Programmation orientée objet en PHP

Premier test

PHPUnit est une bibliothèque qui va nous permettre de tester nos composants logiciels. Nous pourrons ainsi vérifier le bon fonctionnement de notre application à plusieurs niveaux :

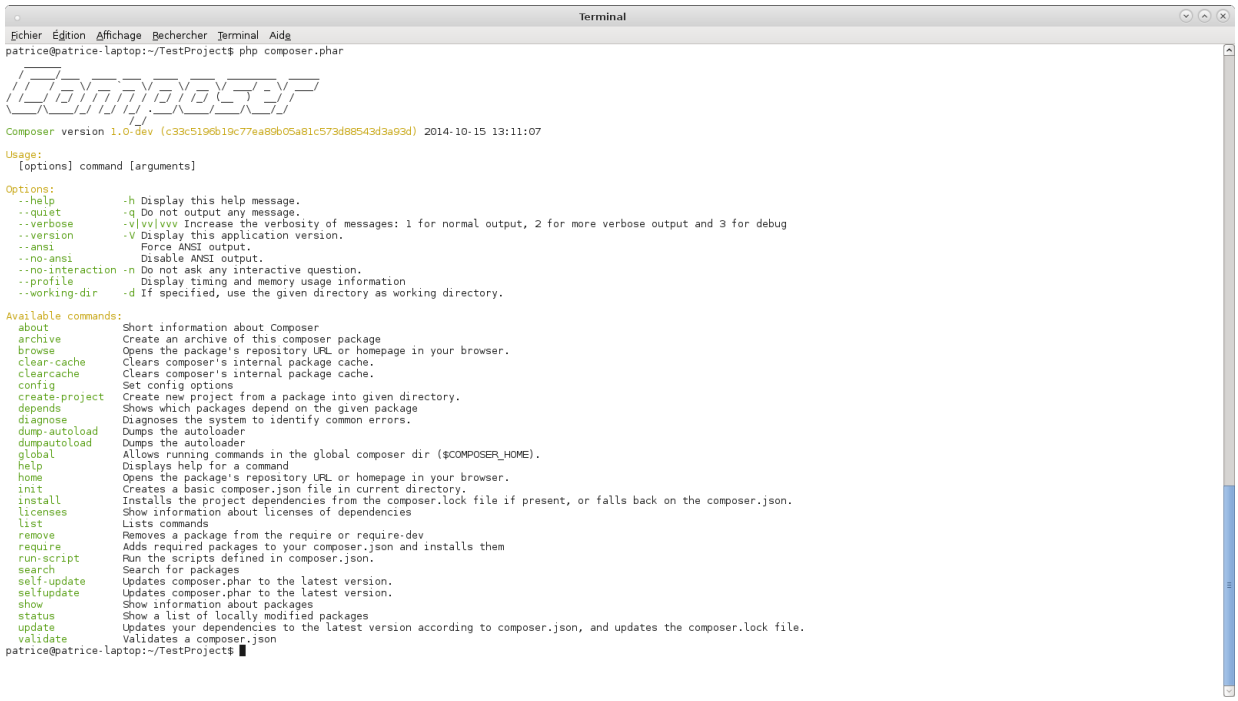
- au niveau du composant lui-même : on parle ici de test unitaire, on essaie de tester la plus petite unité de code possible (souvent une fonction)
- au niveau de plusieurs composants : on vérifie ici que l'intégration de nouveaux composants n'entraîne pas un dysfonctionnement des autres composants : on parle ici de test d'intégration
- au niveau de l'application elle-même : on vérifie le bon fonctionnement d'une fonctionnalité entière : on parle ici de test fonctionnel

Les tests permettent aussi de mettre en évidence des problèmes en terme de performance, de robustesse, de fiabilité, ou encore de vulnérabilité.

