TDD W PHP: TESTY JEDNOSTKOWE Z PHPUNIT – KROK PO KROKU

Arkadiusz Kondas | 7 listopada 2014 | PHP, TDD | 0 Comments

Konkretny wpis na temat wykorzystania bardzo popularnego narzędzia, jakim jest PHPUnit, do tworzenia testów jednostkowych. Od instalacji, przez konfigurację do testowania kodu.

PHPUnit to framework testów jednostkowych, którego autorem jest Sebastian Bergmann. Jak inne tego typu frameworki używa on do testów tzw. asercji, które sprawdzają jak zachowuje się kod poddany testom (można spotkać się również z skrótem SUT – system under test). O idei samych testów jednostkowych możesz przeczytać w Wprowadzeniu do TDD w PHP. PHPUnit jest w pełni obiektowym narzędziem, które czerpie swoje rozwiązania z frameworków typu xUnit.

Trzeba jeszcze rozjaśnić, czym jest ta cała asercja ? Najpierw pozwolę sobie zacytować Wikipedię:

---

*W programowaniu **asercja** (ang.assertion) to predykat (forma zdaniowa w danym języku, która zwraca prawdę lub fałsz), umieszczony w pewnym miejscu w kodzie. Asercja wskazuje, że programista zakłada, że predykat ów jest w danym miejscu prawdziwy.*

Przyznam, że zawile to opisali. Teraz tak bardziej po mojemu: asercja to zwyczajne sprawdzenie czy to co otrzymał dany fragment kodu jest tym czego się spodziewamy. Poniżej krótki fragment (wyjęty z kontektu, omówię go niżej):

```
1 $text = 'Hello World';
2 $this->assertTrue($text === 'Hello World');
```

Zapraszam na nowy blog osobisty dostępny pod adresem: https://arkadiuszkondas.com

WYSZUKIWARKA

Search the site

ZAPISZ SIĘ DO NEWSLETTERA

Informacje o nowych wpis oraz blogu prosto na Twój adres e-mail, bez zbędnego spamu.

Imię *

Email *

Zapisz się!

Programista rzemi
1,763 likes

Like Page Se

Popular Recent Comments

...
17 czerwca, 2015 • 29 komentarzy

W tym przypadku sprawdzamy czy wyrażenie porównania zwraca prawdę. Po napisaniu pierwszego testu sprawa na pewno się rozjaśni :).

> Cały kod źródłowy z tego wpisu dostępny jest publicznie pod adresem: https://github.com/itcraftsmanpl/PHPUnitTests

Ok, po krótkim wstępie możemy przejść do instalacji.

INSTALACJA PHPUNIT

Na potrzeby tego wpisu zainstalujemy PHPUnit globalnie, korzystając z Composera. Cała operacja może wydać się nieco skomplikowana, ale zrobimy ją tylko raz. Potem, do każdego projektu, będziesz mógł używać PHPUnit od ręki.

Uruchamiamy terminal (lub wiersz poleceń) i wpisujemy:

```
composer global require "phpunit/phpunit=4.3.*"
```

Jeżeli wszystko przebiegło prawidłowo, to potrzebujemy dodać odpowiednią ścieżkę do zmiennej systemowej o nazwie PATH. W ten sposób będziemy mogli wywoływać polecenie „*phpunit*" w dowolnym miejscu. W zależności od systemu musimy wykonać:

Linux/Mac:

```
export PATH=$PATH:~/.composer/vendor/bin/
```

Windows

Tutaj będzie trudniej, musimy zmienną Path uaktualnić o poniższą wartość (podmieniamy username) [upewnij się czy dana ścieżka jest prawidłowa, w zależności od wersji systemu może się różnić, zauważyłem również, że jest ona wyświetlana w czasie wykonywania polecenie *composer global require*]:

```
C:\Users\<username>\AppData\Roaming\Composer\vendor
```

