

FACULTÉ DES SCIENCES ET TECHNIQUES MOHAMMEDIA

RAPPORT DE STAGE

3ème année FI : Génie Mathématiques et informatique Année universitaire : 2019/2020

CONCEPTION ET DÉVELOPPEMENT DE TESTS AUTOMATIQUES (NON RÉGRESSION, SÉCURITÉ ET PERFORMANCE) ET SON INTÉGRATION DANS LA CHAÎNE CI/CD DE GitLab

Réalisé par :

BENAMI Anas [élève ingénieur]

Encadré par :

M. KHACHAFI Elmahdi [à remplir]

M. Khadir Omar [Professeur à la FST de Mohammedia]

Dédicace

Au nom du dieu, celui qui fait miséricorde le
miséricordieux

Je dédie ce travail à ma famille qui ne cesse pas de m'encourager et me donner le soutien moral et matériel, à mes amis et à tous ceux qui ont croisé mon chemin et laisser de belles traces sur ma vie.

Remerciements

Je tiens à remercier chaleureusement, mon encadrant M. **KHACHAFI Elmahdi**[poste à ajouter], pour le temps qu'il a consacré à m'apporter les outils méthodologiques indispensables à la conduite de ce modeste travail . Son exigence m'a grandement stimulé.

Mes vifs remerciements vont également à M. **Khadir Omar**[Professeur à la FST de Mohammedia] pour sa patience, ses conseils pleins de sens et pour le suivi et l'intérêt qu'il a porté à mon travail. L'ensemble des camarades de classe pour leurs intérêts et leurs encouragements.

Je remercie également les membres du jury pour l'intérêt qu'ils ont porté à mon projet en acceptant d'examiner mon travail et de l'enrichir par leurs propositions. Aussi mes remerciements au corps professoral et administratif de la faculté des sciences et techniques de Mohammedia qui déploient de grands efforts pour nous assurer une très bonne formation.

Table des matières

1	Cadre Général	10
1.1	Introduction :	10
1.2	Cadre du travail :	10
1.3	Présentation de l'organisme d'accueil	10
1.4	Étude préalable	12
1.4.1	Présentation de la problématique et proposition de solution	12
1.4.2	Objectif de l'étude	13
1.5	Plan de travail	13
1.5.1	Organisation du rapport	13
1.5.2	Diagramme de Gantt	14
1.6	Conclusion	14
2	Conception et développement des tests	15
2.1	Introduction	15
2.2	Les tests de non régression	15
2.2.1	Définition :	15
2.2.2	Les technologies :	16

2.2.2.1	Cucumber	16
2.2.2.2	Selenium	16
2.2.2.3	Langage de programmation	16
2.2.2.4	Environnement de développement	16
2.3	Conception	17
2.3.1	Méthodologie de conception	17
2.3.2	Architecture	17
2.3.3	Génération des rapports :	19
2.4	Les tests de sécurité	21
2.4.1	Les techniques typiques de test de sécurité d'application :	21
2.4.1.1	DAST [Dynamic Application Security Testing]	21
2.4.1.2	Les outils de DAST :	22
2.4.1.3	Nikto :	22
2.4.1.4	Wapiti :	23
2.4.2	Application :	23
2.4.2.1	Premier cas : un site vulnérable	23
2.4.2.2	Deuxième cas : La plateforme Rokhas	25
2.4.2.3	SAST [Static Application Security Testing]	26
2.4.2.4	L'outil de SAST :	26
2.4.2.5	Application :	26
2.5	Les tests de performance	33
2.5.1	Définition :	33
2.5.2	Généralités :	33
2.5.3	Les risques :	33
2.5.4	Outils de test :	34

2.6	Conclusion	34
3	L'intégration dans la chaine CI/CD de GitLab	35
3.1	Introduction	35
3.2	Technologies	35
3.3	Définitions	36
3.3.1	L'intégration continue [CI] :	36
3.3.2	La livraison "Delivery"/Déploiement continue [CD] :	37
3.3.2.1	La différence entre déploiement continu et livraison continue . .	37
3.4	Principe de fonctionnement	38
3.4.1	Le fichier de configuration :	38
3.5	Exemples d'intégration des tests :	39

Table des figures

2.1	Diagramme de Classe [Architecture]	17
2.2	Scenario Utilisateur entre une date de naissance invalide [Plateforme Rokhas]	18
2.3	Rapport du scénario [Attachements des documents PH]	20
2.4	Rapport des scénarios de [Localisation du projet]	20
2.5	Rapport d'un scénario qui a échouer [Capture d'écran]	21
2.6	Output de la commande de test de vulnérabilités [Wapiti]	24
2.7	Rapport généré par Wapiti	24
2.8	Rapport généré par Nikto	25
2.9	Rapports Plateforme Rokhas [0 vulnérabilités]	25
2.10	HOME page SonareQube	27
2.11	Aperçu SonareQube	28
2.12	Problèmes SonareQube	29
2.13	Les Points d'accès de sécurité SonareQube	29
2.14	Quality Gates SonareQube	31
2.15	Rules SonareQube	31
2.16	Scan du code source de l'entreprise	32

3.1	Intégration/Livraison/Déploiement Continue	37
-----	--	----

Introduction Générale

De nos jours le développement des applications et logiciels sont en constante évolution. Les entreprises pensent alors à gagner en temps et en performances afin d'automatiser la procédure de tests logiciels, car la phase de test est une étape incontournable de tout développement informatique, elle a pour objectif de vérifier que le livrable répond bien aux besoins exprimés par l'utilisateur.

Un test est un ensemble de cas à tester pour effectuer une vérification partielle d'un système. Il vise à trouver des bugs. Le but principal est d'identifier un nombre maximum d'anomalies du logiciel dans une courte durée avant livraison des produits aux clients. La qualité sera donc augmentée lorsque les problèmes seront corrigés.

Les tests représentent 30 à 50% du coût de développement, et pour les faire chaque fois d'une manière manuelle c'est lourd premièrement, et deuxièmement il prend beaucoup du temps et risquer car il se fait par un être humain, ceci est un grand problème. L'automatisation des tests est une solution qui permet de réduire la charge de travail et le temps passé à exécuter des tests de logiciels. C'est à dire gain en temps et en productivité.

Elle permet aux testeurs de se concentrer sur les tests des nouvelles fonctionnalités de l'application avec une meilleure fiabilité ainsi que de minimiser les risques d'erreurs.

L'automatisation des tests logiciels présente de nombreux avantages :

- Une vitesse incroyable : Les tests automatisés font des merveilles en vérifiant chaque millimètre. Dans de nombreux cas, il faudrait une éternité pour vérifier les mêmes choses manuellement (si ce n'est carrément impossible).

- Réutilisables : Une fois écrit par un ingénieur QA (Quality Assurance), les tests peuvent être utilisés encore et encore, à l'infini. Les mêmes modules peuvent être réutilisés pour d'autres tests sur le projet.
- Excellente couverture : Grâce à l'automatisation, On peut couvrir un grand nombre de variantes de cas de test. Cela inclut l'interaction avec plusieurs systèmes d'exploitation, navigateurs etc., ainsi que divers scénarios de comportement d'utilisateur, et bien plus encore.
- Rapports pratiques : La génération des journaux de tests prêts qui listent précisément tous les tests effectués et les bugs trouvés.
- Autosuffisance : Les tests automatisés peuvent être exécutés 24 heures sur 24, 7 jours sur 7, sans surveillance, puis vous présenter tous les résultats du test.

Cadre Général

1.1 Introduction :

Ce chapitre est consacré à la présentation de l'organisme d'accueil, la précision du cadre du projet et la problématique puis l'énumération des étapes de travail à réaliser pour achever ce projet.

1.2 Cadre du travail :

Ce stage s'inscrit dans le cadre d'un projet de fin d'études pour l'obtention du diplôme d'ingénieur mathématiques informatique de la Faculté des Sciences et Techniques Mohammedia. Mon stage a été effectué au sein d'une Société de conseil opérationnel en systèmes d'information (Ribatis). Le sujet est intitulé Conception et développement de tests automatiques (non régression, sécurité et performance) et son intégration dans la chaîne CI/CD [l'intégration continue /livraison et déploiement continue] de GitLab.

1.3 Présentation de l'organisme d'accueil

RIBATIS est une entreprise de conseil opérationnel en systèmes d'information. Fondée en 2007, elle dispose d'un positionnement innovant alliant le recul des cabinets de conseil et le

1.3. PRÉSENTATION DE L'ORGANISME ~~ONAF~~ **Cadre Général**

pragmatisme des sociétés d'ingénierie informatique. RIBATIS propose trois familles de services : Conseil, Technologie et Formation.

