

1. Hướng dẫn cài đặt và một số điều cần lưu ý.

Rất đơn giản để cài đặt PHPUnit test. Trước tiên cần phải kiểm tra phiên bản php bạn đang sử dụng là phiên bản nào? Để thuận tiện cho việc cài đặt PHPUnit test vào project. Bên dưới là ảnh các [phiên bản PHPUnit](#) hiện tại và khả năng tương thích với phiên bản PHP.

Supported Versions

Major Version	PHP Compatibility	Initial Release	Support
PHPUnit 8	PHP 7.2, PHP 7.3, PHP 7.4	February 1, 2019	Support ends on February 5, 2021
PHPUnit 7	PHP 7.1, PHP 7.2, PHP 7.3	February 2, 2018	Support ends on February 7, 2020

Previous Versions

Major Version	PHP Compatibility	Initial Release	Support
PHPUnit 6	PHP 7.0, PHP 7.1, PHP 7.2	February 3, 2017	Support ended on February 1, 2019
PHPUnit 5	PHP 5.6, PHP 7.0, PHP 7.1	October 2, 2015	Support ended on February 2, 2018
PHPUnit 4	PHP 5.3, PHP 5.4, PHP 5.5, PHP 5.6	March 7, 2014	Support ended on February 3, 2017

H1. PHPUnit supported versions

Cài đặt PHPUnit 7 yêu cầu PHP 7.1 trở lên. Chạy câu lệnh “composer require –dev phpunit/phpunit ^7” sử dụng [Composer](#). Sau khi cài đặt thành công kiểm tra phiên bản phpunit bằng cách chạy câu lệnh “./vendor/bin/phpunit –version”.

```
→ composer require --dev phpunit/phpunit ^7

→ ./vendor/bin/phpunit --version
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.
```

H2. Install PHPUnit

Hãy chú ý đến file “./vendor/bin/phpunit”. Đây là file dùng để thực thi PHPUnit. Khi chạy lệnh “./vendor/bin/phpunit” trong project vừa cài đặt PHPUnit có tên là “w9_ajax”, sẽ hiển thị tất cả các tùy chọn help.

```
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

Usage: phpunit [options] UnitTest [UnitTest.php]
       phpunit [options] <directory>

Code Coverage Options:

  --coverage-clover <file>      Generate code coverage report in Clover XML format
  --coverage-crap4j <file>      Generate code coverage report in Crap4J XML format
  --coverage-html <dir>         Generate code coverage report in HTML format
  --coverage-php <file>         Export PHP_CodeCoverage object to file
  --coverage-text=<file>        Generate code coverage report in text format
                                Default: Standard output
  --coverage-xml <dir>         Generate code coverage report in PHPUnit XML format

  --whitelist <dir>             Whitelist <dir> for code coverage analysis
  --disable-coverage-ignore     Disable annotations for ignoring code coverage
  --no-coverage                 Ignore code coverage configuration
  --dump-xdebug-filter <file>  Generate script to set Xdebug code coverage filter

Logging Options:
```

H3. PHPUnit help

Do chúng ta sử dụng lệnh Composer để cài đặt nên cần cấu trúc lại project để nó hoạt động với file “./vendor/autoload.php”. Chúng ta cần tạo thư mục ./src, thư mục ./tests và file ./phpunit.xml trong thư mục project đang làm việc (Project đang hướng dẫn có tên là ‘w9_ajax’). Các file mã nguồn được đặt trong thư mục ./src với namespace là App và unit test sẽ được viết vào trong thư mục ./tests với namespace là Tests.

Như vậy cấu trúc hiện tại của project sẽ là:

```
w9_ajax/
├── src
├── tests
├── composer.json
├── composer.lock
└── phpunit.xml
```

Update lại file composer.json như sau:

```
{
    "require-dev": {
        "phpunit/phpunit": "^6.2"
    },
    "autoload": {
        "psr-4": {
            "App\\": "src/"
        }
    },
    "autoload-dev": {
        "psr-4": {
            "Tests\\": "tests/"
        }
    }
}
```

Sau đó chạy lệnh composer:

```
Composer dump-autoload
```

Tiếp theo chúng ta sẽ cấu hình file phpunit.xml. Ban đầu chúng ta chỉ cần một đoạn cấu hình đơn giản nhưng không kém phần quan trọng để chạy được chức năng test của chúng ta. Thêm đoạn code bên dưới vào file phpunit.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit colors="true">
    <testsuites>
        <testsuite name="Application Test Suite">
            <directory>./tests/</directory>
        </testsuite>
    </testsuites>
</phpunit>
```

Trong đoạn code cấu hình này có 2 nơi quan trọng:

- Colors="true" hiển thị màu cho kết quả test.
- < directory > ./tests/ < /directory> khai báo nơi lưu trữ file test cho PHPUnit

- **Một số quy tắc đặt tên:**

Cấu trúc và tên file: File tests được đặt trùng tên với file mã nguồn (không bắt buộc trùng tên nhưng khi làm việc đặt trùng tên để dễ quản lí file) và thêm chữ Test.php ở sau cùng(bắt buộc). Ví dụ chúng ta có các file mã nguồn:

```
./src/Foo.php  
./src/Controller/Bar.php
```

Thì các file test sẽ được tổ chức như sau:

```
./tests/FooTest.php  
./tests/Controller/BarTest.php
```

Tên tệp phải kết thúc bằng “Test.php” lưu ý chữ T phải in hoa. Đây là những file mà PHPUnit có thể nhận biết và tests trong thư mục tests. Hơn nữa các phương thức test đều bắt đầu bằng chữ “test”. Đặt tên phương thức nên mô tả rõ ràng công việc của phương thức đó. Không cần phải ngắn gọn, viết tắt. Ví dụ chúng ta có tên phương thức tests Email:

```
public function testCanBeCreatedFromValidEmailAddress()  
{  
    //Your test code  
}
```

Như vậy đã cài đặt xong và kiểm tra bằng cách tạo một file bất kì trong thư mục tests sử dụng namespace là Tests và extends class TestCase. Tạo một class firstTest với hàm testResultReturnTrue() như bên dưới.

```
<?php  
namespace Tests;  
use PHPUnit\Framework\TestCase;  
class firstTest extends TestCase  
{  
    public function testResultReturnTrue()  
    {  
        $pass = true;  
        $this->assertTrue($pass);  
    }  
}
```

Bây giờ chúng ta chạy lệnh thực thi phương thức trong class firstTest. Lúc cài đặt chúng ta đã nhắc đến file “./vendor/bin/phpunit”. Đây là file rất quan trọng nó sẽ thực thi các phương thức test. Để test tất cả các class trong thư mục tests chúng ta chạy dòng lệnh “./vendor/bin/phpunit”

```
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

.                                                                    1 / 1 (100%)

Time: 745 ms, Memory: 4.00 MB

OK (1 test, 1 assertion)
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ |
```

H4. Thực thi PHPUnit test

Chúng ta sẽ thấy màu xanh lá cây, nó biểu thị cho tất cả các testcase đã được pass. Chúng ta đã chạy 1 file test, 1 test case trong đó có 1 assertion. Đến đây là chúng ta đã hoàn thành bước cài đặt PHPUnit vào project.

2. Testcase assertion và data provider:

❖ Assertion:

Assert là một câu lệnh mục đích nhằm xác nhận một khẳng định luôn đúng tại một đoạn code. Assert định nghĩa điều mà bạn muốn nó xảy ra trên đoạn code cần test.

Ví dụ: bạn có một hàm và muốn nó trả về kết quả sẽ là true thì bạn assert return về true. Nếu Assertion trả về true thì sẽ pass unit test. Kết quả của hàm của bạn sẽ đúng ngược lại Assertion trả về fail.

Xem ví dụ: Tôi muốn kết quả cuối cùng sẽ trả về giá trị là false. Tôi sẽ thực hiện với assertFalse. Nếu kết quả là false tôi sẽ nhận được 1 pass unit test

```
public function testResultReturnFalse()
{
    $pass = false;
    $this->assertFalse($pass);
}
```

Tôi thực thi phương thức test này.

```
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

.                                                                    1 / 1 (100%)

Time: 1.9 seconds, Memory: 4.00 MB

OK (1 test, 1 assertion)
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ |
```

H5. Assertion test 1

Và tôi đã nhận được 1 pass unit test. Ngược lại nếu kết quả là true tôi sẽ nhận được lỗi.

```
public function testResultReturnFalse()
{
    $pass = true;
    $this->assertFalse($pass);
}
```

Tôi thực thi phương thức test này.

```
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

F                                                                    1 / 1 (100%)

Time: 126 ms, Memory: 4.00 MB

There was 1 failure:

1) Tests\firstTest::testResultReturnFalse
Failed asserting that true is false.

C:\wamp64\www\w9_ajax\tests\firstTest.php:11

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

H6. Assertion test 2

PHPUnit test cung cấp cho người dùng rất nhiều [assertion](#). Một số phương thức hay dùng:

- [assertArrayHasKey\(\)](#) : Kiểm tra xem key có trong chuỗi trả về hay không.
- [assertClassHasAttribute\(\)](#) : Kiểm tra xem thuộc tính có tồn tại hay không.
- [assertEquals\(\)](#) : Kiểm tra kết quả trả về và kết quả cho sẵn có bằng nhau không
- [assertContains\(\)](#) : Kiểm tra chuỗi trả về có kí tự ... hay không.
- [assertFalse\(\)](#) : Kiểm tra kết quả trả về có bằng false không.
- [assertTrue\(\)](#) : Kiểm tra kết quả trả về có bằng true hay không.
- [assertSame\(\)](#) : Kiểm tra kết quả trả về có cùng loại giá trị hay không.

Bây giờ chúng ta sẽ thử thực hành Unit test cho một hàm chuyển đổi chuỗi string sang dạng url slug.

