Les Tests

Rappels

Tester ne doit pas être douter !
Un test c'est **confronter** une **assertion** à une **réalité**.

Comment peut-on **tester** notre **code** ou notre futur code ?
En réfléchissant aux valeurs extrêmes et validant le besoin

Quand ça **marche** on les met à la **poubelle** ?
Non on les gardes pour éviter des effets de bords et chiffrer notre couverture de tests

Qui écrit les tests ?
Les developpeurs

Les critères d'un bon test unitaire

1) Unité

Un test unitaire se concentre sur une seule unité, qui est le plus petit élément identifiable de notre application. Selon les contextes et les langages de programmation, plusieurs éléments du code peuvent constituer une unité. Il peut s’agir d’une fonction, d’une méthode de classe, d’un module, d’un objet… Parce qu’ils se concentrent sur les plus petites parties de notre application, les tests unitaires sont des tests de bas niveau. À l’inverse, les tests de haut niveau contrôlent la validité d’une ou plusieurs fonctionnalités complètes.

Les critères d'un bon test unitaire

2) Isolation

Les tests unitaires visant à tester chaque unité en isolation totale par rapport aux autres, ils doivent pouvoir être indépendants des tests lui précédents. Votre suite de tests unitaires doit pouvoir être lancé dans n'importe quel ordre sans affecter le résultat des tests suivants. C'est pourquoi l'utilisation de Mocks et Stubs est indispensable aux tests unitaires.

Les critères d'un bon test unitaire

3) Rapidité

La petite échelle des tests unitaires et le fait qu'ils soient écrits par les développeurs pendant le développement font que les tests unitaires sont souvent très rapides. Ils peuvent ainsi être lancés très fréquemment, idéalement à chaque modification dans le code ou à chaque compilation. Cette façon de procéder permet de repérer les bugs bien plus rapidement : si vous avez accidentellement cassé une fonctionnalité pendant votre dernier changement, vous le saurez immédiatement et n'aurez pas à chercher bien loin pour le réparer. Vous n'êtes bien sûr pas obligés de lancer tous les tests unitaires à chaque fois.

Les critères d'un bon test unitaire

4) L’indépendance

L’écriture de tests unitaires est peut-être l’un des seuls domaines où être un indépendantiste est socialement acceptable. Veillez à isoler vos tests unitaires au maximum et à les rendre totalement indépendants les uns des autres. Ne faites jamais appel à une base de données ou à une API externe même si votre classe en dépend : utilisez toujours des données de test les plus proches possibles des données réelles. On peut utiliser des mocks et des stubs pour simuler le fonctionnement des autres modules qui ne sont pas dans le scope de notre unité, ceux-ci seront testés unitairement de leurs côtés. Pour l'aléatoire, utilisez des seed !

Les critères d'un bon test unitaire

5) Automatisés

Les tests unitaires doivent produire un résultat Pass ou Fail automatiquement. Ils doivent pouvoir être interprétés par un test runner et ne pas demander au développeur de lire ou d'observer manuellement que le test a réussi ou échoué. C'est pourquoi les tests automatisés, qu'ils soient unitaires ou non, sont exécutés par un test runner et évalués par une librairie d'assertion.

Les critères d'un bon test unitaire

6) Rejouabilité

L'intérêt de bons tests unitaires réside dans le fait qu'ils soient idempotents, c’est-à-dire que pour un test donné, quel que soit l'environnement ou le nombre de fois qu'il soit joué, il produise toujours le même résultat. C'est pourquoi il est indispensable de faire abstraction des appels en base de données ou des requêtes HTTP pour avoir un test unitaire robuste.

Bonus

- Gardez vos tests unitaires **très rapides**, jusqu'à une **dizaine** de secondes au maximum
- Avant de **réparer un bug**, **écrivez ou modifiez** un test unitaire pour exposer ce bug
- Choisissez la **bonne unité** pour que votre plan de test couvre un maximum de fonctionnalités
- Utilisez le **template AAA** pour améliorer la lisibilité de votre test : Arrange (création des objets, des données de test et définition des attentes), Act (invocation de la méthode testée), Assert (résultat du test unitaire)

Jest : notre outil de test

Pour la logique et pour du React, Vue, Angular, Node, Typescript

https://jestjs.io/

Expect et **it** ! Et pour faire un peu plus, allez lire cette **cheatsheet** !
https://github.com/sapegin/jest-cheat-sheet

- Code coverage
- Fast and Safe
- API devenue la norme