RIBATIS a réussi le déploiement de plusieurs solutions technologiques à forte valeur ajoutée sur le marché, notamment E-DMAJ, un ERP dédié à la gestion des activités de l'entreprise au quotidien et ATHLETIS un système d'information spécialisé dans la gestion des fédérations, ligues et clubs sportifs.

Actuellement l'entreprise, basée à Casablanca, emploie une équipe d'une vingtaine d'ingénieurs et consultants fonctionnels haut niveau.

□ Métier :

La mission au quotidien est de veiller au bonheur de l'utilisateur et du fonctionnaire en leur offrant une expérience digitale exceptionnelle. A travers les plateformes digitales qu'ils réalisent, ils visent une transformation radicalement positive de la relation citoyen/administration, en la convertissant en une relation de collaboration plutôt qu'une relation de contrainte.

□ Approche :

L'approche est simple : Proposer les Services e-Gov sous forme d'une succession d'étapes faisant intervenir tour à tour le fonctionnaire et l'utilisateur.

Ainsi, au lieu d'avoir deux systèmes distincts : Le 1er servant à soumettre la demande usager (Interface externe) et le 2ème permettant à l'administration de la traiter (Interface interne), nos plateformes les fusionnent en un seul et même e-service. Ceci permet au fonctionnaire et à l'utilisateur de collaborer autour d'un même processus avec comme objectif commun de mener à bien la procédure, tout en respectant la réglementation en vigueur, les limites de responsabilité et la traçabilité des actions.

□ Les plateformes :

- **CASAURBA** : Lancé en octobre 2014 et généralisée à la région Casablanca-Settat à fin 2015, Casaurba est une plateforme digitale collaborative assurant la gestion 100% dématérialisée des autorisations d'urbanisme de +36 communes

- **ROKHAS** : Lancée en Avril 2017, la plateforme Rokhas dématérialise la délivrance des autorisations économiques, en assurant l'équilibre entre développement économique et respect de l'environnement et de la vie en société.
- **CasaOpenData** : Portail de données statistiques ouvertes, alimenté à partir des plateformes digitales Casaurba et Rokhas et publiant des indicateurs précis, actualisés au quotidien et conçus en tant qu'outil d'aide à la décision et d'amélioration continue.
- **KAFC/Jibayat** : La Plateforme Karaz Administration Fiscale Communale (KAFC) est une plateforme de gestion globale et intégrée de l'ensemble des taxes, droits et redevances collectées au niveau des communes.
- **GO HUB** : La plateforme Geohub.ma permet aux institutions, aux entreprises et aux individus de créer facilement et gratuitement leurs propres géoportails. Ils peuvent ainsi, créer, publier, actualiser et valoriser leurs données géographiques.
- **PARAPHEURB** : Parapheur.ma est une plateforme cloud dédiée à la transformation digitale des circuits de signature au sein des administrations publiques et privées.

1.4 Étude préalable

1.4.1 Présentation de la problématique et proposition de solution

Souvent on commence par la conception, le développement, le test puis le déploiement d'une application sur un serveur, après un certain temps, on se rend compte qu'il faut ajouter de nouvelles fonctionnalités à notre application. En général ces nouvelles fonctionnalités ont une forte relation avec les fonctionnalités existantes, et par conséquent il y a une forte probabilité que ces dernières soient affectées de manière inattendue.

La solution est de tester l'intégralité de l'application après chaque modification du code, Pour le faire de manière manuelle, c'est très difficile et prendra beaucoup de temps et risque d'oublier de ne pas tester quelque chose, donc la meilleure solution est l'automatisation des tests.

1.4.2 Objectif de l'étude

- Réduire le délai de livraison des produits (déploiement des applications dans les serveurs de production).
- Réduire la charge de travail et le temps passé à exécuter des tests fonctionnelles.
- Couvrir un grand nombre de variantes de cas de test.
- Repérer rapidement un bug pendant les cycles de développement.
- Éviter au maximum les erreurs humaines.

1.5 Plan de travail

1.5.1 Organisation du rapport

Pour un bon travail il faut un rapport bien structuré qui peut être exploité après la mise en place de ces tests, pour cela le présent rapport sera organisé de comme suit :

Dans le premier chapitre, on a met notre projet dans son cadre général en définissant la société d'accueil et en présentant le sujet avec une étude préalable.

Dans le deuxième chapitre intitulé Conception et développement des tests, nous allons présenter en premier lieu les tests de (non régression, sécurité et performance) et en deuxième lieu le principe de fonctionnement et les outils de chaque test.

Enfin et au niveau du troisième et dernier chapitre intitulé L'intégration dans la chaine

CI/CD de GitLab, on vas présenter les différentes composantes de cette chaine et son rôle dans l'automatisation.

1.5.2 Diagramme de Gantt

Le diagramme de Gantt est un outil de planification des tâches nécessaires pour la réalisation d'un projet quel que soit le secteur d'activité. Il permet de visualiser l'avancement des tâches d'un projet de manière simple et concise, de planifier et suivre les besoins en ressources humaines et matérielles et donc de pouvoir suivre l'avancement du projet.

Le diagramme suivant va représenter les tâches principales à réaliser dans le projet.

Pas Encore

1.6 Conclusion

Dans ce chapitre, nous avons présenté l'organisme d'accueil encerclant le cadre d'étude, la problématique ainsi que les objectifs. Nous verrons comment implémenter ces différents types de test dans le chapitre suivant.

Conception et développement des tests

2.1 Introduction

Les systèmes d'information sont amenés à évoluer régulièrement en raison de la nécessité d'intégrer de nouvelles fonctionnalités, de mettre à jour les évolutions à la demande des différentes entités métiers de l'entreprise. Les stratégies digitales accélèrent encore un peu plus les besoins de transformation des systèmes d'information qui doivent intégrer, de plus en plus, de fonctionnalités hétérogènes dans des délais de plus en plus brefs.

Comment dans ces conditions assurer la fiabilité des systèmes d'information ?

En informatique aussi, *un battement d'ailes d'un papillon peut provoquer une tornade* **La théorie de chaos**, il faut donc tester afin de vérifier que les modifications n'ont pas apporté d'instabilités ou d'anomalies, aux conséquences parfois imprévisibles.

2.2 Les tests de non régression

2.2.1 Définition :

Le **test de non régression** est défini comme un type de test de logiciel pour s'assurer que les nouvelles modifications de code ne devraient pas avoir d'effets secondaires sur les fonc-

tionnalités existantes. Il garantit que l'ancien code fonctionne toujours une fois les dernières modifications de code effectuées.

2.2.2 Les technologies :

2.2.2.1 Cucumber



est un outil qui prend en charge le développement basé sur le comportement (BDD). Il offre un moyen d'écrire des tests que tout le monde peut comprendre, quelles que soient leurs connaissances techniques.

2.2.2.2 Selenium



est un framework de test informatique développé en Java. Il permet d'interagir avec différents navigateurs web de même que le ferait un utilisateur de l'application. Il entre ainsi dans la catégorie des outils de test dynamique facilitant le test fonctionnel .

2.2.2.3 Langage de programmation



est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems.

La société Sun a été ensuite rachetée en 2009 par la société Oracle.

2.2.2.4 Environnement de développement



Est un environnement de développement intégré technologie Java destiné au développement de logiciels informatiques. Il est développé par JetBrains.

2.3 Conception

2.3.1 Méthodologie de conception

Pour faciliter la tâche on va utiliser le langage de modélisation unifié (UML : Unified Modelling Language) c'est une notation qui permet de modéliser un problème de façon standard. Ce langage est né de la fusion de plusieurs méthodes existantes auparavant, et il est devenu un référence en terme de modélisation objet, à un tel point que sa connaissance devient indispensable pour un développeur.