Tạo file UrlSlug.php ở thư mục src:

```
<?php
namespace App;

class UrlSlug {
    public static function slugify($text) {
        // replace non letter or digits by -
        $text = preg_replace('~[^\pL\d]+~u', '-', $text);

        // transliterate
        $text = iconv('utf-8', 'us-ascii//TRANSLIT', $text);

        // remove unwanted characters
        $text = preg_replace('~^[^\w]+~', '', $text);
        // trim
        $text = trim($text, '-');
        // remove duplicate -
        $text = preg_replace('~--+~', '-', $text);
        // lowercase
        $text = strtolower($text);
        if (empty($text)) {
            return 'n-a';
        }
        return $text;
    }
}
```

Chúng ta sẽ sử dụng đoạn code này để thực hành với unit test đầu tiên vì nó dễ hiểu và xảy ra rất nhiều trường hợp gây ra lỗi. Chúng ta tạo thêm file tests/UrlSlugTest.php:

```
<?php
namespace Tests;
use PHPUnit\Framework\TestCase;
use App\UrlSlug;

class UrlSlugTest extends TestCase
{

}
```

Class UrlSlugTest chưa có method nào nên khi chạy phpunit sẽ nhận được thông báo warnings.

```
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

W                                                                    1 / 1 (100%)

Time: 1.95 seconds, Memory: 4.00 MB

There was 1 warning:

1) Warning
No tests found in class "Tests\UrlSlugTest".

WARNINGS!
Tests: 1, Assertions: 0, Warnings: 1.
```

H7. Warning

Chúng ta cần chạy phpunit ngay sau khi tạo file test để chắc chắn một điều rằng chúng đang hoạt động vẫn không bị lẫn lộn tên file hoặc trùng tên các class khác. Điều này sẽ giúp cho bạn tránh những trường hợp không mong đợi trong tương lai bộ test được pass tất cả nhưng PHPUnit không thực hiện test file của bạn có thể do sai sót cách đặt tên.

Bây giờ chúng ta tiến hành viết một method đơn giản cho trường hợp đầu tiên test hàm App\UrlSlug::Slugify xác nhận rằng khi truyền chuỗi string vào nó sẽ trả về chuỗi string dạng slug.

Tôi sẽ tạo một method có tên là testSlugifyReturnsSlugifyString() và chuẩn bị sẵn đầu vào là chuỗi string và đầu ra là kết quả mong muốn của chuỗi đầu vào sau khi chạy qua hàm App\UrlSlug::Slugify, sẽ được viết như sau:

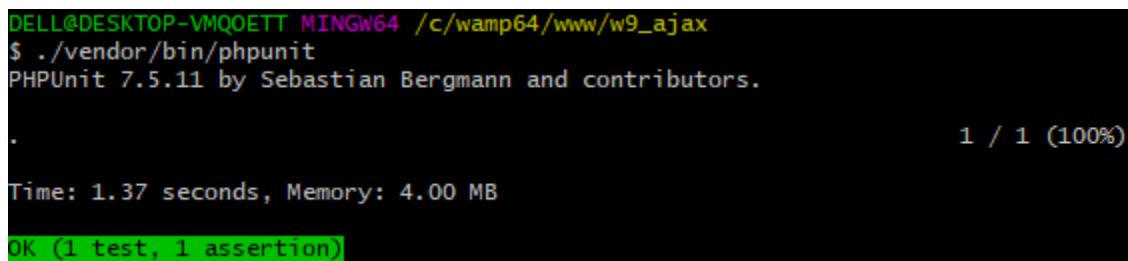
```
<?php
namespace Tests;
use PHPUnit\Framework\TestCase;
use App\UrlSlug;

class UrlSlugTest extends TestCase
{
    public function testSlugifyReturnsSlugifyString()
    {
        $input = 'test slugify returns slugify string';
        $output = 'test-slugify-returns-slugify-string';

        $url = new UrlSlug();
        $result = $url->slugify($input);

        $this->assertEquals($output, $result);
    }
}
```

Tôi sẽ dùng assertEquals để kiểm tra kết quả trả về của hàm slugify() và kết quả mong muốn có trùng khớp không. Chạy PHPUnit:



```
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

.                                                                1 / 1 (100%)

Time: 1.37 seconds, Memory: 4.00 MB

OK (1 test, 1 assertion)
```

