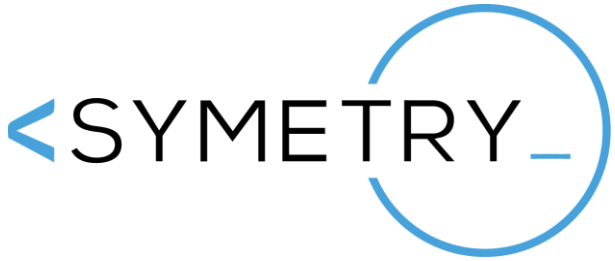
SYMETRY

Formation
Api Platform

fizen

Licence

Formation ApiPlatform



Découverte Api Platform

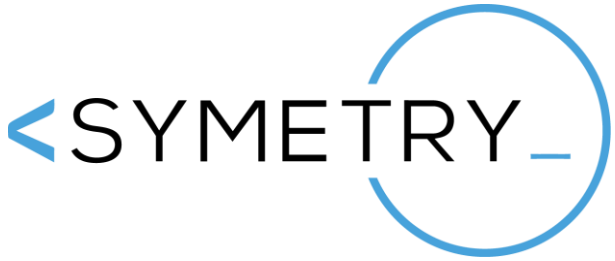
Qu’est-ce qu’Api Platform

- Un bundle Symfony pour créer des APIs
- Compatible REST / GraphQL
- Basé sur la configuration pour l’API
  - Documentation générée automatiquement basée sur la configuration

Intégration dans Symfony

- Couplé à doctrine pour générer ses apis

- Basé sur la configuration des entités pour l'API
  - Documentation générée automatiquement basée sur la configuration



Installation

Installation

Création d’un nouveau projet

```
# Create new empty symfony project called apip-formation
composer create-project symfony/skeleton apip-formation

# Switch to project directory
cd apip-formation

# Require debug pack to help debugging
composer require debug-pack --with-all-dependencies --dev

# Optional: Install security & other tools
composer require security maker

# Install api platform
composer require api
```

