Augmentez votre couverture :

supprimez des tests !

→ Baptiste Langlade

→ Lyon

→ 10+ ans XP

→ ~95 packages Open Source


efalia

GED

→ Armoires

→ Gabarits de documents

→ Documents

→ Bannettes

→ Documents

Tests fonctionnels

→ Armoires

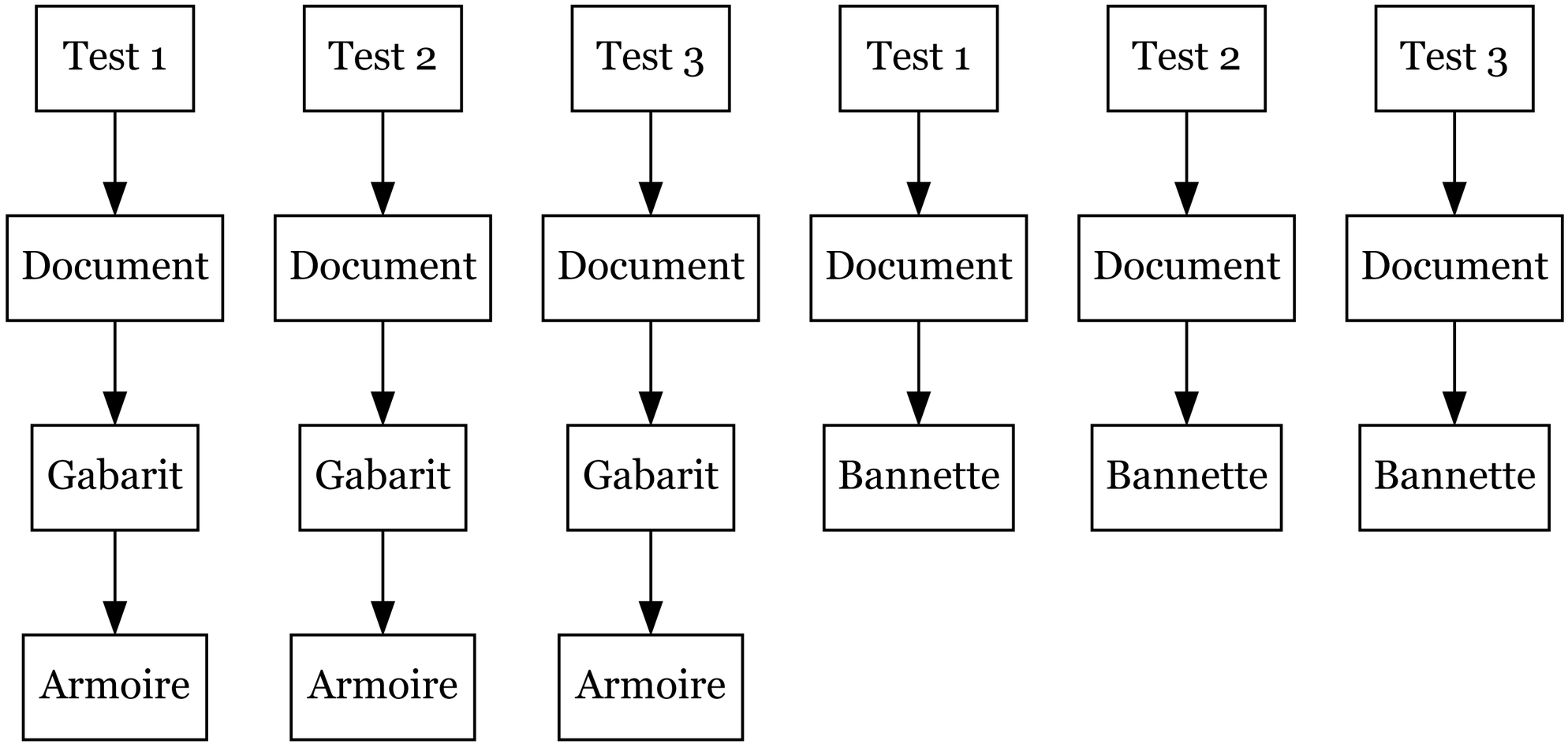
→ Gabarits de documents

→ **Documents**

→ Bannettes

→ **Documents**

Verrouiller document



→ Armoires

→ Gabarits de documents

→ **Documents**

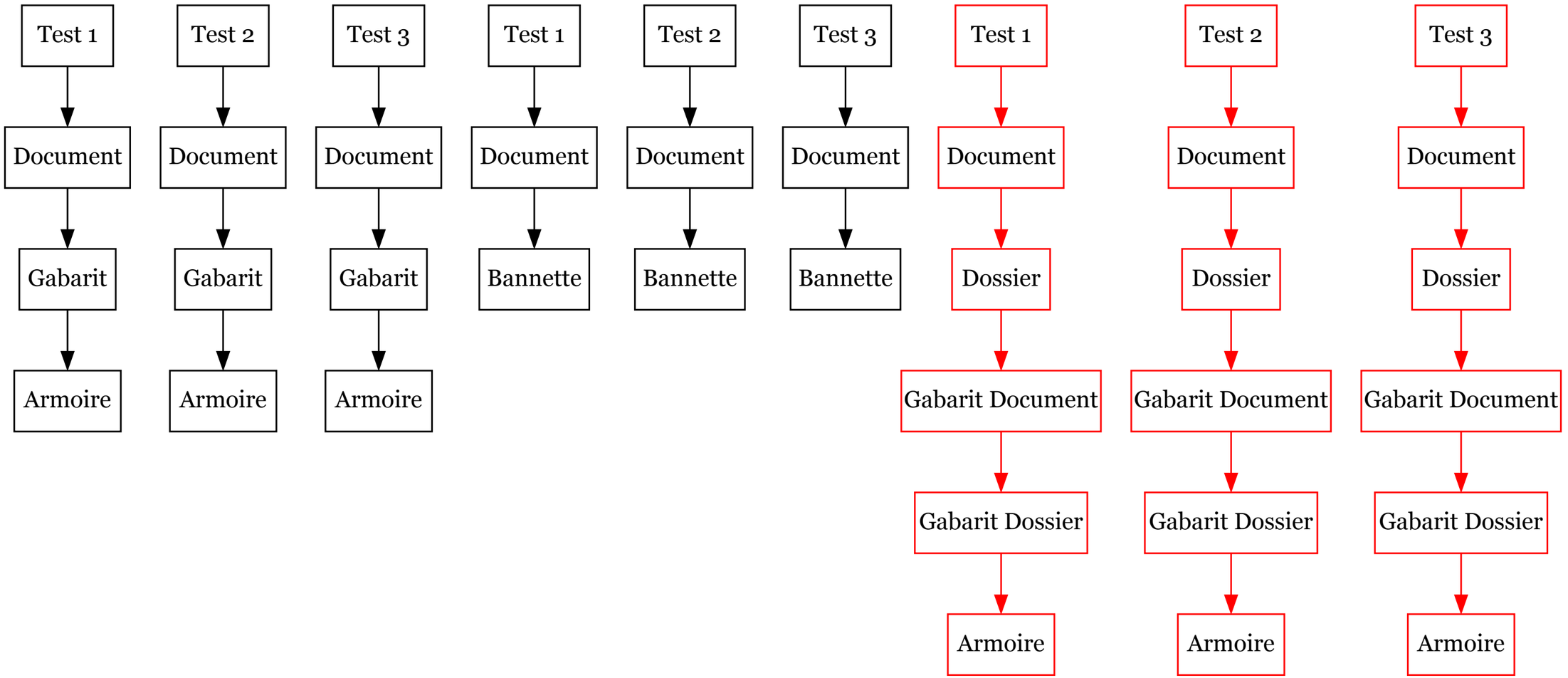
→ Gabarits de dossiers

→ Dossiers > Gabarits de documents > **Documents**

→ Bannettes

→ **Documents**

Verrouiller document



Après 3 ans

→ 730 tests (350 positifs, 380 négatifs)

→ 1h15 de temps d'exécution

Complexité Exponentielle

Property Based Testing

Loi de Murphy

Tests locaux & CI

Tests en dur

```php
final class ArmoireTest extends TestCase
{
    public function testCreationArmoire()
    {
        $response = $this->makePost('/api/armoires', [
            'nom' => 'foobar',
        ]);

        $this->assertSame(201, $response->getStatusCode());
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    public function testCreationGabaritDeDocument()
    {
        $response = $this->makePost('/api/armoires', [
            'nom' => 'foobar',
        ]);

        $this->assertSame(201, $response->getStatusCode());
        $armoire = \json_decode($response->getContent(), true);

        $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
            'nom' => 'baz',
        ]);

        $this->assertSame(201, $response->getStatusCode());
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    public function testCreationGabaritDeDocument()
    {
        $response = $this->makePost('/api/armoires', [
            'nom' => 'foobar',
        ]);

        $this->assertSame(201, $response->getStatusCode());
        $armoire = \json_decode($response->getContent(), true);

        $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
            'nom' => 'baz',
        ]);

        $this->assertSame(201, $response->getStatusCode());
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    public function testCreationGabaritDeDocument()
    {
        $response = $this->makePost('/api/armoires', [
            'nom' => 'foobar',
        ]);

        $this->assertSame(201, $response->getStatusCode());
        $armoire = \json_decode($response->getContent(), true);

        $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
            'nom' => 'baz',
        ]);

        $this->assertSame(201, $response->getStatusCode());
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    public function testCreationGabaritDeDocument()
    {
        $armoire = $this->creerArmoire('foobar');

        $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
            'nom' => 'baz',
        ]);

        $this->assertSame(201, $response->getStatusCode());
    }
}
```