1. Klikamy prawym na „Mój komputer"
2. Wybieramy „Zaawansowane ustawienia systemu"
3. Z karty „Zaawansowane" wybieramy „Zmienne środowiskowe"
4. Z dolnej listy (zmienne systemowe) wybieramy Path i klikamy „Edytuj"

...
7 listopada, 2014 • 12 komentarzy

...
23 września, 2014 • 2 komentarze

...
24 czerwca, 2014 • 28 komentarzy

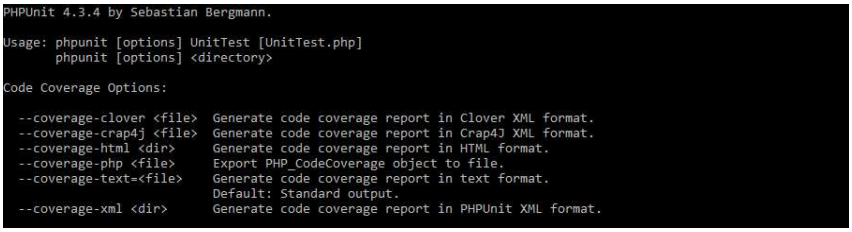
...
9 sierpnia, 2014 • 13 komentarzy

Next »

KATEGORIE

5. Na końcu dopisujemy średnik (oddziela nową wartość) a następnie wklejamy wyżej przedstawioną ścieżkę

Aby przetestować, czy wszystko jest prawidłowo zainstalowane, restartujemy wiersz poleceń (aby odświeżyła się zmienna Path) i wpisujemy samo *phpunit*. W efekcie powinno wyświetlić się coś w stylu:



Wywołanie polecenia phpunit w wierszu poleceń.

Instalacja lokalna

Warto wspomnieć, że istnieje również możliwość instalacji PHPUnit bezpośrednio do wybranego projektu (lokalnie). Tutaj również korzystamy z Composera, edytując plik *composer.json*:

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

lub bezpośrednio w konsoli, w katalogu głównym projektu (tam gdzie znajduje się plik *composer.json*), wywołujemy polecenie:

```
composer require "phpunit/phpunit=4.3.*"
```

W takim przypadku, PHPUnit trzeba będzie wywoływać poprzez wpisanie następującego polecenia:

```
vendor/bin/phpunit
```

Kolory

Jeżeli korzystasz z systemu Windows i chcesz aby komunikaty w wierszu poleceń były kolorowe (a na pewno chcesz) to polecam instalację ANSICON (wystarczy pobrać, rozpakować i odpalić exeka).

KONFIGURACJA PHPUNIT

- Powershell
- Programowanie
- Programowanie obiektowe
- Refaktoryzacja
- SOLID
- SQL
- TDD
- Testy automatyczne
- Testy funkcjonalne
- Tips & Tricks
- Web services
- Windows Forms
- Wzorce
- Wzorce projektowe

ARCHIWA

- luty 2018
- październik 2017
- maj 2017
- kwiecień 2017
- marzec 2017
- grudzień 2016
- listopad 2016
- październik 2016
- lipiec 2016
- czerwiec 2016
- maj 2016
- kwiecień 2016
- marzec 2016
- luty 2016
- październik 2015
- wrzesień 2015
- czerwiec 2015
- maj 2015
- marzec 2015
- luty 2015
- styczeń 2015
- grudzień 2014
- listopad 2014

Przed przystąpienie do pisania pierwszego testu, skonfigurujemy PHPUnit tak, aby był jak najbardziej wygodny w działaniu. W katalogu główny projektu należy utworzyć plik *phpunit.xml* o następującej zawartości:

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit
        backupGlobals="false"
        backupStaticAttributes="false"
        bootstrap="vendor/autoload.php"
        colors="true"
        convertErrorsToExceptions="true"
        convertNoticesToExceptions="true"
        convertWarningsToExceptions="true"
        processIsolation="false"
        stopOnFailure="false"
        syntaxCheck="false"
        >
    <testsuites>
        <testsuite name="Application Test Suite">
            <directory>./tests/</directory>
        </testsuite>
    </testsuites>
</phpunit>
```

W ten sposób uruchamiając PHPUnita będzie on zaczytywał domyślną konfigurację. Tym samym do przeprowadzenie testów kodu wystarczy wywołać polecenie *phpunit*. Poniżej krótkie omówienie najważniejszych parametrów:

boostrap – tutaj podajemy ścieżką do pliku który zostanie wczytany za każdym razym, gdy uruchomi się proces testowania; najlepszym rowiązaniem jest dodanie autoloadera który dostarcza nam Composer (domyślnie *vendor/autoload.php*)

colors – ustawiając na true uzyskamy w konsoli kolorowe komunikaty

stopOnFailure – ustawiamy na false – dzięki temu pierwszy błąd nie zatrzyma testowania, a testy będą wykonywane do końca

directory – w tym tagu ustawiamy ścieżkę w której mieścić się będą nasze testy, PHPUnit zaciągnie je w ten sposób automatycznie

W razie wątpliwości z resztą opcji, polecam dokumentację.

PIERWSZY TEST

Ok, nareszcie możemy przejść do napisania pierwszego testu. W głównym katalogu tworzymy nowy folder „tests" (może być dowolnie inny, pamiętaj tylko żeby zaktualizować odpowiednio tag *directory* w pliku *phpunit.xml*). Nasz pierwszy test będzie do bólu klasyczny. Tworzymy nowy plik „*ExampleTest.php*" z następującą zawartością:

- październik 2014
- wrzesień 2014
- sierpień 2014
- lipiec 2014
- czerwiec 2014

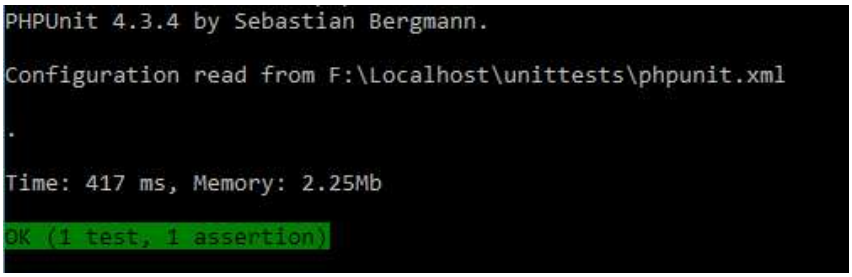
TAGI

.NET ASP .NET Automation automatyzacja Azure baza danych Bundles C# call for papers Cloud composer dajsiepoznac dajsiepoznac2016 dokumentacja future processing git Helpers jakość javascript json konferencja laravel machine learning markdown Microsoft MVC mysql oprogramowanie performance php php-ml php 5.6 php 7 phpunit phpunit.xml podsumowanie quality excites Selenium system kontroli wersji systemy uczące się tdd tdd w php Tests testy jednostkowe unittest

