Symfony : 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/symfony_api_platform.git`
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. Créer la migration : `php bin/console make:migration`
6. Exécuter la migration : `php bin/console d:m:m`
7. Envoyez les fixtures (les tuples) à la base de données : `php bin/console d:f:l`
8. 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`

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


Explication

- `api_platform` : l'identifiant de toutes les routes liées à l'API exposée par **API Platform**.
- `resource: .` : indique que la ressource de la route est le répertoire actuel, ce qui permet à **API Platform** de scanner toutes les ressources et configurations disponibles.
- `type: api_platform` : demande à **Symfony** qu'il doit déléguer les requêtes à **API Platform** pour traiter et répondre aux appels API.
- `prefix: /api` : définit un préfixe `/api` à toutes les routes générées par **API Platform**.

Symfony

Contenu de `config/package/api_platform.yaml`

```
api_platform:
    title: Hello API Platform
    version: 1.0.0
    defaults:
        stateless: true
        cache_headers:
            vary: ['Content-Type', 'Authorization', 'Origin']
```

Explication

- Ce fichier permet de configurer le bundle (ou package) **API Platform** utilisé par **Symfony**
- `title: Hello API Platform` et `version: 1.0.0` : permettent de fournir un titre et une version dans la documentation.
- `defaults` : définit des valeurs par défaut pour toutes les ressources et opérations configurées dans **API Platform**.
- `stateless: true` : indique que l'API est `stateless` (sans état, n'utilise aucune donnée de session).
- `cache_headers` : configure les en-têtes **HTTP** de mise en cache par défaut.
- `Vary` : indique que la réponse peut changer en fonction de certains en-têtes spécifiques de la requête.
  - `Content-Type` : signifie que la réponse peut varier en fonction du type de contenu demandé.
  - `Authorization` : signifie que la réponse peut varier selon que l'utilisateur est connecté ou selon son rôle.
  - `Origin` : indique que la réponse peut varier en fonction de l'origine.

Explication

- Ce fichier permet de configurer le bundle (ou package) **API Platform** utilisé par **Symfony**
- `title: Hello API Platform` et `version: 1.0.0` : permettent de fournir un titre et une version dans la documentation.
- `defaults` : définit des valeurs par défaut pour toutes les ressources et opérations configurées dans **API Platform**.
- `stateless: true` : indique que l'API est `stateless` (sans état, n'utilise aucune donnée de session).
- `cache_headers` : configure les en-têtes **HTTP** de mise en cache par défaut.
- `Vary` : indique que la réponse peut changer en fonction de certains en-têtes spécifiques de la requête.
  - `Content-Type` : signifie que la réponse peut varier en fonction du type de contenu demandé.
  - `Authorization` : signifie que la réponse peut varier selon que l'utilisateur est connecté ou selon son rôle.
  - `Origin` : indique que la réponse peut varier en fonction de l'origine.

Exemple

- Supposons qu'une API peut renvoyer des données **JSON** ou **XML** selon l'en-tête de la requête `Content-Type`.
- Sans `Vary: Content-Type`, le cache pourrait toujours renvoyer la même réponse (par exemple, **JSON**) même si un autre client demande **XML**.

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\Metadata\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;


#[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\Metadata\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;


#[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

Dans `config/packages/api_platform.yaml`**, ajoutons la propriété suivante**

```
api_platform:
    formats:
        json: ['application/json']
        jsonld: ['application/ld+json']
```

Symfony

Dans `config/packages/api_platform.yaml`**, ajoutons la propriété suivante**

```
api_platform:
    formats:
        json: ['application/json']
        jsonld: ['application/ld+json']
```


Pourquoi garder **JSON-LD** activé ?

API Platform utilise **JSON-LD** comme format principal pour :

- Fournir un support natif pour le Web sémantique (**L**inked **D**ata).
- Permettre une auto-documentation de l'**API**.

