Symfony : API REST

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

Service web (ou **WS** pour **W**eb **S**ervice) ?

- Programme (ensemble de fonctionnalités exposées en temps réel et sans intervention humaine)
- Accessible via un réseau internet, ou intranet
- Indépendant de tout système d'exploitation
- Indépendant de tout langage de programmation
- Utilisant un système standard d'échange (**XML** ou **JSON**), ces messages sont généralement transportés par des protocoles internet connus **HTTP** (ou autres comme **FTP**, **SMTP**...)
- Pouvant communiquer avec d'autres WS

Symfony

Les WS peuvent utiliser les technologies web suivantes :

- **HTTP** (Hypertext Transfer Protocol) : le protocole, connu, utilisé par le World Wide Web et inventé par Roy Fiedling.
- **REST** (Representational State Transfer) : une architecture de services Web, créée aussi par Roy Fielding en 2000 dans sa thèse de doctorat.
- **SOAP** (Simple object Access Protocol) : un protocole, défini par Microsoft et IBM ensuite standarisé par **W3C**, permettant la transmission de messages entre objets distants (physiquement distribués).
- **WSDL** (Web Services Description Language) : est un langage de description de service web utilisant le format **XML** (standardisé par le **W3C** depuis 2007).
- **UDDI** (Universal Description, Discovery and Integration) : un annuaire de WS.

HTTP, **REST** (Representational State Transfer) et **RESTful** ?

- Les API REST sont basées sur le protocole **HTTP** (architecture client/serveur) et utilisent le concept de ressource.
- Une ressource est identifiée par une URI unique.
- L'API REST utilise donc des méthodes suivantes pour l'échange de données entre client et serveur
  - `GET` pour la récupération,
  - `POST` pour l'ajout,
  - `DELETE` pour la suppression,
  - `PUT` pour la modification,
  - ...
- Plusieurs formats possibles pour les données échangées : texte, **XML**, **JSON**...
- **RESTful** est l'adjectif qui désigne une API REST.

Symfony

Rôle du contrôleur dans une application **MVC**

- Le contrôleur reçoit une requête **HTTP** et communique avec modèle, service, constructeur de formulaires pour retourner une réponse **HTTP** contenant une page **HTML**.
- Le contrôleur peut aussi retourner une réponse **HTTP** ne contenant pas de vue.
- Il retourne des données sous format **JSON**, **XML**...

Symfony

Rôle du contrôleur dans une application **MVC**

- Le contrôleur reçoit une requête **HTTP** et communique avec modèle, service, constructeur de formulaires pour retourner une réponse **HTTP** contenant une page **HTML**.
- Le contrôleur peut aussi retourner une réponse **HTTP** ne contenant pas de vue.
- Il retourne des données sous format **JSON**, **XML**...

Ceci est l'objet de ce chapitre.

Symfony

Étapes à suivre avant de commencer le cours

1. Allez à `https://github.com/Achreftun/symfony_rest.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

Utilisation de **Postman** : simulateur d'un client REST

- Aller sur internet et chercher **Postman**
- Télécharger puis installer l'application **Postman**
- Démarrer l'application

Il existe plusieurs autres tel que ARC (pour Advanced REST Client)...

Symfony

Créons un contrôleur `ApiPersonneController`

```
php bin/console make:controller ApiPersonne
```

Symfony

Créons un contrôleur `ApiPersonneController`

```
php bin/console make:controller ApiPersonne
```

Le résultat est

```
created: src/Controller/ApiPersonneController.php
```

Symfony

Contenu généré de `ApiPersonneController`

```php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\Routing\Annotation\Route;

class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne')]
    public function index(): JsonResponse
    {
        return $this->json([
            'message' => 'Welcome to your new controller!',
            'path' => 'src/Controller/ApiPersonneController.php',
        ]);
    }
}
```

Pour retourner les tuples de la table `Personne` sous format `JSON`

```
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne')]
    public function index(PersonneRepository $personneRepository)
    {
        $personnes = $personneRepository->findAll();
        $json = json_encode($personnes);
        return $json;
    }
}
```

Pour retourner les tuples de la table `Personne` sous format `JSON`

```
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne')]
    public function index(PersonneRepository $personneRepository)
    {
        $personnes = $personneRepository->findAll();
        $json = json_encode($personnes);
        return $json;
    }
}
```

Erreur

```
The controller must return a Symfony\Component\HttpFoundation\Response
object but it returned a string
```

Pour retourner une réponse HTTP contenant les tuples sous format `JSON`

```php
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne')]
    public function index(PersonneRepository $personneRepository)
    {
        $personnes = $personneRepository->findAll();
        $json = json_encode($personnes);
        $reponse = new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
        return $reponse;
    }
}
```

Pour retourner une réponse HTTP contenant les tuples sous format `JSON`

```php
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne')]
    public function index(PersonneRepository $personneRepository)
    {
        $personnes = $personneRepository->findAll();
        $json = json_encode($personnes);
        $reponse = new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
        return $reponse;
    }
}
```

Objets JSON vides car les attributs sont privés.

