Unit testing with PHPUnit

PHPUnit is the most popular PHP testing framework. It is simple for configuration and usage. Also, the  
framework supports code coverage reports and has a lot of additional plugins. Codeception from the  
previous recipe uses PHPUnit for own work and writing unit tests. In this recipe, we will create a  
demonstration shopping cart extension with PHPUnit tests.

Getting ready

Create a new yii2-app-basic application using the Composer package manager, as described in the  
official guide at <http://www.yiiframework.com/doc-2.0/guidestart-installation.html>.

How to do it...

First, we must create a new empty directory for own extension.

Preparing extension structure

1. First, create the directory structure for your extension:

book

1 cart

I— src  
'— tests

To work with the extension as a Composer package, prepare the book/cart/composer. j son file  
like this:

{

"name": "book/cart",

"type": "yii2-extension",

"require": {

"yiisoft/yii2": "~2.0"

},

"require-dev": {

"phpunit/phpunit": "4.\*"

},

"autoload": {

"psr -4": {

"book\\cart\\": "src/",

"book\\cart\\tests\\": "tests/"

}

},

"extra": {

"asset-installer-paths": {

"npm-asset-library": "vendor/npm",

"bower-asset-library": "vendor/bower"

}

}

}

1. Add the book/cart/. gitignore file with the following lines:

/vendor

/composer.lock

1. Add the following lines to the PHPUnit default configuration file book/cart/phpunit. xml. dist  
   like this:

<?xml version="1.0" encoding="utf-8"?>

<phpunit bootstrap="./tests/bootstrap.php"  
colors="true"

convertErrorsToExceptions="true"

convertNoticesToExceptions="true"

convertWarningsToExceptions="true"

stopOnFailure="false">

<testsuites>

<testsuite name="Test Suite">

<directory>./tests</directory>

</testsuite>

</testsuites>

<filter>

<whitelist>

<directory suffix=".php">./src/</directory>

</whitelist>

</filter>

</phpunit>

1. Install all the dependencies of the extension:

composer install

1. Now we must get the following structure:

book  
1 cart

* src
* tests
* .gitignore
* composer.j son
* phpunit.xml.dist
* vendor

Writing extension code

To write the extension code, follow these steps:

1. Create the book\cart\cart class in the src directory:

<?php

namespace book\cart;

use book\cart\storage\StorageInterface;

use yii\base\Component;

use yii\base\InvalidConfigException;

class Cart extends Component  
{

/\*\*

* @var StorageInterface  
  \*/

private $\_storage;

/\*\*

* @var array  
  \*/

private $\_items;

public function setStorage($storage)

{

if (is\_array($storage)) {

$this->\_storage = \Yii::createObject($storage);

} else {

$this->\_storage = $storage;

}

}

public function add($id, $amount = 1)

{

$this->loadItems();  
if (isset($this->\_items[$id])) {

$this->\_items[$id] += $amount;

} else {

$this->\_items[$id] = $amount;

}

$this->saveItems();

}

public function set($id, $amount)

{

$this->loadItems();

$this->\_items[$id] = $amount;

$this->saveItems();

}

public function remove($id)

{

$this->loadItems();  
if (isset($this->\_items[$id])) {  
unset($this->\_items[$id]);

}

$this->saveItems();

}

public function clear()

{

$this->loadItems();

$this->\_items = [];

$this->saveItems();

}

public function getItems()

{

$this->loadItems();  
return $this->\_items;

}

public function getCount()

{

$this->loadItems();

return count($this->\_items);

}

public function getAmount()

{

$this->loadItems();

return array\_sum($this->\_items);

}

private function loadItems()

{

if ($this->\_storage === null) {

throw new InvalidConfigException('Storage must be set');

}

if ($this->\_items === null) {

$this->\_items = $this->\_storage->load();

}

}

private function saveItems()

{

$this->\_storage->save($this->\_items);

}

}

1. Create Storageinterface interface in the src/storage subdirectory:  
   <?php

namespace book\cart\storage;

interface StorageInterface  
{

/\*\*

* @return array  
  \*/

public function load();

/\*\*

* @param array $items  
  \*/

public function save(array $items);