Lancer le projet

```
php -S localhost:8000 -t public
```

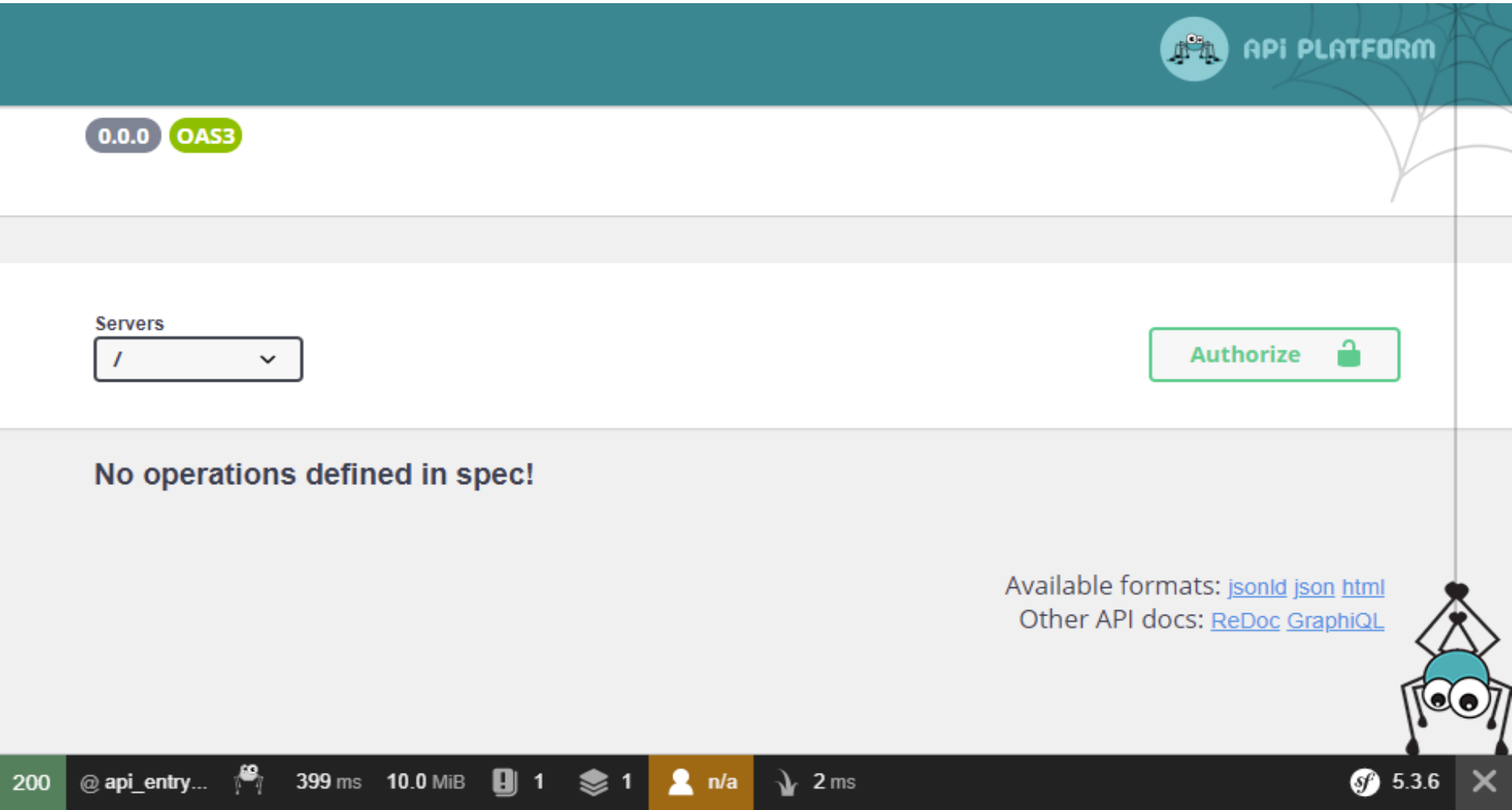
Installation

Visit http://localhost:8000/api

Connecter la base de données (.env)