2.3.2 Architecture

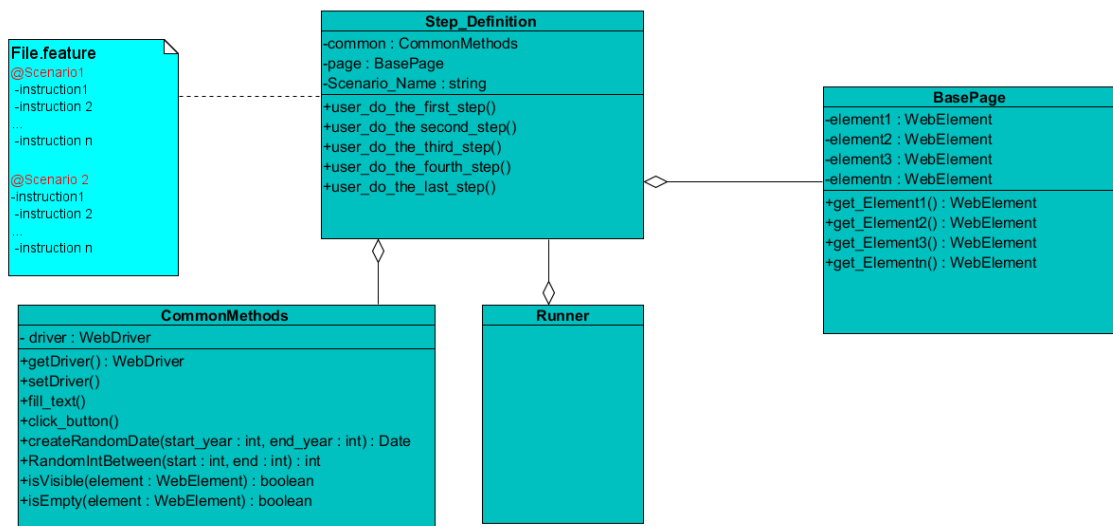


FIGURE 2.1 – Diagramme de Classe [Architecture]

D'abord on commence par la création d'un fichier avec l'extension **.feature** de Cucumber, dans lequel on écrit nos scénarios avec n'importe quelle langue (L'anglais, Le francais...), chaque scénario est une liste d'étapes à suivre par Cucumber. Pour que Cucumber comprenne les scénarios, ils doivent suivre certaines règles de syntaxe de base, appelées Gherkin.

2.3. CONCEPTION

Chapitre 2. Conception et développement des tests

Exemple :

Dans cet exemple on test le scénario dans lequel l'utilisateur saisi une date de naissance invalide (Dans le future).

J'ai choisi d'écrire le scénario en anglais, les mots en bleu sont les données de test. Dans le champs nom il va saisir Mahjoubi et pour le prénom il va écrire Fahd etc ...

MAÎTRE D'OUVRAGE :

☒ PERSONNE PHYSIQUE ☐ PERSONNE MORALE ☐ ADMINISTRATION PUBLIQUE

Maître d'ouvrage existant : - ? x Qualité du maître d'ouvrage : ?

En choisissant un client de votre portfolio, ses données seront automatiquement affichées. La qualité du maître d'ouvrage choisie influence la liste des documents exigés.

Civilité *	Nom *	Prénom *
Nom du père *	Nom de la mère *	
CIN/Passeport/Titre de séjour *	Date fin validité pièce identité *	Lieu délivrance pièce identité *
Nationalité	Date de naissance *	Lieu de naissance *
Maroc		
Profession	Téléphone *	Email
Adresse *	Ville *	Pays * Maroc

Réinitialiser Valider

```
@InvalidDateOfBirth
Scenario: Invalid date of birth
  And User enter his Quality
  And User Enter his Civility
  And User enter his last_name "Mahjoubi"
  And User enter his first_name "Fahd"
  And User enter his father's name "Mohammed"
  And User enter his mother's name "Fatima"
  And User enter his CIN "0890123"
  And User enter his expiration date of CIN
  And User enter his city of delivery of CIN "Casablanca"
  And User enterhis date of birth day
  And User entre his place of birth "Casablanca"
  And User enter his phone number "0666871394"
  And User enter his email "fahd78@gmail.com"
  And User enter his address "hay el ouahda casablanca 1987"
  And User enter his city's name "Casablanca"
  Then User Clicks the submit button!
```

FIGURE 2.2 – Scenario Utilisateur entre une date de naissance invalide [Plateforme Rokhas]

2.3. CONCEPTION

Chapitre 2. *Conception et développement des tests*

La deuxième chose, on crée la classe **BasePage** (C'est une classe Java qui contient comme attributs les éléments web [Bouton, inputText, checkBox... etc] de la page web qu'on veut tester) on récupère les éléments web à l'aide du framework Selenium.

Nb : Dans cet exemple les éléments sont les attributs du formulaire.

Après on crée la classe **Step Definition** qui est le coeur de notre travail. Dans cette classe on crée, pour chaque étape (instruction) du scénario, une méthode qui traduit les mots en code java en utilisant le framework Selenium .

En fin on crée la classe **Runner** qui est responsable de lancement des scénarios à exécuter, En différencier entre les scénarios par l'annotation @ Entête de chaque scénario, dans notre cas le scénario s'appelle InvalidDateOfBirth.

Nb : On peut lancer le nombre de scénarios qu'on veut.

La classe **Commond Methodes** juste pour garder la notion du clean code.

2.3.3 Génération des rapports :

Cucumber nous aide beaucoup, car il nous donne la possibilité de générer des rapports avec différentes formats (HTML, json, xml ...) après chaque exécution du test.

Exemple : Un seul scénario

Comme vous voyez, on a les graphiques qui nous indiquent par pourcentage, le taux des étapes réussies [en vert], échouées [en rouge] et les étapes ignorées [en bleu].

Il nous liste aussi tous les étapes du scénario et indique devant chaque étape [Pass, Fail ou bien Skip].

2.3. CONCEPTION Chapitre 2. Conception et développement des tests

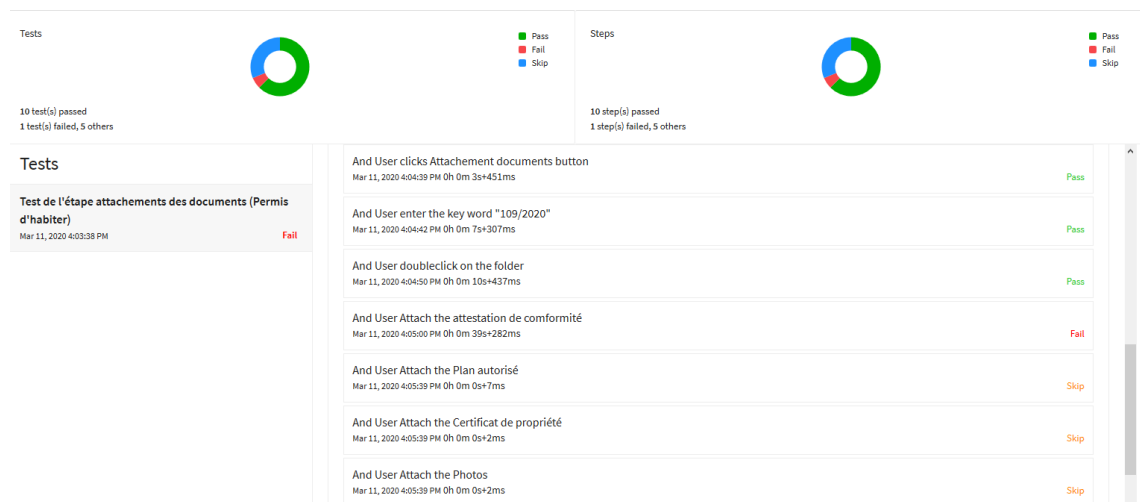


FIGURE 2.3 – Rapport du scénario [Attachements des documents PH]

NB : une fois Cucumber rencontre une étape échouée, il ignore toutes les étapes qui la suivent.

Exemple : Plusieurs scénarios

Search		
Categories		@NatureProjet(Type1)
@NatureProjet(Type1)		Passed: 1
	@NatureProjet(Type2)	1
	@NatureProjet(Type3)	1
	@RéférencefoncièreExistant	1
	@RéférencefoncièreExistant2	1
		Timestamp TestName Status
		Feb 18, 2020 12:26:56 PM Test de la partie Localisation projet.Scenario: Nature of Project Type1 Pass

FIGURE 2.4 – Rapport des scénarios de [Localisation du projet]

En cas d'échec d'un test il fait une capture d'écran pour qu'on puisse savoir exactement qu'est ce qui est passé.