```
1  class ExampleTest extends PHPUnit_Framework_TestCas
2
3      public function testGreetings()
4      {
5          $greetings = 'Hello World';
6          $this->assertEquals('Hello World', $greetin
7      }
8
9  }
```

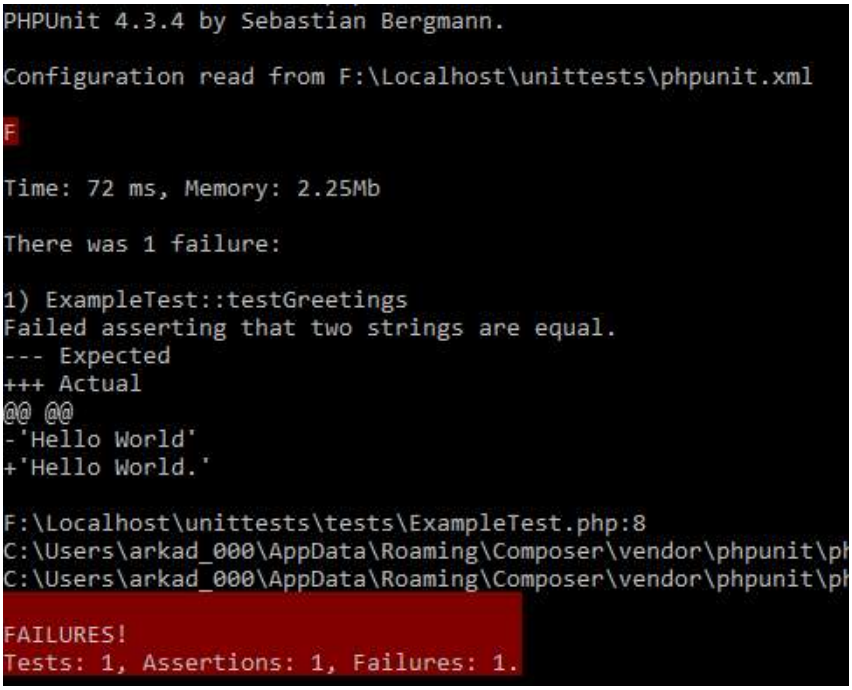
Objaśnimy teraz jego strukturę. **Nazwa pliku** oraz **nazwa klasy** pokrywa się, jednocześnie nazwa powinna wskazywać testowany obiekt i kończyć się słowem Test (np. UserTest lub RegisterEventTest). Każda testowana klasa powinna dziedziczyć po klasie ***PHPUnit_Framework_TestCase.*** *Z*apewnia ona dostęp do wszystkich możliwych asercji, oraz paru innych przydatnych metod (m. in. *setUp* oraz *tearDown*). Każda metoda powinna testować jeden element (funkcjonalność) i być poprzedzona słowem ***test*** (np. *testInstance* lub *testAddsNumbers*)

Sprawdzimy teraz czy nasz test przechodzi (jest zielony). Wpisz w konsoli *phpunit* i sprawdź czy masz przed oczami coś podobnego do:



Zawartość konsoli po przeprowadzeniu pierwszego testu.

Na pewno zielona linia „OK (1 test, 1assertion)” wpada w oko. Oznacza ona, że wszystkie testy przeszły (w ilości 1) oraz łącznie była jedna asercja. Nad linią z czasem i zabraną pamięcią, znajduje się pojedyncza kropka i nie jest przypadkowa. Reprezentuje ona pojedynczy test. Gdy dodamy kolejny, zobaczymy kolejnę kropkę (o ile się powiedzie). W przypadku gdy dany test zawiedzie, zamiast kropki wyświetli się litera F (od *failure*). Przykład poniżej:



Zawartość konsoli po nieudanym teście.

WYBRANE RODZAJE ASERCJI

Przedstawię teraz kilka najpopularniejszych asercji dostępnych w PHPUnit. W momencie w którym tworzyłem ten wpis, całkowita liczba asercji to 93. Na szczęście ich nazwy są na tyle logiczne, że korzystając z dowolnego IDE z funkcją podpowiadania, każdy odnajdzie to czego potrzebuje.

assertTrue

```
1  $value = 10.99;
2  $this->assertTrue(is_numeric($value), 'Opcjonalna w
```

Najprostszy rodzaj, w tym przypadku zakładamy, że oczekiwana wartość to logiczne *true.* Większość asercji posiada swój odwrotny odpowiednik. Czyli, jeżeli chcemy założyć, że oczekiwana wartość to *false*, to możemy skorzystać z metody ***assertFalse.*** Ostatnim parametrem (w każdej asercji) jest opcjonalna wiadomość, która wyświetli się w przypadku niepowodzenia.

assertEquals

```
1  $expected = '10';
2  $this->assertEquals($expected, 5+5);
```

Tym razem podajemy dwa parametry (tak jest w większej części asercji), wartość oczekiwana jako pierwszy, oraz wartość faktyczna

(np. zwrócona) jako drugi. W przypadku gdy typy muszę się zgadzać, można wykorzystać ***assertSame***

assertContains