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/30,
            api/adresses/31
        ]
    },

]
```

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

Critère	PUT	PATCH
Définition	Remplace complètement une ressource	Modifie partiellement une ressource
Données envoyées	Objet complet	Uniquement les champs modifiés
Utilisation principale	Mise à jour complète	Modification partielle
Impact sur les données	Peut supprimer des champs omis	Ne modifie que les champs spécifiés
Cas d'usage	Remplacement de la ressource	Mise à jour de certains champs

Et si nous voulions exposer seulement quelques méthodes

```php
// les use précédents

use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\GetCollection;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    operations: [
        new Get(),
        new GetCollection()
    ]
)]
class Personne
{
    // ....
}
```

Et si nous voulions exposer seulement quelques méthodes

```php
// les use précédents

use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\GetCollection;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    operations: [
        new Get(),
        new GetCollection()
    ]
)]
class Personne
{
    // ....
}
```


Les méthodes disponibles

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

Symfony

Exemple avec les méthodes d'écriture (L'insertion est la seule action non autorisée)

```
// les use précédents

use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\Put;
use ApiPlatform\Metadata\Delete;
use ApiPlatform\Metadata\GetCollection;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    operations: [
        new Get(),
        new GetCollection(),
        new Delete(),
        new Put()
    ]
)]
class Personne
{
    // ....
}
```

Symfony

Pour définir une contrainte sur le paramètre de la méthode `Get`

```
// les use précédents

use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\Put;
use ApiPlatform\Metadata\Delete;
use ApiPlatform\Metadata\GetCollection;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    operations: [
        new Get(requirements: ['id' => '\d+']),
        new GetCollection(),
        new Delete(),
        new Put()
    ]
)]
class Personne
{
    // ....
}
```

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

```php
// les use précédents

use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\Put;
use ApiPlatform\Metadata\Delete;
use ApiPlatform\Metadata\GetCollection;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    operations: [
        new Get(
            requirements: ['id' => '\d+'],
            uriTemplate: '/person/{id}',
            status: 301
        ),
        new GetCollection(),
        new Delete(),
        new Put()
    ]
)]
class Personne
{
    // ....
}
```

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

```php
// les use précédents

use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\Put;
use ApiPlatform\Metadata\Delete;
use ApiPlatform\Metadata\GetCollection;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    operations: [
        new Get(
            requirements: ['id' => '\d+'],
            uriTemplate: '/person/{id}',
            status: 301
        ),
        new GetCollection(),
        new Delete(),
        new Put()
    ]
)]
class Personne
{
    // ....
}
```

Pour tester `localhost:8000/api/person/1`

Symfony

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

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

Symfony

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

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


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

Symfony

Et une autre manière d'interdire certains verbes HTTP

```
// les use précédents

use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\GetCollection;

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

Symfony

Et une autre manière d'interdire certains verbes HTTP

```php
// les use précédents

use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\GetCollection;

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

Ainsi, les seules actions autorisées sont les actions de lecture.

Symfony

On ne peuy pas combiner les attributs verbes (comme `#[PUT()]` indépendamment après `#[ApiResource(operations: [new Get()])]`

```
#[ApiResource(operations: [new Get()])]
#[PUT()] //  incorrecte]
class Personne
{
    // ....
}
```

Symfony

On ne peuy pas combiner les attributs verbes (comme `#[PUT()]` indépendamment après `#[ApiResource(operations: [new Get()])]`

```
#[ApiResource(operations: [new Get()])]
#[PUT()] //  incorrecte]
class Personne
{
    // ....
}
```


La solution

```
#[ApiResource(operations: [new Get(), new Put()])]
class Personne
{
    // ....
}
```

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:
  title: Hello API Platform
  version: 1.0.0
  defaults:
    stateless: true
    cache_headers:
      vary: ["Content-Type", "Authorization", "Origin"]
    pagination_items_per_page: 5
  formats:
    json: ["application/json"]
    jsonld: ['application/ld+json']
```

Symfony

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

```
api_platform:
  title: Hello API Platform
  version: 1.0.0
  defaults:
    stateless: true
    cache_headers:
      vary: ["Content-Type", "Authorization", "Origin"]
    pagination_items_per_page: 5
  formats:
    json: ["application/json"]
    jsonld: ['application/ld+json']
```

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)]
#[GetCollection(paginationItemsPerPage: 3)]
#[Get]
class Personne
{
    // ....
}
```

Symfony

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

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[GetCollection(paginationItemsPerPage: 3)]
#[Get]
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

Symfony

Ou avec `ApiResource`

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(paginationItemsPerPage: 30)]
class Personne
{
    // ....
}
```

Pour pouvoir spécifier le numéro de la page

```
api_platform:
    title: Hello API Platform
    version: 1.0.0
    defaults:
        stateless: true
        cache_headers:
            vary: ['Content-Type', 'Authorization', 'Origin']
        pagination_items_per_page: 5
    formats:
        json: ['application/json']
        jsonld: ['application/ld+json']
    collection:
        pagination:
            page_parameter_name: page_number
```

