

# Créer une application avec Laravel 5.5 – Les tests

Pour être honnête je n'aime pas coder des tests, j'ai souvent l'impression de perdre mon temps. Mais je reconnais leur grande utilité. Dans l'idéal il faudrait commencer par eux ([Test-driven development](#)) ou au moins établir les tests au fur et à mesure. Le grand intérêt des tests à mes yeux est de s'assurer qu'on a pas tout cassé d'un côté en bricolant d'un autre côté. Dans ce chapitre on va mettre en place des tests pour la galerie.

Il y a deux grandes catégories de tests dans Laravel : Http (**PHPUnit**) et Navigateur (**Dusk**). Dans ce chapitre on ne s'intéressera qu'à la première catégorie.

## On se prépare

Laravel est pensé pour intégrer facilement des tests et il comporte de base [PHPUnit](#) avec un fichier **phpunit.xml** à la racine :

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit backupGlobals="false"
    backupStaticAttributes="false"
    bootstrap="vendor/autoload.php"
    colors="true"
    convertErrorsToExceptions="true"
    convertNoticesToExceptions="true"
    convertWarningsToExceptions="true"
    processIsolation="false"
    stopOnFailure="false">
    <testsuites>
        <testsuite name="Feature">
                                                    <directory
suffix="Test.php">./tests/Feature</directory>
        </testsuite>

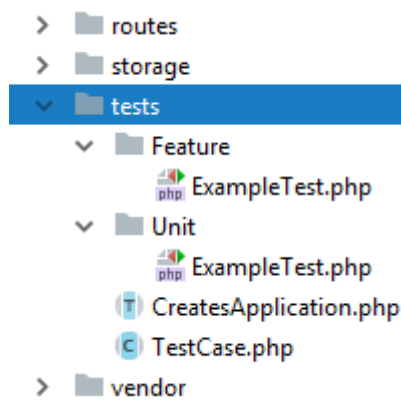
        <testsuite name="Unit">
            <directory suffix="Test.php">./tests/Unit</directory>
        </testsuite>
```

```

</testsuites>
<filter>
    <whitelist processUncoveredFilesFromWhitelist="true">
        <directory suffix=".php">./app</directory>
    </whitelist>
</filter>
<php>
    <env name="APP_ENV" value="testing"/>
    <env name="CACHE_DRIVER" value="array"/>
    <env name="SESSION_DRIVER" value="array"/>
    <env name="QUEUE_DRIVER" value="sync"/>
</php>
</phpunit>

```

On trouve aussi un dossier spécifique déjà un peu garni :



On va supprimer les deux exemples qui ne nous serviront pas.

Le principal souci lors de test réside dans la base de données. On va régler ça en utilisant **sqlite** en mémoire :

```

<php>
    ...
    <env name="DB_CONNECTION" value="sqlite"/>
    <env name="DB_DATABASE" value=":memory:"/>
</php>

```

On va s'arranger pour recréer la base avant chaque test, et tant qu'à faire on va la remplir de données.

Toutes les classes de test héritent de **TestCase.php** :

```
<?php
```

```
namespace Tests;
```

```
use Illuminate\Foundation\Testing\TestCase as BaseTestCase;
```

```
abstract class TestCase extends BaseTestCase
{
    use CreatesApplication;
}
```

Cette classe n'est pas très garnie à la base mais c'est le lieux idéal pour préparer les tests.

On va utiliser le trait **RefreshDatabase** qui est destiné à régénérer la base. Si on regarde la fonction qui effectue la régénération de la base en mémoire on trouve :

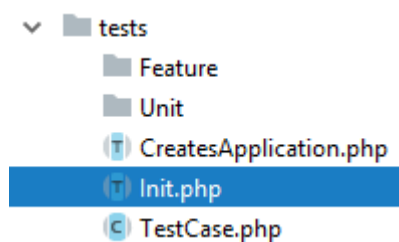
```
protected function refreshInMemoryDatabase()
{
    $this->artisan('migrate');

    $this->app[Kernel::class]->setArtisan(null);
}
```

Donc une simple migration. Pour la galerie il nous faut plus que ça :

- une population (seed)
- un chargement des catégories qui sont partagées par défaut par toutes les vues.

On va donc créer un autre trait pour surcharger cette fonction :



Avec ce code :

```
<?php
```

```
namespace Tests;
```

```
use Illuminate\Contracts\Console\Kernel;
use App\Models\Category;
```

```

trait Init
{
    /**
     * Refresh the in-memory database.
     *
     * @return void
     */
    protected function refreshInMemoryDatabase()
    {
        $this->artisan('migrate');

        $this->artisan('db:seed');

        view ()->share ('categories', Category::all ());

        $this->app[Kernel::class]->setArtisan(null);
    }
}

```

Et on met à jour **TestCase.php** :

```
<?php
```

```
namespace Tests;
```

```

use Illuminate\Foundation\Testing\TestCase as BaseTestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;
use App\Models\User;

```

