

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**
- Utilisant **Swagger UI** (<https://swagger.io/tools/swagger-ui/>) pour visualiser et interagir avec les **API REST**

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**
- Utilisant **Swagger UI** (<https://swagger.io/tools/swagger-ui/>) pour visualiser et interagir avec les **API REST**

Installation avec **Flex**

composer require api

Symfony

Contenu de `config/routes/api_platform.yaml`

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

© Achref EL MOUELHI ©

Symfony

Contenu de `config/routes/api_platform.yaml`

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

Pour éviter le conflit avec nos travaux précédents, remplaçons `prefix: /api` par `prefix: /ws`

Symfony

Contenu de `config/routes/api_platform.yaml`

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

Pour éviter le conflit avec nos travaux précédents, remplaçons `prefix: /api` par `prefix: /ws`

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

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

Symfony

Ajoutons l'annotation `@ApiResponse` à notre entité `Personne`

```
namespace App\Entity;

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

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResponse
 */
class Personne
{
```

Symfony

Ajoutons l'annotation `@ApiResponse` à notre entité `Personne`

```
namespace App\Entity;

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

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResponse
 */
class Personne
{
```

La même chose pour l'entité `Adresse`

Symfony

Pour tester

Allez sur `localhost:8000/ws`

Symfony

Deux types d'opérations

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

Symfony

Et si nous voulions exposer seulement quelques méthodes

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResponse(
 *     collectionOperations={"get"},
 *     itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

© Achref EL

Symfony

Et si nous voulions exposer seulement quelques méthodes

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResponse(
 *     collectionOperations={"get"},
 *     itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Les méthodes disponibles

- GET : /ws/personnes
- GET : /ws/personnes/{id}
- DELETE : /ws/personnes/{id}
- PUT : /ws/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)
 * @ApiResponse(
 *     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)
 * @ApiResponse(
 *     collectionOperations={
 *         "get"={"path":"/person", "status":301}
 *     },
 *     itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Pour tester `localhost:8000/ws/person`

Symfony

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

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::
    class)
 * @ApiResponse(
     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)
 * @ApiResponse(
     routePrefix="/library"
 *     collectionOperations={
         "get"={"path"="/person", "status"=301}
     },
 *     itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Pour tester localhost:8000/ws/library/person

Symfony

Pour définir le nombre d'objets à retourner par requête (page)

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::
    class)
 * @ApiResponse(
 *     attributes={"pagination_items_per_page"]=3}
 *     collectionOperations={
 *         "get"={"path="/person", "status"]=301}
 *     },
 *     itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```


Symfony

Pour définir le nombre d'objets à retourner par requête (page)

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::
    class)
 * @ApiResponse(
 *     attributes={"pagination_items_per_page"]=3}
 *     collectionOperations={
 *         "get"={"path="/person", "status"]=301}
 *     },
 *     itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Pour tester localhost:8000/ws/person?page=1

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 `Groups` et les deux attributs `normalizationContext` et `denormalizationContext`

Nouveau contenu après avoir ajouté l'attribut `dateEnregistrement` et le groupe `personne:write`

```
class Personne
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups({"personne:read", "personne:write"})
     */
    private $id;
    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups({"personne:read", "personne:write"})
     */
    private $nom;
    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups({"personne:read", "personne:write"})
     */
    private $prenom;
    /**
     * @ORM\ManyToMany(targetEntity=Adresse::class, inversedBy="personnes", cascade={"remove",
     "persist"})
     * @Groups({"personne:read", "personne:write"})
     */
    private $addresses;
    /**
     * @ORM\Column(type="datetime")
     * @Groups("personne:read")
     */
    private $dateEnregistrement;
```

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

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResponse(
 *     itemOperations={"get", "put", "delete"},
 *     normalizationContext={"groups"={"personne:read"}},
 *     denormalizationContext={"groups"={"personne:write"}}
 * )
 */
class Personne
```

© Achref EL MOU

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

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResponse(
 *     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

Nouveau contenu de l'entité Adresse

```
/**
 * @ORM\Entity(repositoryClass=AdresseRepository::class)
 * @ApiResource()
 */
class Adresse
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups({"personne:read", "personne:write"})
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups({"personne:read", "personne:write"})
     */
    private $rue;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups({"personne:read", "personne:write"})
     */
    private $codePostal;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups({"personne:read", "personne:write"})
     */
    private $ville;

    /**
     * @ORM\ManyToMany(targetEntity=Personne::class, mappedBy="adresses")
     */
    private $personnes;
```

Symfony

Remarques

- En tentant 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**

© Achille

Symfony

Remarques

- En tentant 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

Explication

- 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`