}

and SessionStorage class:  
namespace book\cart\storage;  
use Yii;

class SessionStorage implements StorageInterface  
{

public $sessionKey = 'cart';

public function load()

{

return Yii::$app->session->get($this->sessionKey, []);

}

public function save(array $items)

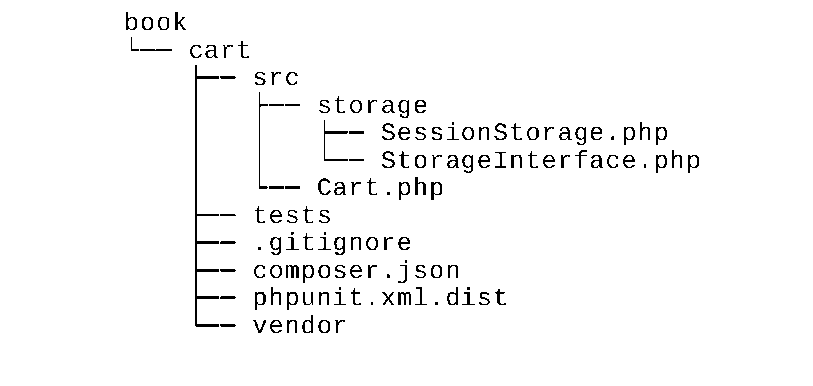
{

Yii::$app->session->set($this->sessionKey, $items);

}

}

1. Now we must get the following structure:



Writing extension tests

To conduct the extension test, follow these steps:

1. Add the book/cart/tests/bootstrap.php entry script for PHPUnit:

<?php

defined('YII\_DEBUG') or define('YII\_DEBUG', true);  
defined('YII\_ENV') or define('YII\_ENV', 'test');

require( DIR . '/../vendor/autoload.php');

require( DIR . '/../vendor/yiisoft/yii2/Yii.php');

1. Create a test base class by initializing the Yii application before each test and by destroying the  
   application afterwards:

<?php

namespace book\cart\tests;

use yii\di\Container;  
use yii\web\Application;

abstract class TestCase extends \PHPUnit\_Framework\_TestCase  
{

protected function setUp()

{

parent::setUp();

$this->mockApplication();

}

protected function tearDown()

{

$this->destroyApplication();  
parent::tearDown();

}

protected function mockApplication()

{

new Application([

'id' => 'testapp',

'basePath' => DIR ,

'vendorPath' => dirname( DIR ) . '/vendor',

]);

}

protected function destroyApplication()

{

\Yii::$app = null;

\Yii::$container = new Container();

}

}

1. Add a memory-based clean fake class that implements the StorageInterface interface:

<?php

namespace book\cart\tests\storage;

use book\cart\storage\StorageInterface;

class FakeStorage implements StorageInterface

{

private $items = [];

public function load()

{

return $this->items;

}

public function save(array $items)

{

$this->items = $items;

}

}

It will store items into a private variable instead of working with a real session. It allows to run tests  
independently (without real storage driver) and also improves testing performance.

4. Add the cartTest class:

<?php

namespace book\cart\tests;  
use book\cart\Cart;

use book\cart\tests\storage\FakeStorage;

class CartTest extends TestCase  
{

/\*\*

\* @var Cart  
\*/

private $cart;

public function setUp()

{

parent::setUp();

$this->cart = new Cart(['storage' => new FakeStorage()]);

}

public function testEmpty()

{

$this->assertEquals([], $this->cart->getItems());

$this->assertEquals(0, $this->cart->getCount());

$this->assertEquals(0, $this->cart->getAmount());

}

public function testAdd()

{

$this->cart->add(5, 3);

$this->assertEquals([5 => 3], $this->cart->getItems());

$this->cart->add(7, 14);

$this->assertEquals([5 => 3, 7 => 14], $this->cart->getItems());  
$this->cart->add(5, 10);

$this->assertEquals([5 => 13, 7 => 14], $this->cart->getItems());

}

public function testSet()

{

$this->cart->add(5, 3);

$this->cart->add(7, 14);

$this->cart->set(5, 12);

