

Rapport OTI - Projet Web avancé

1) Description du projet et enjeux des tests	1
2) Définitions des tests	1
3) Code coverage, évaluation Sonar et changements sous jacent	2
4) Retour sur les tests	4

1) Description du projet et enjeux des tests

Le projet web avancé est une application de conversion de données de type ICS en JSON afin de permettre leur stockage sur un serveur distant et leur consultation ultérieure.

Il comprend 4 classes majeures JS qui sont :

- Creneau, l'entité majeure
- Conversion, la partie algorithmique avec conversion et envoi des données
- Affichage, la partie récupération des informations et affichage
- Exception, contenant les exceptions du projet

et deux fichiers HTML :

- Conversion.html
- Affichage.html

Ce projet se prête bien à notre cadre de test avec des interactions utilisateurs, un rendu affichage et des interactions serveurs. Il permettra de définir différents types de test et d'exploiter l'ensemble des outils mis à disposition.

2) Définitions des tests





L'ensemble de l'application est testé, cependant une attention particulière a été portée à la classe Conversion contenant en grande partie la logique de l'application.

Celle-ci a été à la fois testée avec QUnit/Sinon (pour mocker les appels serveurs) au niveau de la logique applicative mais également avec Selenium IDE puisqu'elle contient plusieurs types d'interactions utilisateurs et donc de retours à l'affichage.

Concernant la classe Affichage qui correspond au reste de la logique applicative, elle a uniquement été testée en QUnit/Sinon, les interactions utilisateurs étant réduites à un retour à la page de Conversion.

3) Code coverage, évaluation Sonar et changements sous jacent

a) Karma JS

all files src/									
92% Statements 69/75		88.89% Branches 16/18		82.35% Functions 14/17		92% Lines 69/75			
File ▲		Statements		Branches		Functions		Lines	
affichage.js		100%	27/27	100%	2/2	100%	6/6	100%	27/27
conversion.js		87.8%	36/41	87.5%	14/16	75%	6/8	87.8%	36/41
creneau.js		100%	4/4	100%	0/0	100%	1/1	100%	4/4
exception.js		66.67%	2/3	100%	0/0	50%	1/2	66.67%	2/3

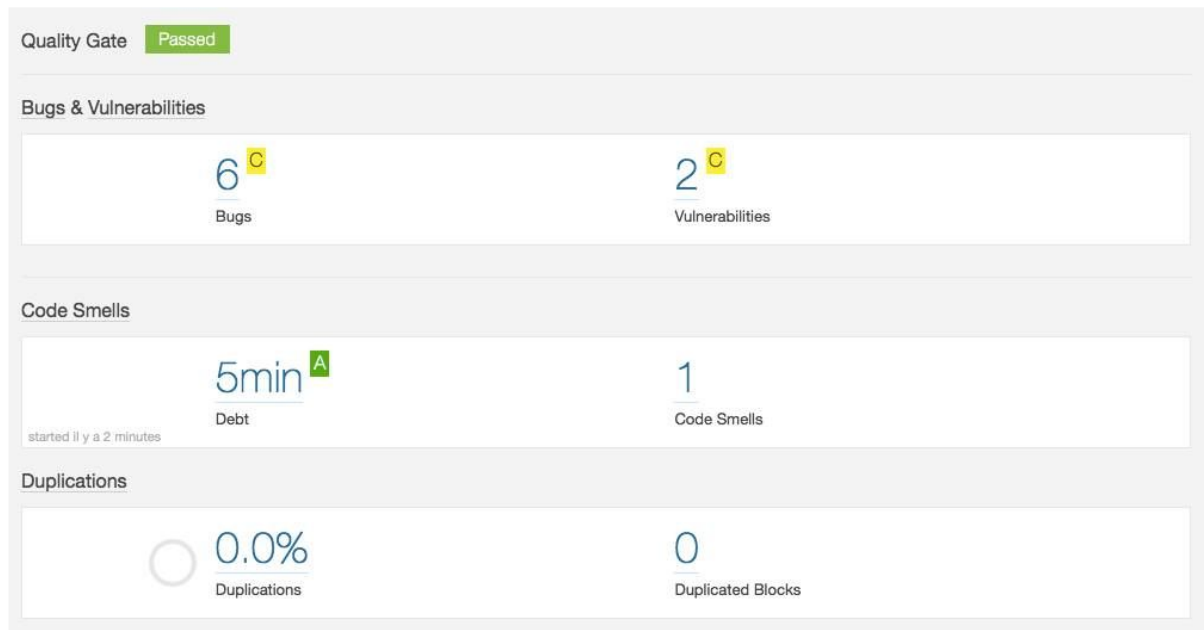
Code coverage Karma JS

L'ensemble du contenu src/ a été testé. La classe cliente du serveur étant extérieur au projet, elle est donc considérée comme une librairie et n'a pas été testée.

