

Tests fonctionnels automatisés

L'automatisation des tests fonctionnels consiste à créer des scénarii de tests qui pourront ensuite être reproduits à la demande (exécutés) au cours du développement.

- La réalisation d'un scénario permet de mieux identifier le besoin exprimé
- La mise en place du test relatif à une fonctionnalité permet de tester cette fonctionnalité (en cours et en fin d'implémentation)
- L'exécution d'une suite de tests permet de vérifier la non-régression d'un projet suite à une modification ou l'introduction d'une nouvelle fonctionnalité.

-- Mise en place des outils pour PHP

-- Composer

Composer est un gestionnaire de paquets compatible GIT permettant d'installer ou de mettre à jour les librairies incluses dans un projet à partir d'un fichier de configuration **composer.json**, déclarant les dépendances du projet.

Installation

Sous Windows :

- télécharger et installer [Composer-Setup.exe](#)
- Ajouter le dossier d'installation de composer dans la variable PATH de windows pour pouvoir exécuter composer directement en ligne de commande.

Vérifier l'installation :

Dans un terminal : Frapper **composer -v** puis Entrée ←



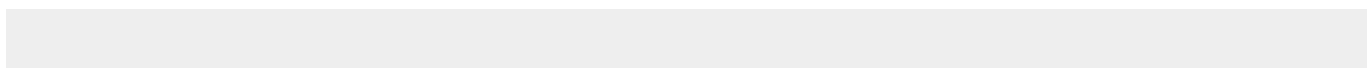
```
C:\Users\jc>composer -v  
  
Composer version 1.0-dev (c41079192f38f0fc446b17baa8f628dcb3b61e7d) 2015-09-28 09:38:16
```

-- PHPUnit et WebDriver

PHPUnit va permettre de réaliser des tests unitaires (différents des tests fonctionnels).

Pour la partie fonctionnelle, nous utiliserons **Selenium Server** + **Facebook WebDriver**, pour émuler les interactions utilisateur dans un navigateur.

Créer le fichier **composer.json** à la racine de votre projet :



```
{
  "require-dev": {
    "facebook/webdriver": "dev-master",
    "phpunit/phpunit": "~4.8"
  }
}
```

Dans le terminal :

A partir du dossier de votre projet, Frapper composer install puis Entrée ←

Vérifiez l'installation correcte des packages dans le dossier **vendor** du projet.

-- PhpUnit

PhpUnit permet de réaliser des tests unitaires, relatifs à la bonne exécution de parties de code (fonction ou méthode).

-- Configuration

Créer un dossier **tests/** à la racine du projet à tester.

Créer le fichier [TestHelper.php](#) adapté qui permettra de faire les inclusions nécessaires avant le lancement de PhpUnit.

Créer le fichier **PHPunit.xml** dans le même dossier, faisant référence au fichier TestHelper.php, et permettant de lancer tous les tests inclus dans le dossier courant (./) et ses sous-dossiers :

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit bootstrap="TestHelper.php"
  backupGlobals="false"
  backupStaticAttributes="false"
  verbose="true"
  colors="true"
  convertErrorsToExceptions="true"
  convertNoticesToExceptions="true"
  convertWarningsToExceptions="true"
  processIsolation="false"
  stopOnFailure="false"
  syntaxCheck="true">
  <testsuite name="Testsuite">
    <directory>./</directory>
  </testsuite>
</phpunit>
```

Créer un premier test (dans le cas présent, on utilise le micro-framework):

```
<?php
```

```
use micro\orm\DAO;
class DAOTest extends \PHPUnit_Framework_TestCase{

    /* (non-PHPdoc)
     * @see PHPUnit_Framework_TestCase::setUp()
     */
    protected function setUp() {
        global $config;
        Logger::init();

        extract($config["database"]);
        $db=$config["database"];
        DAO::connect($db["dbName"],@$db["serverName"],@$db["port"],@$db["user"],@$db["password"]);
    }

    private function loadOne($className){
        $objects=DAO::getAll($className);
        $this->assertArrayHasKey(0, $objects);
        $this->assertNotNull($objects[0]);
        $o=DAO::getOne($className, $objects[0]->getId());
        $this->assertNotNull($o);
        $this->assertEquals($objects[0]->getId(), $o->getId());
    }
    public function testLoadModels(){
        $models=["Ticket","Faq","Message","Statut","User","Categorie"];
        foreach ($models as $model){
            $this->loadOne($model);
        }
    }
}
```

Aller en invite de commandes dans le dossier tests du projet et exécutez :

phpunit puis

```
C:\xampp\htdocs\helpdesk\tests>phpunit
PHPUnit 4.8.9 by Sebastian Bergmann and contributors.

Runtime:       PHP 5.6.3 with Xdebug 2.2.6
Configuration: C:\xampp\htdocs\helpdesk\tests\PHPUnit.xml

..

Time: 15.47 seconds, Memory: 4.50Mb

OK (2 tests, 26 assertions)

C:\xampp\htdocs\helpdesk\tests>
```

-- Selenium

Selenium permet de réaliser des tests fonctionnels. Il permet de réaliser ce que l'utilisateur pourrait entreprendre, pour mettre en oeuvre une fonctionnalité, puis de tester les résultats obtenus.

-- Configuration

From:

<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:

<http://slamwiki2.kobject.net/slam5/testsfonctionnels/automatises?rev=1443548681>

Last update: **2019/08/31 14:37**