H8. Kết quả trường hợp test đầu tiên

Chúng ta đã nhận được pass 1 test, 1 assertion. Như vậy là hàm slugify() hoạt động tốt tuyệt vời. Nhưng hãy xem lại nào, chúng ta chỉ mới thực hiện một trường hợp kí tự a-z và dấu cách. Nếu như gặp trường hợp string đầu vào có kí tự UTF8 và kí tự đặc biệt (@!#\$%^&*()\|/.) không phải là tiếng Anh thì sao nhỉ? Hoặc trường hợp rỗng thì sẽ như thế nào? Thật sự có quá nhiều trường hợp có thể xảy ra cần phải test, vì vậy chúng ta sẽ viết một bộ test gồm rất nhiều kịch bản xảy ra:

```
public function testSlugifyReturnsSlugifyString()
{
    $input = 'test slugify returns slugify string';
    $output = 'test-slugify-returns-slugify-string';

    $url = new UrlSlug();
    $result = $url->slugify($input);

    $this->assertEquals($output, $result);
}
```

```
public function testSlugifyNumberReturnsSlugifyString()
{
    $input = ' 333 test 11 slugify11 returns222 slugify string';
    $output = '333-test-11-slugify11-returns222-slugify-string';

    $url = new UrlSlug();
    $result = $url->slugify($input);

    $this->assertEquals($output, $result);
}
```

```
public function testSpecialCharactersReturnsSlugifyString()
{
    $input = 'test @@@\Special !@#!@#() \\\//Characters### @!Returns
&&Slugify #%String';
    $output = 'test-special-characters-returns-slugify-string';

    $url = new UrlSlug();
    $result = $url->slugify($input);

    $this->assertEquals($output, $result);
}
```

```

public function testUTF8NoEnglishCharacterReturnsSlugifyString()
{
    $input = 'tést UTF8 Nö English Chäräcter Returns Slugify String';
    $output = 'test-utf8-no-english-character-returns-slugify-string';

    $url = new UrlSlug();
    $result = $url->slugify($input);
    $this->assertEquals($output, $result);
}

```

```

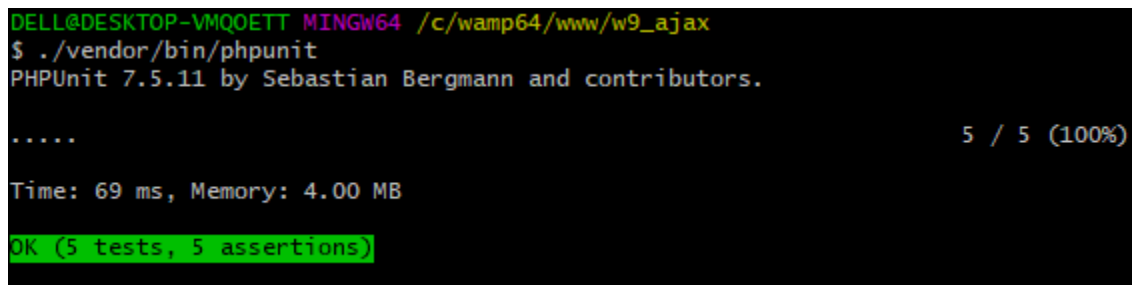
public function testEmptyStringsReturnsSlugifyString()
{
    $input = '';
    $output = 'n-a';

    $url = new UrlSlug();
    $result = $url->slugify($input);

    $this->assertEquals($output, $result);
}

```

Bên trên là 5 method 5 trường hợp khác nhau tôi đã chuẩn bị để test hàm slugify(). Có thể sẽ có thêm trường hợp khác nữa. Tìm càng nhiều trường hợp xảy ra sẽ giúp cho code hoạt động tốt hơn. Bây giờ tôi sẽ chạy PHPUnit:



```

DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

..... 5 / 5 (100%)

Time: 69 ms, Memory: 4.00 MB

OK (5 tests, 5 assertions)

```

H9. Kết quả test hàm slugify

Chúng ta đã nhận được pass 5 tests, 5 assertions. Hàm slugify() hoạt động khá tốt. Nhưng bạn hãy xem lại 5 method vừa tests, chúng ta dễ dàng nhận thấy đang có vấn đề trùng lặp code. Để khắc phục được tình trạng trùng lặp, PHPUnit đã hỗ trợ công cụ cho chúng ta khắc phục đó là “@dataProvider”. Nhưng trước khi tìm hiểu về “@dataProvider” chúng ta tìm hiểu Annotations là gì.

❖ Annotations:

Annotation là dạng cú pháp đặc biệt được khai báo trong [docblocks](#) của method.

Chú ý: một comment docblocks bao gồm bắt đầu là `/**` và kết thúc là `*/`.

Annotation nằm trong các kiểu khác sẽ bị bỏ qua.

```
/**
 * @dataProvider
 */
public function tearDownSomeFixtures()
{
    // ...
}
```

PHPUnit cho chúng rất nhiều [annotation](#), nhưng trước hết chúng ta sẽ đề cập đến `@dataProvider`.

❖ [@dataProvider](#)

`dataProvider` là một method có thể chấp nhận các đối số tùy ý. Các đối số này được cung cấp bởi 1 hoặc nhiều method data provider. Thay vì tạo nhiều phương thức test trùng lặp, chúng ta chỉ cần tạo duy nhất một method dữ liệu để test gồm các tham số tương ứng với dữ liệu biến đổi giữa các phép thử.

