Valider dynamiquement de la donnée avec Symfony

Marion Hurteau
github.com/MarionLeHerisson/validation

Hello World 👋

Marion Hurteau

@MarionHerisson

/MarionLeHerisson

marion.hurteau1@gmail.com

👩‍🎓 Architecture Logicielle @ESGI

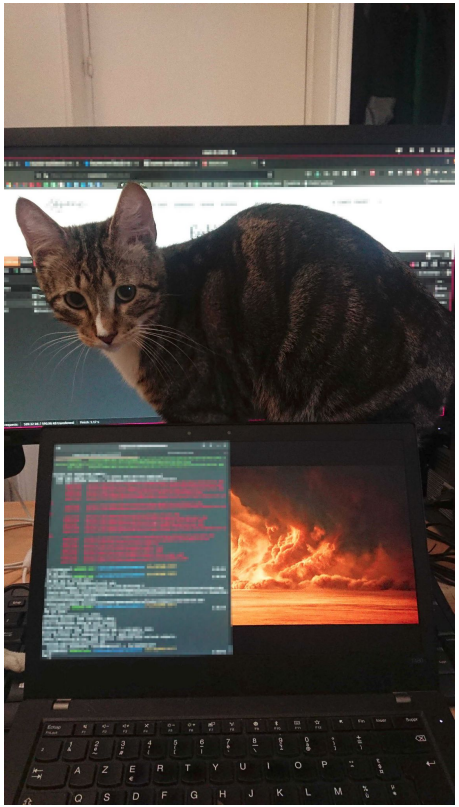
👩‍💻 JoliCode depuis 2019

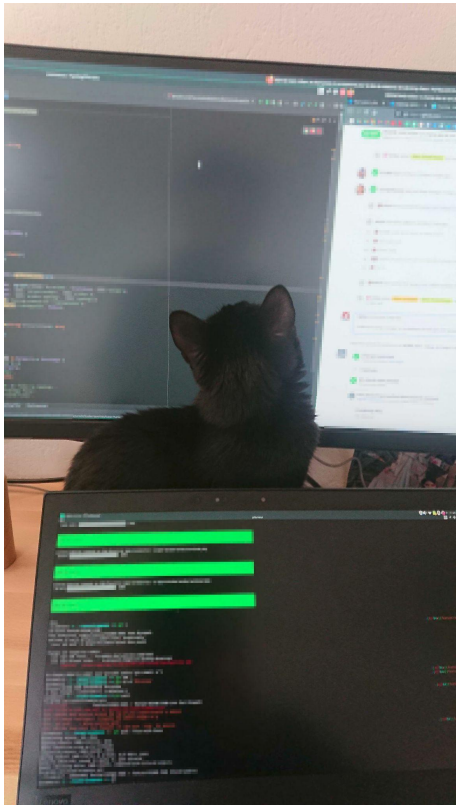
Hello World 👋

🐈‍⬛ Nessie & Oscar 🐈

🌱 ~40 plantes

💻 RPG



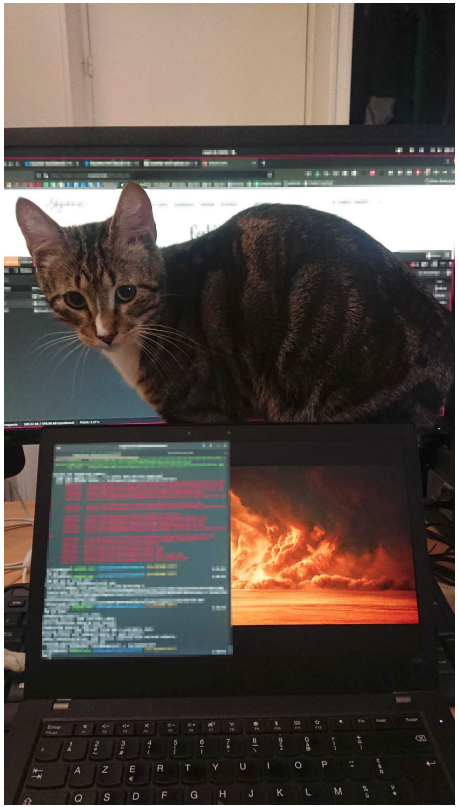


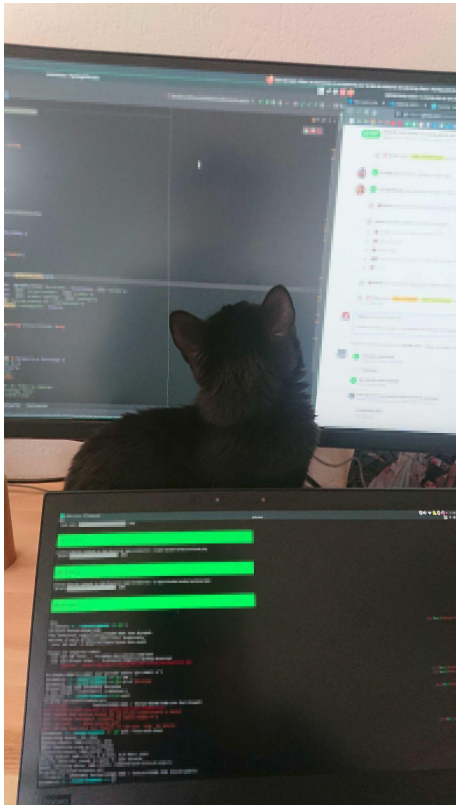
Hello World 👋

🐈‍⬛ Nessie & Oscar 🐈

🌱 ~40 plantes

💻💻 RPG 



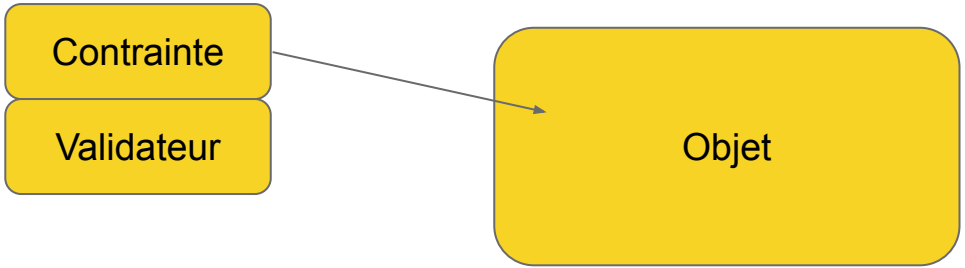


La Validation ?

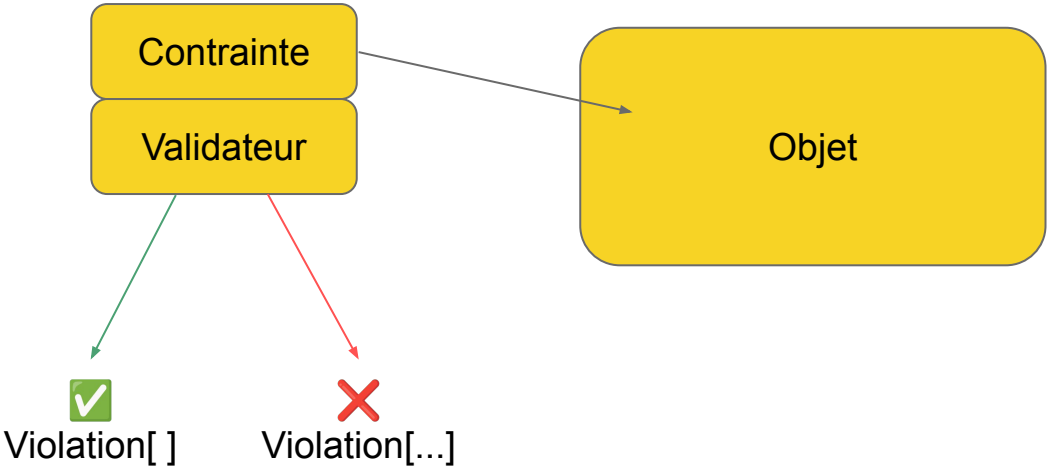
ABRA's nickname?

_ _ _ _ _ _ _ _ _ _ _

Principe



Principe



Contrainte

```php
#[\Attribute...]
class NotBlank extends Constraint
{
    ...
}
```

Validateur

```php
class NotBlankValidator extends ConstraintValidator
{
    public function validate(mixed $val, Constraint $cons)
    {
        ...
    }
}
```

Entité

```php
class Pokemon
{
    #[Assert\NotBlank]
    private string $name;

    ...
}
```

Violations ?

```
Symfony\Component\Validator\ConstraintViolation {#282 ▼
  -message: "This value should be of type int."
  -messageTemplate: "This value should be of type {{ type }}."
  -parameters: array:2 [▶]
  -plural: null
  -root: App\Entity\Evolution {#254 ▶}
  -propertyPath: "[level]"
  -invalidValue: "FOO"