Données aléatoires

```
composer require --dev innmind/black-box
```

```php
use Innmind\BlackBox\{PHPUnit\BlackBox, Set};

final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCreationArmoire()
    {
        $this
            ->forAll(Set\Elements::of('foobar'))
            ->then(function(string $nom) {
                $response = $this->makePost('/api/armoires', [
                    'nom' => $nom,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
use Innmind\BlackBox\{PHPUnit\BlackBox, Set};

final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCreationArmoire()
    {
        $this
            ->forAll(Set\Elements::of('foobar'))
            ->then(function(string $nom) {
                $response = $this->makePost('/api/armoires', [
                    'nom' => $nom,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
use Innmind\BlackBox\{PHPUnit\BlackBox, Set};

final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCreationArmoire()
    {
        $this
            ->forAll(Set\Elements::of('foobar'))
            ->then(function(string $nom) {
                $response = $this->makePost('/api/armoires', [
                    'nom' => $nom,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
use Innmind\BlackBox\{PHPUnit\BlackBox, Set};

final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCreationArmoire()
    {
        $this
            ->forAll(Set\Elements::of('foobar'))
            ->then(function(string $nom) {
                $response = $this->makePost('/api/armoires', [
                    'nom' => $nom,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

Pour toute string entre 1 et 255 caractères alors je peux créer une armoire

```php
final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCreationArmoire()
    {
        $this
            ->forAll(Set\Strings::between(1, 255))
            ->then(function(string $nom) {
                $response = $this->makePost('/api/armoires', [
                    'nom' => $nom,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCreationArmoire()
    {
        $this
            ->forAll(Set\Strings::between(1, 255))
            ->then(function(string $nom) {
                $response = $this->makePost('/api/armoires', [
                    'nom' => $nom,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCreationArmoire()
    {
        $this
            ->forAll(Set\Strings::between(1, 255))
            ->then(function(string $nom) {
                $response = $this->makePost('/api/armoires', [
                    'nom' => $nom,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    use BlackBox;

    public function testCreationGabaritDeDocument()
    {
        $this
            ->forAll(
                Set\Strings::between(1, 255),
                Set\Strings::between(1, 255),
            )
            ->then(function(string $nomArmoire, string $nomGabarit) {
                $armoire = $this->creerArmoire($nomArmoire);

                $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
                    'nom' => $nomGabarit,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    use BlackBox;

    public function testCreationGabaritDeDocument()
    {
        $this
            ->forAll(
                Set\Strings::between(1, 255),
                Set\Strings::between(1, 255),
            )
            ->then(function(string $nomArmoire, string $nomGabarit) {
                $armoire = $this->creerArmoire($nomArmoire);

                $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
                    'nom' => $nomGabarit,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    use BlackBox;

    public function testCreationGabaritDeDocument()
    {
        $this
            ->forAll(
                Set\Strings::between(1, 255),
                Set\Strings::between(1, 255),
            )
            ->then(function(string $nomArmoire, string $nomGabarit) {
                $armoire = $this->creerArmoire($nomArmoire);

                $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
                    'nom' => $nomGabarit,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    use BlackBox;

    public function testCreationGabaritDeDocument()
    {
        $this
            ->forAll(
                Set\Strings::between(1, 255),
                Set\Strings::between(1, 255),
            )
            ->then(function(string $nomArmoire, string $nomGabarit) {
                $armoire = $this->creerArmoire($nomArmoire);

                $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
                    'nom' => $nomGabarit,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

