


Figure 21 Tools menu (screenshot - TestMaker)

Démarrer l'enregistrement via le bouton **record**.

Sélectionner le **navigateur** à utiliser pour l'enregistrement, et l'**URL** de départ.

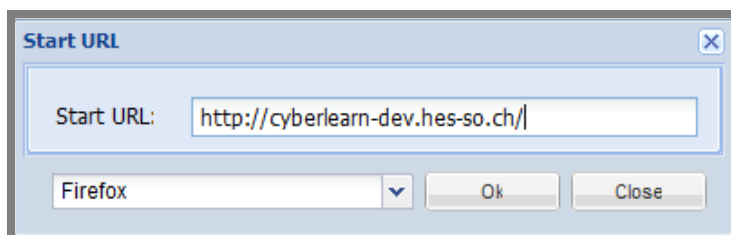


Figure 22 start url Designer (screenshot // TestMaker)

Réaliser les actions souhaitées sur le navigateur, et cliquer sur le bouton **end recording** pour arrêter l'enregistrement.

Nous allons maintenant enregistrer le test réalisé en **sahi** (file/Export Test Object as/Sahi) de manière à pouvoir le réutiliser lors de la création de la suite de tests depuis TestMaker.

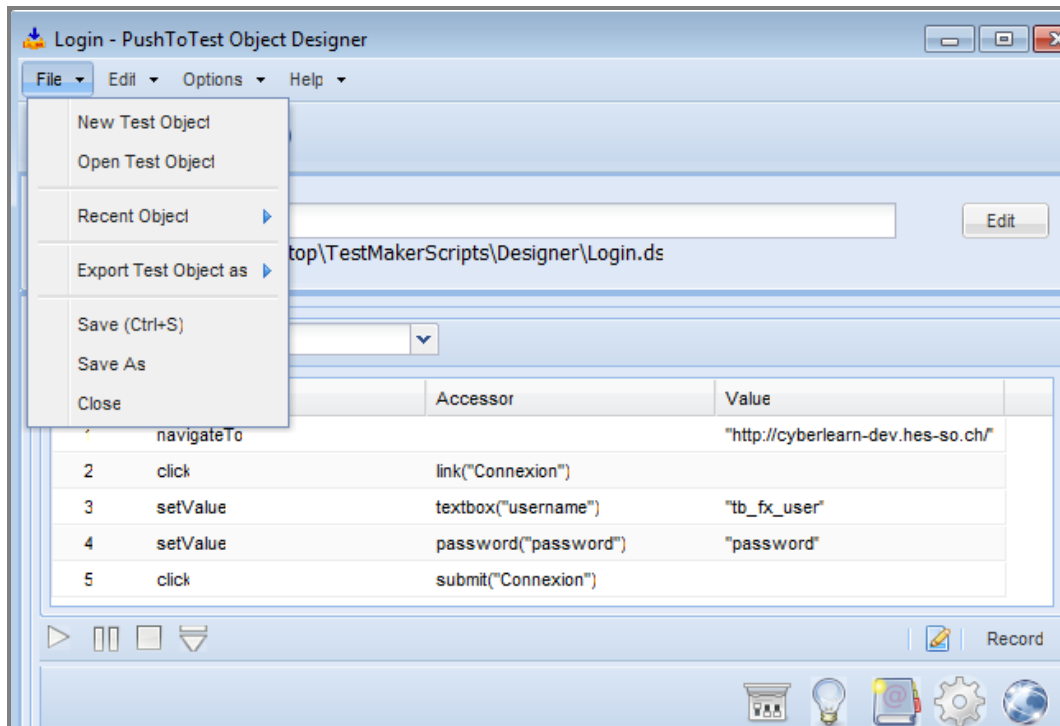


Figure 23 export test (screenshot - Designer)

Retourner maintenant sur la fenêtre principale de TestMaker, et cliquer sur **File/New Load Test** pour créer un nouveau test de charge.

- Attendre quelques secondes car la fenêtre est lente à charger.

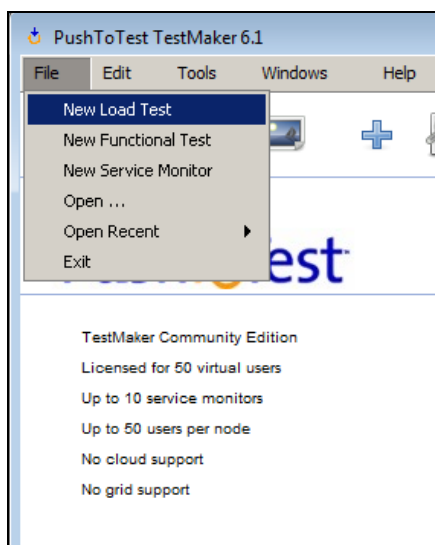


Figure 24 Create test (screenshot - TestMaker)

La fenêtre qui s'ouvre alors est séparée en deux parties :

- La zone de gauche contient tous les tests qui seront présents dans la suite de tests.
- La zone de droite affiche les détails du test sélectionné.

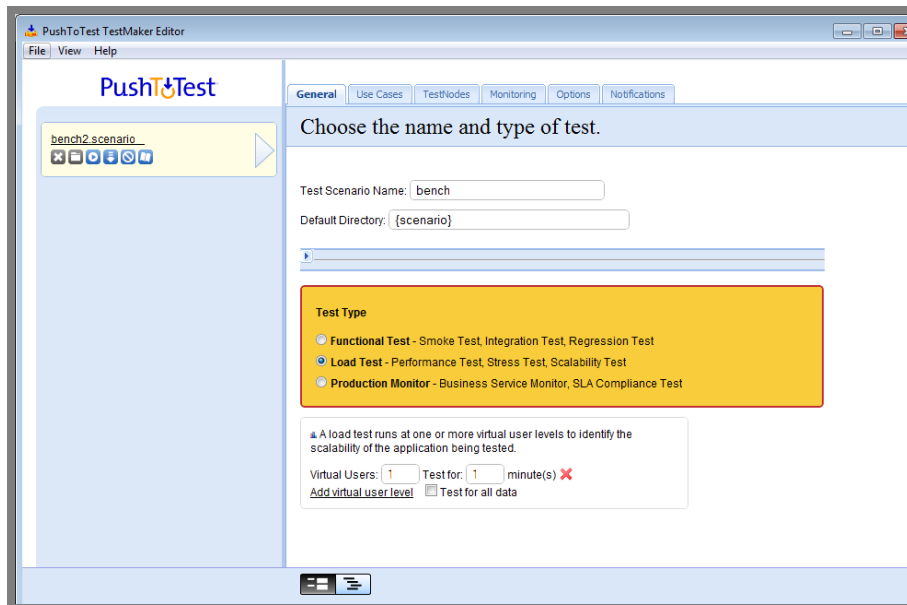


Figure 25 Create load test TestMaker (screenshot)

Nous allons ensuite assigner le test que nous avons précédemment créé avec le Designer au premier scénario de notre suite de tests :

- Onglet Général :
  - Modifier le nom du scénario (Test Scenario Name)
- Onglet Use Cases (Intégrer notre test dans le scénario) :
  - Test Type : **Sahi**
  - Sélectionner le **Script Sahi** enregistré depuis le designer
  - Sélectionner le navigateur qui sera utilisé pour le test

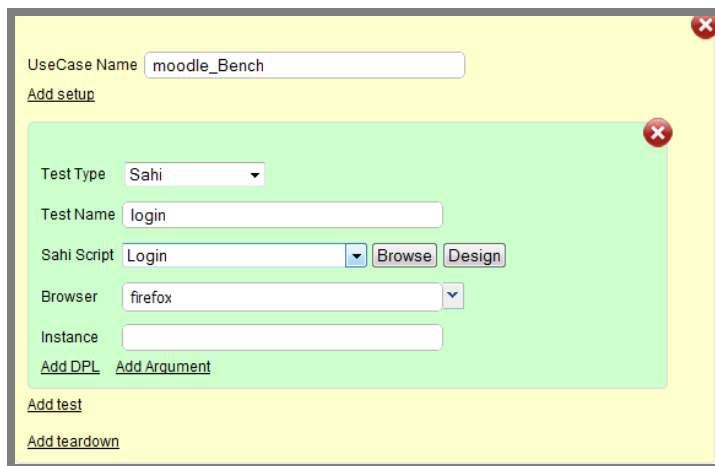


Figure 26 Creation d'un scénario (screenshot,TestStudio)

## 16.5. Exécuter un test

Depuis le volet de gauche de la fenêtre TestMaker, cliquer sur le bouton **Play** situé sous le scénario à exécuter. Pour exécuter la suite de tests complète, cliquer sur **File**, puis sur **Execute scenario**.



Figure 27 scénrios TestMaker (screenshot)

Le site officiel de Testmaker parle de l'ouverture d'une fenêtre présentant l'avancée et les résultats du test. Pourtant, la fenêtre ne s'ouvre pas et le scénario ne s'exécute pas. Aucun enregistrement n'a été constaté dans la console du logiciel (log). Nous pouvons en déduire que l'outil testMaker n'est pas abouti, et ne permet pas de répondre aux besoins de ce travail.

## 16.6. Analyse de l'outil TestMaker

Le projet n'a pas été mis à jour depuis 2011, il a été remplacé par une version payante (appvance).

### 16.6.1. Enregistrement des tests

Le système utilisé par TestMaker pour enregistrer les actions réalisées sur le navigateur (Designer) est fonctionnel. Il a pu être utilisé pour créer et reproduire les scénarios du Benchmark. Par contre, le designer ne fonctionne pas avec les nouveaux navigateurs, Il n'a marché qu'avec la version 8 d'Internet Explorer.

### 16.6.2. Exécution des tests

L'exécution des testés créés n'a pas été possible avec l'outil Testmaker. La fenêtre relative à l'avancement et aux résultats des tests ne se lance pas.

Nous pouvons en déduire que l'outil n'est pas fonctionnel, et qu'il n'est pas capable d'exécuter des tests de performances.

### 16.6.3. Suite de tests

Les fonctionnalités théoriques indiquent que l'outil est capable de réaliser une suite de tests. Mais vu que l'exécution de tests est impossible, nous n'avons pas été en mesure de créer la suite de tests projetée.

### 16.6.4. Résultats obtenus lors du Benchmark

Login	NOK
Accès au cours TB_FX	NOK
Ajout d'un répertoire	NOK

Le Benchmark n'a pas pu être réalisé avec cet outil, du fait que l'exécution des tests n'a pas été possible.

### 16.6.5. Qualité des rapports

Il n'a pas été possible de générer des rapports, car la fenêtre associée ne s'est jamais ouverte.

### 16.6.6. Rapidité/Stabilité

- L'outil prend beaucoup de place sur le disque (850mb), par rapport aux autres logiciels de test.

- L'installation est lente (presque 10 minutes).
- L'outil est très lent.
  - Il faut compter dix secondes lors du chargement des outils (Designer, gestion des scénarios, ...).
- L'outil n'est pas stable.
  - De nombreux plantages ont été constatés, surtout avec l'éditeur de scénarios (message d'erreur suivi de la fermeture du module).

## 17. Selenium 2.0

Nous allons maintenant passer au Benchmark du logiciel Selenium 2.0. Précisons que le développement sera présenté au chapitre suivant (Développement Selenium). Du fait que cet outil ne permet pas directement d'obtenir des informations de performance, il sera lié avec le logiciel JMeter au chapitre **Liaison Selenium-JMeter**.

### 17.1. Modules Selenium

D'après <http://docs.seleniumhq.org/>, Selenium est un logiciel conçu à la base pour automatiser un navigateur web. Néanmoins, il peut être aussi utilisé pour créer des suites de régressions basées sur cette capacité d'automatisation.

Selenium est composé de plusieurs modules :

**Selenium Server** : Le serveur Selenium est utilisé du moment où les tests sont déployés sur plusieurs machines en parallèle (Grid configuration). Cela permet par exemple de tester plusieurs navigateurs en même temps (Chrome et Firefox par exemple).

**Selenium IDE** : disponible sur Firefox, il permet d'enregistrer des actions effectuées sur le navigateur, et de pouvoir les relancer de manière automatique par la suite. Simple d'utilisation, il permet de visualiser, de modifier ou de supprimer les requêtes du test en cours de création.

**Selenium WebDriver** : Introduit avec Selenium 2.0, le WebDriver a été créé pour simplifier la programmation et interférer directement avec le navigateur. En effet, la version précédente injectait du JavaScript dans le navigateur ce qui n'était pas optimal avec les pages web dynamiques. C'est ce module qui sera utilisé lors du développement Selenium avec Eclipse présenté au chapitre suivant.

*Selenium 1.0 + WebDriver = Selenium 2.0*

Précisons que si Selenium peut être utilisé pour créer une suite de régression, il n'est pas utilisable pour effectuer des tests de charge (Load Testing). Il est nécessaire d'installer un logiciel tiers pour que cela soit possible.

La société PushToTest a pourtant créé un logiciel gratuit se présentant comme une surcouche de Selenium, capable de réaliser des tests de charge et de synchroniser différents tests (voir chapitre « Benchmark Test Maker »).

L'outil JMeter peut être configuré pour analyser les scénarios créés depuis Selenium.

## 17.2. Selenium IDE

### 17.2.1. Domaine d'utilisation

Selenium IDE est utilisé pour créer des tests fonctionnels, de manière simple et intuitive (enregistrement des actions du navigateur). Faisant partie de la suite Selenium, il n'est pas utilisable pour réaliser des tests de performance.

La fonction d'exportation vers un code JUnit permet de sauvegarder les tests en Java, de manière à les ouvrir depuis un environnement de développement. Cette fonctionnalité n'est pas tout à fait opérationnelle, et l'exécution des tests démontre que le code généré n'est parfois pas optimal (voir chapitre 17.2.5).

### 17.2.2. Installation

Télécharger Selenium IDE pour firefox [http://docs.seleniumhq.org/download/#side\\_plugins](http://docs.seleniumhq.org/download/#side_plugins)

Faire glisser le fichier téléchargé (.xpi) dans une fenêtre Firefox. Valider l'installation du plugin.

Au redémarrage de Firefox, l'icône de Selenium IDE est visible en haut à droite de la fenêtre.

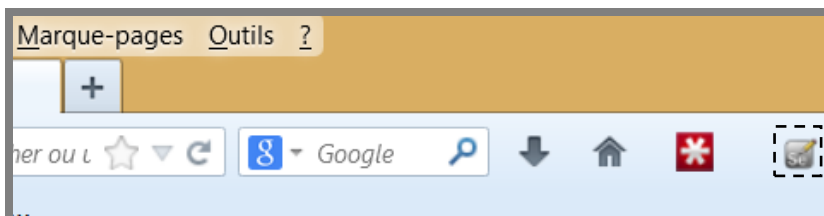


Figure 28 accès Selenium IDE (screenshot - Firefox)



### 17.2.3. Enregistrement d'un test

Commencer par lancer Selenium IDE en cliquant sur l'icône en haut à droite de la fenêtre Firefox. La fenêtre suivante devrait s'ouvrir :

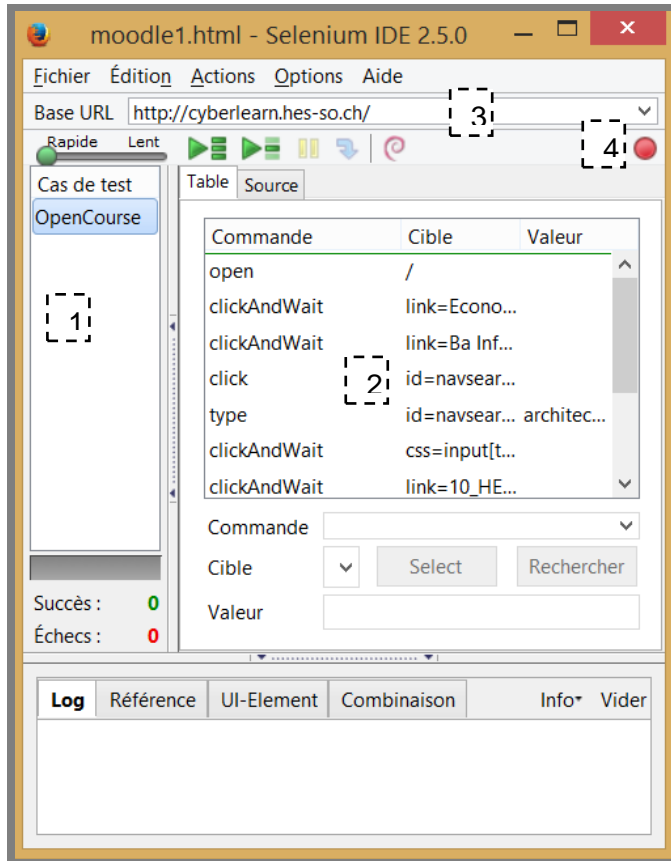


Figure 29 Interface Selenium IDE (screenshot)

1. Affichage de la suite de tests, et des différents scénarios qui la compose.
  - a. Bouton droit sur cette zone pour créer un nouveau test, ou renommer un test existant.
2. Cette zone affiche toutes les actions contenues dans le test sélectionné à gauche
  - a. Les actions peuvent être supprimées avec la touche Delete, et la sélection d'une action affiche les détails de celle-ci en dessous (possibilité de modifier la requête)
3. La barre supérieure affiche l'URL de départ qui sera chargée avant que le test ne s'exécute.
4. Le bouton rouge d'enregistrement permet de démarrer la capture des actions réalisées dans la fenêtre Firefox à partir de laquelle Selenium IDE a été lancé.

#### 17.2.4. Sauvegarder un test

L'enregistrement permet d'exporter et d'importer des tests depuis Selenium IDE uniquement.

Les tests sont enregistrés dans un format de type HTML, les actions sont intégrées dans un tableau.

- Pour procéder à l'enregistrement, cliquer sur **fichier – Sauver le test sous**.
- Pour importer un test existant, cliquer sur **fichier – ouvrir**.

Pour exporter un test en Java (de manière à pouvoir le modifier dans un IDE), cliquer sur **fichier / exporter le test sous / JUnit4 WebDriver**.

#### 17.2.5. Limites du système

L'exportation depuis Selenium IDE vers un langage comme Java pose plusieurs problèmes.

Par exemple, l'instruction suivante, générée par Selenium IDE, ne fonctionne pas dès qu'elle est exécutée avec Selenium WebDriver depuis Eclipse. Nous pouvons remédier à ce problème en utilisant des requêtes plus précises (XPath ou CSS selector) générées depuis l'addon Firefox Firepath. Nous voyons ici un exemple d'instruction générée par Selenium IDE non-fonctionnelle :

```
driver.findElement(By.linkText("Economie et Services")).click();
```

La modification et l'utilisation d'une instruction XPath générée avec l'addon Firefox **Firepath**, permet de corriger le problème :

```
driver.findElement(By.xpath(".*[@id='module307187']/div/div[2]/div/div/p[2]/a")).click();
```

L'outil Selenium IDE peut donc être utilisé dans le but de créer rapidement et simplement un scénario, mais qui devra être manuellement corrigé si certaines étapes ne passent pas lors de l'exécution des tests.

### 17.3. Développement de tests Selenium

Le développement de tests se fait au moyen du système Selenium Web Driver, qui peut être utilisé via différents langages de programmation. La procédure de développement de tests est décrite dans le chapitre suivant (Développement Selenium Web Driver).

## 18. Développement Selenium Web Driver

Considérons à présent le développement avec le module Web Driver de Selenium. Comme cela a été présenté au chapitre précédent (Selenium 2.0), le système Webdriver est utilisé pour développer un test en utilisant la technologie d'automatisation du navigateur via un langage spécifique.

Nous allons commencer par sélectionner un langage, avant de parler de la création de tests, de la syntaxe utilisée et de l'exportation des tests créés. En effet, les tests de performances seront générés au moyen de l'outil JMeter et d'un fichier Java exporté en .jar. Cela sera présenté au chapitre suivant **Benchmark Selenium(Eclipse) – Jmeter**.

### 18.1. Choix d'un langage

Les tests Selenium peuvent être créés en passant par différents langages de programmation. Cette phase de création nécessite un Framework de test unitaire.

Voici une liste officielle des différents langages supportés (<http://docs.seleniumhq.org/>) et des Framework correspondants:

- C# : Utilisation du Framework NUnit
- Java : JUnit ou TestNG disponibles
- Python : unittest, pyunit, robot framework
- Ruby : RSpec, Test::Unit

Le choix s'est porté sur le langage Java et le Framework JUnit pour la raison suivante : L'outil JMeter qui sera utilisé pour analyser les performances des tests, dispose d'un module pour l'importation de code JUnit (JUnit Sampler). Précisons que JUnit est un framework utilisé avec le langage de programmation Java. Il permet de réaliser des tests unitaires, en passant par des annotations (JUnit 4). Ces informations sont tirées du site [www.junit.org](http://www.junit.org).

## 18.2. Création des tests

Les tests créés avec WebDriver peuvent être enregistrés de plusieurs manières différentes :

- En utilisant l'outil Firepath
  - Addon de Firefox, Firepath permet de retrouver des éléments sur une page en utilisant des **CSS Selectors** ou des **requêtes Xpath** qui peuvent ensuite être réutilisés dans le code.
- En utilisant l'outil selenium IDE détaillé au chapitre **Selenium 2.0**, qui nous permet de créer des tests en enregistrant les actions du navigateur Firefox.
  - Rapide, mais les possibilités offertes par Selenium IDE sont plus limitées (il n'est pas possible de récupérer le dernier élément d'une liste par exemple).
  - Le code exporté n'est parfois pas fonctionnel.

Comme nous pouvons le constater, Firepath est le moyen le plus efficace pour repérer un élément sur une page. Un descriptif de cette extension est disponible au chapitre 23.

## 18.3. Création du projet Selenium (Eclipse)

Nous allons maintenant installer un environnement de développement **Java**, et y créer un nouveau projet Selenium Web Driver.

1. Installer l'environnement de développement Eclipse.

<https://www.eclipse.org/downloads/>

2. Créer un nouveau projet Java depuis Eclipse.

3. Télécharger les fichiers suivants :

Serveur Selenium RC : <http://docs.seleniumhq.org/download/>

Client Java Selenium : <http://docs.seleniumhq.org/download/>

Junit : <https://github.com/junit-team/junit/wiki/Download-and-Install>

4. Créer un répertoire **Lib** dans le projet Java.
5. Copier les trois Jars téléchargés dans le répertoire Lib.

6. Bouton droit sur chaque Jar – Build Path – Add to build Path.

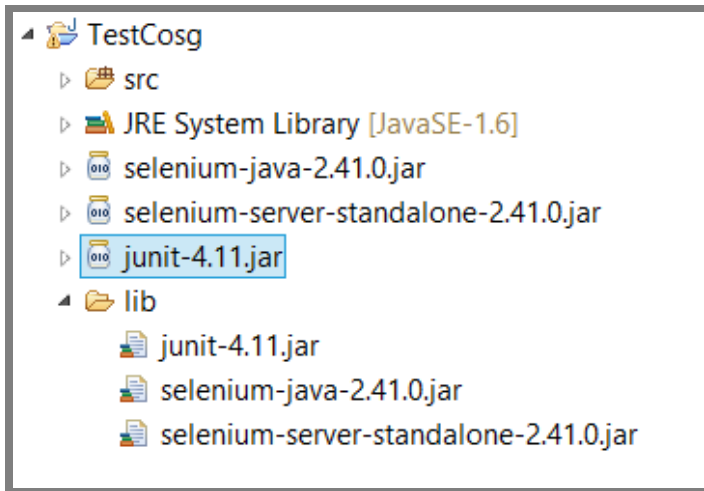


Figure 30 Projet Selenium-Eclipse(screenshot)

## 18.4. Création d'un test initial

Nous allons à présent créer un test à l'intérieur de notre projet Eclipse.

