HTTP Tests

Introduction

Laravel provides a very fluent API for making HTTP requests to your application and examining the output. For example, take a look at the test defined below:

```php
<?php

namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Foundation\Testing\WithoutMiddleware;
use Tests\TestCase;

class ExampleTest extends TestCase
{
    /**
     * A basic test example.
     *
     * @return void
     */
    public function testBasicTest()
    {
        $response = $this->get('/');

        $response->assertStatus(200);
    }
}
```

The `get` method makes a `GET` request into the application, while the `assertStatus` method asserts that the returned response should have the given HTTP status code. In addition to this simple assertion, Laravel also contains a variety of assertions for inspecting the response headers, content, JSON structure, and more.

Customizing Request Headers

You may use the `withHeaders` method to customize the request's headers before it is sent to the application. This allows you to add any custom headers you would like to the request:

```php
<?php

class ExampleTest extends TestCase
{
    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testBasicExample()
    {
        $response = $this->withHeaders([
            'X-Header' => 'Value',
        ])->json('POST', '/user', ['name' => 'Sally']);

        $response
            ->assertStatus(201)
            ->assertJson([
                'created' => true,
            ]);
    }
}
```

The CSRF middleware is automatically disabled when running tests.

Debugging Responses

After making a test request to your application, the `dump` and `dumpHeaders` methods may be used to examine and debug the response contents:

```
<?php

namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Foundation\Testing\WithoutMiddleware;
use Tests\TestCase;

class ExampleTest extends TestCase
{
    /**
     * A basic test example.
     *
     * @return void
     */
    public function testBasicTest()
    {
        $response = $this->get('/');

        $response->dumpHeaders();

        $response->dump();
    }
}
```

Session / Authentication

Laravel provides several helpers for working with the session during HTTP testing. First, you may set the session data to a given array using the `withSession` method. This is useful for loading the session with data before issuing a request to your application:

```
<?php

class ExampleTest extends TestCase
{
    public function testApplication()
    {
        $response = $this->withSession(['foo' => 'bar'])
                         ->get('/');
    }
}
```

One common use of the session is for maintaining state for the authenticated user. The `actingAs` helper method provides a simple way to authenticate a given user as the current user. For example, we may use a model factory to generate and authenticate a user:

```
<?php

use App\User;

class ExampleTest extends TestCase
{
    public function testApplication()
    {
        $user = factory(User::class)->create();

        $response = $this->actingAs($user)
                         ->withSession(['foo' => 'bar'])
                         ->get('/');
    }
}
```

You may also specify which guard should be used to authenticate the given user by passing the guard name as the second argument to the `actingAs` method:

```
$this->actingAs($user, 'api')
```

Testing JSON APIs

Laravel also provides several helpers for testing JSON APIs and their responses. For example, the `json`, `getJson`, `postJson`, `putJson`, `patchJson`, `deleteJson`, and `optionsJson` methods may be used to issue JSON requests with various HTTP verbs. You may also easily pass data and headers to these methods. To get started, let's write a test to make a `POST` request to `/user` and assert that the expected data was

returned:

```php
<?php

class ExampleTest extends TestCase
{
    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testBasicExample()
    {
        $response = $this->postJson('/user', ['name' => 'Sally']);

        $response
            ->assertStatus(201)
            ->assertJson([
                'created' => true,
            ]);
    }
}
```

The `assertJson` method converts the response to an array and utilizes `PHPUnit::assertArraySubset` to verify that the given array exists within the JSON response returned by the application. So, if there are other properties in the JSON response, this test will still pass as long as the given fragment is present.

Verifying An Exact JSON Match

If you would like to verify that the given array is an **exact** match for the JSON returned by the application, you should use the `assertExactJson` method:

```php
<?php

class ExampleTest extends TestCase
{
    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testBasicExample()
    {
        $response = $this->json('POST', '/user', ['name' => 'Sally']);

        $response
            ->assertStatus(201)
            ->assertExactJson([
                'created' => true,
            ]);
    }
}
```

Verifying JSON Paths

If you would like to verify that the JSON response contains some given data at a specified path, you should use the `assertJsonPath` method:

```php
<?php

class ExampleTest extends TestCase
{
    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testBasicExample()
    {
        $response = $this->json('POST', '/user', ['name' => 'Sally']);

        $response
            ->assertStatus(201)
            ->assertJsonPath('team.owner.name', 'foo')
    }
}
```