  -constraint: Symfony\Component\Validator\Constraints\Type {#273 ▶}
  -code: "ba785a8c-82cb-4283-967c-3cf342181b40"
  -cause: null
}
```

Les contraintes

Il en existe beaucoup !

Blank, NotBlank

IsNull, NotNull

IsTrue, IsFalse

Type, Length

```php
#[Assert\Type('string')]
#[Assert\Length(min: 1, max: 10)]
protected string $name;
```

```php
#[Assert\IsFalse]
protected bool $beatTheLeague;
```

```php
#[Assert\IsNull]
protected \DateTime $deletedAt;
```

Il en existe beaucoup !

Email, Regex

Url, Hostname, Ip

CssColor

NotCompromisedPassword

```php
#[Assert\Regex('/^\w+/')]
protected string $description;

#[Assert\Regex(
    pattern: '/\d/',
    message: 'Your name cannot contain a number',
    match: false,
)]
protected string $name;
```

```php
#[Assert\CssColor]
protected string $defaultColor;
// red, #369, #A2A2A2, hsla(0, 0%, 20%, 0.4)
```

```php
#[Assert\NotCompromisedPassword]
protected string $rawPassword;
```

Il en existe beaucoup !

Date, DateTime, Time, Timezone

Language, Locale, Country

File, Image

```php
#[Assert\Country]
protected string $country;
// FR, DE, IT

#[Assert\Country(alpha3: true)]
protected string $country;
// FRA, DEU, ITA
```

Il en existe beaucoup !

Bic, Ibn, CardScheme

Isbn, Issn, Isin

```php
#[Assert\CardScheme(
    schemes: [Assert\CardScheme::AMEX]
)]
protected string $cardNumber;
```

Il en existe beaucoup !

