



RÉPUBLIQUE DU BÉNIN
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ D'ABOMEY-CALAVI

INSTITUT DE FORMATION ET DE
RECHERCHE EN INFORMATIQUE

BP 526 Cotonou Tel : +229 21 14 19 88
<http://www.ifri-uac.bj> Courriel : contact@ifri.uac.bj



MÉMOIRE

pour l'obtention du

Diplôme de Licence en Informatique

Option : Génie Logiciel

Présenté par :

Jolivé Gilbert Mahoudjro **HODEHOU**

Automatisation des tests fonctionnels sur les projets informatiques

Sous la supervision :

Ing. Christian A. **AKPONA**,

Chef Division Génie Logiciel à la Direction Générale des Impôts

Membres du jury :

M. TIDJANI Sanda	Docteur	IFRI	Président
M. AKPONA Christian	Ingénieur Administrateur Système	DGI	Examineur
M. AHLONMESSOU Jonas	Chef Division/Génie Logiciel	DGI	Rapporteur

Année Académique : 2019 - 2020

Sommaire

Dédicace	ii
Remerciements	iii
Résumé	iv
Abstract	v
Liste des figures	vi
Liste des tables	vii
Liste des acronymes	viii
Introduction	1
1 Revue de littérature	2
2 Analyse, choix techniques et conception	10
3 Résultats et discussion	20
Conclusion	32
Bibliographie	33
Webographie	34
Annexe	35
Table des matières	38

Dédicace

A

Mon père **Joseph B. HODEHOU**

Ma mère **Victoire AIVODJI**

Mes frères et sœurs

Remerciements

Nous exprimons notre profonde gratitude à l'Eternel, Dieu Créateur, pour son amour infini, pour l'inspiration et la protection dont il nous comble tous les jours. A travers ce mémoire, nous tenons à remercier :

- Le Professeur Eugène C. EZIN, le Directeur de notre institut pour nous avoir assuré cette formation;
- Ing. Christian A. AKPONA, notre superviseur pour avoir accepté diriger et suivre avec grand intérêt le déroulement de ce projet. Son sens d'écoute, sa disponibilité et ses encouragements nous ont permis de parfaire ce travail;
- Ing. Dario E. DONOU, pour ses sages conseils et ses orientations[4];
- M. Christopher GBASSI, Directeur de Go Digital Bénin pour avoir aidé tout long de mon stage
- Tous les enseignants de notre institut en particulier ceux de la filière Génie Logiciel pour avoir accepté partager une partie de leurs connaissances avec nous;
- Tous les étudiants de la quatrième promotion de Génie Logiciel/Sécurité Informatique/Internet et Multimédia de l'institut de formation et de recherche en informatique;
- Tous ceux qui m'ont aidé à la réalisation de ce modeste mémoire.

Résumé

L'impact des tests sur le progrès et le bon déroulement d'un projet n'est plus à démontrer. Le présent mémoire de fin de formation propose une solution aux problèmes liés aux tests fonctionnels que rencontrent les développeurs et testeurs lors de la mise en place et de l'évolution d'un projet informatique. Cette solution vise à créer, gérer et effectuer ses tests plus rapidement pour n'importe quelle application web.

Nous avons réalisé un système dont l'objectif est d'automatiser les tests fonctionnels afin d'exécuter les tests plus rapidement, d'améliorer la transparence et les rapports et d'être efficace dans les intégrations.

En exploitant le Modèle d'Objet de Document, plusieurs sites web développés en bootstrap ont été testés. Les résultats obtenus sur les données de tests révèlent que 95% des fonctionnalités ont pu être testées avec succès.

La solution fournit donc relativement de bons résultats et des améliorations sont prévues afin d'affiner la précision sur les composants web.

Mots clés : Tests fonctionnels, Composants Web, Automatisation des tests , PHP

Abstract

The impact of testing on the progress and smooth running of a project is well established. This end-of-training thesis proposes a solution to the problems related to functional tests that developers and testers encounter when setting up and developing an IT project. This solution aims to create, manage and perform functional tests automatically for any web application.

We have created a system whose objective is to automate functional tests in order to execute tests faster, improve transparency and reporting and be efficient in integrations.

By exploiting the Document Object Model, several websites developed in bootstrap have been tested. The results obtained from the test data show that 95% of the functionalities could be successfully tested.

The solution therefore provides relatively good results and improvements are planned to refine the accuracy on the web components.

Key words: Functional tests, Web components, Test automation , PHP

Liste des figures

2.1	Architecture du système	11
2.2	configuration de Selenium Grid	14
2.3	Script pour lancer GeckoDriver	15
2.4	Configuration Elkan Formatter	16
2.5	Configuration Paths	17
2.6	Configuration Filters	17
2.7	Personnalisation Formatter	17
2.8	Configuration Context	18
2.9	Configuration Extensions	18
2.10	Écritures et processus des tests	19
3.1	Configuration behat yml	21
3.2	Fichiers de fonctionnalités	22
3.3	Contact.feature	22
3.4	Newsletter.feature	23
3.5	s_inscrire.feature	23
3.6	search.feature	24
3.7	Dossier report	24
3.8	Failed and Sucess	25
3.9	Failed	25
3.10	Success	26
3.11	Commande run.sh	26
3.12	Commande vendor/bin/behat	27
3.13	Site du senum	27
3.14	Résultats de test en console	29
3.15	Rapport du formatter	31

Liste des tables

1.1 Pré-requis de notre système 8

Liste des acronymes

ATDD :

Acceptance Test-Driven Development [5](#)

BDD :

Behavior-Driven Development [5](#)

CSV :

Comma-Separated Values [31](#)

DDD :

Domain-Driven Design [5](#)

DOM :

Document Object Model [31](#)

IE :

Internet Explorer [8](#)

PHP :

Hypertext Preprocessor [7](#), [12](#), [19](#)

QA :

Quality Assurance [8](#)

TDD :

Test-Driven Development [5](#)

UI :

User Interface [6](#)

UX :

User Experience [6](#)

Watir :

Web Application Testing in Ruby [7](#)

Introduction Générale

Contexte

Les projets font partie intégrante du quotidien dans l'industrie de l'informatique et cela ne cesse d'accroître. De nos jours, les ordinateurs aident les humains dans la plupart des aspects de nos vies quotidiennes, et les applications informatiques sont rendues très puissantes et complexes. Tester une application informatique simple n'est pas une tâche très difficile, mais tester un grand logiciel ayant plusieurs fonctionnalités est une tâche très complexe.

Problématique

L'industrie informatique débourse beaucoup d'argent et de temps pour tester de tels logiciels. Elle cherche toujours de nouvelles méthodes pour épargner temps et argent nécessaire à ces tests. De plus, les entreprises ne disposent pas des ressources pour effectuer tous les cas possibles de tests.

Objectif

Le présent travail vise à mettre en place une solution informatique automatisée dont l'objectif sera de faciliter la mise en place des tests fonctionnels et l'exécution de ses tests par un automate.

Organisation du document

Pour mener à bien ce travail, trois chapitres ont été abordés. Le premier chapitre définit le contexte d'étude et l'état de l'art en abordant la généralité sur les tests informatiques et sur l'automatisation de ses tests. Le deuxième quant à lui porte sur le procédé d'analyse et de conception utilisé ainsi que sur les choix techniques réalisés pour mener à bien la tâche. Dans le troisième chapitre, nous présenterons le prototype de notre travail tout en mettant en avant les résultats issus de nos simulations après les avoir exposés. Enfin, une conclusion synthétise notre travail et présente les perspectives envisagées.

Revue de littérature

1.1 Introduction

Les procédures de vérification partielle d'un logiciel apportent un effet positif dans le cycle de développement des projets informatiques. Les tests du logiciel permettent de créer un produit qui répond pleinement aux besoins ou attentes des utilisateurs. Dans cette démarche, nous présenterons dans ce chapitre une généralité sur les tests en informatique et sur les tests automatisés.

1.2 Généralité sur les tests en informatique

1.2.1 Qu'est-ce qu'un test en informatique

Le test d'un produit logiciel est l'une des étapes les plus importantes du processus de développement et de production. Il permet d'évaluer efficacement la qualité des systèmes et des applications.

1.2.2 Quels sont les types de tests en informatique

Les tests de logiciels sont généralement classés en deux types : les tests fonctionnels et non-fonctionnels.

Différence clé :

- Les tests fonctionnels vérifient chaque fonction, caractéristique du logiciel, tandis que les tests non-fonctionnels vérifient les aspects non-fonctionnels tels que les performances, l'utilisabilité, la fiabilité, etc.
- Les tests fonctionnels peuvent être effectués manuellement alors que les tests non-fonctionnels sont difficiles à effectuer manuellement.
- Les tests fonctionnels sont basés sur les exigences du client, tandis que les tests non-fonctionnels sont basés sur les attentes du client.
- Les tests fonctionnels ont pour objectif de valider les actions logicielles, tandis que les tests non-fonctionnels ont pour objectif de valider les performances du logiciel.
- Les tests fonctionnels sont effectués avant les tests non-fonctionnels.

1.2.2.1 Tests fonctionnels

Les tests fonctionnels visent à vérifier la conformité du logiciel avec les exigences énoncées dans le cahier des charges. Les tests complets des fonctionnalités déclarées et les tests des fonctionnalités de base uniquement peuvent être effectués.

1.2.2.1.1 Test unitaire : vérifie qu'une unité distincte de la base de code fonctionne correctement. Le test unitaire montre si la logique du code est appropriée et fonctionne. Il améliore la lisibilité du code, cela permet aux développeurs de comprendre le code source plus rapidement et les modifications sont plus faciles à implémenter.

1.2.2.1.2 Test d'intégration : est effectué lorsque deux ou plusieurs fonctions ou composants du logiciel sont intégrés pour former un système. Il vérifie essentiellement le bon fonctionnement du logiciel lorsque les composants sont fusionnés pour fonctionner comme une seule unité.

1.2.2.1.3 Test de système : est chargé de la vérification de haut niveau de la fonctionnalité de l'ensemble du programme ou du système dans son ensemble. Il comprend plusieurs types de tests de logiciels qui vous permettront de vérifier le logiciel dans son ensemble (logiciel, matériel et réseau) conformément aux exigences pour lesquelles il a été créé.

1.2.2.1.4 Test d'intégrité : test des méthodes et processus utilisés pour accéder et gérer les bases de données, pour s'assurer que les méthodes d'accès, processus et règles de données fonctionnent comme attendus et que lors des accès à la base de données, les données ne sont pas corrompues ou inopinément effacées, mises à jour ou créées.

1.2.2.1.5 Test de fumée : est effectué avant le test réel du système pour vérifier si les fonctionnalités critiques fonctionnent correctement afin d'effectuer des tests approfondis supplémentaires.

1.2.2.1.6 Test d'interface : vérifie si la communication entre deux systèmes logiciels différents est effectuée correctement.

1.2.2.1.7 Test de régression : est effectué afin de vérifier si de nouvelles fonctions, améliorations et défauts corrigés affectent la fonctionnalité existante du produit et si d'anciens défauts se produisent.

1.2.2.2 Tests Non Fonctionnels

Les tests non-fonctionnels vérifient la conformité aux exigences non-fonctionnels.

1.2.2.2.1 Test de performance : est effectué afin de déterminer la vitesse à laquelle le système ou une partie de celui-ci fonctionne sous une certaine charge.

1.2.2.2.2 Test de charges : est conçu pour vérifier les performances du système sous des charges standard et pour déterminer le pic maximal possible auquel le système fonctionne correctement.

1.2.2.2.3 Test de contraintes : vérifie la stabilité et la fiabilité du système. Ce test mesure principalement le système sur sa robustesse et ses capacités de traitement des erreurs dans des conditions de charge extrêmement élevée.

1.2.2.2.4 Test de volume : vérifie la capacité de l'application à gérer les données du volume en fournissant un environnement en temps réel. L'application est testée pour son exactitude et sa fiabilité dans des conditions défavorables.

1.2.2.2.5 Test de sécurité : est une stratégie de test utilisée pour vérifier la sécurité du système, ainsi que pour analyser les risques associés à la fourniture d'une approche approximative de la protection de l'application, les attaques de pirates, les virus, l'accès non autorisé aux données sensibles.

1.2.2.2.6 Test de compatibilité : est un type de test logiciel utilisé pour assurer la compatibilité du système avec divers autres objets tels que d'autres navigateurs Web, plates-formes matérielles, utilisateurs (dans le cas où il s'agit d'un type d'exigence très spécifique, tel qu'un utilisateur qui parle et ne peut lire qu'une langue particulière), les systèmes d'exploitation, etc.

