PHPUnit-Mink Documentation

Release 1.0.0

Alexander Obuhovich

Nov 24, 2022

Contents

This library is an extension for PHPUnit, that allows to write tests with help of Mink.

CHAPTER 1

Overview

This library allows to perform following things:

- use Mink for browser session control
- each test in a test case can use independent browser session
- all tests in a test case can share browser session between them
- Selenium server connection details are decoupled from tests using them
- perform individual browser configuration for each test in a test case
- support for Sauce Labs
- remote code coverage collection

Each mentioned above features is described in more detail below.

CHAPTER 2

Service Integrations

2.1 Getting Started

Below you'll find all needed information to find your way across the library.

2.1.1 Installation

Library can be installed using Composer like so:

1. define the dependencies in your `composer.json`:

```
{
    "require": {
        "aik099/phpunit-mink": "~2.0"
    }
}
```

2. install/update your vendors:

```
$ curl http://getcomposer.org/installer | php
$ php composer.phar install
```

2.1.2 Basic Usage

1. sub-class test case class from `\aik099\PHPUnit\BrowserTestCase` class (line 5)

2. define used browser configurations in static `$browsers` property of that class (line 8-21)
3. access Mink session by calling `$this->getSession()` method in your test (line 26)
4. access browser configuration by calling `$this->getBrowser()` method in your test (line 40)

```
1  <?php
2
3  use aik099\PHPUnit\BrowserTestCase;
4
5  class GeneralTest extends BrowserTestCase
6  {
7
8      public static $browsers = array(
9          array(
10             'driver' => 'selenium2',
11             'host' => 'localhost',
12             'port' => 4444,
13             'browserName' => 'firefox',
14             'baseUrl' => 'http://www.google.com',
15         ),
16     );
17
18     public function testUsingSession()
19     {
20         // This is Mink's Session.
21         $session = $this->getSession();
22
23         // Go to a page.
24         $session->visit('http://www.google.com');
25
26         // Validate text presence on a page.
27         $this->assertTrue($session->getPage()->hasContent('Google'));
28     }
29
30     public function testUsingBrowser()
31     {
32         // Prints the name of used browser.
33         echo sprintf(
34             "I'm executed using '%s' browser",
35             $this->getBrowser()->getBrowserName()
36         );
37     }
38
39 }
```

2.1.3 Selenium in Cloud

When using Selenium-based solution for automated testing in the cloud (e.g. Sauce Labs or BrowserStack) you need to specify following settings:

- `'type' => 'saucelabs'` or `'type' => 'browserstack'`
- `'apiUsername' => '...'`
- `'apiKey' => '...'`

instead of `host` and `port` settings. In all other aspects everything will work the same as if all tests were running locally.

```php
<?php

use aik099\PHPUnit\BrowserTestCase;

class BrowserConfigExampleTest extends BrowserTestCase
{

    public static $browsers = array(
        // Sauce Labs browser configuration.
        array(
            'type' => 'saucelabs',
            'apiUsername' => '...',
            'apiKey' => '...',
            'browserName' => 'firefox',
            'baseUrl' => 'http://www.google.com',
        ),
        // BrowserStack browser configuration.
        array(
            'type' => 'browserstack',
            'api_username' => '...',
            'api_key' => '...',
            'browserName' => 'firefox',
            'baseUrl' => 'http://www.google.com',
        ),
        // Regular browser configuration.
        array(
            'driver' => 'selenium2',
            'host' => 'localhost',
            'port' => 4444,
            'browserName' => 'chrome',
            'baseUrl' => 'http://www.google.com',
        ),
    );

}
```

2.1.4 Continuous Integration

When website under test isn't publicly accessible, then:

1. secure tunnel needs to be created from website under test to server, that runs the tests
2. created tunnel identifier needs to specified in the `PHPUNIT_MINK_TUNNEL_ID` environment variable

Note: Before v2.1.0 the environment variable was called `TRAVIS_JOB_NUMBER`.

How to Create a Tunnel

- SauceLabs: https://wiki.saucelabs.com/display/DOCS/Sauce+Connect+Proxy
- BrowserStack: http://www.browserstack.com/automate/php#setting-local-tunnel

2.2 Configuring Browser

The browser needs to be configured in a test case before being able to access Mink session. All possible ways of browser configuration are described below.

