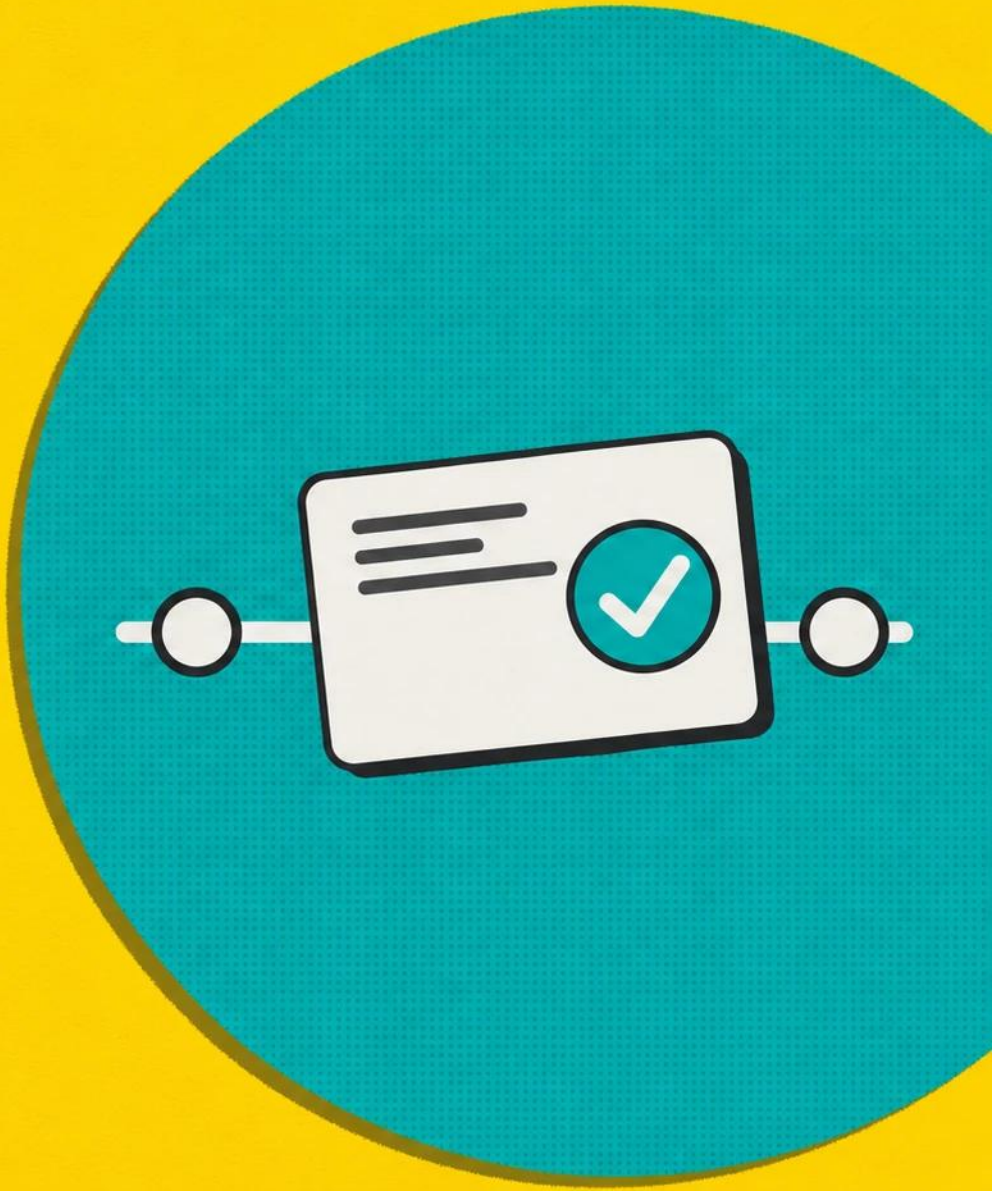
1.17

TEMPS 1 · LE PORTFOLIO · SÉANCE 4

Les premiers tests

Thibault Six · IFOSUP Wavre · 2026-2027

5XDYN · Projet web dynamique · 1.17



? Comment vérifier qu'une page de votre portfolio répond bien, sans ouvrir le navigateur ?

