# Приемочные тесты -кратко

Приемочные тесты рассчитаны 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.

В чем разница между функциональными и приемочными тестами? Комнды используются одни и те же.

В приемочных тестах можно поверить работу java скриптов так тон запускаются в браузере.

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'

# url: 'http://myappurl.local'

Пересоздаем базовые классы тестов:

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 возьмем тот же тест страницы авторизации, который использовали при функциональном тестировании.

Последовательность действий такая же. Сначала генерируем тест:

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:

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)

Давайте посмотрим более детальный вывод:

codecept run acceptance --steps

Мы должны увидеть шаг за шагом последовательность выполняемых действий.

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');  
});

## Дополнения.

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

**<?php**// 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*);

## Selenium

$I->executeInSelenium(**function**(RemoteWebDriver $web) {  
 $web->getKeyboard()->pressKey(WebDriverKeys::*ENTER*);  
 $driver->executeScript('alert('*OK*');');  
 $rows = $driver->findElements(WebDriverBy::*cssSelector*('table tr'));  
});

## WebHelper

**<?php  
  
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));  
 }  
}

## **Пример создания приемочных тестов согласно функционального программирования (Cept).**

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**.

1. **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'.

## Пример тестирования формы.

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"

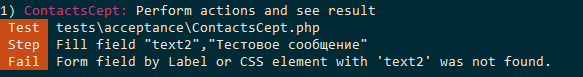
I fill field "text","Тестовое сообщение"

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».

## **Пример тестирования файловой системы:**

**<?php** $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 проверки сохранение данных из формы заполненной пользователем в базу данных.

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">

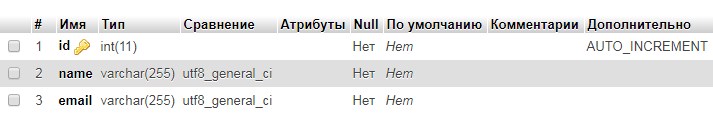
<input type="email" name="email">

<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

## 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!');

?>

##### 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.)

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:

codecept 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));

}

}

?>