Course - Laravel Framework

HTTP Test

Laravel cung cấp một API rất linh hoạt để thực hiện các yêu cầu HTTP tới ứng dụng của bạn và kiểm tra các phản hồi.

Tags: HTTP Test, laravel

Giới thiệu

Laravel cung cấp một API rất linh hoạt để thực hiện các yêu cầu HTTP tới ứng dụng của bạn và kiểm tra các phản hồi. Ví dụ: hãy xem kiểm tra tính năng được khai báo bên dưới:

```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 test_a_basic_request()
  {
    $response = $this->get('/');

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

Phương thức get thực hiện một yêu cầu GET vào ứng dụng, trong khi phương thức khẳng địnhStatus khẳng định rằng phản hồi được trả về phải có mã trạng thái HTTP đã cho. Ngoài xác nhận đơn giản này, Laravel cũng chứa nhiều xác nhận khác nhau để kiểm tra tiêu đề phản hồi, nội dung, cấu trúc JSON và hơn thế nữa.

Tạo các yêu cầu request gửi đi

Để đưa ra yêu cầu request đối với ứng dụng của mình, bạn có thể gọi các phương thức get, post, put, patch hoặc delete trong bài kiểm tra của bạn. Các phương thức này không thực sự

đưa ra một yêu cầu HTTP "thực" nào cho ứng dụng của bạn. Thay vào đó, toàn bộ yêu cầu mạng được mô phỏng nội bộ.

Thay vì trả về một đối tượng `Illuminate\Http\Response`, các phương thức yêu cầu kiểm tra trả về một đối tượng `Illuminate\Testing\TestResponse`, cung cấp nhiều xác nhận hữu ích cho phép bạn kiểm tra phản hồi của ứng dụng:

```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 test_a_basic_request()
  {
    $response = $this->get('/');

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

Nói chung, mỗi bài kiểm tra của bạn chỉ nên đưa ra một yêu cầu đối với ứng dụng của bạn. Hành vi không mong muốn có thể xảy ra nếu nhiều yêu cầu được thực hiện trong một phương pháp thử nghiệm.

> Để thuận tiện, middleware CSRF sẽ tự động bị vô hiệu hóa khi chạy thử nghiệm.

Tùy biến header của request

Bạn có thể sử dụng phương thức `withHeaders` để tùy biến tiêu đề của yêu cầu request trước khi nó được gửi đến ứng dụng. Phương pháp này cho phép bạn thêm bất kỳ tiêu đề tùy chỉnh nào bạn muốn vào yêu cầu:

```php
<?php

namespace Tests\Feature;
use Tests\TestCase;

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

    $response->assertStatus(201);
  }
}
```

Các Cookie

Bạn có thể sử dụng phương thức `withCookie` hoặc `withCookies` để đặt giá trị cookie trước khi đưa ra yêu cầu request. Phương thức `withCookie` chấp nhận tên và giá trị của cookie làm hai đối số của nó, trong khi phương thức `withCookies` chấp nhận một mảng các cặp tên/giá trị:

```php
<?php

namespace Tests\Feature;
use Tests\TestCase;
```

```
class ExampleTest extends TestCase
{
  public function test_interacting_with_cookies()
  {
    $response = $this->withCookie('color', 'blue')->get('/');

    $response = $this->withCookies([
      'color' => 'blue',
      'name' => 'Taylor',
    ])->get('/');
  }
}
```

Session / Xác thực người dụng

Laravel cung cấp một số hàm trợ giúp để tương tác với session trong suốt quá trình kiểm tra HTTP. Đầu tiên, bạn có thể đặt dữ liệu session thành một mảng nhất định bằng phương thức `withSession`. Nó hữu ích để tải session với dữ liệu trước khi đưa ra yêu cầu cho ứng dụng của bạn:

```
<?php

namespace Tests\Feature;
use Tests\TestCase;

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

Session của Laravel thường được sử dụng để duy trì trạng thái cho người dùng hiện đã được xác thực. Do đó, phương thức trợ giúp `actingAs` sẽ cho cách thức đơn giản để xác thực một người dùng nào đó là người dùng hiện tại. Ví dụ: chúng ta có thể sử dụng model

factory để tạo và xác thực người dùng:

```php
<?php

namespace Tests\Feature;

use App\Models\User;
use Tests\TestCase;

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

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

Bạn cũng có thể mô tả bảo vệ guard nào nên được sử dụng để xác thực người dùng đã cho bằng cách truyền tên bảo vệ guard làm đối số thứ hai cho phương thức `actingAs`. Bộ bảo vệ guard được cung cấp cho phương thức `actingAs` cũng sẽ trở thành bộ bảo vệ mặc định trong suốt thời gian thử nghiệm:

```php
$this->actingAs($user, 'web')
```