1.2.2.2.7 Test d'installation : permet de rechercher des erreurs qui se produisent dans le processus d'installation qui affectent la perception de l'utilisateur et sa capacité à utiliser le logiciel installé.

1.2.2.2.8 Test de récupération : consiste à tester dans quelle mesure une application est capable de récupérer après des pannes, des pannes matérielles et d'autres problèmes similaires.

1.2.2.2.9 Test de fiabilité : est un type de test logiciel, qui vérifie si le logiciel peut effectuer une opération sans échec pendant une période de temps spécifiée dans un environnement particulier.

1.2.2.2.10 Test de d'utilisabilité : est une méthode de test visant à établir le degré de convivialité, la capacité d'apprentissage, la compréhensibilité et l'attractivité pour les utilisateurs d'un produit développé dans le contexte de conditions données. Identifier les problèmes associés à un mécanisme d'interface spécifique pour déterminer s'il existe des problèmes avec la commodité de l'interface pour la navigation, l'utilisation de la fonctionnalité principale.

1.2.2.2.11 Test de conformité : est un test qui permet de valider si le système développé répond aux normes prescrites par l'organisation ou non.

1.2.2.2.12 Test de localisation : est le processus de test d'une version localisée d'un produit logiciel. Vérification de la traduction des éléments de l'interface utilisateur, vérification de la traduction des messages du système et des erreurs.

1.3 Généralité sur les tests automatisés

1.3.1 Qu'est-ce qu'un test automatisé

L'automatisation du test est un processus de vérification logiciel qui comprend la mise en œuvre de fonctions de base et d'étapes de test telles que le lancement, l'initialisation, l'exécution, l'analyse et la sortie automatique du résultat à l'aide d'outils spécialisés. Les tests automatisés sont un analogue des tests fonctionnels manuels, qui sont effectués par un programme de robot et non par une personne.

1.3.2 Qu'est le BDD, le TDD, le DDD et l'ATDD

Ce sont des pratiques de tests utilisés par les testeurs. Développement conduit par le comportement (BDD), le développement piloté par le test (TDD), le développement piloté par le test d'acceptation (ATDD)

BDD : Le [BDD](#) est une méthode de programmation agile qui encourage la collaboration entre les développeurs, les ingénieurs qualité et les intervenants non-techniques ou commerciaux participant à un projet logiciel. Il encourage les équipes à utiliser la conversation et les exemples concrets pour formaliser une compréhension commune de la façon dont l'application doit se comporter[10].

DDD : Le [DDD](#) est le concept selon lequel la structure et le langage de votre code (noms de classe, méthodes de classe, variables de classe) doivent correspondre au domaine métier. Par exemple, si votre logiciel traite des demandes de prêt, il peut avoir des classes telles que `LoanApplication` et `Customer` et des méthodes telles que `AcceptOffer` et `Withdraw`.

TDD : Le [TDD](#) est une méthode de développement de logiciel, qui consiste à concevoir un logiciel par petit pas, de façon itérative et incrémentale, en écrivant chaque test avant d'écrire le code source et en remaniant le code continuellement.

ATDD : Le [ATDD](#) est une méthodologie de développement basée sur la communication entre les clients commerciaux, les développeurs et les testeurs.

1.3.3 L'enjeu des tests automatisés

1.3.3.1 Quel est le but des tests automatisés

L'automatisation des tests ne doit pas accélérer le processus de développement, l'objectif doit être une meilleure qualité logicielle grâce à une couverture des tests plus élevée. Le temps de développement peut certainement être raccourci, car des erreurs peuvent également être trouvées plus tôt si les tests automatisés s'exécutent plus souvent, mais cela ne devrait pas être l'objectif général. Les tests automatisés sont toujours un gain de temps ainsi qu'une économie de coûts. Parce que des milliers de scripts de test peuvent être exécutés en peu de temps après la création des tests automatisés, alors que dans les tests manuels, cela prendrait beaucoup de temps et de main-d'œuvre[9].

1.3.3.2 Test Manuel et Test Automatisés

Dans les tests manuels, les testeurs effectuent manuellement des tests sans utiliser d'outils d'automatisation. Le test manuel est le niveau de test le plus bas et le plus simple qui ne nécessite pas beaucoup de connaissances supplémentaires.

Les tests automatisés impliquent l'utilisation de logiciels spéciaux (en plus des tests) pour surveiller l'exécution des tests et comparer le résultat réel attendu du programme. Ce type de test permet d'automatiser les tâches fréquemment répétées qui sont nécessaires pour maximiser la couverture des tests.

Certaines tâches de tests, telles que les tests de régression de bas niveau, peuvent être longues si elles sont effectuées manuellement. En outre, les tests manuels peuvent ne pas trouver efficacement certaines classes d'erreur. Dans de tels cas, l'automatisation peut aider à économiser le temps et les efforts de l'équipe de projet et de l'argent. Les tests manuels peuvent être répétitifs et ennuyeux. Dans le même temps, l'automatisation peut aider à éviter cela - l'ordinateur fera tout pour vous. Ainsi, sur des projets réels, une combinaison de tests manuels et automatisés est souvent utilisée, et le niveau d'automatisation dépendra à la fois du type de projet et des caractéristiques des processus de production dans l'entreprise. Il faut noter que les tests manuels et automatisés ne se cannibalisent pas, mais s'inscrivent dans une logique de complémentarité, chaque approche répondant à une organisation précise et à un besoin à un instant t.

Les tests automatisés

Ils réduisent considérablement les coûts d'erreur qui peuvent survenir pendant le fonctionnement, car les dysfonctionnements, les problèmes de performances ou les incompatibilités système d'un service informatique peuvent être constatés avant la mise en service et corrigés en conséquence.

C'est un bon outil pour l'assurance qualité des logiciels et c'est pourquoi dans de nombreux projets logiciels complexes et volumineux, une très grande proportion des tests sont effectués par des procédures de test automatisées.

Avantages Clés :

- Amélioration de la qualité des tests, car lors de l'utilisation d'outils d'automatisation, le « facteur humain » n'affecte pas la qualité des tests.
- La possibilité d'effectuer de tels types de tests, qui ne peuvent pas être effectués manuellement, ou nécessitent des coûts importants (équipement supplémentaire, personnel).
- Accélérez le processus de test sans perdre en qualité. Effectuer la même quantité de travail par méthode manuelle prend plus de temps. L'utilisation d'outils d'automatisation pour les tests vous permet d'exécuter des scripts déjà écrits sans autres modifications.
- Pendant les tests, des rapports sur les résultats des applications logicielles sont envoyés et enregistrés automatiquement.

Les tests manuels

Contrairement au test automatisé, le test manuel vous permet de tester l'**UI** et **UX** de votre produit : l'affichage correct du texte, les liens, les images. Cela vous permet de déceler des bugs qui seraient visibles par vos utilisateurs, mais n'auraient pas pu être repérés par un robot.

1.4 Problématique

1.4.1 Scénario : Maintien d'un site avec un cahier des charge de près de 500 pages et 200 fonctionnalités

Openware est une entreprise spécialisée dans les tests informatiques. Elle lui a été confiée de tester et de relever les bugs et régressions d'un site internet disposant de près de 200 fonctionnalités. L'objectif est d'être en mesure de pouvoir tester toutes les fonctionnalités et de relever les différents problèmes du site. Openware se retrouve donc confronter à réaliser sur certains tests des cas de fonctionnalités redondantes. Notamment pour sauvegarder une données dans une base ou lire un fichier, il faut à chaque fois passer par la connexion d'un utilisateur ou par la déconnexion d'un autre utilisateur. Effectuer donc les tests manuellement et un rapport devient un travail fastidieux.

1.5 Objectif Général

L'objectif est de mettre en place une solution qui permet de gérer plus facilement les tests fonctionnels sur les projets informatiques afin de s'assurer du bon fonctionnement des applications.

1.6 Étude de l'existant

1.6.1 Présentation des solutions existantes

1.6.1.1 Selenium

Selenium est un framework portable pour tester les applications web. Selenium fournit un outil de lecture pour créer des tests fonctionnels sans avoir besoin d'apprendre un langage de script de test (Selenium IDE). Il fournit également un langage spécifique au domaine de test (sélénien) pour écrire des tests dans un certain nombre de langages de programmation populaires, notamment C#; Groovy; Java; Perl; [PHP](#); Python; Ruby et Scala . Les tests peuvent ensuite s'exécuter sur la plupart des navigateurs Web modernes. Selenium est une suite d'outils pour automatiser les navigateurs Web[8].

1.6.1.2 Watir

Watir prononcé water, est une famille open source de bibliothèques Ruby pour l'automatisation des navigateurs Web. Il pilote Internet Explorer, Firefox, Chrome, Opera et Safari, et est disponible en tant que gemme RubyGems. développé par Bret Pettichord et Paul Rogers.

1.6.1.3 Soap UI

Soap UI est une application open source permettant le test de service web dans une architecture orientée services (SOA). Ses fonctionnalités figurent l'inspection des services web, l'invocation, le développement, la simulation, le mocking, les tests fonctionnels, les tests de charge et de conformité.

1.6.1.4 Cucumber

Cucumber est un outil logiciel qui prend en charge le développement axé sur le comportement (BDD). Au centre de l'approche BDD cucumber se trouve son analyseur de langage ordinaire appelé Gherkin. Il permet de spécifier les comportements logiciels attendus dans un langage logique que les clients peuvent comprendre. En tant que tel, Cucumber[6] permet l'exécution de la documentation des fonctionnalités écrite en texte destiné aux entreprises. Il est souvent utilisé pour tester d'autres logiciels. Il exécute des tests d'acceptation automatisés écrits dans un style de développement axé sur le comportement (BDD).

1.6.2 Résumé des exigences de notre système

Il faut rappeler les exigences de notre projet à mettre en place afin de pouvoir les comparer avec le système pour en dégager les exigences qui répondent ou pas à notre projet. Nous avons donc 9 exigences listées ci-dessous :

1. Le système doit pouvoir comprendre un langage naturel ;
2. Le système doit pouvoir simuler l'interaction entre le navigateur et l'application ;
3. Le système doit pouvoir piloter un navigateur de manière native, comme le ferait un utilisateur réel ;
4. Le système doit pouvoir générer un rapport propre et lisible après un test ;
5. Le QA Testeur doit pouvoir imprimer le rapport ;
6. Le système doit pouvoir effectuer les tests sur une application développée en bootstrap ;
7. Le système doit pouvoir simuler l'interaction dans les principaux navigateur (Chrome , Firefox , Safari , IE , Edge) ;
8. Le système doit pouvoir piloter les principaux navigateur ;
9. Le système doit pouvoir prendre des captures d'écran.

TABLE 1.1 – Pré-requis de notre système

Système Existant	Requis de notre système								
	1	2	3	4	5	6	7	8	9
Selenium	Non	Oui	Oui	Oui	Non	Oui	Oui	Oui	Oui
Watir	Non	Oui	Oui	Oui	Non	Oui	Oui	Oui	Oui
Soap UI	Non	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Cucumber	Oui	Oui	Oui	Oui	Non	Oui	Oui	Oui	Oui

Ainsi, grâce au tableau comparatif, on peut déduire que les solutions existantes ne répondent pas à toutes les exigences de notre système. Par contre Soap UI et Cucumber répondent à 90% de ces exigences. Nous allons donc nous baser sur Cucumber, car facile d'utilisation et rapide pour une prise en main pour mettre en place notre propre solution d'automatisation des tests fonctionnels.

1.7 Conclusion

Pour conclure s'il est judicieux d'investir dans l'automatisation des tests, imaginez la situation. Supposons qu'il existe une certaine société X, dans laquelle tous les experts ont toujours testé la fonctionnalité manuellement. Autrement dit, l'automatisation est une sorte d'expérience qui, selon l'idée, devrait confirmer l'hypothèse selon laquelle les tests à l'aide du programme réduiront le temps de vérification du site et amélioreront la qualité du résultat de sortie. Le chapitre suivant fera l'objet d'une présentation de notre solution à travers sa conception et les outils utilisés pour sa réalisation.

