**Приемочные тесты -кратко**

Приемочные тесты расчитаны на то, что их создавать и выполнять может не только программист. А, например, тестировщик, менеджер или даже заказчик. Основное требование - это наличие браузера. И доступность приложения по указанному в конфигурации адресу.

Можно использовать как PHP-браузер, так и обычный браузер через [Selenium](http://www.seleniumhq.org/). Мы остановимся на PHP-браузере. Указываем об этом в файле tests/acceptance.suite.yml:

class\_name: AcceptanceTester

modules:

enabled: [PhpBrowser, \Helper\Acceptance]

config:

PhpBrowser:

url: 'http://localhost:8000'

Пересоздаем базовые классы тестов:

$ ./vendor/bin/codecept build

Этот модуль требует доступности приложения по адресу http://localhost:8000. Если это условие соблюдено можем приступать к тестам.

Структура и стиль написания идентичны функциональным тестам. Поэтому за пример возмем тот же тест страницы авторизации, который использовали при функциональном тестировании.

Последовательность действий такая же. Сначала генерируем тест:

$ ./vendor/bin/codecept generate:cept acceptance AuthPage

Потом заполняем его (это файл tests/acceptance/AuthPageCept.php):

$I = new AcceptanceTester($scenario);

$I->wantTo('Проверить авторизацию на сайте.');

$I->amOnPage('/');

$I->click('Авторизация');

$I->fillField('username','Admin');

$I->fillField('password','1q2w3e');

$I->click('Войти');

$I->see('Добро пожаловать, Admin', 'h1');

И запускам тесты:

$ ./vendor/bin/codecept run acceptance

При успешном выполнении увидим такой результат:

Acceptance Tests (1) ---------------------------------------------------

✔ AuthPageCept: Perform actions and see result (0.01s)

---------------------------------------------------

Time: 168 ms, Memory: 12.25MB

OK (1 test, 0 assertions)

## Настройка приемочных тестов

Каждый набор тестов имеет свой собственный стартовый (bootstrap) файл.Он расположен в директории, содержащей набор тестов и называется\_bootstrap.php. Этот файл исполняется перед каждым тестом из соответствующего набора. Любые подготовительные операции для набора тестов следует писать именно в этом файле.

Настройка глобального окружения tests/\_bootstrap.php

<?php

ini\_set('xdebug.max\_nesting\_level', 1000);

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

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

require\_once(\_\_DIR\_\_ . '/../../vendor/autoload.php');

require\_once(\_\_DIR\_\_ . '/../../vendor/yiisoft/yii2/Yii.php');

Yii::setAlias('@tests', dirname(\_\_DIR\_\_));

Нам нужно настроить файл конфига — tests/acceptance.suite.yml

После изменения конфига мы должны перестроить наш сodeception, делается это командой

codecept build

Глобальная конфигурация

actor: Engineer

paths:

tests: tests

log: tests/\_log

data: tests/\_data

helpers: tests/\_helpers

settings:

bootstrap: \_bootstrap.php

colors: true

memory\_limit: 1024M

log: true

modules:

config:

Db:

dsn: 'pgsql:host=localhost;dbname=DBNAME'

user: 'USER'

password: 'PASS'

dump: tests/\_data/database.dump.sql

populate: true

cleanup: true

##### Примечание

Для успешного запуска запуска *acceptance* тестов нужно указать базовый URL сайта в конфиге приемочных тестов *tests/acceptance.suite.yml*:

# Конфигурация приемочных тестов.

# Запуск тестов в браузере с помощью WebDriver или PhpBrowser.

# If you need both WebDriver and PHPBrowser tests - create a separate suite.

class\_name: WebEngineer

modules:

enabled:

- PhpBrowser

- WebHelper

- WebDriver

- Db

config:

PhpBrowser:

url: 'http://localhost:8000/'

WebDriver:

url: 'http://localhost:8000/'

browser: 'firefox'

wait: 2

**После этого примените конфигурацию**:

codecept build -c app/tests/codeception

### **Настройка**

Модули могут быть настроены в файле настроек тестового набора или глобально в codeception.yml.  
Обязательные параметры должны быть определены в $requiredFields свойстве класса модуля. Вот как это сделано в модуле Db.

<?php

class Db extends \Codeception\Module {

protected $requiredFields = array('dsn', 'user', 'password');

?>

Следующий раз вы когда запустите набор тестов без установки эти значений будет выброшено исключение.

Для необязательных параметров, вы должны установить значения по умолчанию. Свойство $config используется чтобы определить необязательные параметры, а так же их значения. В модуле Selenium мы используем адрес и порт сервера по умолчанию.

<?php

class Selenium extends \Codeception\Util\MinkJS

{

protected $requiredFields = array('browser', 'url');

protected $config = array('host' => '127.0.0.1', 'port' => '4444');

}

?>

Параметры адрес и порт могут быть переопределены в настройках набора тестов. Значения устанавливаются в секции modules:configфайла настроек.

modules:

enabled:

- Selenium

- Db

config:

Selenium:

url: 'http://mysite.com/'

browser: 'firefox'

Db:

cleanup: false

repopulate: false

Обязательные и необязательные параметры могут доступны через свойство $config. Используйте $this->config['parameter'] чтобы получить значение параметра.

### **Динамическая настройка**

начиная с версии 1.6.2

Если вы хотите переопределить настройки модуля во время исполнения, вы можете использовать \_reconfigure метод модуля.  
Вы можете вызвать его из класса помощника и передать в него все поля, которые хотите изменить.

<?php

$this->getModule('Selenium2')->\_reconfigure(array('browser' => 'chrome'));

?>

В конце теста все ваши изменения откатятся до значений из файла настроек.

## Парни

Основной концепцией Codeception является представление тестов как действий человека. Мы будем называть этого человека "**парнем**" (Guy). У нас есть **CodeGuy**, который выполняет функции/методы и тестирует код. Также у нас есть **TestGuy** - опытный тестировщик, который тестирует приложение целиком и знает кое-что о его внутреннем устройстве. Еще у нас есть **WebGuy**, пользователь, который работает с нашим приложением через интерфейс, который мы предоставляем.

Каждый из этих "парней" представляет из себя PHP класс с набором действий, который он может совершать. Как Каждый из этих "парней" имеет различные способности. Эти способности не постоянны, Вы можете расширять их. Вы даже можете создавать новых "парней", но запомните: один "парень" на один набор тестов.

Классы "парней" не пишутся, они генерируются следующей командой:

$ php codecept.phar build

## Напишем простой сценарий

По умолчанию тесты пишутся в виде последовательных сценариев. Чтобы PHP-файл стал корректным сценарием, его имя должно содержать суффикс "Cept".

Предположим, что мы создали файл tests/acceptance/SigninCept.php

Мы можем сделать это следующей командой:

$ php codecept.phar generate:cept acceptance Signin

<?php

$I = new WebGuy($scenario);

?>

Сценарий всегда начинается с инициализации класса нашего "парня". После этого написание сценария сводится к написанию $I-> и выбору соответствующего действия из списка авто-дополнения.

Давайте авторизуемся на нашем сайте. Мы предполагаем, что у нас есть страница "login", на которой для авторизации необходимо ввести логин и пароль. После авторизации мы перекидываем пользователя на страницу, где он видит текст Hello, %username%. Давайте разберемся как написать этот сценарий, используя Codeception.