Ví dụ:

```
/**
 * @dataProvider additionProvider
 */
public function testAdd($a, $b, $expected)
{
    $this->assertSame($expected, $a + $b);
}

public function additionProvider()
{
    return [
        [0, 0, 0],
        [0, 1, 1],
    ];
}
```

Như ví dụ trên ta có một method `testAdd($a, $b, $expected)` gồm 3 biến. Bên trong hàm có phương thức `assertSame` so sánh kết quả giữa `$expected` và `$a + $b`.

Một method data provider sẽ trả về một mảng dữ liệu chứa các tập input. Theo ví dụ ở trên thì ta có 2 tập input. Mỗi tập input là một mảng giá trị gồm 3 phần tử.

Nó được truyền vào method `testAdd($a, $b, $expected)` với 2 tập input `[0,0,0]` và `[0,1,1]`. 3 phần tử này tương ứng với vị trí là `$a`, `$b`, `$expected`. Thử tính trước kết quả: $0 + 0 = 0$ và $0 + 1 = 1$. Như vậy chúng ta sẽ nhận được 1 pass tests gồm 2 tests, 2 assertions. Thử chạy PHPUnit.

```
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

..                                                                  2 / 2 (100%)

Time: 1.86 seconds, Memory: 4.00 MB

OK (2 tests, 2 assertions)
```

H10. data provider test

Đúng như kết quả dự đoán chúng ta đã nhận được pass tests. Chỉ cần 1 method `testAdd($a, $b, $expected)` nhưng bạn đã tests 2 tests. Chúng ta đã bỏ qua được sự trùng lặp code. Bây giờ thử áp dụng vào bài tập trước làm trên hàm `App\UrlSlug::slugify`. Đổi sang dataProvider ta sẽ được như sau:

```

/**
 * @param string $input
 * @param string $output
 * @dataProvider providerTestSlugifyReturnsSlugifyString
 */
public function testSlugifyReturnsSlugifyString($input, $output)
{
    $url = new UrlSlug();
    $result = $url->slugify($input);

    $this->assertEquals($output, $result);
}
public function providerTestSlugifyReturnsSlugifyString()
{
    return [
        ['test slugify returns slugify string', 'test-slugify-returns-slugify-string'],

        [' 333 test 11 slugify11 returns222 slugify string', '333-test-11-slugify11-returns222-slugify-string'],

        ['test @@@\\Special !@#!@#() \\//Characters### @!Returns &&Slugify #String', 'test-special-characters-returns-slugify-string'],

        ['tést UTF8 Nö English Chäräcter Returns Slugify String', 'test-utf8-no-english-character-returns-slugify-string'],

        ['', 'n-a']
    ];
}

```

Lần tests trước chúng ta cần đến 5 method hoặc có thể nhiều hơn nữa để tests nhưng bây giờ chỉ cần 2 method. Method thứ nhất là `testSlugifyReturnsSlugifyString($input, $output)` và method thứ hai `providerTestSlugifyReturnsSlugifyString` là `dataProvider` chứa tất cả các tập cần tests. Chạy thử PHPUnit:

```

DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

.....                                     5 / 5 (100%)

Time: 1.69 seconds, Memory: 4.00 MB

OK (5 tests, 5 assertions)

```

H11. UrlSlug - Provider

3. Test các phương thức Private/Protected:

Phần trước chúng ta đã test các phương thức ở dạng public. Theo thông thường thì tạo new object và từ object đó trở đến các phương thức ở dạng public để test. Nhưng đối với các phương thức ở dạng private/protected bạn sẽ không thể gọi được trực tiếp đến các phương thức đó thông qua object đã tạo.

Sử dụng lại class UrlSlug.php của phần trước. Chúng ta viết thêm 2 method như sau:

```
private function funcStringPrivate($string)
{
    return $string;
}

protected function funcStringProtected($string)
{
    return $string;
}
```

Có vẻ 2 method này rất đơn giản chỉ đưa vào 1 biến string và trả về biến string đó. Nhưng nó sẽ giúp chúng ta có 1 ví dụ đơn giản và dễ hiểu về việc gọi trực tiếp tới method private/protected.

Chúng ta sẽ thử tạo một method phpunit tests và tests hai method vừa tạo theo cách thông thường thử nhé.

```
public function testMethodPrivateReturnsString()
{
    $string = 'private';
    $url = new UrlSlug();
    $result = $url->funcStringPrivate($string);
    $this->assertEquals($string, $result);
}
```

Chạy PHPUnit:

```
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

E                                                                    1 / 1 (100%)

Time: 58 ms, Memory: 4.00 MB

There was 1 error:

1) Tests\UrlSlugTest::testMethodPrivateReturnsString
Error: Call to private method App\UrlSlug::funcStringPrivate() from context 'Tests\UrlSlugTest'

C:\wamp64\www\w9_ajax\tests\UrlSlugTest.php:22

ERRORS!
Tests: 1, Assertions: 0, Errors: 1.
```

H12. tests method private

Chúng ta đã nhận được thông báo lỗi. Không thể gọi trực tiếp đến các phương thức private/protected.

