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 ↵



-- 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>
```

-- Premier test

Créer un premier test (juste pour vérifier le bon fonctionnement de PHPunit):

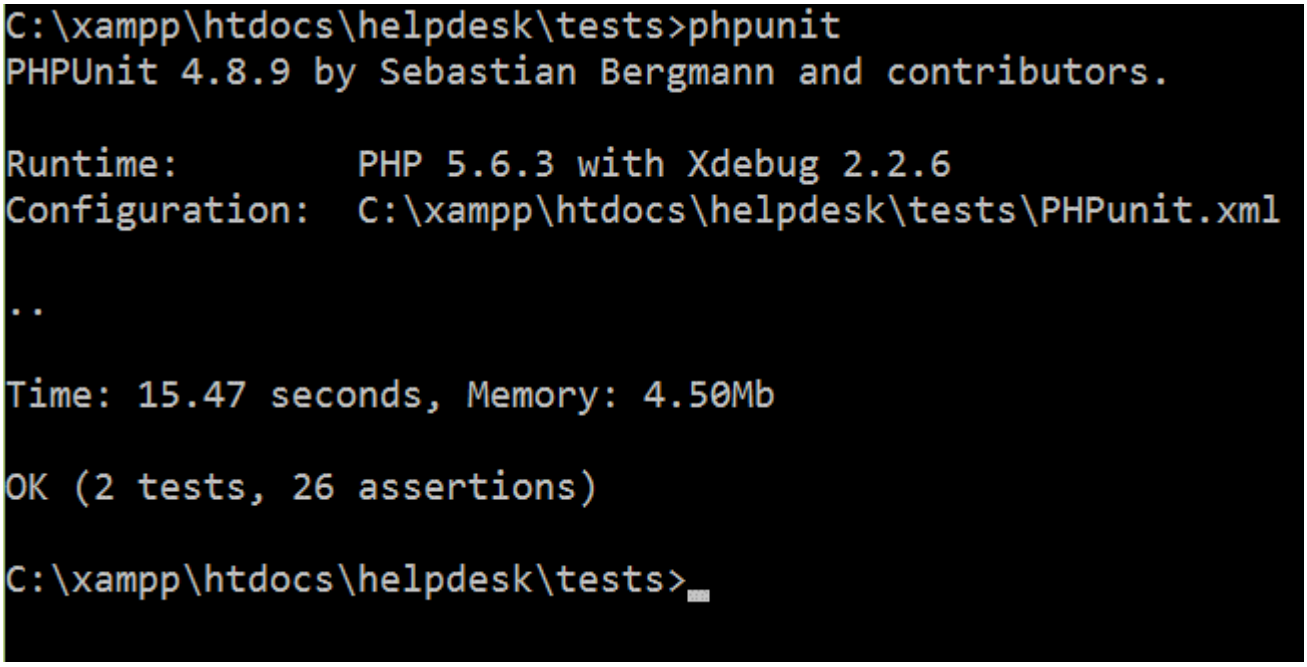
```
<?php
class PHPUnitTest extends \PHPUnit_Framework_TestCase {
    private $variable;
    /* (non-PHPdoc)
     * @see PHPUnit_Framework_TestCase::setUp()
     */
    protected function setUp() {
        $this->variable=1;
    }

    public function testIncVariable(){
        $this->assertEquals($this->variable, 1);
        for($i=0;$i<10;$i++){
            $this->variable+=1;
        }
        $this->assertEquals(11, $this->variable);
    }