```
namespace App\DataPersister;

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

class PersonneDataPersister implements DataPersisterInterface
{
    private $entityManager;

    public function __construct(EntityManagerInterface $entityManager)
    {
        $this->entityManager = $entityManager;
    }

    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));
        $this->entityManager->persist($data);
        $this->entityManager->flush();
    }

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

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));
    }
    $this->entityManager->persist($data);
    $this->entityManager->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));
    }
    $this->entityManager->persist($data);
    $this->entityManager->flush();
}
```

Remarque

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

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

```
namespace App\Entity;

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

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource
 */
class Personne
{
    // contenu précédent
    /**
     * @ORM\ManyToMany(targetEntity=Adresse::class, inversedBy="personnes", cascade={"persist"
     * })
     * @Groups({"personne:read", "personne:write"})
     * @ApiSubresource
     */
    private $adresses;
```

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

```
namespace App\Entity;

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

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource
 */
class Personne
{
    // contenu précédent
    /**
     * @ORM\ManyToMany(targetEntity=Adresse::class, inversedBy="personnes", cascade={"persist"})
     * @Groups({"personne:read", "personne:write"})
     * @ApiSubresource
     */
    private $adresses;
```

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

@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
 - ...

@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/>

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
```

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
```

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

Symfony

Valeurs possibles de properties de SearchFilter

- `partial` \equiv `ville like %texte%`
- `exact` \equiv `ville = texte`
- `end` \equiv `ville like %texte`
- `start` \equiv `ville like texte%`



Symfony

Valeurs possibles de propriétés de SearchFilter

- `partial` \equiv `ville like %texte%`
- `exact` \equiv `ville = texte`
- `end` \equiv `ville like %texte`
- `start` \equiv `ville like texte%`

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

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

```
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
```

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

```
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 : `/api/adresses?ville=mars&codePostal[gt]=13006`

Symfony

Valeurs possibles de `properties` de `RangeFilter`

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

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

```
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(
 *     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)

```
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(
 *     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 :

`/api/personnes?adresses=49`

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

```
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(
 *     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

```
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(
 *     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 :

`/api/personnes?adresses.ville=mars`

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 utilisant les web services définis par **Symfony**.

JWT : JSON Web Token

- 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"  
}
```

© Achref EL MOU

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

```
HMACHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),  
    secret)
```

© Achref EL MOUELHI ©

Symfony

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

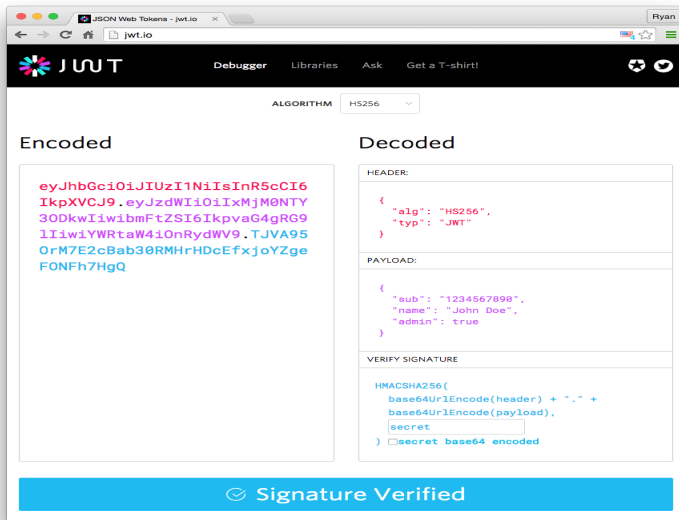
```
HMACSHA256 (  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload) ,  
    secret)
```

Résultat

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4  
gRG91IiwiaXNTb2NpYWwiOnRydWV9.  
4pcPyMD09o1PSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Symfony

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



The screenshot shows the JWT.io debugger interface in a web browser. The browser tab is titled "JSON Web Tokens - jwt.io". The address bar shows "jwt.io". The page has a dark header with the JWT logo and navigation links: "Debugger", "Libraries", "Ask", and "Get a T-shirt!". A user profile icon for "Ryan" is in the top right. Below the header, there's a section for "ALGORITHM" with a dropdown menu set to "HS256". The main content is split into two columns: "Encoded" and "Decoded".

Encoded:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0Ij0iOnRydwV9.TJVA95OrM7E2cBab30RMhRHDcEfxjoYZgeFONFh7HgQ
```

Decoded:

HEADER:

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

PAYLOAD:

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

VERIFY SIGNATURE

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

Below the decoded section, there is a checkbox labeled "secret base64 encoded" which is currently unchecked.