L'installation de PHPUnit sera assurée par un outil de gestion de dépendances appelé Composer. Pour le dire vite, Composer va nous permettre de télécharger les bibliothèques dont nous aurons besoin dans notre application. Ces bibliothèques seront renseignées dans un fichier JSON. L'installation de Composer peut se faire de plusieurs façons, soit sur votre système, soit dans votre projet sous forme d'une archive phar. Dans notre projet nous utiliserons l'archive phar.

Nous créons un projet TestProject dans notre répertoire personnel, et nous mettons à la racine de notre projet l'archive composer.phar téléchargée à partir du site :

Dans un terminal, nous nous plaçons dans le répertoire de notre projet et exécutons la commande *php composer.php.*



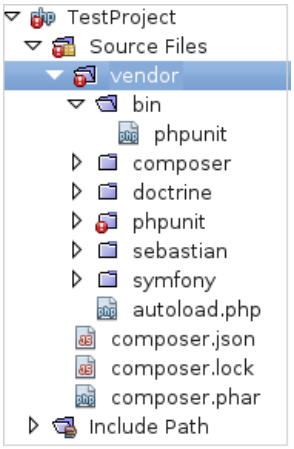
Nous obtenons ainsi la liste des commandes de Composer.

Nous éditons à la racine de notre projet un fichier composer.json qui indiquera toutes les dépendances que devra

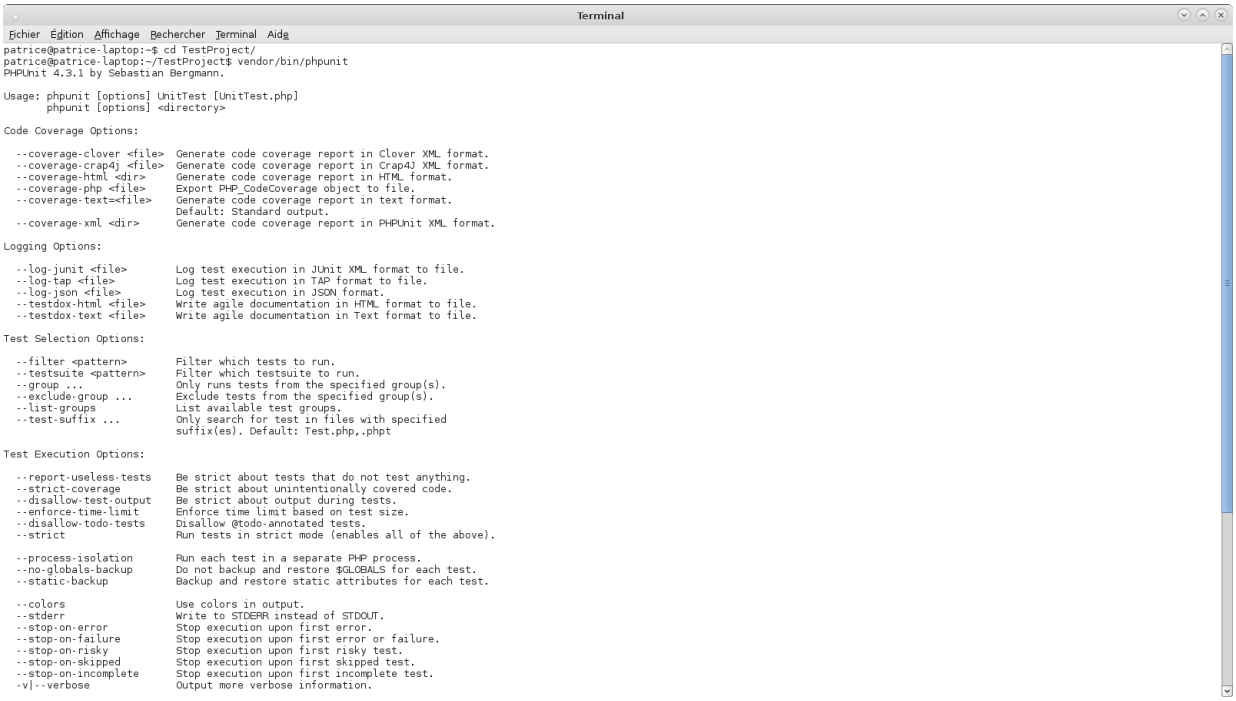
gérer Composer :

```
{
    "require-dev": {
        "phpunit/phpunit": "4.3.*"
    }
}
```