2.2.1 Per Test Configuration

It is possible to configure browser individually for each test within a test case by creating an instance of `\aik099\PHPUnit\BrowserConfiguration\BrowserConfiguration` class in `setUp` method in of test case class and setting it via `setBrowser` method.

```php
<?php

use aik099\PHPUnit\BrowserTestCase;

class PerTestBrowserConfigTest extends BrowserTestCase
{

    /**
     * @before
     */
    protected function setUpTest()
    {
        // To create regular browser configuration via BrowserConfigurationFactory.
        $browser = $this->createBrowserConfiguration(array(
            // options goes here (optional)
        ));

        // To create "Sauce Labs" browser configuration via BrowserConfigurationFactory.
        $browser = $this->createBrowserConfiguration(array(
            // required
            'type' => 'saucelabs',
            'apiUsername' => 'sauce_username',
            'apiKey' => 'sauce_api_key',
            // optional options goes here
        ));

        // To create "BrowserStack" browser configuration via BrowserConfigurationFactory.
        $browser = $this->createBrowserConfiguration(array(
            // required
            'type' => 'browserstack',
            'api_username' => 'bs_username',
            'api_key' => 'bs_api_key',
            // optional options goes here
        ));

        // Options can be changed later (optional).
        $browser->setHost('selenium_host')->setPort('selenium_port')->setTimeout(30);
        $browser->setBrowserName('browser name')->setDesiredCapabilities(array(
            'version' => '6.5'
        ));
        $browser->setBaseUrl('http://www.test-host.com');

        // Set browser configuration to test case.
```

(continues on next page)

(continued from previous page)

```
44          $this->setBrowser($browser);
45
46          parent::setUpTest();
47      }
48
49  }
```

2.2.2 Per Test Case Configuration

In case, when all tests in a test case share same browser configuration it's easier to specify it via static `$browsers` property (array, where each item represents a single browser configuration) in that test case class.

```
1   <?php
2
3   use aik099\PHPUnit\BrowserTestCase;
4
5   class PerTestCaseBrowserConfigTest extends BrowserTestCase
6   {
7
8       public static $browsers = array(
9           array(
10              'driver' => 'selenium2',
11              'host' => 'localhost',
12              'port' => 4444,
13              'browserName' => 'firefox',
14              'baseUrl' => 'http://www.google.com',
15          ),
16          array(
17              'driver' => 'selenium2',
18              'host' => 'localhost',
19              'port' => 4444,
20              'browserName' => 'chrome',
21              'baseUrl' => 'http://www.google.com',
22          ),
23      );
24
25  }
```

Note: When several browser configurations are specified in `$browsers` array, then each test in a test case will be executed against each of browser configurations.

2.2.3 Browser Session Sharing

As a benefit of shared (per test case) browser configuration, that was described above is an ability to not only share browser configuration, that is used to create Mink session, but to actually share created sessions between all tests in a single test case. This can be done by adding `sessionStrategy` option (line 14) to the browser configuration.

```
1   <?php
2
3   use aik099\PHPUnit\BrowserTestCase;
4
```

(continues on next page)

(continued from previous page)

```
class CommonBrowserConfigTest extends BrowserTestCase
{

    public static $browsers = array(
        array(
            'driver' => 'selenium2',
            'host' => 'localhost',
            'port' => 4444,
            'browserName' => 'firefox',
            'baseUrl' => 'http://www.google.com',
            'sessionStrategy' => 'shared',
        ),
    );

}
```

2.2.4 Selecting the Mink Driver

With the help of `driver` and `driverOptions` browser configuration settings (since v2.1.0) it's possible to specify which Mink driver to use. This file demonstrates how to use each driver:

```
<?php

use aik099\PHPUnit\BrowserTestCase;

class DriverShowCaseTest extends BrowserTestCase
{

    public static $browsers = array(
        array(
            'driver' => 'goutte',

            // Defaults for this driver.
            'driverOptions' => array(
                'server_parameters' => array(),
                'guzzle_parameters' => array(),
            ),

        ),
        array(
            'driver' => 'sahi',

            // Defaults for this driver.
            'port' => 9999,
            'driverOptions' => array(
                'sid' => null,
                'limit' => 600,
                'browser' => null,
            ),
        ),

        array(
            'driver' => 'selenium2',

            // Defaults for this driver.
```

(continues on next page)