Utiliser Sqlite pour la base de données

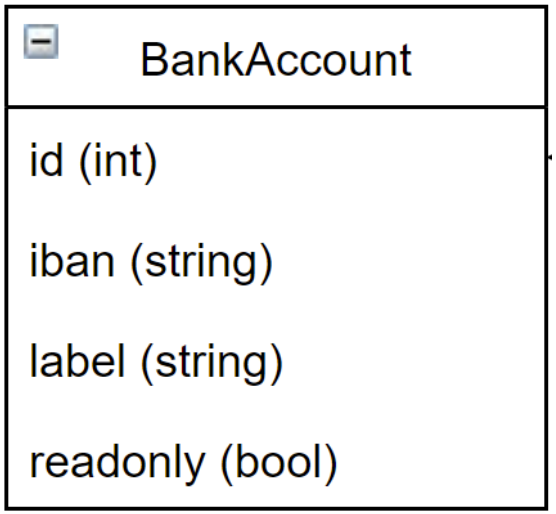
Installation



Exercice

Créer une entité compte bancaire

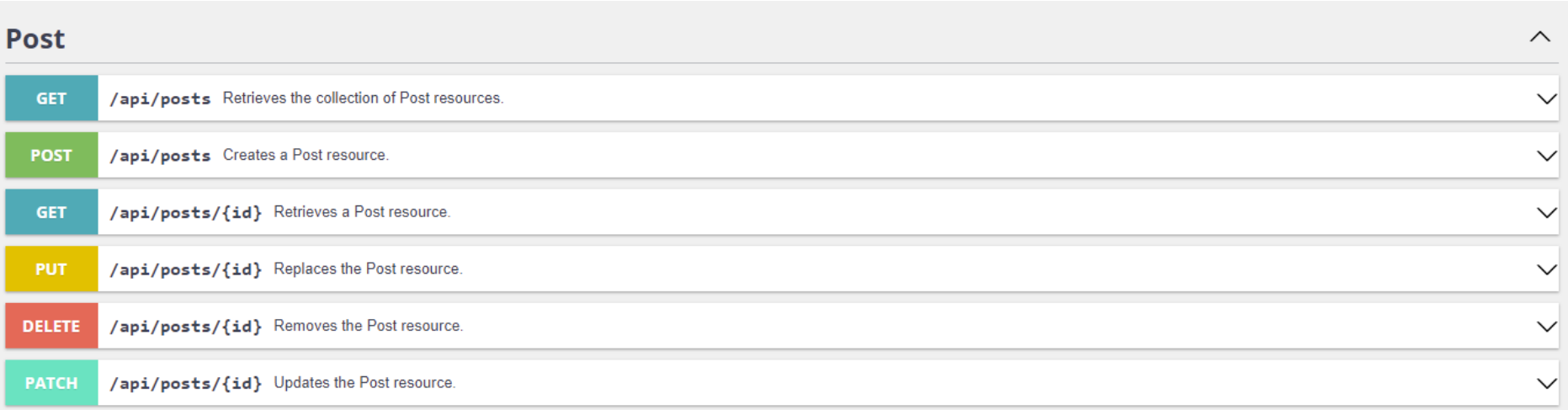
```
bin/console make:entity
```



Expose entity as Api resource

```php
/**
 * @ORM\Entity
 * @ApiResource
 */
class Post
{
    private $id;
}
```

```php
#[ORM\Entity]
#[ApiResource]
class Post
{
    private $id;
}
```



GET /api/posts


```
{
    "@context": "/api/contexts/Post",
    "@id": "/api/posts",
    "@type": "hydra:Collection",
    "hydra:member": [
      {
        "@id": "/api/posts/1",
        "@type": "Post",
        "id": 1,
        "title": "test"
      },
      {
        "@id": "/api/posts/2",
        "@type": "Post",
        "id": 2,
        "title": "test"
      }
    ],
    "hydra:totalItems": 2
  }
```

GET /api/posts/1


```
{
    "@context": "/api/contexts/Post",
    "@id": "/api/posts/1",
    "@type": "Post",
    "id": 1,
    "title": "test"
}
```

Personnaliser les endpoints

```
#[ApiResource(
    collectionOperations: ['get', 'post'],
    itemOperations: [
        'get' => [
            'security' => 'is_granted("READ", object.getPost())',
        ],
    ],
    shortName: 'the_comments',
    paginationEnabled: false
)]
```

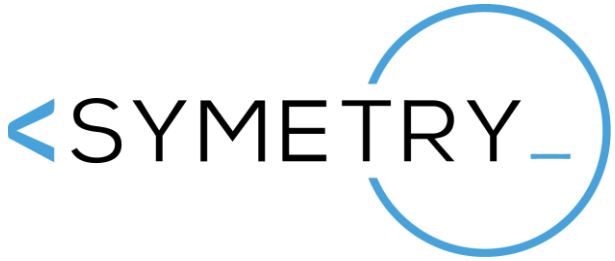
Exercice

Modifier l'entité BankAccount

Ne garder que les endpoints suivants

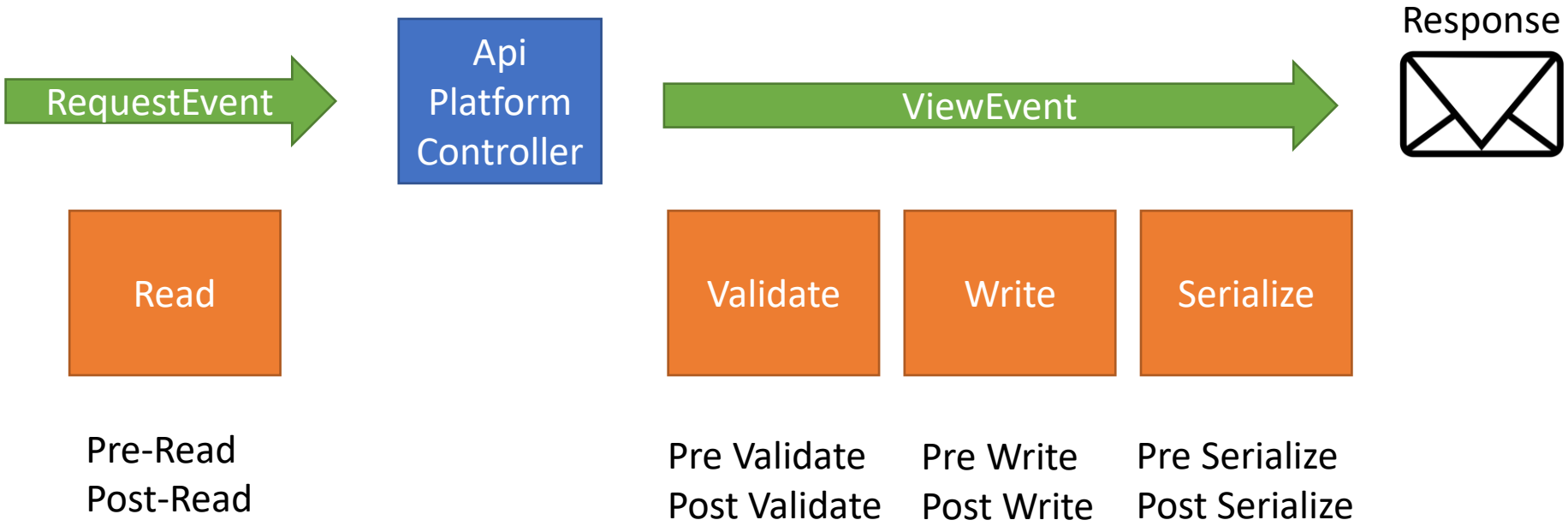
- Liste des comptes bancaires
- l'affichage d'un compte bancaire
- Modification de compte bancaire