Un test fonctionnel envoie une vraie requête à votre application, sans navigateur, et vérifie sa réponse.

le navigateur ✕ pas ouvert

tests/Feature/PortfolioTest.php

un test Pest : une phrase qui décrit le comportement attendu, une fonction qui le vérifie

une requête `GET /projets` →

← la réponse

votre application Laravel

la même que pour un visiteur

- [ ] le code HTTP
- [ ] la redirection
- [ ] la page rendue
- [ ] les données enregistrées

tests/Feature
un test fonctionnel : une requête, une réponse

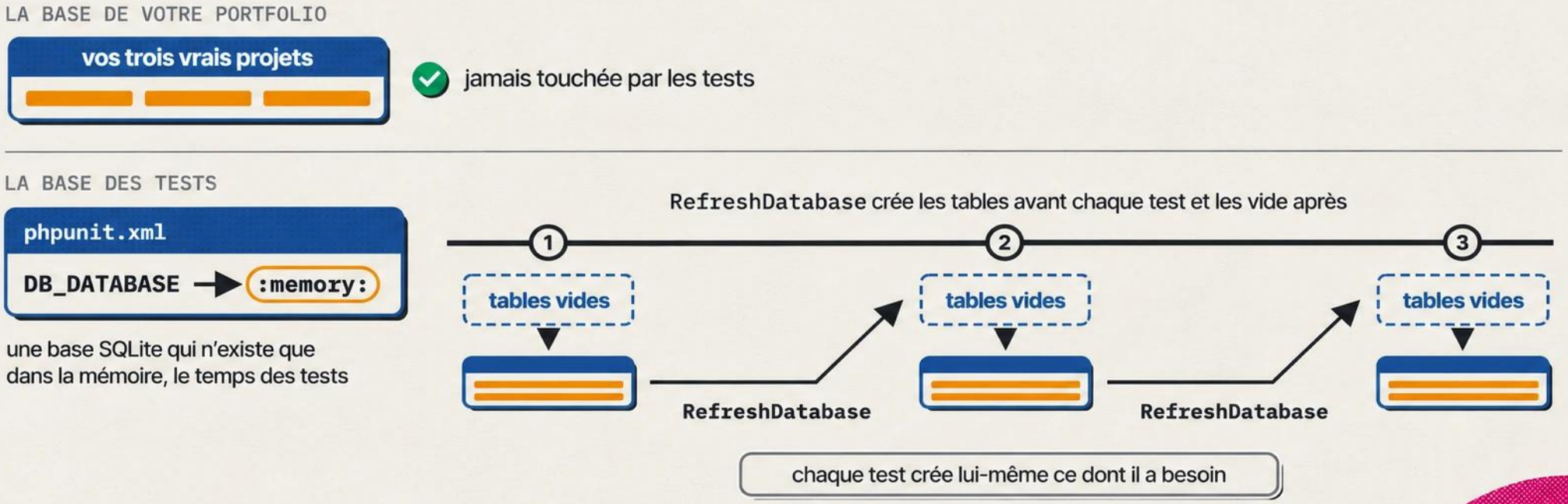
`$ php artisan test` lance tous les tests, ceux du kit compris

tests/Unit
un test unitaire : une classe seule, sans requête

Suivez la requête : elle part du test, traverse l'application et revient au test. Le navigateur reste fermé.

? Vos tests vont créer des projets. Que deviennent vos trois vrais projets pendant les tests ?

Chaque test démarre avec une base vide, créée dans la mémoire : vos trois vrais projets ne sont jamais touchés.



En haut, la base de votre portfolio reste à l'écart. En bas, chaque test retrouve des tables vides.

