Unit testing with Behat

Behat is a BDD framework for testing your code with human-readable sentences that describes code  
behavior in various use cases.

Getting ready

Create an empty directory for a new project.

How to do it...

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

Preparing extension structure

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

book

1 cart

I— src  
'— features

1. To work 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": {

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

"behat/behat": "A3.1"

},

"autoload": {

"psr -4": {

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

"book\\cart\\features\\": "features/"

}

},

"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:

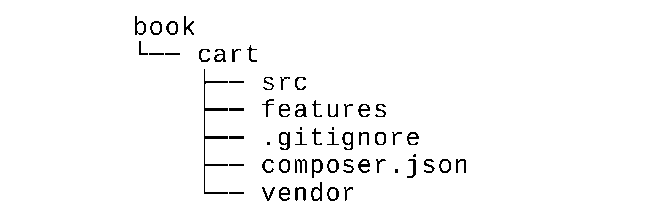
/vendor

/composer.lock

1. Install all the dependencies of the extension:

composer install

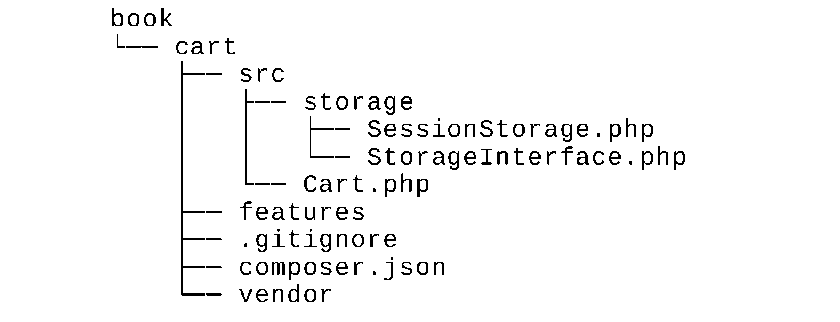
5. Now we get the following structure:



Writing extension code

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

Finally, we get the following structure:



Writing extension tests

1. Add the book/cart/features/bootstrap/bootstrap . php entry script:  
   <?php

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

require\_once DIR . '/../../vendor/yiisoft/yii2/Yii.php';

1. Create the features/cart.feature file and write cart testing scenarios:

Feature: Shopping cart  
In order to buy products  
As a customer

I need to be able to put interesting products into a cart

Scenario: Checking empty cart  
Given there is a clean cart  
Then I should have 0 products  
Then I should have 0 product  
And the overall cart amount should be 0

Scenario: Adding products to the cart  
Given there is a clean cart  
When I add 3 pieces of 5 product  
Then I should have 3 pieces of 5 product  
And I should have 1 product  
And the overall cart amount should be 3

When I add 14 pieces of 7 product  
Then I should have 3 pieces of 5 product

And I should have 14 pieces of 7 product

And I should have 2 products

And the overall cart amount should be 17

When I add 10 pieces of 5 product

Then I should have 13 pieces of 5 product

And I should have 14 pieces of 7 product

And I should have 2 products

And the overall cart amount should be 27

Scenario: Change product count in the cart

Given there is a cart with 5 pieces of 7 product

When I set 3 pieces for 7 product

Then I should have 3 pieces of 7 product

Scenario: Remove products from the cart

Given there is a cart with 5 pieces of 7 product

When I add 14 pieces of 7 product

And I clear cart

Then I should have empty cart

1. Add the storage test features/storage . feature file:

Feature: Shopping cart storage

I need to be able to put items into a storage

Scenario: Checking empty storage  
Given there is a clean storage  
Then I should have empty storage

Scenario: Save items into storage  
Given there is a clean storage

When I save 3 pieces of 7 product to the storage  
Then I should have 3 pieces of 7 product in the storage

1. Add implementation for all steps in the features/bootstrap/CartContext.php file:  
   <?php

use Behat\Behat\Context\SnippetAcceptingContext;  
use book\cart\Cart;

use book\cart\features\bootstrap\storage\FakeStorage;  
use yii\di\Container;  
use yii\web\Application;

require\_once DIR . '/bootstrap.php';

class CartContext implements SnippetAcceptingContext  
{

/\*\*

* @var Cart
* \*/

private $cart;

/\*\*

* @Given there is a clean cart  
  \*/

public function thereIsACleanCart()