Tests dynamiques

```php
final class CreerArmoire
{
    public function __construct(private string $nom) {}

    public function __invoke(TestCase $test): array
    {
        $response = $test->makePost('/api/armoires', [
            'nom' => $this->nom,
        ]);

        $test->assertSame(201, $response->getStatusCode());

        return \json_decode($response->getContent(), true);
    }

    public static function any(): Set
    {
        return Set\Strings::between(1, 255)->map(
            static fn($nom) => new self($nom),
        );
    }
}
```

```php
final class CreerArmoire
{
    public function __construct(private string $nom) {}

    public function __invoke(TestCase $test): array
    {
        $response = $test->makePost('/api/armoires', [
            'nom' => $this->nom,
        ]);

        $test->assertSame(201, $response->getStatusCode());

        return \json_decode($response->getContent(), true);
    }

    public static function any(): Set
    {
        return Set\Strings::between(1, 255)->map(
            static fn($nom) => new self($nom),
        );
    }
}
```

```php
final class CreerArmoire
{
    public function __construct(private string $nom) {}

    public function __invoke(TestCase $test): array
    {
        $response = $test->makePost('/api/armoires', [
            'nom' => $this->nom,
        ]);

        $test->assertSame(201, $response->getStatusCode());

        return \json_decode($response->getContent(), true);
    }

    public static function any(): Set
    {
        return Set\Strings::between(1, 255)->map(
            static fn($nom) => new self($nom),
        );
    }
}
```

```php
final class CreerArmoire
{
    public function __construct(private string $nom) {}

    public function __invoke(TestCase $test): array
    {
        $response = $test->makePost('/api/armoires', [
            'nom' => $this->nom,
        ]);

        $test->assertSame(201, $response->getStatusCode());

        return \json_decode($response->getContent(), true);
    }

    public static function any(): Set
    {
        return Set\Strings::between(1, 255)->map(
            static fn($nom) => new self($nom),
        );
    }
}
```

```php
final class CreerArmoire
{
    public function __construct(private string $nom) {}

    public function __invoke(TestCase $test): array
    {
        $response = $test->makePost('/api/armoires', [
            'nom' => $this->nom,
        ]);

        $test->assertSame(201, $response->getStatusCode());

        return \json_decode($response->getContent(), true);
    }

    public static function any(): Set
    {
        return Set\Strings::between(1, 255)->map(
            static fn($nom) => new self($nom),
        );
    }
}
```

```php
final class CreerArmoire
{
    public function __construct(private string $nom) {}

    public function __invoke(TestCase $test): array
    {
        $response = $test->makePost('/api/armoires', [
            'nom' => $this->nom,
        ]);

        $test->assertSame(201, $response->getStatusCode());

        return \json_decode($response->getContent(), true);
    }

    public static function any(): Set
    {
        return Set\Strings::between(1, 255)->map(
            static fn($nom) => new self($nom),
        );
    }
}
```

```php
final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCreationArmoire()
    {
        $this
            ->forAll(CreerArmoire::any())
            ->then(function(CreerArmoire $creerArmoire) {
                $creerArmoire($this);
            });
    }
}
```

```php
final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCreationArmoire()
    {
        $this
            ->forAll(CreerArmoire::any())
            ->then(function(CreerArmoire $creerArmoire) {
                $creerArmoire($this);
            });
    }
}
```

```php
final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCreationArmoire()
    {
        $this
            ->forAll(CreerArmoire::any())
            ->then(function(CreerArmoire $creerArmoire) {
                $creerArmoire($this);
            });
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    use BlackBox;

    public function testCreationGabaritDeDocument()
    {
        $this
            ->forAll(
                CreerArmoire::any(),
                Set\Strings::between(1, 255),
            )
            ->then(function(CreerArmoire $creerArmoire, string $nomGabarit) {
                $armoire = $creerArmoire($this);

                $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
                    'nom' => $nomGabarit,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    use BlackBox;

    public function testCreationGabaritDeDocument()
    {
        $this
            ->forAll(
                CreerArmoire::any(),
                Set\Strings::between(1, 255),
            )
            ->then(function(CreerArmoire $creerArmoire, string $nomGabarit) {
                $armoire = $creerArmoire($this);

                $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
                    'nom' => $nomGabarit,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

```php
final class GabaritDeDocumentTest extends TestCase
{
    use BlackBox;

    public function testCreationGabaritDeDocument()
    {
        $this
            ->forAll(
                CreerArmoire::any(),
                Set\Strings::between(1, 255),
            )
            ->then(function(CreerArmoire $creerArmoire, string $nomGabarit) {
                $armoire = $creerArmoire($this);

                $response = $this->makePost("/api/armoires/{$armoire['id']}/gabarits-de-documents", [
                    'nom' => $nomGabarit,
                ]);