? Chaque test part d'une base vide. Comment la liste des projets aura-t-elle de quoi montrer ?

tests/Pest.php applique RefreshDatabase à tous les tests de Feature, et chaque test crée ses données avec une fabrique.

tests/Pest.php

```
pest()->extend(TestCase::class)
    ->use(RefreshDatabase::class)
    ->in('Feature');
```

- `pest()->extend(TestCase::class)` — démarre l'application de Laravel
- `->use(RefreshDatabase::class)` — une base vide à chaque test
- `->in('Feature');` — pour chaque test du dossier Feature

écrit par le kit : vous n'y touchez pas

tests/Feature/PortfolioTest.php

```
Project::factory()->count(2)->create();
```

les fabriques du chapitre 1.06

la base du test

un projet au titre inventé	`published_at` → `now()`
un projet au titre inventé	`published_at` → `now()`

la fabrique publie les projets : `published_at` vaut `now()` dans sa méthode `definition()`

pour un brouillon `Project::factory()->draft()->create()`

À gauche, la règle posée par le kit pour tout le dossier. À droite, la ligne qui remplit la base d'un test.

? Un test Pest, c'est une phrase et une fonction. Que fait chaque ligne de la fonction ?

Un test prépare ses données, envoie une requête avec get(), puis vérifie la réponse avec assertOk() et assertSee().

la phrase : le comportement attendu

tests/Feature/PortfolioTest.php

```php
test('the projects page shows a title', function () {
    Project::factory()->create(['title' => 'Quiz des capitales']);

    $this->get('/projets')
        ->assertOk()
        ->assertSee('Quiz des capitales');
});
```

① **préparer**

② **envoyer** — envoie une requête GET et renvoie la réponse

③ **vérifier**

`assertOk()` : le code 200

`assertSee()` : cherche un texte dans le HTML de la réponse

✓ **ce test passe**

Lisez la fonction de haut en bas : préparer, envoyer, vérifier. La phrase du haut dit ce qui est vérifié.

? Le même test avec le titre « Météo Wavre » échoue. Qu'y a-t-il vraiment dans le HTML reçu ?

Le HTML d'une page Inertia porte les props en JSON, où l'accent est échappé : assertSee() n'y trouve plus le titre.

✕ the projects page shows an accented title — To contain: Météo Wavre **ce que le test cherchait**

au chapitre 1.07

la fin de la réponse

```
<script data-page="app" type="application/json">
{"component":"projects\/Index","props":{…,"projects":[{"title":"M\u00e9t\u00e9o Wavre",…}]},…}
</script><div id="app"></div>
```

les props de la page, en JSON ; l'accent y est échappé

vide : Vue la remplira dans le navigateur

`assertSee()` cherche une chaîne n'importe où dans la réponse, sans savoir où elle devrait être

En haut, ce que le test cherchait. Dessous, ce que la réponse contient vraiment : du JSON et une div vide.

? Plutôt qu'un texte dans le HTML, que peut-on vérifier sur une page Inertia ?

assertInertia() vérifie la page rendue par le contrôleur : son composant avec component(), et ses props avec has().

```
use Inertia\Testing\AssertableInertia as Assert;
```

en tête du fichier

tests/Feature/PortfolioTest.php — extrait

```
$this->get('/projets')
    ->assertOk()
    ->assertInertia(fn (Assert $page) => $page
        ->component('projects/Index')
        ->has('projects', 2)
    );
```

une fonction fléchée : elle reçoit la page et enchaîne les vérifications

```
avant : Project::factory()->count(2)->create();
```

la page rendue

`projects/Index` **le composant**

vérifie aussi que `resources/js/pages/projects/Index.vue` existe

`projects` **les props**

la prop existe et contient deux éléments

Chaque vérification vise une partie de la page : la flèche du haut le composant, celle du bas la prop.

? Comment vérifier le titre exact d'un projet, accent compris, sans passer par le HTML ?

where('project.title', 'Météo Wavre') lit une valeur des props : le point descend d'un niveau, et l'accent ne gêne plus.

```php
Project::factory()->create(['title' => 'Météo Wavre', 'slug' => 'meteo-wavre']);
```

le slug est fixé : l'adresse visitée en dépend, et la fabrique en aurait inventé un autre

tests/Feature/PortfolioTest.php — extrait

```php
$this->get('/projets/meteo-wavre')
    ->assertOk()
    ->assertInertia(fn (Assert $page) => $page
        ->component('projects/Show')
        ->where('project.title', 'Météo Wavre')
    );
```

la page rendue : projects/Show

`project`

le point descend d'un niveau dans les props

`title` → `'Météo Wavre'`

✓ la comparaison porte sur la valeur elle-même, avant tout encodage : l'accent ne gêne plus