<?php

$I = new WebGuy($scenario);

$I->wantTo('log in as regular user');

$I->amOnPage('/login');

$I->fillField('Username','davert');

$I->fillField('Password','qwerty');

$I->click('Login');

$I->see('Hello, davert');

?>

Прежде чем запустить тест, мы должны убедиться, что сайт работает на локальном веб-сервере. Откройте tests/acceptance.suite.yml и замените URL на URL Вашего приложения:

config:

PhpBrowser:

url: 'http://myappurl.local'

Если у Вас нет запущенного веб-сервера Вы можете использовать [PHP Built-in Web Server](http://php.net/manual/en/features.commandline.webserver.php) который доступен, начиная с версии PHP 5.4

После того как Вы установили корректный URL, Вы можете запустить тест, используя следующую команду:

$ php codecept.phar run

Вот, что мы должны увидеть на выходе:

Suite acceptance started

Trying log in as regular user (SigninCept.php) - Ok

Suite functional started

Suite unit started

Time: 1 second, Memory: 21.00Mb

OK (1 test, 1 assertions)

Давайте посмотрим более детальный вывод:

$ php codecept.phar run acceptance --steps

Мы должны увидеть шаг за шагом последовательность выполняемых действий.

Suite acceptance started

Trying to log in as regular user (SigninCept.php)

Scenario:

\* I am on page "/login"

\* I fill field "Username" "davert"

\* I fill field "Password" "qwerty"

\* I click "Login"

\* I see "Hello, davert"

OK

Time: 0 seconds, Memory: 21.00Mb

OK (1 test, 1 assertions)

Это очень простой сценарий, который Вы можете воспроизвести для своего собственного сайта. Эмулируя действия пользователя, Вы можете протестировать весь Ваш сайт подобным образом.

### Правила описания тестов

Каждый тест обязан содержать метод wantTo(), в котором вы описываете суть теста: "Я хочу выполнить такое-то действие, после чего увидеть такой-то результат".

$I = new WebEngineer($scenario);

$I->wantTo('see User list');

$I->amOnPage('/users');

// .. далее идут действия

### Методы тестирования

#### Вывод сообщений

Вывод отладочной информации при запуске тестов с параметром --debug:

codecept\_debug('текст или массив'); // ф-ция может быть вызвана в любом месте

$this->debug('текст или массив'); // метод debug доступен в Helper классах

Сообщение в консоль об ожидаемом результате:

$I->expect('что будет такой-то результат');

// \* I expect что будет такой-то результат

Вывод комментария, просто текст:

$I->comment('⚑ Проверка блока «Панель навигации»');

// \* ⚑ Проверка блока «Панель навигации»

#### Отправка запросов

Посетить страницу:

$I->amOnPage('/');

Отправить AJAX запрос:

$I->sendAjaxPostRequest('/path', ['param' => 'value']);

Отправить форму:

$I->submitForm('.css', [

'settings[value][]' => 'test'

]);

#### Манипуляция DOM элементами

$I->fillField('email', 'email'); // Заполнить поле

$I->appendField('#pass', 'pass'); // Добавить текст в поле

$I->checkOption('#css'); // Отметить чекбокс

$I->attachFile('input[type="file"]', 'file.xls');

$text = $I->grabTextFrom('#email'); // Получить текст элемента

#### События

Нажать на элемент (ссылку, кнопку):

$I->click('.css');

Нажать клавишу:

$I->pressKey('.css', WebDriverKeys::ENTER);

#### Методы проверки

Строгие и не строгие методы:

$I->see('text'); // Строгий метод проверки, наличие текста ОБЯЗАТЕЛЬНО

$I->canSee('text'); // Не останавливать проверку, если текст отсутствует

Методы проверки состояния:

$I->seeInTitle('title');

$I->seeCurrentUrlEquals('/path');

$I->seeEquals('text');

$I->seeContains('text');

$I->dontSee('text');

$I->seeLink('name');

$I->seeElement('.css');

$I->seeCheckboxIsChecked('#css');

$I->seeInField('field[name]', 'text');

#### Таймаут

Ждать 5 сек ИЛИ до выполнения условия (*until*):

$this->webDriver->wait(5)->until(function(RemoteWebDriver $web) {

$el = $web->findElement(WebDriverBy::className('expect-class'));

WebDriverExpectedCondition::visibilityOf($el);

});

Приостановить сценарий для отладки теста (работает только в *--debug* режиме):

$I->pauseExecution();

#### Сессии

Выполнить действия в отдельной сессии. Создание собственного экземпляра $I и параллельной сессии:

$I = new WebEngineer($scenario);

$nick = $I->haveFriend('nick');

// ..или

$I->does(function(WebGuy $nick) {

$nick->click('Milestones');

$nick->canSeeInTitle('Milestones');

});

#### Selenium

$I->executeInSelenium(function(RemoteWebDriver $web) {

$web->getKeyboard()->pressKey(WebDriverKeys::ENTER);

$driver->executeScript('alert('OK');');

$rows = $driver->findElements(WebDriverBy::cssSelector('table tr'));

});

#### WebHelper

namespace Codeception\Module;

use Codeception\Module;

use Codeception\TestCase;

use RemoteWebDriver;

use WebDriverBy;

class WebHelper extends Module

{

/\*\*

\* @var \RemoteWebDriver

\*/

private $webDriver;

public function \_before(TestCase $test)

{

$this->webDriver = $this->getWebDriver();

$this->webDriver->manage()->window()->setPosition(new \WebDriverPoint(320, 200));

}

private function getWebDriver()

{

/\*\* @var \Codeception\Module\WebDriver $webDriver \*/

$webDriver = $this->getModule('WebDriver');

return $webDriver->webDriver;

}

/\*\*

\* @param string $selector

\*

\* @return \WebDriverElement

\*/

public function findOne($selector)

{

$marker = 'marker-'.md5(uniqid(rand(), true));

$script = sprintf('$("%s").addClass("%s")', addslashes($selector), $marker);

$this->webDriver->executeScript($script);

return $this->webDriver->findElement(WebDriverBy::className($marker));

}

}

Воспользуемся генератором

cept generate:cept acceptance CheckTest

Для тестов можно генерировать саб классы, в которых будет описываться то или иное конкретное действие, как это работает вы поймете чуть дальше, а пока давайте сгенерируем один такой саб-класс

cept generate:stepobject acceptance **CheckUserSteps**

и он будет создан в папке tests/acceptance/\_steps.

Пример теста:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | $I = **new** \AcceptanceTester\CheckUserSteps($scenario);  $I->wantTo('see main');  $I->amInMainUi();  $I->wantTo('see news');  $I->amInNewsUi();  $I->amInOneNewsUi();  $I->wantTo('regoster');  $I->amInregisterUi();  $first\_customer = $I->imagineCustomer();  $I->fillCustomerDataForm($first\_customer);  $I->submitCustomerDataForm();  $I->seeIAmInList(); |

После запуска теста, фреймворк поочередно пойдет выполнять все эти методы, а вот их реализация как раз хранится в файликах steps которые я приведу ниже, в нашем случае  пример кода нашего CheckUserSteps

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | <?php  **namespace** AcceptanceTester;  **class** CheckUserSteps **extends** \AcceptanceTester  {  **public** **function** amInMainUi(){          $I = $this;          $I->amOnPage('/index.php'); // я на главное странице          $I->see('Hello all'); // я вижу текст Hello all        }  **public** **function** amInNewsUi(){          $I = $this;          $I->amOnPage('/news'); // я на странице новостей          $I->see('новые новости'); // я вижу текст "новые новости"        }  **public** **function** amInOneNewsUi() {        $I = $this;        $I->seeLink('details'); // я вижу ссылку details        $I->click(['link' => 'details']); // я кликаю по ссылке details        $I->see('One news'); // я вижу текст "One news"      }  **public** **function** amInAuthUi(){          $I = $this;          $I->amOnPage('/auth'); // я на странице авторизации          $I->see('авторизироваться'); // я вижу текст "авторизироваться"        }  **public** **function** imagineCustomer()   {  /\*тут нам понадобиться чудесный класс Faker который устанавливает фейковые данные,  установить его можно так : php composer.phar require "fzaninotto/faker:\*"</pre>  <pre>//тестирую на yii2 подключаю класс именно так,вы можете как вам угодно,  ваша задача создать объект класса faker \*/   $faker = \Faker\Factory::create();  **return** [   'CustomerRecord[name]' => $faker->name,//имя   'CustomerRecord[birth\_date]' => $faker->date('Y-m-d'),//дата   'CustomerRecord[notes]' => $faker->sentence(8),//текст   'PhoneRecord[number]' => $faker->phoneNumber//номер телефона   ];   }  **public** **function** fillCustomerDataForm($fieldsData)   {   $I = $this;  **foreach** ($fieldsData **as** $key => $value)   $I->fillField($key, $value); //заполняем поля формы name birth\_date..   }  **public** **function** submitCustomerDataForm()   {   $I = $this;   $I->click('Submit');//кликаем по кнопке сабмита   }  **public** **function** seeIAmInList()   {   $I = $this;   $I->seeCurrentUrlMatches('/users/');//проверяем попалили на нужную ли страницу   }  } |