Gỡ rối các phản hồi response

Sau khi đưa ra yêu cầu kiểm tra đối với ứng dụng của bạn, các phương thức `dump`, `dumpHeaders` và `dumpSession` có thể được sử dụng để kiểm tra và gỡ lỗi nội dung phản hồi response:

```php
<?php
```

```php
namespace Tests\Feature;
use Tests\TestCase;

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

Ngoài ra, bạn có thể sử dụng các phương thức `dd`, `ddHeaders` và `ddSession` để dump thông tin về phản hồi response và sau đó dừng thực thi:

```php
<?php

namespace Tests\Feature;
use Tests\TestCase;

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

```
        $response->ddHeaders();
        $response->ddSession();
        $response->dd();
    }
}
```

Xử lý exception

Đôi khi bạn có thể muốn kiểm tra xem ứng dụng của mình có đang đưa ra một exception cụ thể hay không. Để đảm bảo rằng exception không bị trình xử lý exception của Laravel bắt và trả về dưới dạng phản hồi HTTP response, thì bạn có thể gọi phương thức `withoutExceptionHandling` trước khi thực hiện yêu cầu của mình:

```
$response = $this->withoutExceptionHandling()->get('/');
```

Ngoài ra, nếu bạn muốn đảm bảo rằng ứng dụng của mình không sử dụng các tính năng đã *không được sử dụng* (deprecated) bởi ngôn ngữ PHP hoặc các thư viện mà ứng dụng của bạn đang sử dụng, bạn có thể gọi phương thức `withoutDeprecationHandling` trước khi đưa ra yêu cầu của mình. Khi tính năng xử lý *không dùng nữa* (deprecate) bị vô hiệu hóa, các cảnh báo *không dùng nữa* (deprecate) sẽ được chuyển đổi thành exception, do đó khiến thử nghiệm của bạn không thành công:

```
$response = $this->withoutDeprecationHandling()->get('/');
```

Kiểm tra JSON API

Laravel cũng cung cấp một số hàm trợ giúp để kiểm tra các API JSON và phản hồi response của chúng. Ví dụ: các phương thức `json`, `getJson`, `postJson`, `putJson`, `patchJson`, `deleteJson` và `optionsJson` có thể được sử dụng để đưa ra các yêu cầu JSON với các động từ HTTP khác nhau. Bạn cũng có thể dễ dàng truyền dữ liệu và tiêu đề cho các phương thức này. Để bắt đầu, hãy viết một bài kiểm tra để thực hiện một yêu cầu `POST` tới `/api/user` và xác nhận rằng dữ liệu JSON mong đợi đã được trả về:

```
<?php

namespace Tests\Feature;
```

```php
use Tests\TestCase;

class ExampleTest extends TestCase
{
  /**
   * A basic functional test example.
   *
   * @return void
   */
  public function test_making_an_api_request()
  {
    $response = $this->postJson('/api/user', ['name' => 'Sally']);
    $response->assertStatus(201)
      ->assertJson([
        'created' => true,
      ]);
  }
}
```

Ngoài ra, dữ liệu phản hồi JSON có thể được truy cập dưới dạng các biến mảng trên phản hồi, giúp bạn thuận tiện trong việc kiểm tra các giá trị riêng lẻ được trả về trong phản hồi JSON:

```php
$this->assertTrue($response['created']);
```

> Phương thức assertJson giúp chuyển đổi phản hồi response thành một mảng và vận dụng `PHPUnit::assertArraySubnet` để xác minh rằng mảng đã cho tồn tại trong phản hồi JSON response được ứng dụng trả về. Vì vậy, nếu có các thuộc tính khác trong phản hồi JSON response, thì bài kiểm tra này sẽ vẫn vượt qua miễn là có phân đoạn đã cho.

Khẳng định JSON khớp tuyệt đối

Như đã đề cập trước đây, phương thức `assertJson` có thể được sử dụng để khẳng định rằng một phân đoạn JSON tồn tại trong phản hồi JSON response. Nếu bạn muốn xác minh rằng một mảng nhất định khớp chính xác với JSON mà ứng dụng của bạn trả về, thì bạn nên sử dụng phương thức `assertExactJson`:

```php
<?php

namespace Tests\Feature;
use Tests\TestCase;

class ExampleTest extends TestCase
{
  /**
   * A basic functional test example.
   *
   * @return void
   */
  public function test_asserting_an_exact_json_match()
  {
    $response = $this->postJson('/user', ['name' => 'Sally']);
    $response->assertStatus(201)
      ->assertExactJson([
        'created' => true,
      ]);
  }
}
```