1. Installer la dernière version de Firefox (version 29 minimum).
2. Créer une nouvelle classe dans le projet Java.
3. Y importer un script créé depuis Selenium IDE, ou même créé manuellement (chapitre 16.6. Syntaxe JUnit)

## 18.5. Exécution d'un test

Nous allons maintenant exécuter la classe que nous venons de créer. Pour ce faire, cliquer sur la flèche verte en haut de la fenêtre du logiciel Eclipse.



Figure 31 start test (screenshot - Eclipse)

Le navigateur Firefox va s'exécuter, et réaliser les opérations contenues dans le/les tests de la classe.

D'une fois que les tests sont terminés, la fenêtre d'Eclipse nous indique si les tests se sont passés correctement ou pas. Sur l'image suivante, nous remarquons qu'un test ne s'est pas passé correctement (testWorkbench2).

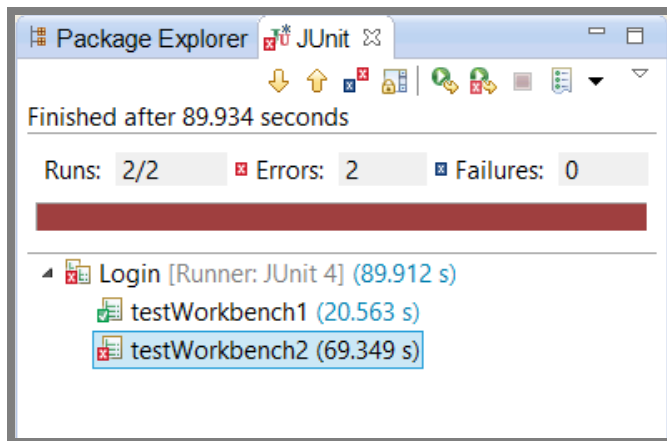


Figure 32 Résultats exécution JUnit (screenshot)

## 18.6. Syntaxe JUnit

### 18.6.1. Annotations

Les annotations sont obligatoires avec JUnit 4, et trois d'entre elles vont être particulièrement utiles pour développer des tests avec Selenium :

- L'annotation **@Test** précède un test JUnit 4.
- L'annotation **@Before** permet d'exécuter un code avant chaque test.
- L'annotation **@After** permet d'exécuter un code après chaque test.

Voici un exemple pour l'annotation **@Before** :

```
@Before //Avant les tests----
public void setUp() throws Exception { //Méthode exécutée
}
```

### 18.6.2. Syntaxe WebDriver

Abordons ensuite la syntaxe WebDriver, qui est en général intégrée dans une méthode située après une annotation **@test**.

Le site de Selenium propose un guide détaillant la base de la syntaxe Java :

<https://code.google.com/p/selenium/wiki/GettingStarted>

Ci-dessous est présentée la syntaxe qui concerne la définition du driver (ici pour Firefox). Cette ligne permet d'instancier un nouveau navigateur :

```
@Before //Avant les tests----  
public void setUp() throws Exception { //Méthode exécutée  
}
```

Accès à une page en passant par le navigateur créé :

```
baseUrl = "http://cyberlearn.hes-so.ch/";  
driver.get(baseUrl + "/");
```

Cliquer sur l'élément atteignable par la requête **Xpath** fournie (obtenue avec l'extension Firepath) :

```
driver.findElement(By.xpath(".*[@id='module307187']/div/div[2]/div/  
div/p[2]/a")).click();
```

Remplir un champ avec le texte « moodle », depuis un champ identifié par une requête **Xpath** :

```
WebElement rechercher =  
driver.findElement(By.xpath(".*[@id='s']"));  
rechercher.sendKeys("moodle");  
rechercher.submit();
```

Contrôler qu'un élément donné contient bien la chaîne « Résultats » :

```
assertEquals("Résultats",  
driver.findElement(By.xpath(".*[@id='content']/small/b[1]")).getText());
```

## 18.7. Limites du système

La simple utilisation de Selenium Web Driver avec un environnement de développement permet d'exécuter des scénarios utilisateur et d'en vérifier le bon fonctionnement mais ne

permet pas d'en tirer des informations de performance.

C'est pour cette raison que les tests développés vont maintenant devoir être intégrés dans le logiciel JMeter, comme cela sera décrit dans le prochain chapitre (Benchmark WebDriver – Jmeter).

## 19. Benchmark WebDriver(Eclipse) – Jmeter

Après avoir parlé du logiciel Selenium et du développement Java via le module Web Driver, nous allons passer à l'installation de l'outil JMeter. Celui-ci nous permettra de réaliser des tests de performance à partir des classes créées en Java.

Les informations présentes dans ce chapitre sont principalement tirées du site officiel de JMeter : <http://jmeter.apache.org/usermanual/get-started.html>

### 19.1. Installation de base

- Télécharger l'outil JMeter depuis ici : [http://jmeter.apache.org/download\\_jmeter.cgi](http://jmeter.apache.org/download_jmeter.cgi)
- Extraire l'archive
- Pour exécuter JMeter, se déplacer dans le répertoire **bin** et lancer le fichier **JMeter.bat** pour Windows, ou **JMeter** pour Linux.
- Télécharger le **Serveur Selenium (RC)** depuis ici :  
<http://docs.seleniumhq.org/download/>
  - Le fichier a normalement été téléchargé lors de la création du projet Eclipse (chapitre Développement Web Driver)
- Copier le fichier dans le répertoire **lib** de JMeter.

### 19.2. Installation des plugins Jmeter

Les plugins JMeter permettent notamment de créer des graphiques de performances avancés. Leur utilisation sera décrite en détail au chapitre 22 : **Benchmark JMeter - Web Driver Sampler**.

- Télécharger le Web Driver ici (JMeterPlugins-Standard-1.1.3.zip) : <http://jmeter-plugins.org/downloads/all/>
- Extraire l'archive, et copier le contenu dans le répertoire de Jmeter.



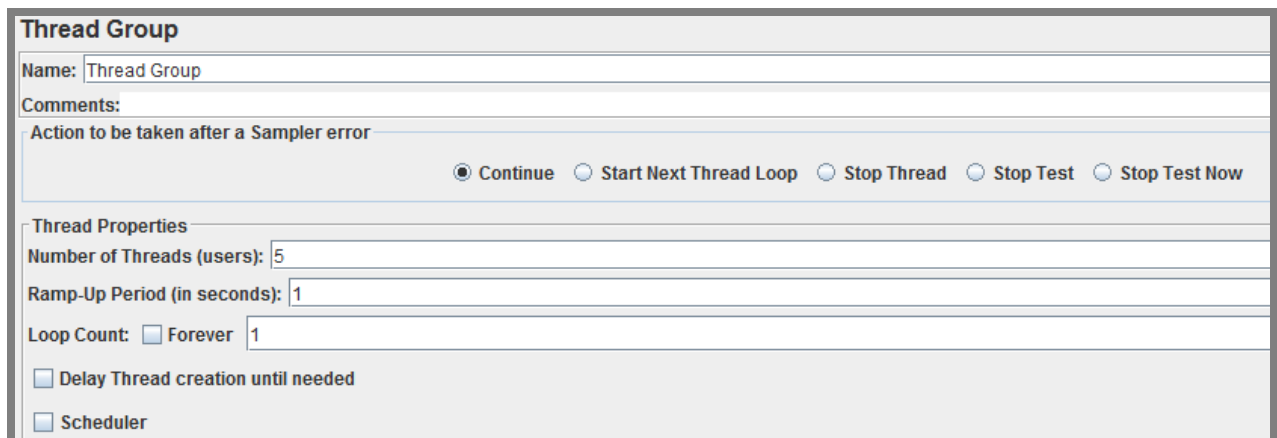
## 19.3. Composants Jmeter

L'exécution de scénarios Selenium depuis JMeter nécessite plusieurs composants qui vont être décrits ici. Leur description se base sur le site officiel de JMeter : [http://jmeter.apache.org/usermanual/test\\_plan.html](http://jmeter.apache.org/usermanual/test_plan.html)

### 19.3.1. Thread Group

Ce composant est à la base de d'un test JMeter. Il permet de définir le nombre d'instances du test qui seront exécutées (Number of Threads), et le temps qu'il faudra pour que toutes ces instances soient lancées (Ramp-Up Period).

Par exemple, si 5 instances sont définies pour 100 secondes, l'outil JMeter exécutera les tests dans le délai imparti (environ 1 instance chaque 20 secondes). Cela évite de surcharger la machine.



The screenshot shows the 'Thread Group' configuration window in JMeter. It includes fields for 'Name' (set to 'Thread Group'), 'Comments', and 'Action to be taken after a Sampler error' (with radio buttons for 'Continue', 'Start Next Thread Loop', 'Stop Thread', 'Stop Test', and 'Stop Test Now'). Below these are 'Thread Properties' with fields for 'Number of Threads (users)' (set to 5), 'Ramp-Up Period (in seconds)' (set to 1), and 'Loop Count' (with a checkbox for 'Forever' and a field set to 1). At the bottom, there are checkboxes for 'Delay Thread creation until needed' and 'Scheduler'.

Figure 33 Thread group JMeter (screenshot)

### 19.3.2. JUnit Request

Au lieu d'utiliser les outils de tests proposés avec JMeter, ce composant va scanner un fichier **.jar** fourni, à la recherche d'une classe JUnit. Ce composant peut donc être utilisé pour transférer un test Selenium exporté en Java depuis Eclipse vers JMeter.

- Ne pas oublier de cocher **Search for JUnit 4 annotations-**
  - Notre test créé depuis Eclipse utilise JUnit 4.
- Sélectionner sous **Classname**, le nom de la classe à utiliser-
- Sélectionner sous **Test Method**, la méthode à exécuter.

**JUnit Request**

Name: JUnit Request

Comments:

☒ Search for JUnit 4 annotations (instead of JUnit 3)

Package Filter:

Classname: moodle\_blog

Constructor String Label:

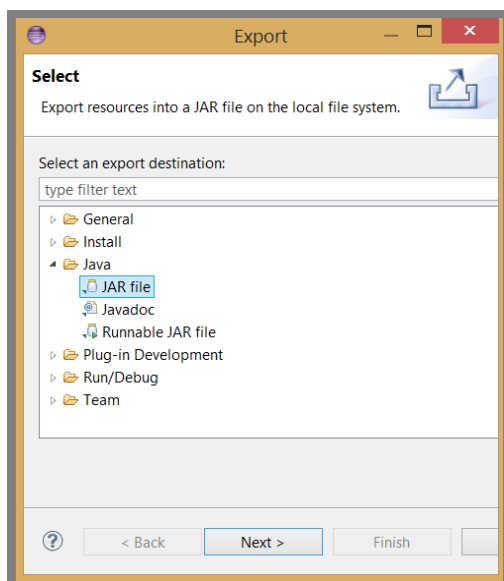
Test Method: testOpenCourse

## 19.4. Exécution des tests WebDriver (Eclipse)

Nous allons maintenant détailler la procédure visant à exporter et exécuter des tests Selenium créés depuis Eclipse (voir chapitre **Développement Selenium**)

Commençons par exporter une classe JUnit créée depuis Eclipse en **.Jar**.

- Sélectionner la classe/projet à exporter depuis Eclipse.
- Bouton droit – Export - .Jar



**Figure 34** Exportation projet Selenium(screenshot - Eclipse)

Exporter vers le répertoire **lib/JUnit** de JMeter.

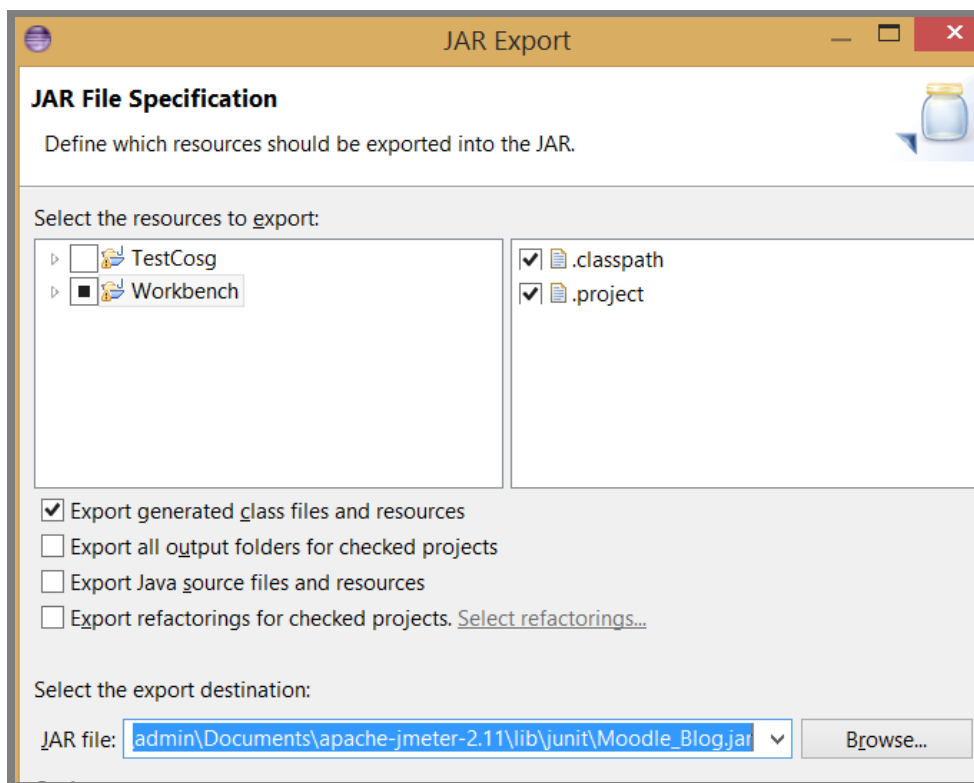


Figure 35 Exportation .jar (screenshot - Eclipse)

Démarrer maintenant JMeter (jmeter.bat), et insérer les éléments suivants dans la zone de gauche :

- **Thread Group** (add – Threads – Thread Group)
  - JUnit Request (add- Sampler – JUnit à l'intérieur du Thread Group).
- **Result Tree** (add – Listener – View Results Tree)

Configurer le **JUnit Request** pour qu'il pointe vers la méthode souhaitée.

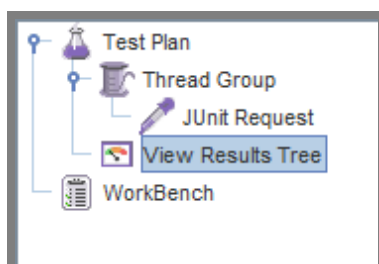


Figure 36 Structre JMeter 1 (screenshot)

## 19.5. Création d'une suite de tests

La création d'une suite de tests passe par plusieurs éléments de type JUnit Request. En effet, chaque élément n'est en mesure de fournir qu'une seule information de performance.

Par exemple pour le JUnit Request exécutant la méthode de connexion sur le site Moodle, nous constatons qu'une seule valeur de performance est affichée (Load time).

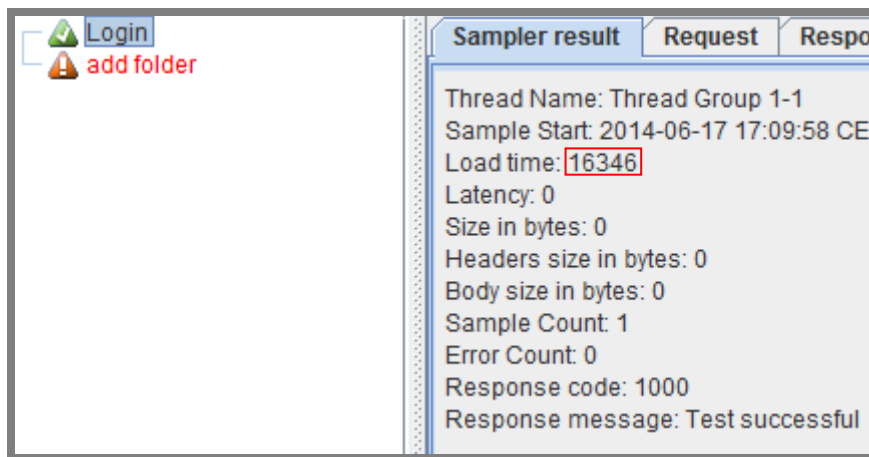


Figure 37 Load Time Sampler (screenshot - JMeter)

Cette situation nécessite donc de créer plusieurs JUnit Request de manière à obtenir une information de performance pour chaque test.

C'est à ce niveau qu'une limite importante de la combinaison JMeter/Web Driver apparaît. Les JUnit Request créés sont suivis de la fermeture du navigateur ouvert par JMeter.

Cela a pour effet de supprimer tous les cookies créés, et ne permet pas de lancer une suite de tests avec des informations de performances pertinentes.

## 19.6. Analyse de l'outil JMeter – Web Driver(Eclipse)

### 19.6.1. Enregistrement des tests

Comme cela a été décrit au chapitre relatif au développement Selenium, l'enregistrement peut se faire de deux manières différentes :

- Avec le plugin Firefox Selenium IDE, l'enregistrement des tests est aisé et intuitif. Mais les difficultés rencontrées avec cet outil (requêtes parfois inutilisables), nous oblige, parfois, à modifier manuellement les tests créés.
- La seconde solution vise à créer les tests manuellement avec le plugin Firepath pour nous aider à construire les requêtes.

L'enregistrement des tests est donc fonctionnel, mais nécessite une bonne connaissance de la syntaxe utilisée et un certain temps pour mener à bien la création des scénarios de test.

Relevons tout de même que l'intuitivité de l'environnement de développement(Eclipse) simplifie grandement la création des scripts de test.

### 19.6.2. Exécution des tests

L'exécution des tests est fonctionnelle mais soulève quelques difficultés :

- Selenium a été conçu pour automatiser un navigateur web, ce qui introduit une consommation importante en termes de performances (maximum 20 navigateurs (threads) en parallèle).
- Les performances obtenues pour le Benchmark peuvent varier, c'est pour cela que les résultats individuels sont très peu pertinents. Seule une moyenne permet d'obtenir des résultats plus probants (sf. Chapitre 22 – plugins JMeter).
- Chaque modification des tests est liée à une recompilation en .jar depuis Eclipse et au redémarrage de JMeter, ce qui est assez lourd.

### 19.6.3. Suite de tests

Il n'a pas été possible de créer une suite de tests, car tous les tests créés sont indépendants. Chaque test est en effet lié à la fermeture du navigateur (pertes des cookies et de la session). Par exemple, le scénario d'ajout de ressource ne contiendra pas la session Moodle qui a été créée lors de la connexion au site (login).

### 19.6.4. Résultats obtenus lors du Benchmark

Login	OK
Accès au cours TB_FX	OK
Ajout d'un répertoire	OK

Même si les scénarios sont fonctionnels, la suite de tests n'a pas pu être réalisée, et les performances globales n'ont donc pas pu être calculées.

### 19.6.5. Rapidité/Stabilité

- L'outil JMeter et l'environnement de développement Eclipse sont très stables. Nous n'avons détecté aucun problème de stabilité (pas de plantages, l'interface est tout à fait fonctionnelle).
- L'installation des logiciels est très rapide, mais nécessite l'installation de nombreux modules ce qui ralentit considérablement la procédure.

### 19.6.6. Qualité des rapports

Les rapports générés sont à la hauteur des attentes, hormis que les performances globales n'ont pas pu être calculée, faute d'avoir pu créer une suite de tests. Précisons que la génération de graphiques au moyen des plugins JMeter est décrite au chapitre 22 : Benchmark JMeter – Web Driver Sampler.

Avec cette combinaison d'outils, nous avons donc été en mesure de produire les données suivantes :

- Analyse des threads
- Gestion des temps d'exécution des scénarios en fonction des threads

## 20. Benchmark FunkLoad

Nous allons maintenant procéder au Benchmark du logiciel FunkLoad et en premier lieu, à l'installation de l'outil et de ses composants.

D'après <https://funkload.nuxeo.org/intro.html#features>, Le logiciel Funkload utilise une technologie qui vise à reproduire les requêtes GET/POST du Benchmark directement au sein de l'outil. Contrairement à Selenium, Funkload ne lance pas de navigateur. Cela augmente nettement les performances du logiciel en termes d'utilisateurs virtuels, mais empêche la réalisation de certaines actions (clic sur un bouton, chargements dynamiques, ...).

### 20.1. Installation de base

Télécharger et installer Python 2.7 pour Windows

- <https://www.python.org/downloads/>
- Python est installé par défaut dans **C:\Python27**

## Installer **setupTools**

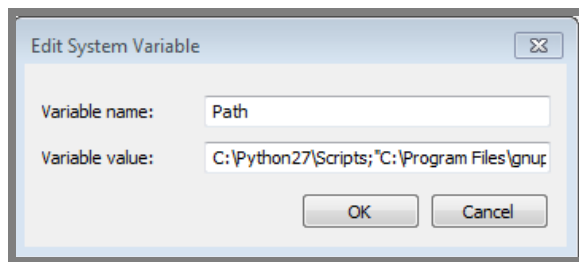
- <https://pypi.python.org/pypi/setuptools>
- Installé dans **C:\Python27\Scripts**

Au moyen du terminal Windows, se positionner dans le répertoire **C:\Python27\Scripts** et exécuter les commandes suivantes :

```
Easy_install docutils
easy_install webunit
easy_install -f http://funkload.nuxeo.org/snapshots/ -U funkload
```

Ajouter python à la variable d'environnement en lançant manuellement le script fourni par python.

- C:\Python27\Tools\Scripts\ win\_add2path.py



**Figure 38** Variable d'environnement Python (screenshot - XP sp3)

Exécuter maintenant la commande depuis le répertoire Scripts **fl-install-demo** pour que FunkLoad installe les fichiers de démonstration.

## 20.2. Installer TCPWatch

TCPWatch va nous permettre d'enregistrer des tests FunkLoad, en passant par un proxy défini sur le navigateur utilisé pour enregistrer les tests.

- Télécharger TcpWatch ici : <https://pypi.python.org/pypi/tcpwatch>
- Extraire l'outil dans le répertoire Python (C:\python27).
- Se déplacer dans le répertoire extrait à l'aide d'un terminal.
- Exécuter les commandes suivantes :

```
python setup.py build
python setup.py install
```

### 20.3. Installer GnuPlot

GnuPlot est utilisé par Funkload pour générer des rapports avancés. En effet, les commandes de base Funkload ne permettent pas de générer des visualisations avancées (graphiques, tendances, ...).