Analyse, choix techniques et conception

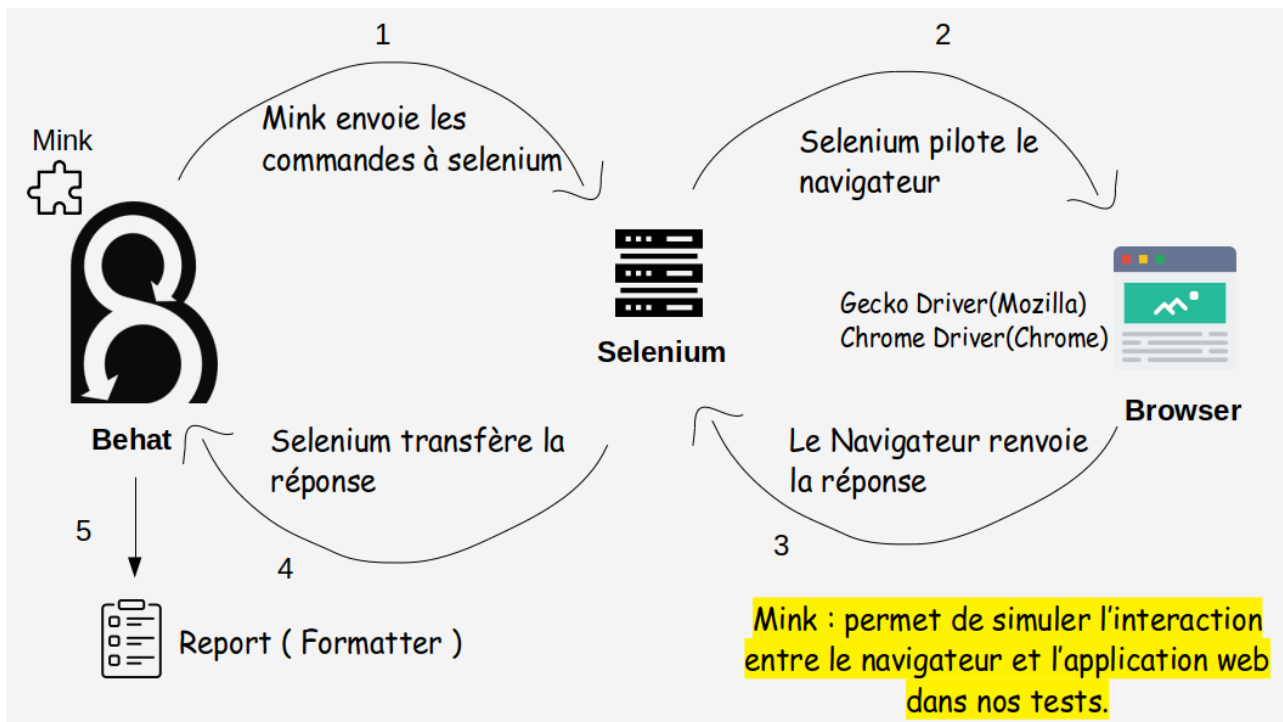
2.1 Introduction

Le but de la phase d'analyse est de formuler et de formaliser les exigences du système. Ceci est accompli en établissant ce que le système doit faire, selon les exigences et les attentes des utilisateurs finaux du système. La modélisation de ces exigences est ensuite effectuée sous la forme de modèles commerciaux, de données, d'événements et de processus pour démontrer la compréhension des exigences. À la fin de ce chapitre, nous devons être capables de savoir ce que fait le système ainsi que les technologies à utiliser pour la concevoir.

2.2 Analyse

2.2.1 Architecture du Système

Nous avons utilisé un serveur Selenium qui recevra les informations fournies par Mink (extension de Behat) et qui pilotera ensuite le navigateur pour les tests et générer un rapport grâce à notre formatter. Pour être clair sur l'architecture, voici la figure [2.1](#) qui résume le rôle de chacun :



Source : Jolivé Hodehou , Architecture du système

FIGURE 2.1 – Architecture du système

2.2.2 Entités du système

La définition d'une stratégie est une première étape essentielle pour une initiative d'automatisation de test réussie, mais tout votre travail acharné peut encore se dérouler si votre organisation ne possède pas une culture qui soutient vos objectifs. L'automatisation des tests est plus efficace lorsqu'il existe une collaboration entre différents membres : les analystes commerciaux, les développeurs, les testeurs et bien sûr, les ingénieurs en automatisation.

2.2.2.1 Les développeurs

Les développeurs ont également un rôle important dans l'automatisation des tests et ils en retirent sans doute le plus d'avantages. Si l'automatisation des tests fonctionnels est bien faite, les développeurs peuvent progresser beaucoup plus rapidement avec le développement de leurs fonctionnalités. En effet, ils n'ont pas besoin d'être aussi craintifs quant à l'impact que leur code peut avoir sur la base de code existante, car ils peuvent rapidement exécuter les tests automatisés contre leurs nouvelles modifications et corriger tout ce qu'ils ont pu casser avant d'intégrer réellement le code. Un autre domaine critique est de s'assurer que les fonctionnalités qu'ils fournissent peuvent être facilement automatisées, les développeurs peuvent également aider à mettre à jour les tests qui échouent en raison des nouvelles modifications qu'ils ont apportées. C'est une activité qu'ils font mieux parce que cela les oblige à répondre à la question « le changement de comportement attendu est-il valide ». Si la réponse est non, ils peuvent ajuster leur code avant que tout dommage ne soit causé. Ce retour rapide est extrêmement précieux. Ainsi, même sans diriger le projet d'automatisation des tests, les développeurs peuvent grandement contribuer à son succès.

2.2.2.2 Les QA testeurs

Même avec un projet d'automatisation de test réussi, vous avez toujours besoin de vos testeurs manuels. Beaucoup font l'erreur de croire que l'automatisation des tests remplace les tests. Ce n'est pas le cas. Les testeurs sont toujours nécessaires pour explorer et interroger en profondeur vos applications. Ce seront eux qui découvriront vos bugs les plus coûteux. Et en plus de leurs tests, ils peuvent également être d'une grande aide à l'initiative d'automatisation des tests en fournissant des informations sur les tests qu'il est préférable d'automatiser et les domaines de l'application qui peuvent nécessiter le plus de couverture par l'automatisation des tests. Pour ceux qui s'intéressent à l'automatisation des tests, ils peuvent également contribuer au tri et à la maintenance des scripts de test. Il est très important que votre organisation comprenne que l'automatisation des tests ne remplace pas les tests et qu'elle ne doit pas être considérée comme plus précieuse. Croire vraiment cela et s'assurer que vos testeurs comprennent également cela favorise une culture où les testeurs adoptent l'automatisation des tests et la considèrent comme un assistant pour eux par rapport à une menace potentielle pour leur travail.

2.3 Choix technique

2.3.1 Choix du langage de programmation et du framework

Un langage de programmation permet de décrire les structures des données qui seront manipulées par l'appareil informatique et quelles seront les manipulations. Il offre un ensemble de notions telles que les instructions, les variables, les types, et les procédures, qui peuvent être utilisées comme primitives pour développer des algorithmes.

Un framework désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel.

Nous mettons en place un système informatique, ce qui nous revient à choisir les langages de programmation web et infrastructure logicielle.

Au nombre des langages de programmation web qui existe, pour la réalisation de notre travail nous allons nous pencher sur un langage : le [PHP](#) et un framework : Behat[2]

2.3.1.1 PHP

PHP est un langage de programmation libre, principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale. PHP est un langage impératif orienté objet. Il est considéré comme une des bases de la création de sites web dits dynamiques mais, également des applications web.

Avantages :

- une communauté de développeurs partageant sans cesse des milliers de scripts PHP ;
- est Open Source et distribué sous une license autorisant la modification et la redistribution ;
- se combine très bien avec MySQL et d'autres bases de données ;
- est intégré dans de nombreux serveurs web (Apache par exemple) ;

- plusieurs framework de tests logiciel sont développés en PHP (Behat[2] , PHPUnit ,Codeception ,PHPsec etc..).

Inconvénients :

- une gestion partielle de la POO (Programmation Orientée Objet) ;
- aucun typage des variables, c'est très pénible pour la documentation et les contrôles des fonctions/méthodes ;
- des noms de fonctions pas cohérents.

2.3.1.2 Behat

Behat est un framework de test pour faire du développement axé sur le comportement. Cela consiste à rédiger plusieurs scénarios en langage Gherkin[5], proche de l'anglais naturel, avec indentation comme syntaxe, dans des fichiers .feature. Ces tests peuvent également tester le JavaScript[3].

Avantages :

- intégration de langage ubiquitaire facile en réutilisant le code ;
- offre un point de vue moins technique et un peu plus fonctionnel ;
- les tests sont écrits en langage naturel ;
- le travail conjoint entre le fonctionnel et la technique permet un suivi projet plus conséquent ;
- dans la mesure où le client est assez mature avec la méthodologie agile, on peut faire une recette fonctionnelle en BDD ;
- pas de régression fonctionnelle ;
- tests fonctionnels automatisables.

Inconvénients :

- Behat prend un certain temps pour la mise en oeuvre ;
- la discussion avec les parties prenantes et les développeurs peut s'étendre beaucoup jusqu'à ce que vous arriviez à une compréhension mutuelle de la façon dont une fonctionnalité sera développée.

2.3.2 Choix du serveur Sélénium et des drivers navigateurs**Selenium Server**

Selenium Server est essentiel pour exécuter des tests Selenium RC et des tests WebDriver sur des machines distantes via Selenium Grid. Selenium WebDriver est utilisé pour effectuer des tests d'automatisation de diverses applications Web sur différents navigateurs. Selenium Server est essentiellement basé sur la façon dont on entend utiliser Selenium WebDriver. Lors de la création d'une session avec le serveur Selenium, il faut envoyer ses capacités souhaitées. Ces fonctionnalités fournissent au

serveur Selenium les informations nécessaires telles que le navigateur et le système d'exploitation à utiliser. Le serveur Selenium gère ensuite toutes les demandes du BrowserDriver. Cela inclut également l'arrêt automatique en cas de problème pendant un test[7].

Serveur Selenium Standalone

Le serveur Selenium Standalone est un fichier java jar utilisé pour démarrer le serveur Selenium. Il s'agit d'un serveur proxy intelligent qui permet aux tests Selenium d'acheminer des commandes vers des instances de navigateur Web distantes. L'objectif est de fournir un moyen simple d'exécuter des tests en parallèle sur plusieurs machines.

Pour utiliser une grille Selenium , on peut télécharger le fichier JAR selenium-server-standalone. Tous les composants sont disponibles via le serveur sélénium. Le JAR autonome contient le serveur Selenium distant et les liaisons côté client qui vous permettent d'effectuer les tests sans avoir besoin de fichiers jar spécifique.

En outre, vous pouvez télécharger le serveur à partir de la page officielle de sélénium et démarrer le serveur via la ligne de commande en utilisant la commande ci-dessous :

```
java -jar selenium-server-standalone-3.11.0.jar
```

Une fois que le serveur est opérationnel, exécutez les scénarios de test. Procédez à la configuration de Selenium Grid et effectuez des tests sur Grid ou exécutez des tests avec Selenium WebDriver (voir figure 2.2).

```
9 selenium2:
0 wd_host: http://localhost:4444/wd/hub
1 browser: firefox
2 capabilities: {'browser':'firefox', 'marionette':true}
```

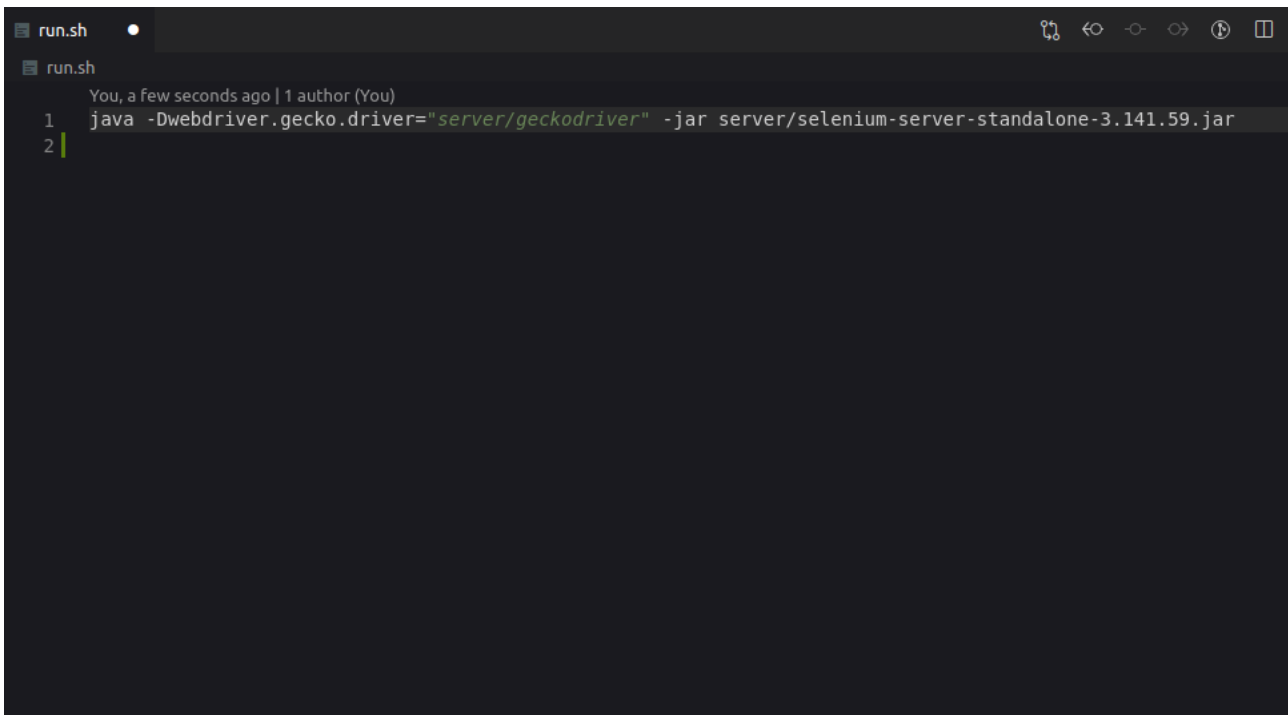
FIGURE 2.2 – configuration de Selenium Grid

Avantages du serveur Selenium Standalone

- permet la distribution des tests sur une machine distante ou plusieurs machines.
- effectue des tests d'automatisation sur différents navigateurs avec différentes versions.
- un seul serveur effectue toutes les tâches sans l'aide de pilotes externes.