                $this->assertSame(201, $response->getStatusCode());
            });
    }
}
```

→ CreerArmoire

→ CreerGabaritDeDocument

→ CreerDocumentDansArmoire

→ CreerBannette

→ CreerDocumentDansBannette

VerrouillerDocument

→ CreerDocumentDansArmoire

→ CreerDocumentDansBannette

```php
final class VerrouillerDocument
{
    public function __construct(private $creerDocument) {}

    public function __invoke(TestCase $test)
    {
        $document = ($this->creerDocument)($test);

        // reste du test
    }

    public static function any(): Set
    {
        return Set\Either::any(
            CreerDocumentDansArmoire::any(),
            CreerDocumentDansBannette::any(),
        )->map(static fn($creerDocument) => new self($creerDocument));
    }
}
```

```php
final class VerrouillerDocument
{
    public function __construct(private $creerDocument) {}

    public function __invoke(TestCase $test)
    {
        $document = ($this->creerDocument)($test);

        // reste du test
    }

    public static function any(): Set
    {
        return Set\Either::any(
            CreerDocumentDansArmoire::any(),
            CreerDocumentDansBannette::any(),
        )->map(static fn($creerDocument) => new self($creerDocument));
    }
}
```

```php
final class VerrouillerDocument
{
    public function __construct(private $creerDocument) {}

    public function __invoke(TestCase $test)
    {
        $document = ($this->creerDocument)($test);

        // reste du test
    }

    public static function any(): Set
    {
        return Set\Either::any(
            CreerDocumentDansArmoire::any(),
            CreerDocumentDansBannette::any(),
        )->map(static fn($creerDocument) => new self($creerDocument));
    }
}
```

```php
final class VerrouillerDocument
{
    public function __construct(private $creerDocument) {}

    public function __invoke(TestCase $test)
    {
        $document = ($this->creerDocument)($test);

        // reste du test
    }

    public static function any(): Set
    {
        return Set\Either::any(
            CreerDocumentDansArmoire::any(),
            CreerDocumentDansBannette::any(),
        )->map(static fn($creerDocument) => new self($creerDocument));
    }
}
```

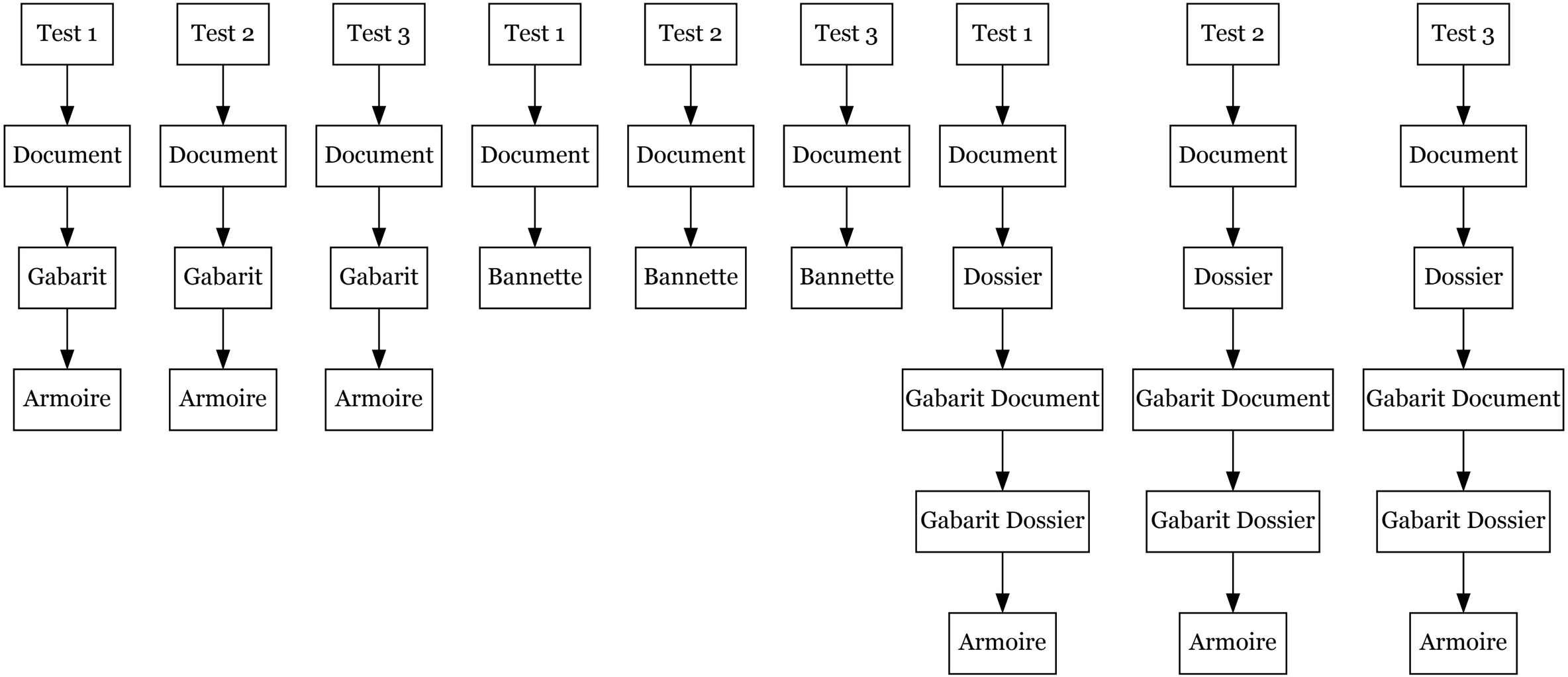
```php
final class VerrouillerDocument
{
    public function __construct(private $creerDocument) {}

