live search...

Acceptance Testing

Acceptance testing can be performed by a non-technical person. That person can be your tester, manager or even client. If you are developing a web-application (and you probably are) the tester needs nothing more than a web browser to check that your site works correctly. You can reproduce an acceptance tester's actions in scenarios and run them automatically. Codeception keeps tests clean and simple as if they were recorded from the words of an actual acceptance tester.

It makes no difference what (if any) CMS or framework is used on the site. You can even test sites created with different languages, like Java, .NET, etc. It's always a good idea to add tests to your website. At least you will be sure that site features work after the latest changes were made.

Sample Scenario

Let's say the first test you would want to run, would be signing in. In order to write such a test, we still require basic knowledge of PHP and HTML:

```php
<?php
$I->amOnPage('/login');
$I->fillField('username', 'davert');
$I->fillField('password', 'qwerty');
$I->click('LOGIN');
$I->see('Welcome, Davert!');
```

This scenario can be performed either by PhpBrowser or by a "real" browser through Selenium WebDriver.

	PhpBrowser	**WebDriver**

Follow @codeception

Browser Engine	Guzzle + Symfony BrowserKit	Chrome or Firefox
JavaScript	No	Yes
`see` / `seeElement` checks if…	…text is present in the HTML source	…text is actually visible to the user
Read HTTP response headers	Yes	No
System requirements	PHP with cURL extension (http://php.net/manual/book.curl.php)	Selenium Standalone Server, Chrome or Firefox
Speed	Fast	Slow

We will start writing our first acceptance tests with PhpBrowser.

PhpBrowser

This is the fastest way to run acceptance tests since it doesn't require running an actual browser. We use a PHP web scraper, which acts like a browser: It sends a request, then receives and parses the response. Codeception uses Guzzle (http://guzzlephp.org) and Symfony BrowserKit (http://symfony.com/doc/current/components/browser_kit.html) to interact with HTML web pages.

Common PhpBrowser drawbacks:

- You can only click on links with valid URLs or form submit buttons
- You can't fill in fields that are not inside a form

We need to specify the `url` parameter in the acceptance suite config:

```
# acceptance.suite.yml
actor: AcceptanceTester
modules:
    enabled:
        - PhpBrowser:
            url: http://www.example.com/
        - \Helper\Acceptance
```

We should start by creating a 'Cept' file:

```
<?php
// tests/acceptance/SigninCept.php
$I = new AcceptanceTester($scenario);
$I->wantTo('sign in');
```

The `$I` object is used to write all interactions. The methods of the `$I` object are taken from the PhpBrowser Module (http://codeception.com/docs/modules/PhpBrowser). We will briefly describe them here:

```
<?php
$I->amOnPage('/login');
```

We will assume that all actions starting with `am` and `have` describe the initial environment. The `amOnPage` action sets the starting point of a test to the `/login` page.

With the `PhpBrowser` you can click the links and fill in the forms. That will probably be the majority of your actions.

Click

Emulates a click on valid anchors. The URL referenced in the `href` attribute will be opened. As a parameter, you can specify the link name or a valid CSS or XPath selector.

Follow @codeception

```
<?php
$I->click('Log in');
// CSS selector applied
$I->click('#login a');
// XPath
$I->click('//a[@id=login]');
// Using context as second argument
$I->click('Login', '.nav');
```

Codeception tries to locate an element by its text, name, CSS or XPath. You can specify the locator type manually by passing an array as a parameter. We call this a **strict locator**. Available strict locator types are:

- id
- name
- css
- xpath
- link
- class

```
<?php
// By specifying locator type
$I->click(['link' => 'Login']);
$I->click(['class' => 'btn']);
```

There is a special class `Codeception\Util\Locator` (http://codeception.com/docs/reference/Locator) which may help you to generate complex XPath locators. For instance, it can easily allow you to click an element on the last row of a table:

```
$I->click('Edit' , \Codeception\Util\Locator::elementAt('//table/tr', -1));
```

Forms

Clicking links is probably not what takes the most time during the testing of a website. The most routine waste of time goes into the testing of forms. Codeception provides several ways of testing forms.