- La classe d'exception ne regroupe qu'une exception ne comprenant qu'une méthode qui est la surcharge de toString.
- L'affichage a été testé dans son intégralité, que ce soit au niveau de la réception des données et des différents appels en découlant grâce au mock de la méthode "avoirToutLeTableau".
- Concernant conversion.js, une grande partie de la classe a été testée, la partie restante concernant certains appels serveurs à mocker.
- Creneau.js est une entité qui ne contient pas de méthode à tester.

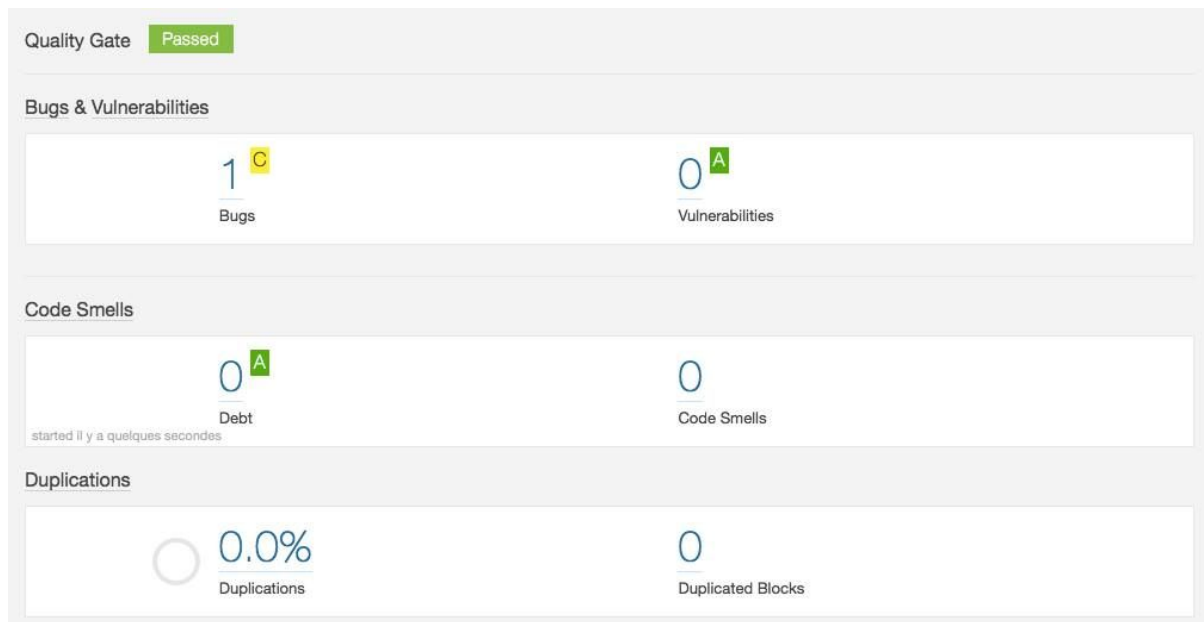
L'objectif des 100% sans une attention particulière aux différents cas de tests n'est pas une bonne pratique. En effet, il est d'autant plus important de tester scrupuleusement les différents cas que de simplement "passer" par l'ensemble des fonctions sans réellement les tester. Le cap des 80% recommandés a été respecté, certains cas de tests étant difficiles à effectuer (interaction niveau serveur).

b) Sonar



Après finalisation des tests, un premier relevé de Sonar sur la qualité du code. Plusieurs anomalies ont été remontées, pour certaines classiques comme des variables non utilisées, ou des traces dans la console et d'autres plus complexes comme la création de fonctions dans une boucle ou l'utilisation de la méthode "for..in" qui semble être dépréciée.

Des modifications ont donc résultées à ce premier relevé. Les consoles et variables non utilisées ont été supprimées. La fonction "for..in" a été remplacée par Object(obj).keys introduit par ECMA 5.



Après un deuxième relevé, un seul "bug" résulte. Il correspond au callback de la méthode Object(obj).keys défini directement à l'intérieur d'une boucle. Ils ont volontairement été laissés ici pour une meilleure lisibilité du code.

4) Retour sur les tests

Le projet de test a permis de mettre en évidence l'importance des tests et de la qualité du code dans une application que ce soit au niveau de la logique applicative (et des performances en découlant) ou des retours à l'utilisateur. Les différents types de tests ont permis de répondre à ces "besoins". Ceci a permis de démontrer que se focaliser sur le pourcentage de code coverage est un mauvais indice puisqu'il délaisse bien souvent les cas de tests. Sonar a permis la remontée de nombreux "bugs" qu'il est essentiel d'éviter puisqu'il s'agit le plus souvent de mauvaises pratiques facilement remédiables. Le projet a donc vu des modifications suite à ces retours et est maintenant plus facilement maintenable grâce à sa couverture de tests.