1. Télécharger GnuPlot depuis ici (version 4.2 minimum):  
<http://www.tatsuromatsuoka.com/gnuplot/Eng/winbin/>
2. Installer le logiciel via l'installateur fourni.
3. Ajouter le chemin de l'installation Gnuplot dans la variable d'environnement :
  - a. C:\Program Files\gnuplot\bin

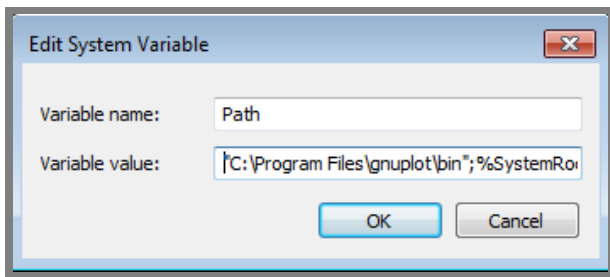


Figure 39 Variable d'environnement Gnuplot (screenshot - XP sp3)



Contrôler que Gnuplot est bien accessible en utilisant la commande **wgnuplot** dans un terminal (ce qui devrait lancer le logiciel Gnuplot)

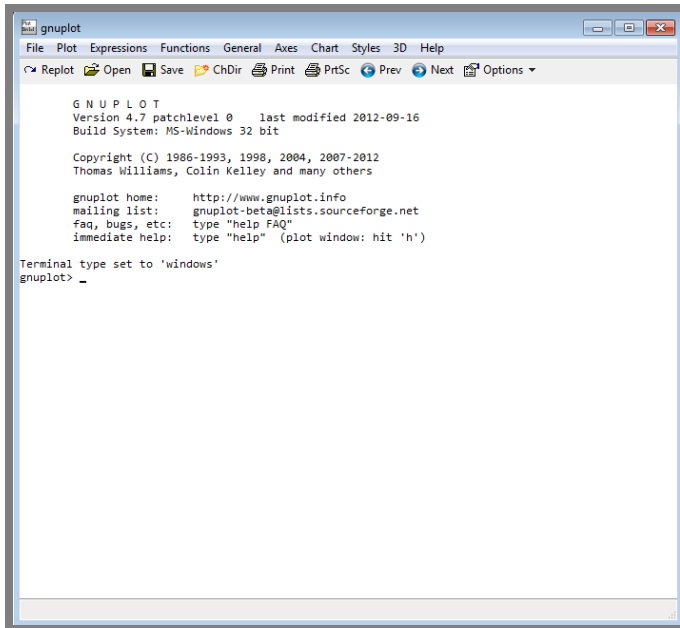


Figure 40 Interface Gnuplot (screenshot)

## 20.4. Configuration du proxy avec Firefox

Nous allons maintenant configurer le navigateur Firefox pour qu'il envoie les requêtes réalisées vers FunkLoad.

L'utilisation de TcpWatch requiert l'utilisation d'un proxy depuis le navigateur utilisé pour enregistrer le test.

Voici la procédure pour que les requêtes du navigateur passent par le port **8090** utilisé par TcpWatch :

Dérouler le menu Firefox au moyen de l'icône située en haut à droite de la fenêtre :

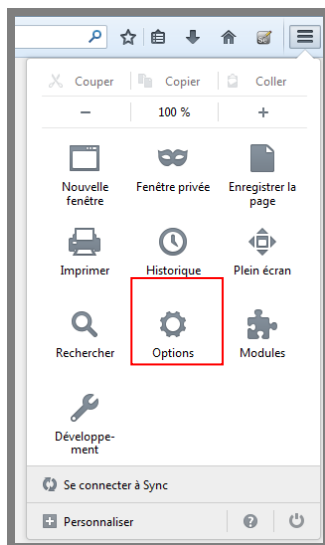


Figure 41 Menu Firefox v28 (screenshot)

Ouvrir les paramètres de connexion depuis l'onglet Réseau :

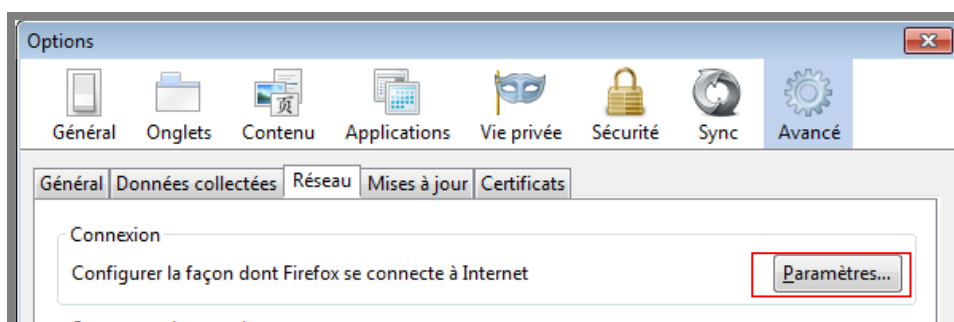


Figure 42 Onglet réseau Firefox v28 (screenshot)

Utiliser le proxy **localhost** (machine locale) et le port **8090** (utilisé par Tcpwatch).

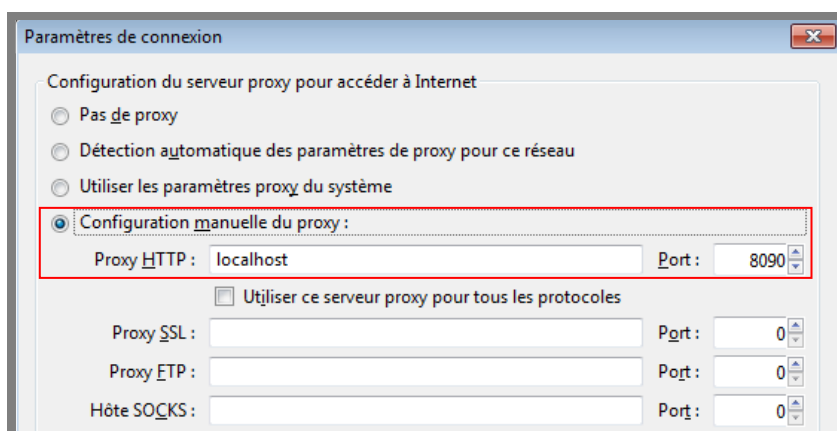
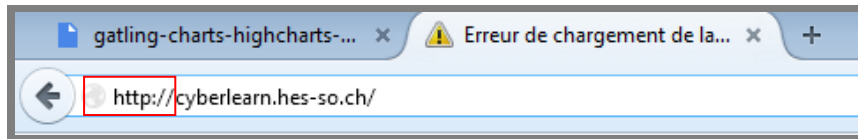


Figure 43 Configuration proxy Firefox 28 (screenshot)

Le trafic généré par le navigateur ne doit pas être crypté, de manière à ce qu'il puisse être analysé par FunkLoad. Cela signifie qu'il faut accéder via **HTTP** aux sites cryptés par défaut.



## 20.5. Enregistrer un test

Nous allons maintenant enregistrer un test via TcpWatch ainsi que le proxy que nous venons de définir sur le navigateur Firefox.

1. Installer **TCPWatch** si ce n'est pas déjà fait.
2. Se positionner avec le terminal Windows dans le répertoire où l'on souhaite que le test soit enregistré.
3. Utiliser commande **fl-record [name]** pour démarrer l'enregistrement d'actions depuis un navigateur. Ces actions seront enregistrées dans un fichier **[name].py**.
  - Utilisation du proxy **localhost :8090** sur le navigateur utilisé.
  - Utiliser les touches **Ctrl+C** pour arrêter l'enregistrement.
4. Le navigateur configuré peut maintenant être utilisé pour se rendre sur le site Moodle et pour réaliser les opérations du scénario.
  - Pour voir si l'enregistrement se passe bien, contrôler que le contenu du terminal ouvert se modifie lors d'une requête sur le navigateur.

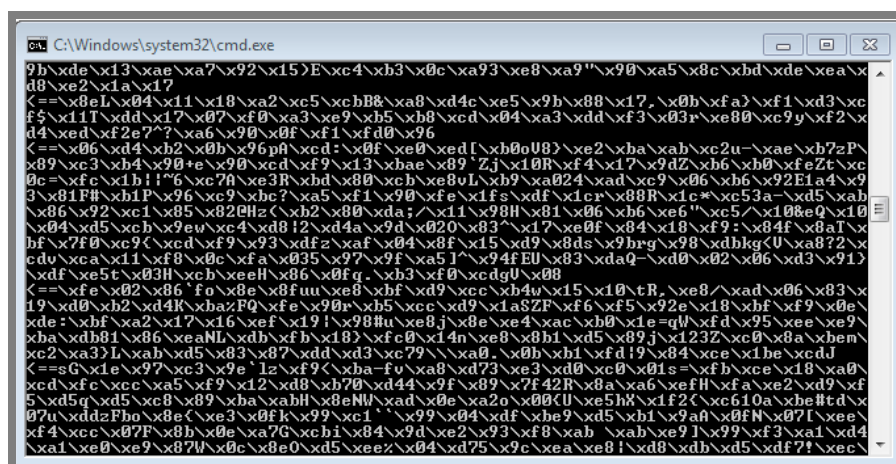


Figure 44 terminal enregistrement test Funkload (screenshot)

Utiliser les touches Ctrl+C sur le terminal pour arrêter l'enregistrement.

Deux fichiers ont maintenant été créés :

- **Test\_[name].py**
  - Contient le code Python du test enregistré.
  - Bouton droit – Edit with IDLE pour éditer le contenu.
- **[name].conf**
  - Ce fichier contient la configuration du test dont fait partie l'URL de départ ou encore, le nombre d'utilisateurs virtuels.

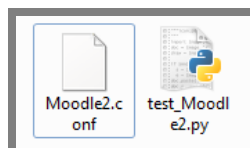


Figure 45 Fichiers Funkload (screenshot)

## 20.6. Développement Funkload

Maintenant que nous avons enregistré un premier test, nous allons comprendre la syntaxe utilisée par le logiciel FunkLoad.

Le développement de tests pour Funkload se fait en Python. Les tests sont séparés en deux parties :

- Le test à proprement parler.
- Le fichier de configuration.

### 20.6.1. Fichier de test

Le fichier de test **test\_[class].py** contient toutes les requêtes qui seront exécutées durant le test.

Le code ci-dessous permet d'appeler l'URL de départ située dans le fichier de configuration du test (section main).

```
def setUp(self):  
    self.logd("setUp")
```

```
""" Chargement de l'url de départ depuis Moodle2.conf"""  
self.server_url = self.conf_get('main', 'url')
```

## 20.7. Fichier de configuration

Le fichier de test (.py) est lié à un fichier de configuration (.conf). Celui-ci doit porter le nom de la classe python (**[class].conf**).

Le fichier de configuration est séparé en sections, dont voici les fonctions principales :

1. Section **[main]** : contient l'url de départ du test, ou la description du test.

```
[main]  
title=Moodle2  
description= Test accès liste cours informatique de gestion (moodle)  
  
# the server url to test  
url=http://cyberlearn.hes-so.ch
```

2. Section **[Bench]** : Cette section est relative à l'exécution du test. Elle permet de définir le nombre d'utilisateurs virtuels, et la durée de chaque cycle de test.
  - Par exemple dans le script ci-dessous, deux cycles seront exécutés pendant 60 secondes, avec respectivement 2 et 4 utilisateurs virtuels.

```
[bench]  
  
# cycles = list of cycles with their number of concurrent users  
cycles = 2;4  
  
# duration = duration of a cycle in seconds  
duration = 60
```

## 20.8. Exécuter un test

Nous allons maintenant comprendre comment exécuter un test créé avec Funkload :

Enregistrer un test avec la commande **fl-record** (voir la procédure « Enregistrer un test »)

```
fl-record[testName]
```

Se placer avec un terminal dans le répertoire du testé créé, puis exécuter le test avec la commande **fl-run-test**.

```
fl-run-test -dV test_[name].py
```

La commande retourne le temps d'exécution du test si celui-ci s'est déroulé sans difficultés, ou alors le descriptif de l'erreur si un problème est survenu.

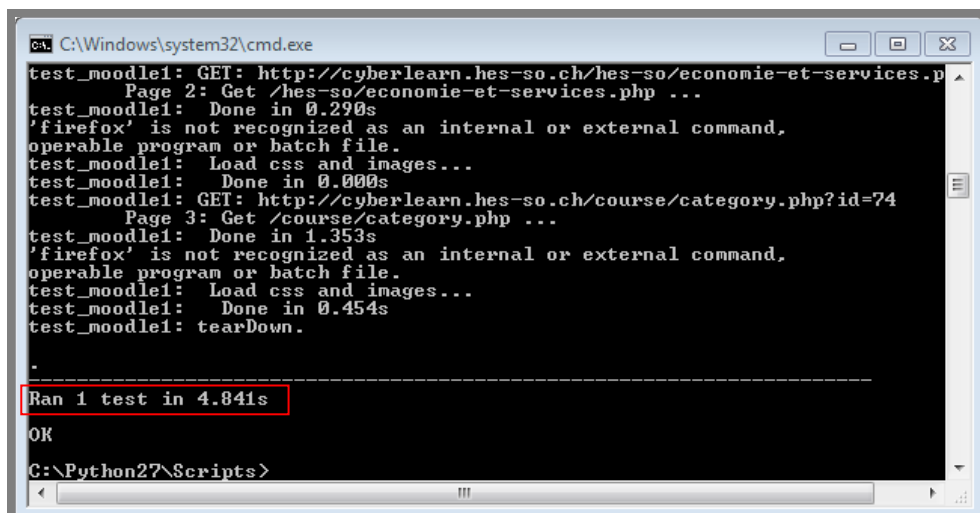


Figure 46 test Funkload exécuté (screenshot)

La commande **fl-run-test** a pour effet de créer trois nouveaux fichiers dans le répertoire du projet Funkload :

- **[name]-test.xml**
  - Contient le détail des opérations de l'exécution du test. Tous les temps des requêtes exécutées y sont indiqués.
- **Moodle2-test.log**
  - Utile pour disposer de la trace exacte des erreurs.

- **Test\_[name].pyc**
  - Code compilé généré par Python (<https://docs.python.org>).

## 20.9. Création d'une suite de tests

Après avoir parcouru le site officiel de FunkLoad et effectué diverses recherches, nous n'avons pas trouvé de solution satisfaisante pour créer une suite de tests avec cet outil.

En effet la commande **fl-run-bench** a été conçue pour exécuter un seul test, et ne peut pas être utilisée pour exécuter et générer des rapports sur une classe complète qui contiendrait plusieurs scénarios.

En plaçant les requêtes des tests à la suite dans la même méthode, il est néanmoins possible d'automatiser l'exécution des tests, mais sans que le rapport généré ne fasse de synthèse des scénarios utilisateur. Seuls les temps des sous-requêtes et les temps globaux seront affichés avec cette solution.

## 20.10. Génération de rapports

Comme cela est décrit dans le chapitre sur les attentes du logiciel de test, il faudra que soit possible l'analyse des résultats fournis par le logiciel, sous forme de graphique par exemple. Pour ce faire, commencer par repérer dans le fichier Python (.py) les éléments suivants :

- Le nom de la classe (**Moodle1**).

```
class Moodle1(FunkLoadTestCase):
```

- Le nom de la méthode de test (**test\_moodle1**).

```
Def test_moodle1(self):
```

- Le nom du fichier Python (**test\_Moodle1.py**).

Nous allons maintenant définir les éléments suivants dans le fichier **[name].conf** :

- Le nombre de cycles par test.
- La durée des cycles (30 secondes par exemple).
- Les utilisateurs virtuels par cycle.

```
[bench]
```

```
# cycles = list of cycles with their number of concurrent users
```

```
cycles = 1
```

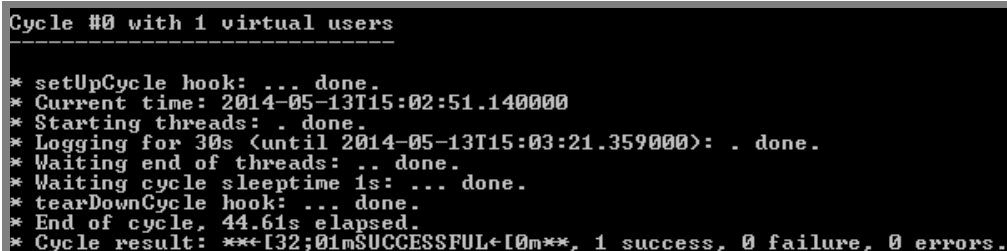
```
# duration = duration of a cycle in seconds
```

```
duration = 20
```

Pour exécuter le test avec les paramètres définis dans le fichier **[name].conf**, utiliser la commande présentée plus bas. Elle va nous permettre de générer un rapport au format XML qui contiendra les résultats des tests créés à partir des données définies précédemment (cycles, utilisateurs, ...).

```
fl-run-bench [filename.py] [class].[method]
```

```
fl-run-bench test_moodle1.py Moodle1.test_moodle1
```



```
Cycle #0 with 1 virtual users
-----
* setUpCycle hook: ... done.
* Current time: 2014-05-13T15:02:51.140000
* Starting threads: . done.
* Logging for 30s (until 2014-05-13T15:03:21.359000): . done.
* Waiting end of threads: .. done.
* Waiting cycle sleeptime 1s: ... done.
* tearDownCycle hook: ... done.
* End of cycle, 44.61s elapsed.
* Cycle result: ***[32;01mSUCCESSFUL+[0m**, 1 success, 0 failure, 0 errors.
```

Figure 47 Exécution test de performance Funkload (screenshot - 1 thread)

Deux fichiers ont maintenant été créés dans le répertoire du test :

- Moodle2-bench.xml
  - Détail des temps de tests (données brutes)
- Moodle2-bench.log
  - Fichier log relatif au déroulement.

Le fichier XML généré n'est pas utilisable directement (données brutes). Il doit donc être interprété au moyen de statistiques pour obtenir des résultats probants.



Pour ce faire, Funkload utilise le logiciel Gnuplot qui doit être précédemment installé. La commande suivante génère un fichier html qui se base sur le fichier XML précédemment créé par la commande fl-run-bench.

```
fl-build-report --html simple-bench.xml
```

```
C:\Users\VM-client1\moodle2>fl-build-report --html moodle2-bench.xml
Creating html report: ...done:
C:\Users\VM-client1\moodle2\test_moodle2-20140514T103520\index.html
```

Figure 48 Génération rapport html Funkload (screenshot)

Un nouveau répertoire a été créé, contenant le bilan de l'analyse réalisée par GnuPlot. On y retrouve notamment un fichier **index.html** permettant d'afficher les différents graphiques (format .png) et les tableaux d'analyse dans un navigateur web. Détaillons à présent le rapport fourni par le logiciel lors de l'exécution du processus de connexion(login).

En haut du rapport, nous retrouvons les différentes pages qui ont été chargées durant le test.

- 7 Page detail stats
  - 7.1 PAGE 001: Get /
  - 7.2 PAGE 002: Get /login/index.php
  - 7.3 PAGE 003: Post /login/index.php
  - 7.4 PAGE 004: Get /login/index.php

Figure 49 Chargement pages rapport Funkload (screenshot)

Le rapport nous présente aussi les détails de l'exécution. Ici nous testons la plateforme en trois étapes de 30 secondes, avec respectivement 1, 2 et 3 utilisateurs virtuels.

- Launched: 2014-06-23 14:23:30
- From: VM-client1-PC
- Test: test\_Login.py Login.test\_login
- Target server: <http://cyberlearn-dev.hes-so.ch>
- Cycles of concurrent users: [1, 2, 3]
- Cycle duration: 30s

Figure 50 Détails exécution test Funkload (screenshot)

Le logiciel nous présente le temps moyen de chargement des requêtes réalisées (axe Y), en fonction des utilisateurs virtuels définis (axe X).

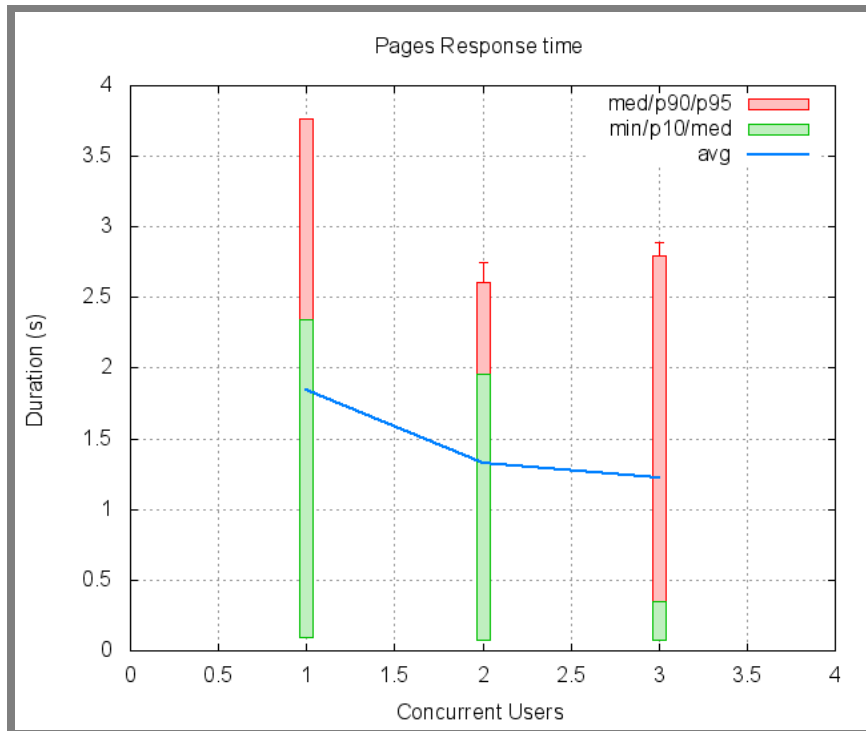


Figure 51 Graphique rapport Funkload (screenshot)

La suite du rapport présente un graphique similaire pour toutes les autres requêtes réalisées.

## 20.11. Analyse du logiciel FunkLoad

### 20.11.1. Enregistrement des tests

Le système d'enregistrement des tests basé sur TcpWatch est efficace, mais il lui manque certaines fonctions utiles. En effet, ce système ne permet pas d'enregistrer les tests à un autre endroit que là où le terminal est lancé. Il est donc nécessaire de créer une variable d'environnement pour rendre le script **fl-record** disponible partout ailleurs.

De plus, le terminal n'affiche pas les requêtes réalisées depuis le navigateur mais plutôt des suites de caractères incompréhensibles. De ce fait, il ne permet pas de modifier dynamiquement le test.

### 20.11.2. Suite de tests

L'outil ne permet pas de réaliser une suite de tests permettant d'obtenir un rapport présentant les performances de chaque scénario utilisateur.