Let's submit this sample form inside the Codeception test:

```
<form method="post" action="/update" id="update_form">
     <label for="user_name">Name</label>
     <input type="text" name="user[name]" id="user_name" />
     <label for="user_email">Email</label>
     <input type="text" name="user[email]" id="user_email" />
     <label for="user_gender">Gender</label>
     <select id="user_gender" name="user[gender]">
          <option value="m">Male</option>
          <option value="f">Female</option>
     </select>
     <input type="submit" name="submitButton" value="Update" />
</form>
```

From a user's perspective, a form consists of fields which should be filled in, and then a submit button clicked:

```
<?php
// we are using label to match user_name field
$I->fillField('Name', 'Miles');
// we can use input name or id
$I->fillField('user[email]','miles@davis.com');
$I->selectOption('Gender','Male');
$I->click('Update');
```

To match fields by their labels, you should write a `for` attribute in the `label` tag.

Follow @codeception

From the developer's perspective, submitting a form is just sending a valid POST request to the server. Sometimes it's easier to fill in all of the fields at once and send the form without clicking a 'Submit' button. A similar scenario can be rewritten with only one command:

```
<?php
$I->submitForm('#update_form', array('user' => array(
     'name' => 'Miles',
     'email' => 'Davis',
     'gender' => 'm'
)));
```

The `submitForm` is not emulating a user's actions, but it's quite useful in situations when the form is not formatted properly, for example, to discover that labels aren't set or that fields have unclean names or badly written IDs, or the form is sent by a JavaScript call.

By default, `submitForm` doesn't send values for buttons. The last parameter allows specifying what button values should be sent, or button values can be explicitly specified in the second parameter:

```
<?php
$I->submitForm('#update_form', array('user' => array(
     'name' => 'Miles',
     'email' => 'Davis',
     'gender' => 'm'
)), 'submitButton');
// this would have the same effect, but the value has to be explicitly specified
$I->submitForm('#update_form', array('user' => array(
     'name' => 'Miles',
     'email' => 'Davis',
     'gender' => 'm',
     'submitButton' => 'Update'
)));
```

Assertions

In the `PhpBrowser` you can test the page contents. In most cases, you just need to check that the required text or element is on the page.

The most useful method for this is `see()` :

```
<?php
// We check that 'Thank you, Miles' is on the page.
$I->see('Thank you, Miles');
// We check that 'Thank you, Miles' is inside an element with 'notice' class.
$I->see('Thank you, Miles', '.notice');
// Or using XPath locators
$I->see('Thank you, Miles', "//table/tr[2]");
// We check this message is *not* on the page.
$I->dontSee('Form is filled incorrectly');
```

You can check that a specific HTML element exists (or doesn't) on a page:

```
<?php
$I->seeElement('.notice');
$I->dontSeeElement('.error');
```

We also have other useful commands to perform checks. Please note that they all start with the `see` prefix:

Follow @codeception

```php
<?php
$I->seeInCurrentUrl('/user/miles');
$I->seeCheckboxIsChecked('#agree');
$I->seeInField('user[name]', 'Miles');
$I->seeLink('Login');
```

Conditional Assertions

Usually, as soon as any assertion fails, further assertions of this test will be skipped. Sometimes you don't want this - maybe you have a long-running test and you want it to run to the end. In this case, you can use conditional assertions. Each `see` method has a corresponding `canSee` method, and `dontSee` has a `cantSee` method:

```php
<?php
$I->canSeeInCurrentUrl('/user/miles');
$I->canSeeCheckboxIsChecked('#agree');
$I->cantSeeInField('user[name]', 'Miles');
```

Each failed assertion will be shown in the test results, but it won't stop the test.

