Symfony 6 : API Platform

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Symfony

API Platform

- Framework **PHP** open-source
- Créé par Kévin Dunglas et distribué par **Symfony**
- Permettant de générer des services **REST** pour des entités **Doctrine**
- Documentation officielle : `https://api-platform.com/`
- Utilisant **Swagger UI** pour visualiser et interagir avec les **API REST**

Symfony

Swagger UI

- Générateur de documentation au format **JSON** pour les services **API REST**
- Permettant également de visualiser la documentation des ressources exposées
- Documentation officielle : `https://swagger.io/tools/swagger-ui/`

Symfony

Étapes à suivre avant de commencer le cours

1. Allez à `https://github.com/Achreftun/api-platform-symfony`
2. Clonez le dépôt dans votre espace de travail
3. Installez les dépendances : `composer install` ou `composer update`
4. Créez la base de données : `php bin/console d:d:c`
5. Appliquez les migrations : `php bin/console make:migration` ou `php bin/console d:m:m`
6. Envoyez les fixtures (les tuples) à la base de données : `php bin/console d:f:l`
7. Lancez le server : `symfony server:start`

Symfony

Installation avec Flex (déjà installé dans le projet cloné)

```
composer require api
```

Symfony

Contenu ajouté dans `.env`

```
###> nelmio/cors-bundle ###
CORS_ALLOW_ORIGIN='^https?://(localhost|127\.0\.0\.1)(:[0-9]+)?$'
###< nelmio/cors-bundle ###
```

Symfony

Contenu ajouté dans `.env`

```
###> nelmio/cors-bundle ###
CORS_ALLOW_ORIGIN='^https?://(localhost|127\.0\.0\.1)(:[0-9]+)?$'
###< nelmio/cors-bundle ###
```

Un fichier de configuration `config/packages/nelmio_cors.yaml` **généré avec le contenu suivant**

```
nelmio_cors:
    defaults:
        origin_regex: true
        allow_origin: ['%env(CORS_ALLOW_ORIGIN)%']
        allow_methods: ['GET', 'OPTIONS', 'POST', 'PUT', 'PATCH', 'DELETE']
        allow_headers: ['Content-Type', 'Authorization']
        expose_headers: ['Link']
        max_age: 3600
    paths:
        '^/': null
```

Symfony

Deux autres fichiers de configuration générés

- `config/routes/api_platform.yaml` : pour la configuration des routes
- `config/packages/api_platform.yaml` : pour le reste de configurations

Symfony

Contenu de `config/routes/api_platform.yaml`

```yaml
api_platform:
    resource: .
    type: api_platform
    prefix: /api
```

Symfony

Contenu de `config/routes/api_platform.yaml`

```
api_platform:
    resource: .
    type: api_platform
    prefix: /api
```

Commentez ou supprimez les lignes ajoutées précédemment dans `config/serices.yaml`

```
# get_set_method_normalizer:
#     class: Symfony\Component\Serializer\Normalizer\
  GetSetMethodNormalizer
#     tags: [serializer.normalizer]
```

Symfony

Remarque

Pour qu'une entité soit considérée comme une ressource **REST**, on utilise l'attribut `#[ApiResource()]`.

Symfony

Ajoutons `#[ApiResource()]` **à notre entité** `Personne`

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;


#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource()]
class Personne
{
    // ....
}
```

Symfony

Ajoutons `#[ApiResource()]` **à notre entité** `Personne`

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;


#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource()]
class Personne
{
    // ....
}
```

La même chose pour l'entité `Adresse`.

Symfony

Pour tester

Allez à `localhost:8000/api`

Symfony

Pour tester

Allez à `localhost:8000/api`

Remarques

- Par défaut, **API Platform** retourne les données au format **JSON-LD**
  - **JSON-LD** : **J**SON for **L**inking **D**ata
  - Documentation officielle : `https://json-ld.org/`
- Pour retourner des données au format **JSON**, **XML** ou autre, il est possible de configurer **API Platform** dans `config/packages/api_platform.yaml`

Symfony

Contenu initial de `config/packages/api_platform.yaml`

```
api_platform:
    mapping:
        paths: ['%kernel.project_dir%/src/Entity']
    patch_formats:
        json: ['application/merge-patch+json']
    swagger:
        versions: [3]
```

Symfony

Contenu initial de `config/packages/api_platform.yaml`

```
api_platform:
    mapping:
        paths: ['%kernel.project_dir%/src/Entity']
    patch_formats:
        json: ['application/merge-patch+json']
    swagger:
        versions: [3]
```

Spécifions maintenant le format attendu

```
api_platform:
    mapping:
        paths: ['%kernel.project_dir%/src/Entity']
    patch_formats:
        json: ['application/merge-patch+json']
    swagger:
        versions: [3]
    formats:
        json: ['application/json']
```

Symfony

Extrait du résultat retourné en allant à `http://127.0.0.1:8000/api/personnes`


```
[
    {
        "id": 1,
        "nom": "Leconte",
        "prenom": "Diane",
        "adresses": [
            "/api/adresses/1",
            "/api/adresses/2",
            "/api/adresses/3",
            "/api/adresses/4"
        ]
    },
    {
        "id": 2,
        "nom": "Charrier",
        "prenom": "Richard",
        "adresses": [
            "/api/adresses/5",
            "/api/adresses/6",
            "/api/adresses/7",
            "/api/adresses/8",
            "/api/adresses/9"
        ]
    }
]
```

Symfony

Remarque

- La documentation **Swagger** n'est plus affichée au format **HTML** mais au format **JSON**.
- Pour afficher la documentation **Swagger** au format **HTML**, il faut l'autoriser dans `config/packages/api_platform.yaml`.

Symfony

Remarque

- La documentation **Swagger** n'est plus affichée au format **HTML** mais au format **JSON**.
- Pour afficher la documentation **Swagger** au format **HTML**, il faut l'autoriser dans `config/packages/api_platform.yaml`.