Pour pouvoir spécifier le numéro de la page

```
api_platform:
    title: Hello API Platform
    version: 1.0.0
    defaults:
        stateless: true
        cache_headers:
            vary: ['Content-Type', 'Authorization', 'Origin']
        pagination_items_per_page: 5
    formats:
        json: ['application/json']
        jsonld: ['application/ld+json']
    collection:
        pagination:
            page_parameter_name: page_number
```

URL pour récupérer les 3 personnes de la deuxième page

`localhost:8000/api/personnes?page_number=2`

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

```php
class Personne
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?int $id = null;

    #[ORM\Column(length: 255, nullable: true)]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?string $nom = null;

    #[ORM\Column(length: 255, nullable: true)]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?string $prenom = null;

    /**
     * @var Collection<int, Adresse>
     */
    #[ORM\ManyToMany(targetEntity: Adresse::class, cascade: ['persist', 'remove'])]
    #[Groups(['personne:read', 'personne:write'])]
    private Collection $adresses;

    // ...

}
```

Symfony

Nouveau contenu de l'entité `Adresse`

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

    #[ORM\Column(length: 255, nullable: true)]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?string $rue = null;

    #[ORM\Column(length: 255, nullable: true)]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?string $codePostal = null;

    #[ORM\Column(length: 255, nullable: true)]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?string $ville = null;

    // ...

}
```

Et si l'association est bidirectionnelle

```php
class Adresse
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?int $id = null;

    #[ORM\Column(length: 255, nullable: true)]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?string $rue = null;

    #[ORM\Column(length: 255, nullable: true)]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?string $codePostal = null;

    #[ORM\Column(length: 255, nullable: true)]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?string $ville = null;

    /**
     * @var Collection<int, Personne>
     */
    #[ORM\ManyToMany(targetEntity: Personne::class, mappedBy: 'adresses', cascade: ['persist',
      'remove'])]
    #[Groups(['adresse:read', 'adresse:write'])]
    private Collection $personnes;

    public function __construct()
    {
        $this->personnes = new ArrayCollection();
    }
    // ...
}
```

Et dans `Personne`

```php
class Personne
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?int $id = null;

    #[ORM\Column(length: 255, nullable: true)]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?string $nom = null;

    #[ORM\Column(length: 255, nullable: true)]
    #[Groups(['personne:read', 'personne:write', 'adresse:read', 'adresse:write'])]
    private ?string $prenom = null;

    /**
     * @var Collection<int, Adresse>
     */
    #[ORM\ManyToMany(targetEntity: Adresse::class, inversedBy: 'personnes', cascade: ['persist'
      , 'remove'])]
    #[Groups(['personne:read', 'personne:write'])]
    private Collection $adresses;

    public function __construct()
    {
        $this->adresses = new ArrayCollection();
    }

    // ...

}
```

Symfony

Ajoutons les attributs suivants à l'attribut `#ApiResource`

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

Symfony

Ajoutons les attributs suivants à l'attribut `#ApiResource`

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


Explication

- Les attributs ayant le groupe `"personne:read"` seront accessibles en lecture
- Les attributs ayant le groupe `"personne:write"` seront accessibles en écriture
- 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

Et dans `Adresse`

```
#[ORM\Entity(repositoryClass: AdresseRepository::class)]
#[ApiResource(
    normalizationContext: ["groups" => ['adresse:read']],
    denormalizationContext: ["groups" => ['adresse:write']]
)]
class Adresse
{
    // ....
}
```

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

Extrait du résultat retourné en allant à `http://127.0.0.1:8000/api/adresses/59`


```
[
    {
        "id": 59,
        "rue": "impasse de Legendre",
        "codePostal": "17758",
        "ville": "Rodrigues-les-Bains",
        "personnes": [
            {
                "id": 28,
                "nom": "Rodriguez",
                "prenom": "Auguste"
            }
        ]
    },
}
```

Ajouter une personne avec **Postman**

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

```
{
    "nom": "Mitroglou",
    "prenom": "Kostas",
    "adresses": [
        {
            "rue": "republique",
            "codePostal": "13002",
            "ville": "Marseille"
        },
        {
            "rue": "picpus",
            "codePostal": "75012",
            "ville": "Paris"
        }
    ]
}
```

- Cliquer sur `Send`

Ajouter une adresse avec **Postman**

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

```
{
    "rue": "avenue de lacanau",
    "codePostal": "13700",
    "ville": "Marignane",
    "personnes": [
        {
            "nom": "Rodriguez",
            "prenom": "Louis"
        }
    ]
}
```

- Cliquer sur `Send`

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 `ProcessorInterface` pour intercepter les données utilisateur et ajouter la date

Symfony

Nouveau contenu après avoir ajouté l'attribut `dateEnregistrement`

```
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: Types::DATETIME_MUTABLE, nullable: false)]
    #[Groups(['personne:read'])]
    private ?\DateTimeInterface $dateEnregistrement = null;

    // ...

