**PHP Unit Testing Documentation**

**Creating Test**

To create a new test class, we can either create a new file manually or run the helpful Artisan make:test command provided by Laravel.

php artisan make:test UserTest

[The most important thing to notice here is the test prefix on the method name](https://semaphoreci.com/community/tutorials/getting-started-with-phpunit-in-laravel)

public function testExample(){

$response = $this->get('/');  
  
 $response->assertStatus(200);  
}

Like the Test suffix for class names, this test prefix tells PHPUnit what methods to run when testing. If you forget the test prefix, then PHPUnit will ignore the method.

There is a lot of information within this file, however, the most important section, for now, is the testsuite directory definition: phpunit.xml file that Laravel provides.

Basic code of phpunit.xml

*<?*xml version="1.0" encoding="UTF-8"*?>*<phpunit>  
 <testsuites>  
 <testsuite name="Unit">  
 <directory suffix="Test.php">./tests/Unit</directory>  
 </testsuite>  
  
 <testsuite name="Feature">  
 <directory suffix="Test.php">./tests/Feature</directory>  
 </testsuite>  
 </testsuites>  
</phpunit>

You can run your PHPUnit tests by running the phpunit command:

./vendor/bin/phpunit

Output:

PS C:\xampp\htdocs\prac\_nirob\blog> ./vendor/bin/phpunit

PHPUnit 9.5.26 by Sebastian Bergmann and contributors.

.... 4 / 4 (100%)

Time: 00:00.124, Memory: 18.00 MB

OK (4 tests, 4 assertions)

## Writing Basic Test

To help cover the basic assertions that PHPUnit provides, we will first create a basic class that provides some simple functionality.

Create a new file in your ./app/ directory called Box.php, and copy this example class:

Code of Box.php

namespace App;  
  
class Box{  
  
 */\*\*  
 \* @var array  
 \*/* protected $items = [];  
  
 */\*\*  
 \* Construct the box with the given items.  
 \* @param array $items  
 \*/* public function \_\_construct($items = []){  
 $this->items = $items;  
 }

*/\*\*  
 \* Check if the specified item is in the box.  
 \* @param string $item  
 \* @return bool  
 \*/* public function has($item){  
 return *in\_array*($item, $this->items);  
 }

*/\*\*  
 \* Remove an item from the box, or null if the box is empty.  
 \* @return string  
 \*/* public function takeOne(){  
 return *array\_shift*($this->items);  
 }  
  
  
 */\*\*  
 \* Retrieve all items from the box that start with the specified letter.  
 \* @param string $letter  
 \* @return array  
 \*/* public function startsWith($letter){  
 return *array\_filter*($this->items, function ($item) use ($letter) {  
 return *stripos*($item, $letter) === 0;  
 });  
 }  
  
}

Next, open your ./tests/Feature/UserTest.php class (that we created earlier), and remove the testExample method that was created by default. You should be left with an empty class.

class UserTest extends TestCase{  
   
  
}

**PHP Unit Assertions**

We will now use eight of the basic PHPUnit assertions to write tests for our Box class. [There are many assertions available](https://laravel.com/docs/6.x/http-tests#available-assertions), but the most common are:

1. assertTrue()
2. assertFalse()
3. assertEquals()
4. assertNull()
5. assertContains()
6. assertCount()
7. assertEmpty()
8. assertStatus()

## assertTrue() and assertFalse()

They are perfect for testing methods that return boolean values. In our Box class, we have a method called has($item), which returns true or false when the specified item is in the box or not.

So, to test/Feature/UserTest.php write a test for this in PHPUnit, we can do the following:

*# Import Box class*use App\Box;  
  
class UserTest extends TestCase  
{  
  
 *# Test function for Box class* public function testBoxContents(){  
 $box = new Box(['toy']);  
 $this->assertTrue($box->has('toy'));  
 $this->assertFalse($box->has('ball'));  
 }  
  
}

Note how we only pass a single parameter into the assertTrue() and assertFalse() methods, and it is the output of the has($item) method. If you run the ./vendor/bin/phpunit command now, you will notice the output includes:

PS C:\xampp\htdocs\prac\_nirob\blog> ./vendor/bin/phpunit

Time: 00:00.126, Memory: 18.00 MB

OK (4 tests, 5 assertions)

If you swap the assertFalse() for assertTrue() and run the phpunit command again

*# swap value the assertFalse() for assertTrue()*public function testBoxContents(){  
 $box = new Box(['toy']);  
 $this->assertTrue($box->has('ball'));  
 $this->assertFalse($box->has('toy'));  
  
}

The output will look like this:

PHPUnit 9.5.26 by Sebastian Bergmann and contributors.