Choice

```php
const ELEMENTS = ['fire', 'water', 'plant']

#[Assert\Choice(choices: self::ELEMENTS)]
public string $element;
```

All, AtLeastOneOf, Sequentially

```php
#[Assert\AtLeastOneOf([
    new Assert\Length(min: 10),
    new Assert\Email(),
])]
protected string $userName;
```

Il en existe beaucoup !

Compound

```php
class PasswordRequirements extends Compound
{
    protected function getConstraints(array $options): array
    {
        return [
            new Assert\NotBlank(),
            new Assert\Type('string'),
            new Assert\Length(['min' => 12]),
            new Assert\NotCompromisedPassword(),
        ];
    }
}
```

Il en existe beaucoup !

Compound

Callback

```php
class Author
{
    #[Assert\Callback]
    public function validate(ExecutionContextInterface $ctx, $payload)
    {
        // ...
    }
}
```

Il en existe beaucoup !

Compound

Callback

Expression

```php
class BlogPost
{
    ...

    public static function loadValidatorMetadata(ClassMetadata $metadata)
    {
        $metadata->addPropertyConstraint('isTechnicalPost', new Assert\Expression([
            'expression' => 'this.getCategory() in ["php", "symfony"] or value == false',
        ]));
    }

    ...
}
```

Il en existe beaucoup !

Compound

Callback

Expression

Valid

```php
class Pokemon
{
    #[Assert\Valid]
    private Element $element;
}
```

```php
class Element
{
    #[Assert\NotBlank]
    #[Assert\Length(max: 5)]
    protected string $name;

    #[Assert\NotBlank]
    protected array $effectiveAgainst;
}
```

Aller un peu plus loin

UniqueEntity

```php
#[ORM\Entity]
#[UniqueEntity(
    fields: ['host', 'port'],
    errorPath: 'port',
    message: 'This port is already in use on that host.',
)]
class Service
{
```

```php
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity]
#[UniqueEntity('email')]
class User
{
    #[ORM\Column(name: 'email', type: 'string', length: 255, unique: true)]
    #[Assert\Email]
    protected $email;
}
```

Auto-mapping validation

```php
#[ORM\Entity]
#[Assert\EnableAutoMapping()]
class Pokemon
{
    #[Column(type: "integer", nullable: false)]
    private string $id;

    #[Column(type: "string", length: 25)]
    private string $name;

    ...
}
```

Auto-mapping validation

```yaml
# config/packages/validator.yaml

framework:
    validation:
        ...
        auto_mapping:
            App\Entity\: []
```

Différentes manières de valider

Sur une propriété

```php
class Pokemon
{
    #[Assert\NotBlank]
    #[Assert\Length(
        min: 3
    )]
    private string $name;

    ...
}
```

Dans un formulaire

```
$builder
    ->add('name', TextType::class, [
        'required' => true,
        'constraints' => [
            new NotBlank(),
            new Length(['min' => 3]),
        ],
    ])
;
```

Sur une méthode (un getter ou autre)

```php
#[Assert\IsTrue(message: 'The password cannot match your name.')]
public function isPasswordSafe(): bool
{
    // ... return true or false
}
```

Sur une classe entière

```php
#[Assert\Callback]
public function validate(ExecutionContextInterface $context, $payload): void
{
    // check stuff

    if (/* some condition */) {
        $context->buildViolation('Error !')
            ->atPath('name')
            ->addViolation();
    }
}
```

Sur une classe entière

```php
#[Assert\Callback('myCustomValidateFunction')]
class Pokemon
{
    ...

    public function myCustomValidateFunction($object, ExecutionContextInterface $context, $payload)
    {
        // Validation logic
    }
}
```

Quand ?

Dans mon code

```php
$violations = $validator->validate($pikachu);
```

Lors d'un submit

```php
// ValidationListener.php:50

$this->validator->validate($form);
```

Quand ?

Groupes de validation

```php
class Pokemon
{
    #[Assert\Length(min: 1, max: 10, groups: ['creation'])]
    private string $name;

    #[Assert\NotBlank(groups: ['creation'])]
    private array $attacks;

    #[Assert\NotBlank]
    private int $level;
}
```

Créer sa contrainte

Contrainte personnalisée

```php
use App\Validator\Constraint\ContainsAlphanumeric;

class Author
{
    #[ContainsAlphanumeric]
    private string $name;

    ...
}
```

Contrainte personnalisée

```php
#[\Attribute]
class ContainsAlphanumeric extends Constraint
{
    public $message = 'The string "{{ string }}" is not valid';

    public $path = 'name';
}
```

Une contrainte

Un validateur

```php
class ContainsAlphanumericValidator extends ConstraintValidator
{
    public function validate($value, Constraint $constraint)
    {
        // validation logic

        $this->context->buildViolation($constraint->message)
                ->setParameter('{{ string }}', $value)
                ->addViolation();
    }
}
```