Qu'est-ce que GeckoDriver?

GeckoDriver est un moteur de navigateur Web utilisé dans de nombreuses applications développées par Mozilla Foundation et Mozilla Corporation. GeckoDriver est le lien entre vos tests dans Selenium et le navigateur Firefox. GeckoDriver est un proxy permettant d'utiliser des clients compatibles W3C WebDriver pour interagir avec les navigateurs basés sur Gecko. La figure 2.3 montre le script java pour le démarrage de GeckoDriver. Pour Mozilla Firefox jusqu'à la version 47, nous n'avons jamais eu besoin de GeckoDriver. Mais Mozilla Firefox après la version 47, est livré avec Marionette, qui est un pilote d'automatisation pour Mozilla.

A screenshot of a terminal window with a dark background. The window title is 'run.sh'. The prompt is 'run.sh'. Below the prompt, there is a message: 'You, a few seconds ago | 1 author (You)'. The script content is as follows:

```
1 java -Dwebdriver.gecko.driver="server/geckodriver" -jar server/selenium-server-standalone-3.141.59.jar
2 |
```

FIGURE 2.3 – Script pour lancer GeckoDriver

Qu'est-ce que ChromeDriver?

Un WebDriver est un outil open source pour les tests automatisés d'applications Web sur de nombreux navigateurs. Il offre des capacités de navigation vers les pages Web, la saisie utilisateur, l'exécution de JavaScript et bien d'autres. ChromeDriver est un serveur autonome qui implémente le protocole filaire de WebDriver pour Chromium.

Le but principal de ChromeDriver est de lancer Google Chrome. Sans cela, il n'est pas possible d'exécuter des scripts de test Selenium dans Google Chrome ni d'automatiser une application Web. C'est la raison principale pour laquelle vous avez besoin de ChromeDriver pour exécuter des cas de test sur le navigateur Google Chrome.

2.3.3 Choix du Formatter de données (Elkan)

Le formater permet de générer des rapports de test fonctionnel après l'exécution de nos tests. Ces rapports résument les verdicts de réussite ou d'échec des éléments du journal de test. Ils sont générés à partir de l'exécution du test sous forme de fichiers HTML utilisant des conceptions de rapport prédéfinies.

Voici ci dessous les caractéristiques de notre formater. **Caractéristiques**

- Générer un rapport de test
- Créez une capture d'écran sur les étapes ayant échoué
- Créez une capture d'écran sur les étapes de résultat
- Enregistrez le code source en html sur les étapes échouées (débogage facile)

- Possibilité de définir le titre, la description et le logo du projet dans le rapport de test
- Bouton d'impression du rapport
- Bouton de téléchargement d'un fichier csv
- Filtrer sur les scénarios réussis ou échoués

Utilisation de base

Activez l'extension en spécifiant sa classe dans votre behat.yml (voir figure 2.4).

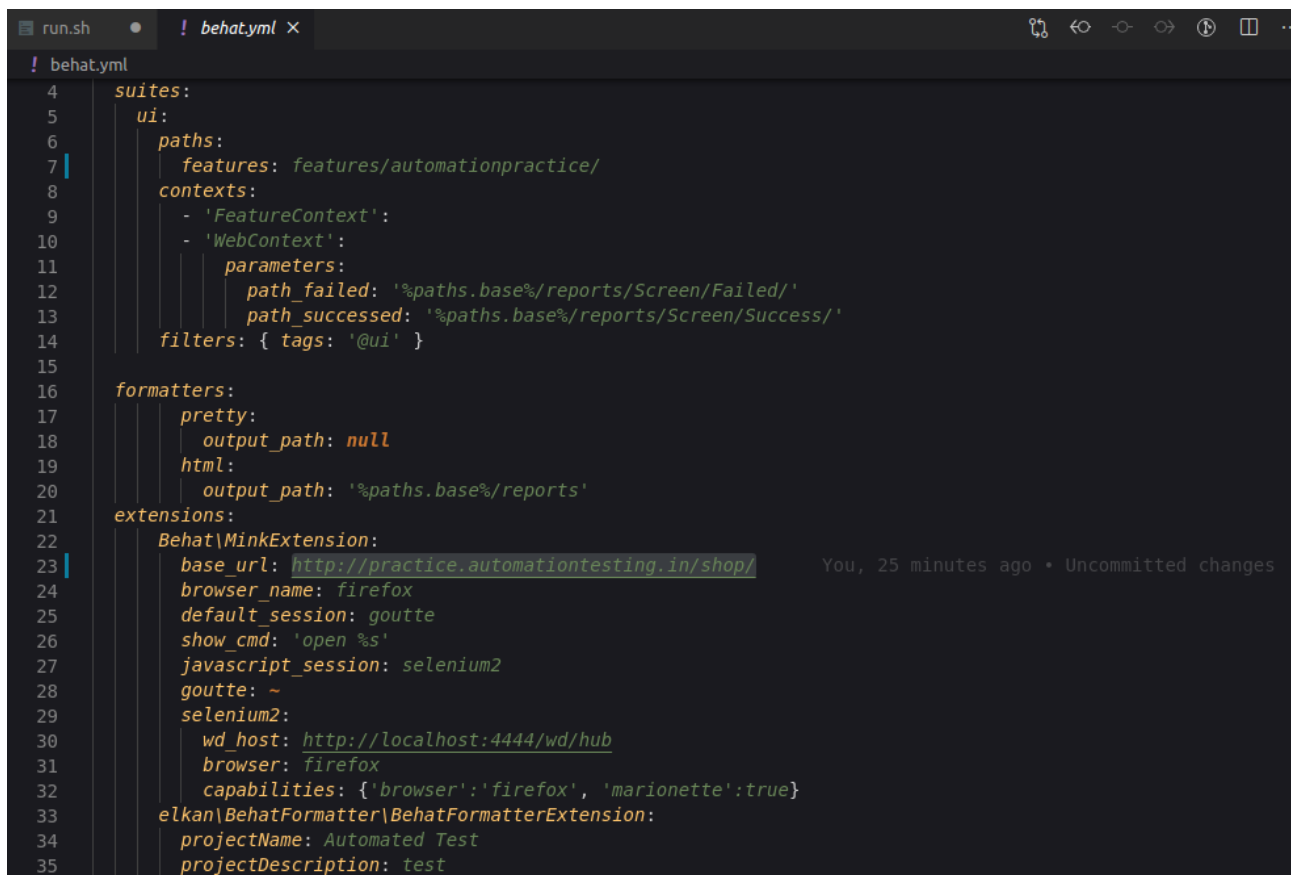


FIGURE 2.4 – Configuration Elkan Formatter

2.4 Configuration du projet

Behat dispose d'un système de configuration très puissant basé sur des fichiers de configuration et des profils YAML .

2.4.1 Environnement projet - Behat

Toute la configuration s'effectue dans un seul fichier de configuration au format YAML . Behat essaie de charger behat.yml ou config / behat.yml par défaut.

Paths

Le premier bloc de configuration est les paths . Les paramètres sous ce bloc de configuration indiquent à Behat où trouver vos fichiers de fonctionnalités et d'amorçage (voir figure 2.5).

```

autoload:
  - '%paths.base%/features/bootstrap/'
suites:
  ui:
    paths:
      features: features/automationpractice/

```

FIGURE 2.5 – Configuration Paths

- Le paramètre `features` définit où Behat recherchera vos fichiers `*.feature`. Le répertoire donné sera scanné de manière récursive.
- Le paramètre `autoload` définit le répertoire à partir duquel Behat chargera automatiquement tous les fichiers `*.php`.

Filters

Un autre bloc de configuration très utile est le bloc de filters (voir figure 2.6). Ce bloc définit les filtres par défaut (nom ou tag) pour vos fonctionnalités. Si vous vous retrouvez à taper les mêmes filtres encore et encore d'une exécution à l'autre, il serait plus efficace pour vous de les définir comme paramètres.

```

filters: { tags: '@ui' }

```

FIGURE 2.6 – Configuration Filters

Formatter

Le bloc du formatter (voir figure 2.7) est fait pour personnaliser votre formateur de sortie.

```

formatters:
  pretty:
    output_path: null
  html:
    output_path: '%paths.base%/reports'
extensions:

```

FIGURE 2.7 – Personnalisation Formatter

Context

Pour utiliser une classe de contexte par défaut différente ou fournir des paramètres utiles pour le constructeur de contexte à partir de votre `behat.yml`. Utilisez le bloc de contexte pour définir ces options (voir figure 2.8).

```

contexts:
  - 'FeatureContext':
  - 'WebContext':
    parameters:
      path_failed: '%paths.base%/reports/Screen/Failed/'
      path_succeeded: '%paths.base%/reports/Screen/Success/'

```

FIGURE 2.8 – Configuration Context

Extensions

Le bloc d'extensions permet d'activer des extensions pour votre suite ou pour un profil spécifique de la suite (voir figure 2.9)

```

extensions:
  Behat\MinkExtension:
    base_url: http://practice.automationtesting.in/shop/
    browser_name: firefox
    default_session: goutte
    show_cmd: 'open %s'
    javascript_session: selenium2
    goutte: ~
    selenium2:
      wd_host: http://localhost:4444/wd/hub
      browser: firefox
      capabilities: {'browser':'firefox', 'marionette':true}
  elkan\BehatFormatter\BehatFormatterExtension:
    projectName: Automated Test
    projectDescription: test
    projectImage: features/logo.png
    name: html
    renderer: Twig
    file_name: Index
    print_args: true
    print_outp: true
    loop_break: true
    show_tags: true

```

FIGURE 2.9 – Configuration Extensions

2.4.2 Ecritures et processus des tests

Nous devons avant tout écrire nos scénarios de tests fonctionnels (fichier .feature) que nous allons placer dans un répertoire features à la racine du projet. Il est possible de les mettre dans un sous-répertoire, cela dépend de la configuration dans le fichier behat.yml. Nous allons partir sur des tests permettant de s'assurer du bon fonctionnement d'une newsletter(voir figure 2.10).

```

features > automationpractice > 3newsletter.feature
1  @ui
2  Feature: Newsletter
3
4  @javascript
5  Scenario: should sign up for the newsletter
6      Given I am on the homepage
7      When I wait for "2" seconds
8      And I scroll to the bottom
9      And I fill the input of placeholder "Your email address" with the value "test@test.com"
10     And I click input with type "submit"
11     Then I wait for "2" seconds

```

FIGURE 2.10 – Écritures et processus des tests

@ui : est le nom de notre suite, rassemble des tests sur un certain sujet. Par exemple, tous les tests pour l'opération d'authentification d'un site Web peuvent vivre sous une seule suite. Dans notre cas d'utilisation, le nom de notre suite est ui.

Feature : décrit une activité spécifique. Par exemple, la newsletter, la connexion au site Web peut être une fonctionnalité, la réinitialisation du mot de passe peut être une autre fonctionnalité, et ainsi de suite.

@Javascript : est un tag qu'on met avant nos scénarios qui nécessite l'utilisation du JavaScript dans le navigateur.

Scenario : décrit un cas d'utilisation spécifique. Par exemple, un succès d'inscription ou encore un échec à une newsletter.