# Модули и помощники

Codeception использует модульную архитектуру чтобы создать комфортное тестовое окружение для каждого написанного вами набора тестов.  
Модули позволяют выбрать действия (actions) и утверждения (assertions), которые будут выполнены в тестах.

Все действия и утверждения, которые будут выполнены "объектом-парнем" объявлены в модулях. Это может выглядеть будто Codeception ограничивает вас в тестировании, но это не правда. Вы можете расширить набор тестов своими собственными действиями и утверждениями, описав их в своем модуле.

Давайте посмотрим на этот тест.

<?php

$I = new TestGuy($scenario);

$I->amOnPage('/');

$I->see('Hello');

$I->seeInDatabase('users', array('id' => 1));

$I->seeFileFound('running.lock');

?>

Тест может оперировать различными сущностями: страница может быть загружена с помощью модуля Symfony1, база данных проверена с помощью модуля Db, а наличие файла может быть с помощью модуля Filesystem.

Модули привязан к классам "Парней" c помощью файла настроек тестового набора.  
Например, в tests/functional.suite.yml мы можем увидеть.

class\_name: TestGuy

modules:

enabled: [Symfony1, Db, Filesystem]

У класса TestGuy есть методы объявленные в модулях. На самом деле, класс не содержит все эти методы, но он выступает в качестве прокси для них. Он знает какой модуль выполняет данное действие и передает ему параметры. Чтобы ваша IDE могла видеть все методы класса TestGuy используйте команду build. Команда генерирует определение класса TestGuy, копируя сигнатуры методов из настроенных модулей.

## Стандартные модули

У Codeception в комплекте есть много модулей, которые могут помочь вам запускать тесты для разных целей и в разных тестовых окружениях. Количество модулей не постоянное -- оно должно расти с увеличением поддерживаемых фреймворков и ORM.  
Весь список можно увидеть в сайдбаре правой части страницы.

Все модуи документированы. Вы можете ознакомиться с их детальным описанием на странице [GitHub](https://github.com/DavertMik/Codeception/tree/master/docs/modules).

## Помощники

Codeception не ограничивает вас в использовании модулей только из основного репозитория. Несомненно, в вашем проекте вы можете захотеть добавить собственные действия в тестовые наборы. Запустив команду bootstrap, Codeception сгенерирует для вас три заготовленных модуля, по одному для каждого набора. Эти модули называются 'Помощниками', их можно найти по следующему путиtests/helpers.

Это хорошая идея определять недостающие действия или команды утверждения (assertion) в помощниках.

Скажем, мы собираемся расширить класс TestHelper. По умолчанию он связан с классом TestGuy и набором функциональных тестов.

Что касается действий, все очень просто. Каждое действие, которое вы определяете является публичной функцией. Опишите публичную функцию, запустите команду build и вы увидите, что добавлена новая функция в класс TestGuy. Примечание: публичные методы с префиксом \_ рассматриваются как скрытые и не будут добавлены в класс вашего "Парня".

С утверждениями немного сложнее. В первую очередь, рекомендуется добавлять префикс see или donSee для всех утверждающих действий. В философии Codeception, все тесты выполняются людьми, например "парнями". Ожидаемый результат, который они видят (или не видят) это то, что мы используем для утверждения.

Называйте ваши утверждения подобным образом:

<?php

seePageReloaded();

seeClassIsLoaded($classname);

dontSeeUserExist($user);

?>

И затем используйте их в своих тестах:

<?php

$I = new TestGuy($scenario);

$I->seePageReloaded();

$I->seeClassIsLoaded('TestGuy');

$I->dontSeeUserExist($user);

?>

Каждая see или dontSee функция требует по крайней мере одно утверждение. Codeception использует утверждения из PHPUnit.

Вы можете объявлять утверждения используя asertXXX методы модуля.  
Codeception использует утверждения из PHPUnit. Таким образом, в случае когда вам не хватает некоторых утверждений, вы можете использовать статичные методы PHPUnit из класса PHPUnit\_Framework\_Assert.

<?php

function seeClassExist($class)

{

$this->assertTrue(class\_exists($class));

// или

\PHPUnit\_Framework\_Assert::assertTrue(class\_exists($class));

}

?>

В ваших помощниках вы можете использовать эти утверждения:

<?php

function seeCanCheckEverything($thing)

{

$this->assertTrue(isset($thing), "this thing is set");

$this->assertFalse(empty($any), "this thing is not empty");

$this->assertNotNull($thing, "this thing is not null");

$this->assertContains("world", $thing, "this thing contains 'world'");

$this->assertNotContains("bye", $thing, "this thing doesn`t contain 'bye'");

$this->assertEquals("hello world", $thing, "this thing is 'Hello world'!");

}

?>

Просто введите $this->assert чтобы увидеть их все.

Так же, у каждого модуля есть специальные методы $this->assert и $this->assertNot. Оба метода принимают одинаковые параметры и полезны если вам нужно объявить как положительное так и отрицательное утверждение в вашем модуле. Эти функции принимают массив как параметр, где первое значение массива это название функции утверждения из PHPUnit.

<?php

$this->assert(array('Equals',$int,3));

$this->assertNot(array('internalType',$int,'bool'));

$this->assert(array('Contains', array(3,5,9), 3));

?>

Давайте посмотрим как определить оба see и dontSee действия без дублирования кода.

assert($this->proceedSeeClassExist($class));

}

public function dontSeeClassExist($class)

{

$this->assertNot($this->proceedSeeClassExist($class));

}

protected function proceedSeeClassExist($class)

{

return array('True',get\_class($class));

}

?>

Для dontSeeClassExist, будет вызван assertFalse.

### **Разрешение конфликтов**

Что случится если у вас есть два модуля, которые содержат действия с одинаковыми названиями?  
Codeception позволяет вам переопределять действия меняя порядок модулей.  
Действие из второго модуля будет загружено, а действие из первого модуля будет проигнорировано.  
Порядок модулей описывается в файле настроек тестового набора.

### **Взаимодействие Модулей**

Возможно, что вы захотите получить доступ к внутренним свойствам или функциям из другого модуля. Например, для вашего модуля понадобилось соединение из модуля Doctrine или браузер из модуля Symfony.

Модули могут взаимодействовать с другими модулями через метод getModule. Пожалуйста, обратите внимание, что этот метод может выбросить исключение если требуемый модуль не был загружен.

Давайте представим, что мы пишем модуль, который переподключается к базе данных. Предположим он использует значение свойства dbhиз модуля Db.