}
```

Symfony

Rien à changer dans l'entité `Adresse`

```
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 `ProcessorInterface`

Symfony

`ProcessorInterface`

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

Symfony

`ProcessorInterface`

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

Créons `PersonneProcessor`

```
php bin/console make:state-processor
PersonneProcessor
```

Symfony

Contenu de la classe `PersonneProcessor` **définie dans** `src/state`

```php
<?php

namespace App\State;

use ApiPlatform\Metadata\Operation;
use ApiPlatform\State\ProcessorInterface;

class PersonneProcessor implements ProcessorInterface
{

    public function process(mixed $data, Operation $operation, array $uriVariables = [], array
       $context = []): void
    {
    }
}
```

Symfony

Commençons par injecter `EntityManagerInterface` **dans le constructeur de** `PersonneProcessor`

```php
<?php

namespace App\State;

use ApiPlatform\Metadata\Operation;
use Doctrine\ORM\EntityManagerInterface;
use ApiPlatform\State\ProcessorInterface;

class PersonneProcessor implements ProcessorInterface
{
    public function __construct(private EntityManagerInterface $em)
    {
    }

    public function process(mixed $data, Operation $operation, array $uriVariables = [], array
       $context = [])
    {

    }
}
```

Symfony

Ajoutons la date du jour aux personnes interceptées

```php
<?php

namespace App\State;

use ApiPlatform\Metadata\Operation;
use Doctrine\ORM\EntityManagerInterface;
use ApiPlatform\State\ProcessorInterface;

class PersonneProcessor implements ProcessorInterface
{
    public function __construct(private EntityManagerInterface $em)
    {
    }

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

Symfony

Associons `PersonneProcessor` **à notre entité** `Personne`

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    processor: PersonneProcessor::class,
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
class Personne
{
    // ...
}
```

Symfony

Exercice

- Modifier la dernière personne ajoutée
- Vérifier que la date d'enregistrement a été mise à jour

Symfony

Exercice

- Modifier la dernière personne ajoutée
- Vérifier que la date d'enregistrement a été mise à jour

Pour permettre à API Platform de traiter les requêtes PATCH, il faut configurer `config/packages/api_platform.yaml`

```
patch_formats:
    json: ['application/json']
    jsonmergepatch: ['application/merge-patch+json']
```

Symfony

	application/json	**application/merge-patch+json**
Utilisation principale	Créer ou remplacer une ressource entière (POST, PUT).	Mettre à jour partiellement une ressource (PATCH).
Contenu envoyé	Ressource complète avec toutes ses propriétés.	Seules les propriétés à mettre à jour sont envoyées.
Comportement	Remplace la ressource entière.	Fait une mise à jour partielle en fusionnant les changements.

Symfony

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

```
public function process(mixed $data, Operation $operation, array
   $uriVariables = [], array $context = [])
{
    if ($operation instanceof Post) {
        $dateTimeZone = new \DateTimeZone('Europe/Paris');
        $data->setDateEnregistrement(new \DateTime('now', $dateTimeZone
           ));
        $this->em->persist($data);
        $this->em->flush();
        return $data;
    }
}
```

Symfony

State Provider

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

Symfony

State Provider

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

Créons `PersonneProvider`

```
php bin/console make:state-provider PersonneProvider
```

Symfony

Contenu de la classe `PersonneProvider` **définie dans** `src/state`

```
namespace App\State;

use ApiPlatform\Metadata\Operation;
use ApiPlatform\State\ProviderInterface;

class PersonneProvider implements ProviderInterface
{
    public function provide(Operation $operation, array $uriVariables = [], array $context =
       []): object|array|null
    {
        // Retrieve the state from somewhere
    }
}
```

Symfony

Injectons `EntityManagerInterface` **dans le constructeur de la classe** `PersonneProvider`

```
namespace App\State;

use ApiPlatform\Metadata\Operation;
use ApiPlatform\State\ProviderInterface;
use Doctrine\ORM\EntityManagerInterface;

class PersonneProvider implements ProviderInterface
{
    public function __construct(private EntityManagerInterface $em)
    {
    }

    public function provide(Operation $operation, array $uriVariables = [], array $context =
       []): array|object
    {
        // Retrieve the state from somewhere
    }
}
```