2.4. LES TESTS DE SÉCURITÉ

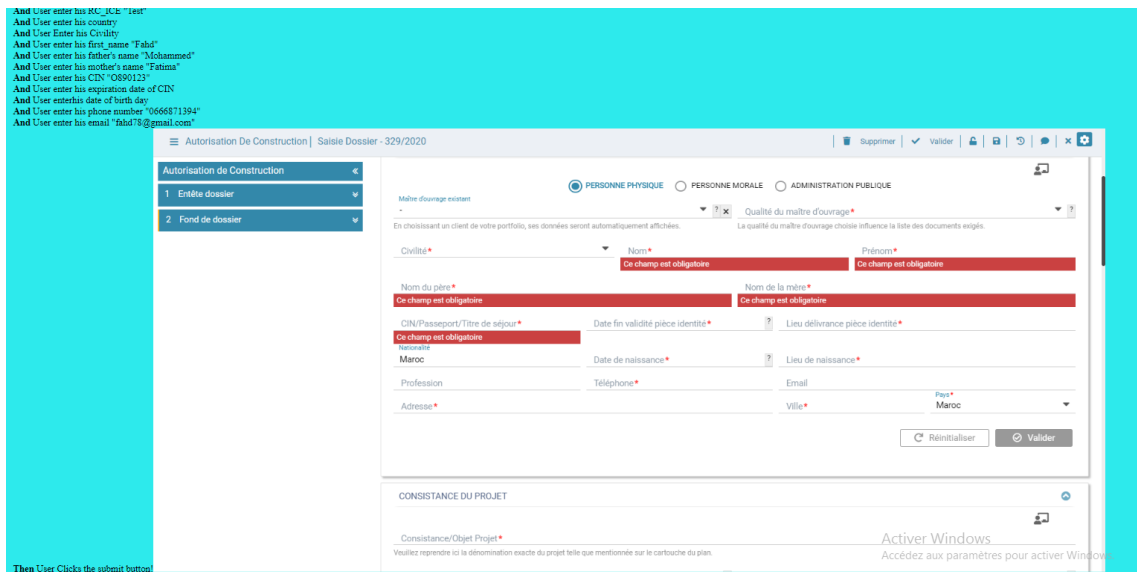


FIGURE 2.5 – Rapport d'un scénario qui a échouer [Capture d'écran]

2.4 Les tests de sécurité

Il est claire que les conséquences des violations de la sécurité sont dévastatrices, aujourd'hui, les tests de sécurité jouent un rôle vital dans toutes les applications Web, car les pirates continuent d'inventer chaque jour de nouvelles techniques qui peuvent être pour l'argent, la reconnaissance et même pour le plaisir. Les personnes ayant moins de compétences en piratage peuvent détruire une application Web si l'application est mal sécurisée.

L'objectif principal des tests de sécurité est de déterminer les vulnérabilités d'un système et de déterminer si ses données et ses ressources sont protégées contre les intrus potentiels.

2.4.1 Les techniques typiques de test de sécurité d'application :

2.4.1.1 DAST [Dynamic Application Security Testing]

Un outil de test de sécurité d'application dynamique teste les applications et services Web en utilisant des attaques fictives via HTTP / HTTPS, semblable à ce qu'un pirate informatique ferait. Cette méthode est appelée test en boîte noire. Simplement, un scanner DAST n'a

pas besoin d'avoir accès au code et au fonctionnement interne d'une application. En fait, il automatise ce qu'un pirate informatique ferait dans une application en direct. Ainsi, lorsqu'une vulnérabilité est trouvée, cela signifie généralement que la vulnérabilité pourrait être exploitée. Souvent, un bon scanner DAST montrera que la vulnérabilité est réelle en l'exploitant en toute sécurité et nous montrera comment résoudre le problème. Cela permet aux pentesters et aux développeurs de gagner du temps. Cependant, le plus grand atout de DAST a un prix : les scanners DAST ne peuvent tester qu'une application qui fonctionne, ce qui signifie que nous ne pouvons numériser avec un scanner DAST qu'une fois l'application créée avec succès.

2.4.1.2 Les outils de DAST :

Utilisation :

Les scanners DAST de vulnérabilités peuvent être utilisés dans des objectifs licites ou illicites :

- **Objectifs licites** : les experts en sécurité informatique ou les entreprises utilisent les scanners de vulnérabilités pour trouver les failles de sécurité des systèmes informatiques et des systèmes de communications de leurs entreprises dans le but de les corriger avant que les pirates informatiques ne les exploitent.
- **Objectifs illicites** : les pirates informatiques utilisent les mêmes outils pour trouver les failles dans les systèmes des entreprises pour les exploiter à leur avantage.

2.4.1.3 Nikto :



est un scanner de vulnérabilité deligne de commande de logiciel gratuit qui analyse les serveurs web pour les fichiers / CGI dangereux. Il effectue des vérifications génériques et spécifiques au type de serveur. Il capture et imprime également tous les cookies reçus. Le code Nikto lui-même est un logiciel gratuit, mais les fichiers de données qu'il utilise pour piloter le programme ne le sont pas.

2.4.1.4 Wapiti :



est un outil open source qui analyse les applications Web pour détecter de multiples vulnérabilités, notamment les injections de base de données, cross-site scripting, XXE injection, des injections SQL, XPath, PHP, ASP et JSP.

Il effectue également des tâches de test de pénétration supplémentaires, telles que la recherche de fichiers potentiellement dangereux sur les serveurs, Les résultats collectés sont automatiquement stockés dans un fichier HTML.

2.4.2 Application :

2.4.2.1 Premier cas : un site vulnérable

On a chercher sur Internet sur des sites vulnérables pour s'assurer que nos outils peuvent détecter des vulnérabilités s'elles existent.

http ://www.dvwa.co.uk/ Ses principaux objectifs sont d'aider les professionnels de la sécurité à tester leurs compétences et leurs outils dans un environnement juridique, à aider les développeurs Web à mieux comprendre les processus de sécurisation des applications Web et à aider les enseignants / étudiants à enseigner / apprendre la sécurité des applications Web dans un environnement de salle de classe.

• Output de wapiti :

Parmi les avantages de Wapiti et Nikto qu'ils sont facile à utiliser, il suffit d'écrire quelques commandes pour lancer un scan.

Après chaque scan Wapiti génère un rapport décrit la catégorie et le nombre du vulnérabilités détecter.


```

21  $ curl -s http://www.dwa.co.uk/ -v 2 --color
22
23
24  Wapiti-3.0.1 (wapiti.sourceforge.net)
25  [*] You are lucky! Full moon tonight.
26  [+] GET http://www.dwa.co.uk/ (0)
27  [+] GET http://www.dwa.co.uk/js/jquery.nivo.slider.pack.js (1)
28  [+] GET http://www.dwa.co.uk/index.php (1)
29  [+] GET http://www.dwa.co.uk/js/jquery.1.6.2.min.js (1)
30  [+] GET http://www.dwa.co.uk/.google-analytics.com/ga.js (1)
31  [+] GET http://www.dwa.co.uk/favicon.ico (1)
32  [+] GET http://www.dwa.co.uk/css/all.css (1)
33  [+] GET http://www.dwa.co.uk/nivo-slider/themes/default/default.css (1)
34  [+] GET http://www.dwa.co.uk/css/nivo-slider.css (1)
35  [*] Saving scan state, please wait...
36
37  Note
38  =====
39
40  This scan has been saved in the file /var/lib/gitlab-runner/.wapiti/scans/www.dwa.co.uk_folder_cdc80896.db
41
42  [+] Wapiti found 9 URLs and forms during the scan
43
44  [*] Loading modules:
45
46      mod_crlf, mod_exec, mod_form, mod_sql, mod_xss, mod_backup, mod_htaccess, mod_blindsql, mod_permanentxss, mod_nikt
47      o, mod_delay, mod_buster, mod_shellshock, mod_methods, mod_ssrif
48
49  [*] Launching module exec
50
51  [+] GET http://www.dwa.co.uk/ (0)
52  [+] GET http://www.dwa.co.uk/.google-analytics.com/ga.js (1)
53  [+] GET http://www.dwa.co.uk/css/all.css (1)
54  [+] GET http://www.dwa.co.uk/css/nivo-slider.css (1)
55  [+] GET http://www.dwa.co.uk/favicon.ico (1)
56  [+] GET http://www.dwa.co.uk/index.php (1)
57  [+] GET http://www.dwa.co.uk/js/jquery.1.6.2.min.js (1)
58  [+] GET http://www.dwa.co.uk/js/jquery.nivo.slider.pack.js (1)
59  [+] GET http://www.dwa.co.uk/nivo-slider/themes/default/default.css (1)
60
61  [*] Launching module file
62
63  [*] Saving scan state, please wait...