Comments

Within a long scenario, you should describe what actions you are going to perform and what results should be achieved. Comment methods like `amGoingTo`, `expect`, `expectTo` help you in making tests more descriptive:

```php
<?php
$I->amGoingTo('submit user form with invalid values');
$I->fillField('user[email]', 'miles');
$I->click('Update');
$I->expect('the form is not submitted');
$I->see('Form is filled incorrectly');
```

Grabbers

These commands retrieve data that can be used in the test. Imagine your site generates a password for every user and you want to check that the user can log into the site using this password:

```php
<?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!');
```

Grabbers allow you to get a single value from the current page with commands:

```php
<?php
$token = $I->grabTextFrom('.token');
$password = $I->grabTextFrom("descendant::input/descendant::*[@id = 'password']");
$api_key = $I->grabValueFrom('input[name=api]');
```

Cookies, URLs, Title, etc

Actions for cookies:

```php
<?php
$I->setCookie('auth', '123345');
$I->grabCookie('auth');
$I->seeCookie('auth');
```

Actions for checking the page title:

Follow @codeception

```
<?php
$I->seeInTitle('Login');
$I->dontSeeInTitle('Register');
```

Actions for URLs:

```
<?php
$I->seeCurrentUrlEquals('/login');
$I->seeCurrentUrlMatches('~$/users/(\d+)~');
$I->seeInCurrentUrl('user/1');
$user_id = $I->grabFromCurrentUrl('~$/user/(\d+)/~');
```

WebDriver

A nice feature of Codeception is that most scenarios are similar, no matter of how they are executed. `PhpBrowser` was emulating browser requests but how to execute such test in a real browser like Chrome or Firefox? Selenium WebDriver can drive them so in our acceptance tests we can automate scenarios we used to test manually. In such tests, we should concentrate more on **testing the UI** than on testing functionality.

"WebDriver (https://www.w3.org/TR/webdriver/)" is the name of a protocol (specified by W3C) to drive browsers automatically. This specification is implemented for all modern desktop and mobile browsers. Codeception uses facebook/php-webdriver (https://github.com/facebook/php-webdriver) library from Facebook as PHP implementation of WebDriver protocol.

To control the browsers you need to use a program or a service to start/stop browser sessions. In the next section, we will overview the most popular solutions.

Local Setup

Selenium Server

Selenium Server (http://www.seleniumhq.org/) is a de-facto standard for automated web and mobile testing. It is a server that can launch and drive different browsers locally or remotely. WebDriver protocol was initially created by Selenium before becoming a W3C standard. This makes Selenium server the most stable complete implementation of WebDriver for today.
Selenium Server is also recommended by Codeception team.

To control browsers Selenium Server uses official tools maintained by browser vendors, like ChromeDriver (https://sites.google.com/a/chromium.org/chromedriver) for Chrome or GeckoDriver (https://github.com/mozilla/geckodriver) for Firefox. This makes Selenium quite heavy to install, as it requires Java, browsers, Chrome or GeckoDriver and GUI (display server) to run browsers in.

- Follow Installation Instructions (http://codeception.com/docs/modules/WebDriver#Selenium)
- Enable RunProcess (http://codeception.com/extensions#RunProcess) extension to start/stop Selenium automatically *(optional)*.

PhantomJS

PhantomJS is a customized WebKit-based headless browser (https://en.wikipedia.org/wiki/Headless_browser) built for programmatic usage only. It doesn't display a browser window and doesn't require GUI (display server) to be installed. This makes PhantomJS highly popular for Continuous Integration systems. PhantomJS needs only one binary with no extra dependencies which make it the simplest WebDriver tool to install.

However, it should be noted that PhantomJS is not a real browser, so the behavior and output in real browsers may differ from PhantomJS. And the most important: **PhantomJS is not maintained** anymore. So use it at your own risk.