    public function testVariable(){
        $this->assertEquals($this->variable, 1);
    }
    /* (non-PHPdoc)
     * @see PHPUnit_Framework_TestCase::tearDown()
     */
    protected function tearDown() {
        $this->variable=0;
    }
}
```

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

phpunit puis Entrée ↵



-- Caractéristiques d'une classe de test PHPunit

1. Une classe PHPunit dérive de **PHPUnit_Framework_TestCase**. Son nom doit commencer par **Test...**
2. Les méthodes de test contenues dans la classe sont publiques et se terminent par **...test**
3. Les méthodes qu'il est possible de surdéfinir :
    1. **setUpBeforeClass** ⇒ exécutée une seule fois avant l'appel du constructeur de la classe
    2. **setUp** ⇒ exécutée avant chaque test (méthode se terminant par ...test)
    3. **tearDownAfterClass** ⇒ exécutée une seule fois après le dernier tearDown
    4. **tearDown** ⇒ exécutée après chaque test (méthode se terminant par ...test)

-- 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

-- Selenium Server

Le serveur Selenium permet de contrôler les navigateurs, pour simuler les interactions entre l'utilisateur et l'application Web.
Selenium peut fonctionner avec les navigateurs disposant du webDriver adéquat associé.

1. Télécharger Selenium Server Standalone sur http://docs.seleniumhq.org/download/
2. Télécharger en même temps et au même emplacement le webDriver pour Chrome (chromeDriver.exe)

Création d'un fichier de démarrage du serveur :

Créer un dossier **server** dans le projet Web :

1. Copier le fichier **selenium-server-standalone-xxxx.jar** dans ce dossier
2. copier **chromeDriver.exe** dans un sous dossier driver
3. créer dans le dossier **server** le fichier **start-server.bat** :

```
java -jar selenium-server-standalone-2.47.1.jar -
Dwebdriver.chrome.driver=driver/chromedriver.exe
```

En invite de commande, Tester le lancement du serveur Selenium :

Frapper **start-server.bat** puis Entrée ↵

```
Setting system property webdriver.chrome.driver to driver/chromedriver.exe      13:59
:47.119 INFO - Java: Oracle Corporation 25.60-b23
13:59:47.120 INFO - OS: Windows 10 10.0 amd64                                   13:59
:47.127 INFO - v2.47.1, with Core v2.47.1. Built from revision 411b314
13:59:47.176 INFO - Driver class not found: com.opera.core.systems.OperaDriver  13:59
:47.177 INFO - Driver provider com.opera.core.systems.OperaDriver is not registered
                                                                  13:59:47.271 IN
FO - RemoteWebDriver instances should connect to: http://127.0.0.1:4444/wd/hub
                                                        13:59:47.271 INFO - Selen
ium Server is up and running
```

Frapper CTRL+C pour l'arrêter

-- PHP webServer

Pour les tests, nous utiliserons PHP en tant que serveur Web (plutôt que Apache) :

Démarrage de PHP en tant que serveur Web :

```
php -S 127.0.0.1:8090
```

Création d'un fichier de routage
PHP ne supportant pas les .htaccess comme apache et n'ayant pas de module de réécriture d'url, il est nécessaire d'émuler un pseudo-routage :

Pour le cas : micro-framework :
Créer le fichier **.htrouter.php** dans le dossier app de votre application web :

```
<?php
if (preg_match('/\.(?:png|jpg|jpeg|gif|ttf|eot|svg|woff|woff2|js|css)$/',
$_SERVER["REQUEST_URI"])) {
    return false;
} else {
    $_GET["c"]=substr($_SERVER["REQUEST_URI"],1);
    include __DIR__ . '/startup.php';
}
```

Création d'un fichier de lancement du server php
Créer le fichier startPhpServer.bat dans le dossier racine de votre application :

```
php -S 127.0.0.1:8090 app/.htrouter.php
```

Aller à l'adresse 127.0.0.1:8090 de votre navigateur pour tester la réponse.

-- Préparation Selenium

Créer la classe **AjaxUnitTest** dans le dossier **tests** pour faciliter la manipulation de l'objet webDriver :

```
<?php

/**
 * Class AjaxUnitTest
 */
abstract class AjaxUnitTest extends UnitTestCase {
    use \WebDriverAssertions;
    use \WebDriverDevelop;
    protected static $url = 'http://127.0.0.1:8090/';
    /**
    * @var \RemoteWebDriver
```

```
    */
    protected static $webDriver;


