Parot Arnaud 2019-2021

E6

PHPUnit

Contexte

Dans le cadre d'un de mes stages professionnels, j'ai été chargé de la réalisation de tests unitaires et fonctionnels pour la société Delta CE tout en étant encadré par l'équipe de développeur présente pour répondre mes questions et me guider sur cette tâche.

La société souhaitait disposer de tests PHP pour faciliter le développement de nouvelles fonctionnalités sur son application de gestion de communauté d'entreprise afin de ne plus avoir besoin de tester manuellement les fonctionnalités après l'ajout d'une nouvelle ou la migration vers une nouvelle version de PHP.

Description de l'entreprise

Delta CE est une entreprise créer en 2012 dont l'activité est de développer et proposer son propre logiciel de gestion de communauté d'entreprise, l'entreprise propose donc à d'autre professionnels d'utiliser l'application pour gérer leur communauté d'entreprises moyennant un abonnement payant.

Le service propose les fonctionnalités suivantes :

- Système de chat textuel
- Echange de fichiers
- Conversations vocales
- Création de groupes et comité
- Administration facile d'un groupe ou comité
- Formule d'abonnement avec plus ou moins d'avantages celons le prix et les besoins
- Allocations et gestion faciles de planning et temps dédié au comité

En termes de clients, l'entreprise dispose de plus de 300 clients CSE à leur actif et plus de 250 000 utilisateurs à l'aube de leur 7 ans de maîtrise dans ce domaine.

Expression des besoins

L'entreprise Delta CE souhaite mettre en place un système de test sur leur application de gestion de comité d'entreprise afin de faciliter le déploiement de nouvelles fonctionnalité et de mise à jour, pour se faire j'ai dû utiliser le système de base de Laravel (PHPUnit) pour effectuer et développer ces tests.

Intérêt du projet

L'intérêt d'utiliser un système de test tel que PHPUnit est de simplifier les tests nécessaires à la vérification lors de développement de nouvelles fonctionnalités ou lors d'un changement de version de PHP au niveau de l'application.

Cela évite les tests manuels qui peuvent d'avérer long et qui peuvent s'apparenter à une perte de temps car cela demande beaucoup de temps si l'application dispose de nombreuses fonctionnalités.

Avantage de PHPUnit

L'utilisation de PHPUnit dispose de nombreux avantages, en voici quelques-uns principaux pour le projet réalisé :

- Permet les tests en PHP
- Assez simple d'utilisation
- Dispose d'un bon nombre de fonctions de test
- Permet de tester presque tout
- Permet de tester les requêtes http par des api
- Prise en charge de tests unitaires et fonctionnels

Inconvénients de PHPUnit

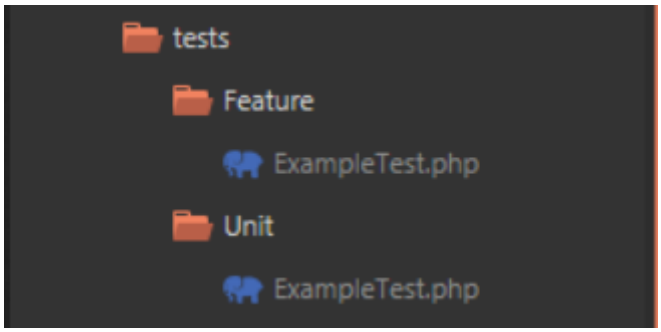
L'utilisation de PHPUnit présente néanmoins quelques inconvénients :

- Certains tests sont néanmoins difficiles à réaliser
- PHP n'est de base pas fait pour réaliser des tests

Réalisation et mise en pratique

Installation

Dans le cadre su projet réaliser en entreprise, il m'a fallu tout d'abord installer Laravel pour pouvoir effectuer les tests désirés, à la suite de l'installation du Framework, 2 dossiers sont créés par défaut dans la catégorie « Test ». Un des dossiers se nomme « Unit », c'est dans celui-ci que nous allons regrouper tous nos tests unitaires, c'est-à-dire tous nos tests qui ont pour but de tester une partie spécifique du code bien précise. L'autre dossier créé se nomme « Feature », c'est dans ce dossiers que seront regroupé les tests principaux permettant de tester une fonctionnalité entière comme tester des envois de mails, tester des appels de fonctions renvoyant des réponses complexe tel que fonctions de connexions ou de vérification de mots de passe. Les 2 dossiers sont par ailleurs fournis avec un fichier d'exemple « ExempleTest.php » qui permet de regarder comment fonctionne un test unitaire et fonctionnel.



