
Documentation Symfony (Anglais)

Tests unitaire et fonctionnels

Présentation test unitaire et fonctionnel

Un test unitaire est un test qui ne vérifie qu'une fonction isolée d'un programme. Ce qui signifie que la fonction ne doit pas dépendre d'une classe ou fonction autre que la librairie standard (les fonctions PHP classique), et ne doit pas dépendre de la présence d'une base de données ou d'un fichier de configuration particulier.

Inversement, un test fonctionnel va vérifier la fonctionnalité d'une application de la même manière qu'un utilisateur normal via un script dédié. Par exemple, la création d'un nouvel agent, la génération d'un tableau de statistique ou l'envoi d'un mail sont testés via un test fonctionnel.

À la différence d'un test unitaire, le test fonctionnel dépend d'un environnement de test valide, la qualité du test fonctionnel est directement dépendante de l'environnement de test et des jeux de test utilisés.

Configuration de PHPUnit

Tout d'abord, nous avons besoin de télécharger PHPUnit qui se charge de l'exécution de nos test,

Nous allons utiliser la version 4.8, car notre version de PHP est la 5.4. Les version plus récente de PHPUnit ne sont disponibles qu'a partir de la version 5.6 de PHP.

https://phpunit.de/

Télécharger le fichier PHAR et de la même manière que pour composer, créer un fichier phpunit.bat que l'on ajoute à notre PATH, voir le tutorial composer

vérifier que en ligne de commande on peut exécuter la commande *phpunit*.

Nous allons maintenant ajouter un fichier de configuration pour phpunit dans le répertoire **app** de Symfony. Créer un fichier **phpunit.xml.dist** avec le contenu suivant :

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- http://phpunit.de/manual/4.1/en/appendixes.configuration.html -->
<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:noNamespaceSchemaLocation="http://schema.phpunit.de/4.1/phpunit.xsd"
                 backupGlobals="false"
                 colors="true"
                 bootstrap="bootstrap.php.cache"
>
        <testsuites>
                <testsuite name="Atelier Symfony">
                        <directory>../src/AppBundle/Tests</directory>
                </testsuite>
        </testsuites>

</phpunit>
```

Ceci fait, créer un répertoire **Tests** dans le répertoire de notre bundle. C'est dans ce répertoire que nous allons ajouter nos tests.

Création d'un premier test

Pour notre premier test, nous allons créer un test unitaire qui vérifie si un chiffre est multiple de 2 ou non, créer un nouveau fichier **TestUnitaire.php** dans le répertoire **Test** et ajouter le code suivant.

```
<?php
namespace AppBundle\Tests;
```

```
class TestUnitaire
{

        protected function setUp()
        {

        }


        private function multiple($value,$multiple)
        {
                if(($value % $multiple) == 0)
                        return true;
                else
                        return false;
        }



        public function testmultiple2()
        {

                $result = $this->multiple(8,2);


                $this->assertTrue($result);
        }

        public function testnonmultiple2()
        {
                $result = $this->multiple(3,2);

                $this->assertFalse($result);
        }

        protected function tearDown()
        {
```

```
            parent::tearDown();
        }


}
```

la première fonction teste que 8 est bien multiple de 2, la seconde test que 3 n'est bien pas multiple de 2. Les fonction de type **assertTrue** et **assertFalse** vérifie la valeur de notre résultat ainsi **assertTrue(true)** ne renverra pas d'erreur car la valeur en paramètre en bien true, alors que **assertTrue(false)** renverra une erreur et le test sera donc considéré comme échoué.

La fonction **setUp()** est toujours appelée avant l'exécution d'un test, la fonction **tearDown()** est toujours appelée après l'exécution d'un test.

Maintenant, ouvrons un invite de commande sur le répertoire racine de l'application Symfony et saisissons la commande suivante :

```
phpunit -c app/ > test_output.html
```

cela va exécuter nos tests et stocker le résultat dans un fichier **test_output.html**.

Test fonctionnel de notre interface de création d'agent

Symfony dispose d'un webcrawler, qui permet d'automatiser les interactions utilisateur avec une page web. Pour cela, nous allons créer un nouveau fichier test nommé **AgentTest.php** avec le contenu suivant.

documentation officielle (en anglais) : http://symfony.com/doc/current/testing.html

```
<?php
namespace AppBundle\Tests;
```

```
use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Component\DomCrawler\Crawler;


class AgentTest extends WebTestCase
{

        protected function setUp()
        {
                self::bootKernel();
        }


        public function testNouvelAgent()
        {
           $client = static::createClient();

           $crawler = $client->request('GET','/formulaire');

           $form = $crawler->filter('form')->form();



                $form['form[nom]'] = 'Agent nom';

                $form['form[prenom]'] = 'Agent prénom';

                $form['form[dateentree][month]'] = '2';
                $form['form[dateentree][day]'] = '2';
                $form['form[dateentree][year]'] = '2';

                $client->submit($form);

                $content =  $client->getResponse()->getContent();

                //vérification de nos données postés
                $this->assertContains('Agent nom',$content);
```

```
                $this->assertContains('Agent prénom',$content);
                $this->assertContains('03/03/2013',$content);
                $this->assertContains('Donnée disponible',$content);
        }


        protected function tearDown()
        {
                parent::tearDown();
        }
}
```

Ce code créer un nouveau crawler et effectue une requête à l'adresse *formulaire* de notre application, puis il filtre sur l'élément form et rempli les champs de formulaire avec nos données de test.

Enfin, **$client->submit($form)** envoie nos données au serveur. Nous vérifions ensuite à partir de la réponse si nos données de test sont bien présentes sur la page.

- Previous
- Next