Làm thế nào để trực tiếp test các phương thức private/protected mà không phải dùng gián tiếp thông qua phương thức public? Điều này khá dễ dàng bằng cách sử dụng [Reflectionclass](#) trong PHP. Tạo method invokeMethod() như sau:

```
/**
 * Call protected/private method of a class.
 *
 * @param object &$object    Instantiated object that we will run method
on.
 * @param string $methodName Method name to call
 * @param array $parameters Array of parameters to pass into method.
 *
 * @return mixed Method return.
 */
public function invokeMethod(&$object, $methodName, array $parameters =
array())
{
    $reflection = new \ReflectionClass(get_class($object));
    $method = $reflection->getMethod($methodName);
    $method->setAccessible(true);

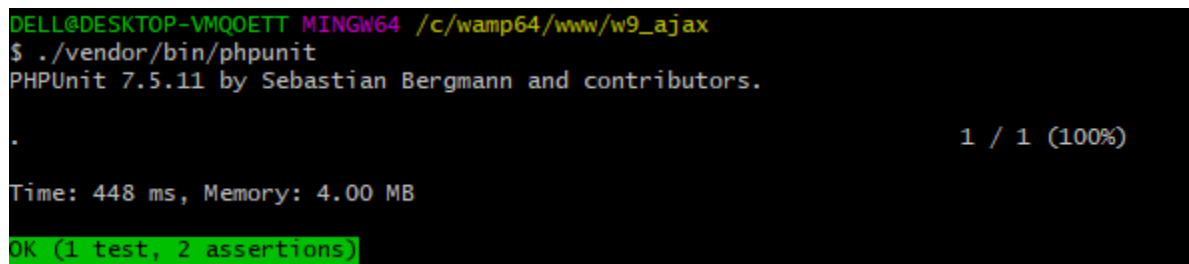
    return $method->invokeArgs($object, $parameters);
}
```


Sử dụng method `invokeMethod()` chúng ta có thể dễ dàng gọi trực tiếp các method Private/Protected. Chúng ta sẽ có method test:

```
public function testInvokeMethodStringPrivateReturnString()
{
    $string = 'private';
    $url = new UrlSlug();
    $result1 = $this->invokeMethod($url, 'funcStringPrivate', [$string]);
    $result2 = $this->invokeMethod($url, 'funcStringProtected',
[$string]);

    $this->assertEquals($string, $result1);
    $this->assertEquals($string, $result2);
}
```

Chạy PHPUnit:



```
DELL@DESKTOP-VMQOETT MINGW64 /c/wamp64/www/w9_ajax
$ ./vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

.                                                                    1 / 1 (100%)

Time: 448 ms, Memory: 4.00 MB

OK (1 test, 2 assertions)
```

H13. invokeMethod test

Nhận được pass tests. Như vậy thông qua method `invokeMethod()` chúng ta có thể gọi trực tiếp đến các method ở dạng private/protected để tests.

4. Coverage Report và CRAP

❖ Coverage Report:

Nếu gặp một khối lượng lớn method để tests chúng ta sẽ gặp tình trạng bỏ sót. Có một công cụ rất mạnh đi kèm với PHPUnit. Công cụ Coverage report tạo các tệp HTML tĩnh giúp bạn xem số liệu thống kê, bao gồm bao nhiêu phần được kiểm tra và mức độ phức tạp code của bạn.

Nó còn cho bạn biết được bạn đã bỏ lỡ bất kỳ dòng code nào. Chẳng hạn như nó không kích hoạt một câu lệnh if hoặc try, catch.

Để tạo được báo cáo code Coverage yêu cầu [Xdebug](https://xdebug.org/wizard.php). Để cài đặt xdebug bạn truy cập vào trang web: <https://xdebug.org/wizard.php>.

Chạy command line câu lệnh: **php -i**. Php info sẽ hiện ra và bạn copy từ dòng phpinfo() đến hết (toàn bộ đầu ra của phpinfo). Sau đó bạn paste vào ô textarea trên website như hình bên dưới và nhấn nút “Analyse my phpinfo() output”.

```
phpinfo()
PHP Version => 7.2.18

System => Windows NT DESKTOP-VMQOETT 10.0 build 17134 (Windows 10) AMD64
Build Date => Apr 30 2019 23:25:56
Compiler => MSVC15 (Visual C++ 2017)
Architecture => x64
Configure Command => cscript /nologo configure.js "--enable-snapshot-build"
"--enable-debug-pack" "--with-pdo-oci=c:\php-snap-
build\deps_aux\oracle\x64\instant
client_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-
build\deps_aux\oracle\x64\i
nstantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-
com-d
o
tnet=shared" "--without-analyzer" "--with-pgo"
Server API => Command Line Interface
Virtual Directory Support => enabled
Configuration File (php.ini) Path => C:\Windows
Loaded Configuration File => C:\wamp64\bin\php\php7.2.18\php.ini
Scan this dir for additional .ini files => (none)
Additional .ini files parsed => (none)
PHP API => 20170718
PHP Extension => 20170718
Zend Extension => 320170718
```

1

The information that you upload will not be stored. The script will only use a few regular expressions to analyse the output and provide you with instructions. You can see the code [here](#).

