

**Augmentez votre  
couverture :  
supprimez des tests !**

- Baptiste Langlade
- Lyon
- 10+ ans XP
- ~95 packages Open Source



**GED**

→ Armoires

→ Gabarits de documents

→ Documents

→ Bannettes

→ Documents

# **Tests fonctionnels**

→ Armoires

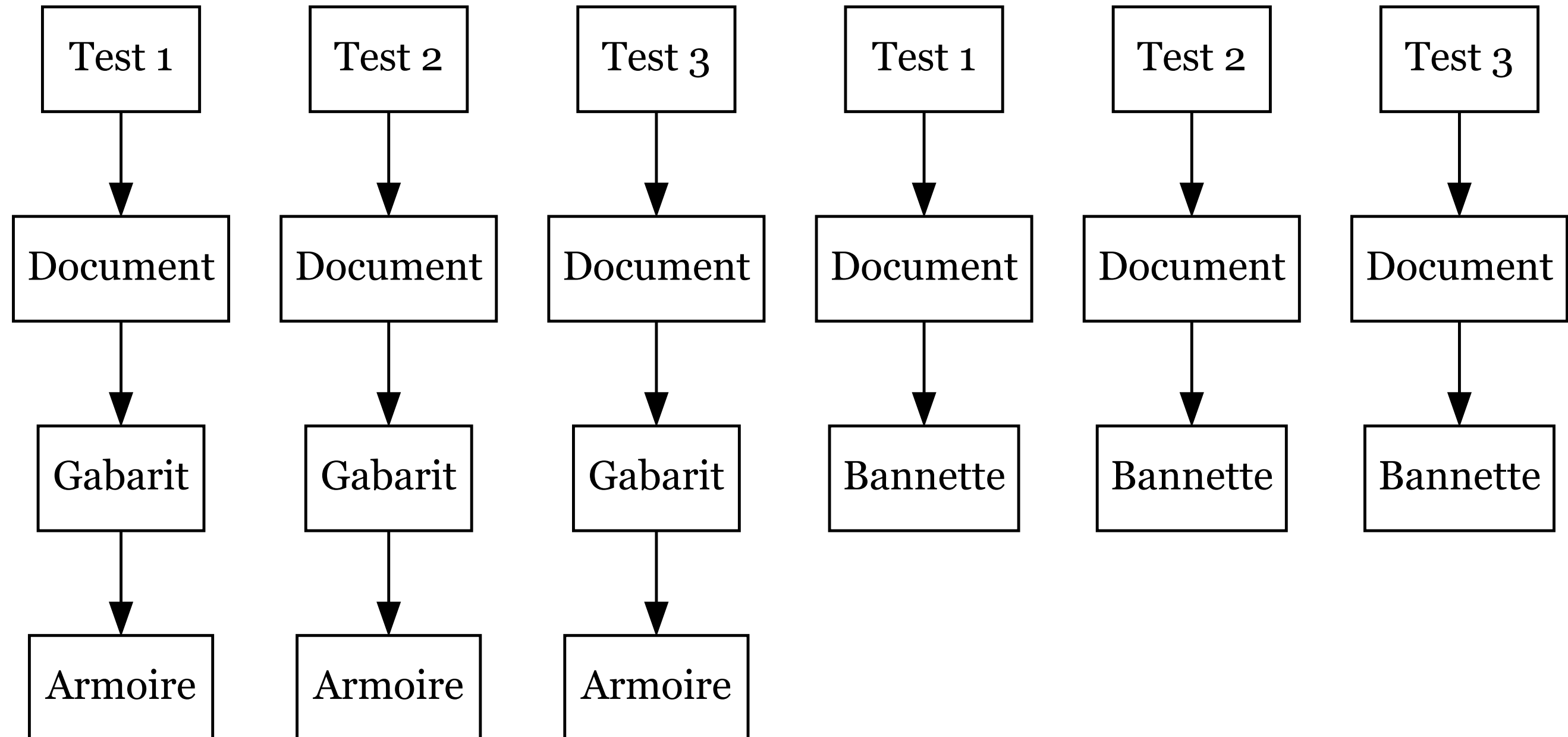
→ Gabarits de documents

→ **Documents**

→ Bannettes

→ **Documents**

# Verrouiller document





→ Armoires

→ Gabarits de documents

→ **Documents**

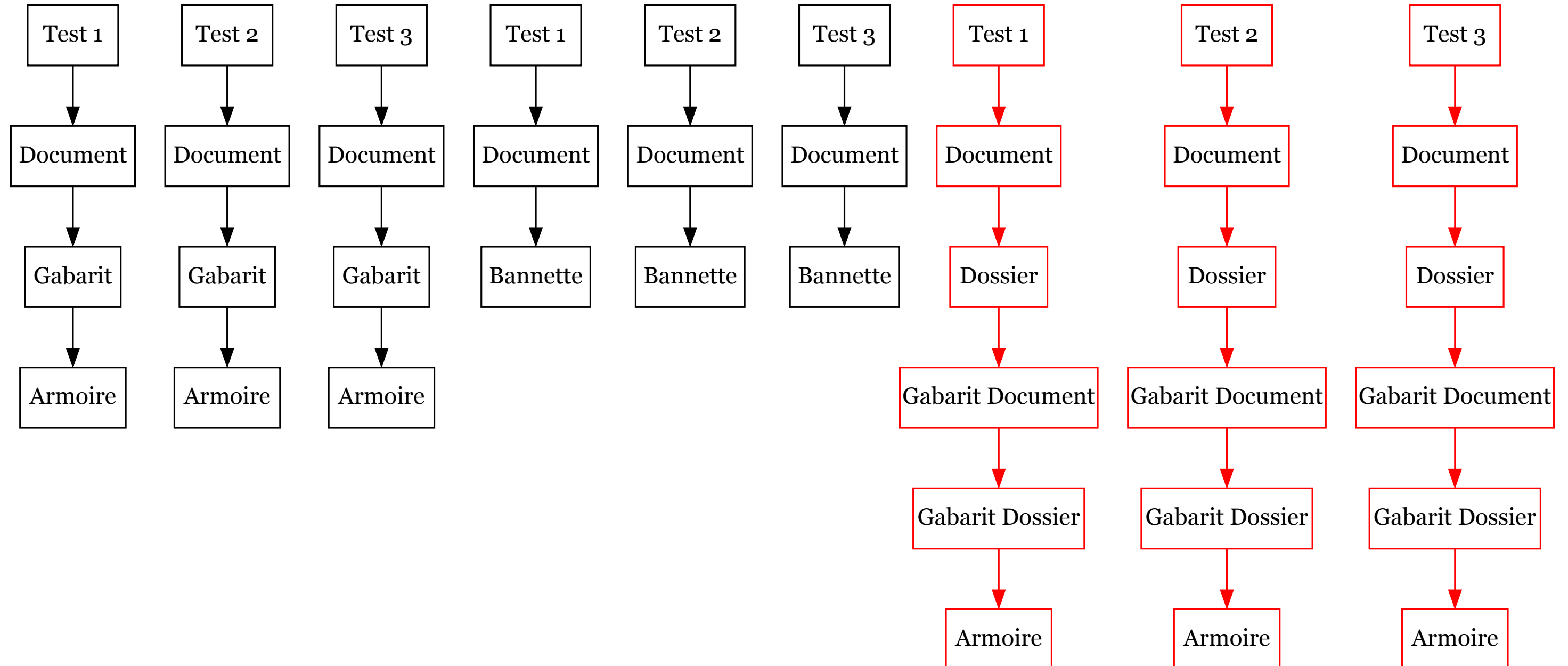
→ Gabarits de dossiers

→ 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**

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

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

```
final class GabaritDeDocumentTest extends TestCase
{
    public function testCréationGabaritDeDocument()
    {
        $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());
    }
}
```

```
final class GabaritDeDocumentTest extends TestCase
{
    public function testCréationGabaritDeDocument()
    {
        $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());
    }
}
```

```
final class GabaritDeDocumentTest extends TestCase
{
    public function testCréationGabaritDeDocument()
    {
        $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());
    }
}
```

```
final class GabaritDeDocumentTest extends TestCase
{
    public function testCréationGabaritDeDocument()
    {
        $armoire = $this->créerArmoire('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
```