### 20.11.3. Résultats obtenus lors du Benchmark

Login	OK
Accès au cours TB_FX	OK
Ajout d'un répertoire	NOK

Le Benchmark n'a pas pu être réalisé entièrement car la technologie utilisée par le logiciel FunkLoad (requêtes GET/POST) ne permet pas de simuler le clic sur le bouton d'ajout de répertoire, ce qui empêche la création de la ressource. Les détails de ce problème sont expliqués au chapitre **21.9.7. Problèmes liés à l'ajout d'une ressource**

### 20.11.4. Stabilité/rapidité

Que ce soit lors de la création ou lors de l'exécution des tests, le logiciel a été très stable et aucun plantage n'a été constaté. De plus, l'outil est plutôt rapide, du fait de son interface basique en ligne de commande.

### 20.11.5. Qualité des rapports

Tout d'abord, nous pouvons dire que le format des rapports (html) permet de les déplacer facilement, ou de les rendre disponibles depuis un serveur Web. Les rapports sont clairs, ils affichent les résultats globaux et ceux des sous-requêtes. Par contre, du fait qu'il n'a pas été possible de créer une suite de tests, les résultats ne sont donc pas regroupés en fonction du scénario (login, ajout de ressource, ...)

Nous constatons que la génération des rapports est une opération fastidieuse qui nécessite d'employer deux lignes de commande complexes, de relever le nom du fichier, celui de la classe, et de se positionner dans le bon répertoire.

## 21. Benchmark Gatling

D'après <https://github.com/excilys/gatling/wiki/First-Steps-with-Gatling>, Gatling est un outil de test de performances Open Source basé sur l'exécution de requêtes GET/POST. Cette technologie permet à l'outil de lancer plusieurs centaines d'utilisateurs virtuels en parallèle. Cela le rend particulièrement efficace pour réaliser des tests de charge. Précisons que Gatling utilise son propre navigateur, et ne simule pas une instance d'un navigateur grand public (comme Chrome ou Firefox).

Nous allons commencer par détailler l'installation de Gatling, avant de parler de la création et de l'exécution des tests, de la suite de tests et des rapports générés.

Finalement, nous analyserons les capacités de cet outil à partir des informations que nous tirerons du Benchmark.

### 21.1. Installation

- Télécharger l'outil Gatling depuis le site <http://gatling-tool.org/> rubrique **Download**.
- Décompresser l'archive (éviter le répertoire **C:\Program Files** car les permissions y sont faibles).
- Se positionner dans le répertoire **[Gatling\_Install]\bin** et exécuter le script **gatling.bat** .
  - Si un problème survient lors du lancement du script avec l'instruction **-server**, supprimer les instructions suivantes dans le fichier.bat à la ligne 40. Cela est lié à la version du JRE (32 bits).
    - -server
    - -XX:+OptimizeStringConcat
- Gatling est maintenant installé, et peut être exécuté via le script défini plus haut.

### 21.2. Enregistrer un test

Lancer le script **recorder.bat** situé dans **[Gatling\_Install]\bin** .

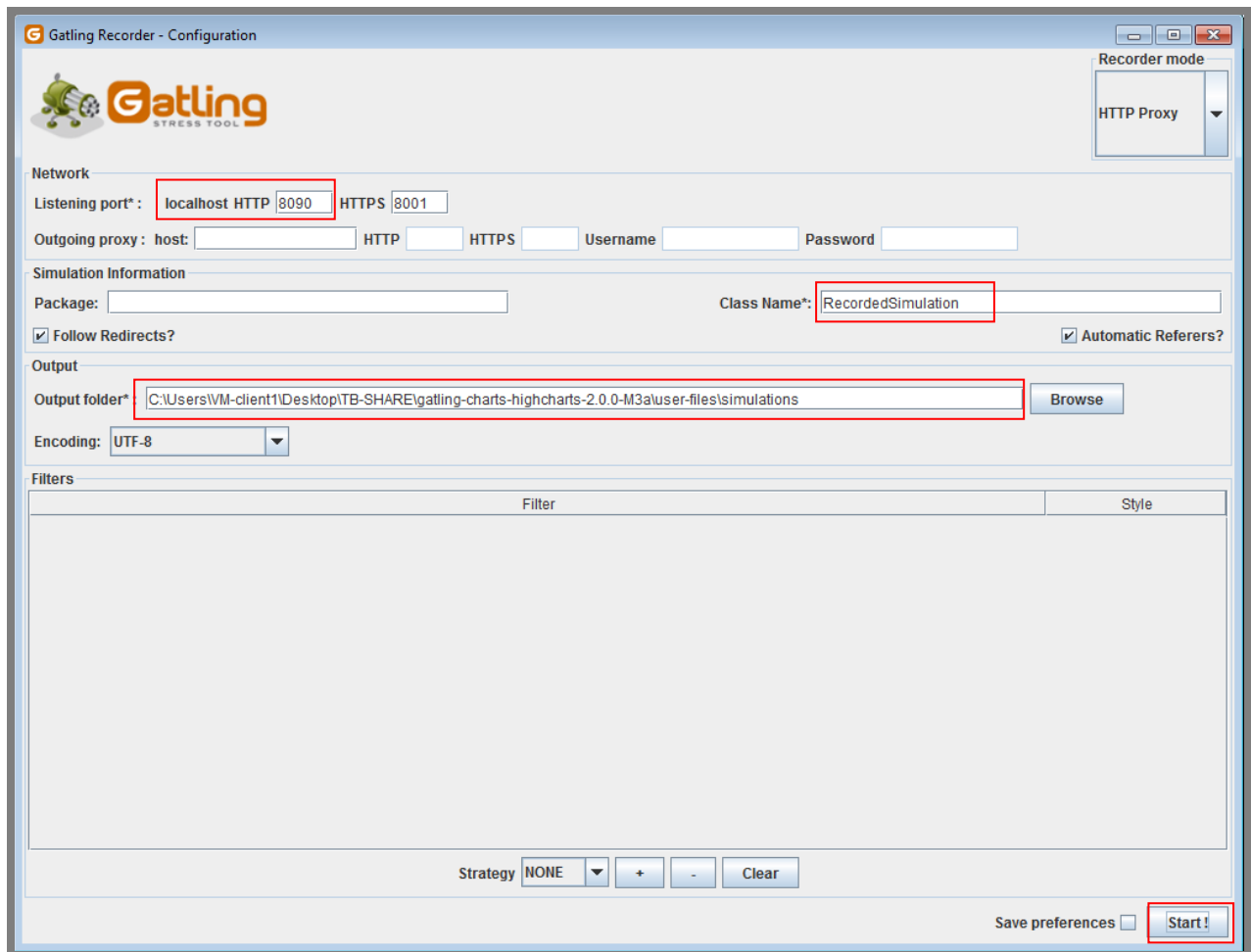


Figure 52 Gatling - recorder window

- Sélectionner le port (proxy) utilisé par le navigateur pour les requêtes http et https.
  - En fonction de la redirection proxy configurée sur le navigateur utilisé, pour enregistrer le test. La configuration du proxy avec Firefox est décrite au chapitre 19.4.
- Entrer une destination pour le test qui sera créé dans la zone de saisie **Output Folder**
- Donner un nom au scénario qui sera créé dans la zone de saisie **Class Name**
- Cliquer sur **Start** pour démarrer l'enregistrement.

Précisons que l'outil Gatling construit des scénarios qui contiennent non seulement le chargement des pages du test, mais aussi toutes les sous-requêtes. Par exemple, le chargement de la page Moodle doit être suivi de l'appel de toutes les sous-requêtes (images, scripts, ...).

Cela complexifie considérablement les scripts de test. Précisons que le logiciel Funkload, testé précédemment, télécharge automatiquement les requêtes sous-jacentes.

200	GET	/	cyberlearn.hes-so.ch	html	22.32 Ko	→ 345 ms				
200	GET	embedded-wayf.js	wayf.switch.ch	js	109.28 Ko	→ 448 ms				
200	GET	awstats_misc_tracker.js?screen=12...	cyberlearn.hes-so.ch	js	0 Ko	→ 165 ms				
200	GET	Session	cyberlearn.hes-so.ch	html	0 Ko	→ 166 ms				
200	GET	yui_combo.php?3.7.3/build/cssres...	cyberlearn.hes-so.ch	css	3.51 Ko	→ 50 ms				
200	GET	yui_combo.php?3.7.3/build/widge...	cyberlearn.hes-so.ch	css	1.14 Ko	→ 82 ms				
200	GET	yui_combo.php?3.7.3/build/cssbut...	cyberlearn.hes-so.ch	css	10.18 Ko	→ 101 ms				
200	GET	yui_combo.php?moodle/13994408...	cyberlearn.hes-so.ch	css	2.56 Ko	→ 88 ms				
200	GET	all	cyberlearn.hes-so.ch	css	393.55 Ko	→ 339 ms				

Figure 53 requêtes chargement Moodle (screenshot - Chrome)

## 21.3. Exécuter un test

- Lancer le script gatling.bat situé dans **[Gatling\_Install]\bin**
- Le programme va maintenant rechercher les tests présents dans le répertoire **[Gatling\_Install]\user-files\simulations**. Il est donc nécessaire de placer les fichiers de tests créés dans ce répertoire.
- S'il n'y a pas d'erreurs de compilation dans les fichiers de test, l'outil affiche la liste des tests présents et nous demande lequel nous souhaitons exécuter :

```

C:\Windows\system32\cmd.exe
-M3a"lib\*; "C:\Users\UM-client1\Desktop\TB-SHARE\gatling-charts-highcharts-2.0.0-M3a"conf; "C:\Users\UM-client1\Desktop\TB-SHARE\gatling-charts-highcharts-2.0.0-M3a"user-files;
C:\Users\UM-client1\Desktop\TB-SHARE\gatling-charts-highcharts-2.0.0-M3a\bin>set
COMMAND=cp "C:\Users\UM-client1\Desktop\TB-SHARE\gatling-charts-highcharts-2.0.0-M3a"lib\*; "C:\Users\UM-client1\Desktop\TB-SHARE\gatling-charts-highcharts-2.0.0-M3a"conf; "C:\Users\UM-client1\Desktop\TB-SHARE\gatling-charts-highcharts-2.0.0-M3a"user-files; io.gatling.app.Gatling
C:\Users\UM-client1\Desktop\TB-SHARE\gatling-charts-highcharts-2.0.0-M3a\bin>java
-XX:+UseThreadPriorities -XX:ThreadPriorityPolicy=42 -Xms512M -Xmx512M -Xmn100M -XX:+HeapDumpOnOutOfMemoryError -XX:+AggressiveOpts -XX:+UseFastAccessorMethods -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:+CMSClassUnloadingEnabled -XX:SurvivorRatio=8 -XX:MaxTenuringThreshold=1 -XX:CMSInitiatingOccupancyFraction=75 -XX:+UseCMSInitiatingOccupancyOnly -cp "C:\Users\UM-client1\Desktop\TB-SHARE\gatling-charts-highcharts-2.0.0-M3a"lib\*; "C:\Users\UM-client1\Desktop\TB-SHARE\gatling-charts-highcharts-2.0.0-M3a"conf; "C:\Users\UM-client1\Desktop\TB-SHARE\gatling-charts-highcharts-2.0.0-M3a"user-files; io.gatling.app.Gatling
Choose a simulation number:
[0] RecordedSimulation
[1] advanced.AdvancedExampleSimulation
[2] basic.BasicExampleSimulation

```

Figure 54 Affichage des tests (screenshot - Gatling)

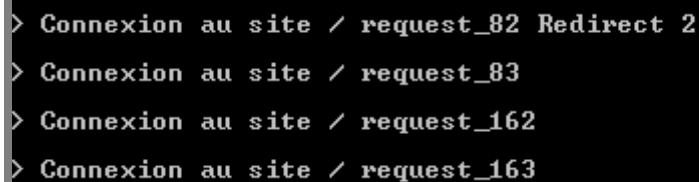
- Sélectionner maintenant le test à exécuter (0 par exemple).
- D'une fois que le test aura été exécuté par Gatling, un rapport sera généré dans le répertoire **[Gatling\_Install]\results**

- A l'intérieur du répertoire du rapport, cliquer sur **index.html** pour afficher les résultats du test.

## 21.4. Simplifier un test généré par le Recorder

Comme chaque navigateur moderne, Gatling mémorise le résultat des requêtes dans un cache, ce qui évite de devoir télécharger plusieurs fois la même ressource. Les requêtes redondantes sont donc évitées par Gatling.

Pour simplifier le script, les requêtes qui n'ont pas été exécutées peuvent être supprimées. Nous constatons par exemple que sur l'image suivante, le logiciel passe de la requête 83 à la requête 162. Nous pouvons donc supprimer les requêtes 84 à 161.



```
> Connexion au site / request_82 Redirect 2
> Connexion au site / request_83
> Connexion au site / request_162
> Connexion au site / request_163
```

Figure 55 requêtes annulées Gatling (screenshot)

## 21.5. Syntaxe utilisée

### 21.5.1. Exec/pause

L'instruction **.exec** permet de lancer une requête GET ou POST sur le serveur.

Nous voyons ici une première requête **.exec** permettant d'accéder à une page sur le site Moodle :

```
.exec(http("Open Course")
      .get("""/course/view.php?id=4861""")
      .headers(headers_2))
.pause(953 milliseconds)
```

- Dans cet exemple, la requête a été nommée **Open Course**. Ce nom nous permettra de reconnaître facilement la requête dans le rapport de performances qui sera généré plus tard.
- Elle pointe vers l'url **[BaseUrl]/course/view.php?id=4861**
- La requête utilise le header **header\_2** pour s'exécuter. Le concept de header est expliqué dans le glossaire.
- Elle sera suivie d'une pause de 953 millisecondes. Les pauses sont calculées automatiquement lors de l'enregistrement du test.

Voici une autre requête `.exec` permettant d'exécuter un scénario externe **Scenario\_Login.scala**, lui-même stocké la chaîne **chain\_login**. Ce système permet de séparer les scénarios en modules.

```
val chain_login = exec(Scenario_Login.login)
```

### 21.5.2. Group

L'instruction `.group` permet de regrouper plusieurs requêtes de type **exec**. Cette instruction est très importante car les performances générées par Gatling sont regroupées en fonctions des groupes définis. Par exemple en créant un groupe pour le scénario **login**, nous retrouverons les performances du scénario, et celles de toutes les sous-requêtes.

```
.group("login"){
  exec(http("request_85")
    .get("/hes-so/economie-et-services.php")
    .headers(headers_1))
}
```

### 21.5.3. Set Up

La commande Set Up est utilisée pour exécuter un scénario avec un certain nombre d'utilisateurs virtuels et cela pendant un laps de temps défini. Cette instruction sera décrite en détail lors de la création de la suite de tests.

```
setUp(Scenario_AddFolder.addFolder.inject(atOnce(1user))).protocols(httpProtocol)
```



#### 21.5.4. Fichiers de test

Nous allons ici approfondir les composantes d'un fichier de test Gatling. Les fichiers de test sont composés de quatre parties, comme cela est décrit dans le guide fourni sur le site de Gatling : <https://github.com/excilys/gatling/wiki/First-Steps-with-Gatling>

- The HTTP protocol configuration
  - Contient l'URL de base et sert de point de départ pour les requêtes du script.
- The headers definition
  - Contient la liste de tous les entêtes utilisés par les requêtes du scénario.
- The scenario définition
  - Composé de deux méthodes principales.
    - Pause (Pour mettre le test en pause).
    - Exec (pour réaliser une requête get/post).

```
.exec(http("request_1")  
.get("https://aai-logon.hes-so.ch/idp/Authn/UserPassword")  
.headers(headers_1))  
.pause(375 milliseconds)
```

- The simulation definition
  - Cette partie est utilisée pour définir la charge que l'on souhaite injecter sur le serveur.
    - Nombre d'utilisateurs virtuels.
    - Durée de la charge.

#### 21.6. Création d'une suite de tests

La génération d'une suite de tests Gatling nécessite de disposer des fichiers suivants :

- Un fichier de simulation.
- Un ou plusieurs scénarios.

- Un ou plusieurs fichiers Header.

L'exécution d'une suite de tests passe par le lancement d'un fichier de simulation, qui appellera ensuite les différents scénarios prévus.

Chaque requête présente dans un scénario est liée à un header. Les headers sont regroupés au sein d'un ou plusieurs fichiers prévus à cet effet.

Chaque scénario peut être exécuté par un certain nombre d'utilisateurs virtuels, et cela pendant un laps de temps donné. Ces informations sont renseignées dans le fichier de simulation.

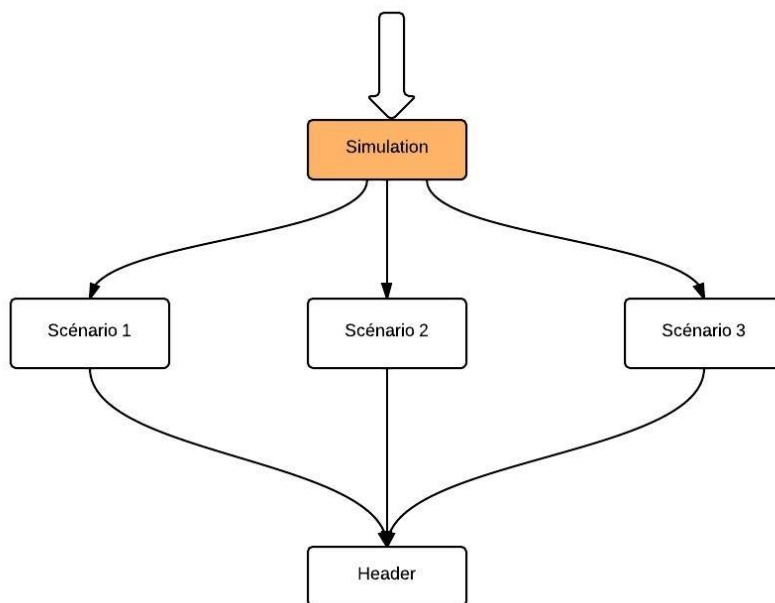


Figure 56 Schéma suite de tests Gatling (screenshot - Visio)

### 21.6.1. Fichier de simulation

La première étape pour créer une suite de tests avec Gatling consiste à créer le fichier de simulation qui sera appelé lors du lancement de la suite de tests. Ce fichier contient l'url de départ et l'appel des scénarios.

Le script ci-dessous nous présente une suite au départ du site <http://cyberlearn-dev.hes-so.ch/>, et exécutant le scénario **HomePage**, suivi du scénario **LoadCourse**.

```
class TestSuite_Moodle extends Simulation {

    val httpProtocol = http.baseUrl("http://cyberlearn-dev.hes-so.ch/").disableFollowRedirect /** URL de départ */

    setUp(Scenario_HomePage.scn.inject(ramp(10 users) over (10 seconds)), /** Appel du scénario pour la page d'accueil */
        Scenario_LoadCourse.otherScn.inject(nothingFor(30 seconds), ramp(5 users) over (20 seconds))) /** Appel du scénario d'ouverture de cours */

    .protocols(httpProtocol)
}
```

### 21.6.2. Création des scénarios

Maintenant que nous avons créé une simulation, nous pouvons ajouter un nouveau fichier **.scala** dans le répertoire de notre suite de tests, qui contiendra les instructions du scénario.

```
package advanced

import io.gatling.core.scenario.Simulation
import io.gatling.core.Predef._
import io.gatling.http.Predef._
import scala.concurrent.duration._
import bootstrap._
import assertions._
import Headers._

object Scenario_HomePage {

    val scn = scenario("Chargement de la page Moodle DEV")
    .group("Chargement de la page Moodle DEV"){
```

```

    /** -----requêtes----- */
    }
}

```

- Le nom de l'objet (**Scenario\_HomePage**) est utilisé pour appeler le scénario depuis la simulation.
- Le nom du scénario est placé dans une variable (ici scn).
  - Le scénario sera identifié dans le rapport généré par Gatling.
- Quand le chargement d'une page est lié à plusieurs requêtes, celles-ci peuvent être groupées via l'instruction **.group('name')**.
  - Cela permet d'obtenir les temps de chargement pour le groupe réalisé, et non pas seulement pour les sous-requêtes.
  - Il est possible de créer plusieurs groupes dans un scénario.
- Les requêtes peuvent être copiées depuis un fichier de test généré avec l'enregistreur Gatling (chapitre 13.2.)

## 21.7. Contrôles

L'outil Gatling n'utilise pas un navigateur Web pour réaliser les requêtes pour des questions de rapidité. Cette situation ne nous permet pas de voir si les bonnes pages sont chargées. Un moyen pour s'en assurer est de contrôler qu'il n'y a pas de redirections automatiques.

Par exemple, l'ajout d'un fichier dans un cours Moodle où nous ne sommes pas inscrits provoque une redirection vers la page d'accueil du site Moodle.

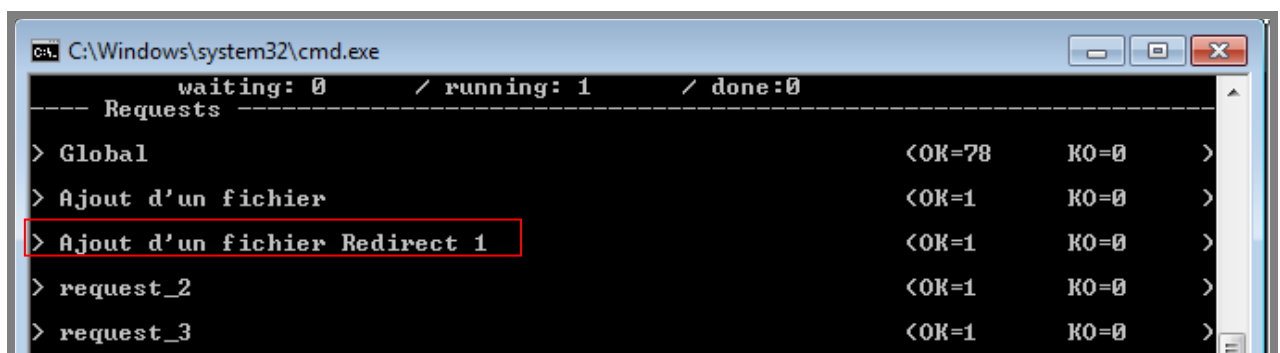


Figure 57 - Ajout d'un fichier sans connexion à Moodle – erreur (screenshot - Gatling)

> Ajout d'un fichier	<OK=1	KO=0	>
> request_465	<OK=1	KO=0	>
> request_466	<OK=1	KO=0	>
> request_468	<OK=1	KO=0	>

Figure 58 - Ajout d'un fichier en tant que professeur (screenshot - Gatling)

## 21.8. Rapports

Les rapports générés lors du lancement de la suite de tests ([[gatling\\_install](#)]/result) sont enregistrés sous forme de fichier html.

Les requêtes des scénarios utilisateur y sont regroupées, ce qui nous permet de visualiser les performances des scénarios et des sous-requêtes (en dépliant le groupe avec la flèche située à gauche du scénario).