- Follow Installation Instructions (http://codeception.com/docs/modules/WebDriver#PhantomJS)
- Enable RunProcess (http://codeception.com/extensions#RunProcess) extension to start/stop PhantomJS automatically *(optional)*.

ChromeDriver

Follow @codeception

ChromeDriver was created by Google to control Chrome and Chromium browsers programmatically. It can be paired with Selenium Server (http://codeception.com/docs/03-AcceptanceTests#Selenium-Server) or used as a standalone tool to drive Chrome browser. It is simpler to set up than Selenium Server, however, it has limited support for WebDriver protocol.

- Follow Installation Instructions (http://codeception.com/docs/modules/WebDriver#ChromeDriver)
- Enable RunProcess (http://codeception.com/extensions#RunProcess) extension to start/stop ChromeDriver automatically *(optional)*.

Configuration

To execute a test in a browser we need to change the suite configuration to use **WebDriver** instead of `PhpBrowser`.

Modify your `acceptance.suite.yml` file:

```
actor: AcceptanceTester
modules:
    enabled:
        - WebDriver:
            url:
            browser: chrome
        - \Helper\Acceptance
```

See WebDriver Module (http://codeception.com/docs/modules/WebDriver) for details.

Please note that actions executed in a browser will behave differently. For instance, `seeElement` won't just check that the element exists on a page, but it will also check that element is actually visible to the user:

```
<?php
$I->seeElement('#modal');
```

While WebDriver duplicates the functionality of PhpBrowser, it has its limitations: It can't check headers since browsers don't provide APIs for that. WebDriver also adds browser-specific functionality:

Wait

While testing web application, you may need to wait for JavaScript events to occur. Due to its asynchronous nature, complex JavaScript interactions are hard to test. That's why you may need to use waiters, actions with `wait` prefix. They can be used to specify what event you expect to occur on a page, before continuing the test.

For example:

```
<?php
$I->waitForElement('#agree_button', 30); // secs
$I->click('#agree_button');
```

In this case, we are waiting for the 'agree' button to appear and then click it. If it didn't appear after 30 seconds, the test will fail. There are other `wait` methods you may use, like waitForText (http://codeception.com/docs/modules/WebDriver#waitForText), waitForElementVisible (http://codeception.com/docs/modules/WebDriver#waitForElementVisible) and others.

If you don't know what exact element you need to wait for, you can simply pause execution with using `$I->wait()`

```
<?php
$I->wait(3); // wait for 3 secs
```

SmartWait

since 2.3.4 version

It is possible to wait for elements pragmatically. If a test uses element which is not on a page yet, Codeception will wait for few extra seconds before failing. This feature is based on Implicit Wait (http://www.seleniumhq.org/docs/04_webdriver_advanced.jsp#implicit-waits) of Selenium. Codeception enables implicit wait only

Follow @codeception

when searching for a specific element and disables in all other cases. Thus, the performance of a test is not affected.

SmartWait can be enabled by setting `wait` option in WebDriver config. It expects the number of seconds to wait. Example:

```
wait: 5
```

With this config we have the following test:

```
<?php
// we use wait: 5 instead of
// $I->waitForElement(['css' => '#click-me'], 5);
// to wait for element on page
$I->click(['css' => '#click-me']);
```

It is important to understand that SmartWait works only with a specific locators:

- `#locator` - CSS ID locator, works
- `//locator` - general XPath locator, works
- `['css' => 'button'']` - strict locator, works

But it won't be executed for all other locator types. See the example:

```
<?php
$I->click('Login'); // DISABLED, not a specific locator
$I->fillField('user', 'davert'); // DISABLED, not a specific locator
$I->fillField(['name' => 'password'], '123456'); // ENABLED, strict locator
$I->click('#login'); // ENABLED, locator is CSS ID
$I->see('Hello, Davert'); // DISABLED, Not a locator
$I->seeElement('#userbar'); // ENABLED
$I->dontSeeElement('#login'); // DISABLED, can't wait for element to hide
$I->seeNumberOfElements(['css' => 'button.link'], 5); // DISABLED, can wait only for one element
```