$this->assertEquals([5 => 12, 7 => 14], $this->cart->getItems());

}

public function testRemove()

{

$this->cart->add(5, 3);

$this->cart->remove(5);

$this->assertEquals([], $this->cart->getItems());

}

public function testClear()

{

$this->cart->add(5, 3);

$this->cart->add(7, 14);

$this->cart->clear();

$this->assertEquals([], $this->cart->getItems());

}

public function testCount()

{

$this->cart->add(5, 3);

$this->assertEquals(1, $this->cart->getCount());  
$this->cart->add(7, 14);

$this->assertEquals(2, $this->cart->getCount());

}

public function testAmount()

{

$this->cart->add(5, 3);

$this->assertEquals(3, $this->cart->getAmount());  
$this->cart->add(7, 14);

$this->assertEquals(17, $this->cart->getAmount());

}

public function testEmptyStorage()

{

$cart = new Cart();

$this->setExpectedException('yii\base\InvalidConfigException');  
$cart->getItems();

}

}

5. Add a separated test for checking the SessionStorage class:

<?php

namespace book\cart\tests\storage;

use book\cart\storage\SessionStorage;  
use book\cart\tests\TestCase;

class SessionStorageTest extends TestCase  
{

/\*\*

\* @var SessionStorage  
\*/

private $storage;

public function setUp()

{

parent::setUp();

$this->storage = new SessionStorage(['key' => 'test']);

}

public function testEmpty()

{

$this->assertEquals([], $this->storage->load());

}

public function testStore()

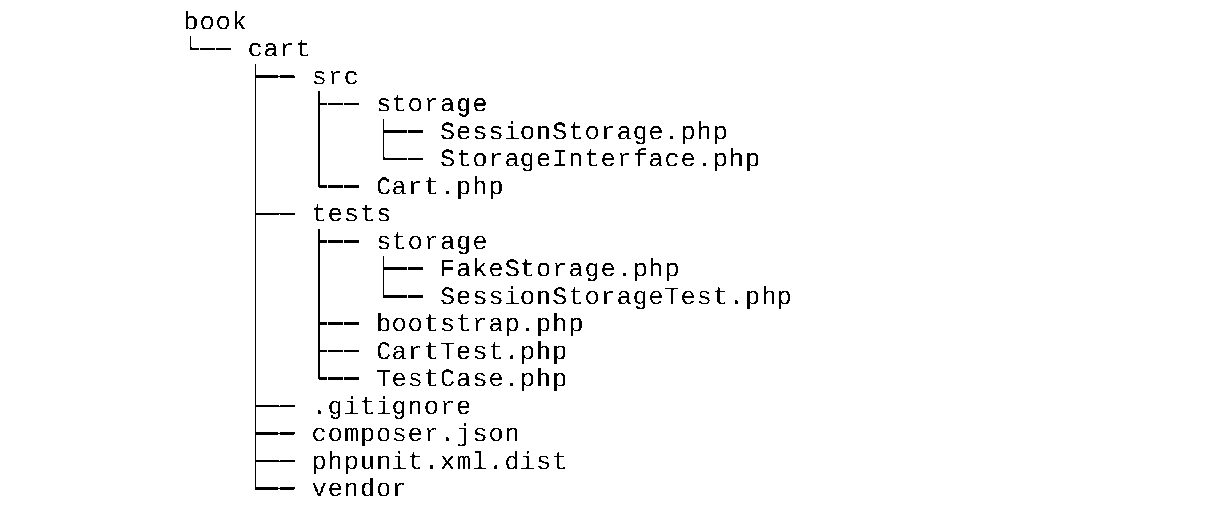
{

$this->storage->save($items = [1 => 5, 6 => 12]);  
$this->assertEquals($items, $this->storage->load());

}

}

6. Right now we must get the following structure:



Running tests

During the installation of all dependencies with the composer install command, the Composer package  
manager installs the PHPUnit package into the vendor directory and places the executable file phpunit in  
the vendor/bin subdirectory.

Now we can run the following script:

cd book/cart  
vendor/bin/phpunit

We must see the following testing report:

PHPUnit 4.8.26 by Sebastian Bergmann and contributors.