STATISTICS (Click here to show more)		Expand all groups   Collapse all groups									
Requests ^	Executions				Response Time (ms)						
	Total	OK	KO	% KO	Min	Max	Mean	Std Dev	95th pct	99th pct	Req/s
Global Information	1285	1285	0	0 %	40	2380	75	151	110	530	17
▸ Chargement de la page Moodle DEV	10	10	0	0 %	14500	15530	14661	294	15530	15530	0
▸ Page économie et services	5	5	0	0 %	6100	6460	6214	126	6460	6460	0
▸ Cours informatique de gestion	5	5	0	0 %	16340	16650	16464	107	16650	16650	0
▸ Ouverture du cours tb-fx	5	5	0	0 %	2240	2900	2530	232	2900	2900	0

Figure 59 Résultats bruts Gatling (screenshot)

Le graphique suivant nous présente le chargement des utilisateurs virtuels (threads). Pour cet exemple, nous avons déployé 10 et 5 threads.

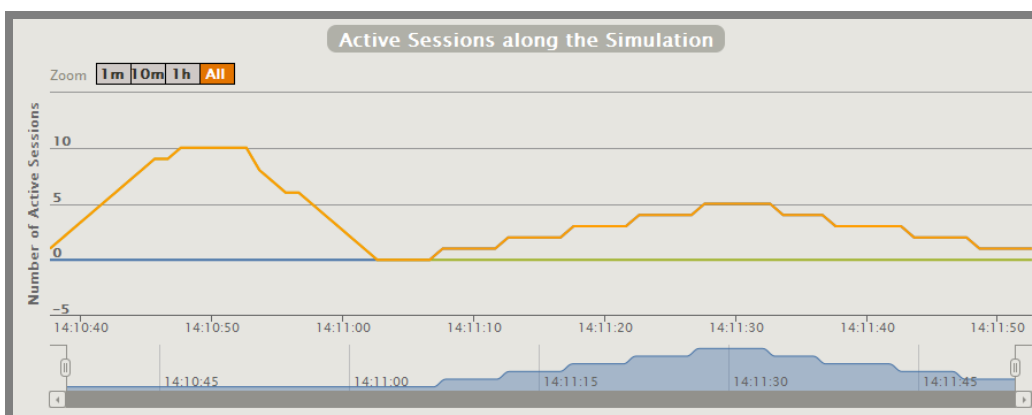


Figure 60 Graphique threads Gatling (screenshot)

Ci-dessous, nous visualisons les performances d'un scénario en fonction des threads (ici

pour le scénario « ouverture cours Tb\_Fx »). La courbe bleue présente les performances du scénario, et la courbe orange les threads déployés. La courbe bleue (environ 3 secondes) correspond à la valeur médiane présentée sur la figure 59. La courbe bleue ne démarre pas au début car d'autres scénarios ont été réalisés précédemment (ouverture de la page Moodle et login).

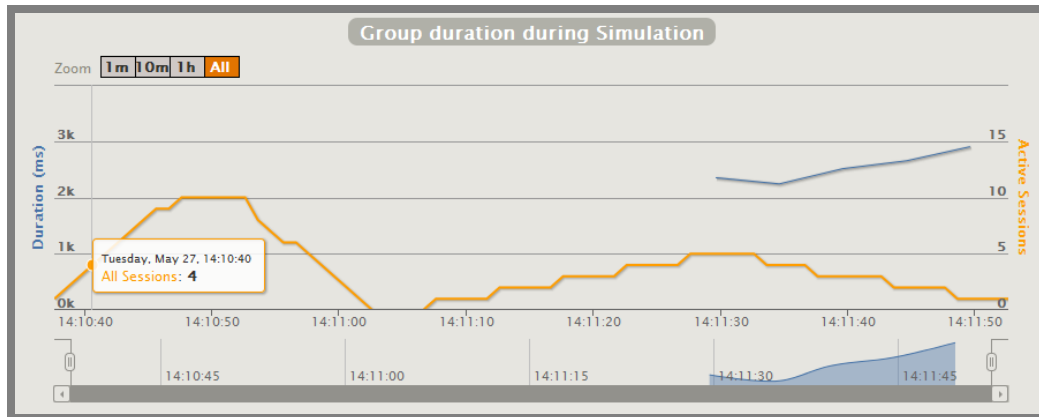


Figure 61 Graphique temps requêtes Gatling (screenshot)

## 21.9. Analyse de l'outil Gatling

### 21.9.1. Enregistrement des tests

Le module fourni par Gatling pour enregistrer les tests nous permet de choisir la destination du fichier de test, ou encore le port utilisé par le proxy. Choses qui n'étaient pas possibles avec l'outil Funkload, qui utilise aussi un proxy.

Nous relevons cependant un petit bémol avec l'enregistrement de tests Gatling. En effet, il n'est pas possible de modifier dynamiquement les tests en cours de création (pas moyen d'ajouter ou de supprimer des requêtes depuis l'enregistreur).

### 21.9.2. Exécution des tests

L'exécution des tests passe par un terminal. Aucune instruction ne doit néanmoins être tapée, car le logiciel nous propose, au lancement de l'application, la liste de tous les tests qui peuvent être exécutés. La génération des rapports est couplée à l'exécution des tests, ce qui rend la procédure beaucoup plus simple que celle de son homologue Funkload.

### 21.9.3. Suite de tests

La suite de tests est créée depuis un fichier de simulation, contenant la liste des scénarios et les détails d'exécutions (utilisateurs virtuels, temps d'exécution, ...). Cette solution est

fonctionnelle, même si moins intuitive qu'une interface graphique (comme le volet de gauche de JMeter).

#### 21.9.4. Résultats obtenus lors du Benchmark

Login	OK
Accès au cours Tb_Fx	OK
Ajout d'une ressource	NO OK

Le Benchmark n'a pas pu être réalisé entièrement avec cet outil, du fait que l'ajout d'une ressource n'a pas été possible. Le chapitre **21.9.7** présente les détails de cette situation.

#### 21.9.5. Qualité des rapports

Les rapports sont générés au format html et peuvent donc être transportés facilement. Le contenu des rapports nous a permis d'analyser correctement les performances du Benchmark :

- Analyses des performances globales, des scénarios et des sous-requêtes
- Visualisation des threads générés durant l'exécution de la suite de tests

#### 21.9.6. Rapidité/Stabilité

- L'outil prend environ 120mb sur le disque, l'installation est quand à elle très simple et rapide.
- Les performances de l'outil sont optimales, l'interface est minimaliste (terminal).
- L'outil est très stable, et nous n'avons pas constaté de problème lors de l'utilisation des fonctionnalités du logiciel.

#### 21.9.7. Problèmes liés à l'ajout d'une ressource

Cependant, nous avons constaté plusieurs problèmes liés au scénario d'ajout de ressources, avec les outils de test de performances n'automatisant pas un navigateur internet. En effet, les sites modernes comme celui de Moodle utilisent des pages Web dynamiques, dont les interactions ne passent pas forcément par une requête GET ou POST.

Nous verrons que si un clic sur un bouton génère un code sous-jacent, les outils testés ne

passant pas par l'automatisation d'un navigateur, à savoir Funkload et Gatling, sont incapables de l'exécuter.

Pour mieux comprendre ce problème, remarquons que l'ajout d'une ressource sur Moodle passe par l'envoi de paramètres POST à la page <http://cyberlearn-dev.hes-so.ch/course/modedit.php>

La requête en question est appelée lors du clic sur le bouton de validation de création de ressource qui se trouve au fond de la page de création :



**Figure 62 Boutons d'enregistrement (screenshot)**

L'automatisation du scénario d'ajout de ressource passe par la répétition de cette requête (appel de la requête POST = ressource ajoutée). L'appel de la requête en cliquant sur un des boutons d'enregistrement présenté plus haut, provoque l'ajout de la ressource, même si toutes les requêtes suivantes sont annulées.

Temps	Durée	Durée totale	Taille	Méthode	État
14:50:33.146	753 ms	753 ms	424	POST	303
14:50:40.802	n/a ms	n/a ms	unknown	GET	Annulé

**Figure 63 Requête POST vers modedit.php (screenshot)**



Un problème se pose lors de la l'exécution de la requête POST, sans cliquer précédemment sur le bouton de validation (Gatling/Funkload). La page modeedit.php retourne alors une erreur, et n'ajoute pas la ressource demandée.

Nom de l'en-tête de la requête	Valeur de l'en-tête de la requête	Nom du paramètre Post	Valeur du paramètre Post
Host	cyberlearn-dev.hes-so.ch	course	4861
User-Agent	Mozilla/5.0 (Windows NT 6.1; rv:29.0) Gecko/20100101 Firefox/29.0	coursemodule	
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8	section	1
Accept-Language	fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3	module	35
Accept-Encoding	gzip, deflate	modulename	folder
Referer	http://cyberlearn-dev.hes-so.ch/course/modedit.php?ac=...	instance	
Cookie	AWSUSER_ID=awsuser_id1401099239367r6839; AWSSESSID=...	add	folder
Connection	keep-alive	update	0
		return	0
		sr	0
		revision	1
		sesskey	pRvcHpGqEv
		_qf_mod_folder_mod_form	1
		name	test_folder2
		introeditor%5Btext%5D	
		introeditor%5Bformat%5D	1

Figure 64 2ème envoi de la requête POST (screenshot)

Paramètre d'action non valide

[Plus d'informations sur cette erreur](#)

**Debug info:**  
 Error code: invalidaction  
**Stack trace:**

- line 476 of /lib/setuplib.php: moodle\_exception thrown
- line 234 of /course/modedit.php: call to print\_error()

Figure 65 Erreur générée par modedit.php (screenshot)

Nous constatons donc que le code sous-jacent aux boutons d'enregistrement ne peut pas être contourné, ni d'ailleurs être exécuté par les outils testés utilisant uniquement le protocole http pour exécuter les scénarios.

Nous allons donc nous concentrer dès maintenant sur les logiciels utilisant la technologie d'automatisation de navigateur (Selenium par exemple) de manière à pouvoir tester intégralement la plateforme Moodle.

## 22. Benchmark JMeter + Web Driver Sampler

L'utilisation de l'outil Selenium Web Driver en combinaison avec JMeter a posé quelques problèmes durant le Benchmark. En voici un résumé :

- Impossibilité d'exécuter une suite de tests. En effet, les tests ne peuvent pas être liés entre eux, il n'a donc pas été possible de tester automatiquement le Benchmark.
- Obligation de recompiler la suite de tests et de l'envoyer vers JMeter après chaque modification (procédure fastidieuse).

Nous avons donc décidé de réaliser un autre Benchmark basé sur JMeter et une extension permettant d'intégrer des codes Selenium Web Driver, directement à l'intérieur du logiciel (sans passer par un environnement de développement externe).

Le mode d'exécution des tests est basé sur l'automatisation d'un navigateur (pas de problème avec les pages dynamiques).

### 22.1. Installation de base

L'installation de JMeter est disponible au chapitre 18.1. **Installation de base.**

### 22.2. Installation du Web Driver Sampler

Cette extension de l'outil JMeter nous permet de créer des tests Selenium Web Driver sans passer par une compilation préalable en .jar (cette méthode a été décrite au chapitre 18, **liaison Selenium(Eclipse) + JMeter**).

Une documentation basique de l'extension est disponible ici : <http://jmeterplugins.org/wiki/WebDriverSampler/>. La procédure présentée ci-après est en partie basée sur ce site.

- L'installation de l'extension commence par le téléchargement du Web Driver ici : (JMeterPlugins-WebDriver-1.1.3.zip): <http://jmeter-plugins.org/downloads/all/>
- Décompresser l'archive dans le répertoire de Jmeter.

### 22.3. Installation des plugins JMeter

Les plugins JMeter permettent de créer des synthèses avancées des résultats obtenus durant les tests (graphiques relatifs aux utilisateurs virtuels ou aux performances des tests).

- Télécharger les plugins JMeter ici (JMeterPlugins-Standard-1.1.3.zip) : <http://jmeter-plugins.org/downloads/all/>
- Décompresser l'archive dans le répertoire de Jmeter.

### 22.4. Création de la structure

Nous allons ici créer la structure qui sera utilisée pour implémenter les scénarios du Benchmark et la suite de tes dans l'outil JMeter.

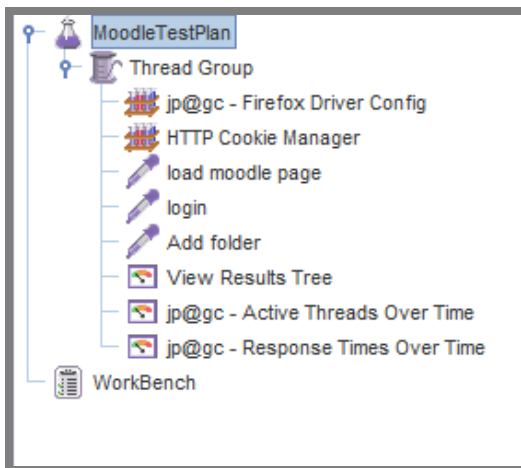
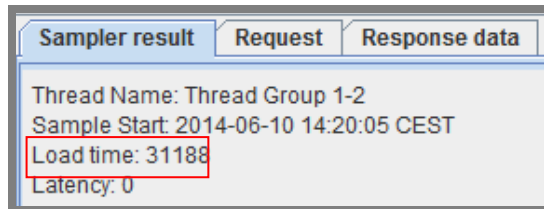


Figure 66 JMeter structure (capture d'écran)

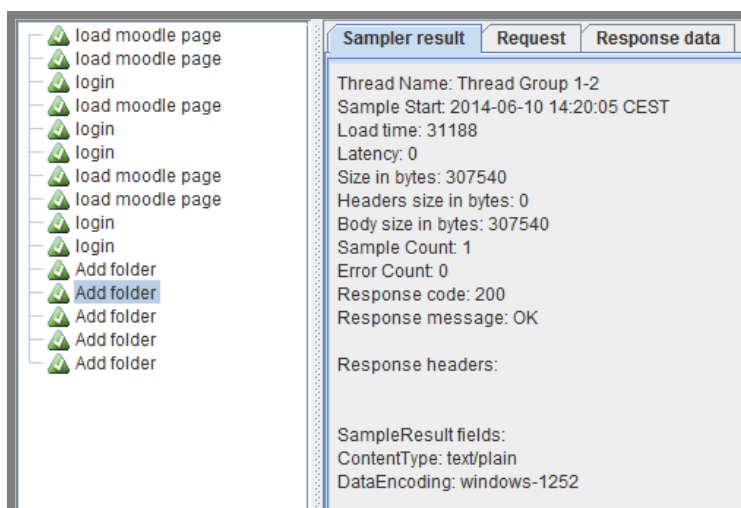
- Créer un Thread Group (*add – threads – Thread Group*)
  - Permet de définir le nombre d'instances de la suite de test qui seront exécutées en parallèle.
- Créer un objet Firefox Driver Config (*add – Config element – Firefox Driver Config*)
  - Cet objet permet de réaliser des tests en passant par le navigateur Firefox.
- Créer un cookie Manager (*add – Config element – http cookie manager*)
  - Permet de gérer les cookies générés par le site Moodle.

- Créer le/les Web Driver Sampler (*add – Sampler – Web driver sampler*)
  - Cet outil permet d'exécuter des scripts Selenium Web Driver (Javascript) sur un navigateur.
  - Un Web Driver Sampler génère au maximum une information de performance.



**Figure 67 JMeter sampler result (screenshot)**

- Chaque mesure de performance doit donc être assignée à un Web Driver Sampler.
- Créer un objet View Results Tree
  - Permet d'afficher l'historique de toutes les requêtes exécutées. Cet élément est aussi le débogueur pour les tests créés, car il affiche les erreurs et son descriptif si un test ne s'est pas exécuté correctement.



**Figure 68 View result tree (screenshot - JMeter)**

- Créer un objet Active Threads Over Time
  - Permet d'afficher sous forme de graphique, le nombre de threads actifs, à chaque instant de la simulation (le nombre de thread à atteindre est défini dans l'objet Thread Group)



**Figure 69 JMeter Threads over time**

- Créer un objet Response Times Over Time
  - Nous permet de comparer les différentes performances obtenues par chaque scénario.
    - Chaque scénario est représenté avec une couleur différente (Load Moodle page, login, add folder).
    - L'onglet Settings nous permet de réaliser des agrégats des performances (pour obtenir une courbe unique).

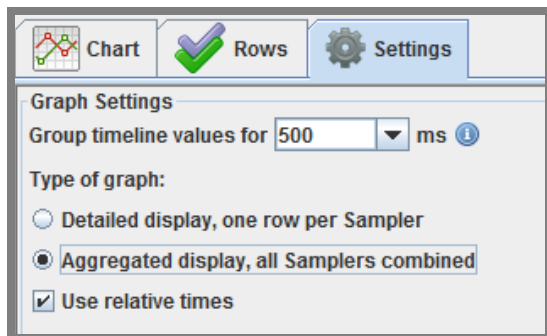


Figure 70 JMeter aggregated display (screenshot)



Figure 71 JMeter Response time (screenshot)

## 22.5. Edition du code

### 22.5.1. Navigation

La ligne suivante permet d'atteindre une certaine URL depuis le script de test (lancement physique du navigateur Firefox).

```
WDS.browser.get('http://cyberlearn-dev.hes-so.ch/')
```

### 22.5.2. Analyse des performances

Les deux lignes qui suivent permettent de mesurer les performances des requêtes qui sont situées entre les deux instructions. Le temps nécessaire pour exécuter les commandes qui seront insérées entre ces deux lignes sera assigné au **Web Driver Sampler** actuel (load time).

Rappelons qu'une seule donnée de performance ne peut être assignée au sampler. Pour réaliser plusieurs mesures, il faudra créer un nouveau Sampler contenant aussi les deux instructions présentées ci-dessous.

```
WDS.sampleResult.sampleStart()  
WDS.sampleResult.sampleEnd()
```

### 22.5.3. Clic sur un bouton/lien

Les commandes qui suivent permettent de localiser un élément (ici un bouton) puis de le sauvegarder dans une variable (connect) pour simuler un clic sur celui-ci.

Il existe plusieurs manières pour localiser un élément :

- **By.id** : Cette méthode permet de sélectionner l'élément en fonction de la valeur id qui lui a été attribuée dans le code. Ce système ne fonctionne que si un id unique est présent.
- **By.xpath** : Cette deuxième méthode utilise une requête XPath pour repérer l'élément recherché. L'addon FireFox **Firepath** permet de générer de telles requêtes.
- **By.cssSelector** : Utilise une requête CSS pour localiser l'élément recherché. Cette méthode peut aussi être réalisée avec l'addon **Firepath** permettant de générer une requête CSS.

Une description de l'addon Firepath est disponible au chapitre 23.

```
var connect = WDS.browser.findElement(pkg.By.id('loginbtn'))  
connect.click()
```

### 22.5.4. Remplir un champ

Pour être en mesure de remplir un champ, il est tout d'abord nécessaire de le repérer dans la page via une des méthodes énumérées plus haut. Puis, il est possible de cliquer dessus pour être sûr qu'il soit bien sélectionné et de vider son contenu avant d'écrire de nouvelles informations.

```
var user = WDS.browser.findElement(pkg.By.id('username')) //find  
user.click() //click  
user.clear() //empty
```

```
user.sendKeys(['tb_fx_user']) //fill
```

### 22.5.5. Valider une alerte javascript

Si une alerte JavaScript est générée par l'application web, il ne sera pas possible de l'atteindre avec une requête standard.

La ligne suivante permet de valider une alerte JavaScript.

```
WDS.browser.switchTo().alert().accept() //Handle Alert
```

### 22.5.6. Navigation entre les Frames

Les frames sont des pages imbriquées dans d'autres pages. Elles sont présentes sur le site Moodle, notamment lors de l'ajout de ressources (module d'édition de texte).

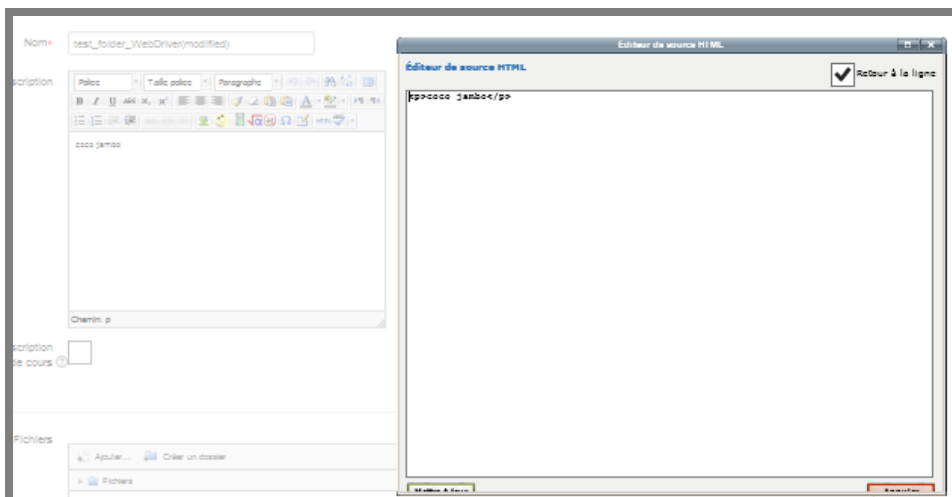


Figure 72 Exemple de Iframe Moodle

La sélection d'éléments sur une frame (page imbriquée dans une autre page) nécessite de réaliser les actions suivantes :

- sélectionner la frame de destination, via une requête Xpath permettant de récupérer l'élément IFrame en fonction de l'id de l'élément.

```
WDS.browser.switchTo().frame(WDS.browser.findElement(pkg.By.xpath("//  
iframe[@id='mce_33_ifr']")))
```

- Réaliser les actions voulues sur cette page
- Revenir sur la frame par défaut.



```
WDS.browser.switchTo().defaultContent();
```

## 22.6. Création d'un test

Pour créer un nouveau test, il suffit de créer un nouveau Web Driver Sampler (voir chapitre 22.4) dans le panneau de gauche de JMeter. Y insérer ensuite le code voulu (la syntaxe est détaillée au chapitre 22.5).

## 22.7. Exécution d'un test

Pour exécuter un test, utiliser le bouton de lancement vert situé en haut de l'écran JMeter. Précisons que JMeter va ensuite exécuter tous les tests présents dans le volet de gauche. Pour exécuter un seul test, il faudra désactiver tous les autres au moyen du raccourci **Ctrl+T**, ou alors via le bouton droit de la souris et l'instruction Toggle.



Figure 73 Bouton de lancement JMeter (screenshot)

## 22.8. Création de la suite de tests

Nous allons maintenant comprendre comment les scénarios créés avec JMeter (Web Driver Sampler) interagissent entre eux.

Au lancement de la suite avec le bouton de lancement, JMeter exécute les scénarios à la suite en partant de celui tout en haut.

Les données du navigateur sont passées d'un scénario à l'autre (cookies, sessions) mais pas les variables définies dans les scénarios. Cela veut dire que le scénario 2 n'aura pas accès aux variables du scénario 1.