Wait and Act

To combine `waitForElement` with actions inside that element you can use the performOn (http://codeception.com/docs/modules/WebDriver#performOn) method. Let's see how you can perform some actions inside an HTML popup:

```
<?php
$I->performOn('.confirm', \Codeception\Util\ActionSequence::build()
    ->see('Warning')
    ->see('Are you sure you want to delete this?')
    ->click('Yes')
);
```

Alternatively, this can be executed using a callback, in this case the `WebDriver` instance is passed as argument

```
<?php
$I->performOn('.confirm', function(\Codeception\Module\WebDriver $I) {
    $I->see('Warning');
    $I->see('Are you sure you want to delete this?');
    $I->click('Yes');
});
```

For more options see `performOn()` reference ([performOn](http://codeception.com/docs/modules/WebDriver#performOn)).

Multi Session Testing

Codeception allows you to execute actions in concurrent sessions. The most obvious case for this is testing realtime messaging between users on a site. In order to do it, you will need to launch two browser windows at the same time for the same test. Codeception has a very smart concept for doing this. It is called **Friends**:

Follow @codeception

```php
<?php
$I->amOnPage('/messages');
$nick = $I->haveFriend('nick');
$nick->does(function(AcceptanceTester $I) {
    $I->amOnPage('/messages/new');
    $I->fillField('body', 'Hello all!');
    $I->click('Send');
    $I->see('Hello all!', '.message');
});
$I->wait(3);
$I->see('Hello all!', '.message');
```

In this case, we performed, or 'did', some actions in the second window with the `does` method on a friend object.

Sometimes you may want to close a webpage before the end of the test. For such cases, you may use `leave()`. You can also specify roles for a friend:

```php
<?php
$nickAdmin = $I->haveFriend('nickAdmin', adminStep::class);
$nickAdmin->does(function(adminStep $I) {
    // Admin does ...
});
$nickAdmin->leave();
```

Cloud Testing

Some environments are hard to be reproduced manually, testing Internet Explorer 6-8 on Windows XP may be a hard thing, especially if you don't have Windows XP installed. This is where Cloud Testing services come to help you. Services such as SauceLabs (https://saucelabs.com), BrowserStack (https://www.browserstack.com/) and others (http://codeception.com/docs/modules/WebDriver#Cloud-Testing) can create virtual machines on demand and set up Selenium Server and the desired browser. Tests are executed on a remote machine in a cloud, to access local files cloud testing services provide a special application called **Tunnel**. Tunnel operates on a secured protocol and allows browsers executed in a cloud to connect to a local web server.

Cloud Testing services work with the standard WebDriver protocol. This makes setting up cloud testing really easy. You just need to set the WebDriver configuration (http://codeception.com/docs/modules/WebDriver#Cloud-Testing) to:

- specify the host to connect to (depends on the cloud provider)
- authentication details (to use your account)
- browser
- OS

We recommend using params (http://codeception.com/docs/06-ModulesAndHelpers#Dynamic-Configuration-With-Params) to provide authorization credentials.

It should be mentioned that Cloud Testing services are not free. You should investigate their pricing models and choose one that fits your needs. They also may work painfully slowly if ping times between the local server and the cloud is too high. This may lead to random failures in acceptance tests.

AngularJS Testing

In the modern era of Single Page Applications, the browser replaces the server in creating the user interface. Unlike traditional web applications, web pages are not reloaded on user actions. All interactions with the server are done in JavaScript with XHR requests. However, testing Single Page Applications can be a hard task. There could be no information of the application state: e.g. has it completed rendering or not? What is possible to do in this case is to use more `wait*` methods or execute JavaScript that checks the application state.

Follow @codeception

For applications built with the AngularJS v1.x framework, we implemented AngularJS module (http://codeception.com/docs/modules/AngularJS) which is based on Protractor (an official tool for testing Angular apps). Under the hood, it pauses step execution before the previous actions are completed and use the AngularJS API to check the application state.