<?php

function reconnectToDatabase() {

$dbh = $this->getModule('Db')->dbh;

$dbh->close();

$dbh->open();

}

?>

Используя функцию getModule вы получаете доступ ко всем публичным методам и свойствам запрошенного модуля.  
Свойство dbh было специально объявлено как публичное чтобы быть доступным для других модулей.  
Эта техника так может быть полезна если вам нужно выполнить последовательность действий взятых из других модулей.  
Например:

<?php

function seeConfigFilesCreated()

{

$filesystem = $this->getModule('Filesystem');

$filesystem->seeFileFound('codeception.yml');

$filesystem->openFile('codeception.yml');

$filesystem->seeInFile('paths');

}

?>

### **Неопределенные действия в помощниках**

В случаи когда у вас есть действие, которое еще не определено, вы можете автоматически создать метод-заглушку для него в соответствующем помощнике. Чтобы это сделать, вы можете использовать команду analyse, которая сканирует все тесты и ищет несуществующие действия в любом подключенном модуле.

Таким образом, вы можете доверить написание тестов технически не подкованным людям или специалистам по качеству. В случае если им будет недоставать некоторых действий они объявят их в тесте.

<?php

$I->doManyCoolThings();

?>

Запустив команду analyze у вас спросят, хотите ли вы добавить doManyCoolThings в текущий Помощник.

### **Хуки**

Каждый модуль может обрабатывать события из запущенного теста. Модуль может быть выполнен до начала теста или после его завершения. Это может быть полезно для действий начальной загрузки или завершающих действий.  
Вы так же можете определить специальное поведение для ситуации когда тест провален. Это может быть полезно отладке.  
Например, модуль PhpBrowser сохраняет текущую страницу в директорию log если тест провален.

Все хуки определены в \Codeception\Module и перечислены здесь. Вы вольны переопределить их в своем модуле.

Пожалуйста, обратите внимание методы с префиксом \_ не добавлены в класс "Парня". Это позволяет объявлять публичные методы, но использовать только для внутренних целей.

### **Отладка**

Как мы уже упоминали, хук \_failed может помочь в отладке проваленного теста. У вас есть возможность сохранить текущее состояние тестов и показать их пользователю.

Вы не ограничены этим. Каждый модуль может выводить внутренние значения, которые могут быть полезны при отладке.  
Например, модуль PhpBrowser выводит http код ответа и текущий адрес каждый раз когда он переходит на новую страницу.  
Таким образом, модули не являются "черными ящиками". Они пытаются показать вам, что происходит в время теста. Это делает отладку ваших тестов менее болезненной.

Чтобы вывести дополнительную информацию, используйте debug и debugSection методы модуля.  
Вот пример того, как это работает для PhpBrowser:

<?php

$this->debug('Request ('.$method.'): '.$uri.' '. json\_encode($params));

$browser->request($method, $uri, $params);

$this->debug('Response code: '.$this->session->getStatusCode());

?>

Этот тест запущенный с помощью модуля PhpBrowser в режиме отладке, выведет что-то похожее на это:

I click "All pages"

\* Request (GET) http://localhost/pages {}

\* Response code: 200

В статье "Установка, настройка и базовое использование фреймворка для тестирования "Codeception" ([ссылка](http://klisl.com/codeception_installation.html)) я уже приводил простой пример создания приемочных тестов. Теперь углубимся в теме приемочного тестирования, так же я напишу как использовать модуль **Selenium**, который позволяет протестировать любые действия посетителя сайта осуществляемые в т.ч. с помощью **Java Scripts**.  
  
Суть приемочных тестов в проверке того, что доступно любому человеку: просмотр содержимого страницы, попытка залогиниться и т.д.  
Как я уже писал, существует два способа написания тестов в Codeception, которым соответствуют два типа наименования файлов: **Cept** и **Cest**. Рассмотрим их.  
  
**1. Пример создания приемочных тестов согласно функционального программирования (Cept).**  
  
Продублирую небольшой пример из первой статьи.  
Файл теста можно сгенерировать автоматически. Создадим файл **HomeCept.php** командой:

codecept generate:cept acceptance Home

Файл появится в папке **tests/acceptance** и сразу будет содержать строки:

<?php

$I = new AcceptanceTester($scenario);

$I->wantTo('perform actions and see result');

При желании, создать файл теста можно и вручную, главное указать в названии файла окончание «**Cept**».  
  
Создадим тест на проверку доступа к главной странице которая также должна вернуть указанный текст (фразу).

<?php

$I = new AcceptanceTester($scenario);

$I->wantTo('Проверить работу главной страницы.');

$I->amOnPage('/');

$I->seeResponseCodeIs(\Codeception\Util\HttpCode::OK);

$I->see('Главная страница'); // Фраза или ее часть с главной страницы

стоит учесть, что метод **seeResponseCodeIs()** будет работать при указании в конфигурационном файле модуля **PHPBrowser** и не будет работать с **WebDriver**.   
  
  
**2. Пример создания приемочных тестов согласно ООП (Cest).**  
  
Файл теста так же можно сгенерировать автоматически. Создадим файл **HomeCest.php**:

codecept generate:cest acceptance Home

Файл появится в папке **tests/acceptance** и сразу будет содержать указанные строки:

<?php

class HomeCest

{

public function \_before(AcceptanceTester $I)

{

}

public function \_after(AcceptanceTester $I)

{

}

// tests

public function tryToTest(AcceptanceTester $I)

{

}

}

Перепишем пример согласно **ООП**, для этого нужно изменить только метод **tryToTest()**:

public function tryToTest(AcceptanceTester $I)

{

$I->wantTo('Проверить работу главной страницы.');

$I->amOnPage('/');

$I->seeResponseCodeIs(\Codeception\Util\HttpCode::OK);

$I->see('Главная страница'); // Фраза или ее часть с главной страницы

}

Вы можете указывать свои произвольные названия методов тестов, но помните, что методы имеющие перед название нижнее подчеркивание не будут выполнены в качестве тестов.   
Так же можете определить метод **\_failed()** в классах **Cest**, который будет вызван в случае, если в тесте произошла ошибка error или он 'провалился'.

##### Запуск тестов.

Для запуска приемочных тестов отдельно от других типов тестов перейти в консоли в корень проекта или в каталог куда были установлены тесты (если установка была не в корень) и выполнить:

codecept run acceptance

Согласно данного теста, главная страница сайта должна вернуть заголовок со статусом 200 и контент должен содержать фразу 'Главная страница'.

#### Следующий пример.

Усложним задачу. Допустим на нашем сайте есть страница контактов и нужно протестировать ее работу, а именно – перейти на нее используя ссылку в меню, заполнить форму, отправить, получить подтверждение об успешной отправке.

<?php

$I = new AcceptanceTester($scenario);

$I->wantTo('go to the contact page and send the form');

$I->amOnPage('/'); //указываем текущую страницу (главная)

$I->click('.menu li a:contains("Контакты")'); //кликаем для перехода на страницу контактов

$I->see('ЗАДАВАЙТЕ ВОПРОСЫ', '#content'); //должна появиться указанная надпись в блоке с id "content"

//Заполняем и сразу отправляем форму с id "contactForm"

$I->submitForm("", ['name'=>'Serj', 'email'=>'serj@ukr.net', 'text'=>'Тестовое сообщение']);

//должна появиться указанная надпись в блоке с class "alert-success"

$I->see('Ваше сообщение отправлено.', '.alert-success');

Стоит отметить, что выборка элементов стандартным для **JQuery** образом, например:

$I->click('.menu li a:contains("Контакты")');

будет работать при использовании **PhpBrowser** и не всегда будет работать с **WebDriver** (при использовании **Selenium**). Для WebDriver поиск элемента по его содержимому делается с использованием класса **Locator** и нужно писать так:

$I->click(Locator::contains('.menu li a', 'Контакты'));

Кроме того есть ограничения и для использования некоторых методов. Так, нельзя использовать метод **submitForm()**.  
Поэтому перепишем данный пример сделав его универсальным:

<?php

use Codeception\Util\Locator;

$I = new AcceptanceTester($scenario);

$I->wantTo('go to the contact page and send the form');

$I->amOnPage('/'); //указываем текущую страницу

$I->click(Locator::contains('.menu li a', 'Контакты')); //кликаем для перехода на страницу контактов

$I->see('ЗАДАВАЙТЕ ВОПРОСЫ', '#content'); //должна появиться указанная надпись в блоке с id "content"

$I->fillField('name','Serj');

$I->fillField('email','serj@ukr.net');

$I->fillField('text','Тестовое сообщение');

$I->click(Locator::contains('#contactForm button', 'Отправить'));

//должна появиться указанная надпись в блоке с class "alert-success"

$I->see('Ваше сообщение отправлено.', '.alert-success');

Запустить тесты можно так же с выводом всех шагов их выполнения:

codecept run acceptance --steps

Для последнего примера вывод в консоль будет такой:

Codeception PHP Testing Framework v2.3.6

Powered by PHPUnit 5.7.23 by Sebastian Bergmann and contributors.