Step : décrit une petite partie de l'ensemble de l'opération que nous exécutons pour tester un scénario. Par exemple, dans notre cas on indique être présent sur la page d'accueil, ensuite attendre 2s et scroller jusqu'en bas de notre navigateur, remplir notre champ input dont le placeholder à une valeur "your email address" avec la valeur "test@test.com", et cliquer sur un input de type submit. Chaque scénario comprend plusieurs étapes dans la plupart des cas.

Step Definition : les étapes sont déjà défini, il s'agit du code [PHP](#) qui permet d'exécuter les étapes. `/vendor/bin/behat -dl` pour avoir la liste des étapes défini. Les étapes sont définies dans un fichier WebContext de notre projet. Et chaque étape correspond à une fonction dans le fichier WebContext.php

2.5 Conclusion

Après avoir délimité le périmètre d'étude et défini nos attentes pour l'application, nous avons présenté dans ce chapitre l'architecture de notre système ainsi que les entités du système et les choix techniques. Ce chapitre était consacré à l'analyse du système, à la conception de la configuration de l'environnement de travail ainsi qu'au choix technique.

Résultats et discussion

3.1 Introduction

Nous présenterons tout au long de ce chapitre, l'utilisation de notre solution pour effectuer les tests fonctionnels sur le site de la semaine du numérique du Bénin ainsi que ses limites. À la fin de ce chapitre, les objectifs doivent avoir été atteints et le système doit être prêt pour être utilisé sur différents projets informatiques.

3.2 Environnement du projet

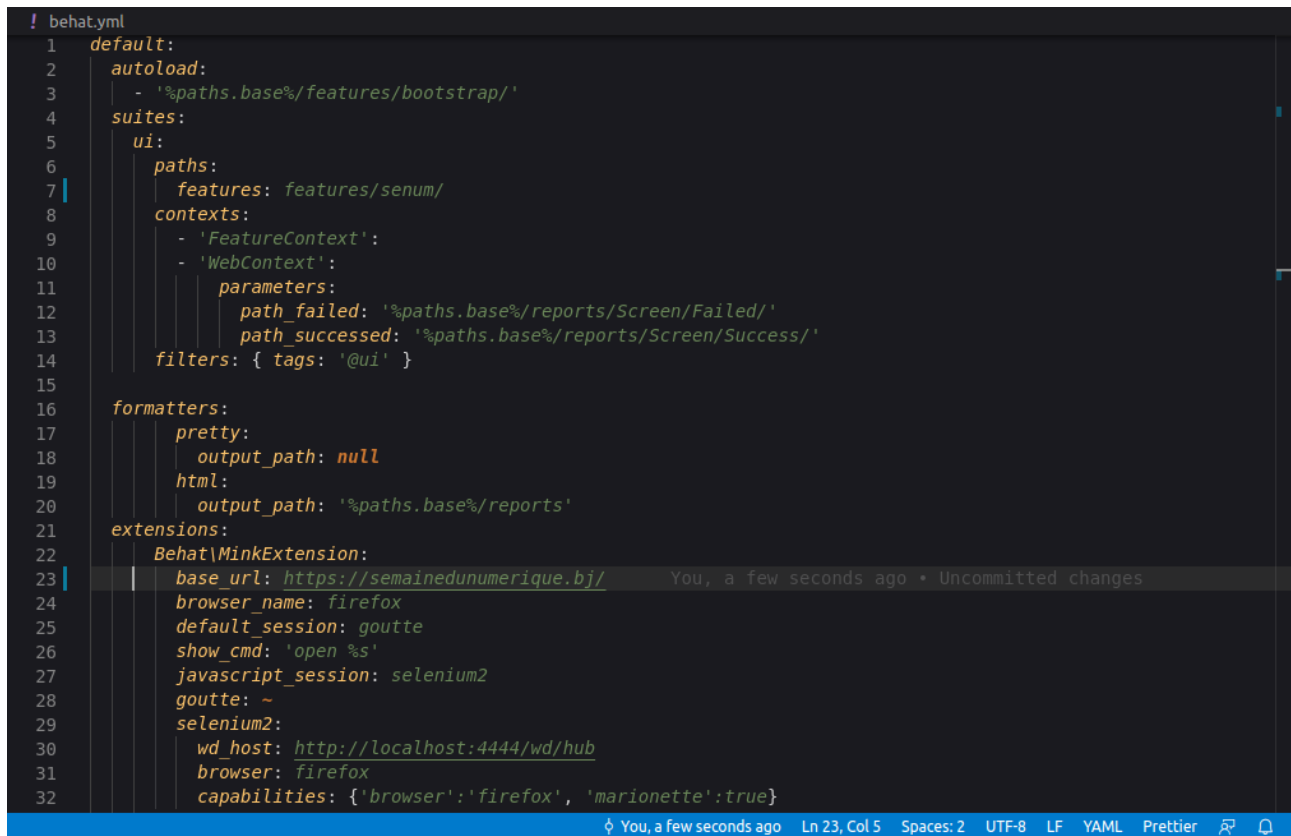
Il s'agira d'écrire des scripts de tests pour certaines fonctionnalités de <https://semainedunumerique.bj/>

Le projet se présente en trois points clés :

1. Le fichier de configuration behat.yml
2. Les fichiers de fonctionnalités
3. Le dossier du formater

3.2.1 Le fichier de configuration behat.yml

C'est le fichier contenant toutes les configurations nécessaires au projet de test.



```

! behat.yml
1  default:
2    autoload:
3      - '%paths.base%/features/bootstrap/'
4    suites:
5      ui:
6        paths:
7          features: features/senum/
8        contexts:
9          - 'FeatureContext':
10            - 'WebContext':
11              parameters:
12                path_failed: '%paths.base%/reports/Screen/Failed/'
13                path_succeeded: '%paths.base%/reports/Screen/Success/'
14            filters: { tags: '@ui' }
15
16    formatters:
17      pretty:
18        output_path: null
19      html:
20        output_path: '%paths.base%/reports'
21    extensions:
22      Behat\MinkExtension:
23        base_url: https://semainedunumerique.bj/
24        browser_name: firefox
25        default_session: goutte
26        show_cmd: 'open %s'
27        javascript_session: selenium2
28        goutte: ~
29        selenium2:
30          wd_host: http://localhost:4444/wd/hub
31          browser: firefox
32          capabilities: {'browser': 'firefox', 'marionette': true}

```

FIGURE 3.1 – Configuration behat.yml

3.2.2 Les fichiers de fonctionnalités

Les fichiers de fonctionnalités sont des fichiers avec une extension `.feature` contenant nos scripts écrits en langage naturel. Ils doivent être placés dans le dossier `features` de notre projet et le chemin doit être spécifié dans notre fichier de configuration. Nous testons quatre fonctionnalités sur `https://semainedunumerique.bj/` pour la présentation de notre prototype (voir figure 3.2)

Les fonctionnalités de :

- contact
- newsletter
- recherche
- inscription

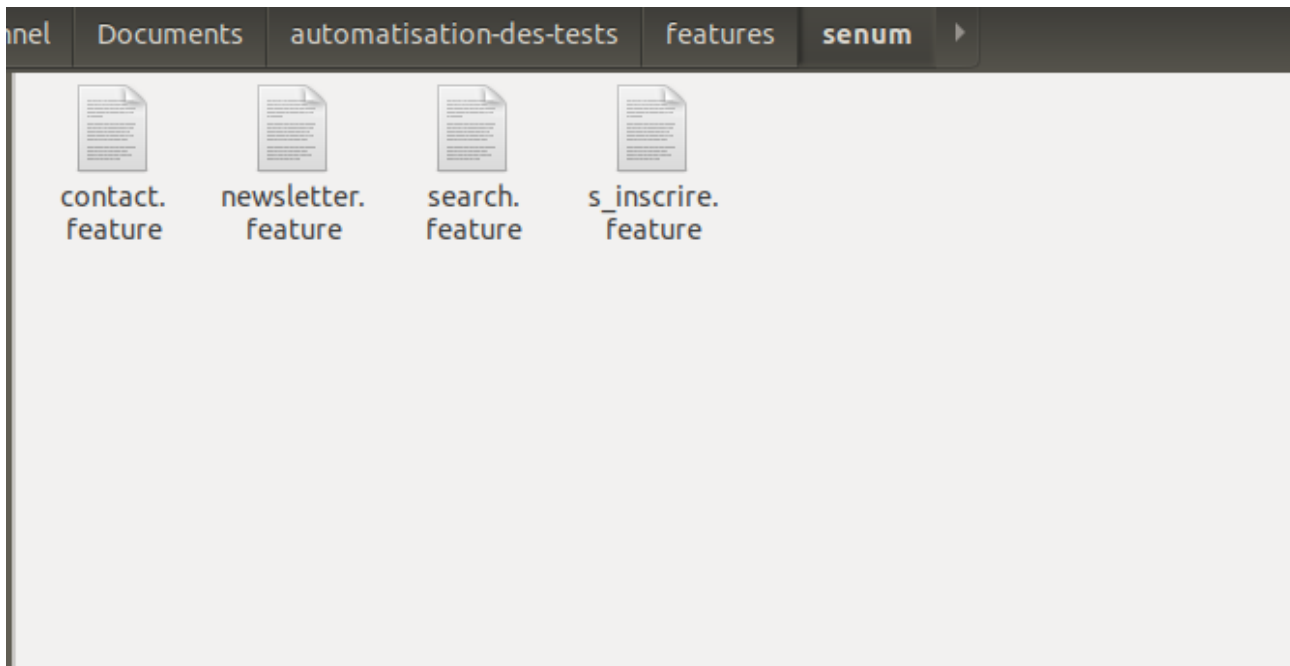


FIGURE 3.2 – Fichiers de fonctionnalités

- **contact.feature**

```

Features > senum > contact.feature
1  @ui
2  Feature: Contact
3
4  @javascript
5  Scenario: Devrait contacter mais retourne une erreur de recaptcha
6      Given I am on the homepage
7      When I wait for "2" seconds
8      And I hover over the link "Ressources"
9      And I wait for "2" seconds
10     And I click link with href "/ressources/contact"
11     And I should see "Ecrivez-nous"
12     And I wait for "3" seconds
13     And I fill the input of placeholder "Name" with the value "Arima Kosei"
14     And I fill the input of placeholder "Email" with the value "arimakosei@yourlie.com"
15     And I fill the input of placeholder "Phone" with the value "001 123456789"
16     And I fill the input of name "subject" with the value "Lorem ipsum..."
17     And I fill the textarea of placeholder "Message" with the value "Bla bla bla bla bla bla"
18     And I wait for "2" seconds
19     And I click input with id "btn-1484924351" and type "submit"
20     And I wait for "5" seconds
21     Then I should see "Champ invalide : reCAPTCHA"
22
23
24
  
```

FIGURE 3.3 – Contact.feature

- **newsletter.feature**

```

features > senum > ≡ newsletter.feature
1  @ui
2  Feature: Newsletter
3
4  @javascript
5  Scenario: Devrait inscrire à la newsletter
6      Given I am on the homepage
7      When I wait for "2" seconds
8      And I scroll to the bottom
9      And I fill the input of class "inputbox" with the value "jolivehodehou7@gmail.com"
10     And I click input with type "submit" and name "Submit"
11     And I wait for "2" seconds
12     Then I should see "Vous êtes déjà inscrit(e)."
13
14
15

```

FIGURE 3.4 – Newsletter.feature

- s_inscrire.feature

```

features > senum > ≡ s_inscrire.feature
1  @ui
2  Feature: S'inscrire
3
4  @javascript
5  Scenario: Devrait inscrire mais retourne une erreur de captcha
6      Given I am on the homepage
7      When I wait for "2" seconds
8      And I click link with id "btn-1501569977601" and href "/s-inscrire"
9      And I wait for "2" seconds
10     And I fill the input of placeholder "Votre nom" with the value "Hodehou"
11     And I fill the input of placeholder "Votre prénom" with the value "Jolivé"
12     And I fill the input of type "email" with the value "jolivehodehou7@gmail.com"
13     And I fill the input of id "fox-m131-textfield1" with the value "229 99 99 99 99"
14     And I fill the input of placeholder "Quelle est votre profession ?" with the value "QA & Test Automatio
15     And I fill the input of id "fox-m131-textfield3" with the value "IFRI"
16     And I wait for "2" seconds
17     And I click input with id "fox-m131-checkbox1" and type "checkbox"
18     And I click input with id "fox-m131-checkbox2" and type "checkbox"
19     And I click input with id "fox-m131-checkbox3" and type "checkbox"
20     And I wait for "2" seconds
21     And I click button with type "submit"
22     And I wait for "5" seconds
23     Then I should see "Champ invalide : reCAPTCHA"
24
25
26

```

FIGURE 3.5 – s_inscrire.feature

- search.feature


```
features > senum > search.feature
1  @ui
2  Feature: Hover Menu and search
3
4  @javascript
5  Scenario: Devrait survoler les menus et faire une recherche
6      Given I am on the homepage
7      When I wait for "2" seconds
8      And I hover over the link "Programme"
9      And I wait for "2" seconds
10     And I hover over the link "Challenge Numérique"
11     And I wait for "2" seconds
12     And I hover over the link "Ressources"
13     And I wait for "2" seconds
14     And I click link with id "offcanvas-toggler" and href "#"
15     And I wait for "5" seconds
16     And I fill the input of placeholder "Recherche..." with the value "Hackerlab"
17     And I wait for "2" seconds
18     And I click left mouse in input of placeholder "Recherche.."
19     And I press enter key in input of placeholder "Recherche.."
20     And I wait for "5" seconds
21     Then I should see "Rechercher "
```

FIGURE 3.6 – search.feature

3.2.3 Le dossier du formatter

Après avoir lancé nos scripts de tests en console notre formatter prends des captures d'écran automatique après chaque step pendant notre test.

