# Функциональные тесты - кратко

У Codeception нет базового модуля для функционального тестирования. Поэтому в конфигурации необходимо включить один из поддерживаемых модулей фреймворков. Нужно это для того, что бы при запуске тестов не использовать веб сервер.

С полным списком модулей можно ознакомится на [GitHub](https://github.com/Codeception/Codeception/tree/master/docs/modules) или [официальном сайте](http://codeception.com/). Если в списке нет подходящего модуля можно интегрировать свой фреймворк. Делается это созданием моста между [BrowserKit](https://github.com/symfony/browser-kit) (его использует Codeception) и нашим приложением.

Мы будем использовать собственный модуль Gateway. Этот модуль подойдет если наш фреймворк использует Symfony HttpKernel.

Сначала создаем сам модуль по адресу tests/modules/Gateway.php:

namespace Codeception\Module;

use Codeception\TestCase;

use Symfony\Component\HttpKernel\Client;

class Gateway extends \Codeception\Lib\Framework

{

public function \_initialize()

{

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

$config = require dirname(\_\_DIR\_\_) . '/config/main.php';

$this->kernel = new \app\framework\Frontend($config);

}

public function \_before(TestCase $test)

{

$this->client = new Client($this->kernel);

$this->client->followRedirects(true);

}

public function isStatusCode($code)

{

return $this->kernel->handle()->getStatusCode() === $code;

}

}

Затем подключаем его в файле конфигурации (tests/functional.suite.yml):

class\_name: FunctionalTester

modules:

enabled: [Gateway, \Helper\Functional]

Пересоздаем базовые классы тестов:

codecept build

Теперь можем приступать к написанию тестов.

С помощью генератора добавим тест для страницы авторизации:

codecept generate:cept functional AuthPage

В созданном файле (tests/functional/AuthPageCept.php) пишем следующий код:

$I = new FunctionalTester($scenario);

$I->amOnPage('/');

$I->click('Авторизация');

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

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

$I->click('Войти');

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

Запускаем функциональные тесты командой:

codecept run functional

При успешном выполнении увидим такой результат:

Functional Tests (1) ------------------------------

✔ AuthPageCept: (0.03s)

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

Time: 165 ms, Memory: 14.25MB

OK (1 test, 1 assertion)

## Functional Testing,(функциональное тестирование)

Здесь, как мне кажется, есть смысл и перейти ко вполне конкретной теории. А именно, описать какими же assert-методами мы располагаем для проверки поведения тестируемых сценариев.  
  
Два самых простых — это **assertFalse()** и **assertTrue()**. Проверяют, является ли полученное значение false и true соответственно. Далее идут уже упомянутый **assertEquals()** и обратный ему **assertNotEquals()**. В их использовании есть нюансы. Так при сравнении чисел с плавающей точкой есть возможность указать точность сравнения. Так же эти методы используются для сравнения экземпляров класса DOMDocument, массивов и любых объектов (в последнем случае равенство будет установлено, если атрибуты объектов содержат одинаковые значения). Так же следует упомянуть **assertNull()** и **assertNotNull()** которые проверяют соответствие параметра типу данных NULL (да-да, не забываем, что в PHP это отдельный тип данных). Этим возможные сравнения не ограничиваются. Нет смысла в рамках этой статьи заниматься перепечаткой документации, потому приведу по возможности структурированный список всех возможных методов. Более детально интересующиеся могут прочитать [здесь](http://www.phpunit.de/manual/current/en/api.html#api.assert)

# Функциональное тестирование

Функциональные тесты это почти то же самое, что и приемочные, однако есть одно существенное различие: они не требуют использования веб сервера для запуска своих сценариев. Другими словами мы будем запускать наше приложение внутри тестов, имитируя запросы и ответы.

Говоря простыми словами, мы устанавливаем переменные $\_REQUEST, $\_GET и $\_POST, затем выполняем скрипт внутри теста, получаем ответ и тестируем все это.  
Функциональное тестирование часто может быть более лучшим решением, чем приемочное, потому как такие тесты не требуют использования веб сервера и могут предложить более подробный отладочный вывод. К примеру, если ваш сайт выбросит исключение, оно будет напечатано в консоли.

Codeception может подключаться к различным веб фреймворкам, поддерживающим функциональное тестирование. К примеру, вы можете запустить функциональные тесты для приложения построенного поверх Zend Framework, Symfony или Symfony2, используя лишь модули поставляемые Codeception! Список поддерживаемых модулей будет разобран позже.

Модули для всех этих фреймворков имеют одинаковый интерфейс, поэтому ваши тесты не будут связаны друг с другом. Вот простой пример функционального теста.

<?php

$I = new TestGuy($scenario);

$I->amOnPage('/');

$I->click('Login');

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

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

$I->click('Enter');

$I->see('Hello, Miles', 'h1');

// $I->seeEmailIsSent() - special for Symfony2

?>

Такой же тест, как и приемочный. Как видите, можно использовать одинаковые методы и для приемочных, и для функциональных тестов.  
Мы рекомендуем тестировать нестабильные части приложения с помощью функциональных тестов, а стабильные с помощью приемочных.

## Ловушки

Приемочные тесты обычно намного медленнее, чем функциональные. Однако функциональные тесты менее стабильны и запускают тестовый фреймворк и приложение в одном окружении.

#### Headers, Cookies, Sessions

Одна из известных проблем функциональных тестов - использование PHP функций работающих с переменными из категории headers,sessions, cookies.  
Как вы знаете, функция header возвратит ошибку, если будет выполнена более одного раза. В функциональных тестах мы запускаем наше приложение несколько раз, таким образом мы получим много ненужных ошибок при отображении результатов.

Разделяемая память

При функциональном тестировании, в отличие от традиционного, приложение PHP не останавливается после выполнения запроса.  
Так как все запросы выполняются в одном контейнере памяти, они не изолированны.  
Таким образом если вы заметили, что ваши тесты магическим образом падают, однако не должны - попробуйте выполнить один тест.  
Это проверит, изолированы ли тесты во время работы. Потому что довольно просто поломать окружение, когда все тесты выполняются в разделяемой памяти.  
Держите память в чистоте, избегайте утечек памяти и очищайте глобальные и статические переменные.

## Основы функционального тестирования

Ваши функциональные тесты располагаются в каталоге tests/functional.  
Для начала вам необходимо включить один из модулей фреймворков в конфигурационный файл тестового набора:tests/functional.suite.yml.

После вам необходимо пересобрать Guy-классы

codecept build

Для генерации теста вы можете использовать стандартную команду генератор generate:cept:

codecept generate:cept functional myFirstFunctional

После чего выполнить тесты с помощью run:

codecept run functional

Используйте опцию --debug для более детального вывода.