Dans le fichier composer.json, le wildcard "*" signifie la version la plus élevée. Dans un terminal nous exécutons la commande *php composer.phar install* et laissons Composer installer nos dépendances. Un répertoire vendor contenant nos dépendances a été créé au sein de notre projet :



Dans le répertoire vendor/bin nous trouvons un exécutable phpunit. C'est cet exécutable qui va nous permettre de lancer nos tests. Si, dans le terminal, nous exécutons la commande *vendor/bin/phpunit* alors s'afficheront les commandes de phpunit. Quand nous exécutons cette commande nous sommes bien à la racine de notre projet. À chaque fois que nous lancerons nos tests nous veillerons à être à la racine de notre projet.

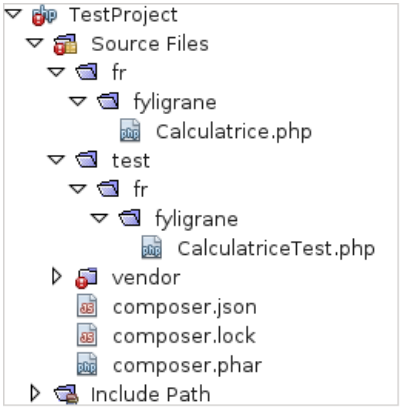


Nous allons créer une classe Calculatrice dans un répertoire fr/fyligrane, et tester ses méthodes. Nous allons tester les plus petites unités de code possible, nous faisons donc du test unitaire.

```
<?php

// fr/fyligrane/Calculatrice.php

namespace fr\fyligrane;

class Calculatrice {

```

```php
    public function additionner($nb1, $nb2) {
        return $nb1 + $nb2;
    }

}
```

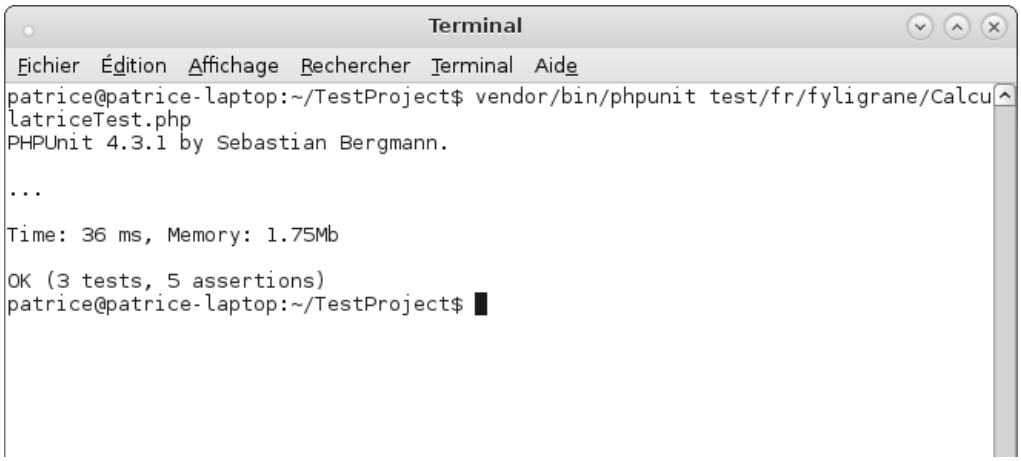
Cette classe ne contient pour l'instant qu'une méthode additionner() qui prend en arguement deux entiers et qui renverra bien sûr la somme de ces deux entiers. Tester le bon fonctionnement de cette méthode revient à vérifier que l'entier retourné est bien égal à la somme des deux entiers passés en argument.
Nous allons créer dans un répertoire test/fr/fyligrane, une classe CalculatriceTest héritant de PHPUnit_Framework_TestCase.