Contrainte personnalisée

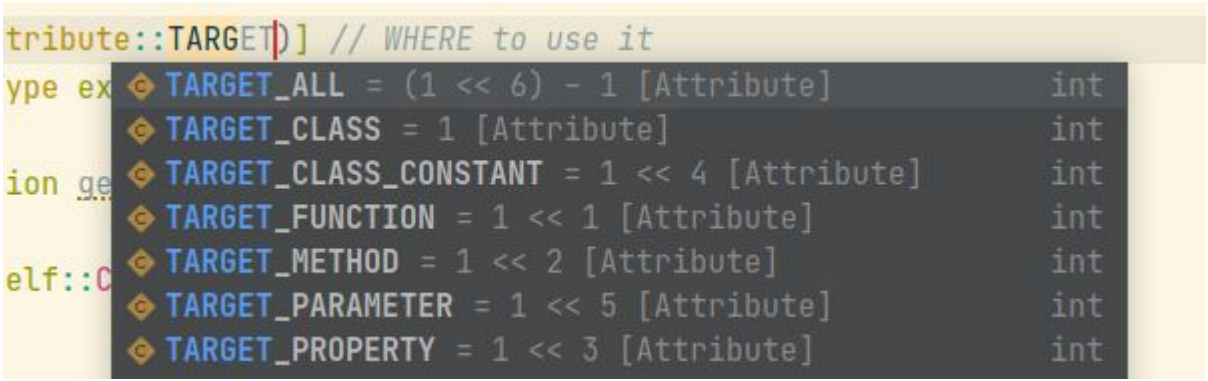
```php
#[\Attribute(\Attribute::TARGET_CLASS)]          // WHERE to use it
class ContainsAlphanumeric extends Constraint
{
    public $message = 'The string "{{ string }}" is invalid.';

    public $path = 'name';

    public function getTargets()
    {
        return self::CLASS_CONSTRAINT;          // WHAT it applies to
    }

    public function validatedBy()
    {
        return static::class.'Validator';
    }
}
```

Contrainte personnalisée



Contrainte personnalisée

- Besoin d'un service externe pour valider ? Non → ExpressionLanguage
- Besoin de réutiliser ma logique ailleurs ? Non → Callback

Oui & oui → Contrainte personnalisée !

Dynamiques ?

Attends, c'est quoi un Pokémon ?

Attends, c'est quoi un Pokémon ?