    public function __invoke(TestCase $test)
    {
        $document = ($this->creerDocument)($test);

        // reste du test
    }

    public static function any(): Set
    {
        return Set\Either::any(
            CreerDocumentDansArmoire::any(),
            CreerDocumentDansBannette::any(),
        )->map(static fn($creerDocument) => new self($creerDocument));
    }
}
```

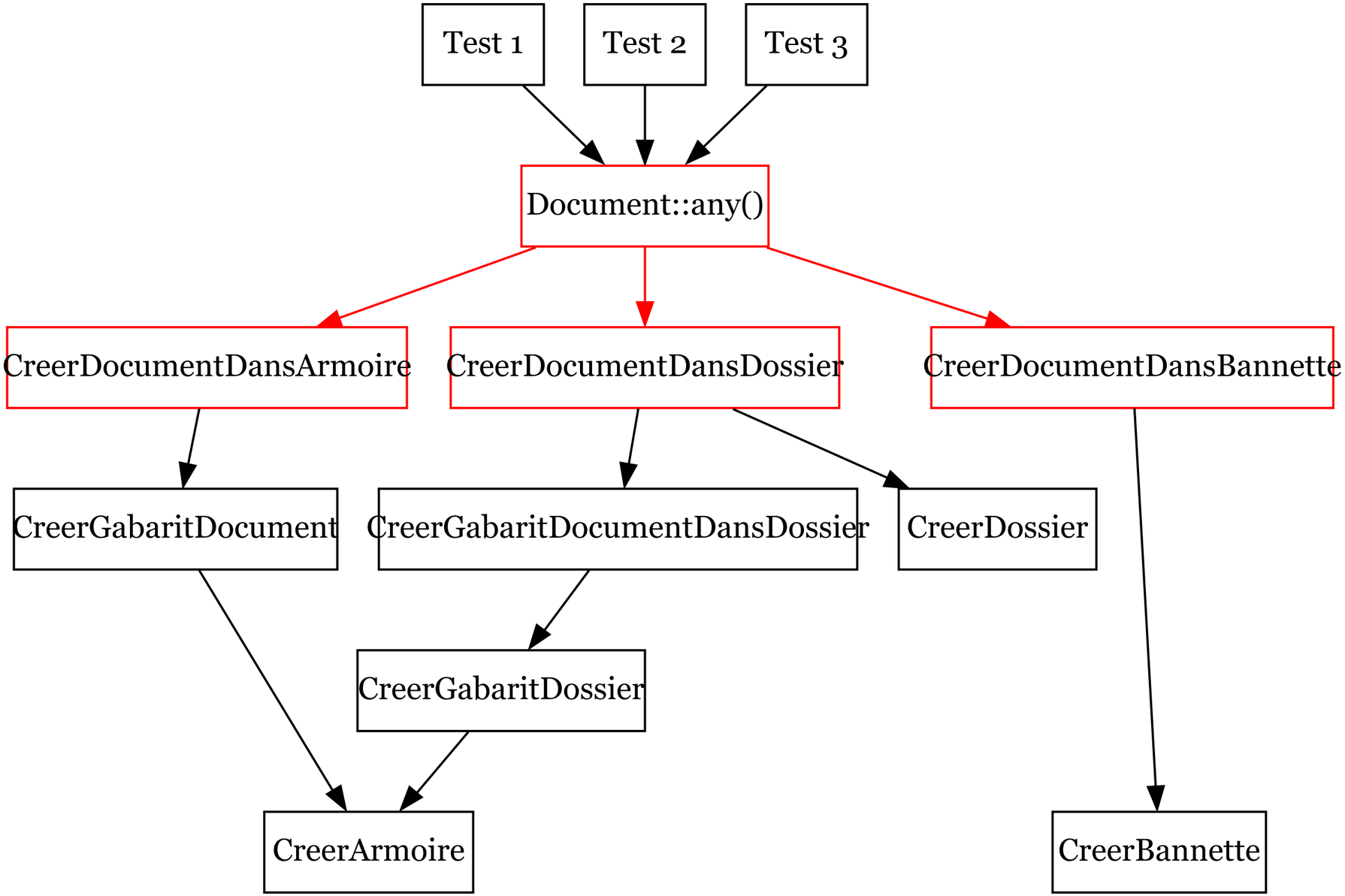
```php
final class VerrouillerDocument
{
    public function __construct(private $creerDocument) {}