Acceptance Tests (1) -----------------------------------------------

ContactsCept: Perform actions and see result

Signature: ContactsCept

Test: tests\acceptance\ContactsCept.php

Scenario --

I am on page "/"

I click ".menu li a:contains("Контакты")"

I see "ЗАДАВАЙТЕ ВОПРОСЫ","#content"

I fill field "name","Serj"

I fill field "email","serj@ukr.net"

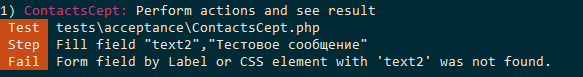
I fill field "text","Тестовое сообщение"

I click "#contactForm button:contains("Отправить")"

I see "Ваше сообщение отправлено.",".alert-success"

PASSED

--------------------------------------------------------------------

Если тест завершится с ошибкой, то удобно смотреть в каком тесте и на какой строке она возникла. Например вместо названия поля “text” я по-ошибке впишу “text2” и получим такую ошибку:  
  
  
Указана причина ошибки: «Поле формы с помощью элемента Label или CSS с «text2» не было найдено».

#### Грабберы.

Грабберы это методы, которые используются для получения значений элементов и полей страницы, что может быть необходимо для использования в тесте.  
Основные из них:  
- получение текстового содержимого элемента:

$token = $I->grabTextFrom('.token');

- получение значения поля:

$api\_key = $I->grabValueFrom('input[name=api]');

Это может понадобиться, например, для ввода только что сгенерированного пароля в поле формы:

<?php

$I->fillField('email','miles@davis.com')

$I->click('Generate Password');

$password = $I->grabTextFrom('#password');

$I->click('Login');

$I->fillField('email','miles@davis.com');

$I->fillField('password', $password);

$I->click('Log in!');

?>

#### SELENIUM.

**PhpBrowser** не позволяет полноценно протестировать страницу на которой используется **JavaScript**, т.к. данный модуль использует библиотеку Curl. Для таких целей подключают модуль WebDriver, который использует **Selenium**.   
Selenium — это проект, предоставляющий драйвера для разных браузеров, которые встраиваются в них и позволяют управлять ими. Также, Selenium содержит сервер, который позволяет управлять большим числом разных браузеров и распределять задания между ними. Selenium сервер написан на Яве, потому она понадобится чтобы его запустить.  
  
При использовании Selenium для тестов будет запускаться браузер и вы увидите, как происходит заполнение полей форм, их отправка на сервер, клики по элементам и прочие действия, которые обычно совершаются пользователем приложения.  
  
В примерах я буду использовать браузер **Firefox**, поэтому в начале указываю как настроить их взаимодействие с Selenium. Возможно так же использование браузера Google Chrome, пример настройки в конце статьи.  
  
Есть несколько способов установки Selenium.

##### 1. Самый простой и удобный с помощью Composer.

Страница расширения <https://github.com/sveneisenschmidt/selenium-server-standalone>   
Выполнив команду:

composer global require --dev se/selenium-server-standalone

вы установите сервер **Selenium** глобально в папку текущего пользователя, у меня это C:\Users\Сергей\AppData\Roaming\Composer\vendor  
  
Или с указанием конкретной версии:

composer global require --dev se/selenium-server-standalone 2.53.1

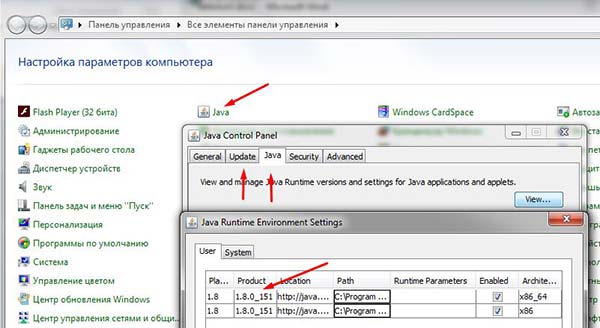
Тут я указал версию 2.53.1, которая будет работать с версией **Java**, установленной сейчас на одном из моих компьютеров (обновить ее я не могу). Можно устанавливать без указания версии, в таком случае установится новейшая версия.   
  
Стоит отметить что для использования данной версии Seleniumа (2.53.1), браузер Firefox должен быть ниже версии 48, например 47.0.1  
  
Если вы устанавливаете новую версию **Seleniumа** (>=3), то потребуется и новая версия Firefox и что еще важно потребуется дополнительно загрузить **geckodriver**. Подробнее про него тут: [http://barancev.github.io/geckodriver](http://barancev.github.io/geckodriver/)  
Скачиваем данный драйвер <https://github.com/mozilla/geckodriver/releases> для своей операционной системы и сохраняем его в каталог указанный в системной переменной **PATH**. Я сохранил в D:\OpenServer 5.2.7\modules т.к. эта папка указана в списке. Или указываем нужный путь к каталогу с драйвером в переменной системной переменной PATH.  
  
  
Запуск сервера Selenium осуществляется командой в консоли:

selenium-server-standalone

Если вы запускаете из Windows и у вас возникает такая ошибка:  
/usr/bin/bash: "C:\Users\...\AppData\Roaming\Composer\vendor\bin\/../se/selenium-server-standalone/bin/selenium-server-standalone": No such file or directory  
то нужно подкорректировать файл C:\Users\{USERNAME}\AppData\Roaming\Composer\vendor\bin\selenium-server-standalone.bat (тут показан путь при глобальной установке Selenium), заменить 2 последние строки на

SET BIN\_TARGET=%~dp0/../se/selenium-server-standalone/bin/selenium-server-standalone.jar

java -jar "%BIN\_TARGET%" %\*

т.к. прописанная там по-умолчанию команда **sh** предназначена для выполнения на **Unix** подобных системах.  
  
А если при выполнении команды возникнет ошибка такого плана:  
Exception in thread "main" java.lang.UnsupportedClassVersionError: org/openqa/grid/selenium/GridLauncherV3 : Unsupported major.minor version 52.0…  
значит **Selenium** не может работать с установленной версией **Java** и нужно ее обновить или установить другую версию Selenium. В **Win-7** для просмотра версии или обновления Java можно открыть в «Панели управления»:   
  
  
Если все равно будут ошибки при запуске Selenium установите версию пониже.  
  
Так же, при экспериментировании с версиями, уже при запуске тестов возможна ошибка такого плана:   
[Facebook\WebDriver\Exception\UnknownServerException] Unable to connect to host 127.0.0.1 on port 7055 after 45000 ms.…  
Для устранения данной ошибки нужно обновить (или подобрать) версию Firefox с которой работает скаченный модуль Selenium. Например с selenium-server-standalone-2.53.1.jar будет работать с Firefox 47.0.1

##### 2. Скачиваем и запускаем Selenium непосредственно.

Переходим на http://www.seleniumhq.org/download/ и нажимаем по ссылке после слов «Download version».  
Для того чтобы скачать предыдущие версии Selenium переходим по ссылке http://selenium-release.storage.googleapis.com/index.html и там скачиваем файл такого типа: selenium-server-standalone-X.X.X.jar  
  
Далее, для использования **Firefox** скачиваем драйвер <https://github.com/mozilla/geckodriver/releases> для своей операционной системы и сохраняем его в каталог указанный в системной переменной **PATH**. Я сохранил в D:\OpenServer 5.2.7\modules, эта папка указана у меня в PATH. Вы можете указать нужный путь к каталогу с драйвером в переменной PATH. В эту же папку я переместил и скачанный модуль Selenium.  
  