Ajoutons le deuxième format dans `config/packages/api_platform.yaml` (Attention à l'ordre)

```
api_platform:
    mapping:
        paths: ['%kernel.project_dir%/src/Entity']
    patch_formats:
        json: ['application/merge-patch+json']
    swagger:
        versions: [3]
    formats:
        json: ['application/json']
        html: ['text/html']
```

Symfony

Deux types d'opération

- Collection operations :
  - **GET** : `/api/personnes`
  - **POST** : `/api/personnes`
- Item operations :
  - **GET** : `/api/personnes/{id}`
  - **DELETE** : `/api/personnes/{id}`
  - **PUT** : `/api/personnes/{id}`
  - **PATCH** : `/api/personnes/{id}`

Symfony

Et si nous voulions exposer seulement quelques méthodes

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: ["get"],
    itemOperations: ["get", "put", "delete"]
)]
class Personne
{
    // ....
}
```

Symfony

Et si nous voulions exposer seulement quelques méthodes

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: ["get"],
    itemOperations: ["get", "put", "delete"]
)]
class Personne
{
    // ....
}
```

Les méthodes disponibles

- **GET** : `/api/personnes`
- **GET** : `/api/personnes/{id}`
- **DELETE** : `/api/personnes/{id}`
- **PUT** : `/api/personnes/{id}`

Symfony

Il est possible de modifier le `path` d'une méthode et le code de la réponse HTTP

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: [
        "get" => [
            "path" => "/person",
            "status" => 301
        ]
    ],
    itemOperations: ["get", "put", "delete"]
)]
class Personne
{
    // ....
}
```

Symfony

Il est possible de modifier le `path` d'une méthode et le code de la réponse HTTP

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: [
        "get" => [
            "path" => "/person",
            "status" => 301
        ]
    ],
    itemOperations: ["get", "put", "delete"]
)]
class Personne
{
    // ....
}
```

Pour tester `localhost:8000/api/person`

Symfony

Pour modifier de préfixer le chemin de toutes les routes d'une ressource

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    routePrefix: "/library",
    collectionOperations: [
        "get" => [
            "path" => "/person",
            "status" => 301
        ]
    ],
    itemOperations: ["get", "put", "delete"]
)]
class Personne
{
    // ....
}
```

Symfony

Pour modifier de préfixer le chemin de toutes les routes d'une ressource

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    routePrefix: "/library",
    collectionOperations: [
        "get" => [
            "path" => "/person",
            "status" => 301
        ]
    ],
    itemOperations: ["get", "put", "delete"]
)]
class Personne
{
    // ....
}
```

Pour tester `localhost:8000/api/library/person`

Symfony

Pagination

- Nombre d'objets **JSON** à retourner pour une requête **GET** de type `Collection`
- Possibilité de le fixer
  - pour toutes les ressources dans `config/packages/api_platform.yaml`
  - pour une ressource donnée avec l'attribut `#[ApiResource()]`

Symfony

Pour fixer le nombre d'éléments par page pour toutes les ressources

```
api_platform:
    mapping:
        paths: ['%kernel.project_dir%/src/Entity']
    patch_formats:
        json: ['application/merge-patch+json']
    swagger:
        versions: [3]
    formats:
        json: ['application/json']
        html: ['text/html']
    defaults:
        pagination_items_per_page: 5
```

Symfony

Pour fixer le nombre d'éléments par page pour toutes les ressources

```
api_platform:
    mapping:
        paths: ['%kernel.project_dir%/src/Entity']
    patch_formats:
        json: ['application/merge-patch+json']
    swagger:
        versions: [3]
    formats:
        json: ['application/json']
        html: ['text/html']
    defaults:
        pagination_items_per_page: 5
```

Pour tester `localhost:8000/api/personnes`

Symfony

Pour fixer le nombre d'éléments par page pour une ressource donnée (par défaut : 30)

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    attributes: [
        "pagination_items_per_page" => 3
    ],
    collectionOperations: ["get"],
    itemOperations: ["get", "put", "delete"]
)]
class Personne
{
    // ....
}
```

Symfony

Pour fixer le nombre d'éléments par page pour une ressource donnée (par défaut : 30)

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    attributes: [
        "pagination_items_per_page" => 3
    ],
    collectionOperations: ["get"],
    itemOperations: ["get", "put", "delete"]
)]
class Personne
{
    // ....
}
```


Pour tester

- `localhost:8000/api/adresses` ⇒ récupère les 5 premières adresses
- `localhost:8000/api/personnes` ⇒ récupère les 3 premières personnes
- `localhost:8000/api/personnes?page=2` ⇒ récupère les 3 personnes de la deuxième page

Pour fixer le nombre d'éléments par page pour une ressource donnée

```
api_platform:
    mapping:
        paths: ['%kernel.project_dir%/src/Entity']
    patch_formats:
        json: ['application/merge-patch+json']
    swagger:
        versions: [3]
    formats:
        json: ['application/json']
        html: ['text/html']
    defaults:
        pagination_items_per_page: 5
    collection:
        pagination:
            page_parameter_name: page_number
```

Pour fixer le nombre d'éléments par page pour une ressource donnée

```
api_platform:
    mapping:
        paths: ['%kernel.project_dir%/src/Entity']
    patch_formats:
        json: ['application/merge-patch+json']
    swagger:
        versions: [3]
    formats:
        json: ['application/json']
        html: ['text/html']
    defaults:
        pagination_items_per_page: 5
    collection:
        pagination:
            page_parameter_name: page_number
```

Pour tester

`localhost:8000/api/personnes?page_number=2` ⇒ récupère les 3 personnes de la deuxième page

Symfony

Question

Comment retourner les adresses d'une personne (pas les **URI**) ?

Symfony

Question

Comment retourner les adresses d'une personne (pas les **URI**) ?

Réponse

En spécifiant les groupes dans les propriétés `normalizationContext` et `denormalizationContext`.

Symfony

Nouveau contenu après avoir ajouté les groupes

```
class Personne
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $id;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $nom;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $prenom;

    #[ORM\ManyToMany(targetEntity: Adresse::class, inversedBy: 'personnes', cascade: ['persist'
       , 'remove'])]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $adresses;

    // ...

}
```

Symfony

Nouveau contenu de l'entité `Adresse`

```php
#[ORM\Entity(repositoryClass: AdresseRepository::class)]
#[ApiResource()]
class Adresse
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $id;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $rue;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $codePostal;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $ville;

    // ...

}
```

Ajoutons les attributs suivants à l'annotation `@ApiResource`

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: ["get", "post"],
    itemOperations: ["get", "put", "delete"],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
class Personne
{
    // ....
}
```

Ajoutons les attributs suivants à l'annotation `@ApiResource`

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: ["get", "post"],
    itemOperations: ["get", "put", "delete"],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
class Personne
{
    // ....
}
```


Explication

- Les attributs ayant le groupe `"personne:read"` seront accessibles en lecture (`GET`)
- Les attributs ayant le groupe `"personne:write"` seront accessibles en mode écriture (`POST`, `PUT` et `DELETE`)
- Les attributs ayant les deux groupes seront accessibles en lecture et écriture
- Les attributs n'ayant aucun groupe ne seront pas pris en compte