    /* (non-PHPdoc)
     * @see PHPUnit_Framework_TestCase::setUpBeforeClass()
     */
    public static function setUpBeforeClass() {
        $capabilities = array(\WebDriverCapabilityType::BROWSER_NAME =>
'firefox',\WebDriverCapabilityType::VERSION=>'41.0');
        self::$webDriver = \RemoteWebDriver::create('http://localhost:4444/wd/hub',
$capabilities);
    }

    public function setUp() {
        parent::setup();
    }

    /* (non-PHPdoc)
     * @see PHPUnit_Framework_TestCase::tearDownAfterClass()
     */
    public static function tearDownAfterClass() {
        if(self::$webDriver!=null)
            self::$webDriver->close();
    }


    /**
     * Loads the relative url $url in web browser
     * @param string $url
     */
    public static function get($url=""){
        $url=self::$url.$url;
        self::$webDriver->get($url);
    }

    /**
     * Returns a given element by id
     * @param string $id HTML id attribute of the element to return
     * @return RemoteWebElement
     */
    public function getElementById($id){
        return self::$webDriver->findElement(\WebDriverBy::id($id));
    }

    /**
     * Tests if an element exist
     * @param string $css_selector
     * @return boolean
     */
    public function elementExists($css_selector){
        return sizeof($this->getElementsBySelector($css_selector))!==0;
    }

    /**
     * Returns a given element by css selector
```

```
     * @param string $css_selector
     * @return RemoteWebElement
     */
    public function getElementBySelector($css_selector){
        return
self::$webDriver->findElement(\WebDriverBy::cssSelector($css_selector));
    }

    /**
     * Returns the given elements by css selector
     * @param string $css_selector
     * @return RemoteWebElement
     */
    public function getElementsBySelector($css_selector){
        return
self::$webDriver->findElements(\WebDriverBy::cssSelector($css_selector));
    }

    /**
     * Return true if the actual page contains $text
     * @param string $text The text to search for
     */
    public function assertPageContainsText($text){
        $this->assertContains($text, self::$webDriver->getPageSource());
    }
}
```

-- Tests Selenium

-- Création des pages à tester (pour l'exemple)

Cas avec utilisation Micro-framework (à adapter pour d'autres utilisations) :

Créer une page affichant “Hello Selenium”, et un formulaire **frm** disposant d'une zone de texte **text**.
Sur le POST du formulaire, l'action est renvoyée vers une page affichant le résultat du post.

```
<?php
use micro\controllers\BaseController;
use micro\utils\RequestUtils;
class Selenium extends BaseController {

    public function __construct() {
        parent::__construct();
    }

    public function index() {
        $this->initialize();
        echo "<h1>Hello Selenium</h1>";
        echo "<form method='POST' action='Selenium/post' name='frm' id='frm'>";
        echo "<input type='text' name='text' id='text'>";
    }

