Unit testing with Atoum

Besides PHPUnit and Codeception, Atoum is a simple unit testing framework. You can use this  
framework for testing your extensions or for testing a code of your application.

Getting ready

Create an empty directory for the new project.

How to do it...

In this recipe, we will create a demonstration shopping cart extension with Atoum tests.

Preparing the extension structure

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

book

1 cart

I— src  
'— tests

1. For working with the extension as a composer package, prepare the book/cart/composer. j son file  
   as follows:

{

"name": "book/cart",

"type": "yii2-extension",

"require": {

"yiisoft/yii2": "~2.0"

},

"require-dev": {

"atoum/atoum": "A2.7"

},

"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 following lines to the book/cart/, gitignore file:

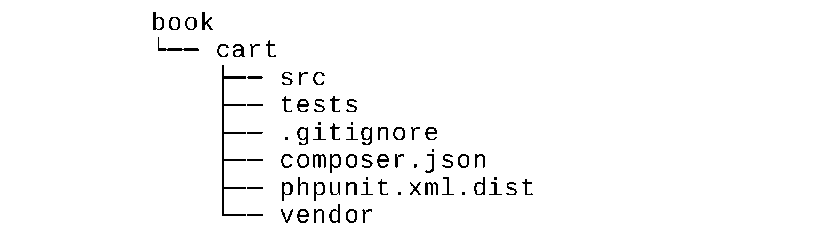
/vendor

/composer.lock

1. Install all the dependencies of the extension:

composer install

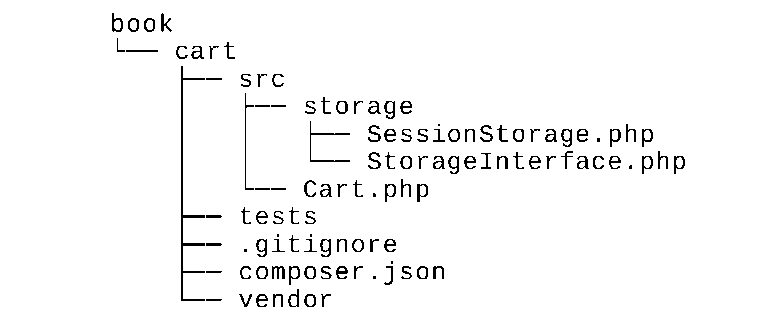
5. Now we will get the following structure:



Writing the extension code

Copy the Cart, Storageinterface, and Sessionstorage classes from the Unit testing with PHPUnit  
recipe.

Finally, we must get the following structure:



Writing the extension tests

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

<?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 after ones:

<?php

namespace book\cart\tests;

use yii\di\Container;

use yii\console\Application;

use mageekguy\atoum\test;

abstract class TestCase extends test  
{

public function beforeTestMethod($method)

{

parent::beforeTestMethod($method);

$this->mockApplication();

}

public function afterTestMethod($method)

{

$this->destroyApplication();  
parent::afterTestMethod($method);

}

protected function mockApplication()

{

new Application([

'id' => 'testapp',

'basePath' => DIR ,

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

'components' => [

'session' => [

'class' => 'yii\web\Session',

],

]

]);

}

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;

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;

}

}

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

1. Add the Cart test class:

<?php

namespace book\cart\tests\units;

use book\cart\tests\FakeStorage;  
use book\cart\Cart as TestedCart;  
use book\cart\tests\TestCase;

class Cart extends TestCase  
{

/\*\*

\* @var TestedCart

\*/

private $cart;

public function beforeTestMethod($method)

{

parent::beforeTestMethod($method );

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

}

public function testEmpty()

{

$this->array($this->cart->getItems())->isEqualTo([]);  
$this->integer($this->cart->getCount())->isEqualTo(0);  
$this->integer($this->cart->getAmount())->isEqualTo(0);

}

public function testAdd()

{

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

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

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

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

}

public function testSet()

{

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

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

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

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

}

public function testRemove()

{

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

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

$this->array($this->cart->getItems())->isEqualTo([]);

}

public function testClear()

{

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

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

$this->cart->clear();

$this->array($this->cart->getItems())->isEqualTo([]);

}

public function testCount()

{

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

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

$this->integer($this->cart->getCount())->isEqualTo(2);

}

public function testAmount()

{

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

$this->integer($this->cart->getAmount())->isEqualTo(3);

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

$this->integer($this->cart->getAmount())->isEqualTo(17);

}

public function testEmptyStorage()

{

$cart = new TestedCart();

$this->exception(function () use ($cart) {  
$cart->getItems();

})->hasMessage('Storage must be set');

}

}

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

<?php

namespace book\cart\tests\units\storage;

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

class SessionStorage extends TestCase  
{

/\*\*

\* @var TestedStorage  
\*/

private $storage;

public function beforeTestMethod($method)

{

parent::beforeTestMethod($method);

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

}

public function testEmpty()

{

$this

->given($storage = $this->storage)