Symfony

Extrait du résultat retourné en allant à `http://127.0.0.1:8000/api/personnes/6`


```
{
    "id": 6,
    "nom": "Cordier",
    "prenom": "Nicole",
    "adresses": [
        {
            "id": 16,
            "rue": "chemin Auguste Maillard",
            "codePostal": "48848",
            "ville": "Klein"
        },
        {
            "id": 17,
            "rue": "chemin Schneider",
            "codePostal": "28800",
            "ville": "Pichon-les-Bains"
        }
    ]
}
```

Symfony

Objectif

- Ajouter un attribut `dateEnregistrement` qui correspond à la date d'insertion de la personne dans la base de données
- La valeur de ce champs ne doit pas être renseigné par l'utilisateur
- On utilise donc les `DataPersister` pour intercepter les données utilisateur et ajouter la date

Symfony

Nouveau contenu après avoir ajouté l'attribut `dateEnregistrement`

```php
class Personne
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $id;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $nom;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $prenom;

    #[ORM\ManyToMany(targetEntity: Adresse::class, inversedBy: 'personnes', cascade: ['persist'
       , 'remove'])]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $adresses;

    #[ORM\Column(type: 'datetime')]
    #[Groups(["personne:read"])]
    private $dateEnregistrement;

    // ...

}
```

Symfony

Rien à changer dans l'entité `Adresse`

```
#[ORM\Entity(repositoryClass: AdresseRepository::class)]
#[ApiResource()]
class Adresse
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $id;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $rue;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $codePostal;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    #[Groups(["personne:read", "adresse:read", "personne:write"])]
    private $ville;

    // ...

}
```

Symfony

Remarques

- En essayant d'ajouter une nouvelle `Personne`, un message d'erreur s'affiche.
- En effet, la valeur de `dateEnregistrement` n'a pas été renseignée et elle n'est pas `nullable` pour **MySQL**.

Symfony

Remarques

- En essayant d'ajouter une nouvelle `Personne`, un message d'erreur s'affiche.
- En effet, la valeur de `dateEnregistrement` n'a pas été renseignée et elle n'est pas `nullable` pour **MySQL**.

Solution

Utiliser `DataPersister`

Symfony

Data Persister

Un objet qui sera instancié avant toute opération d'écriture dans la base de donnés.

Symfony

Exemple

- On crée une classe `PersonneDataPersister` qui implémente l'interface `DataPersisterInterface`
- `DataPersisterInterface` a trois méthodes :
  - `supports` : vérifie si le persister supporte l'entité
  - `persist` : s'exécute à chaque opération de type `POST` ou `PUT`
  - `remove` : s'exécute seulement pour l'opération `DELETE`

Contenu de la classe `PersonneDataPersister` **définie dans** `App/DataPersister`

```php
namespace App\DataPersister;

use App\Entity\Personne;
use Doctrine\ORM\EntityManagerInterface;
use ApiPlatform\Core\DataPersister\ContextAwareDataPersisterInterface;

class PersonneDataPersister implements ContextAwareDataPersisterInterface
{
    public function __construct(private EntityManagerInterface $em)
    {
    }

    public function supports($data, array $context = []): bool
    {
        return $data instanceof Personne;
    }

    public function persist($data, array $context = [])
    {
        $dateTimeZone = new \DateTimeZone('Europe/Paris');
        $data->setDateEnregistrement(new \DateTime('now', $dateTimeZone));
        $this->em->persist($data);
        $this->em->flush();
    }

    public function remove($data, array $context = [])
    {
        $this->em->remove($data);
        $this->em->flush();
    }
}
```

Symfony

Remarque

En faisant la modification d'une personne existante, `dateEnregistrement` sera aussi mise à jour

Symfony

Pour affecter une valeur à `dateEnregistrement` seulement en cas d'ajout (`POST`), on ajoute le test suivant

```
public function persist($data, array $context = [])
{
    if (($context['collection_operation_name'] ?? null)  === 'post') {
        $dateTimeZone = new \DateTimeZone('Europe/Paris');
        $data->setDateEnregistrement(new \DateTime('now', $dateTimeZone
           ));
    }
    $this->em->persist($data);
    $this->em->flush();
}
```

Symfony

Pour affecter une valeur à `dateEnregistrement` seulement en cas d'ajout (`POST`), on ajoute le test suivant

```
public function persist($data, array $context = [])
{
    if (($context['collection_operation_name'] ?? null)  === 'post') {
        $dateTimeZone = new \DateTimeZone('Europe/Paris');
        $data->setDateEnregistrement(new \DateTime('now', $dateTimeZone
           ));
    }
    $this->em->persist($data);
    $this->em->flush();
}
```


Remarque

`collection_operation_name` pour Collection operations et `item_operation_name` pour Item operations

Symfony

Data Provider

Un objet qui sera instancié avant toute opération de lecture de donnés.

- **CollectionProvider** : pour les méthodes qui retournent une collection de données
- **ItemProvider** : pour les méthodes qui retournent un seul élément

Symfony

Exemple avec **CollectionProvider**

On crée une classe `PersonneCollectionDataProvider` qui implémente l'interface `ContextAwareCollectionDataProviderInterface` et `RestrictedDataProviderInterface` qui ont les deux méthodes suivantes

- `supports` : vérifie si le provider supporte l'entité
- `getCollection` : s'exécute à chaque opération de type `GET` et qui retourne une collection de données

Contenu de la classe `PersonneCollectionDataProvider` **définie dans** `App/DataProvider`

```php
namespace App\DataProvider;

use App\Entity\Personne;
use Doctrine\ORM\EntityManagerInterface;
use ApiPlatform\Core\DataProvider\RestrictedDataProviderInterface;
use ApiPlatform\Core\DataProvider\ContextAwareCollectionDataProviderInterface;


final class PersonneCollectionDataProvider implements
  ContextAwareCollectionDataProviderInterface, RestrictedDataProviderInterface
{
    public function __construct(private EntityManagerInterface $em)
    {
    }

    public function supports(string $resourceClass, string $operationName = null, array
      $context = []): bool
    {
        return Personne::class === $resourceClass;
    }

    public function getCollection(string $resourceClass, string $operationName = null, array
      $context = [])
    {
        $personnes = $this->em->getRepository(Personne::class)->findAll();
        return array_map(fn ($elt) => $elt->setNom(strtoupper($elt->getNom())), $personnes);
    }
}
```

Symfony

Remarque

En demandant la liste des personnes, les noms seront retournés en majuscule.