    public function post(){
```

```
        if(RequestUtils::isPost()){
            echo "<div id='result'>".$_POST['text']."</div>";
        }
    }
}
```

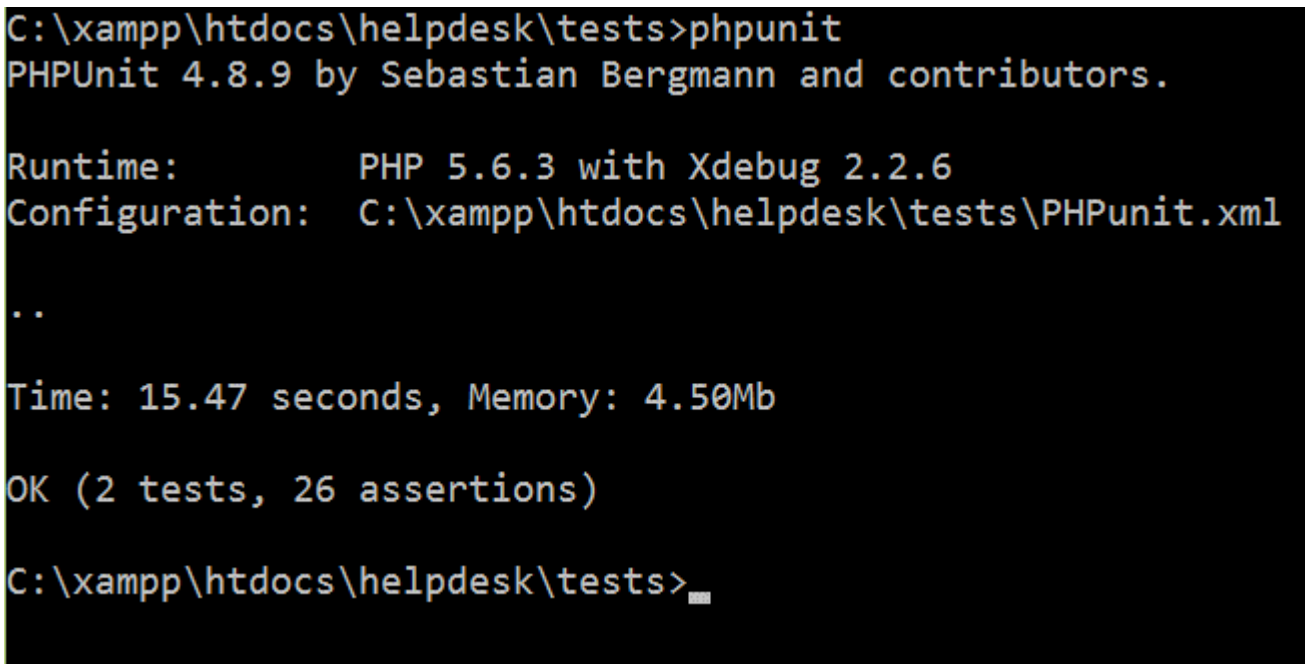
-- Création des tests

Nous allons tester que la page intiale selenium/ affiche bien “Hello Selenium”
Puis qu'une saisie et une validation permettent bien d'afficher la réponse dans la page suivante.