The AngularJS module extends WebDriver so that all the configuration options from it are available.

Debugging

Codeception modules can print valuable information while running. Just execute tests with the `--debug` option to see running details. For any custom output use the `codecept_debug` function:

```
<?php
codecept_debug($I->grabTextFrom('#name'));
```

On each failure, the snapshot of the last shown page will be stored in the `tests/_output` directory. PhpBrowser will store the HTML code and WebDriver will save a screenshot of the page.

Additional debugging features by Codeception:

- pauseExecution (http://codeception.com/docs/modules/WebDriver#pauseExecution) method of WebDriver module allows pausing the test.
- Recorder extension (http://codeception.com/addons#CodeceptionExtensionRecorder) allows to record tests step-by-steps and show them in slideshow
- Interactive Console (http://codeception.com/docs/07-AdvancedUsage#Interactive-Console) is a REPL that allows to type and check commands for instant feedback.

Custom Browser Sessions

By default, WebDriver module is configured to automatically start browser before the test and stop afterward. However, this can be switched off with `start: false` module configuration. To start a browser you will need to write corresponding methods in Acceptance Helper (http://codeception.com/docs/06-ModulesAndHelpers#Helpers).

WebDriver module provides advanced methods for the browser session, however, they can only be used from Helpers.

- _initializeSession (http://codeception.com/docs/modules/WebDriver#_initializeSession) - starts a new browser session
- _closeSession (http://codeception.com/docs/modules/WebDriver#_closeSession) - stops the browser session
- _restart (http://codeception.com/docs/modules/WebDriver#_restart) - updates configuration and restarts browser
- _capabilities (http://codeception.com/docs/modules/WebDriver#_capabilities) - set desired capabilities (https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities) programmatically.

Those methods can be used to create custom commands like `$I->startBrowser()` or used in before/after (http://codeception.com/docs/06-ModulesAndHelpers#Hooks) hooks.

Conclusion

Writing acceptance tests with Codeception and PhpBrowser is a good start. You can easily test your Joomla, Drupal, WordPress sites, as well as those made with frameworks. Writing acceptance tests is like describing a tester's actions in PHP. They are quite readable and very easy to write. If you need to access the database, you can use the Db Module (http://codeception.com/docs/modules/Db).

- **Next Chapter: FunctionalTests > (/docs/04-FunctionalTests)**
- **Previous Chapter: < GettingStarted (/docs/02-GettingStarted)**

Looking for commercial support? Request official consulting service (http://sdclabs.com/codeception?utm_source=codeception.com&utm_medium=docs_bottom&utm_term=link&utm_campaign=reference)

Codeception is a BDD-styled PHP testing framework, brought to you by Codeception Team (http://codeception.com/credits). Logo by Mr. Adnan (https://twitter.com/adnanblog). OpenSource **MIT Licensed**.

Follow @codeception

Thanks to

(https://www.jetbrains.com/phpstorm/) (https://www.rebilly.com/)

(https://github.com/codeception/codeception) (https://twitter.com/codeception)

(http://www.facebook.com/pages/Codeception/288959711204412)

- Installation (/install)
- Credits (/credits)
- Releases (/changelog)
- Commercial Services (http://sdclabs.com/codeception?utm_source=codeception.com&utm_medium=bottom_menu&utm_term=link&utm_campaign=reference)
- License (https://github.com/Codeception/Codeception/blob/master/LICENSE)

Codeception Family

(http://robo.li)

Robo

(http://robo.li)

Modern PHP **Task Runner**. Allows to declare tasks with zero configuration in pure PHP.

(http://codecept.io)

CodeceptJS

(http://codecept.io)

Codeception for **NodeJS**. Write acceptance tests in ES6 and execute in webdriverio, Selenium WebDriver, and Protractor.

© 2011–2017

Follow @codeception