Les tests unitaires

Vous pouvez parfois tomber sur le terme "test unitaire", mais qu'est ce que cela signifie? Il est évident qu'il faut toujours tester nos applications, dans les cas où cela doit fonctionner mais aussi dans les cas où cela ne doit pas fonctionner.

Si on prend le cas d'un formulaire d'inscription simple, juste avec email et mot de passe. Si on entre un email valide et qui n'est pas déjà enregistré, ainsi qu'un mot de passe sécurisé, il doit nous inscrire.

C'est bien si cela fonctionne, mais attention de tester l'inverse aussi. Si je met un email déjà enregistré, suis je bien refusé ? Si je met n'importe quoi dans l'email, suis aussi refusé ?

Ce sont les nombreux tests que l'on doit effectuer lorsque l'on développe une application (web ou non).

Mais un test unitaire alors? Un test unitaire est comme son nom l'indique, non pas un test de toute l'application mais d'une fonctionnalité. Et cela de façon à souvent automatiser les tests, pour qu'ils puissent être relancé régulièrement afin de vérifier que des changements apporté à notre code ne vont pas impacter sur nos fonctionnalités précédements créé.

On peut très bien écrire des tests unitaires à la main mais autant ne pas se compliquer la tâche et utiliser un framework fait pour cela.

PHPUnit

En PHP nous pourrons par exemple utiliser PHPUnit.

Pour cela nous devons l'installer soit avec composer soit en le téléchargeant :

```
composer require --dev phpunit/phpunit ^11
# On pourra vérifier que l'installation a été bien faite via :
./vendor/bin/phpunit --version
```

Ensuite il me faudra créer un fichier de test dont le nom doit se terminer par `Test.php`

Celui ci devra au moins contenir :

```php
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class nomDuFichierTest extends TestCase{}
```

Ensuite nous pourrons importer ce que l'on souhaite tester et le placer dans une ou plusieurs méthodes public dont le nom doit commencer par `test`.

Ces méthodes seront lancées automatiquement.

De plus chacune des méthodes de test doivent contenir au moins une "**assertion**". Ces assertions permettent de vérifier ce que doivent retourner, réaliser nos fonctions.

En voici quelques exemples :

```
// Le tableau possède t-il une clef
$this->assertArrayHasKey("key", array, "message");
// La condition retourne true
$this->assertTrue(condition, "message");
// valeur 1 est plus petite que valeur 2
$this->assertLessThan(valeur1, valeur2, "message")
```

Et on en trouvera divers pour tous les cas possible. Le message s'affichera si le test n'est pas rempli.

Une fois nos assertions réalisées dans la classe, nous pouvons lancer notre fichier via :

```
./vendor/bin/phpunit test
```

Il affichera alors le nombre de test réalisé, les succès et échecs et autres informations comme la durée de ceux-ci.