Nouveau contenu de la classe `PersonneProvider`

```
namespace App\State;

use App\Entity\Personne;
use ApiPlatform\Metadata\Operation;
use ApiPlatform\State\ProviderInterface;
use Doctrine\ORM\EntityManagerInterface;
use ApiPlatform\Metadata\CollectionOperationInterface;

class PersonneProvider implements ProviderInterface
{
    public function __construct(private EntityManagerInterface $em)
    {
    }
    public function provide(Operation $operation, array $uriVariables = [], array $context =
       []): array|object
    {

        if ($operation instanceof CollectionOperationInterface) {
            $personnes = $this->em->getRepository(Personne::class)->findAll();
            return  array_map(fn ($elt) => $elt->setNom(strtoupper($elt->getNom())), $personnes
               );
        }
        $personne = $this->em->getRepository(Personne::class)->find($uriVariables['id']);
        return $personne->setNom(strtoupper($personne->getNom()));
    }
}
```

Symfony

Associons `PersonneProvider` **à notre entité** `Personne`

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    provider: PersonneProvider::class,
    processor: PersonneProcessor::class,
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
class Personne
{
    // ...
}
```

Symfony

Associons `PersonneProvider` **à notre entité** `Personne`

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    provider: PersonneProvider::class,
    processor: PersonneProcessor::class,
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]]
)]
class Personne
{
    // ...
}
```


Remarque

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

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 `#[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 ApiPlatform\Metadata\ApiFilter;
use ApiPlatform\Metadata\ApiResource;
use App\Repository\AdresseRepository;
use ApiPlatform\Doctrine\Orm\Filter\SearchFilter;
use Symfony\Component\Serializer\Attribute\Groups;

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

Symfony

Ajoutons `#[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 ApiPlatform\Metadata\ApiFilter;
use ApiPlatform\Metadata\ApiResource;
use App\Repository\AdresseRepository;
use ApiPlatform\Doctrine\Orm\Filter\SearchFilter;
use Symfony\Component\Serializer\Attribute\Groups;

#[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

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 ApiPlatform\Metadata\ApiFilter;
use ApiPlatform\Metadata\ApiResource;
use App\Repository\AdresseRepository;
use ApiPlatform\Doctrine\Orm\Filter\RangeFilter;
use ApiPlatform\Doctrine\Orm\Filter\SearchFilter;
use Symfony\Component\Serializer\Attribute\Groups;

#[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 ApiPlatform\Metadata\ApiFilter;
use ApiPlatform\Metadata\ApiResource;
use App\Repository\AdresseRepository;
use ApiPlatform\Doctrine\Orm\Filter\RangeFilter;
use ApiPlatform\Doctrine\Orm\Filter\SearchFilter;
use Symfony\Component\Serializer\Attribute\Groups;

#[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)

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(length: 255, 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 ?string $nom = null;
```

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

```
#[ORM\Column(length: 255, 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 ?string $nom = null;
```

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

```
#[ORM\Column(length: 255, nullable: true)]
#[Groups(["personne:read", "adresse:read", "personne:write"])]
#[Assert\Regex(
    pattern: '/\d/',
    match: false,
    message: 'Le prénom ne peut contenir de chiffres',
)]
private ?string $prenom = null;
```

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

```
#[ORM\Column(length: 255, 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 ?string $nom = null;
```

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