Testing File Uploads

The `Illuminate\Http\UploadedFile` class provides a `fake` method which may be used

to generate dummy files or images for testing. This, combined with the `Storage` facade's `fake` method greatly simplifies the testing of file uploads. For example, you may combine these two features to easily test an avatar upload form:

```php
<?php

namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Foundation\Testing\WithoutMiddleware;
use Illuminate\Http\UploadedFile;
use Illuminate\Support\Facades\Storage;
use Tests\TestCase;

class ExampleTest extends TestCase
{
    public function testAvatarUpload()
    {
        Storage::fake('avatars');

        $file = UploadedFile::fake()->image('avatar.jpg');

        $response = $this->json('POST', '/avatar', [
            'avatar' => $file,
        ]);

        // Assert the file was stored...
        Storage::disk('avatars')->assertExists($file->hashName());

        // Assert a file does not exist...
        Storage::disk('avatars')->assertMissing('missing.jpg');
    }
}
```

Fake File Customization

When creating files using the `fake` method, you may specify the width, height, and size of the image in order to better test your validation rules:

```php
UploadedFile::fake()->image('avatar.jpg', $width, $height)->size(100);
```

In addition to creating images, you may create files of any other type using the `create` method:

```php
UploadedFile::fake()->create('document.pdf', $sizeInKilobytes);
```

Available Assertions

Response Assertions

Laravel provides a variety of custom assertion methods for your PHPUnit tests. These assertions may be accessed on the response that is returned from the `json`, `get`, `post`, `put`, and `delete` test methods:

assertCookie
assertCookieExpired
assertCookieNotExpired
assertCookieMissing
assertDontSee
assertDontSeeText
assertExactJson
assertForbidden
assertHeader
assertHeaderMissing
assertJson
assertJsonCount
assertJsonFragment
assertJsonMissing
assertJsonMissingExact
assertJsonMissingValidationErrors
assertJsonPath
assertJsonStructure
assertJsonValidationErrors
assertLocation
assertNoContent
assertNotFound
assertOk
assertPlainCookie
assertRedirect
assertSee
assertSeeInOrder
assertSeeText
assertSeeTextInOrder
assertSessionHas
assertSessionHasInput
assertSessionHasAll
assertSessionHasErrors
assertSessionHasErrorsIn
assertSessionHasNoErrors
assertSessionDoesntHaveErrors
assertSessionMissing
assertStatus
assertSuccessful
assertUnauthorized
assertViewHas
assertViewHasAll
assertViewIs
assertViewMissing

assertCookie

Assert that the response contains the given cookie:

```
$response->assertCookie($cookieName, $value = null);
```

assertCookieExpired

Assert that the response contains the given cookie and it is expired:

```
$response->assertCookieExpired($cookieName);
```

assertCookieNotExpired

Assert that the response contains the given cookie and it is not expired:

```
$response->assertCookieNotExpired($cookieName);
```

assertCookieMissing

Assert that the response does not contains the given cookie:

```
$response->assertCookieMissing($cookieName);
```

assertDontSee

Assert that the given string is not contained within the response:

```
$response->assertDontSee($value);
```

assertDontSeeText

Assert that the given string is not contained within the response text:

```
$response->assertDontSeeText($value);
```

assertExactJson

Assert that the response contains an exact match of the given JSON data:

```
$response->assertExactJson(array $data);
```

assertForbidden

Assert that the response has a forbidden status code:

```
$response->assertForbidden();
```

assertHeader

Assert that the given header is present on the response:

```
$response->assertHeader($headerName, $value = null);
```

assertHeaderMissing

Assert that the given header is not present on the response:

```
$response->assertHeaderMissing($headerName);
```

assertJson

Assert that the response contains the given JSON data:

```
$response->assertJson(array $data, $strict = false);
```

assertJsonCount

Assert that the response JSON has an array with the expected number of items at the given key:

```
$response->assertJsonCount($count, $key = null);
```

assertJsonFragment

Assert that the response contains the given JSON fragment:

```
$response->assertJsonFragment(array $data);
```

assertJsonMissing

Assert that the response does not contain the given JSON fragment:

```
$response->assertJsonMissing(array $data);
```

assertJsonMissingExact

Assert that the response does not contain the exact JSON fragment:

```
$response->assertJsonMissingExact(array $data);
```

assertJsonMissingValidationErrors

Assert that the response has no JSON validation errors for the given keys:

```
$response->assertJsonMissingValidationErrors($keys);
```

assertJsonPath

Assert that the response contains the given data at the specified path:

```
$response->assertJsonPath($path, array $data, $strict = false);
```

assertJsonStructure

Assert that the response has a given JSON structure:

```
$response->assertJsonStructure(array $structure);
```