```

FIGURE 2.6 – Output de la commande de test de vulnérabilités [Wapiti]

Category	Number of vulnerabilities found
Cross Site Scripting	0
Htaccess Bypass	0
Backup file	0
SQL Injection	0
Blind SQL Injection	1
File Handling	0
Potentially dangerous file	0
CRLF Injection	0
Commands execution	0
Resource consumption	2
Internal Server Error	0

FIGURE 2.7 – Rapport généré par Wapiti

La même chose pour Nikto :

2.4. LES TESTS DE SÉCURITÉ

Chapitre 2.4.2

Conception et développement des tests

HTTP Method	GET
Description	/,htaccess: Contains authorization information
Test Links	http://www.dvwa.co.uk:80/htaccess http://178.79.147.242:80/htaccess
OSVDB Entries	OSVDB-3093
Host Summary	
Start Time	2020-04-17 17:22:47
End Time	2020-04-17 18:16:07
Elapsed Time	3200 seconds
Statistics	6544 items checked, 9 errors, 3 findings
Scan Summary	
Software Details	Nikto 2.1.5
CLI Options	-h http://www.dvwa.co.uk/ -mutate -Display V -F htm -output niktoscan.html
Hosts Tested	1
Start Time	Fri Apr 17 17:22:43 2020
End Time	Fri Apr 17 18:16:07 2020
Elapsed Time	3204 seconds

FIGURE 2.8 – Rapport généré par Nikto

2.4.2.2 Deuxième cas : La plateforme Rokhas

- Output de wapiti et de Nikto :

On a attaqué la plateforme Rokhas par les deux outils, et les deux confirment que la plateforme est bien sécurisé, pas des failles de sécurité, pas des vulnérabilités :

Summary	
Category	Number of vulnerabilities found
Cross Site Scripting	0
Htaccess Bypass	0
Backup file	0
SQL Injection	0
Blind SQL Injection	0
File Handling	0
Potentially dangerous file	0
CRLF Injection	0
Commands execution	0
Resource consumption	0

```

16 Starting nikto Scan ...
17 $ nikto -h https://urbaformation.karaz.org/karazal/index.jsp -Display -mutate
18 - Nikto v2.1.5
19 .....
20 + Target IP: 91.134.184.92
21 + Target Hostname: urbaformation.karaz.org
22 + Target Port: 443
23 + Start Time: 2020-04-07 12:44:30 (GMT0)
24 .....
25 + Server: nginx/1.14.0 (Ubuntu)
26 + The anti-clickjacking X-Frame-Options header is not present.
27 + No CGI Directories found (use '-C all' to force check all possible dirs)
28 + 6545 items checked: 0 error(s) and 1 item(s) reported on remote host
29 + End Time: 2020-04-07 12:46:33 (GMT0) (123 seconds)
30 .....
31 + 1 host(s) tested
35 Job succeeded

```

FIGURE 2.9 – Rapports Plateforme Rokhas [0 vulnérabilités]

2.4.2.3 SAST [Static Application Security Testing]

Les scanners SAST, en revanche, ne nécessitent pas d'application en cours d'exécution et fonctionnelle. Au lieu d'analyser l'application de l'extérieur, ils analysent directement le code. Un outil d'analyse statique (SAST) tenterait de savoir comment les données transitent par l'application vers une base de données. Donc, disons d'une source d'entrée (comme un champ de formulaire) jusqu'à un point de synchronisation sensible à la sécurité (comme un appel à une base de données). En analysant ce flux, l'outil vérifie si ces synchronisations sensibles à la sécurité sont sres et nettoyées, et c'est ainsi qu'un outil SAST trouve des vulnérabilités. Ainsi, au lieu d'attaquer l'application de l'extérieur, comme le ferait un DAST, un SAST détermine le fonctionnement interne de notre base de code et essaie de trouver des vulnérabilités de l'intérieur. Pour cette raison, SAST ne nécessite pas que l'application soit fonctionnelle et en cours d'exécution.

2.4.2.4 L' outil de SAST :



est une plateforme open source développée par SonarSource pour une inspection continue de la qualité du code afin d'effectuer des révisions automatiques avec une analyse statique du code pour détecter les bugs, les vulnérabilités de sécurité sur plus de 20 langages de programmation. SonarQube propose des rapports sur le code dupliqué, les normes de codage, les tests unitaires, la couverture du code, la complexité du code, les commentaires, et les failles de sécurité.

2.4.2.5 Application :

Premier cas : une application vulnérable :

Il s'agit d'une application Web vulnérable développée par Cyber Security and Privacy Foundation (www.cysecurity.org). Cette application est destinée aux programmeurs Java et aux autres personnes qui souhaitent en savoir plus sur les vulnérabilités des applications Web et écrire du code sécurisé.

2.4. LES TESTS DE SÉCURITÉ

Chapitre 2 Conception et développement des tests

Après le lancement du scan sur l'application on voit dans SonarQube tous les projets qu'on a scanné :

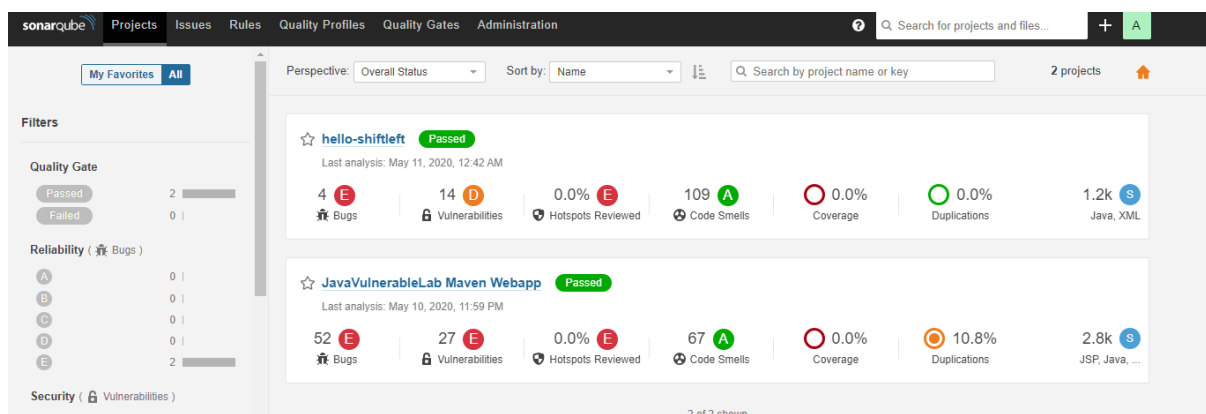


FIGURE 2.10 – HOME page SonareQube

Mots clés :

- **un bug** est un fonctionnement non désiré dans un programme informatique, On parle de bug quand le logiciel ne fait pas ce qu'on lui a demandé, qu'il se bloque et ne répond plus ou qu'il plante.
- **vulnérabilités** ce sont les failles de sécurité.
- **Un point d'accès de sécurité** met en évidence un morceau de code sensible à la sécurité que le développeur doit examiner. Après examen, on constate qu'il n'y a aucune menace ou bien nous devons appliquer un correctif pour sécuriser le code.
- **Les mauvaises odeurs** sont des mauvaises pratiques de conception logicielle qui conduisent à l'apparition de défauts.

En choisit le projet qu'on veut, Dans l'aperçu on voit les résultats. Par exemple pour ce projet le scanner il a trouvé 52 bugs, 27 vulnérabilités, 18 point d'accès de sécurité, 67 mauvaises odeurs.

2.4. LES TESTS DE SÉCURITÉ

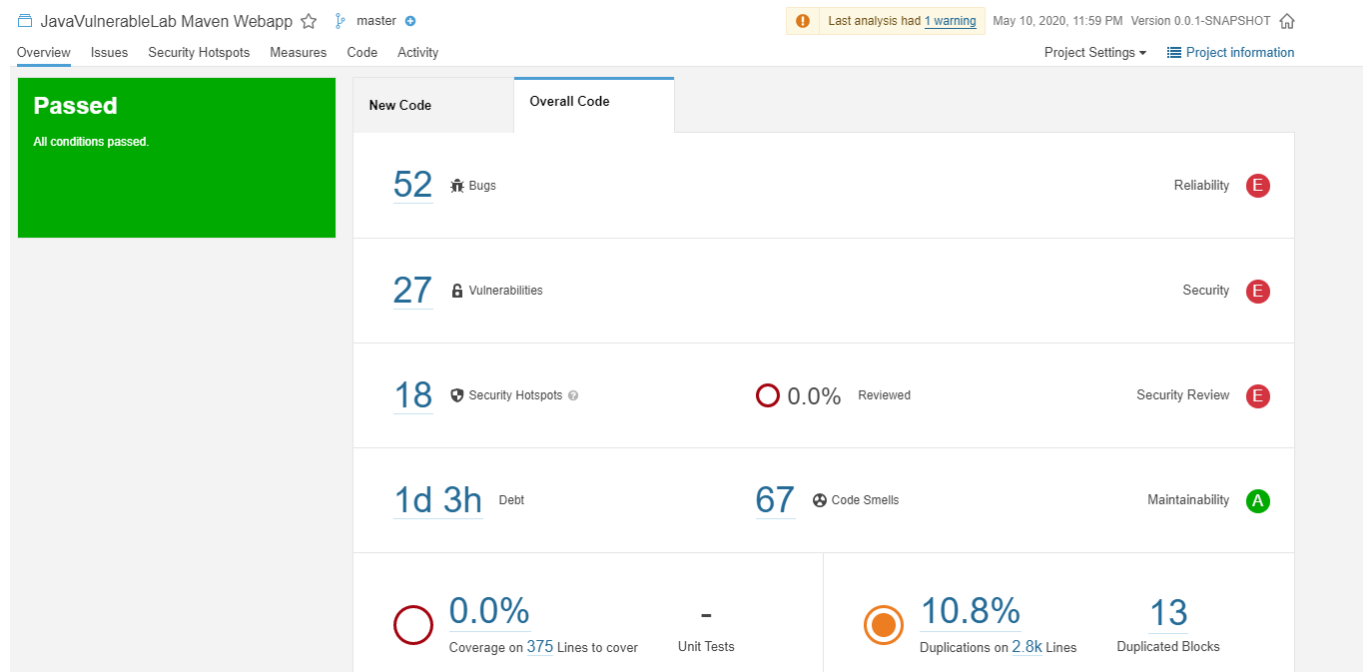


FIGURE 2.11 – Aperçu SonareQube

On remarque devant chaque élément une lettre par exemple devant Bugs on a E de même pour vulnérabilités et les points d'accès. mais pour les code smells on a la lettre A.