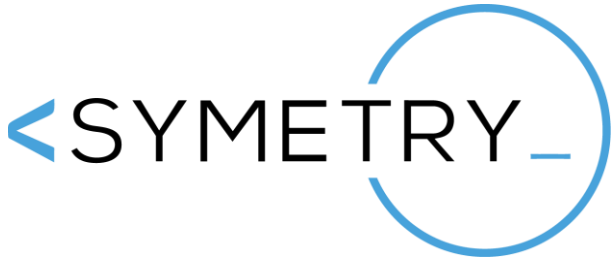
Désactiver la suppression



Events

Event workflow



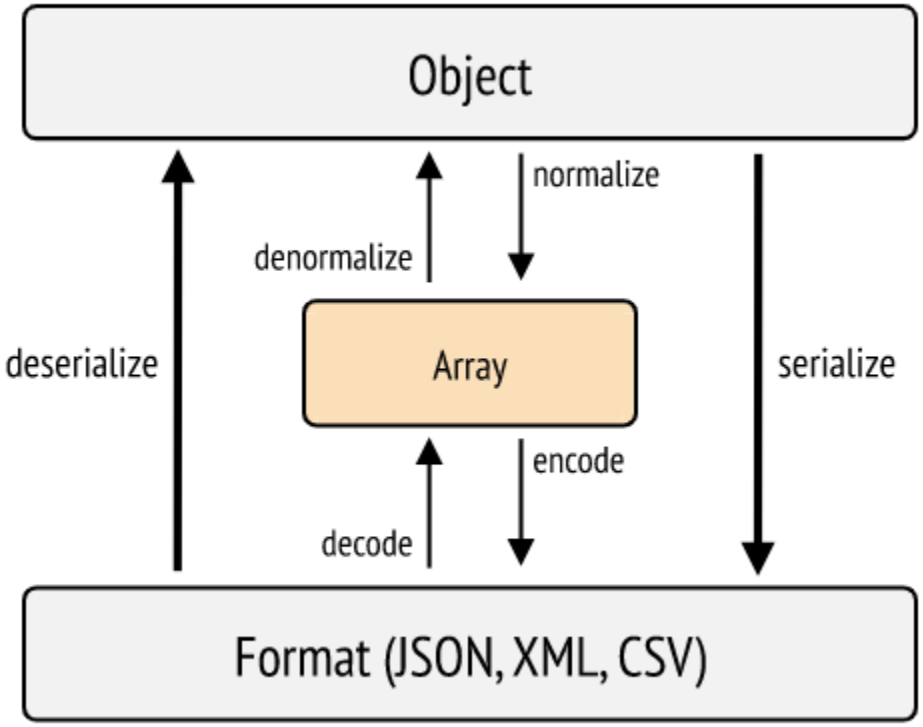


Serializer

Principes

- Le serializer fonctionne sur deux principes
  - Permet de convertir un objet php en une chaine de caractères on parle de serialisation
  - Permet de convertir une chaine de caractères en objet PHP, on parle alors de désérialisation
- Deux étapes dans la sérialization:
  - La normalization permet de convertir un objet en tableau PHP
  - L'encodage permet d'encoder le tableau dans le format souhaité (JSON par exemple)

Rappel




```php
interface SerializerInterface
{
    public function serialize($data, string $format, array $context = []);
    public function deserialize($data, string $type, string $format, array $context = []);
}
```

Context - Introduction

- Les contextes servent à « configurer » le serializer.
- Ils sont utilisés par le serializer:
  - Exemple, format d'une date pour un objet `Datetime`
- Les groupes de sérialisation sont dans les contextes.