```
1 $fruits = ['Apple', 'Orange', 'Grapefruit'];
2 $this->assertContains('Apple', $fruits);
```

Tutaj sprawdzamy czy pierwszy parametr istnieje jako element tablicy (drugi parametr). Jak widać nazewnictwo w pełni przedstawia to czego dotyczą same test i założenie. Automatycznie jeżeli chcemy poszukać klucza w tablicy to korzystamy z ***assertArrayHasKey***.

assertInternalType

```
1 $fruits = ['Apple', 'Orange', 'Grapefruit'];
2 $this->assertInternalType('array', $fruits);
```

Sprawdzamy czy podany parametr (jako drugi) jest konkretnego typu. Typ musi być wyrażony jako string (rodzaje typów).

assertInstanceOf

```
1 $date = new DateTime();
2 $this->assertInstanceOf('DateTime', $date);
```

Sprawdza czy podana zmienna jest obiektem danej klasy.

expectedException

Zdarzają się przypadki, w których będziemy potrzebować upewnić się, że dany test zakończy się wyrzuceniem wyjątku (np. sprawdzamy czy metoda wyrzuca wyjątek w przypadku podania, jako parametru, minusowej wartości). W celu sprawdzenia czy dany test zwrócił wyjątek należy skorzystać z tzw. adnotacji. Adnotacje wstawia się w komentarzu nad metodą:

```
1 /**
2  * @expectedException Exception
3  */
4 public function testMustThrowException()
5 {
6     throw new Exception("Error", 1);
7 }
```

Po spacji wpisujemy nazwę spodziewanego wyjątku (np. *Exception* lub *InvalidNumberException*). Oznaczona w ten sposób metoda musi wyrzucić wyjątek, aby test się powiódł. Do dyspozycji są dodatkowe adnotacje:
@expectedExceptionCode, *@expectedExceptionMessage* oraz *@expe*

Uff … wpis zrobił się trochę długi, ale z mnóstwem przykładów. Jeżeli udało ci się dobrnąć do końca, to przyjmij moje gratulacje, jesteś gotowy to przejścia na wyższy poziom. W następnym wpisie z tej serii zajmiemy się praktycznym testowanie całych klas. Jeżeli chciałbyś już teraz poszerzyć swoją wiedzę to zapraszam do mojego gościnnego wpisu: Testy jednostkowe z PHPUnit oraz Mockery. Jak zawsze zachęcam do komentowania, w miarę możliwości odpowiem na każde, nawet najtrudniejsze pytanie

Tags: php, phpunit, phpunit.xml, tdd, testy jednostkowe, unittest

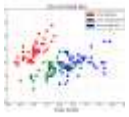
Arkadiusz Kondas

Entuzjasta programowania. Z zawodu web developer. Pragmatyk. Od jakiegoś czasu również przedsiębiorca. Racjonalista. W wolnych chwilach biega i bloguje. Miłośnik gier i grywalizacji. Więcej na jego temat znajdziesz w zakładce „O mnie” tego bloga.

RELATED POSTS

JAK KORZYSTAĆ Z TRAITS W PHP
0 Comments | sie 12, 2014

PHP 7.1 KOLEJNE ZMIANY I NOWOŚCI
1 Comment | lis 27, 2016



WYDAJNOŚĆ PHP I MACHINE LEARNING
3 Comments | kw. 3, 2016

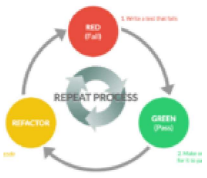
NORMALIZACJA DANYCH – PHP-ML
1 Comment | maj 31, 2016

TDD

HUMBUG – TESTY MUTACYJNE W PHP

GENEROWANIE RAPORTU CODE COVERAGE Z PHPUNIT

CODE COVERAGE W TESTACH JEDNOSTKOWYCH



RED GREEN REFACTOR – TESTY JEDNOSTKOWE

CZWARTA EDYCJA KONFERENCJI QUALITY EXCITES 2015!

TDD W PHP: TESTOWANIE BAZY DANYCH