## 22.9. Génération des rapports

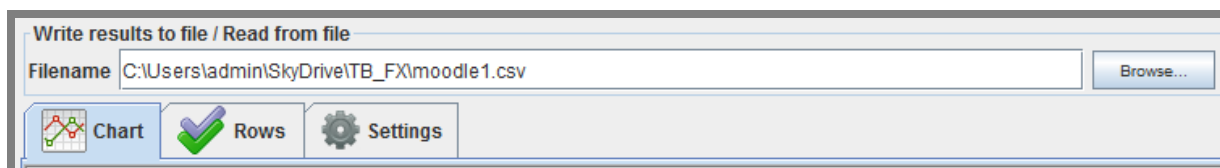
Les rapports sont générés automatiquement dans les trois éléments d'analyses créés dans la structure JMeter

- View result tree
- Active threads over time
- Response times over time

Utiliser l'icône ci-dessous pour supprimer les résultats enregistrés



Pour exporter les résultats des tests, indiquer un nom de fichier dans la barre **filename**. Il sera automatiquement créé s'il n'existe pas. Chaque nouvelle requête y sera intégrée. Pour charger le contenu du fichier, il suffit de frapper la touche Enter sur la zone de saisie.



## 22.10. Analyse de l'outil JMeter + Web Driver Sampler

### 22.10.1. Enregistrement des tests

Cette solution est la seule qui ne propose pas de module spécifique pour enregistrer automatiquement les tests du Benchmark. Néanmoins, vu la complexité des scénarios à créer (navigation entre les frames par exemple), la création manuelle des requêtes ne pose pas de problèmes particuliers. De plus, l'outil Firepath décrit au chapitre 23 nous permet de repérer rapidement les éléments sur une page.

### 22.10.2. Exécution des tests

L'exécution des tests est très simple, l'affichage des éventuelles erreurs (View result tree) est précis et nous permet de repérer facilement ce qui pose problème.

### 22.10.3. Suite de tests

L'outil JMeter propose sans doute le meilleur moyen pour créer une suite de tests par rapports aux autres outils testés. En effet, les scénarios créés dans le volet de gauche sont automatiquement exécutés les uns à la suite des autres. De plus, il est aisé de désactiver momentanément un scénario que nous ne souhaitons pas lancer avec la suite de tests.

### 22.10.4. Résultats obtenus lors du Benchmark

Login	OK
Accès au cours TB_FX	OK
Ajout d'un répertoire	OK

L'outil nous a permis de créer et d'exécuter l'entièreté des scénarios du Benchmark.

### 22.10.5. Stabilité/rapidité

Voir chapitre 19.6.5

### 22.10.6. Qualité des rapports

Voir chapitre 19.6.6

## 23. Nouvelle analyse des outils

### 23.1. Introduction

Nous avons décidé d'introduire un logiciel payant dans notre sélection d'outils, de manière à voir si celui-ci répondrait mieux aux besoins de ce travail. En effet, quatre des cinq logiciels testés ont échoué lors du Benchmark.

- Gatling et FunkLoad utilisent des requêtes GET et POST, qui ne permettent pas de simuler les clics sur les boutons par exemple. Cette situation a amené le Benchmark de ces outils dans une impasse (impossible de simuler l'ajout de ressources sans le code appelé lors du clic sur le bouton)
- Le logiciel TestMaker n'est pas stable, et n'est pas parvenu à réaliser les tests du Benchmark.
- La combinaison JMeter+Selenium Web Driver(Eclipse) ne nous a pas permis de créer une suite de tests automatisée ainsi que de générer des informations de performances.

### 23.2. Telerik Test Studio

#### 23.2.1. Description

Développé par la société Telerik, cet applicatif est une solution de test tout en un, basé sur l'automatisation de navigateurs. Le logiciel se veut intuitif, flexible et abordable. Les différents modules d'enregistrement, d'exécution ou d'analyse des tests sont directement implémentés au sein de l'interface Test Studio.



Figure 74 Logo TestStudio ([www.telerik.com](http://www.telerik.com))

### 23.2.2. Versions

Le produit se décline en trois versions :

- La version « Functional » permet de réaliser des tests fonctionnels, mais sans données de performances.
  - 169\$/mois – 2499\$/licence définitive
- La version « Load » est utilisée pour réaliser des tests de performance, sans offrir la possibilité d'automatiser des scénarios utilisateur.
  - 79\$/mois – 999\$/licence définitive
- La version « Ultimate » lie les tests fonctionnels aux tests de performance.
  - 199\$/mois – 2999\$/licence définitive

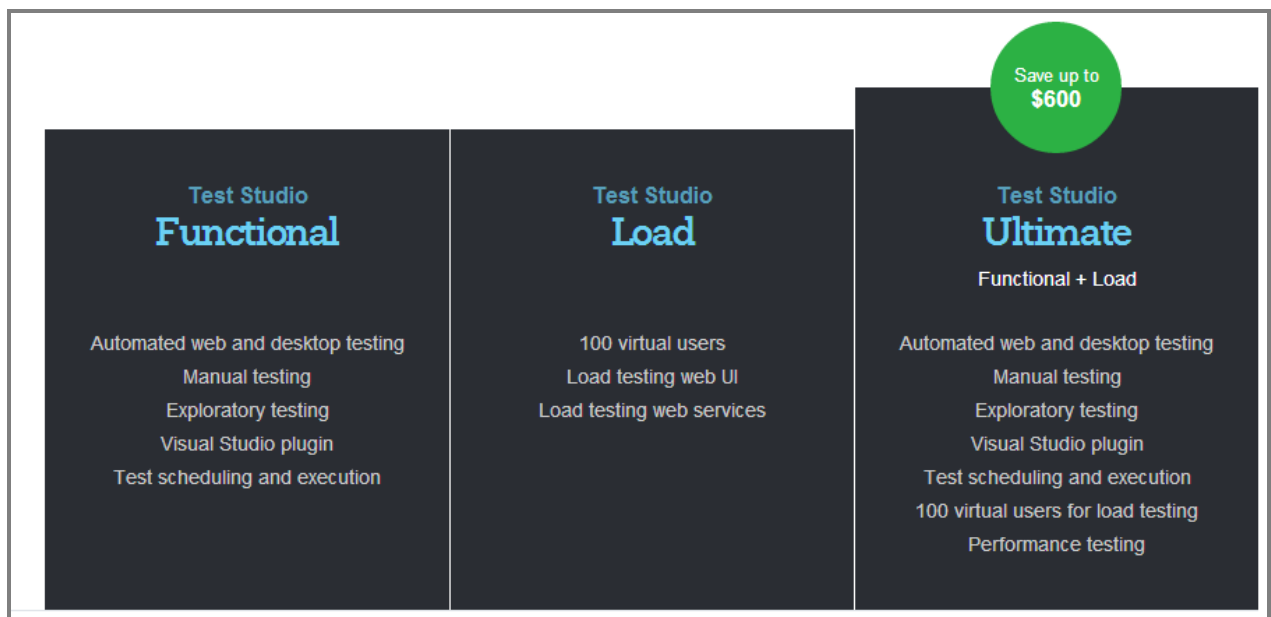


Figure 75 Différentes versions de Test Studio

### 23.3. Analyse théorique des capacités

Functional testing	Oui, mais disponible seulement avec la version Functional et Ultimate.
Performance testing	Oui, avec la version Load ou Ultimate.
Rapports	Le logiciel permet de générer des rapports basés sur les performances globales et celles

	des scénarios.
Licence	Payware.
Dernière mise à jour	2013
Site internet	<a href="http://www.telerik.com/teststudio">http://www.telerik.com/teststudio</a>

Les données théoriques, les procédures d'installation et de gestion des tests qui seront présentées sont tirées du site officiel de Test Studio : <http://www.telerik.com/teststudio> et de <http://docs.telerik.com/teststudio>

## 24. Benchmark Telerik Test Studio

Nous allons maintenant passer au dernier Benchmark de ce travail qui concerne l'outil payant Telerik Test Studio. Cet outil est basé sur l'automatisation d'un navigateur (Chrome, Firefox ou Internet Explorer). Sur ce point, le logiciel est assez similaire à Selenium.

### 24.1. Installation

Commencer par télécharger l'installateur de Test Studio Ultimate : <http://www.telerik.com/teststudio>

Puis, lancer l'installation et suivre les indications à l'écran :

- Choisir le chemin pour l'installation (environ 150mb requis).
- La version de démonstration est valable 30 jours.
- Quand le logiciel le demande, choisir d'installer la version **Ultimate** (tests d'acceptation et tests de performances).

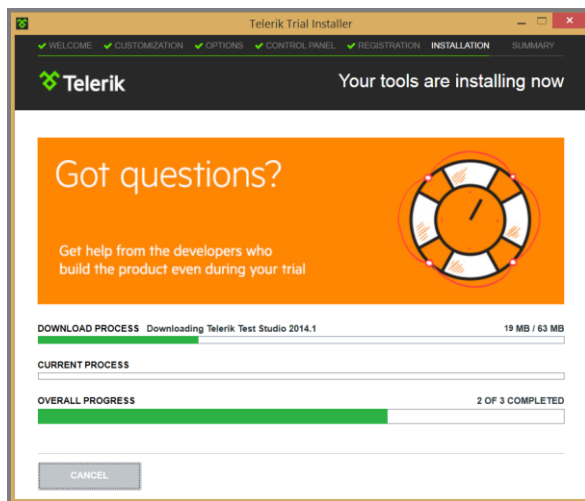


Figure 76 Installation Test Studio (screenshot)

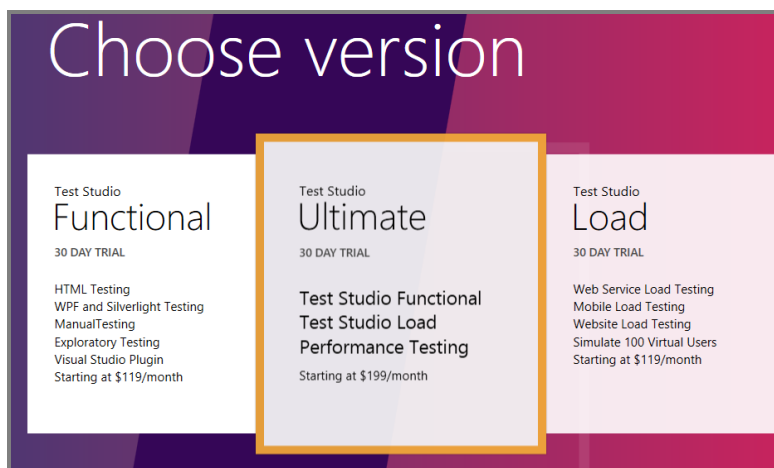


Figure 77 Sélection de la version Test Studio (screenshot)

## 24.2. Enregistrement d'un test

### 24.2.1. Création du projet

Commencer par créer un nouveau projet Test Studio en utilisant le menu supérieur.

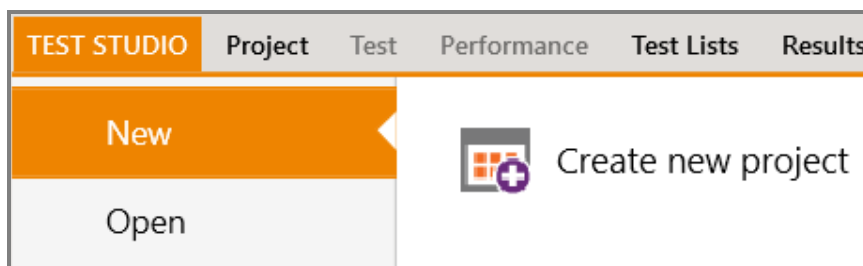
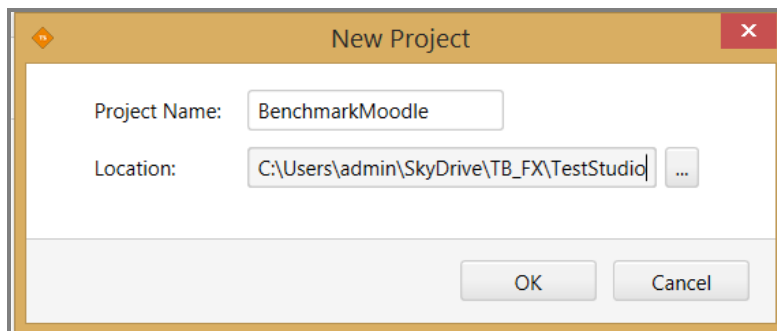


Figure 78 menu projet Test Studio (screenshot)

Définir ensuite un nom et un répertoire pour le projet.



**Figure 79** Création d'un projet Test Studio (screenshot)

### 24.2.2. Création d'un test

Faire un clic droit sur le projet créé, et sélectionner **Add new test** puis le type **Web** pour ajouter un test dans le projet.

Pour modifier un test, cliquer dessus de manière à l'ouvrir dans l'onglet « test » du logiciel Test Studio.

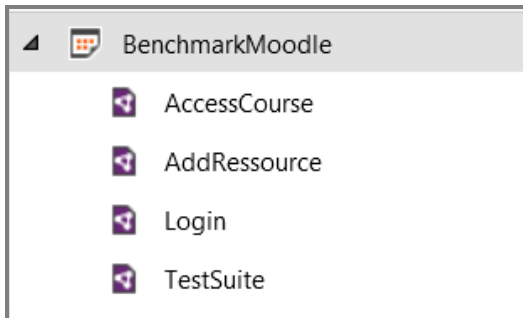


Figure 80 Structure des scénarios Test Studio (screenshot)

Pour ajouter des actions dans le test créé, nous allons enregistrer les actions réalisées dans un navigateur. Utiliser pour cela le bouton **Record**, et sélectionner le navigateur qui sera utilisé pour l'enregistrement du test.

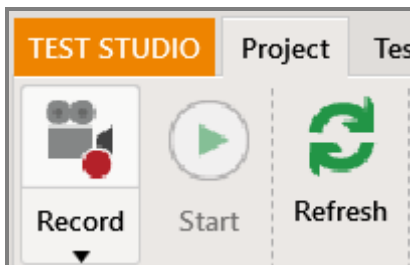


Figure 81 Enregistrement d'un test Test Studio (screenshot)

L'enregistrement terminé, fermer le navigateur ouvert par Test Studio et contrôler que les différentes actions ont été enregistrées dans le test ouvert :

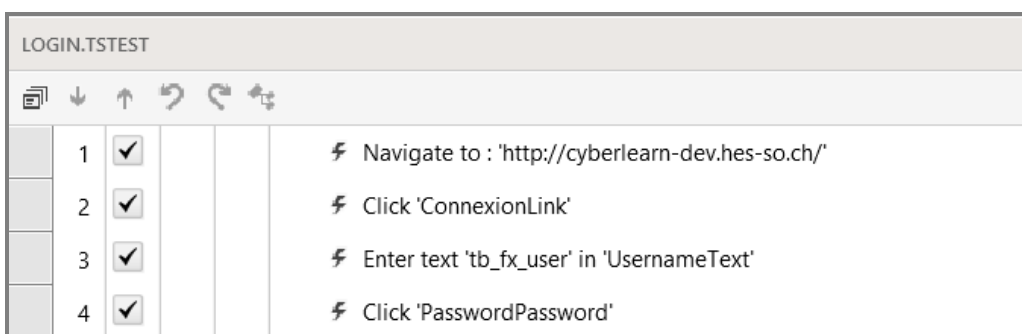


Figure 82 Requête test (screenshot - TestStudio)



Il est possible de modifier les requêtes enregistrées via le module situé en haut à droite. Cela peut être utile pour modifier l'URL de destination d'une requête de navigation par exemple.

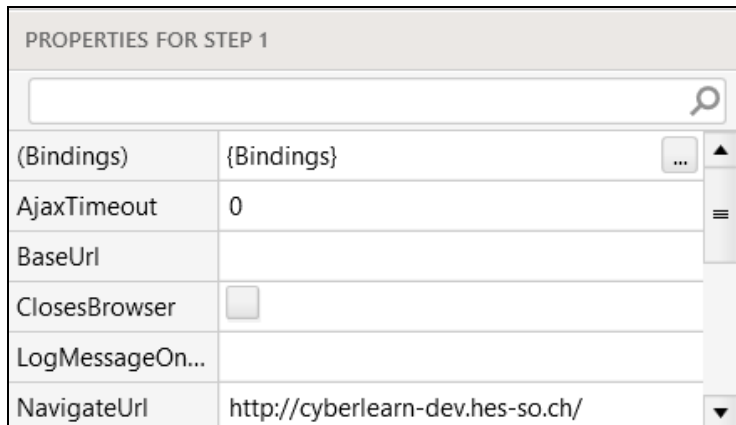


Figure 83 Propriétés d'une requête (screenshot - TestStudio)

### 24.3. Exécution d'un test

Nous allons maintenant exécuter un test précédemment créé. Pour ce faire, commencer par ouvrir le test choisi, cliquer sur **Start** et sélectionner le navigateur qui sera utilisé lors de l'exécution du test.

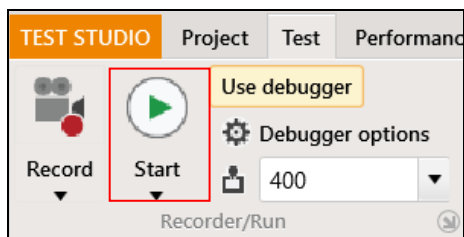



Figure 84 Lancement test (screenshot - TestStudio)

D'une fois que le test a été exécuté, le logiciel présente les résultats sous forme de test d'acceptation (binaire) :

	1	<input checked="" type="checkbox"/>		Navigate to : 'http://cyberlearn-dev.hes-so.ch/'
	2	<input checked="" type="checkbox"/>		Click 'ConnexionLink'
	3	<input checked="" type="checkbox"/>		Enter text 'tb_fx_user' in 'UsernameText'
	4	<input checked="" type="checkbox"/>		Click 'PasswordPassword'
	5	<input checked="" type="checkbox"/>		Enter text 'password' in 'PasswordPassword'
	6	<input checked="" type="checkbox"/>		Click 'LoginbtnSubmit'

Figure 85 Test d'acceptation Test Studio

## 24.4. Création de la suite de tests

Pour créer une suite de tests, commencer par créer un nouveau test dans notre projet. Nous allons maintenant appeler, depuis ce fichier, tous les scénarios qui vont composer notre suite de tests. Pour ce faire, cliquer sur l'icône  située sur le bandeau supérieur. Sélectionner ensuite les scénarios que nous souhaitons ajouter à notre suite de tests.

Nous voyons ici un exemple de suite de tests contenant les trois scénarios du Benchmark :

- Login
- AccessCourse
- AddRessource

	1	<input checked="" type="checkbox"/>		Execute test 'Login.tstest'
	2	<input checked="" type="checkbox"/>		Execute test 'AccessCourse.tstest'
>	3	<input checked="" type="checkbox"/>		Execute test 'AddRessource'

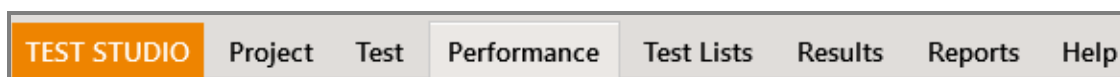
Figure 86 Scénarios TestStudio (screenshot)

La suite de tests peut être exécutée en utilisant le bouton **Start** (tests d'acceptation), ou en générant un rapport de performance comme cela est décrit dans le chapitre suivant.

## 24.5. Génération des rapports

Pour analyser les performances d'un test ou d'une suite de tests, commencer par ouvrir le test ou la suite de tests dont les performances vont être analysées. Puis, cliquer sur l'onglet

**Performance** situé en haut de la fenêtre.



**Figure 87 Bandeau Test Studio**

Si le logiciel le demande, sélectionner un répertoire de destination pour les résultats des performances.

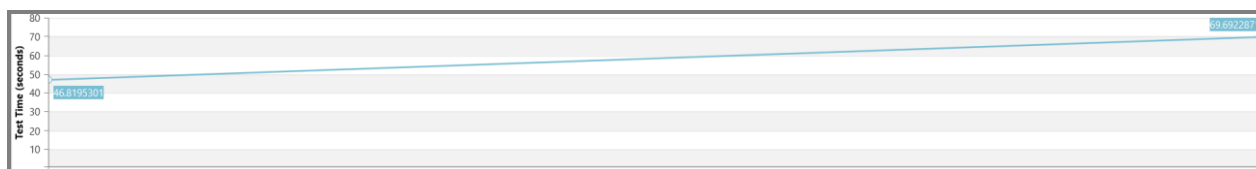
Le bouton **Run** permet de lancer l'analyse des performances sur le test choisi.

L'outil génère ensuite un tableau récapitulatif des performances des différents scénarios/requêtes exécutées.

✓ 1	Execute test 'Login.tstest'		18.234 sec	5.761 sec	12.474 sec
✓ 1	Navigate to : 'http://cyberlearn-dev.hes-so.ch/'		18.729 sec	5.770 sec	12.958 sec
✓ 2	Click 'ConnexionLink'		6.948 sec	5.100 sec	1.849 sec
✓ 3	Enter text 'tb_fx_user' in 'UsernameText'		0.306 sec	0.000 sec	0.306 sec
✓ 4	Click 'PasswordPassword'		0.300 sec	0.000 sec	0.300 sec
✓ 5	Enter text 'password' in 'PasswordPassword'		0.301 sec	0.000 sec	0.301 sec
✓ 6	Click 'LoginbtnSubmit'		0.483 sec	0.399 sec	0.084 sec
✓ 2	Execute test 'AccessCourse.tstest'		8.136 sec	3.072 sec	5.065 sec
✓ 1	Click 'TblLink'		8.371 sec	3.076 sec	5.294 sec
✓ 2	Click 'tblfxTH1Tag'		0.115 sec	0.000 sec	0.115 sec

**Figure 88 Rapport Test Studio**

L'onglet History permet de visualiser, sous forme de graphique, les différents temps d'exécution des instances du test.



**Figure 89 Instances Test Studio**

Il est possible de comparer deux résultats via l'onglet Compare.

1. Execute test 'Login.tstest'	Δ ✓ 2,722 sec
A	18,234 sec
B	15,513 sec

**Figure 90 Comparaison résultats Test Studio**

## 24.6. Analyse de l'outil Test Studio

### 24.6.1. Enregistrement des tests

Test Studio est le seul outil testé qui dispose d'une interface graphique permettant de gérer les tests créés. En effet, l'enregistrement d'un test depuis un navigateur ne retourne pas un script, mais une liste d'actions avec lesquelles il est possible d'interagir. Cette approche facilite la compréhension du test, mais complique les modifications manuelles.

### 24.6.2. Exécution des tests

Il est possible d'exécuter les tests de deux manières différentes. La première permet de contrôler le bon fonctionnement du test (test d'acceptation) et la seconde permet d'évaluer les performances du test.

Dans les deux cas, le logiciel présente des résultats clairs et concis.

### 24.6.3. Suite de tests

La création d'une suite de tests avec Test Studio est simple et rapide. Le problème est que seuls les résultats globaux de la suite de tests sont comparés sous forme de graphique. Les performances des scénarios composant la suite de test sont uniquement présentées sous forme de valeur. Il est donc difficile d'évaluer les résultats globaux des scénarios utilisateur.