```
<?php
class SeleniumTest extends \AjaxUnitTest{
    public static function setUpBeforeClass() {
        parent::setUpBeforeClass();
        self::get("Selenium/index");
    }
    public function testDefault(){
        $this->assertPageContainsText('Hello Selenium');
        $this->assertTrue($this->elementExists("#text"));
        $this->assertTrue($this->elementExists("#frm"));
    }
    public function testValidation(){
        $this->getElementById("text")->sendKeys("okay");
        $this->getElementById("text")->sendKeys("\xEE\x80\x87");
        SeleniumTest::$webDriver->manage()->timeouts()->implicitlyWait(5);
        $this->assertEquals("okay",$this->getElementById("result")->getText());
    }
}
```

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

phpunit puis Entrée ↵



1. Le test fait une requête (get) vers la page /selenium/index
2. Saisi **okay** dans la zone de texte d'id text
3. Valide le formulaire **frm** par la touche ENTREE
4. Vérifie que la page résultat /selenium/post affiche bien le résultat **okay** dans la div d'id **result**

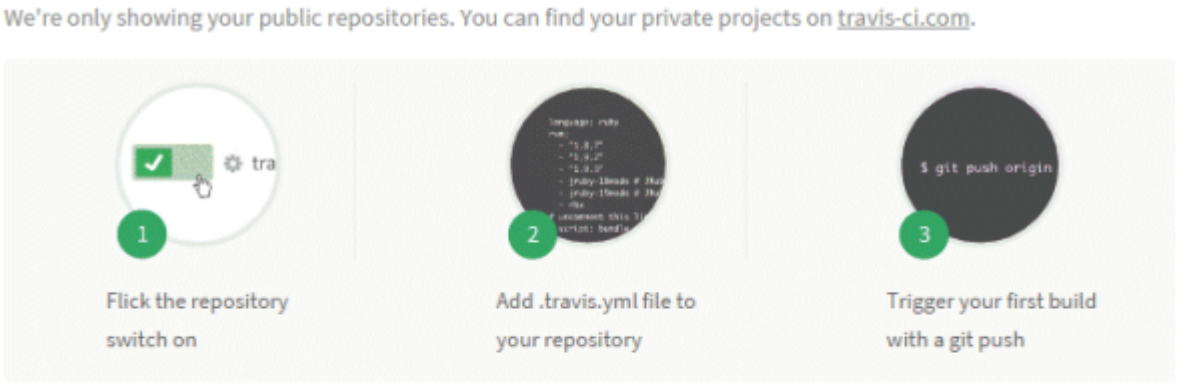
-- Travis-ci

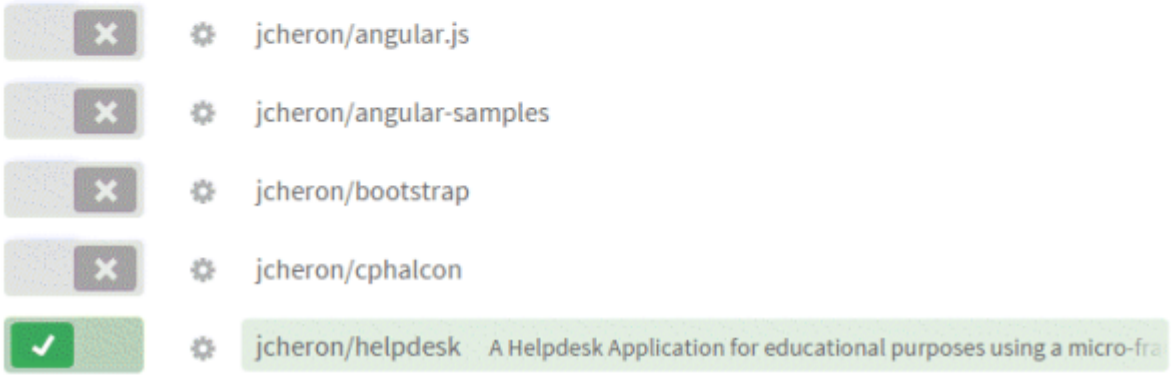
Travis-ci est un service Web open source d'intégration continue des tests sur les projets GitHub.
Une fois configuré, et associé à un projet GitHub, il permet d'exécuter automatiquement l'ensemble des tests à chaque commit Git.

Travis-ci permet de lancer les tests sur une machine virtuelle, en ayant la possibilité de choisir une ou plusieurs configurations logicielles dans le cadre desquelles les tests seront lancés.