...F 4 / 4 (100%)

Time: 00:00.131, Memory: 18.00 MB

There was 1 failure:

1) Tests\Feature\UserTest::testBoxContents

Failed asserting that false is true.

C:\xampp\htdocs\prac\_nirob\blog\tests\Feature\UserTest.php:26

phpvfscomposer://C:\xampp\htdocs\prac\_nirob\blog\vendor\phpunit\phpunit\phpunit:97

FAILURES!

Tests: 4, Assertions: 4, Failures: 1.

This tells us that the assertion on line tests\Feature\UserTest.php:26 failed to assert that a false value was true as we switched the assertFalse() for assertTrue().

Swap it back, and re-run PHPUnit. The tests should again pass, as we have fixed the broken test.

**assertEquals() and assertNull()**

assertEquals() is used to compare the actual value of the variable to the expected value. We want to use it to check if the value of the takeOne() function is an item that is currently in the box. As the takeOne() method returns a null value when the box is empty, we can use assertNull() to check for that too.

Unlike assertTrue(), assertFalse(), and assertNull(), assertEquals() takes two parameters. The first being the expected value, and the second being the actual value.

We can implement these assertions in our class as follows:

*# Test function for and assertNull(), assertEquals() Box class*public function testTakeOneFromTheBox(){  
 $box = new Box(['torch']);  
 $this->assertEquals('torch', $box->takeOne());  
 *// Null, now the box is empty* $this->assertNull($box->takeOne());  
}

Run the phpunit command, and you should see:

Time: 00:00.119, Memory: 18.00 MB

OK (4 tests, 5 assertions)

**assertContains(), assertCount(), and assertEmpty()**

Finally, we have three assertions that work with arrays.

Which we can use to check the startsWith($item) method in our Box class,

assertContains() asserts that an expected value exists within the provided array,

assertCount() asserts the number of items in the array matches the specified amount,

and assertEmpty() asserts that the provided array is empty.

We can implement tests for these like this:

*# tests/Feature/BasicTest.php*

public function testStartsWithALetter(){  
 $box = new Box(['toy', 'torch', 'ball', 'cat', 'tissue']);  
  
 $results = $box->startsWith('t');  
  
 *//$this->assertCount(3, $results);* $this->assertContains('toy', $results);  
 $this->assertContains('torch', $results);  
 $this->assertContains('tissue', $results);  
  
 *// Empty array if passed even* $this->assertEmpty($box->startsWith('s'));  
}

Save and run your tests again:

PHPUnit 9.5.26 by Sebastian Bergmann and contributors.

.... 4 / 4 (100%)

Time: 00:00.121, Memory: 18.00 MB

OK (4 tests, 7 assertions)

**Testing Application**

Unit testing each component in your application works in a lot of situations and should definitely be part of your development process

We previously created a new test file and we skipped the ./tests/Feature/ExampleTest.php file. Open it now, and it should look something like this:

*# tests/Feature/ExampleTest.php*public function testBasicTest(){  
 $response = $this->get('/');  
  
 $response->assertStatus(200);  
}

We can see the test in this case, it is very simple. Without any prior knowledge of how the test helpers work, we can assume it means something like this

php artisan serve

Let’s write our own test now, and take it one step further.

Edit the routes file located at routes/web.php. Add the following route below the Welcome route:

Route::*get*('/alpha', function () {  
 return view('alpha');  
});

For the sake of this tutorial, we will go for a Greek alphabet themed route.

Next, create the view template at ./resources/views/alpha.blade.php, and save some basic HTML with the Alpha keyword:

<!DOCTYPE html>  
<html lang="{{ str\_replace('\_', '-', app()->getLocale()) }}">  
<head>  
 <meta charset="utf-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <title>Alpha</title>  
</head>  
<body>  
<h1>This is the Alpha page</h1>  
</body>  
</html>

Now open it in your browser to ensure it is working as expected with: php artisan serve. Go to http://localhost:8000/alpha, and it should display a friendly “This is the Alpha page.” message.