Time: 906 ms, Memory: 11.50MB  
OK (10 tests, 16 assertions)

Each dot shows a success result of the correspondent test.

Try to deliberately break an own cart by commenting the unset operation:

class Cart extends Component  
{

public function remove($id)

{

$this->loadItems();  
if (isset($this->\_items[$id])) {

// unset($this->\_items[$id]);

}

$this->saveItems();

}

}

Run the tests again:

PHPUnit 4.8.26 by Sebastian Bergmann and contributors.

. . . F

Time: 862 ms, Memory: 11.75MB  
There was 1 failure:

1) book\cart\tests\CartTest::testRemove  
Failed asserting that two arrays are equal.

Expected

+++ Actual  
@@ @@

Array (

+ 5 => 3

)

/book/cart/tests/CartTest.php:52  
FAILURES!

Tests: 10, Assertions: 16, Failures: 1

In this case, we have seen one failure (marked as f instead of dot) and a failure report.

Analyzing code coverage

You must install the XDebug PHP extension from <https://xdebug.org>. For example, on Ubuntu or Debian,  
you can type the following in your terminal:

sudo apt-get install php5-xdebug

On Windows, you must open the php. ini file and add the custom code with path to your PHP installation  
directory:

[xdebug]

zend\_extension\_ts=C:/php/ext/php\_xdebug.dll

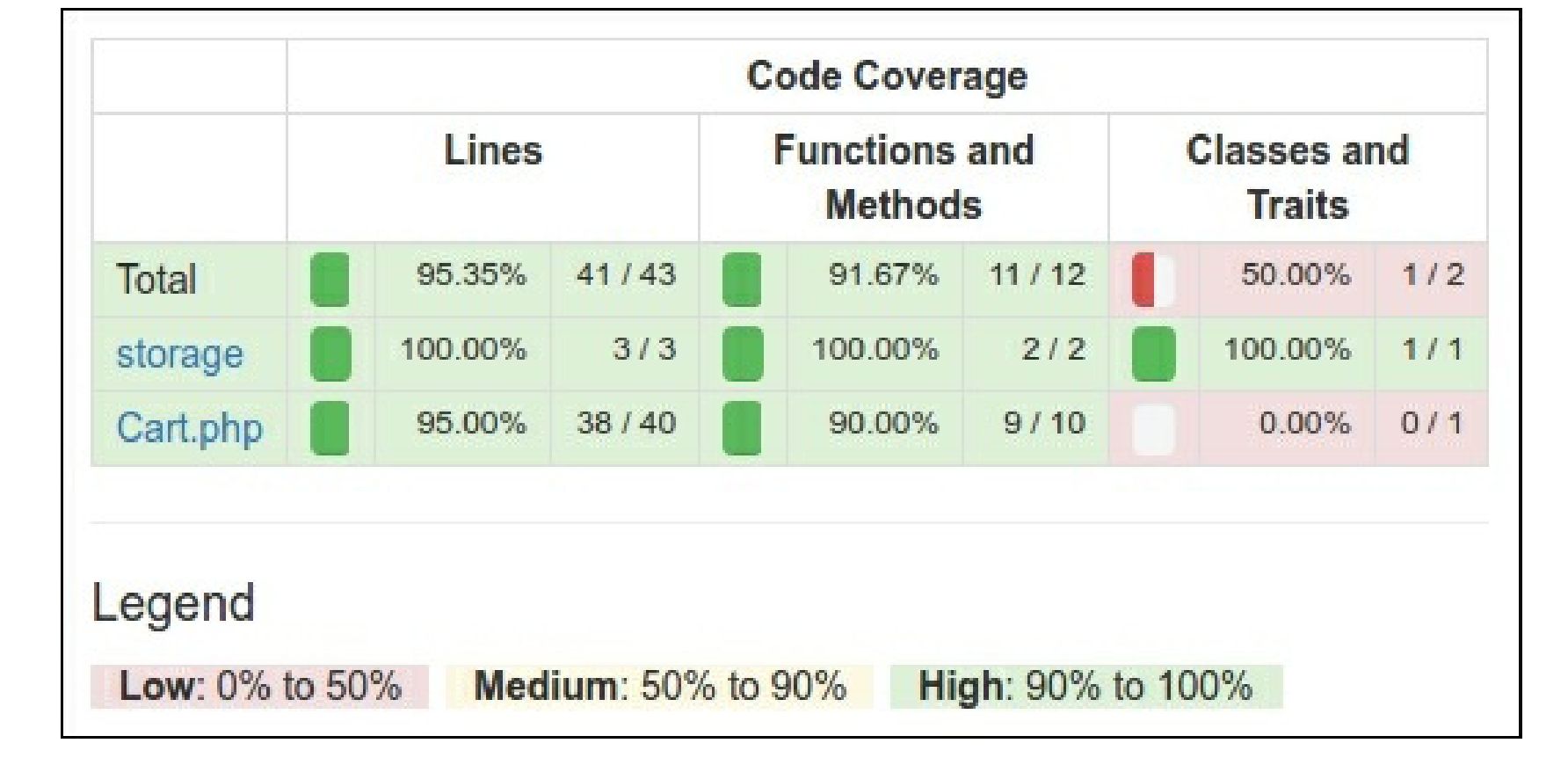
Alternatively, if you use the non-thread safe edition, type the following:

[xdebug]

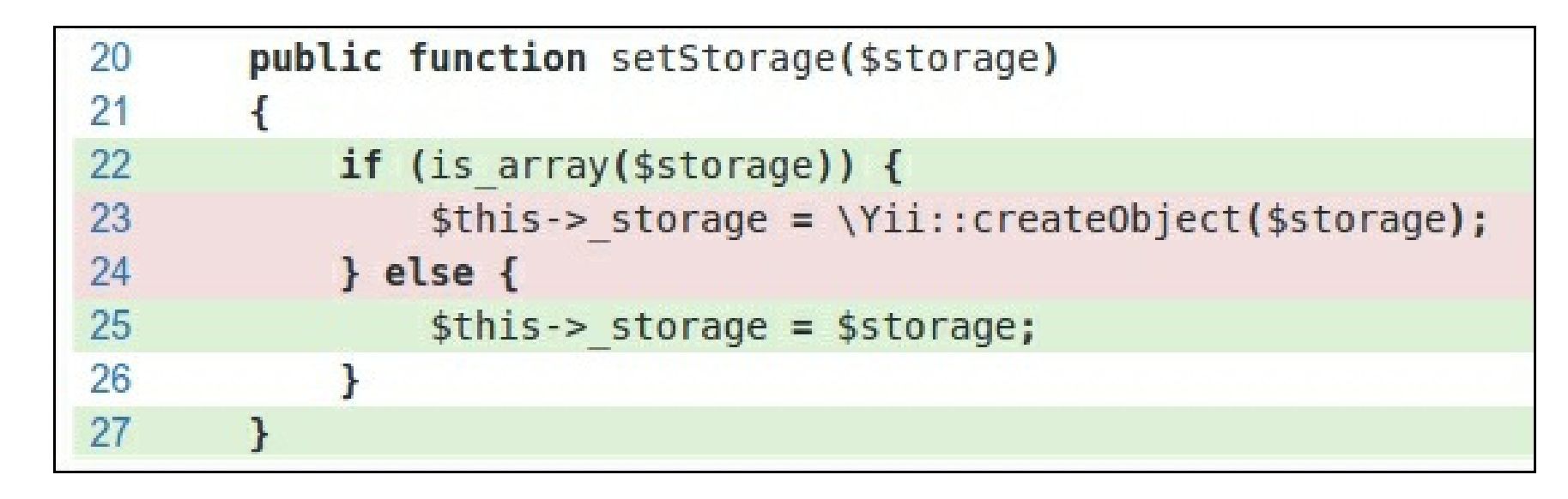
zend\_extension=C:/php/ext/php\_xdebug.dll

After installing XDebug, run the tests again with the --coverage-html flag and specify a report directory:  
vendor/bin/phpunit --coverage-html tests/\_output

After running open the tests/\_output/index. html file in your browser, you will see an explicit coverage  
report for each directory and class:



You can click on any class and analyze which lines of code have not been executed during the testing  
process. For example, open our Cart class report:



In our case, we forgot to test the creating storage from array configuration.