Установка описана в комментариях файла модуля - vendor\codeception\codeception\src\Codeception\Module\WebDriver.php  
  
Перед выполнением тестирования должен быть запущен сервер. Запускаем командой:

java -jar -Dwebdriver.gecko.driver="D:\OpenServer 5.2.7\modules\geckodriver.exe" "D:\OpenServer 5.2.7\modules\selenium-server-standalone-3.4.0.jar"

где нужно указать свои пути к скачанным файлам и в конце изменить версию **Selenium**.  
Тут я указал путь к **geckodriver.exe**, но если он у вас лежит в папке которая указана в системной переменной PATH, то его можно не указывать. Получится:

java -jar "D:\OpenServer 5.2.7\modules\selenium-server-standalone-3.4.0.jar"

Так же вы можете создать файл selenium-server-standalone.bat в папке vendor\bin с указание своих путей к исполняемым файлам для удобства запуска модуля одной короткой строкой по примеру установки с помощью Composer.  
  
Если возникнет ошибка примерно с таким содержанием:  
[Facebook\WebDriver\Exception\SessionNotCreatedException] Unable to create new remote session. desired capabilities = Capabilities [{firefoxOptions=org.openqa.selenium.firefox.FirefoxOptions@7a99c504, browserName=firefox, moz:firefoxOptions=org.openqa.selenium.firefox.FirefoxOptions@7a99c504}], required capabilities = Capabilities [{}]Build info: version: '3.0.0', revision: '350cf60', time: '2016-10-13 10:48:16 -0700'System info: host: 'MICROSOFT-PC', ip: '192.168.56.1', os.name: 'Windows 7', os.arch: 'amd64', os.version: '6.1', java.version: '1.8.0\_40'Driver info: driver.version: FirefoxDriver  
значит запускаемый файл Selenium не соответствует установленной версии Java и скорее всего придется попробовать Selenium более старой версии.  
  
См. так же ошибки описанные в пункте установки с помощью Composer.  
  
У меня работала такая связка:  
Java 1.8.0\_40 (1.8.0\_51)selenium-server-standalone-3.4.0.jarи Firefox 56.0.2  
хотя текущая на момент написания статьи была версия selenium-server-standalone-3.6.0.jar

##### Настройка модуля **WebDriver** (Selenium).

В файле tests/acceptance.suite.yml

actor: AcceptanceTester

modules:

enabled:

- WebDriver:

url: 'http://site.loc/'

browser: 'firefox'

тут:  
- включаем модуль **WebDriver**;  
- в “url” указать стартовый URL вашего сайта;  
- в browser указываем браузер с которым будем работать 'chrome' или 'firefox'. Мы пишем 'firefox'.  
  
**Возможные настройки модуля WebDriver**:

* **url** - Стартовый URL для вашего приложения.
* **browser** - Браузер для запуска.
* **host** - хост сервера Selenium (по умолчанию 127.0.0.1).
* **port** - порт сервера Selenium (по умолчанию 4444).
* **restart** - установите в «false» (по умолчанию) для использования одного и того же окна браузера для всех тестов или установите «true» для создания нового окна для каждого теста. В любом случае, когда все тесты завершены, окно браузера закрывается.
* **start** - автозапуск браузера для тестирования. Может быть отключен, если сеанс браузера начинается с «\_initializeSession» внутри помощника.
* **window\_size** - Начальный размер окна. Установите значение «maximize» или пропишите в формате «640x480».
* **clear\_cookies** - Установите значение «false», чтобы сохранить файлы cookie, или установите значение «true» (по умолчанию), чтобы удалить все куки между тестами.
* **wait** (по умолчанию: 0 секунд) - всякий раз, когда элемент требуется и не находится на странице, подождите несколько секунд, чтобы найти его перед сбоем.
* **capability** - Настраивает Selenium [желаемые возможности] (https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities). Должен быть массив значений ключа.
* **connection\_timeout** - тайм-аут для открытия соединения с удаленным сервером (по умолчанию 30 секунд).
* **request\_timeout** - тайм-аут для запроса вернуть что-то из удаленного сервера selenium (по умолчанию 30 секунд).
* **pageload\_timeout** - время ожидания загрузки страницы до того, как будет выдана ошибка (по умолчанию 0 секунд).
* **http\_proxy** - устанавливает URL-адрес прокси-сервера HTTP для тестирования удаленного сервера.
* **http\_proxy\_port** - устанавливает порт HTTP-прокси-сервера
* **debug\_log\_entries** - сколько записей Selenium для печати с «debugWebDriverLogs» или при сбое (по умолчанию 15).
* **log\_js\_errors** - установите значение «true», чтобы включить возможный отчет JavaScript в HTML, или установите значение «false» (по умолчанию) для деактивации.

Подробнее можно посмотреть в блоке ## Configuration файла vendor\codeception\codeception\src\Codeception\Module\WebDriver.php  
  
При использовании Selenium, метод **see()** позволяет не просто убедится, что текст внутри элемента существует, но также проверит, что элемент действительно виден пользователю.

<?php

$I->see('Confirm','#modal');

?>

Например на странице используется «Bootstrap Accordion» т.е. выводятся в ряд заголовки, кликнув по которым открывается блок соответствующего им текста. До клика по нужному заголовку, блок принадлежащего ему текста скрыт. При использовании **PhpBrowser** проверка текста из скрытого блока на то, что он в данное время не видим пользователю:

$I->dontSee('скрытый текст');

завершится с ошибкой, т.к. PhpBrowser не умеет работать с скрытым текстом и он будет считаться видимым. А вот **Selenium** с этим справится.

##### Пример.

Для примера я использовал сайт со страницей на которой имеются скрытые блоки текста:

<?php

$I = new AcceptanceTester($scenario);

$I->wantTo('go to Services page and click first block');

$I->amOnPage('/'); //начинаем с главной страницы

$I->click(\Codeception\Util\Locator::contains('.menu li a', 'Услуги')); //кликаем для перехода на страницу услуг.

$I->see('ПЕРЕЧЕНЬ УСЛУГ', '#content'); //должна появиться указанная надпись в блоке с id "content"

//Проверяем, что блок невидим на странице

$I->dontSee('Создание сайтов любой сложности');

$I->click(\Codeception\Util\Locator::contains('.panel-title a', ' Создание сайтов/приложений')); //кликаем для вывода содержимого блока на экран

//Проверяем, что блок стал видимым

$I->see('Создание сайтов любой сложности');

По коду видно, что после перехода на страницу "Услуги" проверяется отсутствие надписи 'Создание сайтов любой сложности' из скрытого блока.  
Далее, для показа содержимого скрытого блока, пользователь (а в нашем случае Selenium) кликает по соответствующей ссылке и мы проверяем, что текст из скрытого блока, а вернее его часть 'Создание сайтов любой сложности' стала видима.

##### Запуск тестов в браузере Chrome.

Для запуска тестов в браузере **Chrome**, так же как и при использовании Firefox, необходимо использовать специальный драйвер. Качаем архив по этой [ссылке](https://sites.google.com/a/chromium.org/chromedriver/downloads).  
Под номером релиза указаны версии браузера, которые он поддерживает. Поэтому проверяем свою версию или просто обновляем браузер и скачиваем новейший релиз драйвера.   
Из архива извлекаем файл соответствующий вашей операционной системе и указываем путь к каталогу с драйвером (куда вы его поместили) в переменной окружения **PATH** так же как и при использовании Firefox.   
  