Dans cette classe nous allons créer une méthode testAdditionner() qui effectuera le test de notre méthode additionner(). La classe PHPUnit_Framework_TestCase possède une méthode assertEquals() qui permet de vérifier l'égalité de deux valeurs ou variables : `assertEquals(mixed $expected, mixed $actual[, string $message = ''])` .
Pour exécuter la fonction additonner() nous aurons évidemment besoin d'une instance de Calculatrice.

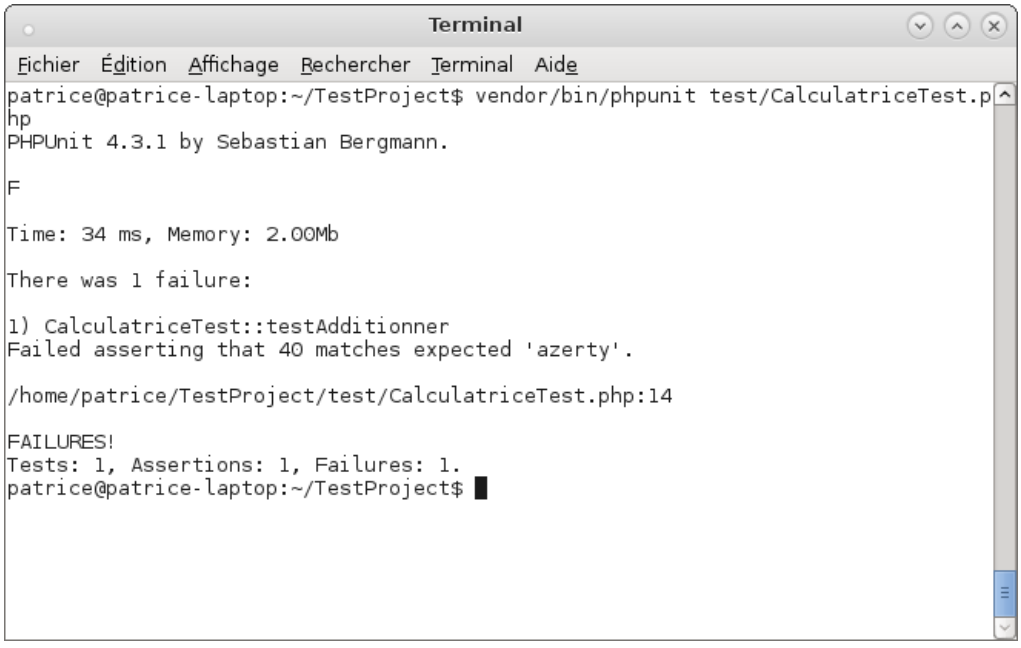
```php
<?php

// test/fr/fyligrane/CalculatriceTest.php

include_once './fr/fyligrane/Calculatrice.php';

use fr\fyligrane\Calculatrice;

class CalculatriceTest extends PHPUnit_Framework_TestCase {

    public function testAdditionner() {
        $calc = new Calculatrice();
        $result = $calc->additionner(15, 25);
        $this->assertEquals(40, $result);
    }

}
```

Pour exécuter ce test nous allons utiliser l'exécutable phpunit et lui passer en paramètre le fichier à tester :



En cas d'erreur, phpunit affiche quelques informations :