Intégration dans Api platform

```php
#[ApiResource(
    ...
    normalizationContext: ['groups' => 'post:list'],
    denormalizationContext: ['groups' => 'post:write'],
)]
class Post
{
    // ...
}
```

Intégration dans Api platform

```
#[ApiResource(
    itemOperations: [
        'GET' => [
            'normalization_context' => [
                'foo' => 'bar',
                'groups' => [
                    post:read',
                ],
            ],
        ],
        'PUT' => [
             'denormalization_context' => [
                'groups' => [
                    post:write',
                ],
            ],
    ]
)]
class Post
{
    // ...
}
```

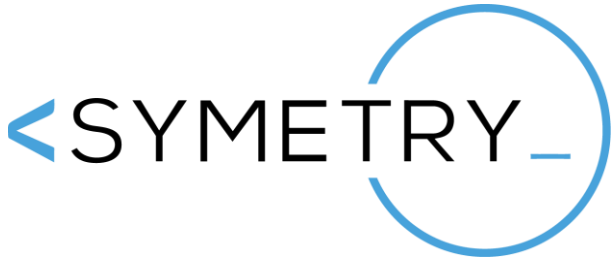
Exemple d’usage des groupes

```php
class Post
{
    #[ORM\Column(name: 'title', type: 'string')]
    #[Groups(['post:read', 'post:write', 'post:list'])]
    private string $title;
}
```

Exercice

Modifier l’api pour y intégrer les règles suivantes:

- N’afficher le nom des comptes bancaires en mode liste
- Désactiver la possibilité de modifier le nom d’un compte bancaire (tous les autres champs restent modifiables)
- Lors de la création, ne pas retourner de données



Validation

Valider les données

Basée sur le même fonctionnement que la validation existante dans Symfony

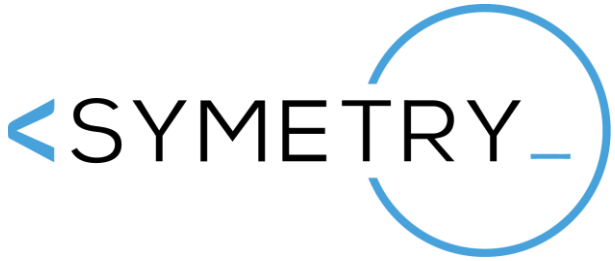
Annotations sur les propriétés, les classes

Validator custom

Exercice

Modifier l’api pour y intégrer les règles suivantes:

- Ajout d’une contrainte pour vérifier que l’iban d’un compte bancaire soit valide
- Uniquement à la création



Security

Vérifier la sécurité de son API

Champ « security » dans la configuration d'un endpoint

Couplé à la sécurité de Symfony (authentification / Voters)

Exemple

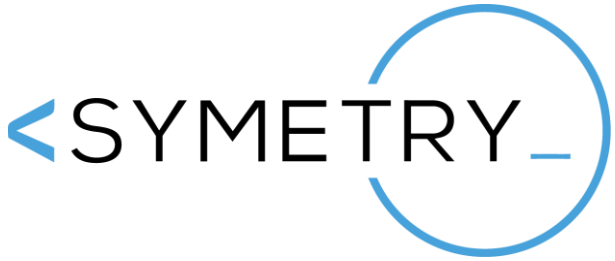
```
#[ApiResource(
    itemOperations: [
        'get' => [
            'security' => 'is_granted("READ", object)',
        ],
    ]
)]
```

```
class BankAccountVoter extends Voter
{
    protected function supports($attribute, $subject)
    {
        return $subject instanceof Post;
    }

    protected function voteOnAttribute($attribute, $subject, TokenInterface $token): bool
    {
        ...
    }
}
```

Exercice

Modifier l’api pour y intégrer les règles suivantes:

- Créer un voter pour vérifier les droits d’accès à un compte bancaire.
- Interdire la modification d’un compte bancaire en « readonly » en utilisant les voteurs



Filters

Exercice

Les filtres servent à modifier la requête exécutée en base de données.

ApiPlatform est fourni avec plusieurs filtres par défaut.