Les lettres représentent le degré de dangerosité.

E : bloqueur ; **D** : critique ; **C** : majeur ; **B** : mineure ; **A** : pas de problème.

Dans l'aperçu pour les vulnérabilités par exemple, il indique la lettre de la plus dangereuse vulnérabilité. Si la plus dangereuse a la lettre C, et tous les autres ont soit B ou bien A, donc il affiche C. et la même chose pour les autres éléments.

Dans l'onglet Problèmes il liste tous les bugs, les vulnérabilités... et il donne le détail pour chaque problème, et il propose aussi des solutions.

2.4. LES TESTS DE SÉCURITÉ

Conception et développement des tests

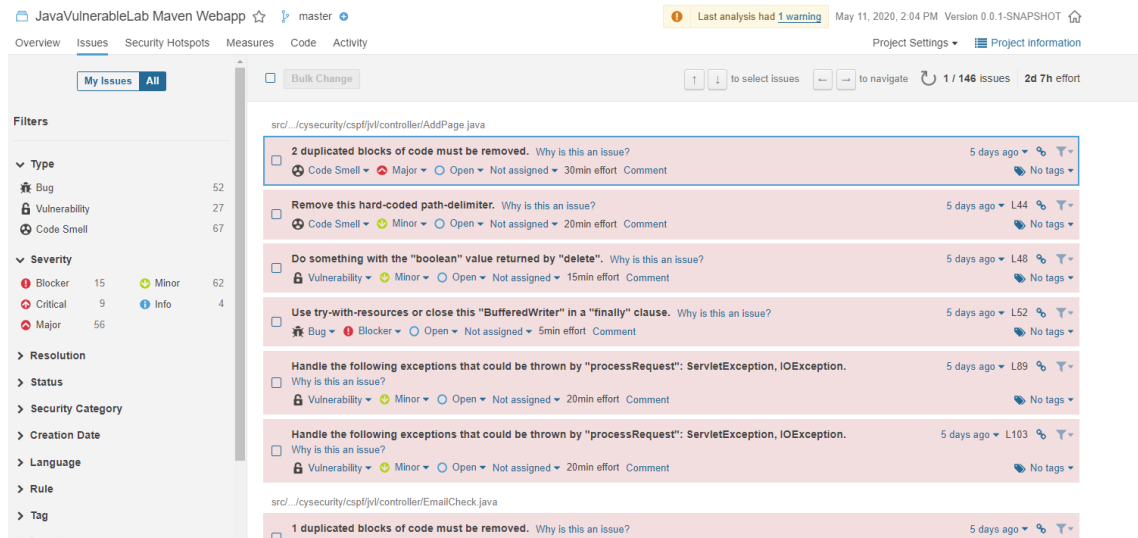


FIGURE 2.12 – Problèmes SonareQube

Mais ce qui nous intéresse le plus, ce sont les points d'accès de sécurité.

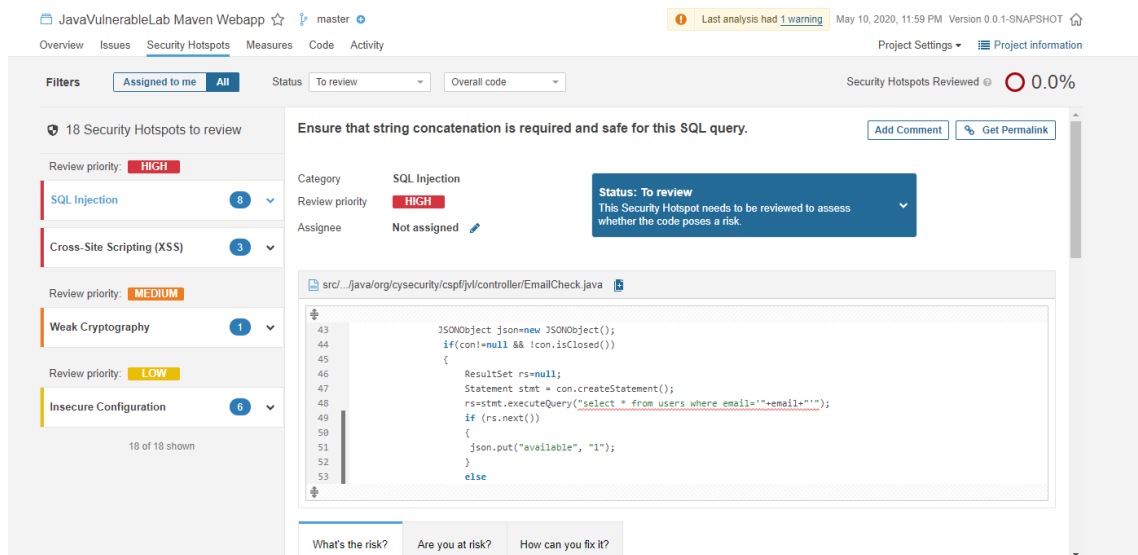


FIGURE 2.13 – Les Points d'accès de sécurité SonareQube

2.4. LES TESTS DE SÉCURITÉ

Chapitre 2 Conception et développement des tests

Lorsque SonarQube détecte un point d'accès de sécurité, il l'ajoute à la liste des points d'accès de sécurité en fonction de sa priorité de révision de haute à basse. Les points d'accès avec une priorité de révision élevée sont les plus susceptibles de contenir du code qui doit être sécurisé et requiert notre attention en premier.

La priorité de révision est déterminée par la catégorie de sécurité de chaque règle de sécurité. Les règles dans les catégories qui sont classées haut sur les standards OWASP Top 10 et CWE Top 25 sont considérées comme ayant une haute priorité d'examen. Les règles dans les catégories qui ne sont pas classées haut ou qui ne sont pas mentionnées dans les standards OWASP Top 10 ou CWE Top 25 sont classées moyenne ou faible.

Note :

Common Weakness Enumeration [CWE] est une liste des vulnérabilités que l'on peut rencontrer dans les logiciels. Cette liste est maintenue par l'organisme MITRE, le projet étant soutenu par la National Cyber Security Division et le Département de la Sécurité intérieure des États-Unis.



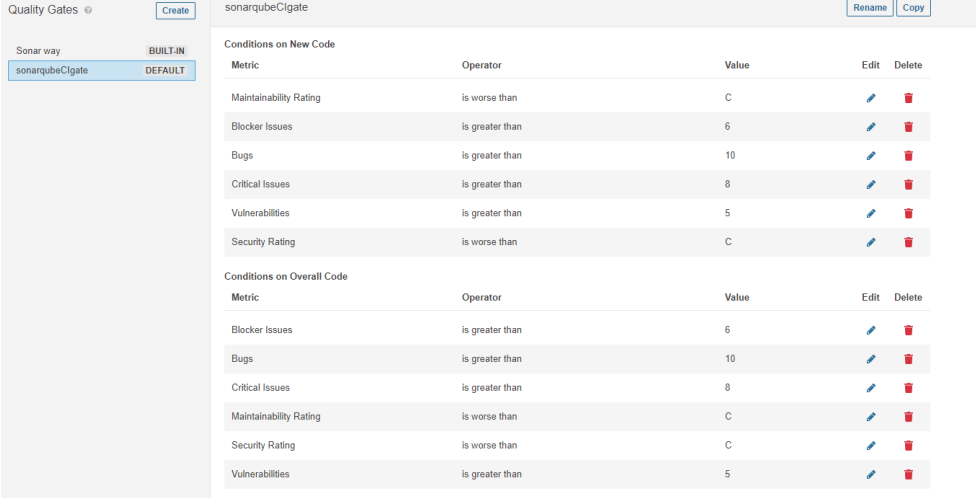
est une fondation à but non lucratif qui travaille à améliorer la sécurité des logiciels. Grâce à des projets de logiciels open source menés par la communauté, des centaines de sections locales dans le monde, des dizaines de milliers de membres et des conférences éducatives et de formation de premier plan, la Fondation OWASP est la source pour les développeurs et les technologues de sécuriser le Web.