Symfony

Exemple avec **ItemProvider**

On crée une classe `PersonneCollectionDataProvider` qui implémente l'interface `ItemDataProviderInterface` et `RestrictedDataProviderInterface` qui ont les deux méthodes suivantes

- `supports` : vérifie si le provider supporte l'entité
- `getItem` : s'exécute à chaque opération de type `GET` et qui retourne un seul

Contenu de la classe `PersonneItemDataProvider` **définie dans** `App/DataProvider`

```php
namespace App\DataProvider;

use App\Entity\Personne;
use Doctrine\ORM\EntityManagerInterface;
use ApiPlatform\Core\DataProvider\ItemDataProviderInterface;
use ApiPlatform\Core\DataProvider\RestrictedDataProviderInterface;

final class PersonneItemDataProvider implements ItemDataProviderInterface,
  RestrictedDataProviderInterface
{
    public function __construct(private EntityManagerInterface $em)
    {
    }

    public function supports(string $resourceClass, string $operationName = null, array
      $context = []): bool
    {
        return Personne::class === $resourceClass;
    }

    public function getItem(string $resourceClass, $id, string $operationName = null, array
      $context = []): ?Personne
    {
        $personne = $this->em->getRepository(Personne::class)->find($id);
        if ($personne) {
            $personne->setNom(strtoupper($personne->getNom()));
        }
        return $personne;
    }
}
```

Symfony

Remarque

En demandant une personne selon son identifiant, son nom sera retourné en majuscule.

Symfony

Ajoutons l'annotation `@ApiSubresource` à l'attribut `adresses` pour récupérer la liste d'adresses d'une personne dont l'`id` est passé dans la route

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: ["get", "post"],
    itemOperations: ["get", "put", "delete"],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
class Personne
{

    // ...

    #[ORM\ManyToMany(targetEntity: Adresse::class, inversedBy: 'personnes', cascade: ['persist'
       , 'remove'])]
    #[Groups(["personne:read", "personne:write"])]
    #[ApiSubresource()]
    private $adresses;

    // ...

}
```

Symfony

Ajoutons l'annotation `@ApiSubresource` à l'attribut `adresses` pour récupérer la liste d'adresses d'une personne dont l'`id` est passé dans la route

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: ["get", "post"],
    itemOperations: ["get", "put", "delete"],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
class Personne
{

    // ...

    #[ORM\ManyToMany(targetEntity: Adresse::class, inversedBy: 'personnes', cascade: ['persist'
       , 'remove'])]
    #[Groups(["personne:read", "personne:write"])]
    #[ApiSubresource()]
    private $adresses;

    // ...

}
```

Ainsi on a une nouvelle route pour le verbe `GET` : `/api/personnes/{id}/adresses`

Symfony

#[ApiFilter()]

- permet de filtrer le résultat de recherche
- peut concerner une entité ou un attribut
- peut utiliser plusieurs classes de filtre
  - `SearchFilter` : principalement pour les champs de type chaîne de caractères
  - `DateFilter` : pour les champs de type date
  - `NumericFilter` et `RangeFilter` : pour les champs de type numérique
  - ...

Symfony

#[ApiFilter()]

- permet de filtrer le résultat de recherche
- peut concerner une entité ou un attribut
- peut utiliser plusieurs classes de filtre
  - `SearchFilter` : principalement pour les champs de type chaîne de caractères
  - `DateFilter` : pour les champs de type date
  - `NumericFilter` et `RangeFilter` : pour les champs de type numérique
  - ...

Liste complète

https://api-platform.com/docs/core/filters/

Symfony

Ajoutons l'annotation `@ApiFilter` à l'entité `Adresse` pour effectuer une recherche selon la ville (par exemple)

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\AdresseRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

#[ORM\Entity(repositoryClass: AdresseRepository::class)]
#[ApiResource()]
#[ApiFilter(
    SearchFilter::class,
    properties: ["ville" => "partial"]
)]
class Adresse
{
    // ...
}
```

Symfony

Ajoutons l'annotation `@ApiFilter` à l'entité `Adresse` pour effectuer une recherche selon la ville (par exemple)

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\AdresseRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

#[ORM\Entity(repositoryClass: AdresseRepository::class)]
#[ApiResource()]
#[ApiFilter(
    SearchFilter::class,
    properties: ["ville" => "partial"]
)]
class Adresse
{
    // ...
}
```

Exemple de route pour le test : `http://localhost:8000/api/adresses?ville=mars`

Symfony

Valeurs possibles de `properties` de `SearchFilter`

- `partial` ≡ `ville like %texte%`
- `exact` ≡ `ville = texte`
- `end` ≡ `ville like %texte`
- `start` ≡ `ville like texte%`

Symfony

Valeurs possibles de `properties` de `SearchFilter`

- `partial` ≡ `ville like %texte%`
- `exact` ≡ `ville = texte`
- `end` ≡ `ville like %texte`
- `start` ≡ `ville like texte%`

Il est aussi possible d'effectuer la recherche à partir de la ressource `Personne` :
`http://localhost:8000/api/personnes/9/adresses?ville=mars`

Symfony

On peut aussi définir des contraintes de recherche sur les nombres en utilisant `RangeFilter`

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\AdresseRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\RangeFilter;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

#[ORM\Entity(repositoryClass: AdresseRepository::class)]
#[ApiResource()]
#[ApiFilter(SearchFilter::class, properties: ["ville" => "partial"])]
#[ApiFilter(RangeFilter::class, properties: ["codePostal"])]
class Adresse
{
    // ...
}
```

Symfony

On peut aussi définir des contraintes de recherche sur les nombres en utilisant `RangeFilter`

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\AdresseRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\RangeFilter;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

#[ORM\Entity(repositoryClass: AdresseRepository::class)]
#[ApiResource()]
#[ApiFilter(SearchFilter::class, properties: ["ville" => "partial"])]
#[ApiFilter(RangeFilter::class, properties: ["codePostal"])]
class Adresse
{
    // ...
}
```

Exemple de route pour le test :
`http://localhost:8000/api/adresses?ville=mars&codePostal[gt]=13006`

Symfony

Valeurs possibles de `properties` de `RangeFilter`

- `gt` ≡ supérieur à
- `lt` ≡ inférieur à
- `gte` ≡ supérieur ou égal à
- `lte` ≡ inférieur ou égal à
- `between` ≡ entre (Exemple : codePostal[between]=13000..13016)

On peut aussi définir des contraintes sur les entités imbriquées (inverses)