Khẳng định trên đường dẫn JSON

Nếu bạn muốn xác minh rằng phản hồi JSON response chứa dữ liệu đã cho tại một đường dẫn được chỉ định, thì bạn nên sử dụng phương thức `assertJsonPath`:

```php
<?php

namespace Tests\Feature;
use Tests\TestCase;

class ExampleTest extends TestCase
{
  /**
   * A basic functional test example.
```

```
     *
     * @return void
     */
    public function test_asserting_a_json_paths_value()
    {
        $response = $this->postJson('/user', ['name' => 'Sally']);
        $response->assertStatus(201)
            ->assertJsonPath('team.owner.name', 'Darian');
    }
}
```

Phương thức `assertJsonPath` cũng chấp nhận một hàm xử lý, có thể được sử dụng để xác định một cách linh động xem có nên truyền hay không:

```
$response->assertJsonPath('team.owner.name', fn ($name) => strlen($name) >= 3);
```

Kiểm định linh hoạt JSON

Laravel cũng cung cấp một phương pháp hay để kiểm tra linh hoạt các phản hồi JSON response của ứng dụng của bạn. Để bắt đầu, hãy truyền một hàm xử lý vào phương thức `assertJson`. Hàm xử lý này sẽ được gọi với một đối tượng của `Illuminate\Testing\Fluent\AssertableJson` có thể được sử dụng để xác nhận đối với JSON đã được ứng dụng của bạn trả về. Phương thức `where` có thể được sử dụng để xác nhận đối với một thuộc tính cụ thể của JSON, trong khi phương thức `missing` có thể được sử dụng để khẳng định rằng một thuộc tính cụ thể bị thiếu trong JSON:

```
use Illuminate\Testing\Fluent\AssertableJson;

/**
 * A basic functional test example.
 *
 * @return void
 */
public function test_fluent_json()
{
    $response = $this->getJson('/users/1');
```

```
  $response
    ->assertJson(fn (AssertableJson $json) =>
      $json->where('id', 1)
        ->where('name', 'Victoria Faith')
        ->missing('password')
        ->etc()
    );
}
```

Tìm hiểu phương thức `etc`

Trong ví dụ trên, bạn có thể nhận thấy rằng chúng tôi đã gọi phương thức `etc` ở cuối chuỗi xác nhận của chúng ta. Phương thức này thông báo cho Laravel biết rằng có thể có các thuộc tính khác hiện diện trên đối tượng JSON. Nếu phương thức `etc` không được sử dụng, quá trình kiểm tra sẽ không thành công nếu các thuộc tính khác mà bạn không thực hiện xác nhận tồn tại trên đối tượng JSON.

Mục đích đằng sau hành vi này là để bảo vệ bạn khỏi việc vô tình tiết lộ thông tin nhạy cảm trong các phản hồi JSON response của bạn bằng cách buộc bạn phải đưa ra khẳng định rõ ràng chống lại thuộc tính hoặc cho phép rõ ràng các thuộc tính bổ sung thông qua phương thức `etc`.

Xác nhận Có mặt/Vắng mặt của thuộc tính attribute

Để khẳng định rằng một thuộc tính có hoặc không có, bạn có thể sử dụng các phương thức `has` và `missing`:

```
$response->assertJson(fn (AssertableJson $json) =>
  $json->has('data')
    ->missing('message')
);
```

Ngoài ra, các phương thức `hasAll` và `missingAll` cho phép xác nhận sự hiện diện hoặc vắng mặt đồng thời của nhiều thuộc tính attribute:

```
$response->assertJson(fn (AssertableJson $json) =>
  $json->hasAll('status', 'data')
```

```
    ->missingAll('message', 'code')
);
```

Bạn có thể sử dụng phương thức `hasAny` để xác định xem có ít nhất một trong danh sách các thuộc tính attribute nào đó hay không:

```
$response->assertJson(fn (AssertableJson $json) =>
  $json->has('status')
    ->hasAny('data', 'message', 'code')
);
```

Khẳng định đối với lại các JSON collection

Thông thường, route của bạn sẽ trả về một phản hồi JSON response chứa nhiều mục, chẳng hạn như nhiều người dùng:

```
Route::get('/users', function () {
  return User::all();
});
```

Trong những tình huống này, chúng ta có thể sử dụng phương thức `has` của đối tượng JSON để đưa ra xác nhận đối với lại những người dùng có trong phản hồi response. Ví dụ: hãy khẳng định rằng phản hồi JSON chứa ba người dùng. Tiếp theo, chúng tôi sẽ đưa ra một số xác nhận về người dùng đầu tiên trong bộ sưu tập bằng phương thức `first`. Phương thức `first` chấp nhận một hàm xử lý nhận được một chuỗi JSON có thể khẳng định khác mà chúng ta có thể sử dụng để thực hiện xác nhận về đối tượng đầu tiên trong bộ sưu tập JSON collection:

```
$response->assertJson(fn (AssertableJson $json) =>
  $json->has(3)
    ->first(fn ($json) =>
      $json->where('id', 1)
        ->where('name', 'Victoria Faith')
        ->missing('password')
        ->etc()
      )
```

```
);
```