```
#[ORM\Column(length: 255, nullable: true)]
#[Groups(["personne:read", "adresse:read", "personne:write"])]
#[Assert\Regex(
    pattern: '/\d/',
    match: false,
    message: 'Le prénom ne peut contenir de chiffres',
)]
private ?string $prenom = null;
```

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)
- Documentation officielle : `https://jwt.io/introduction/`

Symfony

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)

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

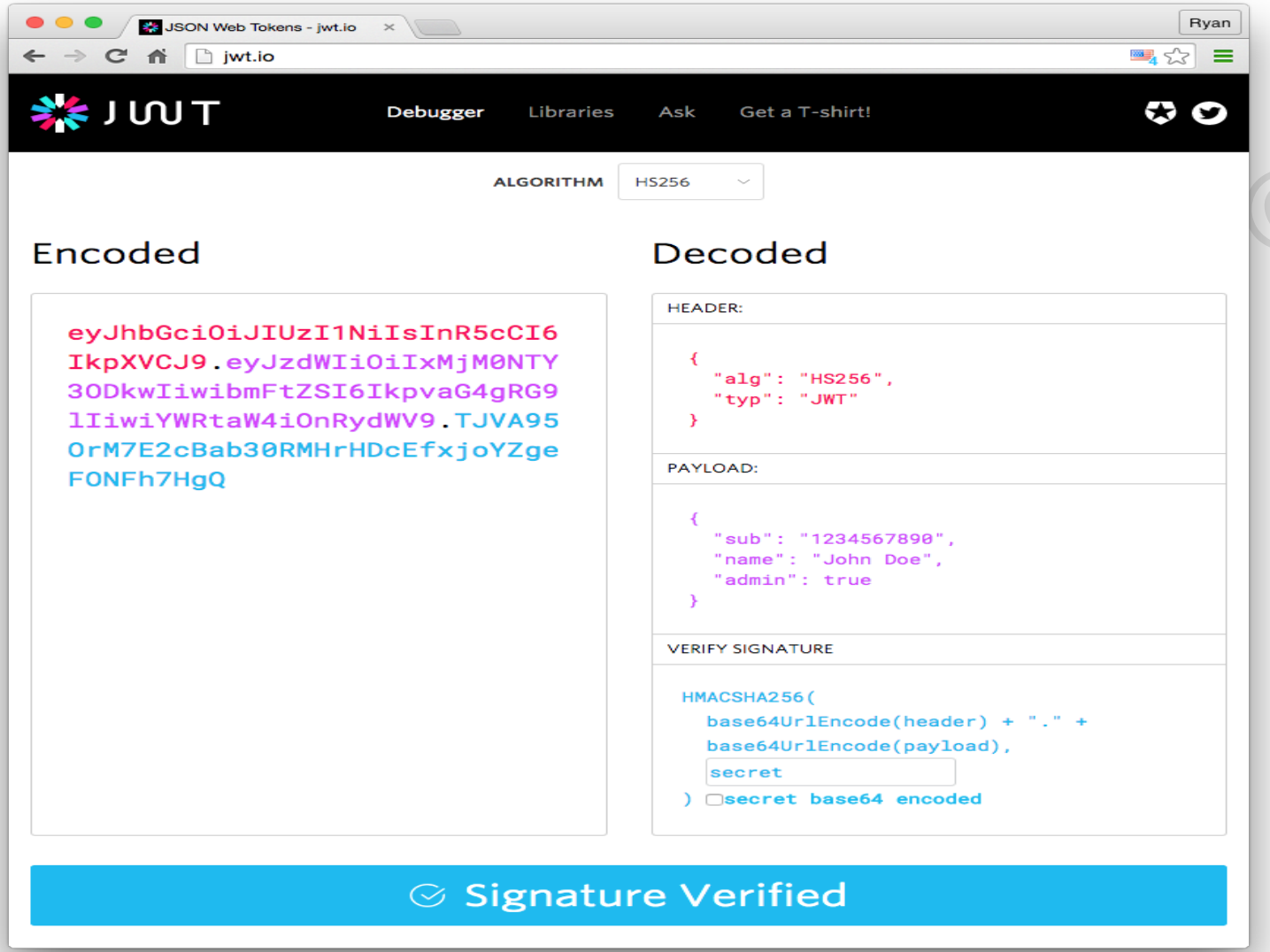
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```


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

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
            stateless: true

    access_control:
        - { path: ^/api$, roles: PUBLIC_ACCESS }
        - { path: ^/*, roles: [IS_AUTHENTICATED_FULLY] }
```

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

Générerons un fichier de données aléatoires pour les utilisateurs

```
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`

```php
class UserFixtures extends Fixture
{
    public function __construct(private UserPasswordHasherInterface $passwordEncoder)
    {
    }
    public function load(ObjectManager $manager): void
    {
        $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` (pour ajouter la date d'enregistrement)

```
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

Mettons à jour `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
            stateless: true # connexion sans état
            http_basic: ~ # active l'authentification HTTP Basic sans option supplémentaire

    access_control:
        - { path: ^/api$, roles: PUBLIC_ACCESS } # autorise Swagger
        - { path: ^/*, roles: [IS_AUTHENTICATED_FULLY] } # exige une authentification
```

Symfony

Pour tester

- Allez à `http://127.0.0.1:8000/api/personnes` et vérifiez l'affichage du popup d'authentification
- Utilisez `john@wick.us` comme nom d'utilisateur et `wick` comme mot de passé et vérifiez que la ressource est accessible

Symfony

Pour tester

- Allez à `http://127.0.0.1:8000/api/personnes` et vérifiez l'affichage du popup d'authentification
- Utilisez `john@wick.us` comme nom d'utilisateur et `wick` comme mot de passé et vérifiez que la ressource est accessible