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Annotation\ApiSubresource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: ["get", "post"],
    itemOperations: ["get", "put", "delete"],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
#[ApiFilter(SearchFilter::class, properties: ["adresses" => "exact"])]
class Personne
{
    // ...
}
```

On peut aussi définir des contraintes sur les entités imbriquées (inverses)

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Annotation\ApiSubresource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: ["get", "post"],
    itemOperations: ["get", "put", "delete"],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
#[ApiFilter(SearchFilter::class, properties: ["adresses" => "exact"])]
class Personne
{
    // ...
}
```

Pour récupérer les personnes qui ont l'adresse avec un `id = 49` :
`http://localhost:8000/api/personnes?adresses=49`

On peut aussi chercher selon un attribut dans l'entité inverse autre que l'identifiant

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Annotation\ApiSubresource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;


#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: ["get", "post"],
    itemOperations: ["get", "put", "delete"],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
#[ApiFilter(SearchFilter::class, properties: ["adresses.ville" => "partial"])]
class Personne
{
    // ...
}
```

On peut aussi chercher selon un attribut dans l'entité inverse autre que l'identifiant

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Annotation\ApiSubresource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;


#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: ["get", "post"],
    itemOperations: ["get", "put", "delete"],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
#[ApiFilter(SearchFilter::class, properties: ["adresses.ville" => "partial"])]
class Personne
{
    // ...
}
```

Pour récupérer les personnes qui ont une adresse dont le nom de la ville contient `mars` :
`http://localhost:8000/api/personnes?adresses.ville=mars`

Symfony

Deux règles d'or de **Microsoft**

- never trust user input
- and always check data as it moves from an untrusted to a trusted domain

Symfony

Principe

- La validation des objets (entité ou autre) se réalise avec le composant `Validator` de **Symfony**
- Pour décider si un objet est valide, il faut définir des règles et et les rattacher aux objets
- Par exemple
  - un mot de passe doit contenir au moins 8 caractères
  - une note est comprise entre 0 et 20
  - ...
- La validation de données permet de garder la base de données dans un état cohérent.

Symfony

Comment ?

- En utilisant les annotations/**attributs**
- On peut aussi les externaliser dans un fichier
  - **XML**
  - **YML**
  - **PHP**

Symfony

Pour Doctrine

On a utilisé le namespace `use Doctrine\ORM\Mapping as ORM;`

Symfony

Pour Doctrine

On a utilisé le namespace `use Doctrine\ORM\Mapping as ORM;`

Pour le validator

```
use Symfony\Component\Validator\Constraints as Assert;
```

Symfony

Syntaxe

- La forme réduite : `@Assert\Contrainte (valeur de l'option par défaut)`
- La forme étendue : `@Assert\Contrainte (option1="valeur1", ... optionN="valeurN")`

Symfony

Syntaxe

- La forme réduite : `@Assert\Contrainte (valeur de l'option par défaut)`
- La forme étendue : `@Assert\Contrainte (option1="valeur1", ... optionN="valeurN")`

Exemple

- La forme réduite : `@Assert\Choice("homme", "femme")`
- La forme étendue : `@Assert\Length(min=10,minMessage= "Votre mot de passe {{ value }} ne contient pas {{ limit }} caractères."))`

Symfony

Remarques

- Les contraintes varient selon le type du champ
- Chaque contrainte a plusieurs options dont une par défaut
- Un champ peut être annotés par plusieurs contraintes

Symfony

Les contraintes générales

- `Blank` : vérifie si la valeur d'un champ est une chaîne vide ou null (`NotBlank` est l'inverse)
- `IsTrue` : vérifie si la valeur d'un champ est true, 1 ou '1' (pareillement pour `IsFalse`)
- `Type(nomType)` : vérifie si la valeur d'un champ est de type `nomType`
- `NotNull` : vérifie si la valeur d'un champ est différente de `null` (pareillement pour `IsNull`)

Symfony

Les contraintes sur les chaînes de caractères

- `Length` : vérifie si la valeur d'un champ vérifie certaines conditions. Elle accepte plusieurs options :
  - `min` et `minMessage` (le message d'erreur à afficher si la contrainte min n'est pas respectée)
  - `max` et `maxMessage`
- `Url` : vérifie si la valeur est une adresse **URL** valide
- `Ip` : vérifie si la valeur d'un champ est une adresse **IP**
- `Json` : vérifie si la valeur d'un champ est un objet au format **JSON**
- ...

Symfony

Les contraintes sur les nombres

- `Range` : comme `Length` mais elle vérifie la valeur et pas la longueur. Elle peut aussi prendre les mêmes paramètres que `Length`
- `DivisibleBy`
- `GreaterThan`
- `EqualTo`
- `Positive`
- `PositiveOrZero`
- ...

Symfony

Les contraintes sur les dates

- `File` : vérifie si la valeur correspond au chemin vers un fichier existant.
- `Image` : comme `File`, elle permet de vérifier la largeur max et min, la hauteur max et min d'une image, la disposition...

Symfony

Les contraintes sur les fichiers

- `Date` : vérifie si la valeur est un objet de type `Datetime`, ou une chaîne de caractères du type `YYYY-MM-DD`
- `Time` : vérifie si la valeur est un objet de type `Datetime`, ou une chaîne de caractères du type `HH:MM:SS`
- `DateTime` : vérifie si la valeur est un objet de type `Datetime`, ou une chaîne de caractères du type `YYYY-MM-DD HH:MM:SS`

Symfony

Autres contraintes

- `Currency` : vérifie si la valeur respecte le codage `3-letter ISO 4217` (EUR, USD...)
- `Isbn`
- `Country` : vérifie si la valeur respecte le codage `ISO 3166-1 alpha-2` (FR, US, TN, ES...)
- ...