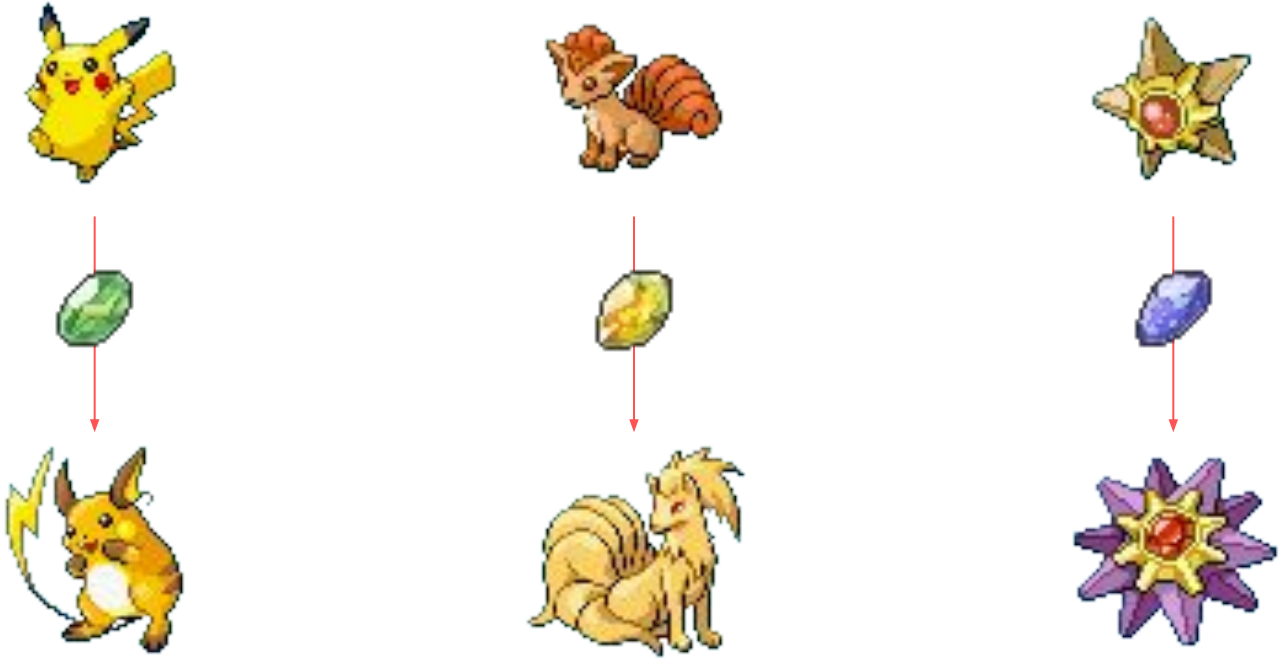
plante

feu

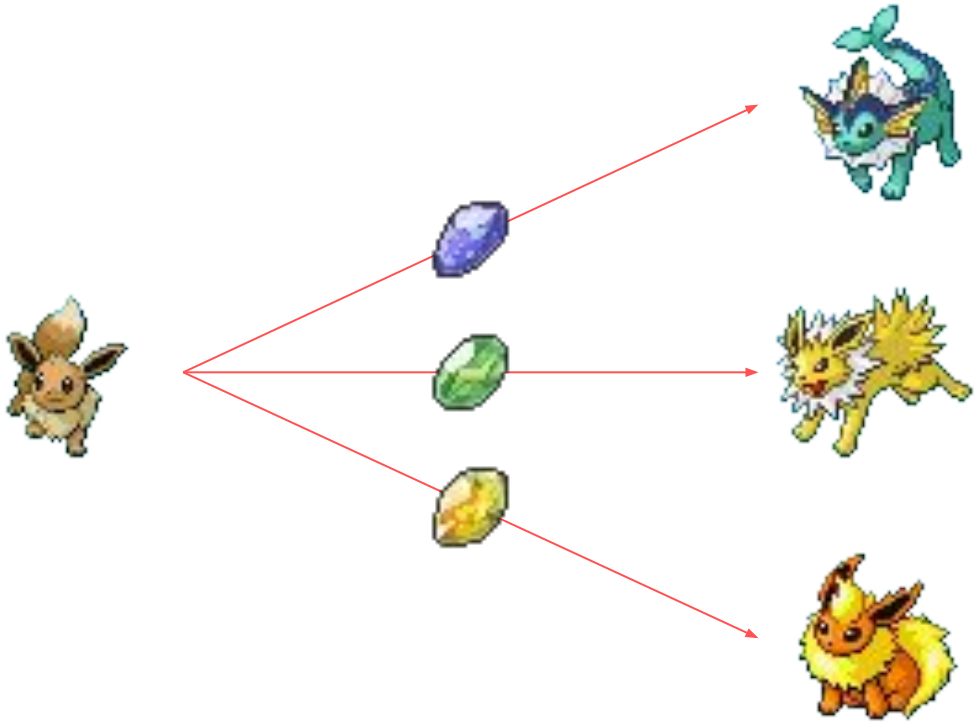
eau

Attends, c'est quoi un Pokémon ?

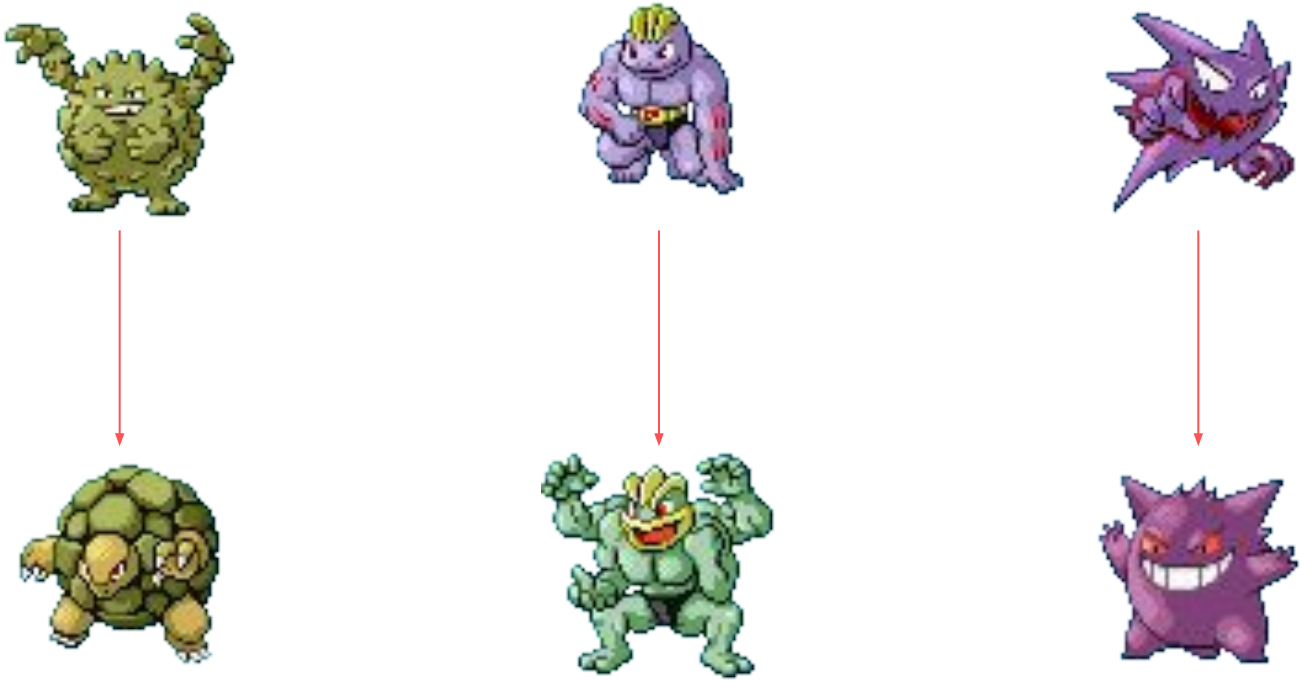
Attends, c'est quoi un Pokémon ?