-- Association du service Travis-ci à GitHub

- Aller à l'adresse https://travis-ci.org/ et connectez-vous avec votre compte Github.
- Choisissez ensuite **Add new repository** et sélectionner dans la liste le projet Git à tester.





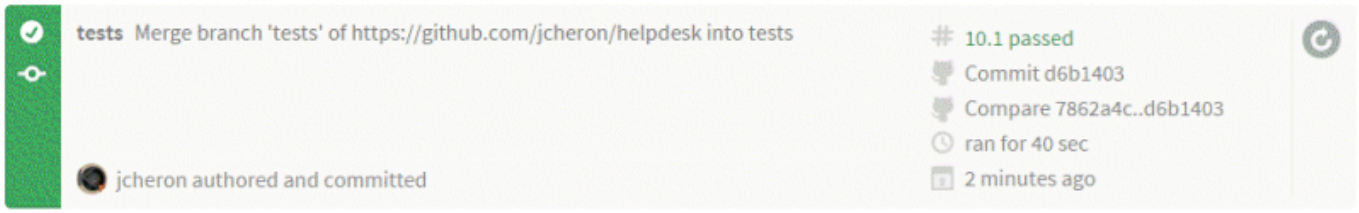
-- Configuration

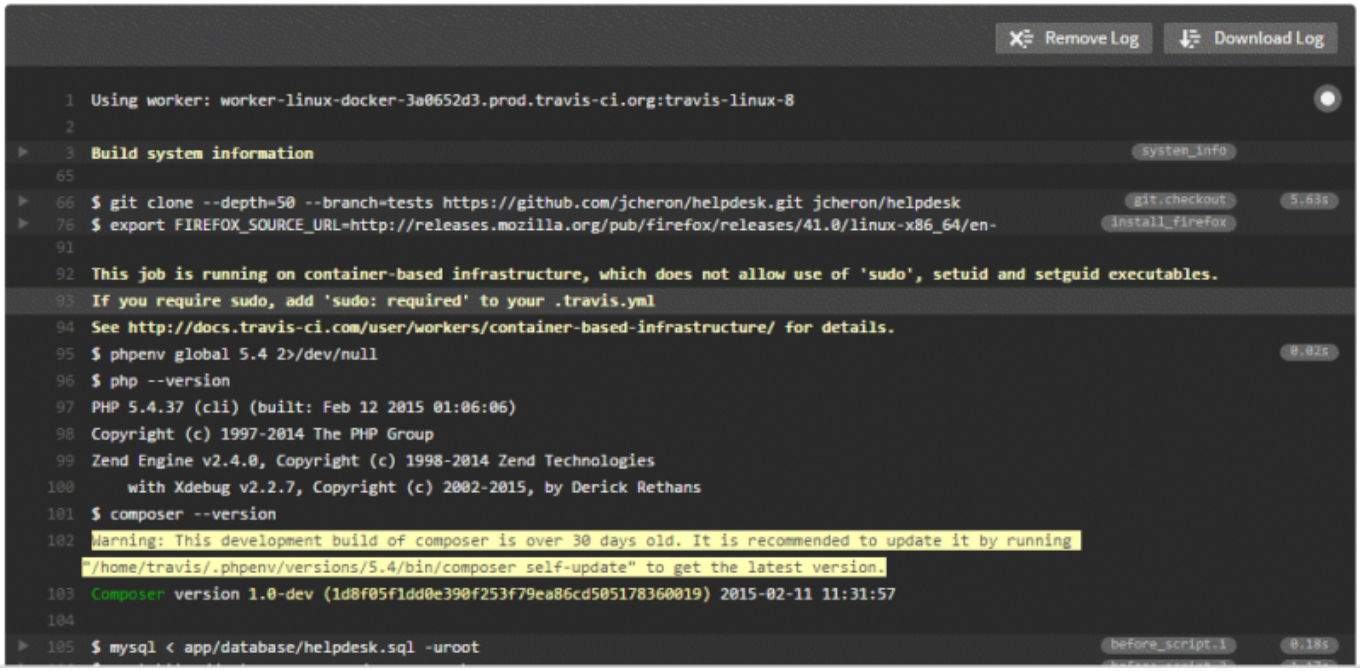
Créer le fichier **.travis.yml** à la racine du projet à tester :

```
language: php
php:
  - 5.4
```

```
  - 5.5
  - 5.6
addons:
  firefox: "41.0"
before_script:
 - mysql < app/database/helpdesk.sql -uroot
 - wget http://getcomposer.org/composer.phar
 - php composer.phar install
 - "sh -e /etc/init.d/xvfb start"
 - "export DISPLAY=:99.0"
 - "wget
http://selenium-release.storage.googleapis.com/2.45/selenium-server-standalone-2.45
.0.jar"
 - "java -jar selenium-server-standalone-2.45.0.jar > /dev/null &"
 - sleep 5
 - php -S 127.0.0.1:8090 app/.htrouter.php &
 - sleep 5
script: (cd tests; phpunit --configuration PHPunit.xml --debug)
```

Effectuer le push vers github, puis aller sur travis-ci pour observer le déroulement des tests :





Si tout est ok, la couleur devient verte, et vous avez le plaisir d'apposer le tag build unknown dans le README.md de votre projet GitHub.

From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
http://slamwiki2.kobject.net/slam5/testsfonctionnels/automatises?rev=1443654176

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