Créer un test

Pour créer un test, il suffit de passer par la console et utiliser la commande de PHP Artisan suivante : « make:test » suivi du nom du test que l'on souhaite effectuer, cette commande créera par défaut un test fonctionnel si rien n'est préciser, le nom du test sera également le nom du fichier et sera par défaut ajouté au dossier « Feature ».

```
php artisan make:test UserTest
```

Si l'on souhaite créer un test unitaire, il suffit d'utiliser la même commande suivie de l'instruction « --unit » en bout de ligne. Cette instruction aura pour effet de créer un test cette fois unitaire et placera le test dans le dossier « Unit » et non plus « Feature ».

```
php artisan make:test UserTest --unit
```

Composition et lancement d'un test

Un test, qu'il soit unitaire ou fonctionnel se compose de la façon suivante :

```php
<?php

namespace Tests\Unit;

use PHPUnit\Framework\TestCase;

class ExampleTest extends TestCase
{
    /**
     * A basic test example.
     *
     * @return void
     */
    public function test_basic_test()
    {
        $this->assertTrue(true);
    }
}
```

Il est composé d'un import permettant d'importer la class « TestCase », de bases nécessaires à la réalisation d'un test. Notre test se compose ensuit d'une class disposant du nom donné au test lors de sa création qui hérite de la class « TestCase », indispensable à la réalisation et au fonctionnement des tests sous PHPUnit.
Ensuite, nous somme libre de réaliser les test que l'on souhaites, chaque test doit être écrit dans une fonctions et un test est réaliser par une méthode commençant par « $this->assert… » puis ensuit par ce que l'on souhaite tester, ici sur l'exemple on souhaite savoir si la fonction nous renvois bien une valeur True, pour cela nous utilisons donc la fonction assertTrue(), il y a des dizaines de fonctions comme celle-ci pour permettre de tester ce que l'on veut le plus simplement possible.

Il y a une multitude de test disponible grâce à PHPUnit, en plus des tests basiques nous avons accès à des test HTTP permettant d'effectuer des tests sur les résultats envoyés par des api ou des services via des requêtes, des tests permettant d'effectuer des tests en console ainsi que des tests permettant d'effectuer des tests sur le navigateur.

Une fonctionnalité très utile aussi est le mocking, permettant de simuler l'appel à une fonction d'autre class ou de simuler un appel à une api par exemple, une class utilisant le mocking se compose de cette façon :

```
use App\Service;
use Mockery;
use Mockery\MockInterface;

public function test_something_can_be_mocked()
{
    $this->instance(
        Service::class,
        Mockery::mock(Service::class, function (MockInterface $mock) {
            $mock->shouldReceive('process')->once();
        })
    );
}
```

Lorsque le programme va tester une fonction qui utilise le mocking, elle va se comporter comme si elle avait été appelée et va renvoyer des valeurs que l'on aura définit ou simplement juste passer la méthode sans l'exécuter, mais le programme aura l'impression qu'elle aura été exécutée.

Pour lancer un test il suffit d'utiliser la commande de PHPUnit suivante :

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

On peut également choisir en plus de lancer le test via Artisan pour permettre un débogage plus facile en cas de bug :

```
php artisan test
```

Le résultat sera ensuite affiché de cette façon en indiquant le nombre de tests qui ont été réaliser ainsi que le nombre d'erreur rencontrer pendant les tests s'il y en a, c'est-à-dire le nombre de test qui ont reçu un résultat différent que le résultat attendu lors de la réalisation du code.