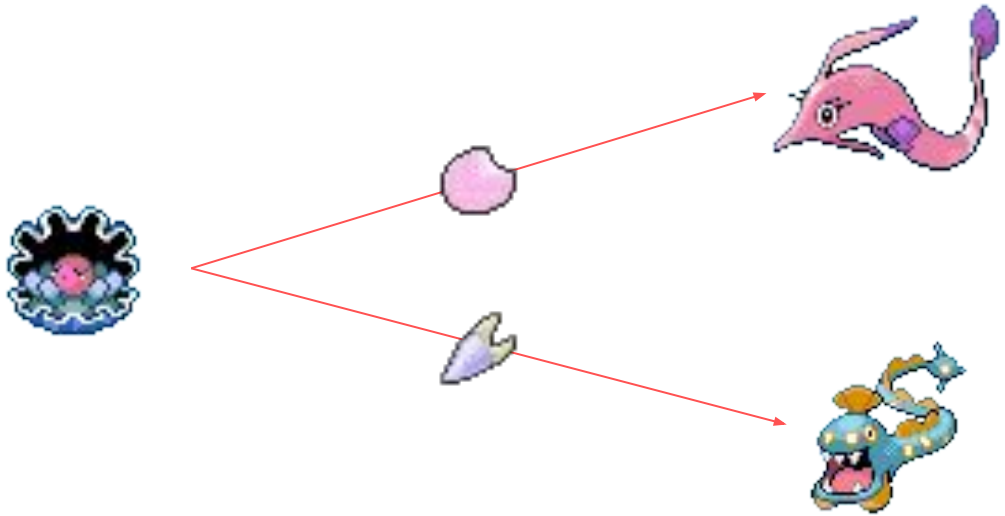
Attends, c'est quoi un Pokémon ?



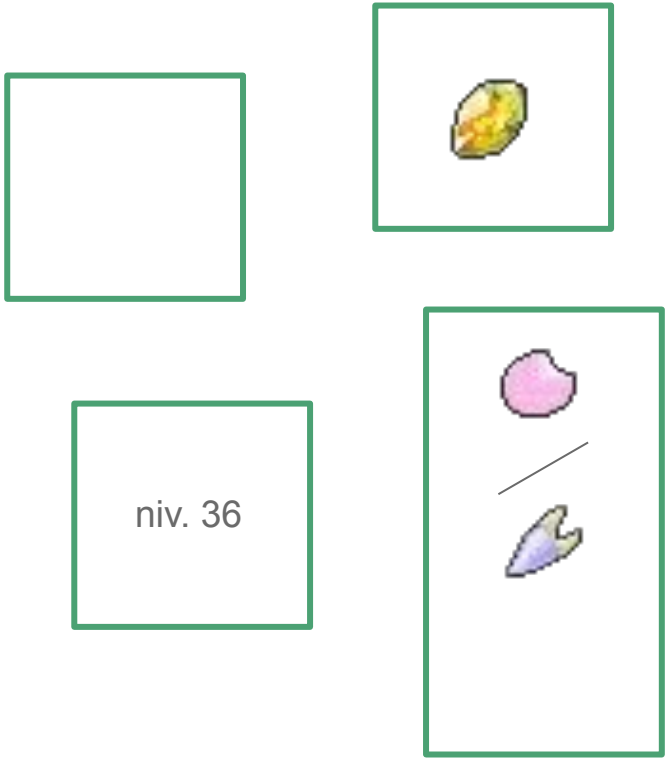
Attends, c'est quoi un Pokémon ?

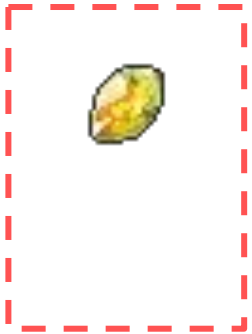


Attends, c'est quoi un Pokémon ?



Cas d'utilisation





Cas d'utilisation

POST /evolution/create

```
{
    "type": "Evolution type name",
    "options": {
        "key": "some options"
    }
}
```



```
class Evolution
{
    public string $type;

    public array $options = [];
}
```

Cas d'utilisation

```
{
    "type": "level",
    "options": {
        "level": 36
    }
}
```

Cas d'utilisation

```
{
    "type": "stone",
    "options": {
        "stones": [
            "water",
            "fire",
            "electric"
        ]
    }
}
```

Avec une Callback

Dans le controller

```php
#[Route('/evolve', name: 'evolve')]
public function evolve(Request $request, ValidatorInterface $validator): Response
{
    ...

    $evoliEvolution = new Evolution();
    $evoliEvolution->setType($stoneEvolution['type']);
    $evoliEvolution->setOptions($stoneEvolution['options']);

    $violations = $validator->validate($evoliEvolution);

    ...
}
```

L'entité

```php
class Evolution
{
    public string $type;
    public array $options = [];

    #[Assert\Callback]
    public function validate(ExecutionContextInterface $context): void
    {
        ...
    }

    private function getLevelConstraints(): array
    {
        ...
    }

    private function getStoneConstraints(): array
    {
        ...
    }

    // Might get more constraints
}
```

Les contraintes dans l'entité

```php
// App\Entity\Evolution.php
private function getLevelConstraints(): array
{
    return [
        new Assert\Collection([
            'level' => [
                new Assert\Type('int'),
            ],
        ]),
    ];
}
```

```json
{
    "type": "level",
    "options": {
        "level": 36
    }
}
```

Les contraintes dans l'entité

```php
// App\Entity\Evolution.php
private function getStoneConstraints(): array
{
    return [
        new Assert\Collection([
            'stones' => [
                new Assert\All([
                    new Assert\Type('string'),
                ])
            ],
        ]),
    ];
}
```

```json
{
    "type": "stone",
    "options": {
        "stones": [
            "water",
            "fire",
            "electric"
        ]
    }
}
```