Symfony

La liste complète

`https://symfony.com/doc/current/validation.html`

Exemple : ajoutons les contraintes suivantes sur le nom dans l'entité `Personne`

```
#[ORM\Column(type: 'string', length: 30, nullable: true)]
#[Groups(["personne:read", "personne:write"])]
#[Assert\Length(
    min : 2,
    max : 20,
    minMessage : "Le nom doit contenir au moins {{ limit }} caractères",
    maxMessage : "Le nom doit contenir au plus {{ limit }} caractères",
)]
private $nom;
```

Exemple : ajoutons les contraintes suivantes sur le nom dans l'entité `Personne`

```
#[ORM\Column(type: 'string', length: 30, nullable: true)]
#[Groups(["personne:read", "personne:write"])]
#[Assert\Length(
    min : 2,
    max : 20,
    minMessage : "Le nom doit contenir au moins {{ limit }} caractères",
    maxMessage : "Le nom doit contenir au plus {{ limit }} caractères",
)]
private $nom;
```

Et la contrainte suivante (exprimée avec une expression régulière) sur le prénom

```
#[ORM\Column(type: 'string', length: 30, nullable: true)]
#[Groups(["personne:read", "personne:write"])]
#[Assert\Regex(
    pattern: '/\d/',
    match: false,
    message: 'Le prénom ne peut contenir de chiffres',
)]
private $prenom;
```

Exemple : ajoutons les contraintes suivantes sur le nom dans l'entité `Personne`

```
#[ORM\Column(type: 'string', length: 30, nullable: true)]
#[Groups(["personne:read", "personne:write"])]
#[Assert\Length(
    min : 2,
    max : 20,
    minMessage : "Le nom doit contenir au moins {{ limit }} caractères",
    maxMessage : "Le nom doit contenir au plus {{ limit }} caractères",
)]
private $nom;
```

Et la contrainte suivante (exprimée avec une expression régulière) sur le prénom

```
#[ORM\Column(type: 'string', length: 30, nullable: true)]
#[Groups(["personne:read", "personne:write"])]
#[Assert\Regex(
    pattern: '/\d/',
    match: false,
    message: 'Le prénom ne peut contenir de chiffres',
)]
private $prenom;
```

Le `use` nécessaire

```
use Symfony\Component\Validator\Constraints as Assert;
```

Symfony

Vérifier en insérant les données suivantes via Postman

```
{
    "nom": "X",
    "prenom": "Malcolm"
}
```


La réponse suivante

```
{
    "type": "https://tools.ietf.org/html/rfc2616#section-10",
    "title": "An error occurred",
    "detail": "nom: Le nom doit contenir au moins 2 caractères",
    "violations": [
        {
            "propertyPath": "nom",
            "message": "Le nom doit contenir au moins 2 caractères",
            "code": "9ff3fdc4-b214-49db-8718-39c315e33d45"
        }
    ]
}
```

Symfony

Remarque

Il est possible de définir un validateur personnalisé.

Symfony

Exercice 2

Créez une application **Angular** qui permet à un utilisateur, via des interfaces graphiques) la gestion de personnes (ajout, modification, suppression, consultation et recherche) en consommant les **API** définies avec **Symfony**.

Symfony

JWT : **J**SON **W**eb **T**oken

- Librairie d'échange sécurisé d'informations
- Utilisant des algorithmes de cryptage comme **HMAC SHA256** ou **RSA**
- Utilisant les jetons (tokens)
- Un jeton est composé de trois parties séparées par un point :
  - entête (**header**) : objet **JSON** décrivant le jeton encodé en base 64
  - charge utile (**payload**) : objet **JSON** contenant les informations du jeton encodé en base 64
  - Une signature numérique = concaténation de deux éléments précédents séparés par un point + une clé secrète (le tout crypté par l'algorithme spécifié dans l'entête)
- Documentation officielle : `https://jwt.io/introduction/`

Symfony

Entête : exemple

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Symfony

Entête : exemple

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Charge utile : exemple

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Symfony

Exemple de construction de signature en utilisant l'algorithme précisé dans l'entête

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Symfony

Exemple de construction de signature en utilisant l'algorithme précisé dans l'entête

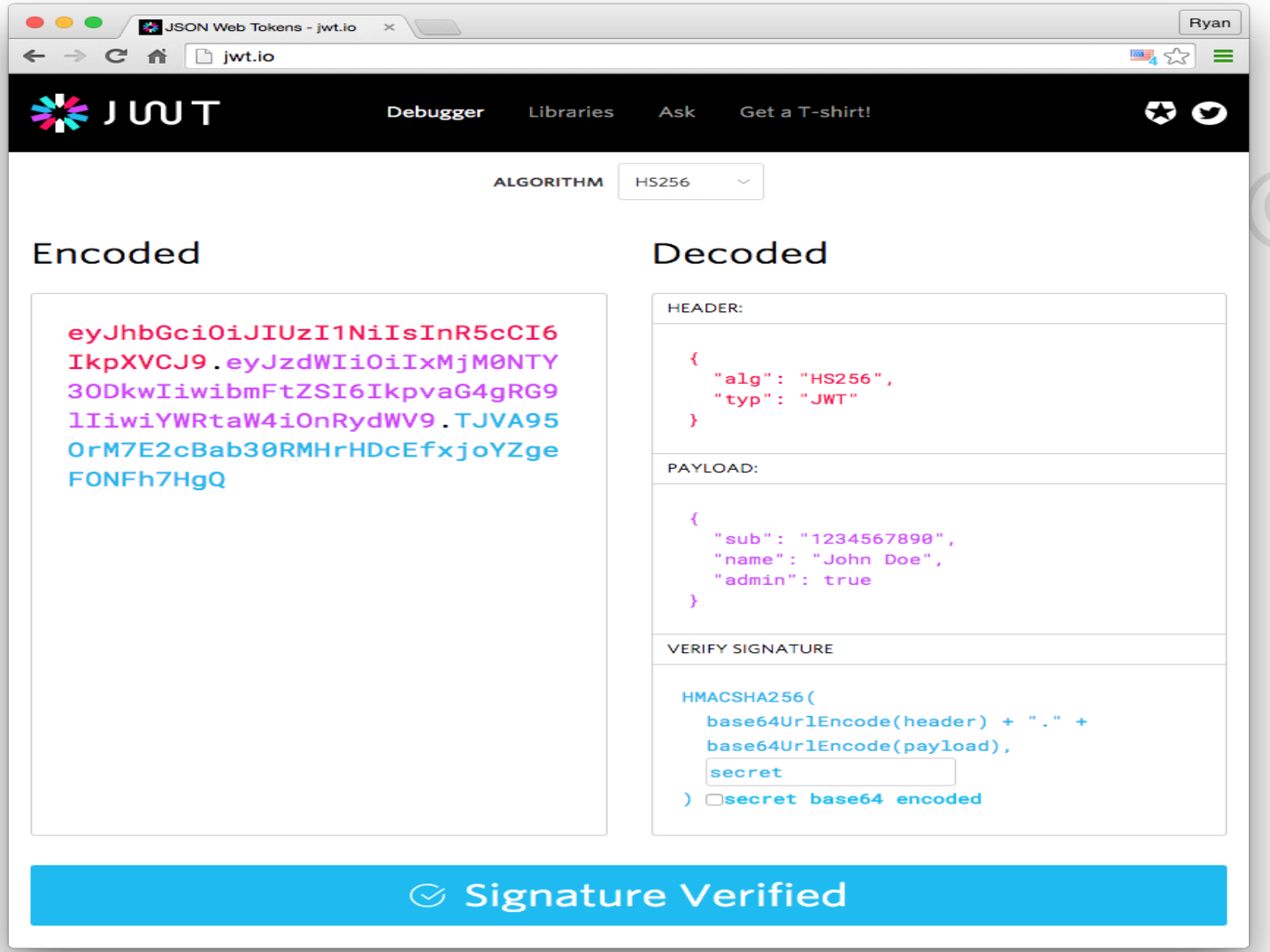
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```


