Testowanie w PHP

v.1.3

Coders Lab
SZKOŁA PROGRAMOWANIA

Plan

- Wprowadzenie do testowania
- PHPUnit
- Test Driven Developement
- Fikstury
- Uruchamianie testów
- Organizacja i konfiguracja testów
- Testowanie bazy danych

Fikstury

Fikstury

- **Fikstury** umożliwiają uruchamianie testów w odpowiednim stanie naszej aplikacji.
- Polegają na włączeniu odpowiednich funkcji przed uruchomieniem i po uruchomieniu każdego testu lub klasy testów.
- Tworzymy je jako metody typu **protected**.

setUp() i tearDown()

Funkcje **setUp()** i **tearDown()** są włączane przed uruchomieniem i po uruchomieniu każdej funkcji testującej.

Zazwyczaj tworzymy w nich obiekty, które będziemy testować.

```
protected function setUp()
{
    parent::setUp();
    $this->testUser = UserManager::createUser();
}

protected function tearDown()
{
    $this->testUser = null;
    parent::tearDown();
}
public function testSetUpName(){
    $this->testUser->setName("Wojtek");
    $this->assertEquals($this->testUser->getName(),
"Wojtek");
}
```

setUpBeforeClass() i tearDownAfterClass()

```
protected static function setUpBeforeClass()
{
    UserManager::CreateDBConnection($host, $usr, $pass, $db);
}

protected static function tearDownAfterClass()
{
    UserManager::CloseDBConnecction;
}
```

Stan globalny podczas testów

- Często części naszej aplikacji opierają się na stanie globalnym (zmienne $_SESSION, $_GET, $_POST, …).
- Ogółem powinniśmy się wystrzegać bazowania na tej komunikacji w testach (szczególnie jednostkowych).
- Taka komunikacja powoduje wiele błędów, które później trudno znaleźć.

Aby korzystać ze zmiennych globalnych, musimy w naszej klasie testów nastawić odpowiednią zmienną:

```
protected $backupGlobals = ['globalVariable'];
```

Możemy wtedy w fiksturach uzupełniać odpowiednie części w tych zmiennych. Np.:

```
protected function setUp() {
    $_POST["userId"] = 1;
}
```

Uruchamianie testów

Uruchamianie testów z konsoli

Jeżeli z komendą **phpunit** podamy tylko katalog, to zostaną wczytane wszystkie pliki z tego katalogu i uruchomione wszystkie klasy dziedziczące po klasach testów.

Możemy jednak wyszczególnić plik testów albo całe grupy testów, które mają być uruchomione (**TestSuites**).

Możliwy output z testów

Nasze testy mogą zwrócić w konsoli następujące wartości:

.	Test przeszedł pozytywnie
F	Któraś z asercji nie została spełniona
E	Podczas wykonywania testu został wywołany błąd
S	Test został pominięty (skipped)
I	Test został oznaczony jako nieskończony (incomplete)

```
public function testSkipped()
{
    $this->markTestSkipped(
    'Test chwilowo wyłączony'
    );
}

public function testIncomplete()
{
    $this->markTestIncomplete(
        'Test nieskończony'
    );
}
```

Najważniejsze opcje konsolowe

Komenda **phpunit** ma wiele opcji, które można dodatkowo włączyć z poziomu konsoli.

Najważniejsze z nich są podane w tabeli.

--help	Wyświetla pomoc
--log-junit --log-json	Wypisanie outputu w podanym formacie
--testsuite	Zostaną włączone testy tylko z podanego scenariusza

Opcje pokrycia kodu

- Kiedy mamy podpięty **xdebug**, możemy też testować pokrycie kodu.
- Oznacza to że PHPUnit sprawdzi, ile procent naszego kodu jest używane podczas testowania.
- Służą do tego opcje:
  --coverage-text
  --coverage-php
  --coverage-html

Wynik

Code Coverage Report

2015-04-01 20:25:29

Summary:
Classes: 0.00% (0/3)
Methods: 41.67% (5/12)
Lines: 87.04% (47/54)
…

Organizacja i konfiguracja testów

Plik konfiguracyjny testów

Konfiguracja PHPUnit odbywa się przez stworzenie pliku **phpunit.xml**.