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

final class ArmoireTest extends TestCase
{
    use BlackBox;

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

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

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

final class ArmoireTest extends TestCase
{
    use BlackBox;

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

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

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

final class ArmoireTest extends TestCase
{
    use BlackBox;

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

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

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

final class ArmoireTest extends TestCase
{
    use BlackBox;

    public function testCréationArmoire()
    {
        $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 je peux  
créer une armoire***

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Tests dynamiques

```
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),
        );
    }
}
```

```
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),
        );
    }
}
```

```
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),
        );
    }
}
```



```
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),
        );
    }
}
```

```
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),
        );
    }
}
```

```
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),
        );
    }
}
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

        // reste du test
    }

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

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

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

        // reste du test
    }

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

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

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

        // reste du test
    }

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

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

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

        // reste du test
    }

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

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

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

        // reste du test
    }

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



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

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

        // reste du test
    }

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

```
namespace Fixtures;
```

```
final class Document
```

```
{
```

```
    public static function any(): Set
```

```
{
```

```
        return Set\Either::any(
```

```
            CreerDocumentDansArmoire::any(),
```

```
            CreerDocumentDansBannette::any(),
```

```
        );
```

```
    }
```

```
}
```

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

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

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

```
namespace Fixtures;
```

```
final class Document
```

```
{
```

```
    public static function any(): Set
```

```
{
```

```
        return Set\Either::any(
```

```
            CreerDocumentDansArmoire::any(),
```

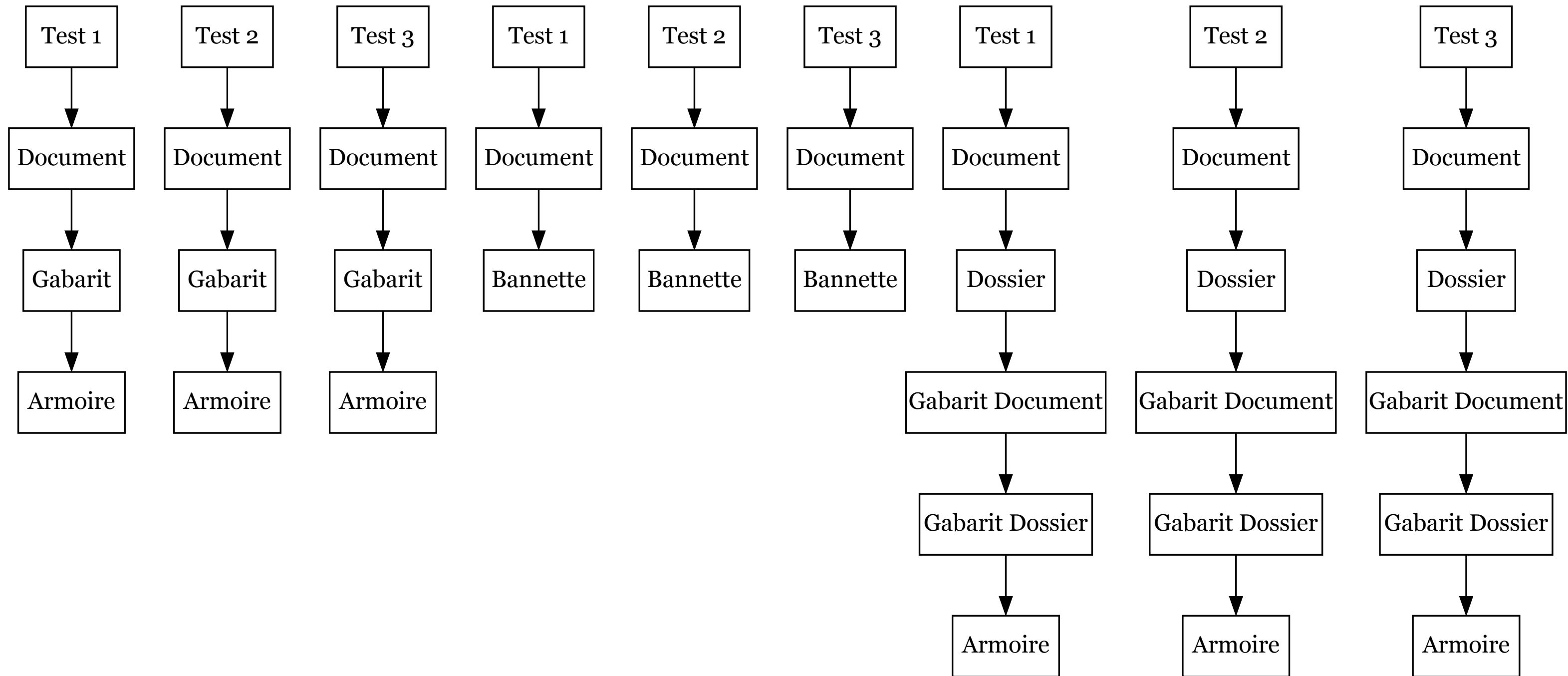
```
            CreerDocumentDansBannette::any(),
```

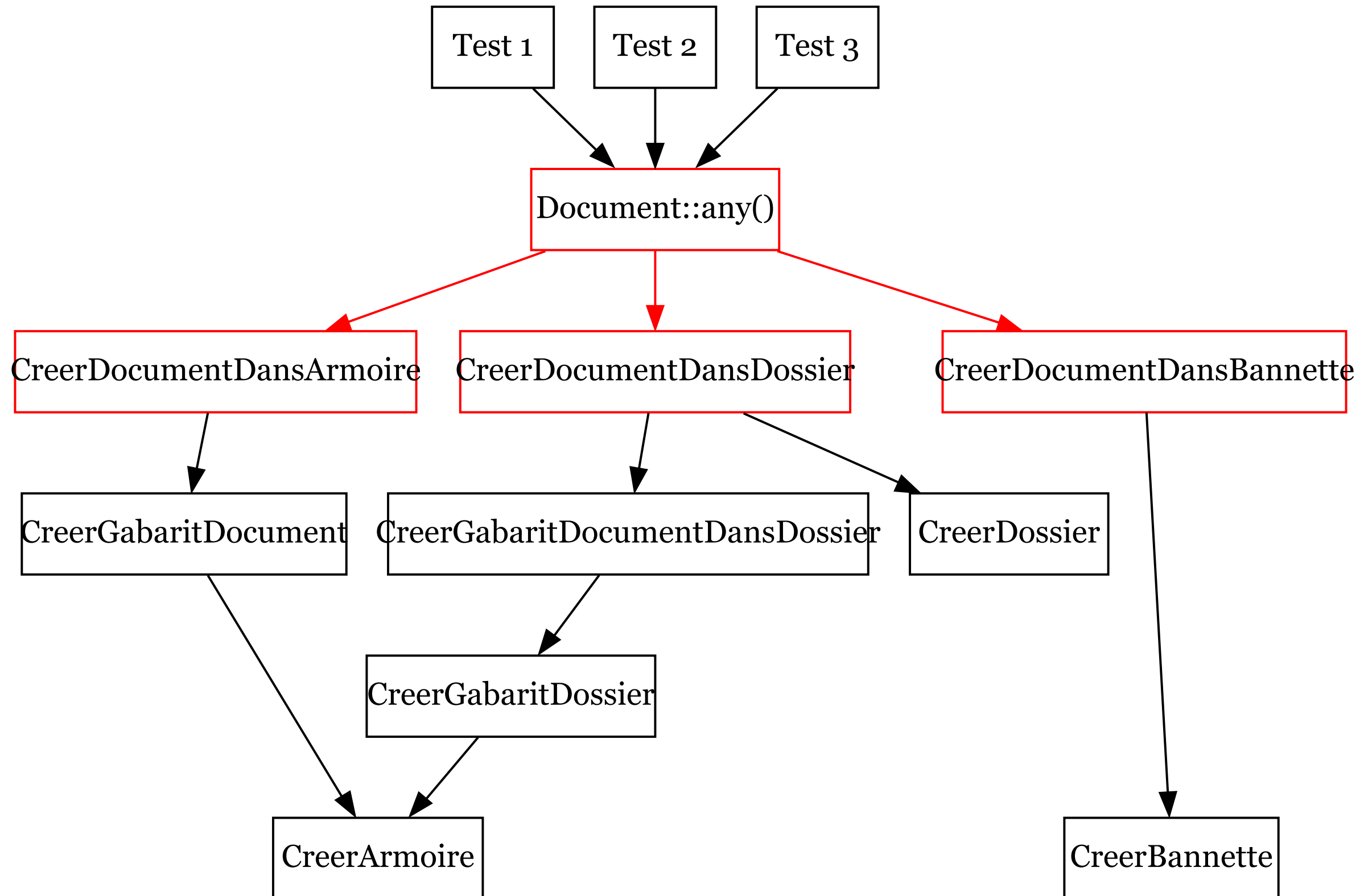
```
            CreerDocumentDansDossier::any(),
```