At the bottom of the interface, a large blue button with a checkmark icon and the text "Signature Verified" is displayed.

Symfony

Plusieurs 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

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`
- et ensuite la commande `php bin/console doctrine:migrations:migrate`

© Achref EL MOUËZ

Symfony

Pour créer la table `User`

- exécutez la commande `php bin/console make:migration`
- et ensuite la commande `php bin/console doctrine:migrations:migrate`

Pour remplir la table `User` avec des données aléatoires

- installez le bundle de fixture `composer require --dev doctrine/doctrine-fixtures-bundle`
- demandez à ce bundle de créer une classe fixtures `php bin/console make:fixtures`
- répondez à The class name of the fixtures to create par `UserFixtures`

Symfony

Contenu généré pour UserFixtures

```
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();
    }
}
```

Nouveau contenu de UserFixtures

```
class UserFixtures extends Fixture
{
    private $passwordEncoder;

    public function __construct(UserPasswordEncoderInterface $passwordEncoder)
    {
        $this->passwordEncoder = $passwordEncoder;
    }
    public function load(ObjectManager $manager)
    {
        $user = new User();
        $user->setEmail('john@wick.us');
        $user->setNom('wick');
        $user->setPassword('john');
        $user->setRoles(['ROLE_ADMIN']);
        $user->setPassword($this->passwordEncoder->encodePassword($user, 'wick'));
        $manager->persist($user);
        $user2 = new User();
        $user2->setEmail('jack@dalton.us');
        $user2->setNom('dalton');
        $user2->setPassword('jack');
        $user2->setPassword($this->passwordEncoder->encodePassword($user2, 'dalton'));
        $manager->persist($user2);
        $manager->flush();
    }
}
```

Nouveau contenu de UserFixtures

```

class UserFixtures extends Fixture
{
    private $passwordEncoder;

    public function __construct(UserPasswordEncoderInterface $passwordEncoder)
    {
        $this->passwordEncoder = $passwordEncoder;
    }
    public function load(ObjectManager $manager)
    {
        $user = new User();
        $user->setEmail('john@wick.us');
        $user->setNom('wick');
        $user->setPassword('john');
        $user->setRoles(['ROLE_ADMIN']);
        $user->setPassword($this->passwordEncoder->encodePassword($user, 'wick'));
        $manager->persist($user);
        $user2 = new User();
        $user2->setEmail('jack@dalton.us');
        $user2->setNom('dalton');
        $user2->setPassword('jack');
        $user2->setPassword($this->passwordEncoder->encodePassword($user2, 'dalton'));
        $manager->persist($user2);
        $manager->flush();
    }
}

```

Les use nécessaires

```

use App\Entity\User;
use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;

```

Symfony

Pour insérer les utilisateurs dans la base de données, exécutez

```
php bin/console doctrine:fixtures:load ou php  
bin/console d:f:l
```

Intégrons le bundle **JWT** dans notre projet

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

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

© Achrel

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 la section `firewalls` de `security.yaml`

```

security:

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        api:
            pattern: ^/ws/
            stateless: true
            anonymous: true
            provider: app_user_provider
            guard:
                authenticators:
                    - lexik_jwt_authentication.jwt_token_authenticator
    main:
        anonymous: true
        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
        guard:
            authenticators:
                - lexik_jwt_authentication.jwt_token_authenticator
    access_control:

        - { path: ^/ws/docs, roles: IS_AUTHENTICATED_ANONYMOUSLY } # Pour autoriser l'accès à Swagger UI
        - { path: ^/authentication_token, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/, roles: IS_AUTHENTICATED_FULLY }

```

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

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

Symfony

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

```
namespace App\Entity;

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

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *     attributes={"security"="is_granted('ROLE_ADMIN')"},
 *     itemOperations={"get", "put", "delete"},
 *     normalizationContext={"groups"={"personne:read"}},
 *     denormalizationContext={"groups"={"personne:write"}}
 * )
 */
class Personne
```

Jersey

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": "wick@wick.us",  
  "password": "wick"  
}
```

- Cliquer sur **Send** puis copier le jeton

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/ws/personnes`
- Dans le Headers, saisir **Content-Type** comme Key et `application/json` comme Value
- Dans Authorization, cliquer sur Type et choisir **raw**, 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)
 * @ApiResponse(
 *     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"}}
 * )
 */
class Personne
```

Symfony

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

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResponse(
 *     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"}}
 * )
 */
class Personne
```

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