Phân phạm vi xác nhận JSON collection

Đôi khi, các route của ứng dụng của bạn sẽ trả về các bộ sưu tập JSON collection được gán các khóa có tên:

```
Route::get('/users', function () {
  return [
    'meta' => [...],
    'users' => User::all(),
  ];
})
```

Khi kiểm tra các route này, bạn có thể sử dụng phương thức `has` để xác nhận số lượng mục trong collection. Ngoài ra, bạn có thể sử dụng phương thức `has` để xác định phạm vi xâu chuỗi các xác nhận assertion:

```
$response->assertJson(fn (AssertableJson $json) =>
  $json->has('meta')
    ->has('users', 3)
    ->has('users.0', fn ($json) =>
      $json->where('id', 1)
        ->where('name', 'Victoria Faith')
        ->missing('password')
        ->etc()
      )
);
```

Tuy nhiên, thay vì thực hiện hai lệnh gọi riêng biệt tới phương thức `has` để khẳng định đối với lại collection `users`, bạn có thể thực hiện một cuộc gọi đơn lẻ mà sẻ cung cấp một hàm xử lý làm tham số thứ ba của nó. Khi làm như vậy, hàm xử lý sẽ tự động được gọi và phân phạm vi đến mục đầu tiên trong bộ sưu tập collection:

```
$response->assertJson(fn (AssertableJson $json) =>
  $json->has('meta')
```

```
    ->has('users', 3, fn ($json) =>
      $json->where('id', 1)
        ->where('name', 'Victoria Faith')
        ->missing('password')
        ->etc()
      )
);
```

Xác nhận các kiểu JSON

Bạn có thể chỉ muốn khẳng định rằng các thuộc tính trong phản hồi JSON thuộc một loại nào đó. Class `Illuminate\Testing\Fluent\AssertableJson` sẽ cung cấp các phương thức `whereType` và `whereAllType` để thực hiện điều đó:

```
$response->assertJson(fn (AssertableJson $json) =>
  $json->whereType('id', 'integer')
    ->whereAllType([
      'users.0.name' => 'string',
      'meta' => 'array'
    ])
);
```

Bạn có thể chỉ định nhiều loại bằng cách sử dụng ký tự `|` hoặc truyền một mảng kiểu làm tham số thứ hai cho phương thức `whereType`. Việc xác nhận sẽ thành công nếu giá trị phản hồi là bất kỳ loại nào được liệt kê:

```
$response->assertJson(fn (AssertableJson $json) =>
  $json->whereType('name', 'string|null')
    ->whereType('id', ['string', 'integer'])
);
```

Phương thức `whereType` và `whereAllType` sẽ nhận dạng các kiểu sau: `string`, `integer`, `double`, `boolean`, `array` và `null`.

Kiểm định upload các tập tin

Lớp Illuminate \ Http \ UploadedFile cung cấp một phương thức giả có thể được sử dụng để tạo tệp hoặc hình ảnh giả để kiểm tra. Điều này, kết hợp với phương pháp giả mạo của mặt tiền Storage, giúp đơn giản hóa đáng kể việc kiểm tra tải lên tệp. Ví dụ: bạn có thể kết hợp hai tính năng này để dễ dàng kiểm tra biểu mẫu tải lên hình đại diện:

```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 test_avatars_can_be_uploaded()
  {
    Storage::fake('avatars');
    $file = UploadedFile::fake()->image('avatar.jpg');
    $response = $this->post('/avatar', [
      'avatar' => $file,
    ]);

    Storage::disk('avatars')->assertExists($file->hashName());
  }
}
```

Nếu bạn muốn khẳng định rằng một tệp nhất định không tồn tại, bạn có thể sử dụng phương thức khẳng địnhMissing được cung cấp bởi mặt tiền Storage:

```php
Storage::fake('avatars');

// ...

Storage::disk('avatars')->assertMissing('missing.jpg');
```

Tùy biến các tập tin giả mạo

Khi tạo tập tin bằng phương thức `fake` do class `UploadedFile` cung cấp, bạn có thể chỉ định chiều rộng, chiều cao và kích thước của hình ảnh (tính bằng kilobyte) để kiểm tra tốt hơn các quy tắc xác thực ứng dụng của bạn:

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

Ngoài việc tạo hình ảnh, bạn có thể tạo tập tin thuộc bất kỳ loại nào khác bằng cách sử dụng phương thức `create`:

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

Nếu cần, bạn có thể truyền đối số `$mimeType` vào phương thức để xác định rõ ràng kiểu MIME sẽ được tập tin trả về:

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

Kiểm tra giao diện hiển thị

Laravel cũng cho phép bạn hiển thị một giao diện mà không cần thực hiện một yêu cầu HTTP request mô phỏng cho ứng dụng. Để thực hiện điều này, bạn có thể gọi phương thức `view` trong bài kiểm tra của bạn. Phương thức `view` chấp nhận tên giao diện và một mảng dữ liệu tùy chọn. Phương thức này trả về một đối tượng của `Illuminate\Testing\TestView`, nó cung cấp một số phương thức để xác nhận assertion một cách thuận tiện về nội dung của giao diện:

```php
<?php

namespace Tests\Feature;
use Tests\TestCase;

class ExampleTest extends TestCase
{
  public function test_a_welcome_view_can_be_rendered()
  {
    $view = $this->view('welcome', ['name' => 'Taylor']);
    $view->assertSee('Taylor');
  }
}
```

Class `TestView` cung cấp các phương thức khẳng định assertion sau: assertSee, assertSeeInOrder, assertSeeText, assertSeeTextInOrder, assertDontSee và assertDontSeeText.

Nếu cần, bạn có thể nhận được nội dung hiển thị thô, được render bằng cách truyền đối tượng `TestView` thành một chuỗi câu:

```php
$contents = (string) $this->view('welcome');
```

Chia sẽ lỗi

Một số giao diện hiển thị có thể phụ thuộc vào các lỗi được chia sẻ trong túi lỗi chung (global error bag) do Laravel cung cấp. Để ngăn chặn túi lỗi với các thông báo lỗi, bạn có thể sử dụng phương thức `withViewErrors`:

```php
$view = $this->withViewErrors([
  'name' => ['Please provide a valid name.']
])->view('form');

$view->assertSee('Please provide a valid name.');
```

Trình bày blade và các component

Nếu cần, bạn có thể sử dụng phương thức `blade` để đánh giá và render chuỗi câu Blade thô. Giống như phương thức `view`, phương thức `blade` trả về một đối tượng của `Illuminate\Testing\TestView`:

```
$view = $this->blade(
  '<x-component :name="$name" />',
  ['name' => 'Taylor']
);

$view->assertSee('Taylor');
```

Bạn có thể sử dụng phương thức component để đánh giá và hiển thị một thành phần Blade. Giống như phương thức `view`, phương thức `component` sẽ trả về một đối tượng của `Illuminate\Testing\TestView`:

```
$view = $this->component(Profile::class, ['name' => 'Taylor']);

$view->assertSee('Taylor');
```

Các phương thức assertions có sẵn

Assertion phản hồi response

Class `Illuminate\Testing\TestResponse` của Laravel cung cấp nhiều phương pháp xác nhận assertion tùy biến mà bạn có thể sử dụng khi kiểm tra ứng dụng của mình. Các xác nhận assertion này có thể được truy cập vào phản hồi được trả về bởi các phương thức kiểm tra `json`, `get`, `post`, `put` và `delete`:

`assertCookie`

Khẳng định rằng phản hồi có chứa cookie đã cho:

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

`assertCookieExpired`

Khẳng định rằng phản hồi có chứa cookie đã cho và nó đã hết hạn:

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

assertCookieNotExpired

Khẳng định rằng phản hồi có chứa cookie đã cho và nó chưa hết hạn:

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

assertCookieMissing

Khẳng định rằng phản hồi không chứa cookie đã cho:

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

assertCreated

Khẳng định rằng phản hồi có mã trạng thái HTTP 201:

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

assertDontSee

Khẳng định rằng chuỗi đã cho không chứa trong phản hồi do ứng dụng trả về. Xác nhận assertion này sẽ tự động thoát khỏi chuỗi câu đã cho trừ khi bạn truyền đối số thứ hai giá trị `false`:

```
$response->assertDontSee($value, $escaped = true);
```

assertDontSeeText

Khẳng định rằng chuỗi câu đã cho không chứa trong văn bản phản hồi. Xác nhận assertion này sẽ tự động thoát khỏi chuỗi câu đã cho trừ khi bạn truyền đối số thứ hai giá trị `false`. Phương thức này sẽ truyền nội dung phản hồi đến hàm `strip_tags` PHP trước khi thực hiện khẳng định assertion:

```php
$response->assertDontSeeText($value, $escaped = true);
```

assertDownload

Khẳng định rằng phản hồi là "tải xuống". Thông thường, điều này có nghĩa là route được gọi mà trả về phản hồi đã trả về phản hồi `Response::download`, `BinaryFileResponse` hoặc `Storage::download`:

```php
$response->assertDownload();
```