Pour accéder à la ressource depuis **Postman**

- Dans la liste déroulante, choisir `GET` puis saisir l'URL vers notre web service `http://localhost:8080/personnes`
- Ensuite cliquer sur `Authorization` et choisissez `Basic Auth`
- Utilisez `john@wick.us` comme nom d'utilisateur et `wick` comme mot de passé et vérifiez que la ressource est accessible

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

Intégrons le bundle JWT dans notre projet

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

Ou l'alias

```
composer require jwt
```

En cas de problème d'installation avec `ext-sodium`

```
composer require jwt --ignore-platform-req=ext-sodium
```

Symfony

Pour la suite, commentons ou désactivons les parties suivantes dans `security.yaml`

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

        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
      #      stateless: true # connexion sans état
      #      http_basic: ~ # active l'authentification HTTP Basic sans option supplémentaire

    access_control:
      #  - { path: ^/api$, roles: PUBLIC_ACCESS } # autorise Swagger
      #  - { path: ^/*, roles: [IS_AUTHENTICATED_FULLY] } # exige une authentification
```

Symfony

Documentation sur **GitHub**

https://github.com/lexik/LexikJWTAuthenticationBundle/blob/3.x/Resources/doc/index.rst

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 ###
```

Symfony

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

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

Modifiez les sections `firewalls` **et** `access_control` **de** `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)|css|images|js)/
            security: false
        main:
            stateless: true
            provider: app_user_provider
            json_login:
                check_path: authentication_token # le nom de la route dans routes.yaml
                username_path: email
                password_path: password
                success_handler: lexik_jwt_authentication.handler.authentication_success
                failure_handler: lexik_jwt_authentication.handler.authentication_failure
            jwt: ~

    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

Explication

- `stateless` : Définit si l'authentification doit être sans état. Pour l'authentification **JWT**, ce paramètre doit être à `true`, car chaque requête doit être indépendante.
- `provider` : Identifie le fournisseur d'utilisateurs lors de l'authentification. `app_user_provider` a été déjà défini dans `providers`.
- `check_path` : Spécifie le chemin utilisé pour l'authentification par token. Ce chemin doit être déjà défini dans `routes.yaml`.
- `username_path` et `password_path` : Indiquent où trouver dans le corps de la requête JSON les informations d'identification.
- `success_handler` et `failure_handler` : Définissent les services `LexikJWTAuthenticationBundle` à utiliser en cas de succès ou d'échec de l'authentification : ça gère l'émission d'un **JWT** en cas de succès, ou retourne une erreur en cas d'échec.
- `jwt:` $\sim$ Indique que l'authentification **JWT** est activée.

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` et vérifier la présence du jeton

Symfony

Par défaut, la durée de vie d'un jeton est de 60 minutes mais on peut la reconfigurer avec `token_ttl` (ttl : time to live) : 2 heures dans cet exemple

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

Symfony

Pour personnaliser le contenu du jeton

```php
<?php

namespace App\EventListener;

use Symfony\Component\HttpFoundation\RequestStack;
use Lexik\Bundle\JWTAuthenticationBundle\Event\JWTCreatedEvent;

class JWTCreatedListener
{
    public function __construct(private RequestStack $requestStack)
    {
        // objet contenant les données de la requête
        $this->requestStack = $requestStack;
    }


    public function onJWTCreated(JWTCreatedEvent $event)
    {
        // Récupérer l'utilisateur actuel
        $user = $event->getUser();

        $payload = $event->getData();

        // Ajouter l'ID utilisateur dans le jeton
        $payload['userId'] = $user->getId();

        $event->setData($payload);
    }
}
```

Symfony

Et enregistrons le dans `config/services.yaml`

```
services:
    acme_api.event.jwt_created_listener:
        class: App\EventListener\JWTCreatedListener
        arguments: [ '@request_stack' ]
        tags:
            - { name: kernel.event_listener, event: lexik_jwt_authentication.on_jwt_created,
              method: onJWTCreated }
```

Symfony

Depuis **Postman**, demandons un jeton et vérifions la présence de `userId`

- 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`

Symfony

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

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
#[ApiResource(
    normalizationContext: ["groups" => ["personne:read"]],
    denormalizationContext: ["groups" => ["personne:write"]],
    security: "is_granted('ROLE_ADMIN')"
)]
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)

```
#[Get]
#[Put(security: "is_granted('ROLE_ADMIN')")]
#[GetCollection]
#[Post(security: "is_granted('ROLE_ADMIN')")]
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    // ...
}
```