```

abstract class TestCase extends BaseTestCase
{
    use CreatesApplication, RefreshDatabase, Init {
        Init::refreshInMemoryDatabase insteadof RefreshDatabase;
    }

    /**
     * Authentication.
     *
     * @return void
     */
    protected function auth($id)
    {
        $user = User::find($id);
    }
}

```

```
        $this->actingAs($user);  
    }  
}
```

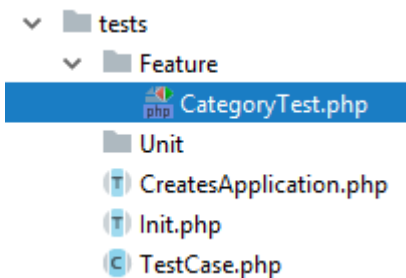
J'ai ajouté une fonction qui (**auth**) nous permettra d'authentifier un utilisateur pour les tests.

Pour terminer ces préparatifs je signale que personnellement je mets le fichier **phar** de PHPUnit à la racine pour me simplifier la vie et la syntaxe des commandes...

## Les catégories

On va maintenant mettre en place des tests pour les catégories : création, modification, suppression et tant qu'à faire les validations qui vont avec.

```
php artisan make:test CategoryTest
```



On a ce code par défaut :

```
<?php
```

```
namespace Tests\Feature;
```

```
use Tests\TestCase;
```

```
use Illuminate\Foundation\Testing\RefreshDatabase;
```

```
class CategoryTest extends TestCase  
{
```

```
    /**  
     * A basic test example.  
     *  
     * @return void  
     */
```

```

    public function testExample()
    {
        $this->assertTrue(true);
    }
}

```

On peut supprimer la référence au trait **RefreshDatabase** qu'on a déjà mis dans **TestCase.php**.

## Création d'une catégorie

### Réussite

On ajoute ce code pour tester la création d'une catégorie :

```

public function testAddCategory()
{
    $this->auth(1);

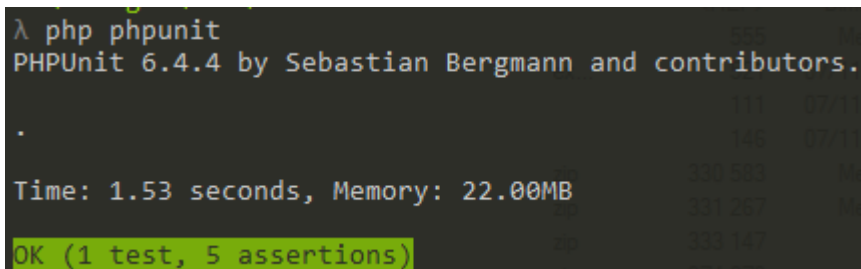
    $this->get('/category/create')
        ->assertSee('Name');

    $response = $this->post('/category', [
        'name' => 'Une catégorie',
    ]);

    $this->assertDatabaseHas('categories', [
        'name' => 'Une catégorie',
        'slug' => 'une-categorie',
    ]);

    $response->assertStatus(302)
        ->assertHeader('Location', url('/'));
}

```



```

λ php phpunit
PHPUnit 6.4.4 by Sebastian Bergmann and contributors.

.
Time: 1.53 seconds, Memory: 22.00MB

OK (1 test, 5 assertions)

```

On a vérifié (on commence par authentifier l'administrateur) :

- qu'on peut bien afficher le formulaire
- qu'on peut soumettre le formulaire
- que la base est bien mise à jour
- que la réponse est correcte

## Échec de validation

On va aussi tester la validation :

```
public function testAddCategoryFail()
{
    $this->auth(1);

    // Required
    $response = $this->post('/category');
    $response->assertSessionHasErrors('name');

    // Unique
    $response = $this->post('/category', [
        'name' => 'Maisons',
    ]);
    $response->assertSessionHasErrors('name');

    // Max length
    $response = $this->post('/category', [
        'name' => str_random(256),
    ]);
    $response->assertSessionHasErrors('name');

    // String
    $response = $this->post('/category', [
        'name' => 256,
    ]);
    $response->assertSessionHasErrors('name');
}
```

OK (2 tests, 13 assertions)

On a vérifié les règles :

- required
- unique

- max
- string

# Modification d'une catégorie

## Réussite

On va tester de la même manière la modification d'une catégorie :

```
public function testUpdateCategory()
{
    $this->auth(1);

    $this->get('/category/2/edit')
        ->assertSee('Maisons');

    $response = $this->put('/category/2', [
        'name' => 'Immeubles',
    ]);

    $this->assertDatabaseHas('categories', [
        'name' => 'Immeubles',
    ]);

    $this->assertDatabaseMissing('categories', [
        'name' => 'Maisons',
    ]);

    $response->assertStatus(302)
        ->assertHeader('Location', url('/'));
}
```

OK (3 tests, 19 assertions)

On a vérifié :

- qu'on peut bien afficher le formulaire
- qu'on peut soumettre le formulaire
- que la base est bien mise à jour
- que la réponse est correcte



# Échec de validation

On va aussi tester la validation :