Le Top 10 OWASP est un document de sensibilisation standard pour les développeurs et la sécurité des applications Web. Il représente un large consensus sur les risques de sécurité les plus critiques pour les applications Web.

Sonarqube nous donne la possibilité d'exiger nos conditions pour décider est ce que le scan passe ou échoue à travers **Quality Gates**

2.4. LES TESTS DE SÉCURITÉ

Chapter 2.4 Conception et développement des tests



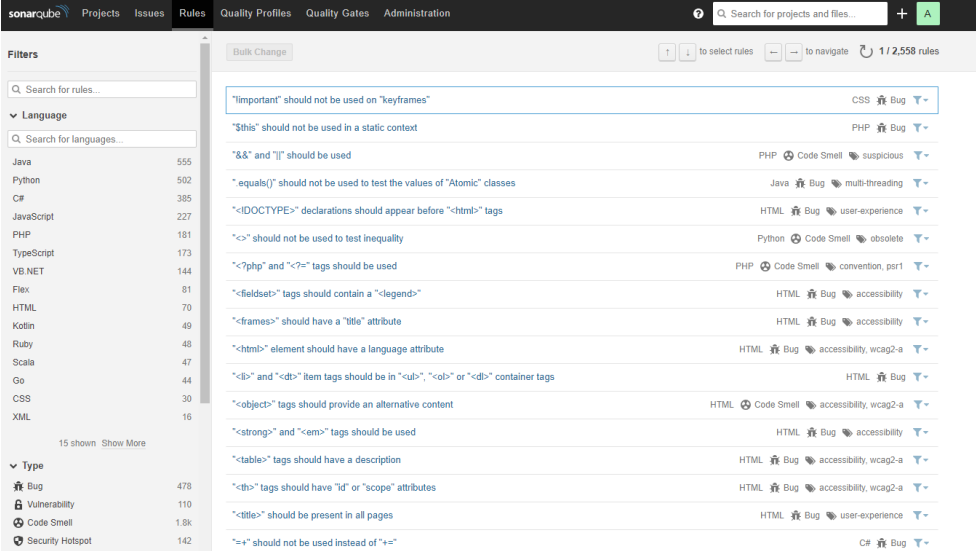
The screenshot shows the 'Quality Gates' configuration in SonarQube. On the left, a sidebar lists 'Sonar way' (BUILT-IN) and 'sonarqubeCIGate' (DEFAULT). The main area is titled 'sonarqubeCIGate' and contains two tables of conditions.

Conditions on New Code				
Metric	Operator	Value	Edit	Delete
Maintainability Rating	is worse than	C		
Blocker Issues	is greater than	6		
Bugs	is greater than	10		
Critical Issues	is greater than	8		
Vulnerabilities	is greater than	5		
Security Rating	is worse than	C		

Conditions on Overall Code				
Metric	Operator	Value	Edit	Delete
Blocker Issues	is greater than	6		
Bugs	is greater than	10		
Critical Issues	is greater than	8		
Maintainability Rating	is worse than	C		
Security Rating	is worse than	C		
Vulnerabilities	is greater than	5		

FIGURE 2.14 – Quality Gates SonareQube

Il présente aussi des règles à respecter pour les différentes langages qui prend en charge afin de minimiser le nombres des bugs.



The screenshot shows the 'Rules' page in SonarQube. On the left, there are filters for 'Language' and 'Type'. The main area displays a list of rules with their descriptions, associated languages, and icons for bug, vulnerability, code smell, and security hotspot.

Rule Description	Language	Icons
"!important" should not be used on "keyframes"	CSS	Bug
"\$this" should not be used in a static context	PHP	Bug
"&&" and " " should be used	PHP	Code Smell, suspicious
"equals()" should not be used to test the values of "Atomic" classes	Java	Bug, multi-threading
"<!DOCTYPE>" declarations should appear before "<html>" tags	HTML	Bug, user-experience
"<>" should not be used to test inequality	Python	Code Smell, obsolete
"<?php" and "<?=" tags should be used	PHP	Code Smell, convention, psr1
"<fieldset>" tags should contain a "<legend>"	HTML	Bug, accessibility
"<frames>" should have a "title" attribute	HTML	Bug, accessibility
"<html>" element should have a language attribute	HTML	Bug, accessibility, wcag2-a
"<div>" and "<div>" item tags should be in "<div>", "<div>" or "<div>" container tags	HTML	Bug
"<object>" tags should provide an alternative content	HTML	Code Smell, accessibility, wcag2-a
"" and "" tags should be used	HTML	Bug, accessibility
"<table>" tags should have a description	HTML	Bug, accessibility, wcag2-a
"<th>" tags should have "id" or "scope" attributes	HTML	Bug, accessibility, wcag2-a
"<title>" should be present in all pages	HTML	Bug, user-experience
"==" should not be used instead of "==="	C#	Bug

FIGURE 2.15 – Rules SonareQube

2.4. LES TESTS DE SÉCURITÉ

Chapitre 2. Conception et développement des tests

Deuxième cas : Analyse d'une partie du code source en cours d'exécution de l'entreprise

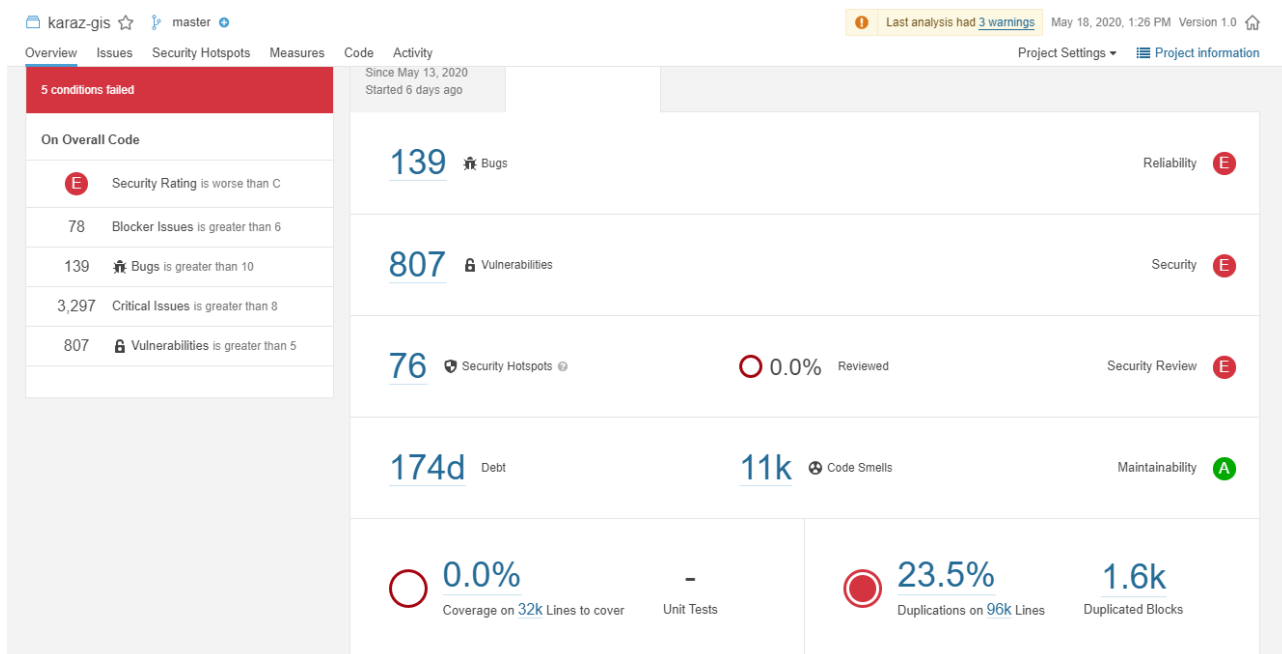


FIGURE 2.16 – Scan du code source de l'entreprise

Cela peut sembler un petit peu bizarre, mais il y a deux facteurs qui sont responsable de ces chiffres.

- Le premier c'est l'utilisation d'une bibliothèque de XML parse qui nécessite une configuration représente des problèmes.
- Le deuxième c'est les faux positifs, sonarqube préfère de déclarer qu'une ligne dangereuse même si elle n'est pas, que déclarer une ligne dangereuse comme une ligne correcte.