Nếu muốn, bạn có thể khẳng định rằng tập tin có thể tải xuống đã được gán một tên tập tin nào đó:

```php
$response->assertDownload('image.jpg');
```

assertExactJson

Khẳng định rằng phản hồi chứa chính xác dữ liệu JSON phù hợp:

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

assertHeader

Khẳng định rằng tiêu đề và giá trị đã cho có trên phản hồi:

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

assertHeaderMissing

Khẳng định rằng tiêu đề đã cho không có trong phản hồi:

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

assertJson

Khẳng định rằng phản hồi chứa dữ liệu JSON đã cho:

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

Phương thức `assertJson` chuyển đổi phản hồi thành một mảng và sử dụng `PHPUnit::assertArraySubset` để xác minh rằng mảng đã cho tồn tại trong phản hồi JSON response được ứng dụng trả về. Vì vậy, nếu có các thuộc tính khác trong phản hồi JSON, thì bài kiểm tra này sẽ vẫn vượt qua miễn là có phân đoạn đã cho.

assertJsonCount

Khẳng định rằng JSON phản hồi có một mảng với số lượng mục dự kiến tại khóa đã cho:

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

assertJsonFragment

Khẳng định rằng phản hồi chứa dữ liệu JSON đã cho ở bất kỳ đâu trong phản hồi:

```
Route::get('/users', function () {
  return [
    'users' => [
      [
        'name' => 'Taylor Otwell',
      ],
    ],
  ];
});

$response->assertJsonFragment(['name' => 'Taylor Otwell']);
```

assertJsonMissing

Khẳng định rằng phản hồi không chứa dữ liệu JSON đã cho:

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

assertJsonMissingExact

Khẳng định rằng phản hồi không chứa dữ liệu JSON chính xác:

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

assertJsonMissingValidationErrors

Khẳng định rằng phản hồi không có lỗi xác thực JSON cho các khóa đã cho:

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

> Phương thức chung hơn assertValid có thể được sử dụng để khẳng định rằng một phản hồi không có lỗi xác thực được trả về dưới dạng JSON và không có lỗi nào được chuyển vào bộ nhớ session.

assertJsonPath

Khẳng định rằng phản hồi chứa dữ liệu đã cho tại đường dẫn được chỉ định:

```
$response->assertJsonPath($path, $expectedValue);
```

Ví dụ: nếu phản hồi JSON do ứng dụng của bạn trả về chứa dữ liệu sau:

```
{
  "user": {
    "name": "Steve Schoger"
  }
}
```

Bạn có thể khẳng định rằng thuộc tính `name` của đối tượng `user` khớp với một giá trị nhất định như sau:

```
$response->assertJsonPath('user.name', 'Steve Schoger');
```

assertJsonStructure

Khẳng định rằng phản hồi có cấu trúc JSON nhất định:

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

Ví dụ: nếu phản hồi JSON do ứng dụng của bạn trả về chứa dữ liệu sau:

```
{
  "user": {
    "name": "Steve Schoger"
  }
}
```

Bạn có thể khẳng định rằng cấu trúc JSON phù hợp với mong đợi của bạn như vậy:

```
$response->assertJsonStructure([
  'user' => [
    'name',
  ]
]);
```

Đôi khi, các phản hồi JSON do ứng dụng của bạn trả về có thể chứa các mảng đối tượng:

```
{
  "user": [
    {
      "name": "Steve Schoger",
      "age": 55,
      "location": "Earth"
    },
    {
      "name": "Mary Schoger",
      "age": 60,
      "location": "Earth"
    }
  ]
}
```

Trong trường hợp này, bạn có thể sử dụng ký tự * để khẳng định với cấu trúc của tất cả các đối tượng trong mảng:

```
$response->assertJsonStructure([
  'user' => [
    '*' => [
      'name',
      'age',
      'location'
    ]
  ]
]);
```

assertJsonValidationErrors

Khẳng định rằng phản hồi có lỗi giám định JSON cho các khóa đã cho. Phương pháp này nên được sử dụng khi xác nhận các phản hồi trong đó lỗi xác thực được trả về dưới dạng cấu trúc JSON thay vì được hiển thị vào session:

```
$response->assertJsonValidationErrors(array $data, $responseKey = 'errors');
```

> Phương thức chung hơn assertInvalid có thể được sử dụng để khẳng định rằng phản hồi có lỗi giám định được trả về dưới dạng JSON hoặc lỗi đã được chuyển vào bộ nhớ session.