Pełny opis wszystkich funkcji służących do konfiguracji znajdziecie tutaj:

- http://phpunit.de/manual/current/en/appendixes.configuration.html

Atrybuty tagu phpunit

Plik **phpunit.xml** musi się zaczynać od tagu **phpunit** zawierającego w sobie wszystkie inne tagi.
Tag ten zawsze musi mieć dwie nastawione wartości:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://schema.phpunit.de/4.5/phpunit.xsd"
```

Najważniejsze opcje tego tagu to:

- **colors** – nastawione na **true** spowoduje kolorowanie testów w konsoli,
- **stopOnError**, **stopOnFailure**, **stopOnIncomplete**, **stopOnSkipped** – nastawienie tych wartości na **true** spowoduje przerwanie testów po spełnieniu odpowiedniego warunku.

Przykładowy plik konfiguracji

```
<phpunit
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://schema.phpunit.de/4.5/phpunit.xsd"
    backupGlobals="true"
    colors="true"
    stopOnError="true"
    verbose="false">
  <!-- ... -->
</phpunit>
```

Scenariusze – test suites

- Testy możemy grupować w scenariusze (zwane też **test suites**).
- Grupujemy przez stworzenie w konfiguracji tagu **testsuites**.
- Tag ten zawiera tagi **testsuite**, które przetrzymują konfigurację poszczególnych scenariuszy.

Każdy tag **testsuite** musi zawierać atrybut **name**.

Tagi te mogą mieć też następujące podtagi:

- **<directory>** – testy zawarte pod tą ścieżką zostaną dodane,
- **<file>** – test znajdujący się w danym pliku zostanie dodany,
- **<exclude>** – test znajdujący się w danym pliku zostanie usunięty ze scenariusza.

Przykładowy test suite

```
<testsuites>
    <testsuite name="All tests">
    <directory>/tests</directory>
    <file>/additional/test/MyTest.php</file>
    <exclude>/tests/quarantine</exclude>
    </testsuite>
</testsuites>
```

Uruchamianie test suitu

Żeby uruchomić dany **testsuite** wystarczy podać jego nazwę zaraz za parametrem:

--testsuite

Komenda

```
phpunit --testsuite "All_tests"
```

Wynik w konsoli

```
PHPUnit 3.7.21 by Sebastian Bergmann.
Configuration read from
C:\xampp\htdocs\phpunit\phpunit.xml


Time: 0 seconds, Memory: 1.75Mb
```

Tag <php>

W tagu <php> możemy ustawiać informacje konfiguracyjne dla PHP.

Możemy też w nim ustawiać początkowe wartości, jakie znajdą się w zmiennych superglobalnych.

Atrybuty tagu <php>

W tagu możemy użyć następujących atrybutów:

- **<includePath>.</includePath>** – rozszerza ścieżkę include,
- **<var name="foo" value="bar"/>** – nastawia wartość zmiennej foo na bar,
- **<post name="foo" value="bar"/>** – dodaje do zmiennej $_POST wartość bar pod kluczem foo,
- **<get name="foo" value="bar"/>** – dodaje do zmiennej $_GET wartość bar pod kluczem foo,
- **<cookie name="foo" value="bar"/>** – dodaje ciasteczko o nazwie foo i wartości bar.

Testowanie bazy danych

Testowanie bazy danych

- Niektóre testy powinny też sprawdzać nasze zapytania SQL skierowane do bazy danych.
- Nie chcemy jednak przeprowadzać testów na bazie danych dostępnej dla użytkownika lub deweloperów. Nie możemy bowiem wtedy zagwarantować niezmienności działania testów.
- Testowanie z użyciem prawdziwej bazy danych powinno być stosowane tylko w celu sprawdzenia, czy nasz kod MySQL jest poprawny.
- W innych przypadkach powinniśmy używać metod pozwalających nam oszukiwać testy i udawać bazę danych.

Dlaczego nie używamy bazy danych?

Testowanie z użyciem baz danych wprowadza wiele komplikacji:

- Jest długie (każdy dostęp do bazy trwa),
- potrzeba wiele czasu, żeby utrzymać poprawny stan bazy testowej.

Poszczególne kroki testowania bazy danych

Testowanie z użyciem bazy danych możemy podzielić na następujące kroki:

- tworzenie/czyszczenie bazy danych,
- nastawianie fikstur,
- przeprowadzanie testów.
- niszczenie/czyszczenie bazy danych.

Instalacja DbUnit

Najłatwiej (jak zawsze) skorzystać nam z composera:

```
{
  "require-dev": {
   "phpunit/phpunit": "3.7.*"
     "phpunit/dbunit": ">=1.2"
  }
  }
```

DbUnit

Co robi za nas DbUnit?

DbUnit zajmuje się za nas kilkoma ważnymi czynnościami:

- przed testem czyści naszą bazę danych (dlatego nie możemy używać jej na bazie produkcyjnej!),
- ładuje do bazy danych **fikstury**.

Czego nie robi DbUnit?

DbUnit nie zrobi za nas:

- tworzenia bazy danych do testów (tabele muszą być przygotowane wcześniej),
- czyszczenia bazy po testach.

Testowanie przy pomocy DbUnit

- Żeby zacząć używać DbUnit nasza klasa testów musi dziedziczyć po klasie **PHPUnit_Extensions_Database_TestCase**.

- Jest to klasa abstrakcyjna, musimy zatem zawsze zaimplementować dwie funkcje:
  getConnection()
  getDataSet()

Jak trzymać informacje dotyczące połączenia?

Najłatwiej trzymać informacje dotyczące połączenia w pliku konfiguracyjny **phpunit**.