    public function __invoke(TestCase $test)
    {
        $document = ($this->creerDocument)($test);

        // reste du test
    }

    public static function any(): Set
    {
        return Set\Either::any(
            CreerDocumentDansArmoire::any(),
            CreerDocumentDansBannette::any(),
        )->map(static fn($creerDocument) => new self($creerDocument));
    }
}
```

```php
namespace Fixtures;

final class Document
{
    public static function any(): Set
    {
        return Set\Either::any(
            CreerDocumentDansArmoire::any(),
            CreerDocumentDansBannette::any(),
        );
    }
}
```

```php
final class VerrouillerDocument
{
    public function __construct(private $creerDocument) {}

    public function __invoke(TestCase $test)
    {
        // implémentation
    }

    public static function any(): Set
    {
        return \Fixtures\Document::any()->map(
            static fn($creerDocument) => new self($creerDocument),
        );
    }
}
```

```php
namespace Fixtures;

final class Document
{
    public static function any(): Set
    {
        return Set\Either::any(
            CreerDocumentDansArmoire::any(),
            CreerDocumentDansBannette::any(),
            CreerDocumentDansDossier::any(),
        );
    }
}
```


Test 1
Document
Gabarit
Armoire
Test 2
Document
Gabarit
Armoire
Test 3
Document
Gabarit
Armoire
Test 1
Document
Bannette
Test 2
Document
Bannette
Test 3
Document
Bannette
Test 1
Document
Dossier
Gabarit Document
Gabarit Dossier
Armoire
Test 2
Document
Dossier
Gabarit Document
Gabarit Dossier
Armoire
Test 3
Document
Dossier
Gabarit Document
Gabarit Dossier
Armoire


Test 1
Test 2
Test 3
Document::any()
CreerDocumentDansArmoire
CreerDocumentDansDossier
CreerDocumentDansBannette
CreerGabaritDocument
CreerGabaritDocumentDansDossier
CreerDossier
CreerGabaritDossier
CreerArmoire
CreerBannette

Test non régression

```php
final class ArmoireTest extends TestCase
{
    public function testNonRegression()
    {
        $creerArmoire = new CreerArmoire('nom invalide');

        $creerArmoire($this);
    }
}
```

```
[positive] App\Tests\Controller\RechercheTest::teste la possibilité de filtrer la recherche de dossiers sur plusieurs métadonnées inclusives:
F