Now that we have the template, we will create a new test. Run the make:test command:

php artisan make:test AlphaTest

public function testExample()  
{  
 $response = $this->get('/alpha');  
  
 $response->assertStatus(200);  
 $response->assertSee('Alpha');  
 $response->assertDontSee('Beta');  
}

Save it and run PHPUnit (./vendor/bin/phpunit),

**Some Test Example Code**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 # Test function for Box class  
 public function testBoxContents(){  
 $box = new Box(['toy']);  
 $this->assertTrue($box->has('toy'));  
 $this->assertFalse($box->has('ball'));  
 }  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 # swap value the assertFalse() for assertTrue()  
 public function testBoxContents(){  
 $box = new Box(['toy']);  
 $this->assertTrue($box->has('toy'));  
 $this->assertFalse($box->has('ball'));  
 }  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 # Test function for and assertNull(), assertEquals() Box class  
 public function testTakeOneFromTheBox(){  
 $box = new Box(['torch']);  
 $this->assertEquals('torch', $box->takeOne());  
 // Null, now the box is empty  
 $this->assertNull($box->takeOne());  
 }  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 # tests/Feature/BasicTest.php  
 public function testStartsWithALetter(){  
 $box = new Box(['toy', 'torch', 'ball', 'cat', 'tissue']);  
  
 $results = $box->startsWith('t');  
  
 //$this->assertCount(3, $results);  
 $this->assertContains('toy', $results);  
 $this->assertContains('torch', $results);  
 $this->assertContains('tissue', $results);  
  
 // Empty array if passed even  
 $this->assertEmpty($box->startsWith('s'));  
 }  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

**Writing Tests First**

**How do I run a particular unit test in Laravel?**

For run phpunit test in laravel by many ways ..

./vendor/bin/phpunit --filter methodName className pathTofile.php

./vendor/bin/phpunit --filter 'namespace\\directoryName\\className::methodName'

For test single class:

./vendor/bin/phpunit tests/Feature/UserTest.php

./vendor/bin/phpunit --filter tests/Feature/UserTest.php

./vendor/bin/phpunit --filter 'Tests\\Feature\\UserTest'

./vendor/bin/phpunit --filter 'UserTest'

For test single method:

./vendor/bin/phpunit --filter testExample

./vendor/bin/phpunit --filter 'Tests\\Feature\\UserTest::testExample'

./vendor/bin/phpunit --filter testExample UserTest tests/Feature/UserTest.php

For run tests from all class within namespace:

./vendor/bin/phpunit --filter 'Tests\\Feature'

**TDD Test-driven development (TDD)**

First of create a new test file which represent specific controller, I create like:

php artisan make:test PostCntrollerTest

Follow 3 approaches

1. Arrange / When
2. Act / Given
3. Assert / Then

Like this:

*/\*\* @test \*/*public function show\_list\_of\_post(){

*// Arrange / When  
   
 // Act / Given  
   
 // Assert Then*

}

So, then create a new post

*/\*\* @test \*/*public function show\_list\_of\_post(){  
 *// Arrange / When* Post::*factory*()->count(10)->create();  
  
 *// Act / Given* $posts = (new PostCntroller)->index();  
  
 *// Assert / Then* $this->assertEquals(15, $posts->count());  
  
}

Now run this test use this command.

./vendor/bin/phpunit

Output:

Error: Class 'Tests\Feature\Post' not found

That’s mean Post Model not created. Lest create a Post Model

php artisan make:model Post

Then, run the command: ./vendor/bin/phpunit

Now, Error: Class 'Database\Factories\PostFactory' not found

Because PostFactory not created.

For create PostFactory: php artisan make:factory PostFactory

Again, run command: ./vendor/bin/phpunit

Show again error,

PDOException: SQLSTATE[HY000] [1049] Unknown database 'laravel'

Database has not connected. So, please connect the database throe phpunit.xml file. And uncomment this line.