Symfony

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

```
#[Get]
#[Put(security: "is_granted('ROLE_ADMIN')")]
#[GetCollection]
#[Post(security: "is_granted('ROLE_ADMIN')")]
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
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(
    operations: [
        new Get(
            name: 'publication',
            uriTemplate : '/fromToken',
            controller : ConnectedController::class,
        )
    ]
)]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    // ...
}
```

Symfony

Contenu de `ConnectedController`

```php
<?php

namespace App\Controller;

use Symfony\Bundle\SecurityBundle\Security;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;


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

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

Symfony

Intégrons le bundle de rafraichissement de JWT dans notre projet

```
composer require gesdinet/jwt-refresh-token-bundle
```

Symfony

Intégrons le bundle de rafraichissement de JWT dans notre projet

```
composer require gesdinet/jwt-refresh-token-bundle
```

En cas de problème d'installation avec `ext-sodium`

```
composer require gesdinet/jwt-refresh-token-bundle
--ignore-platform-req=ext-sodium
```

Symfony

Documentation sur **GitHub**

https://packagist.org/packages/gesdinet/jwt-refresh-token-bundle

Symfony

Vérifiez que le contenu suivant a été généré dans
`config/packages/gesdinet_jwt_refresh_token.yaml`

```
gesdinet_jwt_refresh_token:
    refresh_token_class: App\Entity\RefreshToken
```

Symfony

Vérifiez que le contenu suivant a été généré dans `config/packages/gesdinet_jwt_refresh_token.yaml`

```
gesdinet_jwt_refresh_token:
    refresh_token_class: App\Entity\RefreshToken
```


Pour créer cette entité `RefreshToken` **qui va servir à stocker le jeton de rafraichissement**

```
php bin/console make:entity RefreshToken
```

Symfony

Vérifiez que le contenu suivant a été généré dans `config/packages/gesdinet_jwt_refresh_token.yaml`

```
gesdinet_jwt_refresh_token:
    refresh_token_class: App\Entity\RefreshToken
```


Pour créer cette entité `RefreshToken` **qui va servir à stocker le jeton de rafraichissement**

```
php bin/console make:entity RefreshToken
```

Remarque

N'oublier pas la migration.

Symfony

Contenu de `RefreshToken`

```php
<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use Gesdinet\JWTRefreshTokenBundle\Entity\RefreshToken as
   BaseRefreshToken;

#[ORM\Entity]
#[ORM\Table(name: 'refresh_tokens')]
class RefreshToken extends BaseRefreshToken
{
}
```

Symfony

Modifiez les sections `firewalls` **et** `access_control` **de** `security.yaml`

```
firewalls:
    dev:
        pattern: ^/_(profiler|wdt)
        security: false
    main:
        stateless: true
        provider: app_user_provider
        entry_point: jwt
        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
        jwt: ~
        refresh_jwt:
            check_path: /refresh_token

    refresh:
        pattern:  ^/refresh_token
        stateless: true

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

Symfony

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

```
api_refresh_token:
    path:       /refresh_token
    controller: gesdinet.jwtrefreshtoken::refresh
```

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` et vérifier la présence du jeton

Symfony

Pour obtenir le **Access Token** depuis le **Refresh Token** avec **Postman**

- Dans la liste déroulante, choisir `POST` puis saisir l'URL vers notre web service `http://localhost:8000/refresh_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

```
{
    "refresh_token": "coller votre refresh token ici"
}
```

- Cliquer sur `Send` puis copier le jeton

Symfony

Par défaut, la durée de vie d'un jeton de rafraichissement est de 1 mois mais on peut la reconfigurer (dans `config/packages/gesdinet_jwt_refresh_token.yaml`)

```
gesdinet_jwt_refresh_token:
    refresh_token_class: App\Entity\RefreshToken
    ttl: 2592000
```

Symfony

Par défaut, la durée de vie d'un jeton de rafraichissement est de 1 mois mais on peut la reconfigurer (dans `config/packages/gesdinet_jwt_refresh_token.yaml`)

```
gesdinet_jwt_refresh_token:
    refresh_token_class: App\Entity\RefreshToken
    ttl: 2592000
```


Pour renommer la clé dans l'objet contenant les deux tokens (par défaut, c'est `refresh_token`)

```
gesdinet_jwt_refresh_token:
    refresh_token_class: App\Entity\RefreshToken
    ttl: 2592000
    token_parameter_name: refreshToken
```