Les contraintes dans l'entité

```php
// App\Entity\Evolution.php
private function getStoneConstraints(): array
{
    return [
        new Assert\Collection([
            'stones' => [
                new Assert\All([
                    new Assert\Type('string'),
                ])
            ],
        ]),
    ];
}
```

```json
{
    "type": "stone",
    "options": {
        "stones": [
            "water",
            "fire",
            "electric"
        ]
    }
}
```

Les contraintes dans l'entité

```php
// App\Entity\Evolution.php
private function getStoneConstraints(): array
{
    return [
        new Assert\Collection([
            'stones' => [
                new Assert\All([
                    new Assert\Type('string'),
                ])
            ],
        ]),
    ];
}
```

```json
{
    "type": "stone",
    "options": {
        "stones": [
            "water",
            "fire",
            "electric"
        ]
    }
}
```

Les contraintes dans l'entité

```php
// App\Entity\Evolution.php
private function getStoneConstraints(): array
{
    return [
        new Assert\Collection([
            'stones' => [
                new Assert\All([
                    new Assert\Type('string'),
                ])
            ],
        ]),
    ];
}
```

```json
{
    "type": "stone",
    "options": {
        "stones": [
            "water",
            "fire",
            "electric"
        ]
    }
}
```

La callback (1/2)

```php
#[Assert\Callback]
public function validate(ExecutionContextInterface $context): void
{
    ...

    $constraints = match ($this->type) {
        'level' => $this->getLevelConstraints(),
        'stone' => $this->getStoneConstraints(),
        ...
        default => throw new \UnexpectedValueException(),
    };

    ...
```

La callback (2/2)

```php
#[Assert\Callback]
public function validate(ExecutionContextInterface $context): void
{
    ...

    $context
            ->getValidator()
            ->inContext($context)
            ->atPath('options')
            ->validate($this->options, $constraints)
        ;
    }
}
```

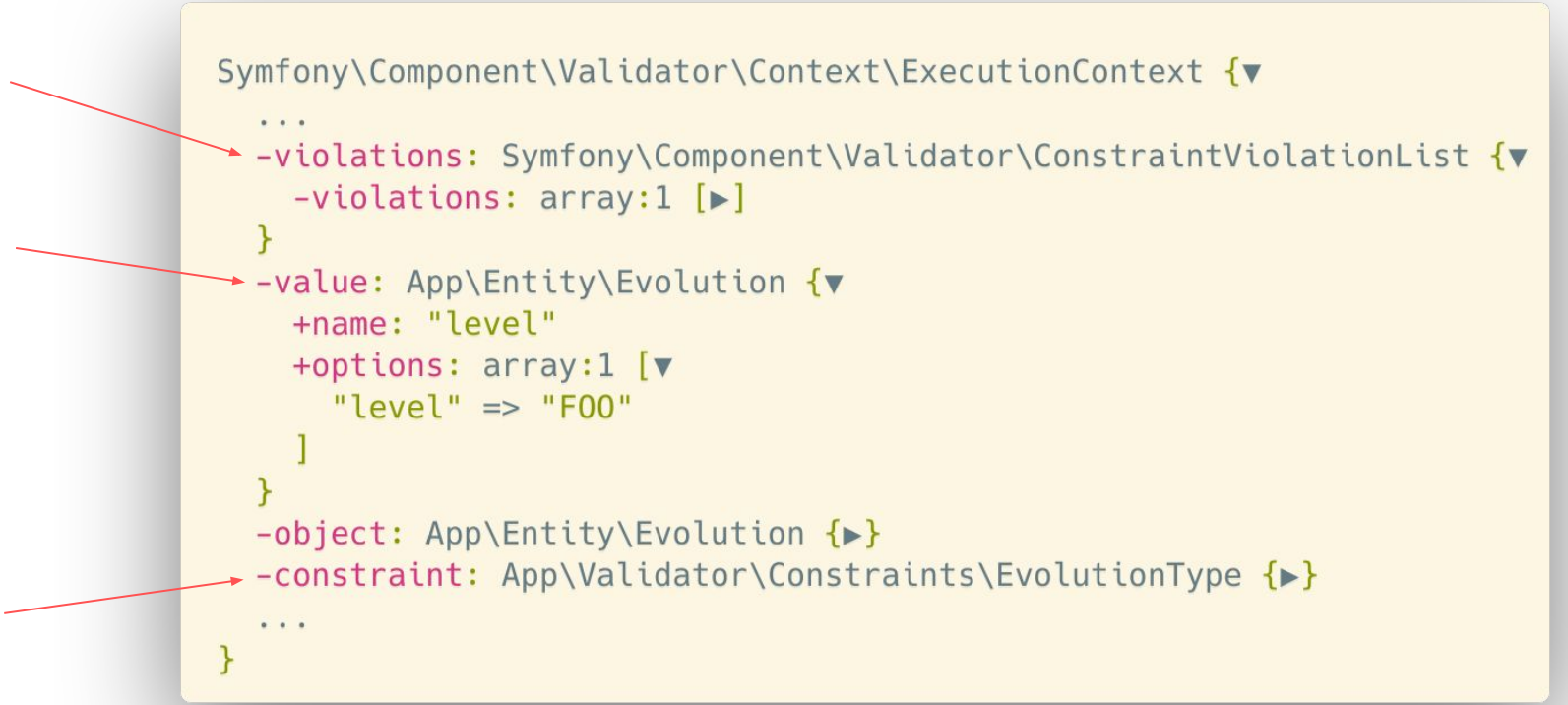
La callback (2/2)

```php
#[Assert\Callback]
public function validate(ExecutionContextInterface $context): void
{
    ...

    $context
            ->getValidator()
            ->inContext($context)
            ->atPath('options')
            ->validate($this->options
        ;
    }
}
```

```json
{
    "type": "stone",
    "options": {
        "stones": [
            "water",
            "fire",
            "electric"
        ]
    }
}
```