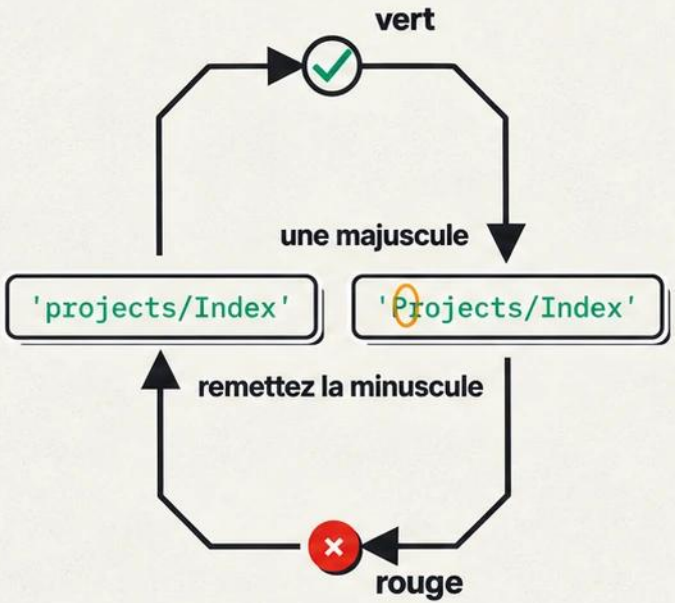
Suivez le chemin project.title dans les props : le point mène de la prop project à sa clé title.

1.17

08

? Vous écrivez 'Projects/Index' avec une majuscule. Que dit le test, et dans quel ordre le lire ?

Un test rouge compare deux valeurs : Expected est ce que vous avez écrit, Actual ce que l'application a rendu.




```
$ php artisan test tests/Feature/PortfolioTest.php
```

```
✗ the projects page lists the published projects
  Unexpected Inertia page component.
Failed asserting that two strings are identical.
--- Expected
+++ Actual
@@ @@
-'Projects/Index'
+'projects/Index'
```

1. le nom du test rouge
2. le message : le composant n'est pas celui attendu
3. Expected : ce que vous avez écrit. Actual : ce que l'application a rendu

sous Windows, Projects et projects désignent le même dossier, mais pas sur le serveur Linux d'OVH

Faites le tour : vert, une majuscule, rouge, la minuscule remise, vert. Le ticket se lit de haut en bas.

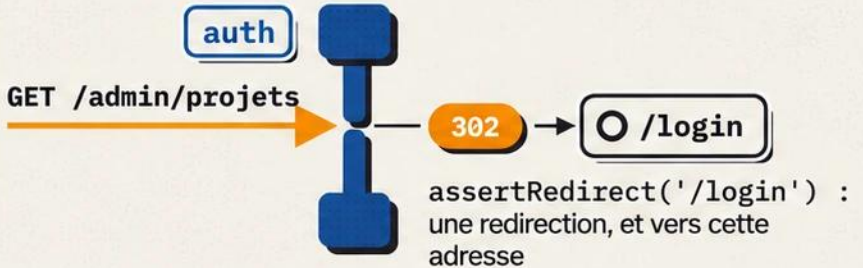
1.17

? Un invité ouvre /admin/projets. Comment tester son renvoi, et l'entrée de l'auteur connecté ?

L'invité est renvoyé vers /login, dit assertRedirect() ; actingAs() connecte un utilisateur pour la suite.