(continued from previous page)

```
            'port' => 4444,
            'driverOptions' => array(),
        ),

        array(
            'driver' => 'zombie',

            // Defaults for this driver.
            'port' => 8124,
            'driverOptions' => array(
                'node_bin' => 'node',
                'server_path' => null,
                'threshold' => 2000000,
                'node_modules_path' => '',
            ),
        ),
    );

}
```

2.2.5 Configuration Options

Each browser configuration consists of the following settings (all optional):

Name	Description
`driver`	Mink driver name (defaults to `selenium2`, since v2.1.0)
`driverOptions`	Mink driver specific options (since v2.1.0)
`host`	host, where driver's server is located (defaults to `localhost`)
`port`	port, on which driver's server is listening for incoming connections (determined by driver)
`timeout`	connection timeout of the server in seconds ('selenium2' driver only, defaults to `60`)
`browserName`	name of browser to use (e.g. `firefox`, `chrome`, etc., defaults to `firefox`)
`desiredCapabilities`	parameters, that allow to fine-tune browser and other 'selenium2' driver options (e.g. 'tags', 'project', 'os', 'version')
`baseUrl`	base url of website, that is tested
`sessionStrategy`	used session strategy (defaults to `isolated`)
`type`	type of configuration (defaults to `default`, but can also be `saucelabs` or `browserstack`)
`apiUsername`	API username of used service (applicable to 'saucelabs' and 'browserstack' browser configurations)
`apiKey`	API key of used service (applicable to 'saucelabs' and 'browserstack' browser configurations)

There are also corresponding setters (e.g. `setHost`) and getters (e.g. `getHost`) for each of mentioned above settings, that allow to individually change them from `setUp` method before test has started.

2.3 Browser Aliases

All previous examples demonstrate various ways how browser configuration can be defined, but they all have same downside - server connection details stay hard-coded in test case classes. This could become very problematic if:

- same test cases needs to be executed on different servers (e.g. each developer runs them on his own machine)

- due change in server connection details each test case class needs to be changed

To solve this problem a browser aliases were introduced. Basically a browser alias is predefined browser configuration, that is available in the test case by it's alias. Here is how it can be used:

1. create base test case class, by extending `BrowserTestCase` class in the project with `getBrowserAliases` method in it
2. the `getBrowserAliases` method will return an associative array of a browser configurations (array key acts as alias name)
3. in any place, where browser configuration is defined use `'alias' => 'alias_name_here'` instead of actual browser configuration
4. feel free to override any part of configuration defined in alias

Note: Nested aliases are also supported.

```php
<?php

use aik099\PHPUnit\BrowserTestCase;

abstract class BrowserAliasTestCase extends BrowserTestCase
{

    public function getBrowserAliases()
    {
        return array(
            'example_alias' => array(
                'driver' => 'selenium2',
                'host' => 'localhost',
                'port' => 4444,
                'browserName' => 'firefox',
                'baseUrl' => 'http://www.google.com',
            ),
        );
    }

}


class ConcreteTest extends BrowserAliasTestCase
{

    public static $browsers = array(
        array(
            'alias' => 'example_alias',
        ),
        array(
            'alias' => 'example_alias',
            'browserName' => 'chrome',
        ),
    );
}
```

2.4 Remote Code Coverage

Browser tests are executed on different machine, then one, where code coverage information is collected (and tests are executed). To solve that problem this library uses remote coverage collection. Following steps needs to be performed before using this feature:

2.4.1 On Remote Server

This is web-server, where website used in tests is located.

1. Install Xdebug PHP extension on web-server
2. Copy `library/aik099/PHPUnit/RemoteCoverage/RemoteCoverageTool.php` into web-server's DocumentRoot directory.
3. Include following code before your application bootstraps:

```
<?php

require_once 'RemoteCoverageTool.php';
\aik099\PHPUnit\RemoteCoverage\RemoteCoverageTool::init();
```

2.4.2 On Test Machine

This is machine, where PHPUnit tests are being executed.

Following code needs to be placed in the `setUp` method of the test case class (that extends `BrowserTestCase` class) to enable remote coverage information collection:

```
<?php

// "host" should be replaced with web server's url
$this->setRemoteCoverageScriptUrl('http://host/');
```

2.4.3 How This Works

1. each test sets a special cookie on website under test
2. when cookie is present, then `RemoteCoverageTool.php` script collects coverage information and stores it on disk
3. once test finishes, then `http://host/?rct_mode=output` url is accessed on remote server, which in turn returns collected coverage information
4. remote coverage information is then joined with coverage information collected locally on test machine