$nom = "4rûêj §i€2è@ikyS9üu!6^ê2ü1I-ù4îZa4Hïw>2èöhl2î@éH£5!îjïüe:363M4j:èîr28ê. 8êp9é¨03fè7ç'ï08".0ü:*ùèôûûkçY^*558!wFôîlL?2rûû0J=26&15&è0ugoS9ûô !ö4F9ü"
Failed
Proofs: 333, Scenarii: 333, Assertions: 6164, Failures: 1
$gabaritDeDossier = array:3 [
  "nom" => "éüm"ï1B:1'60àH5%6V^0ü5¨/L§8ly<üôv256afUö40üPfFKy=8u€06*,6R61û2.39üà#5Sà:5£ê14Sùï'à='4+ïWM5#ö,(=@<ôdZ6ù2öê6/û%=95ç57@+A1ü5KX5jê3>!S?8ü&3ï3*îAg;N_f%?ûüô$X làRùpé5àV?rbz5ö2ô9ùv2#9>@7d@'7ô+Kü=1ü83ûû9§=¨ïzùï6jùûp1!r9ïû9GéiP77)HC1)0&8z)Wüî40"1ü#ê$ 3"
  "description" => "9€Pî65nWîl6I?^?fïè8K@1G4ûi2èfö5qü0931üô&è7§23,43îz1ù $8&'-#:_V/1#"6ùc&i0QZ'1tü%UöïO-gà£CpïeR0ù9à%jîü9*6ü9+Eïp1x§î3ûôX=9LKôqx6îB/58Bî2>#?wï9ül@56q30_q3,%J3Rôà*2(ü365à"
  "metadonnees" => array:1 [
    0 => array:4 [
      "format" => "texte"
      "libelle" => "bïûG7üîFü7IKéaç4g/£<ü)td*7ô4&_öïà.LqQ3îêö!)0,3j70êô0ö51oçq0r#8öZôw6ê60üBïXôçôù6ö-4+7ô7ö!Aö%*è0J0"6580"Fô6IX@(7T9349£52ùn,èû3;=4éH^Hù3dîQ,wU1ïQf(ûG5ûpö_éyûàï0ôXRr<mî89)!!éî@=$/75.2AqN:à>12àü(?ù<48<5Z,3û9Jê^d60dû<6Yû78"
      "aide" => "o9Lz4QAç&.G45ö@T#"aéJA>s+üù"
      "obligatoire" => false
    ]
  ]
]
$rechercheCroisée = array:3 [
  0 => "1%"
  1 => "€ri"
  2 => "¨§,w16!tysohpks66vk"
]

$expected = 0
$actual = 1

Failed to assert that a collection contains 0 element(s)

/__w/mgx/mgx/api/tests/Controller/RechercheTest.php:3929
/__w/mgx/mgx/api/tests/Controller/RechercheTest.php:3860
/__w/mgx/mgx/api/tests.php:43
```

```
$rechercheCroisée = array:3 [
  0 => "1%"
  1 => "€ri"
  2 => "¨§,w16!tysohpks66vk"
]

$expected = 0
$actual = 1

Failed to assert that a collection contains 0 element(s)
```


$rechercheCroisée = array:3 [
0 => "1%"
1 => "€ri"
2 => "¨§,w16!tysohpks66vk"
]
$expected = 0
$actual = 1
Failed to assert that a collection contains 0 element(s)

Composition

- `Set\Strings::madeOf(...Set)`
- `Set->filter()`
- `Set\Composite(callable, ...Set)`
- etc...

Futur

Parcours utilisateur

```php
final class SimulationTest extends TestCase
{
    use BlackBox;

    public function testParcoursUtilisateur()
    {
        $this
            ->forAll(Set\Sequence::of(
                Set\Either::any(
                    CreerDocumentDansArmoire::any(),
                    CreerBannette::any(),
                    // etc...
                ),
            )->atLeast(2))
            ->then(function(array $actions) {
                foreach ($actions as $action) {
                    $action($this);
                }
            });
    }
}
```

```php
final class SimulationTest extends TestCase
{
    use BlackBox;

    public function testParcoursUtilisateur()
    {
        $this
            ->forAll(Set\Sequence::of(
                Set\Either::any(
                    CreerDocumentDansArmoire::any(),
                    CreerBannette::any(),
                    // etc...
                ),
            )->atLeast(2))
            ->then(function(array $actions) {
                foreach ($actions as $action) {
                    $action($this);
                }
            });
    }
}
```

```php
final class SimulationTest extends TestCase
{
    use BlackBox;

    public function testParcoursUtilisateur()
    {
        $this
            ->forAll(Set\Sequence::of(
                Set\Either::any(
                    CreerDocumentDansArmoire::any(),
                    CreerBannette::any(),
                    // etc...
                ),
            )->atLeast(2))
            ->then(function(array $actions) {
                foreach ($actions as $action) {
                    $action($this);
                }
            });
    }
}
```

Tests en conditions réelles

```php
final class CreerArmoire
{
    public function __construct(private string $nom) {}

    public function __invoke(TestCase $test): array
    {
        $response = $test->makePost('/api/armoires', [
            'nom' => $this->nom,
        ]);

        $test->assertSame(201, $response->getStatusCode());

        return \json_decode($response->getContent(), true);
    }

    public static function any(): Set
    {
        return Set\Strings::between(1, 255)->map(
            static fn($nom) => new self($nom),
        );
    }
}
```

Model Checker

Questions

Twitter @Baptouuuu

Github @Baptouuuu/talks