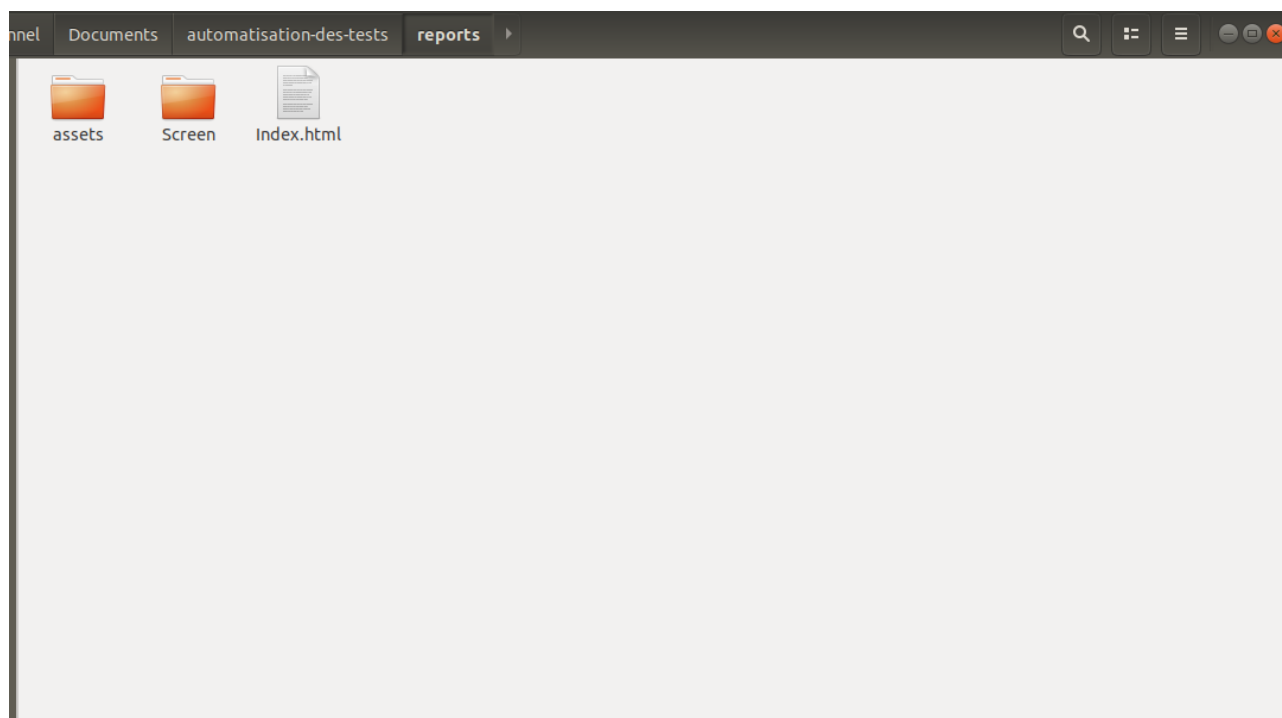


FIGURE 3.7 – Dossier report

Les captures d'écran sont dans le répertoire Screen de notre dossier reports dans notre projet. Ils sont classés dans deux autres sous répertoires success et failed. (Voir figure 3.8)

Success pour les tests qui se sont déroulés avec success et failed dans le cas contraire

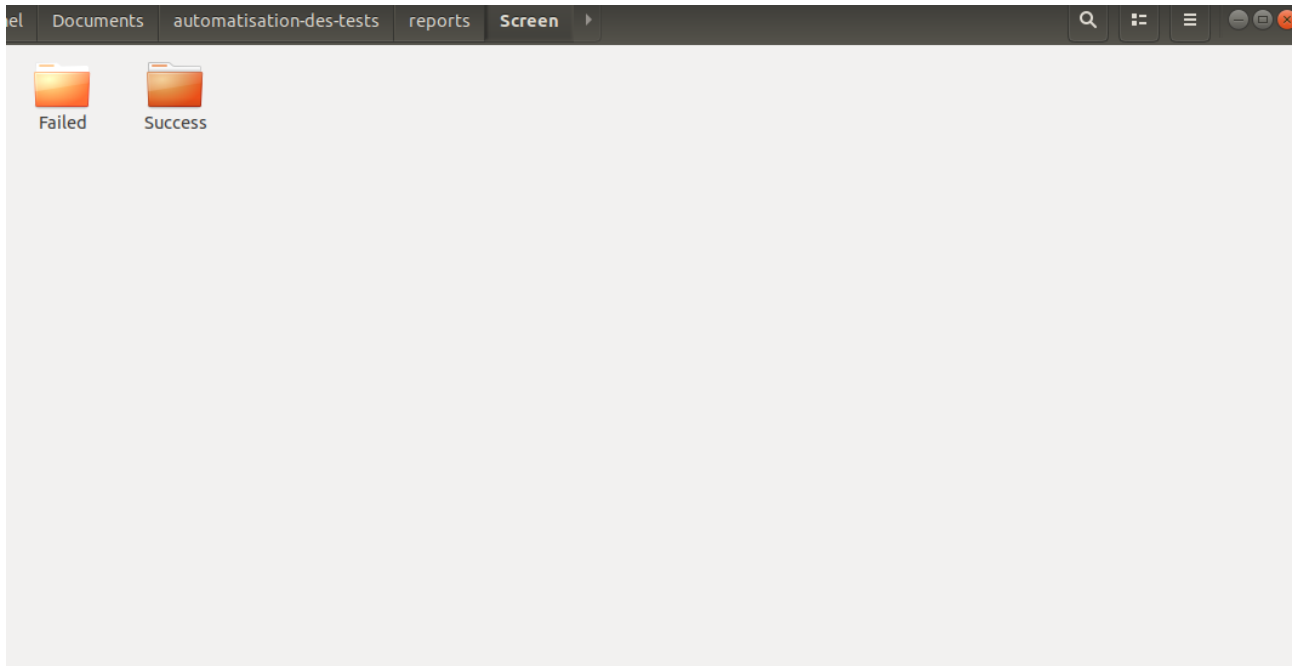


FIGURE 3.8 – Failed and Success

Voici ci-dessous le contenu de nos deux répertoires

Failed

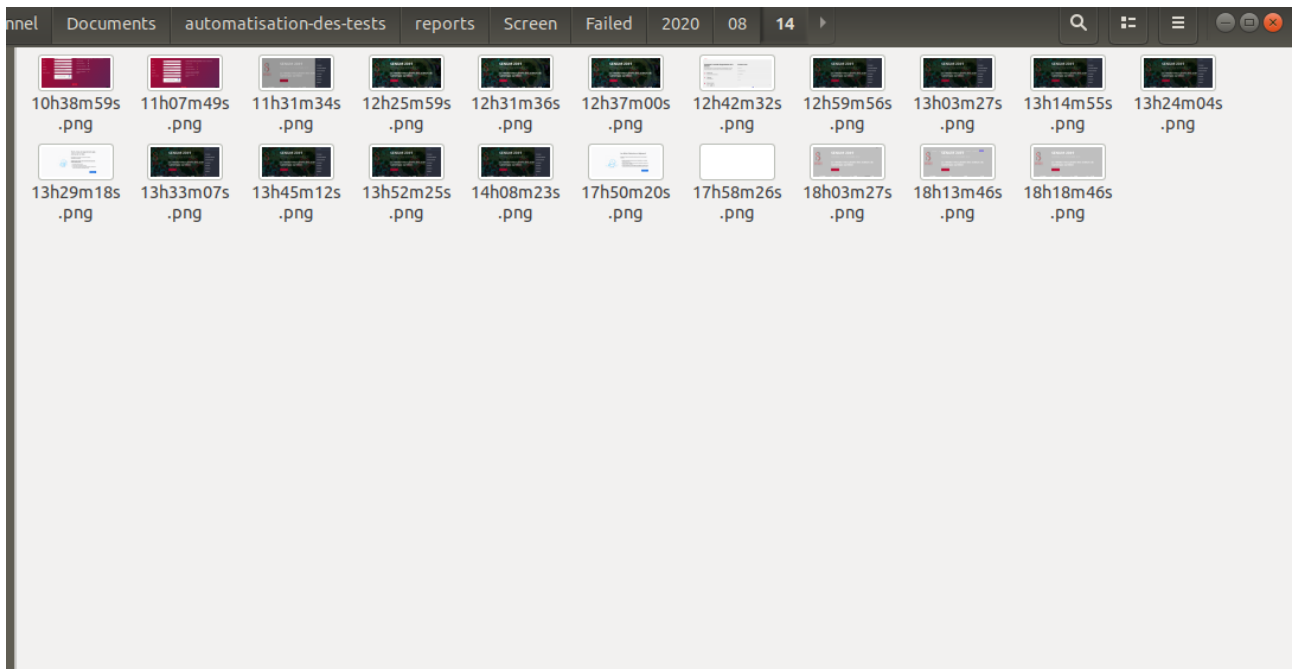


FIGURE 3.9 – Failed

success



FIGURE 3.10 – Success

Le fichier **index.html** à la racine du dossier report nous permet de visualiser le rapport de test.

3.3 Résultat des tests du site (console)

Pour lancer notre test, nous devons être sûrs que notre serveur selenium[1] est démarré et pour cela à la racine du projet tout tapons en console

./run.sh (voir figure 3.11)

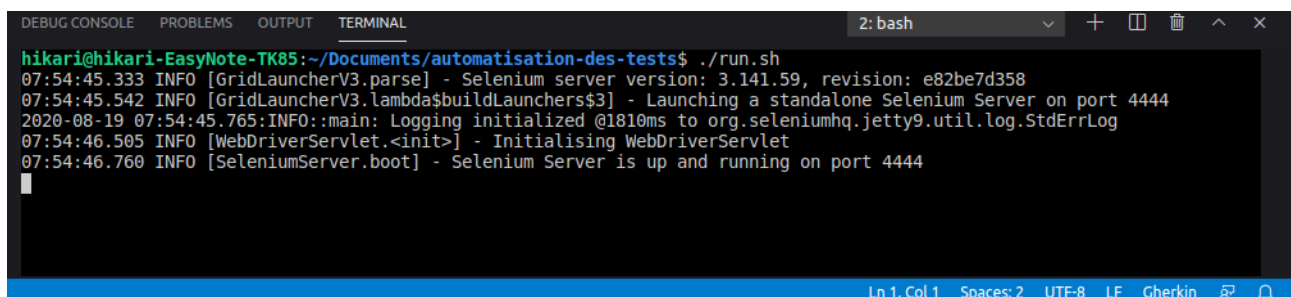
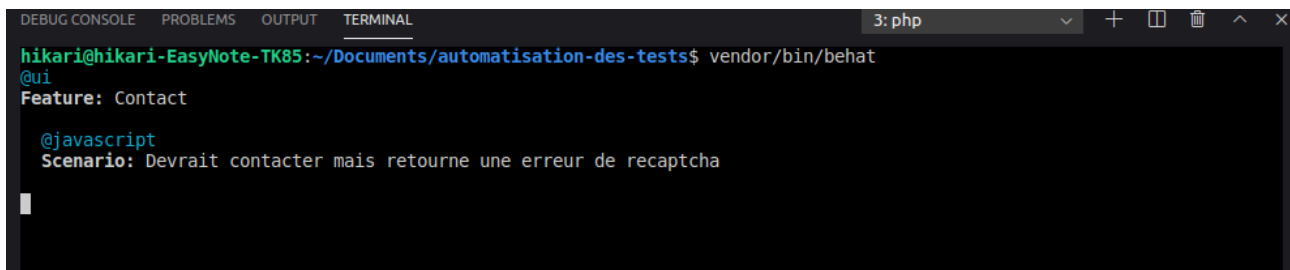