Analyse my phpinfo() output

2

Tiếp theo bạn cần thực hiện 3 bước trên website hướng dẫn để hoàn tất cài đặt.

1. Download [php_xdebug-2.7.2-7.2-vc15-x86_64.dll](#)
2. Move the downloaded file to `c:\wamp64\bin\php\php7.2.18\ext`
3. Update `c:\wamp64\bin\php\php7.2.18\php.ini` and change the line
`zend_extension = c:\wamp64\bin\php\php7.2.18\ext\php_xdebug-2.7.2-7.2-vc15-x86_64.dll`

H15. install xdebug 2

Quá trình cài đặt xdebug chỉ xảy ra 1 lần. Sau khi bạn cài đặt xdebug bạn có thể sử dụng trên tất cả project.

Tạo coverage report:

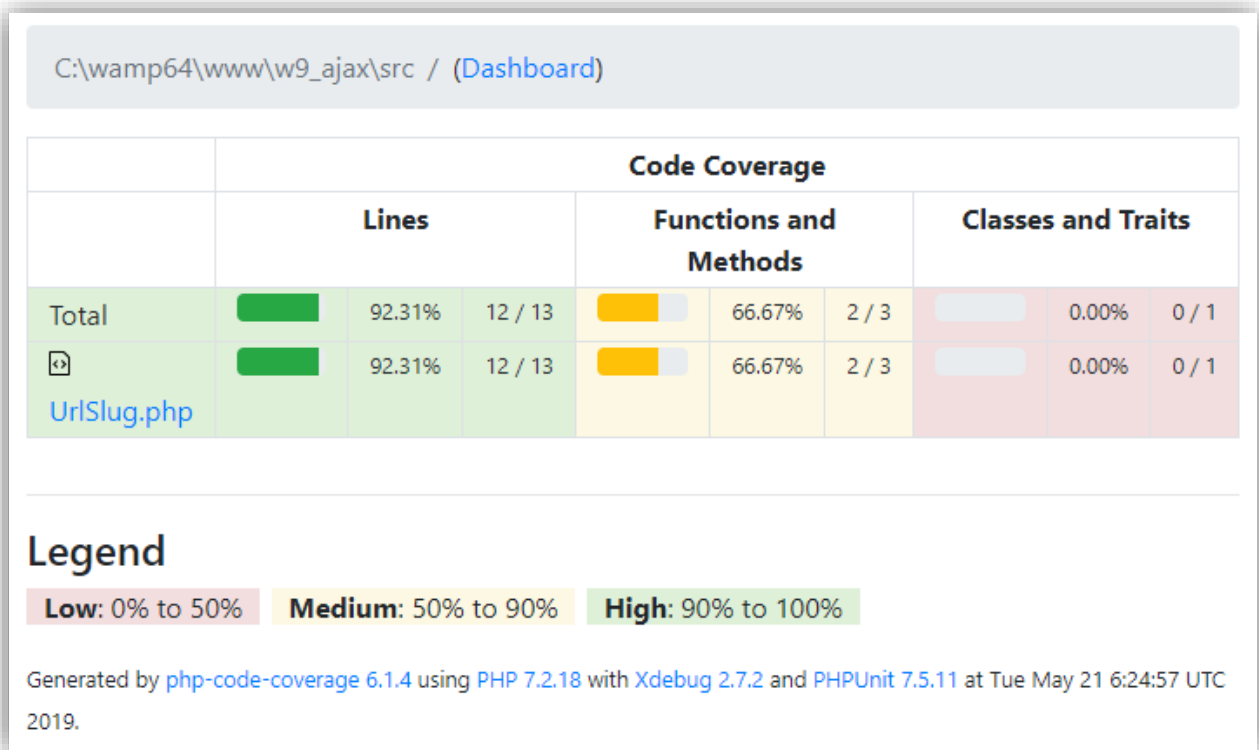
Đầu tiên chúng ta cần update lại file **phpunit.xml**, thêm thẻ **filter**

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit colors="true">
  <testsuites>
    <testsuite name="Application Test Suite">
      <directory>./tests/</directory>
    </testsuite>
  </testsuites>
  <filter>
    <whitelist processUncoveredFilesFromWhitelist="true">
      <directory suffix=".php">./src</directory>
    </whitelist>
  </filter>
</phpunit>
```

Sau đó chạy lệnh phpunit với tham số **--coverage-html <tên thư mục>** để tạo báo cáo:

```
./vendor/bin/phpunit --coverage-html coverage
```













Bạn nhìn vào project sẽ thấy có thư mục coverage thư mục đã được tạo tự động và chứa các tệp HTML. Mở tệp index.html



H16. Page index - coverage report

Chúng ta đang sử dụng 1 class UrlSlug.php để test nên nó chỉ báo cáo cho chúng ta class này. Nếu chúng ta test trên nhiều class khác nhau nó sẽ hiện ra tất cả các class mà chúng ta tests. Bây giờ nhấn vào UrlSlug.php bạn sẽ thấy:

C:\wamp64\www\w9_ajax\src / UrlSlug.php

	Code Coverage									
	Classes and Traits			Functions and Methods				Lines		
Total		0.00%	0 / 1		66.67%	2 / 3	CRAP		92.31%	12 / 13
UrlSlug		0.00%	0 / 1		66.67%	2 / 3	5.01		92.31%	12 / 13
slugify					100.00%	1 / 1	2		100.00%	9 / 9
funcStringPrivate					0.00%	0 / 1	2.15		66.67%	2 / 3
funcStringProtected					100.00%	1 / 1	1		100.00%	1 / 1

```
1  <?php
2  namespace App;
3
4  class UrlSlug
5  {
6      public static function slugify($text)
7      {
8          // replace non letter or digits by -
9          $text = preg_replace('~^[^pL\d]+~u', '-', $text);
10         // transliterate
11         $text = iconv('utf-8', 'us-ascii//TRANSLIT', $text);
12
13
14
15         // remove unwanted characters
16         $text = preg_replace('~^[^-\w]+~', '', $text);
17         // trim
18         $text = trim($text, '-');
```

H17. page class - coverage report

Màu của dòng code: dòng màu xanh là mã đã được phpunit tests thực thi, dòng màu đỏ là mã không được thực thi và màu vàng là vùng mã chết.

Khi hover vào dòng màu xanh, popup sẽ hiện ra thông tin testunit nào đã cover dòng code đó.

```

7 {
8     // replace non letter or digits by -
9     $text = preg_replace('~[^\pL\d]+~u', '-', $text);
10    // transliterate
11    $text = iconv('utf-8', 'us-ascii//TRANSLIT', $text);
12
13
14
15    // remove unwanted characters
16    $text = preg_replace('~[^\w]~u', '', $text);
17    // trim
18    $text = trim($text, '-');
19    // remove duplicate -
20    $text = preg_replace('~--+~', '-', $text);
21    // lowercase
22    $text = strtolower($text);
23    if (empty($text)) {
24        return 'n-a';

```

5 tests cover line 9

- Tests\UrlSlugTest::testSlugifyReturnsSlugifyString with data set #0
- Tests\UrlSlugTest::testSlugifyReturnsSlugifyString with data set #1
- Tests\UrlSlugTest::testSlugifyReturnsSlugifyString with data set #2
- Tests\UrlSlugTest::testSlugifyReturnsSlugifyString with data set #3
- Tests\UrlSlugTest::testSlugifyReturnsSlugifyString with data set #4

H18. hover green code

Như đã nói lúc này, nó sẽ cho bạn thấy được những dòng code mà bạn đã bỏ qua cụ thể hãy xem câu lệnh if ở tám hình bên dưới:

```

29     private function funcStringPrivate($string)
30     {
31         if($string == 'abc')
32         {
33             return 'xyz';
34         }
35         return $string;
36     }

```

H19. red code

Chưa có testcase làm thỏa mãn điều kiện biến string truyền vào là 'abc', do đó dòng code bên trong câu lệnh if sẽ bị bỏ qua không được thực thi.

Chúng ta cần viết thêm 1 testcase nữa làm thỏa mãn điều kiện câu lệnh if này và thực thi nó. Chúng ta có thêm 1 testcase như sau:

```

public function testFuncStringPrivateReturnsXYZWhenStringIsABC()
{
    $string = 'abc';
    $url = new UrlSlug();
    $result = $this->invokeMethod($url, 'funcStringPrivate', [$string]);
    $this->assertEquals('xyz', $result);
}

```

Chúng ta chạy lại lệnh coverage report và reload trang xem lại kết quả bạn sẽ thấy dòng màu đỏ đã chuyển sang màu xanh:

```
29     private function funcStringPrivate($string)
30     {
31         if($string == 'abc')
32         {
33             return 'xyz';
34         }
35         return $string;
36     }
```

H20. green if test

❖ CRAP

Khi nhìn vào bảng coverage report bạn sẽ thấy có một cột **CRAP** (viết tắt bởi Change Risk Analysis and Predictions). Nói một cách dễ hiểu nó là thể hiện độ phức tạp của method. Chẳng hạn bạn có một method getter thông thường thì CRAP sẽ gần bằng 1 (giá trị nhỏ nhất của CRAP). Nếu code trở nên phức tạp hơn thêm vài đoạn if, foreach vào thì CRAP sẽ bắt đầu tăng lên đáng kể.

Tài liệu Tham khảo: <https://jtreminio.com/blog/unit-testing-tutorial-part-i-introduction-to-phpunit/>

Tài liệu hướng dẫn: <https://github.com/taisaoem1368/phpunit-tutorial>