UN INVITÉ

```php
test('guests cannot open the admin', function () {
    $this->get('/admin/projets')->assertRedirect('/login');
});
```



le middleware du groupe admin, chapitre 1.10

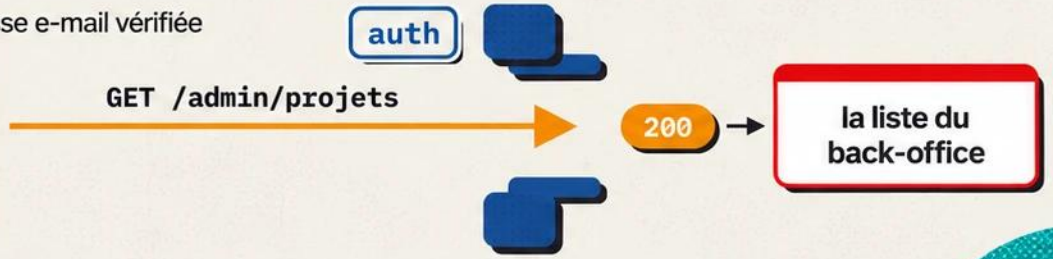
L'AUTEUR CONNECTÉ

fait comme si cet utilisateur s'était connecté, pour les requêtes qui suivent

un utilisateur à l'adresse e-mail vérifiée

extrait

```php
$this->actingAs(User::factory()->create())
    ->get('/admin/projets')
    ->assertOk();
```



Même adresse, même portillon : en haut, l'invité repart vers /login ; en bas, l'auteur connecté passe.

1.17

? L'auteur crée un projet depuis le back-office. Quelles trois choses le test vérifie-t-il ?

Le test envoie les données avec post(), puis vérifie la redirection, le message du flash et la ligne enregistrée en base.

```
test('the author can create a project', function () {
```

l'envoi — extrait

```
$this->actingAs(User::factory()->create())
    ->post('/admin/projets', [
        'title' => 'Mon portfolio',
        'summary' => 'Le site que vous lisez.',
    ])
```

envoie les données comme le ferait votre <Form>

les vérifications — extrait

```
    ->assertRedirect('/admin/projets')
    ->assertInertiaFlash('toast.message', 'Projet créé.');

expect(Project::where('slug', 'mon-portfolio')->exists())->toBeTrue();
```

1. `302` → `/admin/projets` ✓
   comme `to_route('admin.projects.index')` dans `store()`
2. `toast.message` → `'Projet créé.'` ✓
   `assertInertiaFlash()` lit le flash `Inertia::flash('toast', ...)` du chapitre 1.12
3. **la base du test** — `mon-portfolio` ✓
   le projet existe, avec le slug calculé à partir du titre

Un envoi à gauche, trois vérifications à droite : la réponse, le flash, puis la base elle-même.

1.17

? Un envoi sans titre est refusé. Où le serveur range-t-il l'erreur, et comment la tester ?

Un envoi invalide repart vers le formulaire, ses erreurs en session : assertSessionHasErrors('title') les y cherche.

tests/Feature/PortfolioTest.php — extrait

```php
$this->actingAs(User::factory()->create())
    ->post('/admin/projets', ['title' => '', 'summary' => 'Un résumé.'])
    ->assertSessionHasErrors('title');

expect(Project::count())->toBe(0);
```

au chapitre 1.12



Suivez l'envoi refusé : la redirection, la session, puis la base, restée vide.

1.17

? Qu'est-ce qui doit être vrai à la fin ?

À vous, vingt-cinq minutes : cinq tests verts dans PortfolioTest.php, puis deux tests cassés exprès et réparés.

À FAIRE

1. La liste des projets, avec `component()` et `has()`.
2. La fiche d'un projet, avec `where()` sur son titre.
3. Un invité renvoyé vers `/login` depuis `/admin/projets`.
4. La création d'un projet par l'auteur connecté, avec la redirection, le flash et la ligne en base.
5. La création refusée sans titre, avec `assertSessionHasErrors()`.

À VÉRIFIER

25 min

```
$ php artisan test tests/Feature/PortfolioTest.php
```

```
Tests:    5 passed (30 assertions)
```

✓

- [ ] `php artisan test` sans argument : les tests du kit restent verts.
- [ ] `'auth'` retiré du groupe admin dans `routes/web.php` : le test de l'invité passe au rouge.
- [ ] Le message du flash changé dans `store()` : le test de création passe au rouge.

puis réparez

Bonus : un brouillon, `Project::factory()->draft()->create(['slug' => 'pas-fini'])`, et `assertNotFound()` sur `/projets/pas-fini`.

Cinq tests à gauche, les vérifications à droite ; les deux dernières passent un test au rouge, exprès, puis vous réparez.

1.17