->then

->array($storage->load())

->isEqualTo([]);

}

public function testStore()

{

$this

->given($storage = $this->storage)  
->and($storage->save($items = [1 => 5, 6 => 12]))  
->then

->array($this->storage->load())

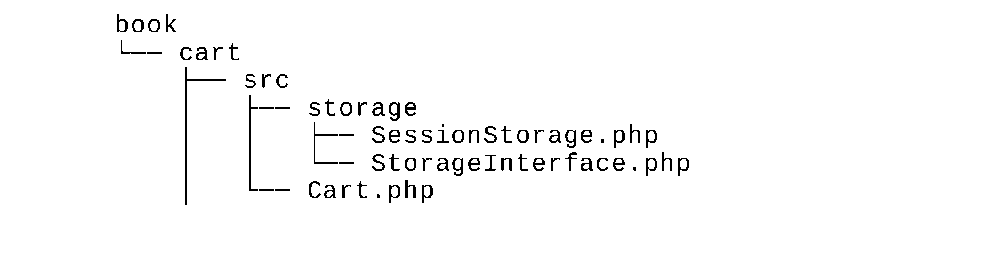
->isEqualTo($items)

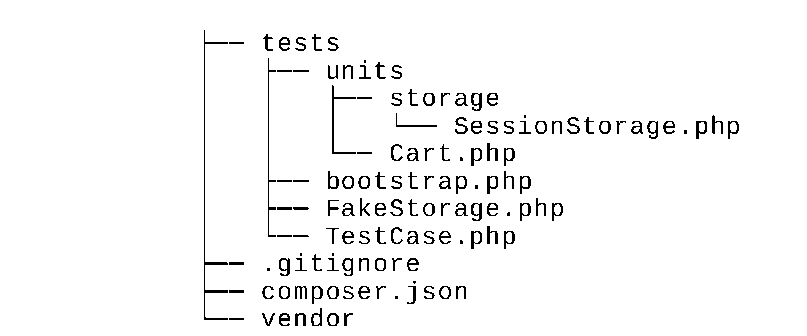
;

}

}

1. Now we will get the following structure:





Running tests

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

Now we can run the following script:

cd book/cart

vendor/bin/atoum -d tests/units -bf tests/bootstrap.php

Also, we must see the following testing report:

* atoum path: /book/cart/vendor/atoum/atoum/vendor/bin/atoum
* atoum version: 2.7.0
* atoum path: /book/cart/vendor/atoum/atoum/vendor/bin/atoum
* atoum version: 2.7.0
* PHP path: /usr/bin/php5
* PHP version:

=> PHP 5.5.9-1ubuntu4.16 (cli)

* book\cart\tests\units\Cart...

[SSSSSSSS ] [8/8]

=> Test duration: 1.13 seconds.

=> Memory usage: 3.75 Mb.

* book\cart\tests\units\storage\SessionStorage...

[SS ] [2/2]

=> Test duration: 0.03 second.

=> Memory usage: 1.00 Mb.

* Total tests duration: 1.15 seconds.
* Total tests memory usage: 4.75 Mb.
* Code coverage value: 16.16%

Each s symbol shows a success result of the correspondent test.

Try to deliberately break the 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:

* atoum version: 2.7.0
* PHP path: /usr/bin/php5
* PHP version:

=> PHP 5.5.9-1ubuntu4.16 (cli)  
book\cart\tests\units\Cart...

[SSFSSSSS ] [8/8]

=> Test duration: 1.09 seconds.

=> Memory usage: 3.25 Mb.

* book\cart\tests\units\storage\SessionStorage...

[SS ] [2/2]

=> Test duration: 0.02 second.

=> Memory usage: 1.00 Mb.

Failure (2 tests, 10/10 methods, 0 void method, 0 skipped method, 0 uncompleted method, 1  
failure, 0 error, 0 exception)!

* There is 1 failure:

=> book\cart\tests\units\Cart::testRemove():

In file /book/cart/tests/units/Cart.php on line 53, mageekguy\atoum\asserters\phpArray()  
failed: array(1) is not equal to array(0)

-Expected

+Actual

@@ -1 +1,3 @@

-array(0) {

+array(1) {

+ [5] =>

+ int(3)

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 the 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, create the book/cart/coverage. php configuration file with coverage report  
options:

<?php

use \mageekguy\atoum;

/\*\* @var atoum\scripts\runner $script \*/

$report = $script->addDefaultReport();