FIGURE 3.11 – Commande run.sh

Ensuite, à la racine du projet dans un nouvel onglet ou fenêtre, nous entrons en console **vendor/bin/behat** pour lancer le test (voir figure 3.12).



```
hikari@hikari-EasyNote-TK85:~/Documents/automatisation-des-tests$ vendor/bin/behat
@ui
Feature: Contact

  @javascript
  Scenario: Devrait contacter mais retourne une erreur de recaptcha
```

FIGURE 3.12 – Commande vendor/bin/behat

La figure 3.13 montre le navigateur dans lequel se déroule le test après la commande ci-dessus



FIGURE 3.13 – Site du senum

À la fin de notre test voici le résultat de notre test en mode console (voir figure 3.14)

```

hikari@hikari-EasyNote-TK85:~/Documents/automatisation-des-tests$ vendor/bin/behav
@ui
Feature: Contact

@javascript
Scenario: Devrait contacter mais retourne une erreur de recaptcha

    Given I am on the homepage
    When I wait for "2" seconds
    And I hover over the link "Ressources"
    And I wait for "2" seconds
    And I click link with href "/ressources/contact"
    And I should see "Ecrivez-nous"
    And I wait for "3" seconds
    And I fill the input of placeholder "Name" with the value "Arima Kosei"
    And I fill the input of placeholder "Email" with the value "behatselenium@gmail.com"
    And I fill the input of placeholder "Phone" with the value "001 123456789"
    And I fill the input of placeholder "Subject" with the value "Lorem ipsum..."
    And I fill the textarea of placeholder "Message" with the value "Bla bla bla bla bla bla bla"
    And I wait for "2" seconds
    And I click button with id "btn-1484924351" and type "submit"
    And I wait for "2" seconds
    Then I should see "Ecrivez-nous"

```

```

DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL  3: bash
@ui
Feature: Newsletter

@javascript
Scenario: Devrait inscrire à la newsletter

    Given I am on the homepage
    When I wait for "2" seconds
    And I scroll to the bottom
    And I fill the input of class "inputbox" with the value "jolivehodehou7@gmail.com"
    And I click input with type "submit" and name "Submit"

    And I wait for "2" seconds
    Then I should see "Vous êtes déjà inscrit(e)."
```

```

@ui
Feature: Hover Menu and search

@javascript
Scenario: Devrait survoler les menus et faire une recherche
    Given I am on the homepage
    When I wait for "2" seconds
    And I hover over the link "Programme"
    And I wait for "2" seconds
    And I hover over the link "Challenge Numérique"
    And I wait for "2" seconds
    And I hover over the link "Ressources"
    And I wait for "2" seconds
    And I click link with id "offcanvas-toggler" and href "#"
    And I wait for "5" seconds
    And I fill the input of placeholder "Recherche..." with the value "Hackerlab"
    And I wait for "2" seconds
    And I click left mouse in input of placeholder "Recherche..."

    And I press enter key in input of placeholder "Recherche..."

    And I wait for "5" seconds

```

Ln 21, Col 41 Spaces: 2 UTF-8 LF Gherkin

```
@ui
Feature: S'inscrire

@javascript
Scenario: Devrait inscrire mais retourne une erreur de recaptcha

    Given I am on the homepage
    When I wait for "2" seconds
    And I click link with id "btn-1501569977601" and href "/s-inscrire"
    And I wait for "2" seconds
    And I fill the input of placeholder "Votre nom" with the value "Hodehou"
    And I fill the input of placeholder "Votre prénom" with the value "Jolivé"
    And I fill the input of type "email" with the value "jolivehodehou7@gmail.com"
    And I fill the input of id "fox-m131-textfield1" with the value "229 99 99 99 99"
    And I fill the input of placeholder "Quelle est votre profession ?" with the value "QA & Test Automation Engineer"
    And I fill the input of id "fox-m131-textfield3" with the value "IFRI"
    And I wait for "2" seconds
    And I click input with id "fox-m131-checkbox1" and type "checkbox"
    And I click input with id "fox-m131-checkbox2" and type "checkbox"
    And I click input with id "fox-m131-checkbox3" and type "checkbox"
    And I wait for "2" seconds
    And I click button with type "submit"
    And I wait for "5" seconds
```

```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL  3: bash

    And I fill the input of placeholder "Votre prénom" with the value "Jolivé"
    And I fill the input of type "email" with the value "jolivehodehou7@gmail.com"
    And I fill the input of id "fox-m131-textfield1" with the value "229 99 99 99 99"
    And I fill the input of placeholder "Quelle est votre profession ?" with the value "QA & Test Automation Engineer"
    And I fill the input of id "fox-m131-textfield3" with the value "IFRI"
    And I wait for "2" seconds
    And I click input with id "fox-m131-checkbox1" and type "checkbox"
    And I click input with id "fox-m131-checkbox2" and type "checkbox"
    And I click input with id "fox-m131-checkbox3" and type "checkbox"
    And I wait for "2" seconds
    And I click button with type "submit"
    And I wait for "5" seconds
    Then I should see "Champ invalide : reCAPTCHA"

Duration of the test: 00:02:27

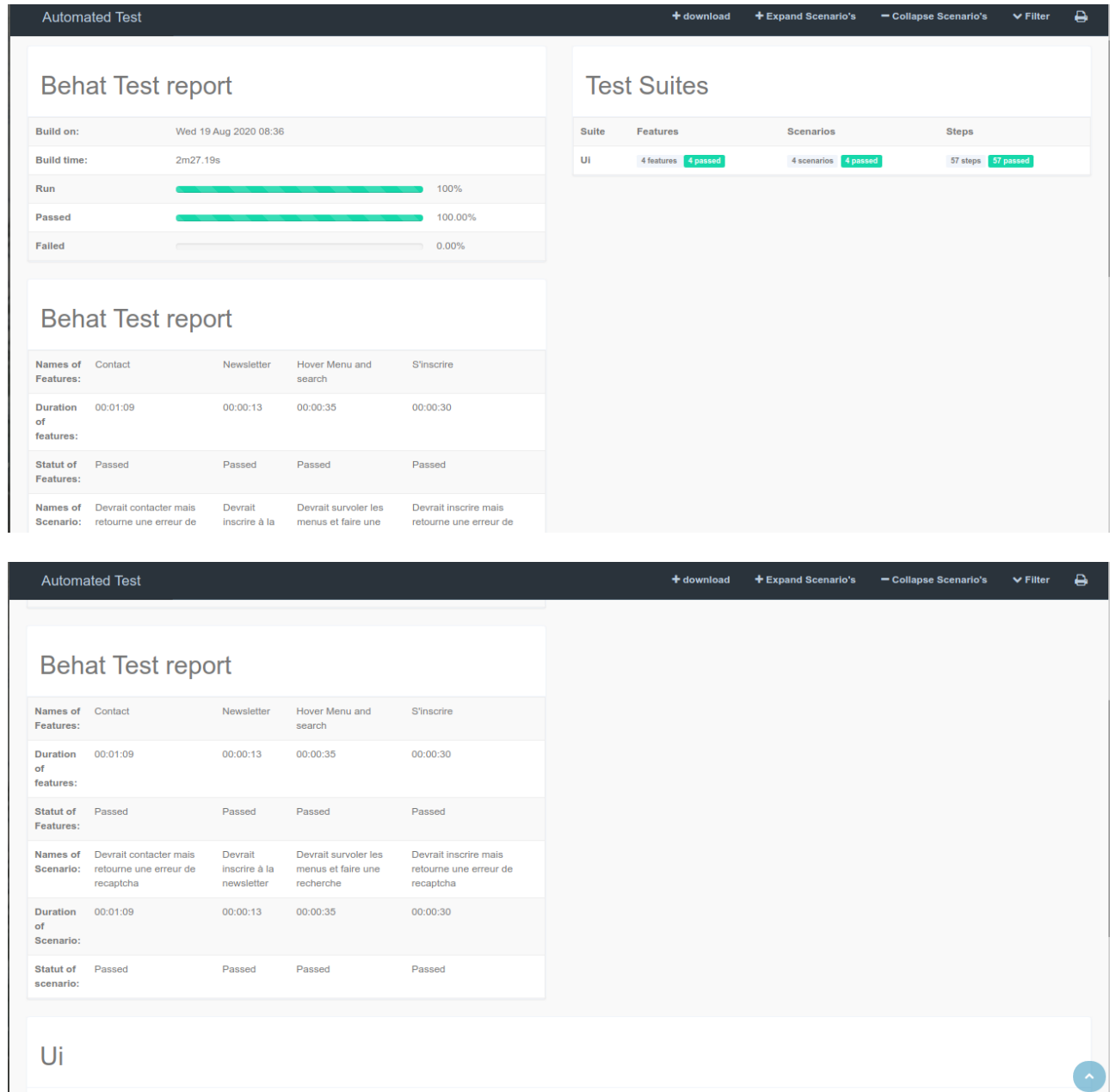
@AfterSuite

4 scénarios (4 succès)
57 étapes (57 succès)
2m27.18s (11.93Mb)
PHP Warning: rename(/home/hikari/Documents/automatisation-des-tests/.tmp_behatFormatter,/home/hikari/Documents/automatisation-des-tests/reports/assets/screenshots): No such file or directory in /home/hikari/Documents/automatisation-des-tests/vendor/eklan/behatformatter/src/Formatter/BehatFormatter.php on line 270
hikari@hikari-EasyNote-TK85:~/Documents/automatisation-des-tests$
```

FIGURE 3.14 – Résultats de test en console

3.4 Rapport Formatter

À la fin du test le formatter nous génère un rapport qui peut être imprimé ou télécharger en fichier csv (voir figure 3.15)



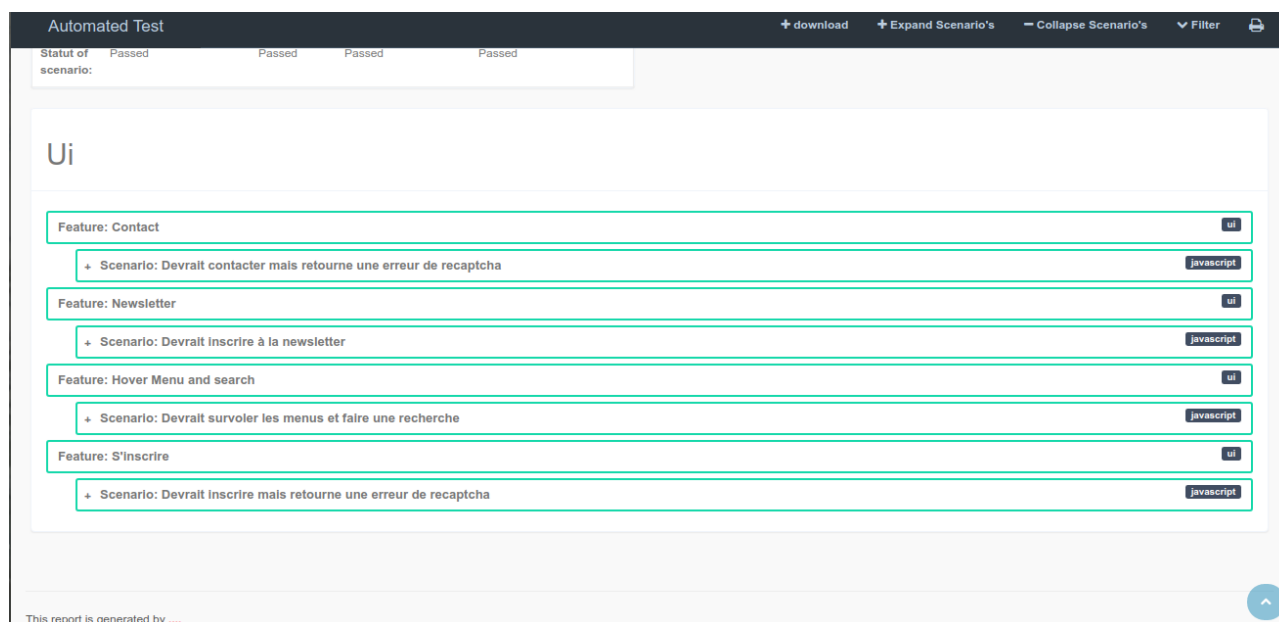


FIGURE 3.15 – Rapport du formatter

3.5 Discussion

Notre solution permet d'automatiser efficacement des tests fonctionnels sur les applications web développées en bootstrap et de générer un rapport de test qui peut être imprimé au format [CSV](#). Les phases de test du prototype nous ont permis de faire ressortir les insuffisances liées à notre approche. En effet au niveau de la simulation des interactions que peut réaliser un utilisateur, il ressort que la solution n'est pas en mesure de simuler certaines interactions avancées comme les reCAPTCHA. Ce qui n'est pas étonnant vu que c'est un processus de reconnaissance des formes par les robots et dont l'utilisation ne peut être simulée par un autre robot, mais que par des êtres humains. Par ailleurs compte tenu de la diversité des techniques web et du [DOM](#) dans le navigateur notre solution teste efficacement que les applications développées en bootstrap. Il est à noter que les tests peuvent uniquement être des tests cross browser et qu'il n'est pas en mesure de donner des informations sur les erreurs de développement et les détails sur le temps de compilation.