В конфигурационном файле **tests/acceptance.suite.yml** указываем url и браузер:

actor: AcceptanceTester

modules:

enabled:

- WebDriver:

url: 'http://site.loc/'

browser: 'chrome'

Далее запускаем Selenium и после этого выполнение тестов.

#### Создание отдельного класса для работы со страницей.

Функционал **Codeception** позволяет вынести действия по взаимодействию с определенной страницей в отдельный класс. Это бывает удобно, т.к. во-первых данный класс могут в дальнейшем использовать несколько отдельных тестов, кроме того сами тесты становятся более читаемыми.   
  
**Пример.**  
Допустим нам нужно создать класс, который бы выполнял переход на страницу входа пользователей и логинился (вводил регистрационные данные). А дальше, например, скрипт переадресует пользователя на страницу текущего пользователя, которую и нужно протестировать.   
Т.е. тестировать нам нужно страницу авторизованного пользователя, но сначала нужно «залогиниться», т.е. приходится взаимодействовать с страницей входа, которая нас не интересует. Вот ее и вынесем в отдельный класс.  
  
Создадим класс для работы со страницей “login” (http:site.com/login).

codecept generate:pageobject LoginPage

Появится файл tests/\_support/Page/LoginPage.php. Нам потребуется такой функционал:

<?php

namespace Page;

class LoginPage

{

// include url of current page

public static $URL = 'login';

protected $actor;

public static function route($param)

{

return static::$URL.$param;

}

public function \_\_construct($I)

{

$this->actor = $I;

}

public function open()

{

$I = $this->actor;

$I->amOnPage(self::$URL);

}

public function enter($login, $password)

{

$I = $this->actor;

$I->fillField('#loginform-username', $login);

$I->fillField('#loginform-password', $password);

$I->click('button[name=login-button]');

}

}

Теперь в тесте той страницы, к которой есть доступ только у зарегистрированных пользователей, можно обращаться к данному классу так:

public function registeredUser (AcceptanceTester $I)

{

$I->wantTo('проверить страницу зарегистрированного пользователя');

$page = new \Page\LoginPage($I);

$page ->open();

$I->see('Вход');

$page->enter('MySecretName', '1111111111');

$I->see('Личный кабинет');

...

При желании, можно вообще вместо

$page = new \Page\LoginPage($I);

$page ->open();

$I->see('Вход');

$page->enter('MySecretName', '1111111111');

сделать все одним вызовом в тесте:

\Page\LoginPage::open($I);

чтобы все действия по авторизации пользователя находились в классе **LoginPage**. Тогда код данного класса (файл tests/\_support/Page/LoginPage.php) будет такой:

<?php

namespace Page;

class LoginPage

{

// include url of current page

public static $URL = 'login';

protected $actor;

public static $login = 'MySecretName';

public static $password = '111111111';

public static function route($param)

{

return static::$URL.$param;

}

public function \_\_construct($I)

{

$this->actor = $I;

}

public static function open($I)

{

$page = new static($I);

$I->amOnPage(self::$URL);

$page->enter(self::$login, self::$password);

}

public function enter($login, $password)

{

$I = $this->actor;

$I->fillField('#loginform-username', $login);

$I->fillField('#loginform-password', $password);

$I->click('button[name=login-button]');

}

}

Статичный интерфейс метода **open()** служит для удоства, чтобы повесить создание объекта класса **LoginPage** на его же метод. Мы передаем конструктору данного класса переменную **I**, являющуюся экземпляром класса **AcceptanceTester** для того, чтобы мы могли выполнить действия такие же как в самом тесте (переходы по страницам, заполнение полей и тд.)

#### Дополнения.

Справка по методам приемочного тестирования с примерами находится в файле vendor\codeception\codeception\docs\03-AcceptanceTests.md  
Вот некоторые:

// отправить AJAX запрос:

$I->sendAjaxPostRequest('/path', ['param' => 'value']);

// проверка на то, что текущий URL «/user/view»:

$I->seeInCurrentUrl('/user/view');

// проверка существования указанной ссылки. В качестве первого параметру методу передается текстовое содержимое ссылки, в качестве второго можно передать URL для уточнения:

$I->seeLink('Услуги');

$I->seeLink('Услуги','/services');

// проверяем отсутствие текста на странице

$I->dontSee('Form is filled incorrectly');

// проверка на включенный чекбокс:

$I->seeCheckboxIsChecked('#agree');

// наличие указанного текста в поле формы:

$I->seeInField('user[name]','Miles');

//проверяет, что заголовок страницы содержит заданную строку.

$I->seeInTitle('Login');

// проверяет, что заголовок страницы не содержит заданную строку.

$I->dontSeeInTitle('Control panel');

// добавить текст в поле

$I->appendField('#pass', 'pass');

// нажать клавишу

$I->pressKey('.css', WebDriverKeys::ENTER);

**Пример тестирования файловой системы:**

$I->amInPath('tests/data/sandbox');

$I->executeCommand('run order --no-exit');

$I->seeFileFound('order.txt', 'tests/\_log');

$I->seeInThisFile("IBSBSBS([BST][BSTF][BST])");

все методы для работы с файловой системой с примерами можно посмотреть в файле vendor\codeception\codeception\tests\data\included\shire\tests\functional\TestGuy.php  
  
  
**Использование полезных функций (класс Locator) для работы с выборкой элементов со страницы.**  
Данные методы можно посмотреть в файле vendor\codeception\codeception\src\Codeception\Util\Locator.php, тут укажу только самые полезные, на мой взгляд.  
Подключаем класс **Locator** в начале тестирующего класса:

use Codeception\Util\Locator;

Отбираем элемент для клика по нему

* по произвольному атрибуту элемента:

$I->click(Locator::find('a', ['href' => 'http://new-yoursite.loc/contacts']));

* отдельно для ссылок:

$I->click(Locator::href('http://new-yoursite.loc/contacts'));

* отбор по содержимому элемента:

$I->click(Locator::contains('.menu li a', 'Контакты'));  
  
Проверка существования указанного селектора на странице:  
Locator::isCSS('#user .hello')

##### Пример использования.

Создадим тест который будет проверять сохранение данных из формы заполненной пользователем в базу данных.  
Допустим форма имеет такой вид:

<div class="user-form">

<form action="" method="POST">

<input type="name" name="name">

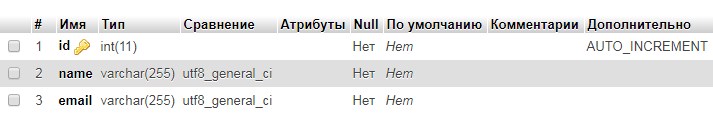
<input type="email" name="email">

<input type="hidden" name="form" value='true'>

<input type="submit">

</form>

</div>

При успехе пользователю выводится сообщение на странице 'Данные сохранены.'  
  
Для теста нам потребуется дамп базы данных которая должна называться "test" (так мы указали в файле конфигурации) и содержать пустую таблицу "users" с такой структурой:  
  
  
Создаем дамп базы (в phpmyadmin это закладка "Экспорт") и сохраняем в **tests/\_data**.  
  