### 24.6.4. Résultats obtenus lors du Benchmark

Login	OK
Accès au cours TB_FX	OK
Ajout d'un répertoire	OK

L'outil nous a permis de créer et d'exécuter les différents scénarios, sans pour autant se distinguer du Benchmark JMeter+Web Driver. En effet, nous avons rencontré les mêmes problèmes concernant la gestion des frames par exemple (module d'édition de texte Moodle).

### 24.6.5. Stabilité/rapidité

Nous avons constaté plusieurs plantages durant le Benchmark, surtout lors de la création de tests. L'outil semble relativement lourd d'après l'interface du logiciel, mais il ne faut compter

qu'un espace d'environ 100Mb pour ce logiciel.

#### 24.6.6. Qualité des rapports

Les rapports sont de qualité, même si l'outil ne permet pas de générer des graphiques pour comparer différentes instances d'un scénario. Le logiciel nous fournit par contre des informations de performance sur chaque sous-requête, et pas seulement sur le scénario complet.

### 25. Extention FirePath

Nous allons maintenant présenter l'installation de l'utilisation du plugin FirePath.

FirePath est une extention au navigateur web gratuit Firefox (<http://www.mozilla.org/fr/firefox/new/>).

Elle permet de générer et tester des requêtes CSS ou Xpath, de manière à repérer un élément sur une page Web.

L'utilité de cet outil est importante dans ce projet, car la plupart des logiciel de test repèrent les éléments avec ce type de technologie (JMeter Web Driver, Eclipse Web Driver, TestMaker, ...).

Pour installer le plugin, commencer par le télécharger depuis le site suivant depuis Firefox : <https://addons.mozilla.org/fr/firefox/addon/firepath/>. Il sera alors installé directement sur le navigateur.

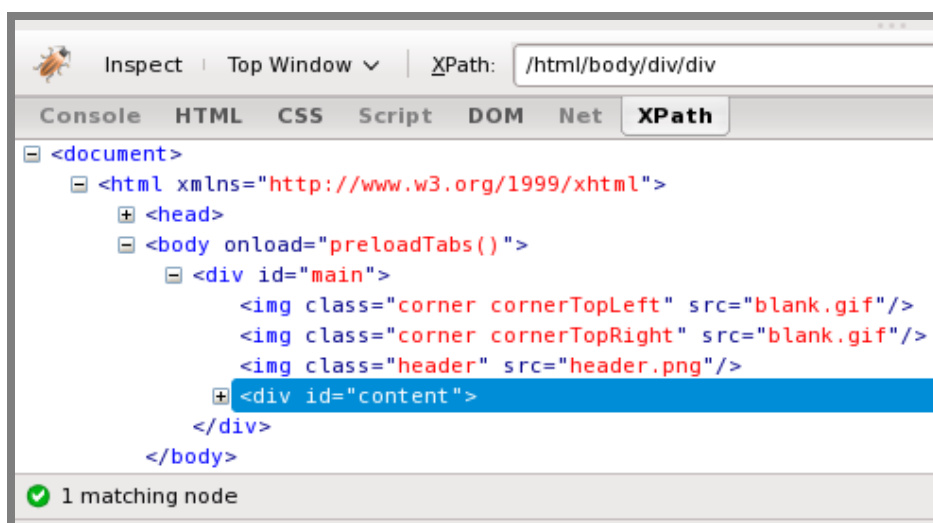


Figure 91 Interface Firepath (addons.mozilla.com)

Sélectionner le type de requête qui sera enregistré par Firepath (1).

L'outil nous propose à ce sujet plusieurs manières pour repérer un élément sur une page :

- **By.id** : Cette méthode permet de sélectionner l'élément en fonction de la valeur id qui lui a été attribuée dans le code. Ce système ne fonctionne que si un id unique est présent.
- **By.cssSelector** : Utilise une requête CSS pour localiser l'élément recherché.
- **By.xpath** : Cette deuxième méthode utilise une requête XPath pour repérer l'élément recherché. Les requêtes Xpath sont plus précises, et permettent par exemple de repérer le dernier élément d'une liste.

Puis utiliser le bouton inspect (2), avant de sélectionner avec le curseur l'élément à repérer sur la page. La requête est alors générée (3).

## 26. Sélection d'un outil

Nous allons tout d'abord réaliser une synthèse des outils testés, avant de déterminer lequel(s) serait potentiellement capables d'implémenter la suite de tests Moodle complète.

### 26.1. Rétrospective des outils testés

#### 26.1.1. PushToTest TestMaker

Nous avons constaté que le logiciel n'était pas stable et que l'interface n'était pas totalement fonctionnelle. De plus, le logiciel TestMaker n'a pas été capable de reproduire les scénarios utilisateurs du Benchmark et d'en tirer des informations de performance.

Cet outil n'est donc pas capable d'implémenter la suite de tests complète.

#### 26.1.2. JMeter + Selenium Web Driver (Eclipse)

Cette combinaison d'outils (JMeter et Eclipse) s'est montrée relativement stable et nous a permis de réaliser la totalité du Benchmark. Par contre, il n'a pas été possible de créer une suite de tests, ni de récupérer les performances individuelles de chaque scénario.

Cet outil n'est donc pas capable d'implémenter la suite de tests complète.

#### 26.1.3. Funkload

Basé sur l'exécution de requêtes GET/POST et non pas sur l'automatisation d'un navigateur (à l'inverse de Selenium Web Driver), l'outil FunkLoad ne nous a pas permis de réaliser une suite de tests pour le Benchmark Moodle. De plus, le problème rencontré avec ce type de

logiciel est qu'il ne simule pas le clic sur les boutons, et les interactions dynamiques (JavaScript notamment). Cette situation a rendu impossible l'ajout de ressources sur le site Moodle, et donc la réalisation complète du Benchmark.

Cet outil n'est donc pas capable d'implémenter la suite de tests complète.

#### **26.1.4. Gatling**

Tout comme l'outil Funkload, Gatling est basé sur l'exécution de requêtes GET/POST. Cet outil nous a permis de réaliser des suites de test fonctionnelles, avec des rapports présentant les résultats de chaque scénario utilisateur, et les détails de chaque sous-requête. Tout comme l'outil Funkload, il ne simule pas le clic sur les boutons, et les interactions dynamiques. Le Benchmark n'a donc pas été réalisé entièrement.

Cet outil n'est donc pas capable d'implémenter la suite de tests complète.

#### **26.1.5. JMeter (Web Driver Sampler)**

Le plugin Selenium Web Driver Sampler présenté comme add-on à l'outil JMeter nous a permis de réaliser l'entièreté du Benchmark. Nous avons été en mesure de créer une suite de tests, et de disposer de rapports pour chaque scénario utilisateur. Le seul bémol de cette solution est que les informations de performance sont relatives aux scénarios utilisateur, et ne permettent pas d'obtenir les performances des sous-requêtes.

Nous en arrivons à la conclusion que JMeter (Web Driver Sampler) peut être utilisé pour implémenter la suite de tests complète.

#### **26.1.6. Test Studio**

Cet outil est le seul logiciel payant que nous avons testé durant ce travail. Les résultats de celui-ci sont assez concluants, nous avons été en mesure de créer une suite de tests, et d'implémenter tous les tests du Benchmark.

Nous pouvons quand même noter que nous avons rencontré les mêmes difficultés qu'avec l'outil JMeter Web Driver (Au niveau de la gestion des frames par exemple).

Cet outil est donc capable d'implémenter la suite de tests complète.

### **26.2. Comparaison entre JMeter et Test Studio**

A ce stade, nous disposons de deux outils capables d'évaluer les performances d'une version de la plateforme Moodle.

D'un côté le logiciel payant **Test Studio**, qui peut être considéré comme une référence dans le domaine des tests Web, et de l'autre, la combinaison de **JMeter** et du plugin Web Driver Sampler (gratuit).

Nous allons ici comparer les deux outils en fonction de plusieurs critères :

- Au niveau de l'enregistrement des tests.
- Au niveau de la suite de tests.
- Au niveau des rapports générés.
- En fonction d'autres paramètres généraux.

### 26.2.1. Enregistrement de tests

JMeter		Test Studio	
Enregistrement manuel avec l'aide du plugin Firefox Firepath.		Enregistrement automatisé avec le module d'enregistrement des actions du navigateur (Chrome,IE,Firefox).	
+	-	+	-
Les modifications sont plus aisées via les scripts Web Driver	Les tests sont plus difficiles à créer (syntaxe)	Création de test rapide et aisée avec l'interface Test Studio.	Les modifications sont plus lentes à réaliser avec l'interface graphique, mais plus intuitives.



### 26.2.2. Suite de tests

JMeter		Test Studio	
Possibilité de créer une suite de tests, l'exécution est couplée avec la génération des rapports de performances.		Possibilité de créer une suite de tests depuis un scénario de base. Les rapports sont générés pour chaque scénario.	
+	-	+	-
Activation et désactivation aisée des scénarios / exécution facile de la suite de tests complète		Activation et désactivation aisée des scénarios (depuis l'éditeur de test) / exécution facile de la suite de tests complète	

### 26.2.3. Rapports

JMeter		Test Studio	
Les rapports sont clairs et précis, mais ne permettent pas de visualiser les performances des sous-requêtes.		Les rapports sont de qualité, mais les comparaisons entre les instances des scénarios ne sont pas suffisantes.	
Graphiques simples et complets.	Les rapports de performances ne fournissent pas d'informations sur les sous-requêtes.	Analyse des scénarios /sous-requêtes.	Comparaison limitées entre les différents threads (2 max).

#### 26.2.4. Généralités

JMeter		Test Studio	
Le logiciel dispose d'une interface claire et pratique.		Test Studio est un logiciel intuitif avant tout, avec une documentation exhaustive et regroupée sur le site officiel.	
+	-	+	-
Gratuit, très stable (pas de plantages constatés)	La documentation est éparpillée, surtout au niveau du sampler Web Driver.	L'interface permet une prise en main rapide du logiciel.	La modification des tests est plus lente à réaliser qu'avec les scripts Web Driver.
Interface très efficace (pas d'onglets, tout est directement accessible)		Documentation complète sur le site officiel.	Stabilité relative, plantages constatés lors de la création d'un test.

#### 26.3. Sélection

Nous allons maintenant sélectionner par élimination l'outil le plus adapté pour tester la plateforme Moodle.

Les deux seuls outils qui ont été en mesure de tester l'entièreté Benchmark, de générer des informations de performance et de réaliser une suite de tests automatisée sont **JMeter + Web Driver Sampler** et **Test Studio**.

Notre décision se base sur les éléments suivants :

- L'objectif principal de ce travail est de comparer les performances de scénarios utilisateur de manière à obtenir des valeurs moyennes. Or le logiciel TestStudio ne synthétise pas correctement les performances des scénarios (pas de comparaisons efficaces entre les différentes instances d'exécution).
- Du fait que l'outil payant **Test Studio** n'amène pas d'améliorations concrètes par rapport à son homologue gratuit, et que l'objectif de ce travail était de trouver avant tout un logiciel gratuit, nous avons décidé de sélectionner l'outil **JMeter + Web Driver Sampler** pour implémenter la suite de tests complète Moodle.

## 27. Définition de la suite de tests Moodle Complète

Maintenant que nous avons choisi l'outil à utiliser pour implémenter la suite de tests complète (JMeter+Web Driver Sampler), nous allons définir ce que contiendra cette suite de tests, avant de l'implémenter dans le logiciel.

Nous voyons ci-dessous les différentes étapes de ce processus de définition (chapitre 8.4) :

- Analyser les besoins de l'application
- Identifier les scénarios
- Définir le plan de test
- Implémenter les tests dans le logiciel
- Exécutez les tests
- Enregistrez les résultats
- Confirmation que les objectifs sont atteints

Notre analyse des besoins de l'application s'est basée sur un fichier contenant tous les tests qui ont été réalisés manuellement par l'administration du système, lors de la migration vers la version actuelle de Moodle. Ce fichier nous a été transmis par la responsable de ce travail.

En fonction du contenu de ce fichier, nous avons déterminé deux grands profils d'utilisateur susceptibles d'utiliser la plateforme : Les **étudiants** et les **enseignants**.

C'est à partir de ces profils que nous avons été en mesure de créer les différents scénarios, et de définir la suite de tests pour les deux profils.

### 27.1. Plan de test : Profil Enseignant

Nous allons maintenant définir les différents scénarios qui seront implémentés pour le profil **Enseignant**.

Nous voyons sur la page suivante l'imbrication des différents scénarios définis pour ce plan de test. Les performances générées par JMeter devront être disponibles pour chacun d'entre eux.

- Le professeur commence par se connecter à la plateforme Moodle Dev via l'utilisateur **tb\_fx\_user**. Puis, il accède au cours **tb\_fx** et active le mode édition.
- Le professeur ajoute alors cinq ressources différentes (File, Etiquette, Devoir, Url, Folder). Il accède ensuite à ces différentes ressources de manière à contrôler la présence des éléments et à analyser les performances de leur chargement.
- Le professeur modifie ensuite les ressources créées (nom, description) avant de les supprimer.

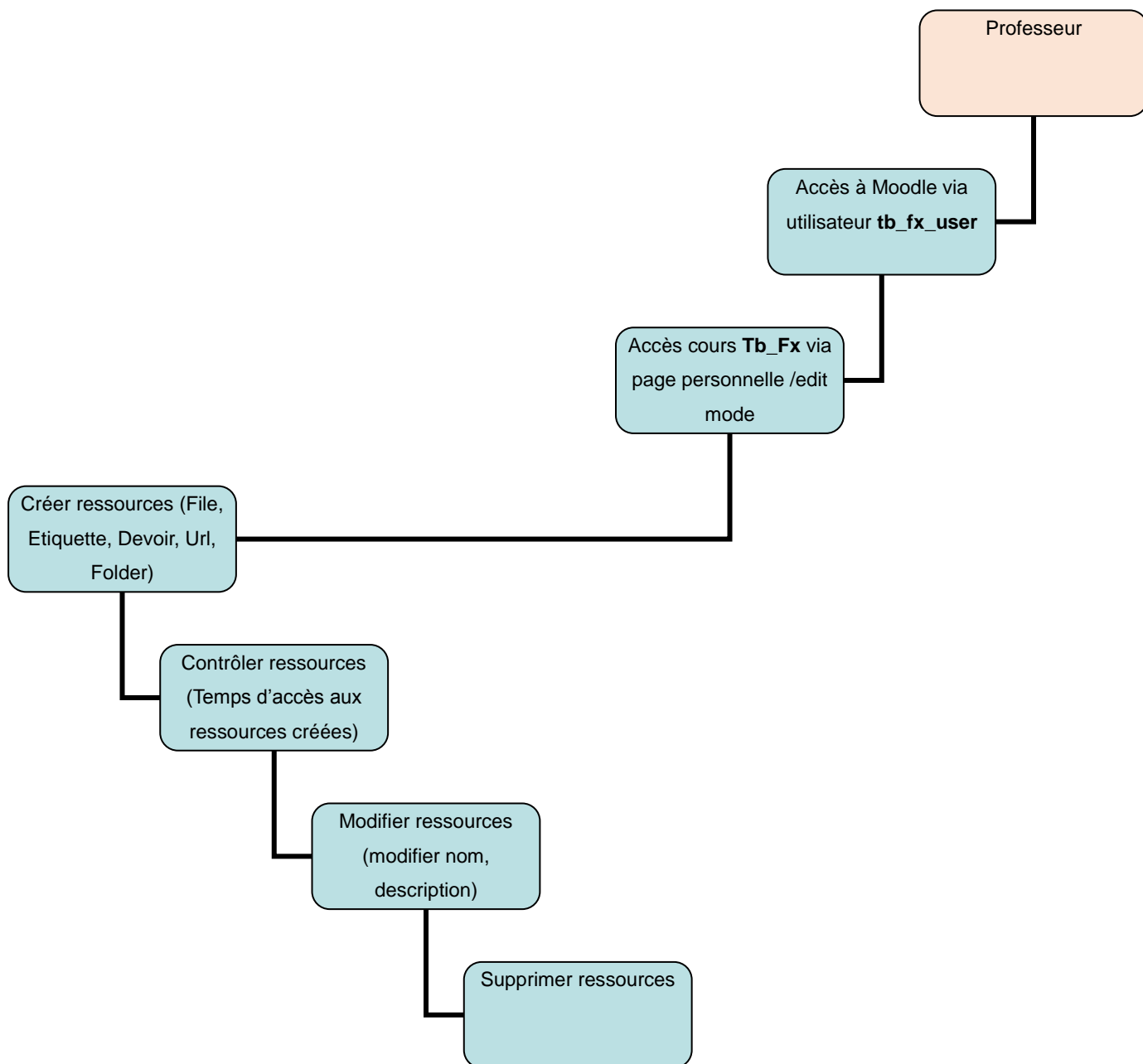


Figure 92 Schéma plan de test - professeur

## 27.2. Plan de test : Profil Etudiant

Nous allons maintenant définir la composition du profil Etudiant.

L'étudiant commence par charger la page d'accueil de **Moodle Dev**, avant de changer la langue de manière à mesurer le temps de chargement de cette fonction. Finalement il accèdera au cours **Tb\_Fx** en utilisant la navigation Moodle.

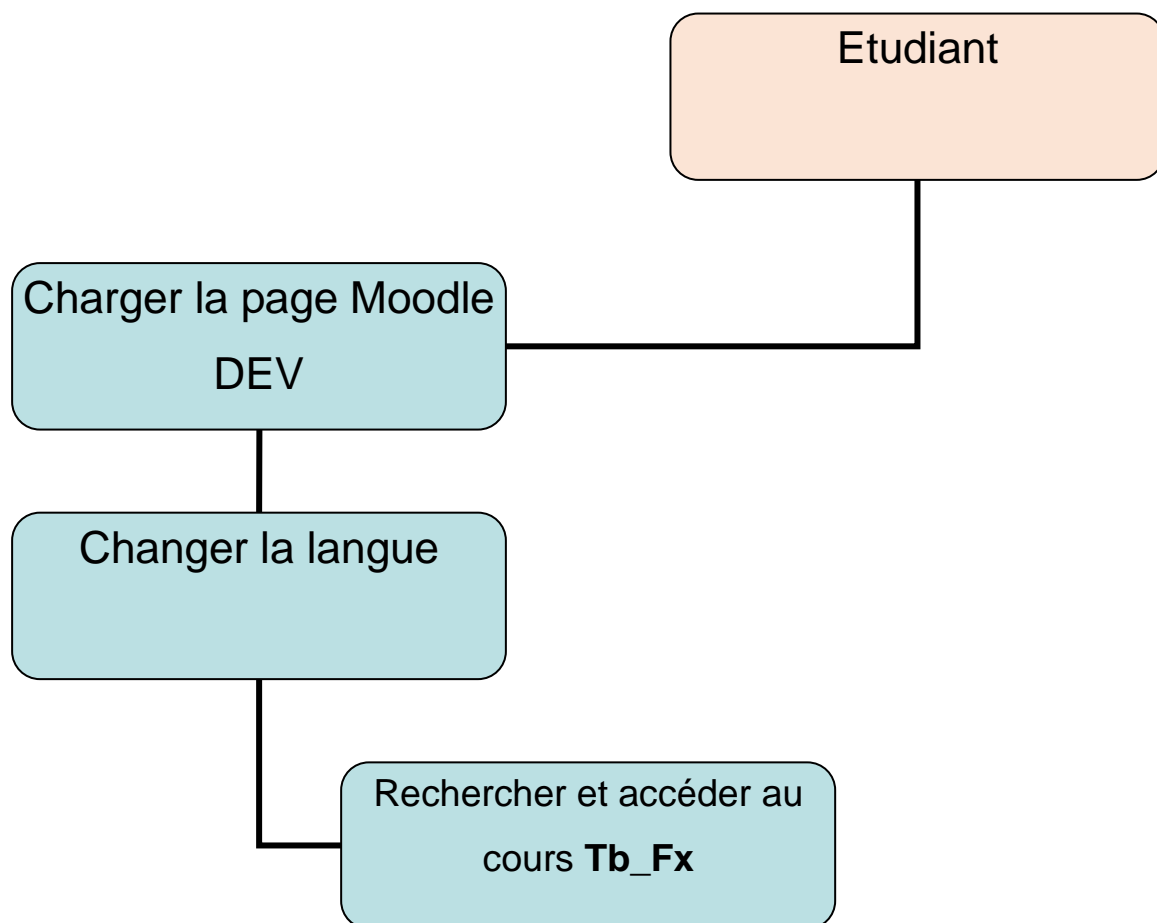


Figure 93 Schéma plan de test - Etudiant

## 28. Implémentation de la suite de tests Moodle complète

Nous avons implémenté la suite de tests définitive dans le logiciel sélectionné après le Benchmark (JMeter + Web Driver Sampler).