assertJsonValidationErrorFor

Khẳng định phản hồi có bất kỳ lỗi giám định JSON nào cho khóa đã cho:

```
$response->assertJsonValidationErrorFor(string $key, $responseKey = 'errors');
```

assertLocation

Khẳng định rằng phản hồi có giá trị URI đã cho trong tiêu đề Vị trí:

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

assertNoContent

Khẳng định rằng phản hồi có mã trạng thái HTTP đã cho và không có nội dung:

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

assertNotFound

Khẳng định rằng phản hồi không tìm thấy mã trạng thái HTTP (404):

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

assertOk

Khẳng định rằng phản hồi có mã trạng thái HTTP 200:

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

assertPlainCookie

Khẳng định rằng phản hồi có chứa cookie không được mã hóa đã cho:

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

assertRedirect

Khẳng định rằng phản hồi là chuyển hướng đến URI đã cho:

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

assertRedirectContains

Xác nhận xem phản hồi có đang chuyển hướng đến một URI có chứa chuỗi đã cho hay không:

```
$response->assertRedirectContains($string);
```

assertRedirectToSignedRoute

Khẳng định rằng phản hồi là một chuyển hướng đến tuyến đường đã ký:

```
$response->assertRedirectToSignedRoute($name = null, $parameters = []);
```

assertSee

Khẳng định rằng chuỗi đã cho được chứa trong phản hồi. Xác nhận này sẽ tự động thoát khỏi chuỗi đã cho trừ khi bạn truyền đối số thứ hai giá trị `false`:

```
$response->assertSee($value, $escaped = true);
```

assertSeeInOrder

Khẳng định rằng các chuỗi đã cho được chứa theo thứ tự trong phản hồi. Xác nhận assertion này sẽ tự động thoát khỏi các chuỗi đã cho trừ khi bạn truyền đối số thứ hai giá trị `false`:

```
$response->assertSeeInOrder(array $values, $escaped = true);
```

assertSeeText

Khẳng định rằng chuỗi đã cho được chứa trong văn bản phản hồi. Xác nhận này sẽ tự động thoát khỏi chuỗi đã cho trừ khi bạn truyền đối số thứ hai giá trị `false`. Nội dung phản hồi sẽ được truyền đến hàm `strip_tags` PHP trước khi xác nhận được thực hiện:

```
$response->assertSeeText($value, $escaped = true);
```

assertSessionHas

Khẳng định rằng session chứa đoạn dữ liệu đã cho:

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

Nếu cần, một hàm xử lý có thể được cung cấp như là đối số thứ hai cho phương thức `assertSessionHas`. Khẳng định sẽ vượt qua nếu hàm xử lý trả về `true`:

```php
$response->assertSessionHas($key, function ($value) {
  return $value->name === 'Taylor Otwell';
});
```

assertSessionHasInput

Khẳng định rằng phiên có một giá trị nhất định trong mảng flash input:

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

Nếu cần, một hàm xử lý có thể được cung cấp làm đối số thứ hai cho phương thức `assertSessionHasInput`. Khẳng định sẽ vượt qua nếu hàm xử lý trả về `true`:

```php
$response->assertSessionHasInput($key, function ($value) {
  return Crypt::decryptString($value) === 'secret';
});
```

assertSessionHasAll

Khẳng định rằng session chứa một mảng các cặp khóa / giá trị nào đó:

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

Ví dụ: nếu session của ứng dụng của bạn chứa khóa `name` và khóa `status`, thì bạn có thể khẳng định rằng cả hai đều tồn tại và có các giá trị được chỉ định như sau:

```php
$response->assertSessionHasAll([
  'name' => 'Taylor Otwell',
  'status' => 'active',
]);
```

assertSessionHasErrors

Khẳng định rằng session có lỗi đối với các $keys đã cho. Nếu $keys là một mảng associative, hãy xác nhận rằng session chứa thông báo lỗi cụ thể (giá trị) cho mỗi trường (khóa). Phương pháp này nên được sử dụng khi kiểm tra các route có lỗi giám định đối với

session thay vì trả về chúng dưới dạng cấu trúc JSON:

```
$response->assertSessionHasErrors(
  array $keys, $format = null, $errorBag = 'default'
);
```

Ví dụ: để xác nhận rằng các trường `name` và `email` có thông báo lỗi giám định đã được hiển thị trong session, bạn có thể gọi phương thức `assertSessionHasErrors` như sau:

```
$response->assertSessionHasErrors(['name', 'email']);
```

Hoặc, bạn có thể khẳng định rằng một trường nhất định có một thông báo lỗi giám định cụ thể:

```
$response->assertSessionHasErrors([
  'name' => 'The given name was invalid.'
]);
```

assertSessionHasErrorsIn

Khẳng định rằng session có lỗi đối với các $keys đã cho trong một túi lỗi (error bag) cụ thể. Nếu $keys là một mảng associative, hãy xác nhận rằng session chứa thông báo lỗi cụ thể (giá trị) cho mỗi trường (khóa), trong túi lỗi:

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

assertSessionHasNoErrors

Khẳng định rằng session không có lỗi xác thực:

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

assertSessionDoesntHaveErrors

Khẳng định rằng session không có lỗi xác thực cho các khóa đã cho:

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

assertSessionMissing

Khẳng định rằng session không chứa khóa đã cho:

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

assertStatus

Khẳng định rằng phản hồi có mã trạng thái HTTP nhất định:

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

assertSuccessful

Khẳng định rằng phản hồi có mã trạng thái HTTP thành công (> = 200 và <300):

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

assertUnauthorized

Khẳng định rằng phản hồi có mã trạng thái HTTP (401) trái phép:

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

assertUnprocessable

Khẳng định rằng phản hồi có mã trạng thái HTTP thực thể không thể xử lý (422):

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

assertValid

Khẳng định rằng phản hồi không có lỗi giám định cho các khóa đã cho. Phương thức này có thể được sử dụng để xác nhận assertion các phản hồi trong đó các lỗi giám định được trả về

dưới dạng cấu trúc JSON hoặc khi các lỗi xác thực đã được đưa sang session:

```
// Assert that no validation errors are present...
$response->assertValid();

// Assert that the given keys do not have validation errors...
$response->assertValid(['name', 'email']);
```

assertInvalid

Khẳng định rằng phản hồi có lỗi giám định cho các khóa đã cho. Phương thức này có thể được sử dụng để xác nhận assertion các phản hồi trong đó các lỗi giám định được trả về dưới dạng cấu trúc JSON hoặc khi các lỗi xác thực đã được đưa sang session:

```
$response->assertInvalid(['name', 'email']);
```

Bạn cũng có thể khẳng định rằng một khóa nhất định có một thông báo lỗi giám định cụ thể. Khi làm như vậy, bạn có thể cung cấp toàn bộ thông báo hoặc chỉ một phần nhỏ của thông báo:

```
$response->assertInvalid([
  'name' => 'The name field is required.',
  'email' => 'valid email address',
]);
```

assertViewHas

Khẳng định rằng giao diện hiển thị phản hồi chứa một phần dữ liệu đã cho:

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

Việc truyền một hàm xử lý làm đối số thứ hai cho phương thức `assertViewHas` sẽ cho phép bạn kiểm tra và đưa ra xác nhận đối với một phần cụ thể dữ liệu hiển thị:

```
$response->assertViewHas('user', function (User $user) {
  return $user->name === 'Taylor';
```

```
});
```

Ngoài ra, dữ liệu hiển thị có thể được truy cập dưới dạng các biến mảng trên phản hồi, cho phép bạn kiểm tra nó một cách thuận tiện:

```
$this->assertEquals('Taylor', $response['name']);
```

assertViewHasAll

Khẳng định rằng phản hồi cho giao diện hiển thị có một danh sách dữ liệu nhất định:

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

Phương pháp này có thể được sử dụng để khẳng định rằng giao diện hiển thị chỉ chứa dữ liệu khớp với các khóa đã cho:

```
$response->assertViewHasAll([
  'name',
  'email',
]);
```

Hoặc, bạn có thể khẳng định rằng dữ liệu chế độ xem hiện có và có các giá trị cụ thể:

```
$response->assertViewHasAll([
  'name' => 'Taylor Otwell',
  'email' => 'taylor@example.com,',
]);
```

assertViewIs

Khẳng định rằng giao diện hiển thị đã cho đã được trả về bởi route:

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

assertViewMissing

Khẳng định rằng khóa dữ liệu đã cho không khả dụng cho giao diện hiển thị được trả về trong phản hồi của ứng dụng:

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

Assertion xác thực người dùng

Laravel cũng cung cấp nhiều xác nhận liên quan đến xác thực assertion mà bạn có thể vận dụng trong các bài kiểm tra tính năng của ứng dụng. Lưu ý rằng các phương thức này được gọi trên chính class thử nghiệm chứ không phải đối tượng `Illuminate\Testing\TestResponse` được trả về bởi các phương thức như `get` và `post`.

`assertAuthenticated`

Khẳng định rằng người dùng đã được xác thực:

```
$this->assertAuthenticated($guard = null);
```

`assertGuest`

Khẳng định rằng người dùng không được xác thực:

```
$this->assertGuest($guard = null);
```

`assertAuthenticatedAs`

Khẳng định rằng một người dùng cụ thể được xác thực:

```
$this->assertAuthenticatedAs($user, $guard = null);
```