Теперь создаем файл теста:

codecept generate:cest acceptance UserFormCest

Открываем его в папке **tests\acceptance** и заполняем:

<?php

class UserFormCest

{

public function \_before(AcceptanceTester $I)

{

}

public function \_after(AcceptanceTester $I)

{

}

public function formFillSave(AcceptanceTester $I)

{

$I->wantTo('Заполняю данные пользователя в форму и сохраняю в БД');

$I->amOnPage('/');

$I->fillField('name','Иванов');

$I->fillField('email','ivanov@gmail.com');

$I->click('.user-form input[type=submit]');

$I->see('Данные сохранены.');

$I->seeInDatabase('users', [

'name' => 'Иванов',

'email' => 'ivanov@gmail.com',

]);

}

}

Перед выполнением каждого тестового метода у нас будет загружаться дамп базы данных который мы указали в настройках:

dump: 'tests/\_data/test.sql'

т.к. мы указали директиву **cleanup** при конфигурировании модуля. Таким образом, если бы в примере было несколько тестирующих методов, то все они имели бы одинаковую БД для старта.  
  
В тесте мы указываем, что находимся на главной странице, далее заполняем 2 поля формы с атрибутом «name» - name и email. Кликаем по кнопке отправки и осуществляем 2 проверки того, что данные действительно были сохранены в базу данных:

$I->see('Данные сохранены.');

тут используем сообщение которое должно отобразиться пользователю при успешном сохранении данных и

$I->seeInDatabase()

данный метод непосредственно использует обращение к базе данных к таблице users для поиска строки с указанными данными.  
  
Запустить тестирование можно обычным способом:

codecept run acceptance

Тестирование Веб Сервисов

Точно так же, как вы тестируете свой сайт, Codeception позволяет тестировать веб сервисы. Достаточно сложно тестировать их вручную, поэтому автоматизация тестирования в данном случае, является достаточно хорошей идеей.  
В Codeception имеется поддержка двух стандартов, SOAP и REST, представленных соответствующими модулями, которые и будут рассмотрены в данной статье.

Для начала, вам необходимо создать новый тестовый набор, который не предоставляется командой bootstrap. Мы рекомендуем назвать его **api** и использовать для него класс ApiGuy.

$ php codecept.phar generate:suite api ApiGuy

Там мы будем вставлять все наши api тесты.

REST

REST веб сервис работает через HTTP используя стандартные методы: GET, POST, PUT (PATCH), DELETE. Они позволяют получить и управлять данными сервиса. Использование веб сервисов требует наличия HTTP клиента, таким образом вам необходим модульPhpBrowser или один из настроенных модулей доступных фреймворков. К примеру мы можем использовать модуль Symfony2 для Symfony2 приложений с целью игнорирования веб сервера и тестирования веб сервиса изнутри.

Конфигурация модулей в api.suite.yml:

class\_name: ApiGuy

modules:

enabled: [PhpBrowser, REST, ApiHelper]

config:

PhpBrowser:

url: http://serviceapp/

REST:

url: http://serviceapp/api/v1/

Модуль REST автоматически подключится к PhpBrowser. В случае исползования Symfony2, Zend, или модулей других фреймворков, он будет подключаться к ним. Не забудьте выполнить команду build после того, как закончите редактировать конфигурацию.

Создадим первый простой тест:

php codecept.phar generate:cept api CreateUser

Назовем его CreateUserCept.php. Будем использовать его для создания пользователя используя веб сервис.

<?php

$I = new ApiGuy($scenario);

$I->wantTo('create a user via API');

$I->amHttpAuthenticated('service\_user','123456');

$I->haveHttpHeader('Content-Type','application/x-www-form-urlencoded');

$I->sendPOST('users', array('name' => 'davert', 'email' => 'davert@codeception.com'));

$I->seeResponseCodeIs(200);

$I->seeResponseIsJson();

$I->seeResponseContains('{ result: ok}');

?>

Модуль REST разработан для использования с сервисами оперирующими данными в JSON формате. К примеру методseeResponseContainsJson конвертирует указанный массив в json и проверит его наличие в ответе от сервиса.

Возможно вам захочется использовать более сложные утверждения assertions для проверки ответов. Это может быть реализовано с помощью написания собственных методов в клаccах [Helper](http://codeception.com/docs/03-Modules#helpers). Для доступа к последнему JSON ответу вам необходимо получить свойствоresponse модуля REST. Продемонстрируем это на примере метода seeResponseIsHtml:

getModule('REST')->response;

\PHPUnit\_Framework\_Assert::assertRegex('^.\*?<\/html>$', $response);

}

}

?>

Точно так же вы можете передавать параметры запроса и заголовки.

SOAP

Веб сервисы SOAP являются более сложным случаем. Вам необходим PHP [собранный с поддержкой SOAP](http://php.net/manual/en/soap.installation.php). Кроме того необходимо хорошее знание XML. SOAP модуль использует специально форматированный POST запрос для подключения к WSDL веб сервисам. Codeception использует PhpBrowser или один из модулей фреймворков для реализации данного взаимодействия. Если вы решите использовать один из модулей фреймворков, модуль SOAP будет автоматически подключен к фреймворку, и вы сможете увеличить скорость выполнения тестов а так же получить более детализированную трассировку и вывод.

Сконфигурируем модуль SOAP для использования с PhpBrowser:

class\_name: ApiGuy

modules:

enabled: [PhpBrowser, SOAP, ApiHelper]

config:

PhpBrowser:

url: http://serviceapp/

SOAP:

endpoint: http://serviceapp/api/v1/

Запрос SOAP может содержать специфическую для приложения информацию, к примеру информацию об аутерификации или об оплате. Эта информация обеспечивается SOAP заголовком внутри элемента <soap:Header> XML запроса. В случае, если вам нужно сгенерировать такой заголовок, вы можете использовать действие haveSoapHeader. К примеру, следующая строка кода

<?php

$I->haveSoapHeader('Auth', array('username' => 'Miles', 'password' => '123456'));

?>

сгенерирует данный XML заголовок

<soap:Header>

<Auth>

<username>Miles</username>

<password>123456</password>

</Auth>

</soap:Header>

Используйте метод sendSoapRequest для определения тела вашего запроса.

<?php

$I->sendSoapRequest('CreateUser', '<name>Miles Davis</name><email>miles@davis.com</email>');

?>

Этот вызов будет конвертирован в следующий XML код:

<soap:Body>

<ns:CreateUser>

<name>Miles Davis</name>

<email>miles@davis.com</email>

</ns:CreateUser>

</soap:Body>

А вот список простых утверждений которые могут быть использованы с SOAP.

<?php

$I->seeSoapResponseEquals('<?xml version="1.0"?><error>500</error>');

$I->seeSoapResponseIncludes('<result>1</result>');

$I->seeSoapResponseContainsStructure('<user><name></name><email></email>');

$I->seeSoapResponseContainsXPath('//result/user/name[@id=1]');

?>

В случае, если вы не хотите использовать блинные XML строки, подумайте об использовании [XmlBuilder](http://codeception.com/docs/reference/xmlbuilder) класса. Он поможет вам создавать сложные XML записи в стиле jQuery.  
В следующем примере покажем использование XmlBuilder (созданного с помощью фабрики SoapUtils) вместо использования XML.

<?php

use \Codeception\Utils\Soap;

$I = new ApiGuy($scenario);

$I->wantTo('create user');

$I->haveSoapHeader('Session', array('token' => '123456'));

$I->sendSoapRequest('CreateUser', Soap::request()->user->email->val('miles@davis.com'));

$I->seeSoapResponseIncludes(Soap::response()

->result->val('Ok')

->user->attr('id', 1)

);

?>

Решение о том, что использовать, XmlBuilder или XML строки остается за вами. XmlBuilder возвращает XML строки, как вы можете видеть.

ВЫ можете расширить существующую функциональность при помощи использования модуля SOAP в вашем классе помощнике. Для доступа к такому SOAP ответу как \DOMDocument вы можете использовать свойство response модуля SOAP.

<?php

class ApiHelper extends \Codeception\Module

{

public function seeResponseIsValidOnSchema($schema)

{

$response = $this->getModule('SOAP')->response;

$this->assertTrue($response->schemaValidate($schema));

}

}

?>