ExecutionContextInterface ?



Design

👍 Rapide

Design

👍 Rapide

👎 Réutilisabilité

👎 Taille de la méthode / classe

👎 Testabilité


AIRMURE♀ N.:32
PV
NIRONDELLE♂ N.:15
PV
40/ 40
EXP
Ce n'est pas très efficace...

Design

👉 Externaliser les contraintes

👉 DIC

Avec des contraintes externes

La Constraint

```php
#[\Attribute(\Attribute::TARGET_CLASS)]
class EvolutionType extends Constraint
{
    public function getTargets()
    {
        return self::CLASS_CONSTRAINT;
    }
}
```

La Constraint

```php
#[\Attribute(\Attribute:
class EvolutionType exte
{
    public function getTargets()
    {
        return self::CLASS_CONSTRAINT;
    }
}
```

```php
#[EvolutionType]
class Evolution
{
    ...
}
```

Ajoutons une interface ✨

```
{
    "type": "level",
    "options": {
        "level": 36
    }
}
```

```
interface EvolutionDefinitionInterface
{
    public static function getType(): string;

    public function getConstraints(): array;
}
```

Définition d'un type

```php
class LevelEvolutionTypeDefinition implements EvolutionDefinitionInterface
{
    public static function getType(): string
    {
        return 'level';
    }

    public function getConstraints(): array
    {
        return [
            new Assert\Collection([
                'level' => [
                    new Assert\Type('int'),
                ],
            ]),
        ];
    }
}
```

Définition d'un type

```php
class StoneEvolutionTypeDefinition implements EvolutionDefinitionInterface
{
    public static function getType(): string
    {
        return 'stone';
    }

    public function getConstraints(): array
    {
        return [
            new Assert\Collection([
                'stones' => [
                    new Assert\All([
                        new Assert\Type('string'),
                    ])
                ],
            ]),
        ];
    }
}
```

On ajoute des tags

```yaml
services:
    ...

    _instanceof:
        App\Evolution\EvolutionDefinitionInterface:
            tags:
                - { name: evolution.type.definition }
    ...

    App\Validator\Constraints\EvolutionTypeValidator:
        arguments:
            $definitions: !tagged_locator { tag: evolution.type.definition, default_index_method: 'getType' }
```

ContainerInterface

```
Symfony\Component\DependencyInjection\Argument\ServiceLocator {▼
  ...
  -serviceTypes: array:2 [▼
    "level" => "App\Evolution\LevelEvolutionTypeDefinition"
    "stone" => "App\Evolution\StoneEvolutionTypeDefinition"
    ...
  ]
}
```

Le ConstraintValidator (1/2)

```php
class EvolutionTypeValidator extends ConstraintValidator
{
    public function __construct(
        private ContainerInterface $definitions,
    ) {}

    public function validate($evolution, Constraint $constraint)
    {
        $type = $evolution->getType() ?? null;

        ...

        if (!$this->definitions->has($type)) {
            $this->context
                ->buildViolation('The type "{typeName}" is not valid.')
                ->setParameter('{typeName}', $type)
                ->addViolation()
            ;

            return;
        }

        ...
```

Le ConstraintValidator (2/2)

```
        ...

        /** @var $definition EvolutionDefinitionInterface */
        $definition = $this->definitions->get($type);

        $this->context
            ->getValidator()
            ->inContext($this->context)
            ->validate($evolution->getOptions(), $definition->getConstraints())
        ;
    }
}
```


MASSKO♂ N.:31
PV
FLOBIO♂ N.:34
PV
0/103
EXP
C'est super efficace!

Design

👉 Contrainte externe :

- Maintenable
- Testable
- Réutilisable

👉 _instanceof & !tagged_locator

- + default_index_method

D’autres cas d’utilisation

```
{
    "type": "responseStatusCodes",
    "options": {
        "statusCodes": [
            200
        ]
    }
}
```

```
{
    "type": "requestMethods",
    "options": {
        "methods": [
            "GET",
            "PUT"
        ]
    }
}
```

```
{
    "type": "url",
    "options": {
        "url": "https://www.pokebip.com/"
    }
}
```

TL;DR:

👉 Validez !

✨ $context->getValidator()->inContext($context)

Et ensuite ?

✅ Validées

👉 Persister

Avec Doctrine ?

En json ?

Merci !

github.com/MarionLeHerisson/validation

Des questions ?

github.com/MarionLeHerisson/validation

@MarionHerisson
marion.hurteau1@gmail.com