```
<?xml version="1.0" encoding="UTF-8" ?>
<phpunit>
    <php>
        <var name="DB_DSN"
value="mysql:dbname=...;host=..." />
        <var name="DB_USER" value="user" />
        <var name="DB_PASSWD" value="passwd" />
        <var name="DB_DBNAME" value="my_db" />
    </php>
</phpunit>
```

getConnection()

Funkcja **getConnection()** tworzy nowe połączenie do bazy danych. Nie jest to jednak połączenie **mysqli**, którego dotychczas używaliśmy, ale specjalne połączenie do testów.

```
public function getConnection() {
    $conn = new PDO(
                        $GLOBALS['DB_DSN'],
                        $GLOBALS['DB_USER'],
                        $GLOBALS['DB_PASSWD']
                        );
      return new PHPUnit_Extensions_Database_DB_DefaultDatabaseConnection($conn, $GLOBALS['DB_NAME']);
}
```

getDataSet()

Metoda ta nastawia początkowy stan bazy danych
(czyli ładuje fikstury do pamięci).

DBUnit daje nam możliwość zdefiniowania
naszych fikstur w następujących rodzajach plików:

- XML,
- YAML,
- CSV.

getDataSet()

```
public function getDataSet() {

  $dataXML = $this->createXMLDataSet('myXmlFixture.xml');

  $dataFlatXML = $this->createFlatXmlDataSet('mydataset.xml');

  $dataYAML = new PHPUnit_Extensions_Database_DataSet_YamlDataSet('file.yml');

  $dataMysql = $this->createMySQLXMLDataSet('file.xml');

  $csv_data_set = new PHPUnit_Extensions_Database_DataSet_CsvDataSet();

  $csv_data_set->addTable('guestbook', 'file.csv');

  return $csv_data_set;
}
```

Format MySqlXML

- Jest to najwygodniejszy format do automatycznego tworzenia fikstur.
- Jest to format zrzutu danych z programu **mysqldump** z włączoną flagą --xml.

mysqldump --xml -t -u [username] -p [database] > file.xml

Format flat XML

Format flat XML jest najprostszym sposobem tworzenia fikstur.

Wystarczy stworzyć plik XML z głównym tagiem **<dataset>**.

Każdy tag jest potem traktowany jako jeden wpis do bazy danych, gdzie:

- nazwa tagu jest nazwą tabeli,
- atrybuty są nazwami kolumn w tabeli,
- wartości atrybutów są wartościami wpisu w tabeli.

Format flat XML

```
<?xml version="1.0" charset="utf-8" ?>
<dataset>
  <user
    date_created="2009−01−01 00:00:00"
    user_id="1"
    username="Test1"
    password="3858f62230ac3c915f300c664312c63f" />

  <user
    date_created="2009−01−02 00:00:00"
    user_id="2"
    username="Test2"
    password="73cf88a0b4a18c88a3996fa3d5b69a46" />
    </dataset>
```

Format XML

- Jest to drugi możliwy zapis fikstury w pliku XML. Choć jest o wiele dłuższy, to zapobiega wprowadzaniu nieświadomie wartości NULL.
- Mamy główny tag **<dataset>**, w którym zawieramy tagi **<table>** z atrybutem name.
- W tagu **<table>** zawieramy atrybuty **<column>** i **<row>**.

Kolejność wpisywania danych jest ważna!

```
<?xml version="1.0" ?>
<dataset>
    <table name="guestbook">
        <column>id</column>
        <column>content</column>
        <column>user</column>
        <column>created</column>
        <row>
            <value>1</value>
            <value>Hello buddy!</value>
            <value>joe</value>
            <value>2010-04-24 17:15:23</value>
        </row>
    </table>
</dataset>
```

Format CSV

- Format, w którym każda tabela musi znajdować się w osobnym pliku.
- W pierwszym rzędzie wpisujemy kolumny tabeli, a w następnych – wartości wpisów.

```
id,content,user,created
1,"Hello buddy!","joe","2010-04-24 17:15:23"
2,"I like it!","nancy","2010-04-26 12:14:20"
```

A co z kluczami obcymi?

- Jeśli tabele mają klucze obce, to musimy wpisywać dane w odpowiedniej kolejności (inaczej MySQL nam nie pozwoli na wpisanie danych).
- Możemy też temu zapobiec dzięki zastosowaniu poniższej funkcji:

```
$conn->getConnection()->query("set foreign_key_checks=0");

$conn->getConnection()->query("set foreign_key_checks=1");
```

Wyłączy błędy

Włączy błędy

Testowanie

- Podczas testowania musimy pamiętać, aby stworzyć połączenie do testowej bazy danych, a nie do produkcyjnej.

- Możemy mieć po prostu osobny plik **connection_test.php** używany tylko do testów.

Czas na zadania

- Przeróbcie ćwiczenia z drugiego dnia znajdujące się w katalogu 1_Testowanie_baz_danych.