PHPUnit in Laravel

Installation and Setup

Step 1: Install PHPUnit and update Composer

```
sudo composer global require phpunit/phpunit sudo composer global update
```

Step 2: Install Laracasts/Integrated

On the new Laravel project root

```
composer require laracasts/integrated --dev
```

Step 3: Extend

```
// tests/TestCase.php use Laracasts\Integrated\Extensions\Laravel as IntegrationTest; class TestCase
extends IntegrationTest { // ... }
```

Extensions

visit($uri)

This will perform a GET request to the given `$uri`, while also triggering an assertion to guarantee that a `200 status` code was returned.

```
$this->visit('/page');
```

see($text)

To verify that the current page contains the given text, you will want to use the see method.

```
$this->visit('/page') ->see('Hello World');
```

click($linkText) or follow($linkText)

To simulate the behavior of clicking a link on the page, the click method is your friend.

```
$this->visit('/page') ->click('Follow Me');
```

seePageIs($uri) and onPage($uri)

In many situations, it can prove useful to make an assertion against the current url.

```
$this->visit('/page') ->click('Follow Me') ->seePageIs('/next-page');
```

type($text, $selector) or fill($text, $selector)

If you need to type something into an input field, one option is to use the type method, like so:

```
$this->visit('search') ->type('Total Recall', '#search');
```

Simply provide the value for the input, and a CSS selector for us to hunt down the input that you are looking for. You may pass an id, element name, or an input with the given "name" attribute. The fill method is an alias that does the same thing.

tick($name) or check($name)

To "tick" a checkbox, call the tick method, and pass either the id or the name of the input.

```
$this->visit('newsletter') ->tick('opt-in') ->press('Save');
```

The check method is an alias for tick. Use either.

select($selectName, $optionValue)

This method allows you to select an option from a dropdown. You only need to provide the name of the select element, and the value attribute from the desired option tag.

```
$this->visit('signup') ->select('plan', 'monthly') ->press('Sign Up');
```

The following HTML would satisfy the example above

```
<form method="POST" action="..."> <select name="plan"> <option value="monthly">Monthly</option> <option value="yearly">Yearly</option> </select> <input type="submit" value="Sign Up"> </form>
```

attachFile($fileInputName, $absolutePath)

Imagine that, as part of filling out a form, you need to attach a file. Easy enough!

```
$this->visit('/page') ->attachFile('input-name', __DIR__.'/foo.txt') ->press('Submit');
```

It is important that you provide an absolute path to the file you wish to attach. Do not use a relative path.

press($submitText)

Not to be confused with click, the press method is used to submit a form with a submit button that has the given text.

```
$this->visit('search') ->type('Total Recall', '#q') ->press('Search Now');
```

When called, this package will handle the process of submitting the form, and following any applicable redirects. This means, we could combine some of previous examples to form a full integration test.

```
$this->visit('/search') ->type('Total Recall', '#q') ->press('Search Now') ->andSee('Search results for "Total Recall"') ->onPage('/search/results');
```

submitForm($submitText, $formData)

For situations where multiple form inputs must be filled out, you might choose to forego multiple type() calls, and instead use the submitForm method.

```
$this->visit('/search') ->submitForm('Search Now', ['q' => 'Total Recall']);
```

This method offers a more compact option, which will both populate and submit the form.

Take special note of the second argument, which is for the form data. You willl want to pass an associative array, where each key refers to the "name" of an input (not the element name, but the "name" attribute). As such, this test would satisfy the following form:

```
<form method="POST" action="/search/results"> <input type="text" name="q" placeholder="Search for something..."> <input type="submit" value="Search Now"> </form>
```

seeInDatabase($table, $data) or verifyInDatabase($table, $data)

For situations when you want to peek inside the database to verify that a certain record/row exists, seeInDatabase or its alias verifyInDatabase will do the trick nicely.

```
$data = ['description' => 'Finish documentation']; $this->visit('/tasks') ->submitForm('Create Task', $data) ->verifyInDatabase('tasks', $data);
```

When calling verifyInDatabase, as the two arguments, provide the name of the table you are interested in, and an array of any attributes for the query.

seeFile($path)

To ensure that a file exists, you may use the seeFile method.

dump()

When you want to quickly spit out the response content from the most recent request, call the dump method, like so:

```
$this->visit('/page')->dump();
```

This will both dump the response content to the console, and save it to a tests/logs/output.txt file, for your review. Please note that this method will die. So no tests beyond this call will be fired. Nonetheless, it is great for the times when you need a better look at what you are working with, temporarily of course.

Example

Real Example

```
// ExampleTest.php use Laracasts\Integrated\Extensions\Laravel as IntegrationTest; class ExampleTest extends IntegrationTest { /** @test */ public function it_verifies_that_pages_load_properly() { $this->visit('/'); } /** @test */ public function it_verifies_the_current_page() { $this->visit('/some-page')->seePageIs('some-page'); } /** @test */ public function it_follows_links() { $this->visit('/page-1') ->click('Follow Me') ->andSee('You are on Page 2') ->onPage('/page-2'); } /** @test */ public function it_submits_forms() { $this->visit('page-with-form') ->submitForm('Submit', ['title' => 'Foo Title']) ->andSee('You entered Foo Title') ->onPage('/page-with-form-results'); // Another way to write it. $this->visit('page-with-form') ->type('Foo Title', '#title') ->press('Submit') ->see('You Entered Foo Title'); } /** @test */ public function it_verifies_information_in_the_database() { $this->visit('database-test') ->type('Testing', 'name') ->press('Save to Database') ->verifyInDatabase('things', ['name' => 'Testing']); } }
```

Notes

- Based on the official documentation
- Converted and extended by Mario Nava