assertJsonValidationErrors

Assert that the response has the given JSON validation errors:

```
$response->assertJsonValidationErrors(array $data);
```

assertLocation

Assert that the response has the given URI value in the `Location` header:

```
$response->assertLocation($uri);
```

assertNoContent

Assert that the response has the given status code and no content.

```
$response->assertNoContent($status = 204);
```

assertNotFound

Assert that the response has a not found status code:

```
$response->assertNotFound();
```

assertOk

Assert that the response has a 200 status code:

```
$response->assertOk();
```

assertPlainCookie

Assert that the response contains the given cookie (unencrypted):

```
$response->assertPlainCookie($cookieName, $value = null);
```

assertRedirect

Assert that the response is a redirect to a given URI:

```
$response->assertRedirect($uri);
```

assertSee

Assert that the given string is contained within the response:

```
$response->assertSee($value);
```

assertSeeInOrder

Assert that the given strings are contained in order within the response:

```
$response->assertSeeInOrder(array $values);
```

assertSeeText

Assert that the given string is contained within the response text:

```
$response->assertSeeText($value);
```

assertSeeTextInOrder

Assert that the given strings are contained in order within the response text:

```
$response->assertSeeTextInOrder(array $values);
```

assertSessionHas

Assert that the session contains the given piece of data:

```
$response->assertSessionHas($key, $value = null);
```

assertSessionHasInput

Assert that the session has a given value in the flashed input array:

```
$response->assertSessionHasInput($key, $value = null);
```

assertSessionHasAll

Assert that the session has a given list of values:

```
$response->assertSessionHasAll(array $data);
```

assertSessionHasErrors

Assert that the session contains an error for the given field:

```
$response->assertSessionHasErrors(array $keys, $format = null, $errorBag = 'defaul
```

assertSessionHasErrorsIn

Assert that the session has the given errors:

```
$response->assertSessionHasErrorsIn($errorBag, $keys = [], $format = null);
```

assertSessionHasNoErrors

Assert that the session has no errors:

```
$response->assertSessionHasNoErrors();
```

assertSessionDoesntHaveErrors

Assert that the session has no errors for the given keys:

```
$response->assertSessionDoesntHaveErrors($keys = [], $format = null, $errorBag = '
```

assertSessionMissing

Assert that the session does not contain the given key:

```
$response->assertSessionMissing($key);
```

assertStatus

Assert that the response has a given code:

```
$response->assertStatus($code);
```

assertSuccessful

Assert that the response has a successful (200) status code:

```
$response->assertSuccessful();
```

assertUnauthorized

Assert that the response has an unauthorized (401) status code:

```
$response->assertUnauthorized();
```

assertViewHas

Assert that the response view was given a piece of data:

```
$response->assertViewHas($key, $value = null);
```

assertViewHasAll

Assert that the response view has a given list of data:

```
$response->assertViewHasAll(array $data);
```

assertViewIs

Assert that the given view was returned by the route:

```
$response->assertViewIs($value);
```

assertViewMissing

Assert that the response view is missing a piece of bound data:

```
$response->assertViewMissing($key);
```

Authentication Assertions

Laravel also provides a variety of authentication related assertions for your PHPUnit tests:

Method	Description
`$this->assertAuthenticated($guard = null);`	Assert that the user is authenticated.
`$this->assertGuest($guard = null);`	Assert that the user is not authenticated.
`$this->assertAuthenticatedAs($user, $guard = null);`	Assert that the given user is authenticated.
`$this->assertCredentials(array $credentials, $guard = null);`	Assert that the given credentials are valid.
`$this->assertInvalidCredentials(array $credentials, $guard = null);`	Assert that the given credentials are invalid.

Become a Laravel Partner

Laravel Partners are elite shops providing top-notch Laravel development and consulting. Each of our partners can help you craft a beautiful, well-architected project.

Our Partners

Laravel

Highlights

- Release Notes
- Getting Started
- Routing
- Blade Templates
- Authentication
- Authorization
- Artisan Console
- Database
- Eloquent ORM
- Testing

Resources

- Laracasts
- Laravel News
- Laracon
- Laracon EU
- Laracon AU
- Jobs
- Certification
- Forums

Partners

- Vehikl
- Tighten Co.
- Kirschbaum
- Byte 5
- 64Robots
- Cubet
- DevSquad
- Ideil
- Cyber-Duck
- ABOUT YOU

Ecosystem

- Vapor
- Forge
- Envoyer
- Horizon
- Lumen
- Nova
- Echo
- Valet
- Mix
- Spark

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in most web projects.

Become A Partner

Cashier

Homestead

Dusk

Passport

Scout

Socialite