<server name="DB\_CONNECTION" value="sqlite"/>   
<server name="DB\_DATABASE" value=":memory:"/>

Then run and output again error:

PDOException: SQLSTATE[HY000]: General error: 1 no such table: posts

Create migration file: php artisan make:migration create\_posts\_table

Use use RefreshDatabase; in this method.

Create controller: php artisan make:controller PostCntroller

Write index method into controller:

class PostCntroller extends Controller  
{  
 function index(){  
 return Post::*all*();  
 }  
}

Now run, output:

Time: 00:00.194, Memory: 26.00 MB

OK (1 test, 1 assertion)

**Show all post:**

Test Controller:

class PostCntrollerTest extends TestCase{

use RefreshDatabase;  
 */\*\* @test \*/* public function show\_list\_of\_post(){  
 *// Arrange / When* Post::*factory*()->count(10)->create();

*// Act / Given* $posts = (new PostCntroller)->index();  
  
 *// Assert / Then* $this->assertEquals(10, $posts->count());  
  
 }  
}

Only controller:

class PostCntroller extends Controller{

function index(){  
 return Post::*all*();  
 }  
}

**Show single post:**

Test Controller:

*/\*\* @test \*/*public function show\_single\_post(){  
 *// Arrange / When* $post = Post::*factory*()->create();  
  
 *// Act / Given* $getPost = (new PostCntroller)->show($post->id);  
  
 *// Assert / Then* $this->assertEquals($post->id, $getPost->id);  
  
}

Only controller:

function show(int $id){  
 return Post::*findOrFail*($id);  
}

When we test our targeted data

Test Controller:

*/\*\* @test \*/*public function show\_single\_post(){  
 *// Arrange / When* $post = Post::*factory*()->create([  
 'title'=>'This is title',  
 'slug'=>'This is slug',  
 'body'=>'This is body'  
 ]);  
  
 *// Act / Given* $getPost = (new PostCntroller)->show($post->id);  
  
 *// Assert / Then* $this->assertEquals($post->id, $getPost->id);  
 $this->assertEquals("This is title", $getPost->title);  
 $this->assertEquals("This is slug", $getPost->slug);  
 $this->assertEquals("This is body", $getPost->body);  
  
}

Only controller:

function show(int $id){  
 return Post::*findOrFail*($id);  
}

**When our post id have not in database, in this case:**

Test Controller:

*/\*\* @test \*/*public function trows\_exception\_if\_wrong\_id\_pass(){  
 *// Arrange / When* Post::*factory*()->create();  
  
 *// Assert / Then* $this->expectException(ModelNotFoundException::class);  
  
 *// Act / Given* $posts = (new PostCntroller)->show(100);  
  
}

And go database>factory>PostFacotry and write:

public function definition()  
{  
 return [  
 'title'=>'This is title',  
 'slug'=>'This is slug',  
 'body'=>'This is body'  
 ];  
}

Now run output is:

Time: 00:00.249, Memory: 28.00 MB

OK (3 tests, 6 assertions)

**Write now we want to create a new post**

Sometimes we test this called is pre-assertion. We will do pre-assertion before real-assertion.

So, in this case we are check our database is empty or not? If is empty then create post, then test, and then ensure to post is created.

Test Controller:

*/\*\* @test \*/*public function create\_new\_post(){  
 *// Arrange* $this->assertDatabaseCount('posts',0);  
 $post = [  
 'title'=>'This is title',  
 'slug'=>'This is slug',  
 'body'=>'This is body'  
 ];  
  
 *// Act* (new PostCntroller)->create($post);  
  
 *// Assert* $this->assertDatabaseCount('posts',1);  
  
}

Only controller:

function create(array $post){  
 return Post::*create*($post);  
}

And Model:

class Post extends Model{  
 use HasFactory;  
  
 protected $guarded = [];  
}

Delete Post

Test Controller:

*/\*\* @test \*/*public function delete\_post(){  
 *// Arrange* $postOne = Post::*factory*()->create();  
  
 $postTwo = Post::*factory*()->create();  
  
 $this->assertDatabaseCount('posts',2);  
  
 *// Act* (new PostCntroller)->delete($postTwo->id);  
  
 *// Assert* $this->assertDatabaseCount('posts',1);  
  
}

Only controller:

function delete(int $id){  
 return Post::*destroy*($id);  
}