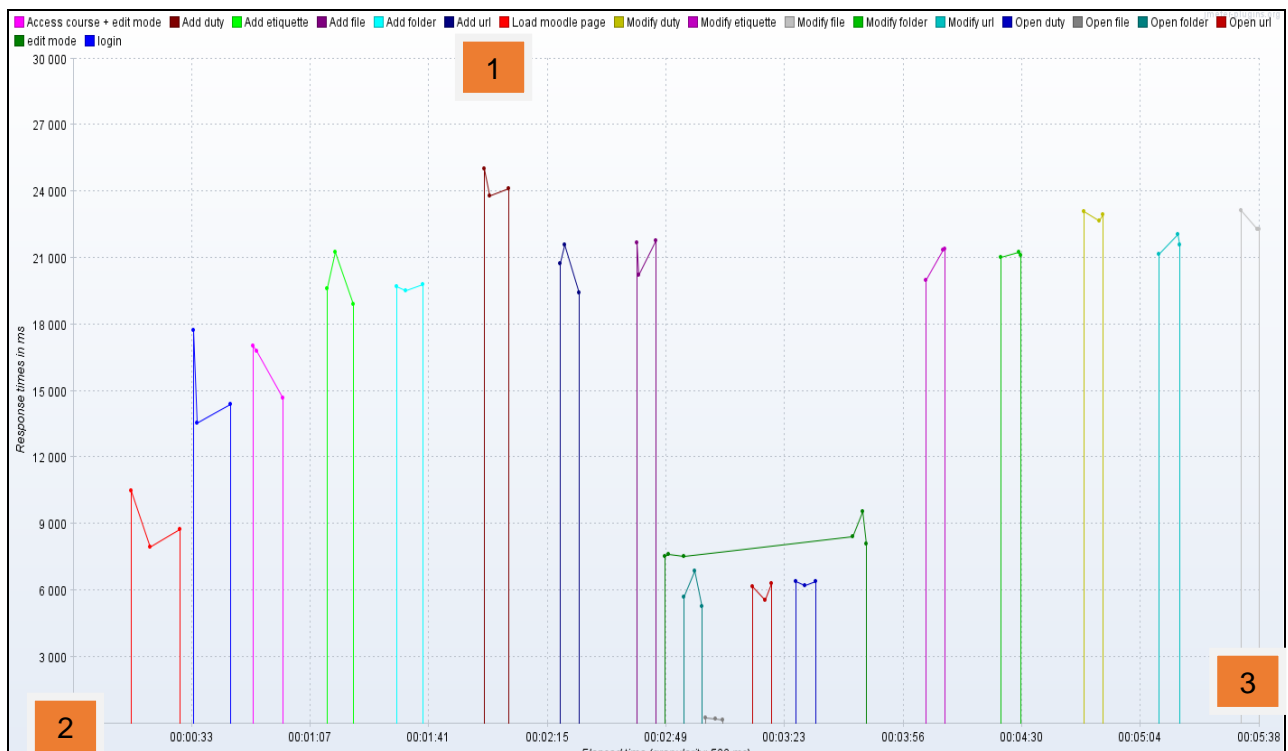
L'implémentation a été réalisée sur le serveur de développement Moodle : <http://cyberlearn-dev.hes-so.ch/>

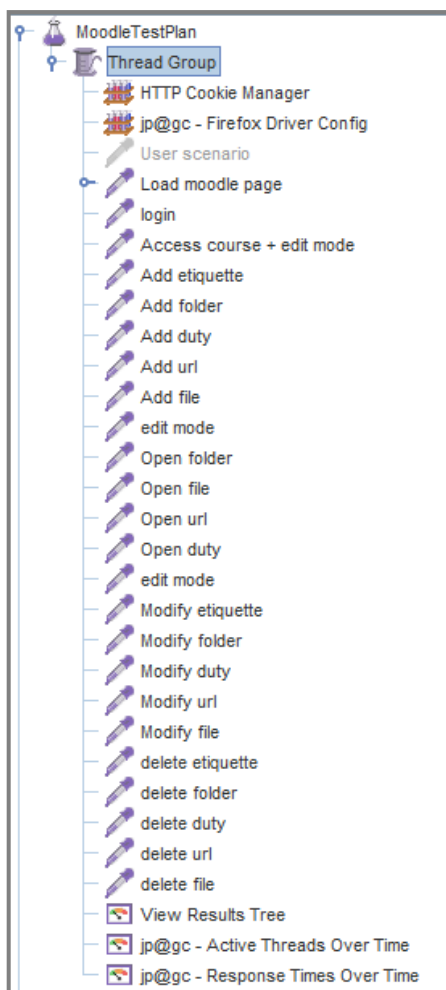
Tous les tests ont été réalisés à partir de la suite de tests définie au chapitre précédent.

### 28.1. Résultats

Tous les tests ont été créés et exécutés avec succès nous permettant d'évaluer les performances de la plateforme Moodle DEV. La capture d'écran ci-dessous présente les résultats générés par JMeter (trois utilisateurs virtuels) :

1. Liste des scénarios utilisateurs
2. Temps de réponse des scénarios
3. Temps total pour exécuter la suite de test (05:38sec)





Nous voyons ci-contre la liste des scénarios qui ont été implémentés avec JMeter. A part le premier scénario, tous correspondent à la suite de test dite « professeur » dont voici les principales étapes :

- Connexion au site (Login)
- Accès au cours (Access course +edit mode)
- Ajout des ressources (Add ...)
- Accès aux ressources (Open ...)
- Modification des ressources (Modify ...)
- Suppression des ressources (Delete ...)

La suite de test dite « Etudiant » est regroupée dans le script « User scenario ».

**Figure 94** Liste des tests JMeter - def (screenshot)

## 28.2. Variables de script

Comme nous le montre la figure précédente, les scénarios utilisateurs ont été exécutés plusieurs fois en parallèle.

Notamment lors de l'ajout des ressources, il est important que les noms de celles-ci soient différents pour chacun des utilisateurs virtuels. Il faut par exemple que le thread 1 crée et modifie des fichiers différents de ceux du thread 2 pour éviter des conflits.

Pour ce faire, une variable relative à la suite de tests a été ajoutée au sommet de chaque script ; elle permet de retrouver l'identifiant du thread en cours (`${__threadNum}`).

Nous disposons aussi d'une autre variable qui a pour but de mémoriser l'identifiant de la frame de description (voir chapitre 22.5.6. **Frame de description**).

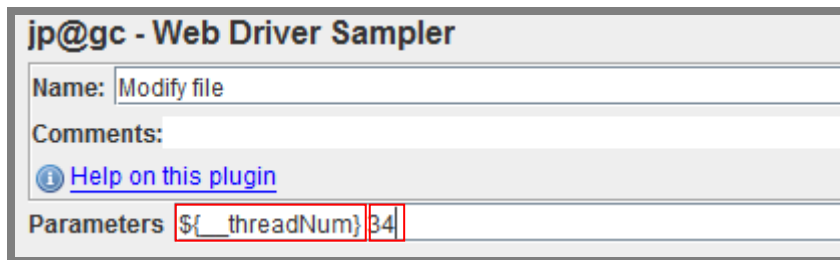


Figure 95 Variables JMeter thread-id (screenshot)

## 28.3. Limitations

### 28.3.1. Contrôle des éléments

Lors de l'exécution de plusieurs threads en parallèle, il arrive que la localisation d'éléments ne fonctionne pas (erreurs visibles dans l'élément Jmeter View Result Tree). Dans ce cas, il est nécessaire d'insérer la ligne de contrôle suivante qui contrôle que l'élément est bien visible.

```
wait.until(ui.ExpectedConditions.presenceOfElementLocated(pkg.By.xpath(".*[@name='repo_upload_file']"))))
```

Il arrive parfois que JMeter ne détecte pas un élément sur la page, cela malgré la commande d'attente. Dans ce dernier cas, la seule solution est de relancer la suite de tests.



### 28.3.2. Frame de description

L'ajout d'une description lors de la création d'un nouvel élément est obligatoire pour la plupart d'entre-eux (étiquette, devoir et répertoire). La modification de la description passe par l'ouverture de l'éditeur de source HTML, car nous n'avons pas été en mesure d'envoyer du texte directement dans le module d'édition (impossible de sélectionner la zone de texte).

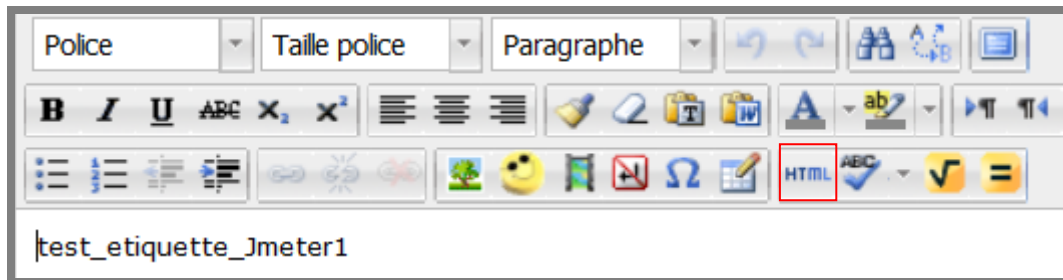


Figure 96 Editeur de texte Moodle (screenshot)

L'éditeur de source HTML est lié à une frame, qui peut être repérée sur la page via l'outil Firepath (chapitre 23). Pour ce faire, il suffit de dérouler la liste située à gauche du module et de sélectionner la frame relative à **source\_editor.htm**.

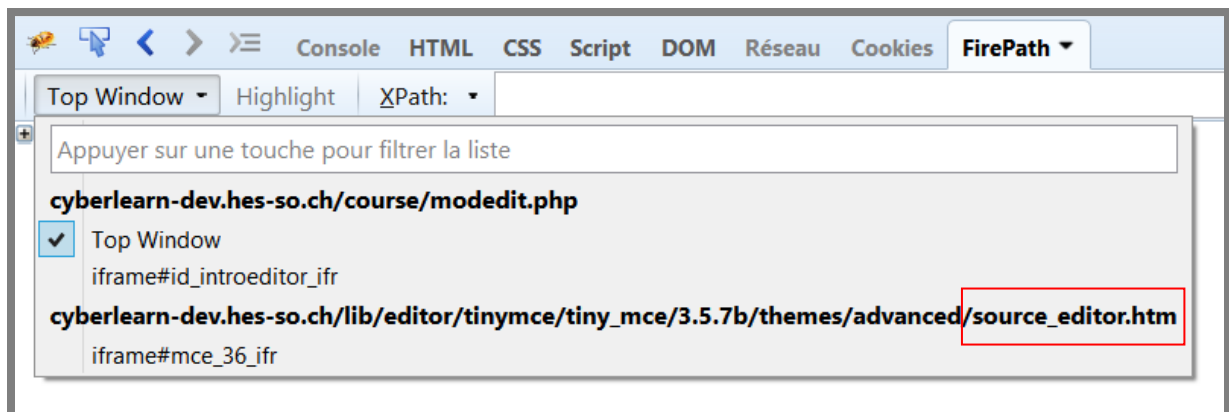


Figure 97 IFrame relative à l'éditeur de code source (screenshot - Firepath)

Il arrive parfois que l'identifiant de cette frame change, ce qui est visible lors de l'exécution de la suite de tests. En effet, nous voyons ci-dessous qu'une erreur est générée depuis le **result tree**, portant sur l'élément **//iframe[@id='mce\_36\_ifr']**.

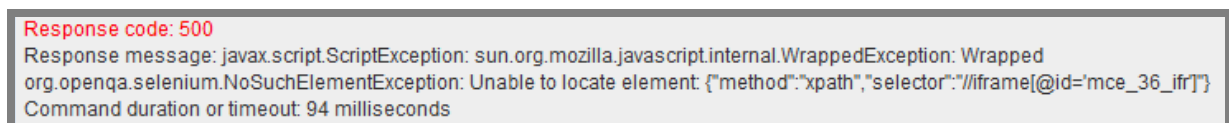


Figure 98 Erreur liée à un mauvais id (screenshot)

Pour remédier à ce problème, il faut alors utiliser l'outil Firepath pour rechercher le nouvel identifiant, et l'insérer dans la variable de script prévu à ce effet :

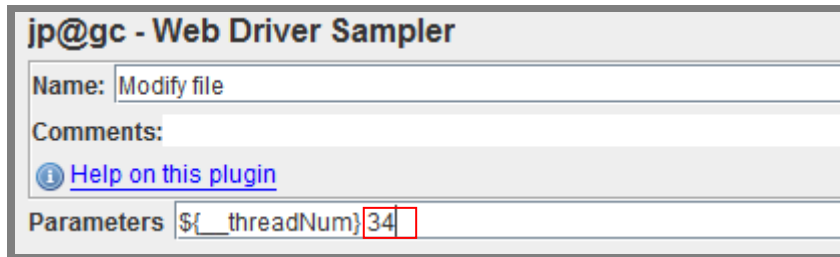


Figure 99 Variables JMeter thread-id (screenshot)

## 28.4. Génération des rapports

La génération des rapports est réalisée automatiquement lors de l'exécution de la suite de test (chapitre 22.7).

- La liste des scénarios réalisés est visible dans le **View result tree**.
- Le graphique des threads est affiché dans l'élément **Active Threads Over Time**.
- Le graphique des temps d'exécution en fonction des threads est disponible dans le **Response Times Over Time**.

Pour obtenir un graphique plus parlant avec une terminaison des lignes de valeur, il est possible de cocher la case « Draw final zeroing lines ».

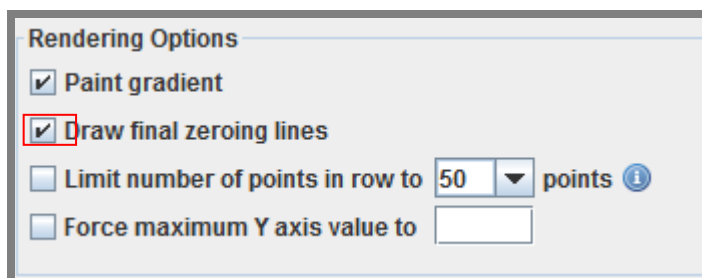


Figure 99 Zeroing lines (JMeter)

## 28.5. Exportation du rapport

L'exportation est automatique, du moment qu'un fichier a été sélectionné dans la barre d'enregistrement située en haut des trois composants JMeter décrits plus haut.

Le fichier sera automatiquement créé s'il n'existe pas. Les rapports peuvent être exportés dans le format .csv ou .jtl (extension propre à JMeter d'après <https://wiki.apache.org/jmeter/JtlFiles> )

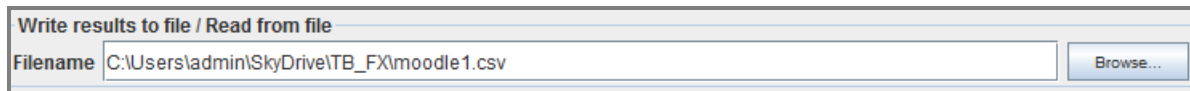


Figure 100 Exportation rapport JMeter (screenshot)

## 29. Implémentation sur le serveur Moodle Old

Nous avons le projet de transposer la suite de tests depuis le serveur de développement vers un serveur de test contenant une ancienne version de l'appliquatif Moodle (Moodle Old), de manière à pouvoir comparer les différences de performances entre ces deux versions.

A cause des importantes différences entre ces deux versions, nous n'avons pas eu le temps d'adapter la suite de test pour ce nouveau système.

## 30. Conclusion

Les objectifs de ce travail étaient de trouver un outil capable de tester automatiquement les performances de processus utilisateurs, réalisés sur une version de la plateforme Moodle implémentée à la HES-SO. En effet, au vu du nombre d'utilisateurs accédant quotidiennement à ce service (plus d'un million de visites en 2013 d'après le rapport d'activités), une baisse des performances liée à l'introduction d'une nouvelle version de l'appliquatif ou à l'intégration d'un développement logiciel n'est guère envisageable.

Pour mener à bien ce travail, nous avons commencé par rechercher plusieurs outils gratuits capables de répondre à la problématique posée. Nous avons sélectionné cinq outils, dont nous avons évalué les capacités en fonction de différents critères (suite de tests, rapports générés, ...). Parmi ces outils gratuits, un seul a été capable de fournir des données de performances adéquates (JMeter). En effet, les interactions avec les pages dynamiques et la suite de tests automatisée ont posé des problèmes à la plupart des logiciels testés.

Afin de contourner cette situation et de mieux appréhender les outils disponibles, nous avons décidé de nous tourner vers le marché des outils payants, de manière à déterminer si ceux-ci

seraient mieux à même de répondre aux besoins requis par ce travail. Nous avons testé un outil payant (Test Studio) dont les performances se sont avérées être relativement similaires au logiciel gratuit sélectionné (JMeter).

Du fait que le but de ce projet était avant tout de trouver un outil d'analyse gratuit, et que la comparaison entre les instances de test fournie par Test Studio n'était pas satisfaisante, nous avons sélectionné l'outil JMeter pour implémenter la suite de tests complète sur le serveur de développement Moodle.

L'implémentation nous a permis de fournir une suite de tests fonctionnelle, basée sur la version Moodle actuellement utilisée à la HES-SO. Cette suite de tests porte sur les principaux processus utilisateur, dont les performances sont générées et peuvent être exportées pour une utilisation ultérieure.

Nous envisageons, par la suite, de transposer la suite créée vers d'autres versions de la plateforme. Cela nous permettrait alors de comparer les résultats obtenus entre la version actuelle et une autre version de l'applicatif.

De même, il serait intéressant de planifier des exécutions régulières et automatisées de la suite de tests, à différentes heures de la journée. Cela permettrait de détecter d'éventuels ralentissements aux heures de pointes.

Dans la même optique, nous pourrions exécuter la suite de tests depuis les différentes HES de Suisse romande, pour pouvoir établir un tableau des performances de celles-ci.

## 31. Références

### 31.1. Livres

Scott Barber (2011) Web Load Testing For Dummies. John Wiley&Sons, Inc.

### 31.2. Sources électroniques

World Wide Web Consortium (W3C).(1999). Header field definition. Récupéré sur: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>

Application Quality Management.(2009). Présentation des scénarios de test. Récupéré sur: [http://pic.dhe.ibm.com/infocenter/clmhelp/v4r0m4/index.jsp?re=1&topic=/com.ibm.rational.test.qm.doc/topics/c\\_testcase\\_overview.html&scope=null](http://pic.dhe.ibm.com/infocenter/clmhelp/v4r0m4/index.jsp?re=1&topic=/com.ibm.rational.test.qm.doc/topics/c_testcase_overview.html&scope=null)

Software Testing(2011). Test case. Récupéré sur: <http://softwaretestingfundamentals.com>

Patrick Cuff.(2010). Difference between acceptance test and functional test ?. Récupéré sur: <http://stackoverflow.com/questions/3370334/difference-between-acceptance-test-and-functional-test>

Anne Furey&Arjan Pottjewijd.(2007). Integrated developement environment. Récupéré sur: <http://searchsoftwarequality.techtarget.com>

Moodle community (2014). History. Récupéré sur: <http://docs.moodle.org/27/en/History>

Vince Bordo.(2011). Overview of User Acceptance Testing. Récupéré sur: <http://www.develop.com/useracceptancetests>

Stat Counter.(2014). Global stats. Récupéré sur: <http://gs.statcounter.com/>

Abbey Maxime (2011). Freeware. Récupéré sur: <http://www.dmoz.org/World/Fran%C3%A7ais/Informatique/Logiciels/Freeware/R%C3%A9pertoire>

Osaxis (2012 ). Gatling, nouvelle solution libre pour les tests de montée en charge. Récupéré sur: <http://www.osaxis.fr/blog/gatling-nouvelle-solution-libre-pour-les-tests-de-montee-en-charge/>

PushToTest (2011). Installation, Configuration, Integration. Récupéré sur: <http://www.pushtotest.com/installation-configuration-deployment-testmaker-6>

Selenium Community (2013). What is Selenium ?. Récupéré sur: <http://docs.seleniumhq.org/>

JUnit (2014). JUnit. Récupéré sur: [www.junit.org](http://www.junit.org)

Apache foundation (2013). Getting started. Récupéré sur: <http://jmeter.apache.org/usermanual/get-started.html>

Selenium(2014). Getting started. Récupéré sur: <https://code.google.com/p/selenium/wiki/GettingStarted>

Apache foundation (2013). Elements of a test plan. Récupéré sur: [http://jmeter.apache.org/usermanual/test\\_plan.html](http://jmeter.apache.org/usermanual/test_plan.html)

Funkload documentation (2014). Introduction. Récupéré sur: <https://funkload.nuxeo.org/intro.html#features>

Stephane Landelle (2013). First steps with Gatling. Récupéré sur: <https://github.com/excilys/gatling/wiki/First-Steps-with-Gatling>

Jmeter (2014). Web driver sampler. Récupéré sur: <http://jmeter-plugins.org/wiki/WebDriverSampler/>

Pierre Tholence (2011). FireXPath. Récupéré sur: <https://addons.mozilla.org/fr/firefox/addon/firepath/>

Telerik (2014). Key Features. Récupéré sur: <http://www.telerik.com/teststudio>

Telerik (2014). Overview documentation. Récupéré sur: <http://docs.telerik.com/teststudio>

## 32. Annexes

### 32.1. Installateurs des outils

Les installateurs des outils testés sont présents sur le cd fourni, dans le répertoire **installateurs\_outils**.

### 32.2. Scripts de test développés

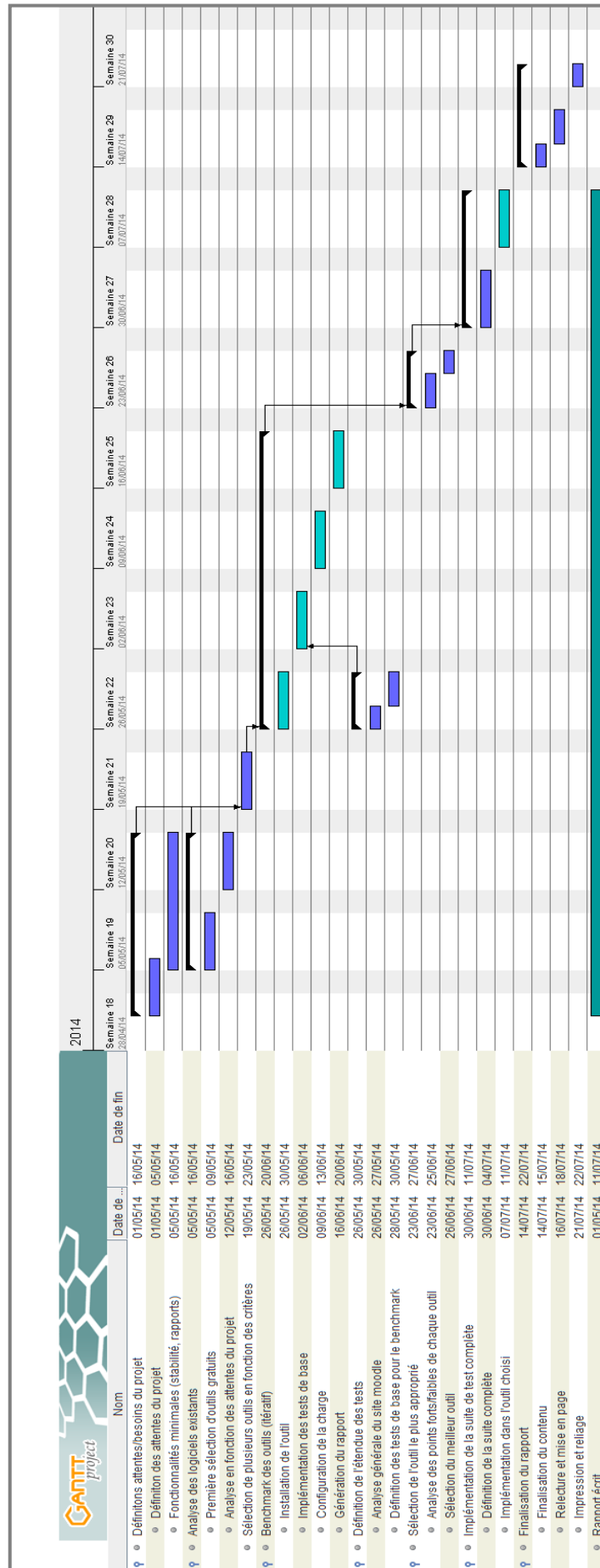
Les suites de tests créées dans le cadre du Benchmark des outils se trouvent dans le dossier **benchmark/outils** du cd.

### 32.3. Suite de test définitive

La suite de tests créée, les résultats de celle-ci et une version du logiciel JMeter sont présents dans le répertoire **Suite de test (Moodle DEV)** du cd fourni.

### 32.4. Planification

La planification est aussi disponible sur le cd au format .gan, dans le répertoire « **annexes** ».





### **32.5. Rapports hebdomadaires**

Les différents rapports hebdomadaires sont présentés sur le cd, dans le répertoire **annexes/Rapports Hebdomadaires**.

### **32.6. Cahier des charges**

Le cahier des charges est disponible au format pdf sur le cd (**annexes/CahierDesCharges**).

## **33. Déclaration**

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

Anne-Dominique Salamin

David Rousseau