```
public function testUpdateCategoryFail()
{
    $this->auth(1);

    // Required
    $response = $this->put('/category/2');
    $response->assertSessionHasErrors('name');

    // Unique
    $response = $this->put('/category/2', [
        'name' => 'Animaux',
    ]);
    $response->assertSessionHasErrors('name');

    // Max length
    $response = $this->put('/category/2', [
        'name' => str_random(256),
    ]);
    $response->assertSessionHasErrors('name');

    // String
    $response = $this->put('/category/2', [
        'name' => 256,
    ]);
    $response->assertSessionHasErrors('name');
}
```

OK (4 tests, 27 assertions)

On a vérifié les règles :

- required
- unique
- max
- string

# Suppression d'une catégorie

On va vérifier qu'on peut supprimer une catégorie :

```
public function testDeleteCategory()
{
    $this->auth(1);

    $response = $this->delete('/category/1');

    $this->assertDatabaseMissing('categories', [
        'name' => 'Paysages',
    ]);

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

OK (5 tests, 29 assertions)

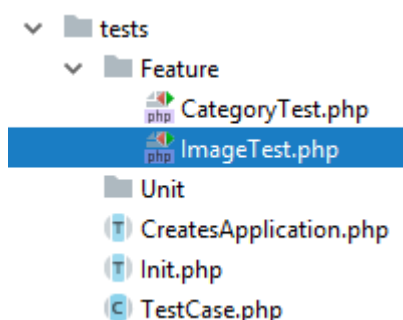
On a vérifié :

- qu'on peut soumettre le formulaire
- qu'on met la base à jour
- qu'on retourne une réponse correcte

## Les images

On va maintenant mettre en place des tests pour les images : création, suppression et tant qu'à faire les validations qui vont avec.

```
php artisan make:test ImageTest
```



# Ajout d'une image

## Réussite

Pour l'ajout d'une image on va devoir :

- simuler (fake) le storage
- simuler (fake) le téléchargement

Voici le test :

```
<?php
```

```
namespace Tests\Feature;
```

```
use Tests\TestCase;
use Illuminate\Http\UploadedFile;
use Illuminate\Support\Facades\Storage;
```

```
class ImageTest extends TestCase
{
    /**
     * Test add image.
     *
     * @return void
     */
    public function testAddImage()
    {
        $this->auth(2);

        $this->get('/')
            ->assertSee('Add an image');

        $this->get('/image/create')
            ->assertSee('Description');

        Storage::fake('images');
        Storage::fake('thumbs');

        $response = $this->post('/image', [
            'image' => UploadedFile::fake()->image('paysage.jpg'),
            'category_id' => 2,
            'description' => 'un beau paysage',
        ],
```

```

    });

    $this->assertDatabaseHas('images', [
        'description' => 'un beau paysage',
    ]);

    $response->assertStatus(302)
        ->assertHeader('Location', url('/image/create'));
    }
}

```

OK (6 tests, 35 assertions)

On a vérifié :

- qu'on a l'item dans le menu
- qu'on affiche le formulaire
- qu'on peut soumettre le formulaire
- qu'on met bien la base à jour
- qu'on renvoie une réponse correcte

## Échec de validation

On va aussi tester la validation :

```

public function testImageFail()
{
    $this->auth(2);

    // Required
    $response = $this->post('/image');
    $response->assertSessionHasErrors(['image', 'category_id']);

    // Image
    $response = $this->post('/image', [
        'image' => 'texte',
    ]);
    $response->assertSessionHasErrors('image');

    // Max
    $response = $this->post('/image', [
        'image' =>
UploadedFile::fake()->image('paysage.jpg')->size(2001),

```

```

        'description' => str_random(256),
    ]);
$response->assertSessionHasErrors(['image', 'description']);

// Exists
$response = $this->post('/image', [
    'category_id' => 10,
]);
$response->assertSessionHasErrors('category_id');
}

```

OK (7 tests, 45 assertions)

On a vérifié les règles :

- required
- unique
- max
- exists

## Suppression d'une image

On va vérifier qu'on peut supprimer une image :

```

public function testDeleteImage()
{
    $this->auth(1);

    $response = $this->delete('/image/20');

    $this->assertDatabaseMissing('images', [
        'id' => 20,
    ]);

    $response->assertStatus(302)
        ->assertHeader('Location', url('/'));
}

```

OK (8 tests, 49 assertions)

On a vérifié :

- qu'on peut soumettre le formulaire
- qu'on met la base à jour
- qu'on retourne une réponse correcte

# Conclusion

Vous trouverez [sur Github](#) la version finale avec 4 autres tests (login, register, profile et locale).

Dans ce chapitre on a :

- préparé les tests avec une base sqlite en mémoire
- établi les tests pour les catégories
- établi les tests pour les images

Ainsi se termine cette série ! Il est fort possible que des choses évoluent au niveau du dépôt sur Github en fonction des réactions et remarques.