Résultat

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4

Symfony

Exemple complet (pour tester `https://jwt.io/#debugger-io`)



Symfony

Pour décoder les deux premières parties d'un jeton

`http://calebb.net/`

Symfony

Étapes

- Préparation de la partie utilisateur (qui va se connecter)
- Préparation de la partie **JWT** (clé publique, clé privée...)
- Gestion de rôles

Symfony

Intégrons le bundle de sécurité dans notre projet

```
composer require symfony/security-bundle
```

Symfony

Intégrons le bundle de sécurité dans notre projet

```
composer require symfony/security-bundle
```

Ou

```
composer require security
```

Symfony

Pour créer la classe `User`

- exécutez la commande `php bin/console make:user`
- répondez à `The name of the security user class` par `User`
- répondez à `Do you want to store user data in the database (via Doctrine)?` par `yes`
- répondez à `Enter a property name that will be the unique "display" name for the user` par `email`
- répondez à `Does this app need to hash/check user passwords?` par `yes`

Symfony

Pour créer la classe `User`

- exécutez la commande `php bin/console make:user`
- répondez à `The name of the security user class` par `User`
- répondez à `Do you want to store user data in the database (via Doctrine)?` par `yes`
- répondez à `Enter a property name that will be the unique "display" name for the user` par `email`
- répondez à `Does this app need to hash/check user passwords?` par `yes`

Le résultat est

```
created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml
```

Nouveau contenu de `security.yaml`

```
security:
    encoders:
        App\Entity\User:
            algorithm: auto

    # https://symfony.com/doc/current/security.html#where-do-users-come
      -from-user-providers
    providers:
        # used to reload user from session & other features (e.g.
          switch_user)
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: lazy
            provider: app_user_provider

    access_control:
```

Symfony

Pour créer la table `User`, exécutez la commande

```
php bin/console make:migration
```

Symfony

Pour créer la table `User`, exécutez la commande

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Symfony

Pour créer la table `User`, exécutez la commande

```
php bin/console make:migration
```

Ensuite

```
php bin/console doctrine:migrations:migrate
```

Ou

```
php bin/console d:m:m
```

Symfony

Pour remplir la table `User` avec des données aléatoires, installez le bundle de fixture

```
composer require --dev doctrine/doctrine-fixtures-bundle
```

Symfony

Pour remplir la table `User` avec des données aléatoires, installez le bundle de fixture

```
composer require --dev doctrine/doctrine-fixtures-bundle
```

Ou

```
composer require --dev orm-fixtures
```

Symfony

Pour remplir la table `User` avec des données aléatoires, installez le bundle de fixture

```
composer require --dev doctrine/doctrine-fixtures-bundle
```

Ou

```
composer require --dev orm-fixtures
```


Pour générer un fichier de données aléatoires

```
php bin/console make:fixtures

The class name of the fixtures to create
UserFixtures
```

Symfony

Contenu généré pour `UserFixtures`

```php
namespace App\DataFixtures;

use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class UserFixtures extends Fixture
{

    public function load(ObjectManager $manager)
    {
        // $product = new Product();
        // $manager->persist($product);

        $manager->flush();
    }
}
```

Symfony

Nouveau contenu de `UserFixtures`

```
class UserFixtures extends Fixture
{
    public function __construct(private UserPasswordHasherInterface $passwordEncoder)
    {
    }
    public function load(ObjectManager $manager)
    {
        $user = new User();
        $user->setEmail('john@wick.us');
        $user->setRoles(['ROLE_ADMIN']);
        $user->setPassword($this->passwordEncoder->hashPassword(
            $user,
            'wick'
        ));
        $manager->persist($user);
        $user2 = new User();
        $user2->setEmail('jack@dalton.us');
        $user2->setPassword($this->passwordEncoder->hashPassword(
            $user2,
            'dalton'
        ));
        $manager->persist($user2);
        $manager->flush();
    }
}
```

Symfony

Modifions également `AppFixtures`

```
class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager): void
    {
        $faker = Factory::create('fr_FR');

        for ($p = 0; $p < 10; $p++) {
            $personne = new Personne;
            $personne->setNom($faker->lastName);
            $personne->setPrenom($faker->firstName);
            $dateTimeZone = new \DateTimeZone('Europe/Paris');
            $personne->setDateEnregistrement(new \DateTime('now', $dateTimeZone));
            for ($c = 0; $c < mt_rand(1, 5); $c++) {
                $adresse = new Adresse;
                $adresse->setRue($faker->streetName);
                $adresse->setVille($faker->city);
                $adresse->setCodePostal($faker->postcode);
                $personne->addAdress($adresse);
            }
            $manager->persist($personne);
        }
        $manager->flush();
    }
}
```

Symfony

Pour insérer l'utilisateur dans la base de données, exécutez

```
php bin/console doctrine:fixtures:load
```

Symfony

Pour insérer l'utilisateur dans la base de données, exécutez

```
php bin/console doctrine:fixtures:load
```

Ou

```
php bin/console d:f:l
```

Symfony

Intégrons le bundle JWT dans notre projet

```
composer require lexik/jwt-authentication-bundle
```

Symfony

Intégrons le bundle JWT dans notre projet

```
composer require lexik/jwt-authentication-bundle
```

Ou l'alias

```
composer require jwt
```

Symfony

Pour créer les clés

- exécutez `mkdir config/jwt` (pour créer un répertoire `jwt` dans `config`)
- exécutez `openssl genrsa -out config/jwt/private.pem -aes256 4096` pour générer une clé privée (n'oubliez pas le passphrase)
- exécutez `openssl rsa -pubout -in config/jwt/private.pem -out config/jwt/public.pem` pour générer une clé public

Symfony

Pour créer les clés

- exécutez `mkdir config/jwt` (pour créer un répertoire `jwt` dans `config`)
- exécutez `openssl genrsa -out config/jwt/private.pem -aes256 4096` pour générer une clé privée (n'oubliez pas le passphrase)
- exécutez `openssl rsa -pubout -in config/jwt/private.pem -out config/jwt/public.pem` pour générer une clé public