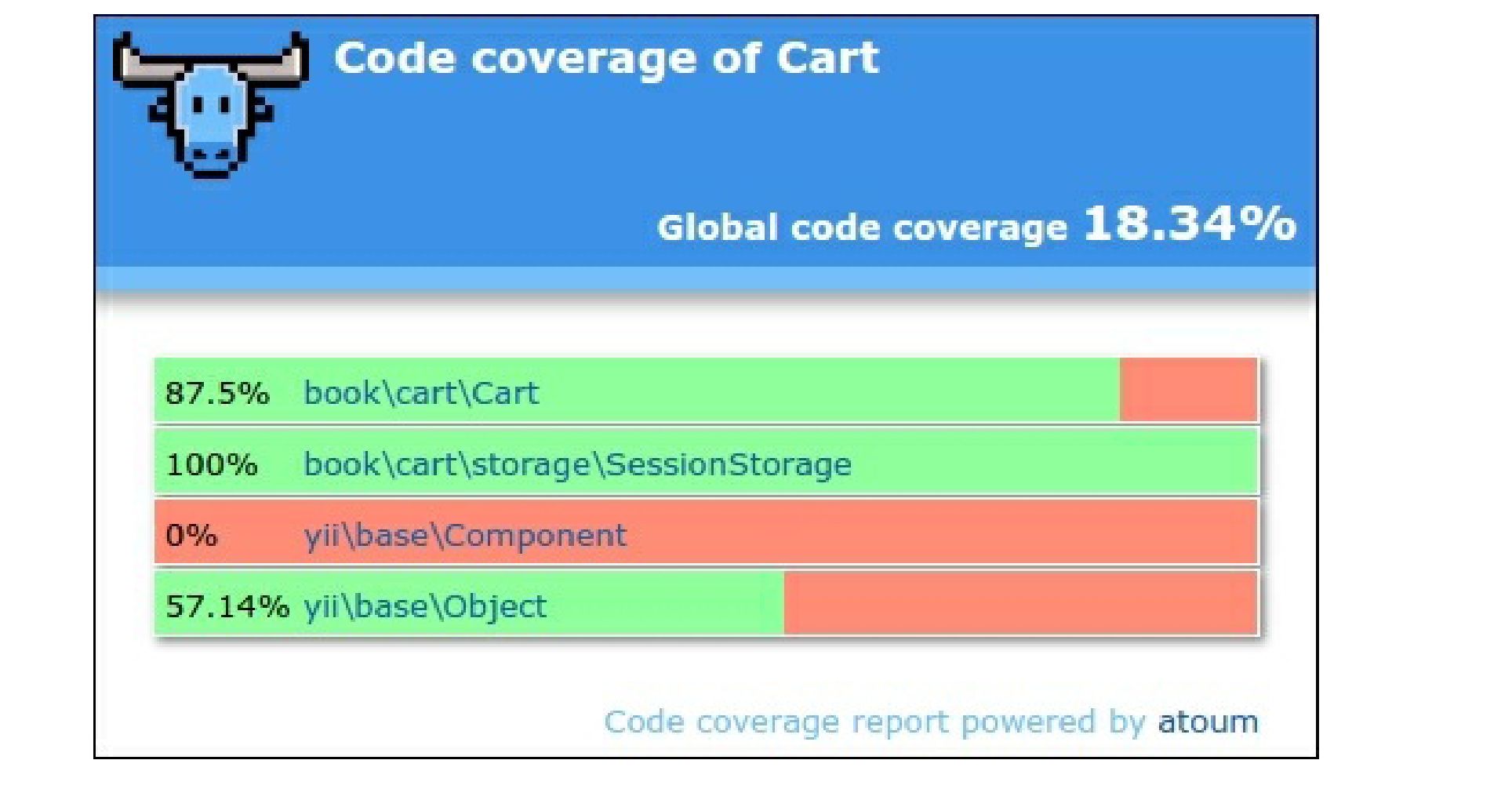
$coverageField = new atoum\report\fields\runner\coverage\html('Cart', DIR .

' /tests/coverage');

$report->addField($coverageField);

Now run the tests again with the -c option to use this configuration:  
vendor/bin/atoum -d tests/units -bf tests/bootstrap.php -c coverage.php

After running the tests, open the tests/coverage/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.

How it works...

The Atoum testing framework supports the Behavior-Driven Design (BDD) syntax flow, as follows:

public function testSome()

{

$this

->given($cart = new TestedCart())

->and($cart->add(5, 13))

->then

->sizeof($cart->getItems())

->isEqualTo(1)

->array($cart->getItems())

->isEqualTo([5 => 3])

->integer($cart->getCount())

->isEqualTo(1)

->integer($cart->getAmount())

->isEqualTo(3);

}

However, you can use the usual PHPUnit-like syntax to write unit tests:

public function testSome()

{

$cart = new TestedCart();

$cart->add(5, 3);

$this

->array($cart->getItems())->isEqualTo([5 => 3])  
->integer($cart->getCount())->isEqualTo(1)

->integer($cart->getAmount())->isEqualTo(3)

;

}

Atoum also supports code coverage reports for analyzing the testing quality.

See also

* For more information about Atoum, refer to <http://docs.atoum.org/en/latest/>
* For sources and usage samples, refer to [https ://github.com/atoum/atoum](https://github.com/atoum/atoum)
* The Unit testing with PHPUnit recipe