2.5 Les tests de performance

2.5.1 Définition :

Les tests de performance et de charge ont pour principal objectif la validation d'une solution logicielle et de son architecture sous-jacente liées à une utilisation simultanée multi-utilisateurs, permettant ainsi d'éviter certains problèmes en production. Ils permettent de garantir une qualité de service applicative dans des conditions réelles d'utilisation.

2.5.2 Généralités :

Les tests de performance devraient théoriquement être réalisés au plus tôt dans le cycle de développement applicatif afin de minimiser les modifications applicatives engendrées par la découverte des anomalies de performance.

Classiquement, les tests de performance sont réalisés dans un des environnements de pré-production afin d'être au plus proche de la réalité technique de production. Les résultats seront alors réellement complets et probants.

Les tests de performance et de charge ne doivent pas, comme les tests fonctionnels unitaires, couvrir 90% des cas de tests, mais doivent couvrir les aspects applicatifs présentant des risques systèmes en charge, des risques d'interactions avec d'autres applicatifs, des risques stratégiques ou financiers sur des fonctionnalités ciblées dans un contexte multi-utilisateurs.

2.5.3 Les risques :

Si une application n'est pas performante, les conséquences pourraient être :

- La qualité d'accueil des clients qui se dégrade.
- La perte de clients.
- La perte de revenu.

2.5.4 Outils de test :



Est un projet de logiciel libre permettant d'effectuer des tests de performance d'applications et de serveurs selon différents protocoles ainsi que des tests fonctionnels. Il est développé au sein de la Fondation Apache (ASF).

Il permet de simuler le comportement de plusieurs utilisateurs agissant de manière simultanée sur une application Web.

JMeter est entièrement écrit en Java, ce qui lui permet d'être utilisé sur tout système d'exploitation supportant une machine virtuelle Java (JVM).



Commercialise une plateforme commerciale de test de charge en libre-service en tant que service (PaaS), qui est compatible avec Apache JMeter open-source ,dans le cadre de test de performance de la Apache Software Foundation.

BlazeMeter a été fondé en 2011 et a été acquis par CA Technologies en 2016.

2.6 Conclusion

Pas encore ..

L'intégration dans la chaîne CI/CD de GitLab

3.1 Introduction

L'approche CI/CD permet d'augmenter la fréquence de distribution des applications grâce à l'introduction de l'automatisation au niveau des étapes de développement des applications. Les principaux concepts liés à l'approche CI/CD sont l'intégration continue, la distribution continue et le déploiement continu. L'approche CI/CD représente une solution aux problèmes posés par l'intégration de nouveaux segments de code pour les équipes de développement et d'exploitation.

3.2 Technologies



Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par plus de douze millions de personnes.



GitLab est un logiciel libre de forge basé sur git proposant les fonctionnalités de wiki, un système de suivi des bugs, l'intégration continue et la livraison continue. Développé par GitLab Inc et créé par Dmitriy Zaporozhets et par Valery Sizov, le logiciel est utilisé par plusieurs grandes entreprises informatiques incluant IBM, Sony, la NASA, Alibaba, Oracle, Leibniz Rechenzentrum, Boeing, Autodata et SpaceX...

3.3 Définitions

3.3.1 L'intégration continue [CI] :

L'intégration continue nous fournit une bonne solution lorsque l'entreprise travaille sur un vaste projet ou un client souhaite avoir un logiciel à la fois complet et complexe. Différentes équipes travaillent à la conception de pans de l'application et les développeurs se chargent de programmer les différentes fonctionnalités. Après un travail de plusieurs mois voire de plusieurs années, l'intégralité du travail doit être regroupée et c'est alors que les problèmes surviennent. Dans un tel cas, la détection et la correction des erreurs, le regroupement de tous les fragments de code peut prendre plusieurs mois pour finalement se rapprocher de la phase de test finale et du déploiement.

Dans le cadre de la continuous integration, l'intégration du nouveau code est effectuée de façon bien plus précoce et pas uniquement lorsque toutes les parties prenantes ont terminé leur sous-domaine. Au lieu de cela, les développeurs intègrent leur code terminé une ou plusieurs fois par jour dans la mainline, le code source qui est accessible par tous les programmeurs. tant donné qu'il s'agit toujours dans ce cas de sections de code relativement courtes, l'intégration est elle aussi plutôt rapide. Seules quelques minutes sont nécessaires à un développeur pour mettre le résultat de son travail à disposition du reste de l'équipe. Si l'on découvre alors une erreur, elle peut être immédiatement localisée et, dans le meilleur des cas, corrigée rapidement.

3.3.2 La livraison "Delivery"/Déploiement continue [CD] :

La livraison continue consiste à automatiser toutes les phases de développement, depuis la modification des lignes de code, en incluant la compilation, l'exécution des tests unitaires, des tests d'intégration et des tests fonctionnels, jusqu'à la livraison d'un package que l'équipe de Production peut déployer à la demande. En déploiement continu, on y ajoute une dernière phase automatisée : le déploiement du package en production sans intervention humaine.

Le déploiement continu (CD) est donc la suite de l'intégration continue. Une fois que les tests sont validés sur l'environnement de dev, il faut mettre en production. Le déploiement continu consiste donc à automatiser les actions de déploiements qui étaient auparavant réalisées manuellement. C'est pour cette raison que l'on parle souvent de CI/CD ensemble, l'un va difficilement sans l'autre.

3.3.2.1 La différence entre déploiement continu et livraison continue

Le déploiement continu c'est un idéal que peu d'entreprises ont réellement mis en place. La plupart du temps les équipes IT préfèrent avoir la main sur la dernière étape du déploiement. Dans ce cas on parle donc de livraison continue, toutes les étapes du déploiement sont automatisées sauf la dernière : la mise en production.

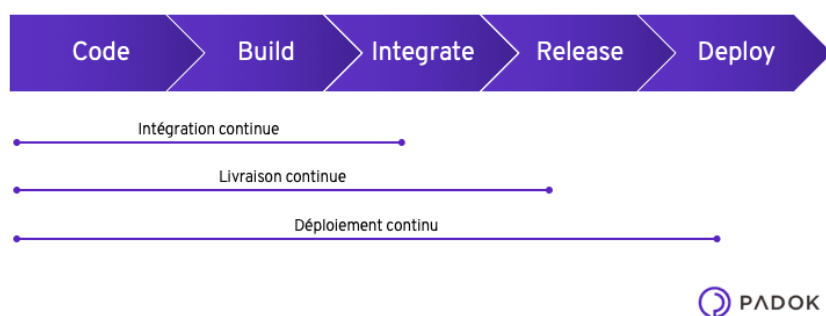


FIGURE 3.1 – Intégration/Livraison/Déploiement Continue

Le déploiement continu inclut donc la livraison continue. Le déploiement continu va plus

loin que la livraison continue en orchestrant automatiquement le déploiement des nouvelles fonctionnalités.

3.4 Principe de fonctionnement

3.4.1 Le fichier de configuration :

En substance, le système GitLab CI / CD comprend trois ingrédients principaux :

- Runners
- Pipelines
- Stages [Étapes]
- Jobs [Tâches]

Expliquons chacun d'eux, du bas de la liste :

- **Jobs** : La tâche est la plus petite unité à exécuter dans GitLab CI / CD. Il est souvent appelé étape de construction. Il peut s'agir d'une tâche de génération ou de compilation ; il peut exécuter des tests unitaires ...
Une seule tâche peut contenir plusieurs commandes (scripts) à exécuter.
- **Stages** : Chaque tâche appartient à un seul étape. Une étape peut contenir zéro, un ou plusieurs tâches à exécuter. Tous les tâches d'une étape s'exécutent en parallèle.
L'étape suivante n'est exécutée que si tous les travaux de l'étape précédente se terminent avec succès - ou s'ils sont marqués comme autorisés à échouer.
- **Pipelines** : Un pipeline est un parapluie pour les tâches et les étapes. Pipeline les orchestre et les rassemble. Le pipeline s'exécute lorsque vous envoyez une nouvelle validation ou une nouvelle balise, exécutant tous les travaux à leurs étapes dans le bon ordre. La configuration entière du pipeline est stockée dans le fichier de configuration **.gitlab-ci.yml**

3.5. ~~EXEMPLES D'INTÉGRATION DES TESTS~~ *Intégration dans la chaîne CI/CD de GitLab*

- **Runners** : GitLab Runner est le projet open source utilisé pour exécuter les tâches et renvoyer les résultats à GitLab. Il est utilisé conjointement avec GitLab CI / CD .

3.5 Exemples d'intégration des tests :

Pas Encore ..