Vérifiez le contenu suivant dans `.env` (remplacez passphrase par sa valeur)

```
###> lexik/jwt-authentication-bundle ###
JWT_SECRET_KEY=%kernel.project_dir%/config/jwt/private.pem
JWT_PUBLIC_KEY=%kernel.project_dir%/config/jwt/public.pem
JWT_PASSPHRASE=symfony
###< lexik/jwt-authentication-bundle ###
```

Modifiez `security.yaml`

```
security:
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    enable_authenticator_manager: true
    providers:
        # used to reload user from session & other features (e.g. switch_user)
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
    firewalls:
        dev:
            pattern: ^/_(profiler|wdt)
            security: false
        api:
            pattern: ^/api/
            stateless: true
            provider: app_user_provider
            jwt: ~
        main:
            json_login:
                check_path: /authentication_token
                username_path: email
                password_path: password
                success_handler: lexik_jwt_authentication.handler.authentication_success
                failure_handler: lexik_jwt_authentication.handler.authentication_failure

    access_control:
        - { path: ^/api/docs, roles: PUBLIC_ACCESS } # Pour autoriser l'accès à Swagger UI
        - { path: ^/authentication_token, roles: PUBLIC_ACCESS }
        - { path: ^/, roles: IS_AUTHENTICATED_FULLY }
```

Symfony

Définissez la route `/authentication_token` **dans** `routes.yaml`

```
authentication_token:
    path: /authentication_token
    methods: ['POST']
```

Symfony

Dans `packages/lexik_jwt_authentication.yaml`, associons l'email utilisé dans l'entité à `username` recherché par JWT en ajoutant les deux dernières lignes

```
lexik_jwt_authentication:
    secret_key: '%env(resolve:JWT_SECRET_KEY)%'
    public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
    pass_phrase: '%env(JWT_PASSPHRASE)%'
    user_identity_field: 'email'
    user_id_claim: 'username'
```

Symfony

Dans `packages/lexik_jwt_authentication.yaml`**, associons l'email utilisé dans l'entité à** `username` **recherché par JWT en ajoutant les deux dernières lignes**

```
lexik_jwt_authentication:
    secret_key: '%env(resolve:JWT_SECRET_KEY)%'
    public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
    pass_phrase: '%env(JWT_PASSPHRASE)%'
    user_identity_field: 'email'
    user_id_claim: 'username'
```


Par défaut, la durée de vie d'un jeton est de 60 minutes mais on peut la reconfigurer avec `token_ttl`

```
lexik_jwt_authentication:
    secret_key: '%env(resolve:JWT_SECRET_KEY)%'
    public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
    pass_phrase: '%env(JWT_PASSPHRASE)%'
    user_identity_field: 'email'
    user_id_claim: 'username'
    token_ttl: 7200
```

Symfony

Pour exiger le rôle `ROLE_ADMIN` aux demandeurs de la ressource `Personne`

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    attributes: ["security" => "is_granted('ROLE_ADMIN')"],
    collectionOperations: ["get", "post"],
    itemOperations: ["get", "put", "delete"],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
#[ApiFilter(SearchFilter::class, properties: ["adresses.ville" => "
  partial"])]
class Personne
{
    // ...
}
```

Symfony

Pour obtenir le jeton avec **Postman**

- Dans la liste déroulante, choisir `POST` puis saisir l'url vers notre web service `http://localhost:8000/authentication_token`
- Dans le `Headers`, saisir `Content-Type` comme `Key` et `application/json` comme `Value`
- Ensuite cliquer sur `Body`, cocher `raw`, choisir `JSON (application/json)` et saisir des données sous format `JSON` correspondant à l'objet personne à ajouter

```
{
    "email": "john@wick.us",
    "password": "wick"
}
```

- Cliquer sur `Send` puis copier le jeton

Symfony

Pour obtenir la liste des personnes avec **Postman**

- Dans la liste déroulante, choisir `GET` puis saisir l'url vers notre web service `http://localhost:8000/api/personnes`
- Dans le `Headers`, saisir `Content-Type` comme `Key` et `application/json` comme `Value`
- Dans `Authorization`, cliquer sur `Type`, choisir `Bearer Token` et coller le token
- Cliquer sur `Send`

Symfony

Il est possible de sécuriser l'accès seulement à certaines méthodes (mais pas toutes)

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: [
        "get",
        "post"  => ["security" => "is_granted('ROLE_ADMIN')"]
    ],
    itemOperations: [
        "get",
        "put" => ["security" => "is_granted('ROLE_ADMIN')"],
        "delete" => ["security" => "is_granted('ROLE_ADMIN')"]
    ],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
#[ApiFilter(SearchFilter::class, properties: ["adresses.ville" => "partial"])]
class Personne
{
    // ...
}
```

Symfony

Il est possible de sécuriser l'accès seulement à certaines méthodes (mais pas toutes)

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    collectionOperations: [
        "get",
        "post"  => ["security" => "is_granted('ROLE_ADMIN')"]
    ],
    itemOperations: [
        "get",
        "put" => ["security" => "is_granted('ROLE_ADMIN')"],
        "delete" => ["security" => "is_granted('ROLE_ADMIN')"]
    ],
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
#[ApiFilter(SearchFilter::class, properties: ["adresses.ville" => "partial"])]
class Personne
{
    // ...
}
```

N'oublions pas de commenter le contenu de la section `access_control` dans `security.yaml`

Symfony

Comment récupérer l'utilisateur connecté ?

- L'utilisateur connecté peut être récupéré depuis le service `Security`
- On crée un contrôleur
  - qu'on référence dans l'entité `User`
  - qui injecte le service `Security` et qui retourne l'utilisateur connecté

Symfony

Définissons une route `fromToken` accessible via le verbe `GET` et qui référence le contrôleur `ConnectedController`

```
#[ORM\Entity(repositoryClass: UserRepository::class)]
#[ApiResource(
    collectionOperations: [
        'from_token' => [
            'path' => '/fromToken',
            'method' => 'GET',
            'controller' => ConnectedController::class,
        ]
    ]
)]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    // ...
}
```

Symfony

Contenu de `ConnectedController`

```php
namespace App\Controller;

use App\Entity\User;
use Symfony\Component\Security\Core\Security;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ConnectedController extends AbstractController
{
    public function __construct(private Security $security)
    {
    }

    public function __invoke($data)
    {
        return $this->json($this->security->getUser(), 200) ;
    }
}
```