```
        );
```

```
    }
```

```
}
```





# **Test non régression**

```
final class ArmoireTest extends TestCase
{
    public function testNonRégression()
    {
        $créerArmoire = new CreerArmoire('nom invalide');

        $créerArmoire($this);
    }
}
```



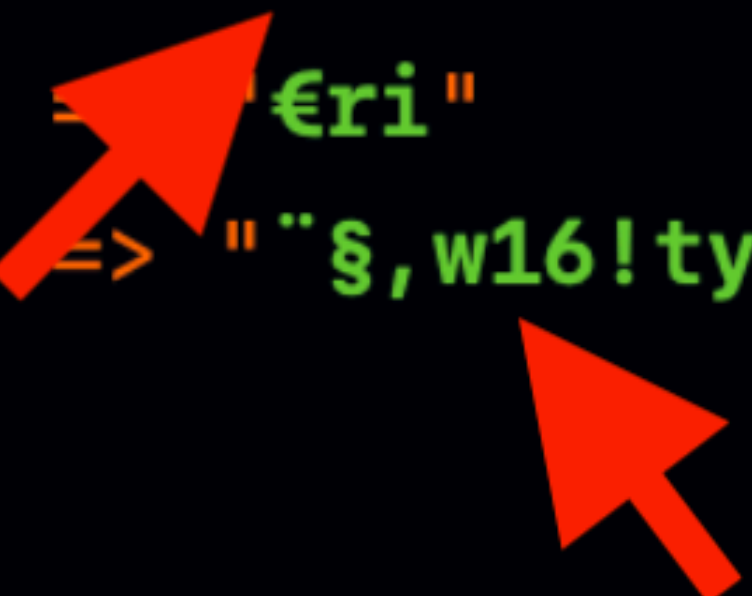
```
$rechercheCroisée = array:3 [  
  0 => "1%"  
  1 => "€ri"  
  2 => "``§,w16!tysohpks66vk"  
]
```

```
$expected = 0
```

```
$actual = 1
```

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

```
$rechercheCroisée = array:3 [  
  0 => "1%"  
  1 => "€ri"  
  2 => "§,w16!tysohpk66vk"  
]
```

Two red arrows are present. One arrow points from the left towards the element at index 1, "€ri". The second arrow points from the bottom right towards the element at index 2, "§,w16!tysohpk66vk".

```
$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**

```
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);
                }
            });
    }
}
```

```
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);
                }
            });
    }
}
```



```
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**

```
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