Usage of component

After publishing the extension on Packagist, we can install a one-to-any project:  
composer require book/cart

Also, enable the component in the application configuration file:

'components' => [

// ...

'cart' => [

'class' => 'book\cart\Cart',

'storage' => [

'class' => 'book\cart\storage\SessionStorage',

],

],

],

As an alternative way without publishing the extension on Packagist, we must set up the @book alias for  
enabling correct class autoloading:

$config = [

'id' => ' basic',

'basePath' => dirname( DIR ),

'bootstrap' => ['log'],

'aliases' => [

'@book' => dirname( DIR ) . '/book',

],

'components' => [

'cart' => [

'class' => 'book\cart\Cart',

'storage' => [

'class' => 'book\cart\storage\SessionStorage',

],

],

// ...

],

]

Anyway, we can use it as the Yii: : $app->cart component in our project:

Yii::$app->cart->add($product->id, $amount);

How it works...

Before creating your own tests, you must just create any subdirectory and add the phpunit. xml or  
phpunit. xml. dist file in the root directory of your project:

<?xml version="1.0" encoding="utf-8"?>

<phpunit bootstrap="./tests/bootstrap.php"  
colors="true"

convertErrorsToExceptions="true"

convertNoticesToExceptions="true"

convertWarningsToExceptions="true"

stopOnFailure="false">

<testsuites>

<testsuite name="Test Suite">

<directory>./tests</directory>

</testsuite>

</testsuites>

<filter>

<whitelist>

<directory suffix=".php">./src/</directory>

</whitelist>

</filter>

</phpunit>

PHPUnit loads configuration from the second file if the first one does not exist in the working directory.  
Also, you can create the bootstrap. php file by initializing autoloader and your framework’s environments:

<?php

defined('YII\_DEBUG') or define('YII\_DEBUG', true);  
defined('YII\_ENV') or define('YII\_ENV', 'test');

require( DIR . '/../vendor/autoload.php');

require( DIR . '/../vendor/yiisoft/yii2/Yii.php');

Finally, you can install PHPUnit via Composer (locally or globally) and use the phpunit console command  
in the directory with the XML configuration file.

PHPUnit scans the testing directory and finds files with the \*Test. php suffix. All your test classes must  
extend the PHPUnit\_Framework\_Testcase class and contain public methods with the test\* prefix like this:

class MyTest extends TestCase  
{

public function testSomeFunction()

{

$this->assertTrue(true);

}

}

In the body of your tests, you can use any of the existing assert\* methods:

$this->assertEqual('Alex', $model->name);

$this->assertTrue($model->validate());

$this->assertFalse($model->save());

$this->assertCount(3, $items);

$this->assertArrayHasKey('username', $model->getErrors());  
$this->assertNotNull($model->author);

$this->assertInstanceOf('app\models\User', $model->author);

Also, you can override the setUp() or tearDown() methods for adding expressions that will be run before  
and after each test method.

For example, you can define own base TestCase class by reinitializing the Yii application:

<?php

namespace book\cart\tests;

use yii\di\Container;  
use yii\web\Application;

abstract class TestCase extends \PHPUnit\_Framework\_TestCase  
{

protected function setUp()

{

parent::setUp();

$this->mockApplication();

}

protected function tearDown()

{

$this->destroyApplication();  
parent::tearDown();

}

protected function mockApplication()

{

new Application([

'id' => 'testapp',

'basePath' => DIR ,

'vendorPath' => dirname( DIR ) . '/vendor',

]);

}

protected function destroyApplication()

{

\Yii::$app = null;

\Yii::$container = new Container();

}

}

Now you can extend this class in your subclasses. Even your test method will work with an own instance  
of the application. It helps to avoid side effects and to create independent tests.

Note

Yii 2.0.\* uses the old PHPUnit 4.\* version for compatibility with PHP 5.4.

See also

* For all information about PHPUnit usage, refer to the official documentation at  
  <https://phpunit.de/manual/current/en/index.html>
* The Testing application with Codeception recipe