3.6 Conclusion

Ce chapitre, nous a permis de présenter des captures d'écrans de notre système et évoqué ses conditions d'utilisation. Pour finir, nous avons mis en évidence les limites de l'application. Les insuffisances liées à la solution seront utilisées comme base pour formuler les perspectives.

Conclusion générale et perspectives

L'automatisation des tests fonctionnels est de plus en plus répandue dans les entreprises. C'est une méthode efficace et qui fait ses preuves. De plus, malgré un coût d'entrée certain, ils permettent de faire gagner du temps et de la qualité dès les premières exécutions en s'assurant que les fonctionnalités clé de l'application sont fonctionnelles. Le but de notre travail était de mettre en place une solution d'automatisation des tests fonctionnels. Plus précisément, nous avons réalisé un outil basé sur le développement axé sur le comportement facile de compréhension et d'utilisation qui permet d'effectuer les tests fonctionnels sur les applications web à travers la rédaction de scénario pour chaque fonctionnalité et de générer des rapports de test lisible. En revanche, la solution n'est pas en mesure de simuler certaines interactions utilisateurs.

Des perspectives intéressantes sont envisagées pour ce travail. Par exemple, dans la vision d'une utilisation plus poussée, il serait intéressant d'écrire de nouveaux contextes et de mieux les explorer afin de pouvoir simuler beaucoup plus d'interaction. Par ailleurs, il est envisageable que dans ses prochaines versions, l'outil puisse effectuer des tests sur d'autres technologies web répandu dans l'industrie de l'informatique. Il serait intéressant de pouvoir automatiser les tests sur d'autres supports comme les téléphones mobiles. D'améliorer le rapport en y ajoutant d'autres informations comme les erreurs de développement et le temps de compilation grâce au moteur de templates Twig. D'autres perspectives seraient d'enregistrer ses informations dans une base de données avec le PHP Data Objects, de les faire importer et nettoyer, pour finalement rendre les données beaucoup plus exploitables et beaucoup plus utiles grâce à Talend Studio, éditeur spécialisé dans l'intégration de données.

Bibliographie

- [1] S. Avasarala. Selenium webdriver practical guide, interactively automate web applications using selenium webdriver. 2014.
- [2] E.-V. Berea. Master thesis, building web applications with behavior-driven development. 2017.
- [3] B. Danchilla. Behat cookbook : Delicious functional testing. 2017.
- [4] D. E. J. P. DONOU. Planification et evaluation numérique. 2017.
- [5] A. H. Ian Dees, Matt Wynne. Cucumber recipes : Automate anything with bdd tools and techniques (pragmatic programmers) 1st edition. 2013.
- [6] A. H. Matt Wynne. The cucumber book : Behaviour-driven development for testers and developers (pragmatic programmers) 1st edition. 2012.
- [7] D. Molina. Selenium fundamentals. 2018.
- [8] N. Palani. Advanced selenium web accessibility testing. 2019.
- [9] R. G. Rex Page. Essential logic for computer science. 2019.
- [10] J. F. Smart. Bdd in action : Behavior-driven development for the whole software lifecycle 1st edition. 2014.

Webographie

- Gitter : BDD framework for PHP 5.3+ , <https://gitter.im/Behat/Behat> consulté le 3 Août 2020
- Konstantin Kudryashov : A php framework for autotesting your business expectations. , <https://docs.behat.org/en/latest/> consulté le 20 Mai 2020
- Behat Community , <https://github.com/Behat> consulté le 25 Juillet 2020
- David Travis : Tests fonctionnels en Drupal 8 avec Behat , <https://blog.clever-age.com/fr/2018/03/08/tests-fonctionnels-en-drupal-8-avec-behat/> consulté le 25 Juillet 2020
- Site Officiel de Documentation PHP , <http://php.net/> consulté le 2 Août 2020
- Wikipedia Behavior Driven Development , https://en.wikipedia.org/wiki/Behavior-driven_development consulté le 25 Juillet 2020
- Mink for Behat , <http://mink.behat.org> consulté le 25 Juillet 2020
- Mink Extension , <https://github.com/Behat/MinkExtension> consulté le 25 Juillet 2020
- Cucumber Tools , <https://github.com/cucumber/cucumber/wiki/Gherkin> consulté le 25 Juillet 2020
- The flexible, fast, and secure template engine for PHP , <http://twig.sensiolabs.org/> consulté le 25 Juillet 2020
- Behatformatter, <https://github.com/ElkanRoelen/behatformatter> consulté le 25 Juillet 2020
- Build fast, responsive sites with Bootstrap, <https://getbootstrap.com/> consulté le 23 Juin 2020
- SENUM 2019, <https://semainedunumerique.bj/> consulté le 23 Juin 2020

Annexe

Cas de test sur le site officiel de bootstrap

Les fonctionnalités qui ont été testées.

- Overview
- Examples

Overview

```
@ui
Feature: overview

  @javascript
  Scenario: overview
    Given I am on the homepage
    When I click link with href "/docs/4.5/getting-started/introduction/"
    And I fill the input of placeholder "Search..." with the value "Jolivé Hodehou"
    Then I wait for "10" seconds
```

Examples

```

features > test_bootstrap > examples.feature
1  @ui
2  Feature: examples
3
4  Background:
5    Given I am on the homepage
6    And I click link with href "/docs/4.5/examples/" and class "nav-link "
7
8  @javascript
9  Scenario: Sign In
10   Given I wait for "5" seconds
11   When I click link with href "/docs/4.5/examples/sign-in/"
12   And I wait for "5" seconds
13   And I fill the input of placeholder "Email address" with the value "Jolivehodehou7@gmail.com"
14   And I wait for "2" seconds
15   And I fill the input of type "password" with the value "Mot de passe"
16   And I wait for "2" seconds
17   And I click input with value "remember-me"
18   And I wait for "2" seconds
19   And I click button with class "btn btn-lg btn-primary btn-block"
20   Then I wait for "2" seconds
21
22
23
24
25

```

Resultats en console

Overview

```

@ui
Feature: overview

@javascript
Scenario: overview

    Given I am on the homepage
    When I click link with href "/docs/4.5/getting-started/introduction/"
    And I fill the input of placeholder "Search..." with the value "Jolivé Hodehou"
    Then I wait for "10" seconds

    Duration of the test: 00:01:15

@AfterSuite
2 scénarios (2 succès)
17 étapes (17 succès)
1m14.77s (11.78Mb)

```

Examples

```

hikari@hikari-EasyNote-TK85:~/Documents/automatisation-des-tests$ vendor/bin/behat
@ui
Feature: examples

  Background:
    Given I am on the homepage
    And I click link with href "/docs/4.5/examples/" and class "nav-link "

@javascript
Scenario: Sign In

  Given I wait for "5" seconds
  When I click link with href "/docs/4.5/examples/sign-in/"
  And I wait for "5" seconds
  And I fill the input of placeholder "Email address" with the value "Jolivehodehou7@gmail.com"
  And I wait for "2" seconds
  And I fill the input of type "password" with the value "Mot de passe"
  And I wait for "2" seconds
  And I click input with value "remember-me"
  And I wait for "2" seconds
  And I click button with class "btn btn-lg btn-primary btn-block"
  Then I wait for "2" seconds

```

Rapport Formatter

Automated Test + download + Expand Scenario's - Collapse Scenario's ▼ Filter

Behat Test report

Build on:	Tue 08 Sep 2020 15:52	
Build time:	1m14.79s	
Run	<div><div></div></div>	100%
Passed	<div><div></div></div>	100.00%
Failed	<div><div></div></div>	0.00%

Behat Test report

Names of Features:	examples	overview
Duration of features:	00:00:46	00:00:29
Statut of Features:	Passed	Passed
Names of Scenario:	Sign In	overview
Duration of Scenario:	00:00:46	00:00:29
Statut of scenario:	Passed	Passed

Test Suites

Suite	Features	Scenarios	Steps
UI	2 features 2 passed	2 scenarios 2 passed	17 steps 17 passed

Table des matières

Dédicace	ii
Remerciements	iii
Résumé	iv
.....	iv
Abstract	v
.....	v
Liste des figures	vi
Liste des tables	vii
Liste des acronymes	viii
Introduction	1
1 Revue de littérature	2
Introduction	2
1.1 Introduction	2
1.2 Généralité sur les tests en informatique	2
1.2.1 Qu'est-ce qu'un test en informatique	2
1.2.2 Quels sont les types de tests en informatique	2
1.2.2.1 Tests fonctionnels	3
1.2.2.1.1 Test unitaire :	3
1.2.2.1.2 Test d'intégration :	3
1.2.2.1.3 Test de système :	3
1.2.2.1.4 Test d'intégrité :	3
1.2.2.1.5 Test de fumée :	3
1.2.2.1.6 Test d'interface :	3
1.2.2.1.7 Test de régression :	3
1.2.2.2 Tests Non Fonctionnels	3
1.2.2.2.1 Test de performance :	3
1.2.2.2.2 Test de charges :	3
1.2.2.2.3 Test de contraintes :	4
1.2.2.2.4 Test de volume :	4
1.2.2.2.5 Test de sécurité :	4
1.2.2.2.6 Test de compatibilité :	4

1.2.2.2.7	Test d'installation :	4
1.2.2.2.8	Test de récupération :	4
1.2.2.2.9	Test de fiabilité :	4
1.2.2.2.10	Test de d'utilisabilité :	4
1.2.2.2.11	Test de conformité :	4
1.2.2.2.12	Test de localisation :	4
1.3	Généralité sur les tests automatisés	5
1.3.1	Qu'est-ce qu'un test automatisé	5
1.3.2	Qu'est le BDD, le TDD, le DDD et l'ATDD	5
	BDD :	5
	DDD :	5
	TDD :	5
	ATDD :	5
1.3.3	L'enjeu des tests automatisés	5
1.3.3.1	Quel est le but des tests automatisés	5
1.3.3.2	Test Manuel et Test Automatisés	6
1.4	Problématique	7
1.4.1	Scénario : Maintien d'un site avec un cahier des charge de près de 500 pages et 200 fonctionnalités	7
1.5	Objectif Général	7
1.6	Étude de l'existant	7
1.6.1	Présentation des solutions existantes	7
1.6.1.1	Selenium	7
1.6.1.2	Watir	7
1.6.1.3	Soap UI	7
1.6.1.4	Cucumber	8
1.6.2	Résumé des exigences de notre système	8
Conclusion		8
1.7	Conclusion	9
2	Analyse, choix techniques et conception	10
	Introduction	10
2.1	Introduction	10
2.2	Analyse	10
2.2.1	Architecture du Système	10
2.2.2	Entités du système	11
2.2.2.1	Les développeurs	11
2.2.2.2	Les QA testeurs	12
2.3	Choix technique	12
2.3.1	Choix du langage de programmation et du framework	12
2.3.1.1	PHP	12
2.3.1.2	Behat	13
2.3.2	Choix du serveur Sélénium et des drivers navigateurs	13
2.3.3	Choix du Formatter de données (Elkan)	15

2.4	Configuration du projet	16
2.4.1	Environnement projet - Behat	16
2.4.2	Ecritures et processus des tests	18
	Conclusion	19
2.5	Conclusion	19
3	Résultats et discussion	20
	Introduction	20
3.1	Introduction	20
3.2	Environnement du projet	20
3.2.1	Le fichier de configuration behat yml	20
3.2.2	Les fichiers de fonctionnalités	21
3.2.3	Le dossier du formatter	24
3.3	Résultat des tests du site (console)	26
3.4	Rapport Formatter	30
3.5	Discussion	31
	Conclusion	31
3.6	Conclusion	31
	Conclusion	32
	Bibliographie	33
	Webographie	34
	Annexe	35
	Table des matières	38