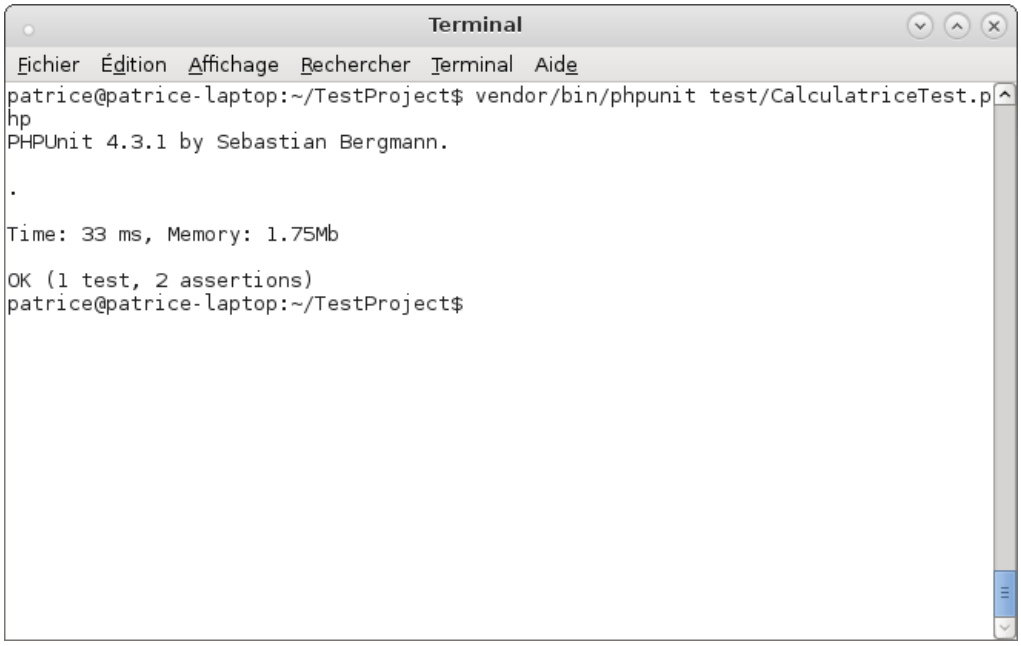
```php
<?php

// test/fr/fyligrane/CalculatriceTest.php

include_once './fr/fyligrane/Calculatrice.php';

use fr\fyligrane\Calculatrice;

class CalculatriceTest extends PHPUnit_Framework_TestCase {

    public function testAdditionner() {
        $calc = new Calculatrice();
        $result = $calc->additionner(15, 25);
        $this->assertEquals("azerty", $result);
    }

}
```



Un test peut comporter plusieurs "assertions" ; dans ce cas le test ne sera concluant que si toutes les assertions se vérifient :

```php
<?php

// test/fr/fyligrane/CalculatriceTest.php

include_once './fr/fyligrane/Calculatrice.php';

use fr\fyligrane\Calculatrice;

class CalculatriceTest extends PHPUnit_Framework_TestCase {

    public function testAdditionner() {
        $calc = new Calculatrice();
        $result = $calc->additionner(15, 25);
        $this->assertEquals(40, $result);
```

```
15          // stupide... Juste pour l'exemple...
16          $this->assertNotFalse($result);
17      }
18
19  }
```



La classe PHPUnit_Framework_TestCase possède un nombre important d'assertions sous la forme de méthode *assert******()*.

Pour que phpunit sache quelles méthodes correspondent à un test, il faut que le nom de ces méthodes commence par test (ici testAdditionner()), ou alors que la méthode porte l'annotation @test (les annotations se situent toujours dans des commentaire de la forme /**.....*/).

```
1   <?php
2
3   // test/fr/fyligrane/CalculatriceTest.php
4
5   include_once './fr/fyligrane/Calculatrice.php';
6
7   use fr\fyligrane\Calculatrice;
8
9   class CalculatriceTest extends PHPUnit_Framework_TestCase {
10
11      /**
12       * @test
13       */
14      public function additionner() {
15          $calc = new Calculatrice();
16          $result = $calc->additionner(15, 25);
17          $this->assertEquals(40, $result);
18          // stupide... Juste pour l'exemple...
19          $this->assertNotFalse($result);
20      }
21
22  }
```

Fin

NAVIGATION

Accueil

ADMINISTRATION

Connecté sous le nom « Arnaud Lemais » (Déconnexion)
POO PHP