{

$this->resetCart();

}

/\*\*

* @Given there is a cart with :pieces of :product product  
  \*/

public function thereIsAWhichCostsPs($product, $amount)

{

$this->resetCart();

$this->cart->set($product, floatval($amount));

}

/\*\*

* @When I add :pieces of :product  
  \*/

public function iAddTheToTheCart($product, $pieces)

{

$this->cart->add($product, $pieces);

}

/\*\*

* @When I set :pieces for :arg2 product  
  \*/

public function iSetPiecesForProduct($pieces, $product)  
{

$this->cart->set($product, $pieces);

}

/\*\*

* @When I clear cart  
  \*/

public function iClearCart()

{

$this->cart->clear();

}

/\*\*

* @Then I should have empty cart  
  \*/

public function iShouldHaveEmptyCart()

{

PHPUnit\_Framework\_Assert::assertEquals(

0,

$this->cart->getCount()

);

}

/\*\*

* @Then I should have :count product(s)

\*/

public function iShouldHaveProductInTheCart($count)

{

PHPUnit\_Framework\_Assert::assertEquals(  
intval($count),

$this->cart->getCount()

);

}

/\*\*

* @Then the overall cart amount should be :amount

\*/

public function theOverallCartPriceShouldBePs($amount)

{

PHPUnit\_Framework\_Assert::assertSame(  
intval($amount),

$this->cart->getAmount()

);

}

/\*\*

* @Then I should have :pieces of :product

\*/

public function iShouldHavePiecesOfProduct($pieces, $product)

{

PHPUnit\_Framework\_Assert::assertArraySubset(

[intval($product) => intval($pieces)],

$this->cart->getItems()

);

}

private function resetCart()

{

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

}

}

5. Also, in the features/bootstrap/StorageContext.php file, add the following:

<?php

use Behat\Behat\Context\SnippetAcceptingContext;  
use book\cart\Cart;

use book\cart\features\bootstrap\storage\FakeStorage;  
use book\cart\storage\SessionStorage;  
use yii\di\Container;  
use yii\web\Application;

require\_once DIR . '/bootstrap.php';

class StorageContext implements SnippetAcceptingContext  
{

/\*\*

* @var SessionStorage
* \*/

private $storage;

/\*\*

* @Given there is a clean storage  
  \*/

public function thereIsACleanStorage()

{

$this->mockApplication();

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

}

/\*\*

* @When I save :pieces of :product to the storage  
  \*/

public function iSavePiecesOfProductToTheStorage($pieces, $product)

{

$this->storage->save([$product => $pieces]);

}

/\*\*

* @Then I should have empty storage  
  \*/

public function iShouldHaveEmptyStorage()

{

PHPUnit\_Framework\_Assert::assertCount(

0,

$this->storage->load()

);

}

/\*\*

* @Then I should have :pieces of :product in the storage  
  \*/

public function iShouldHavePiecesOfProductInTheStorage($pieces, $product)

{

PHPUnit\_Framework\_Assert::assertArraySubset(

[intval($product) => intval($pieces)],

$this->storage->load()

);

}

private function mockApplication()

{

Yii::$container = new Container();  
new Application([

'id' => 'testapp',

'basePath' => DIR ,

'vendorPath' => DIR . '/../../vendor',

]);

}

}

1. Add the features/bootstrap/CartContext/FakeStorage . php file with a fake storage class:  
   <?php

namespace book\cart\features\bootstrap\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;

}

}

1. Add book/cart/behat.yml with contexts definition:

default:

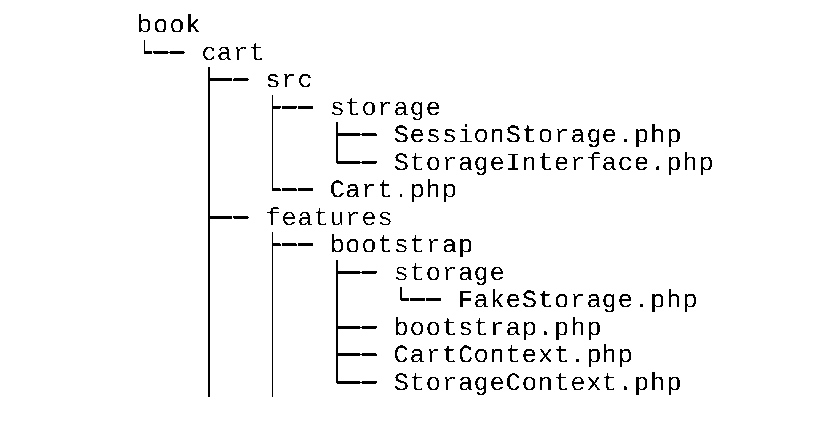
suites:

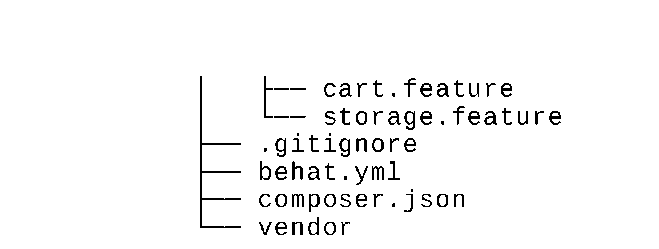
default:

contexts:

* CartContext
* StorageContext

1. Now we will get the following structure:





Now we can run our tests.

Running tests

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

Now we can run the following script:

cd book/cart  
vendor/bin/behat

Also, we must see the following testing report:

Feature: Shopping cart  
In order to buy products  
As a customer

I need to be able to put interesting products into a cart

Scenario: Checking empty cart # features/cart.feature:6

Given there is a clean cart # thereIsACleanCart()

Then I should have 0 products # iShouldHaveProductInTheCart()

Then I should have 0 product # iShouldHaveProductInTheCart()

And the overall cart amount should be 0 # theOverallCartPriceShouldBePs()

Feature: Shopping cart storage

I need to be able to put items into a storage

Scenario: Checking empty storage # features/storage.feature:4

Given there is a clean storage # thereIsACleanStorage()

Then I should have empty storage # iShouldHaveEmptyStorage()

6 scenarios (6 passed)

31 steps (31 passed)

0m0.23s (13.76Mb)

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

class Cart extends Component  
{

public function set($id, $amount)

{

$this->loadItems();

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

$this->saveItems();

}

}

Now run the tests again:

Feature: Shopping cart  
In order to buy products  
As a customer  
Feature: Shopping cart  
In order to buy products  
As a customer

I need to be able to put interesting products into a cart

Scenario: Change product count in the cart # features/cart.feature:31

Given there is a cart with 5 pieces of 7 prod # thereIsAWhichCostsPs()

When I set 3 pieces for 7 product # iSetPiecesForProduct()

Then I should have 3 pieces of 7 product # iShouldHavePiecesOf()

Failed asserting that an array has the subset Array &0 (

7 => 3

).

Scenario: Remove products from the cart # features/cart.feature:36

Given there is a cart with 5 pieces of 7 prod # thereIsAWhichCostsPs()

When I add 14 pieces of 7 product # iAddTheToTheCart()

And I clear cart # iClearCart()

Then I should have empty cart # iShouldHaveEmptyCart()

— Failed scenarios:

features/cart.feature:31

6 scenarios (5 passed, 1 failed)

31 steps (30 passed, 1 failed)

0m0.22s (13.85Mb)

In this case, we have seen one failure and a failure report.

How it works...

Behat is a BDD testing framework. It facilitates writing preceding human-readable testing scenarios to  
low-level technical implementation.

When we write scenarios for every feature, we can use a set of operators:

Scenario: Adding products to the cart  
Given there is a clean cart  
When I add 3 pieces of 5 product  
Then I should have 3 pieces of 5 product  
And I should have 1 product  
And the overall cart amount should be 3

Behat parses our sentences and finds the associated implementation of the sentence in the context class:

class FeatureContext implements SnippetAcceptingContext  
{

/\*\*

\* @When I add :pieces of :product  
\*/

public function iAddTheToTheCart($product, $pieces)

{

$this->cart->add($product, $pieces);

}

}

You can create a single FeatureContex t class (by default) or create a set of specific contexts for feature  
groups and scenarios.

See also

For getting more information about Behat refer to the following URLs:

* <http://docs.behat.org/en/v3.0/>
* [https ://github. c om/B ehat/B ehat](https://github.com/Behat/Behat)

And to get more information about alternative test frameworks, see the other recipes in this chapter.