Découverte des tests unitaires avec PHPUnit

Les tests unitaires testent le code à l'échelle d'une fonction ou d'une classe. Le principe fondamental est de vérifier une fonctionnalité dans un environnement **contrôlé** de sorte que les résultats obtenus ne puissent être contestés.

Prenons le cas d'une fonction qui retourne une liste d'utilisateurs qui provient d'une base de données au format json. On peut imaginer ce type de fonction:

```php
<?php

namespace Solvolabs;

class UserProvider
{
    private $database;

    public function __construct(Database $database)
    {
        $this->database = $database;
    }

    public function getUsers()
    {
        $users = $this->database->execute('some-query');

        return json_encode($users);
    }
}
```

Si nous voulions tester le comportement de la fonction `getUsers` unitairement, il faut alors contrôler le résultat de la fonction `executer` de l'objet `Database`.

Avec le framework de test PHPUnit, vous allez voir comment on peut faire cela.

Installation de PHPUnit

Pour faire simple, vous allez récupérer le dépôt git suivant: https://github.com/mickaelandrieu/learn-testing-with-php-and-js.

Ensuite, il faudra vous assurer que PHP, Javascript et Composer sont bien installés sur votre machine.

> Vous pouvez utiliser l'image Docker de votre choix si vous ne souhaitez pas installer d'exécutables sur votre machine.

Pour cela, quelques commandes de vérification:

```
php -v
node -v
composer --version
```

Vous devriez avoir PHP d'installé sur votre environnement, pour nodejs vous consulterez la documentation officielle.

Pour installer Composer (le gestionnaire de dépendances de PHP), exécutez les instructions suivantes dans l'invite de commandes:

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') ===
'93b54496392c062774670ac18b134c3b3a95e5a5e5c8f1a9f115f203b75bf9a129d5daa8ba
6a13e2cc8a1da0806388a8') { echo 'Installer verified'; } else { echo
'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

Ceci ayant été fait, à la racine du dossier "unit" du dépôt "learn-testing-with-php-and-js", exécutez la commande suivante:

```
composer require --dev phpunit/phpunit:~5.7
```

Cette commande install PHPUnit en version 5.7 dans les dépendances de développement.

Ensuite, vous devriez pouvoir exécuter la commande suivante sans échec:

```
./vendor/bin/phpunit --version
```

Premier test avec PHPUnit

Ce framework de test fournit un ensemble d'assertions, de fonctions qui permettent de **valider l'adéquation** entre une situation courante et la situation espérée.

Imaginons un objet `BlogPost` qui a une fonction `isPublished`.

Nous pourrions tester différentes situations où l'article de blog doit être publié, ou non.

Avec PHPunit, c'est très simple.

```
<?php

namespace Tests\Solvolabs;
```

```php
use PHPUnit\Framework\TestCase;
use Solvolabs\BlogPost;

class BlogPostTest extends TestCase
{
    public function testPublication()
    {
        $blogPost = new BlogPost('Awesome news!');
        $this->assertFalse($blogPost->isPublished());

        $blogPost->publish();

        $this->assertTrue($blogPost->isPublished());
    }
}
```

Quelques informations importantes:

- le nom de la classe de test doit finir par `Test`;
- le nom des fonctions de test doivent commencer par `test`;
- nous étendons la classe `TestCase` pour avoir accès aux assertions;

> La liste des assertions disponibles sont dans la documentation.

TP 1: écrivez les tests de la classe Calculator

Dans le projet "Learn testing with PHP and Javascript", vous trouverez les classes `Calculator` et `CalculatorTest`.

Complétez la classe `CalculatorTest` de sorte à ce que toutes les fonctions soient testées!

Pour exécuter les tests:

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

Pour exécuter les tests de la seule classe Calculator:

```
./vendor/bin/phpunit --filter=CalculatorTest
```

Contrôler son environnement avec des doublures

Précédemment, nous disions qu'il fallait pouvoir contrôler le retour de la fonction `execute` d'un objet `Database`.

Pour cela, nous allons utiliser une fonctionnalité de PHPUnit appelée **Mock**.

Un Mock, c'est une doublure: un objet qui remplacera celui attendu et dont peut contrôler le comportement.

Voici par exemple comment "doubler" l'objet `Database`:

```php
<?php

namespace Tests\Solvolabs;

use PHPUnit\Framework\TestCase;
use Solvolabs\Exemple;
use Solvolabs\Database;

class BlogPostTest extends TestCase
{
    public function testGetUsers()
    {
        $databaseMock = $this->createMock(Database::class);
        $databaseMock->method('execute')->willReturn(
            [
                ['name' => 'Mickaël', 'age' => 32, 'title' => 'Architect'],
                ['name' => 'Rihana', 'age' => 30, 'title' => 'Singer'],
            ]
        );

        $exemple = new Exemple($databaseMock);

        $this->assertSame(
            $exemple->getUsers(),
            $this->expectedJsonOutput()
        );
    }

    private function expectedJsonOutput()
    {
        return '[{"name":"Mickaël","age":32,"title":"Architect"},
{"name":"Rihana","age":30,"title":"Singer"}]';
    }
}
```

Ici, notre test ne dépend pas de la base de données réelle! Donc même si celle-ci est cassée nous pouvons vérifier le comportement de la fonction `getUsers`. Nous nous assurons que le retour json soit en adéquation avec ce que nous attendons réellement. Nous avons exécuté du code en environnement contrôlé et garanti que la fonction `getUsers` fonctionnait.

> Oui mais... et si la fonction `execute` ne fonctionne pas?

Il faudrait également tester unitairement la fonction `execute` de la classe `Database` et confirmer qu'elle fonctionne.

Un test qui validerait le fonctionnement des deux fonctions en même temps serait un test dit "**d'intégration**".

> Nous écrirons des tests d'intégration dans un prochain TP.

TP 2: écrivez les tests de la classe UserProvider

Dans le projet "Learn testing with PHP and Javascript", vous trouverez les classes `UserProvider` et `UserProviderTest`.

Complétez la classe `UserProviderTest` de sorte à ce que toutes les fonctions soient testées!

Pour exécuter les tests:

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

Pour exécuter les tests de la seule classe UserProvider:

```
./vendor/bin/phpunit --filter=UserProviderTest
```

> Cette suite de TP n'a pas pour objectif de former au framework de test PHPUnit mais de comprendre comment fonctionnent les tests unitaires.