Les filtres peuvent servir

- Soit à récupérer des données selon un critère précis
- Soit à trier les données selon une colonne
- Rajouter des groupes de sérialisation à la volée

Exercice

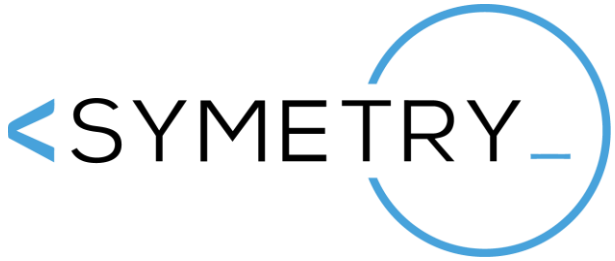
```
#[ApiFilter(OrderFilter::class, properties: ['date'])]
#[ApiFilter(SearchFilter::class, properties: [
    'label' => 'ipartial',
    'transactions.accountNumber.uuid' => 'exact'
])]
#[ApiFilter(RangeFilter::class, properties: ['amount'])]
#[ApiFilter(DateFilter::class, properties: ['date'])]
#[ApiFilter(BooleanFilter::class, properties: ['isQualified'])]
#[ApiFilter(GroupFilter::class, arguments: [
    'parameterName' => 'groups',
    'overrideDefaultGroups' => false,
    'whitelist' => ['bank_account:qualify'],
])]

// Custom filters
#[ApiFilter(AbsoluteNumericValueFilter::class, properties: ['amount'])]
```

Exercice

Modifier l'api pour y intégrer les règles suivantes:

- Créer un filtre pour n'afficher que les comptes bancaires en readonly
- Rajouter un champ created dans BankAccount (datetime)
- Autoriser la modification de ce champ à la création et à la modification

- Pouvoir filtrer sur les dates



Extending Api platform

Création d'une opération custom

```
#[ApiResource(
    collectionOperations: [],
    itemOperations: [
        'qualification' => [
            'method' => 'PUT',
            'path' => '/bank_operations/{id}/qualification',
        ],
    ]
)
```

```
#[ApiResource(
    collectionOperations: [],
    itemOperations: [
        'qualification' => [
            'method' => 'PUT',
            'controller' => QualificationController::class,
            'path' => '/bank_operations/{id}/qualification',
        ],
    ]
)
```

Exemple de controller

```
class OperationQualificationController
{
    public function __invoke(
         Operation $data,
         ValidatorInterface $validator,
         EntityManagerInterface $manager
     ): Operation {
        $validator->validate($data);

        $connection = $manager->getConnection();
        try {
            $manager->getConnection()->beginTransaction();
            $manager->flush();
            $manager->commit();
        } catch (\Exception $exception) {
            $connection->rollBack();
            throw $exception;
        }

        return $data;
    }
}
```

Exercice

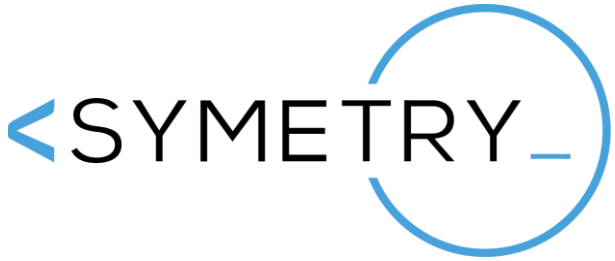
- Créer une entité bank_operation avec
  - Amount
  - date
  - Label

  (tous obligatoires)

  Ajouter un lien many to one / onetomany entre bank account et bank operation

- Créer une route custom /addOperation, qui va seulement ajouter une opération
  - Créer le controller
  - Injecter le validator avec validator interface (celui d'apip) et valider l'opération
  - Faire la configuration Api Platform

  - Rajouter la liste des opérations au get sur le compte bancaire



Tests

Exercice

Composer require test-pack

composer require --dev symfony/browser-kit symfony/http-client

bin/console make:test

Adapter le fichier généré et rajouter des tests sur tous les endpoints précédents

Final test

```
#[ApiResource(
    collectionOperations: [],
    itemOperations: [
        'qualification' => [
            'method' => 'PUT',
            'validate' => false,
            'write' => false,
            'controller' => OperationQualificationController::class,
            'path' => '/bank_operations/{uuid}/qualification',
            'security' => 'is_granted("READ", object.getBankAccount())',
            'normalization_context' => [
                'groups' => [
                    'bank_operation:read',
                    'bank_operation_transaction:read',
                ],
            ],
            'denormalization_context' => [
                'groups' => [
                    'bank_operation:write',
                    'bank_operation_transaction:write',
                ],
            ],
        ],
    ])]
```