Symfony

Solution ⇒ normaliser ou serializer les objets. Pour le faire

- Installez le bundle de serialisation
- Configurez `config/services.yaml`

Symfony

Solution ⇒ normaliser ou serializer les objets. Pour le faire

- Installez le bundle de serialisation
- Configurez `config/services.yaml`

Pour installer le bundle de serialisation

```
composer require symfony/serializer
```

Symfony

Solution ⇒ normaliser ou serializer les objets. Pour le faire

- Installez le bundle de serialisation
- Configurez `config/services.yaml`

Pour installer le bundle de serialisation

```
composer require symfony/serializer
```

Dans `services.yaml`, ajoutez le code suivant pour qu'on puisse utiliser les getters/setters pour accéder aux attributs privés

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

Symfony

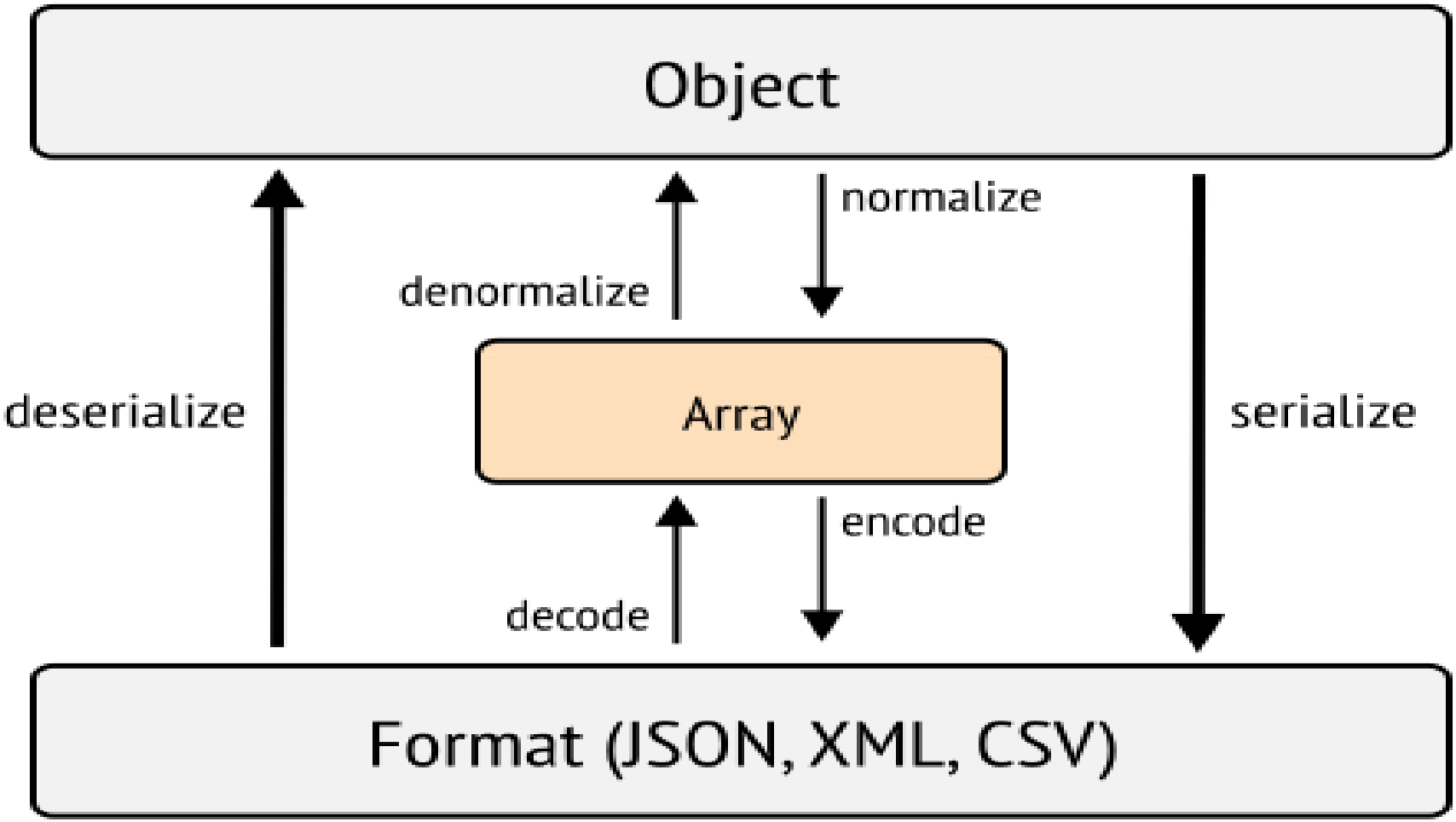


Image de la documentation officielle de Symfony

Pour retourner une réponse HTTP contenant les tuples sous format JSON

```
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\Normalizer\NormalizerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne')]
    public function index(PersonneRepository $personneRepository,
       NormalizerInterface $normalizer)
    {
        $personnes = $personneRepository->findAll();
        $normalized = $normalizer->normalize($personnes);
        $json = json_encode($normalized);
        return new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
    }
}
```

Symfony

Remarque

Pour un meilleur affichage de données au format **JSON** avec **Google Chrome**, installez l'extension **Json Formatter**.

Extrait du résultat retourné


```
[
    {
        "id": 1,
        "nom": "Morin",
        "prenom": "étienne",
        "adresses": [
            {
                "id": 1,
                "rue": "chemin de Langlois",
                "codePostal": "27588",
                "ville": "Fabre-sur-Marechal"
            },
            {
                "id": 2,
                "rue": "avenue Laurent Pons",
                "codePostal": "47827",
                "ville": "Leleu-la-Forêt"
            }
        ]
    },
    {
        "id": 2,
        "nom": "Boyer",
        "prenom": "Marc",
        "adresses": [
            {
                "id": 3,
                "rue": "place Laurent Buisson",
                "codePostal": "61 153",
                "ville": "Barthelemy"
            }
        ]
    },
]
```

Symfony

Transformons l'association entre `Personne` et `Adresse` en bidirectionnelle

- Supprimons tout ce qui concerne `Adresse` dans `Personne`
- Exécutons `php bin/console make:entity Personne` et rajoutons l'attribut `adresses` avec une relation bidirectionnelle `ManyToMany`

Symfony

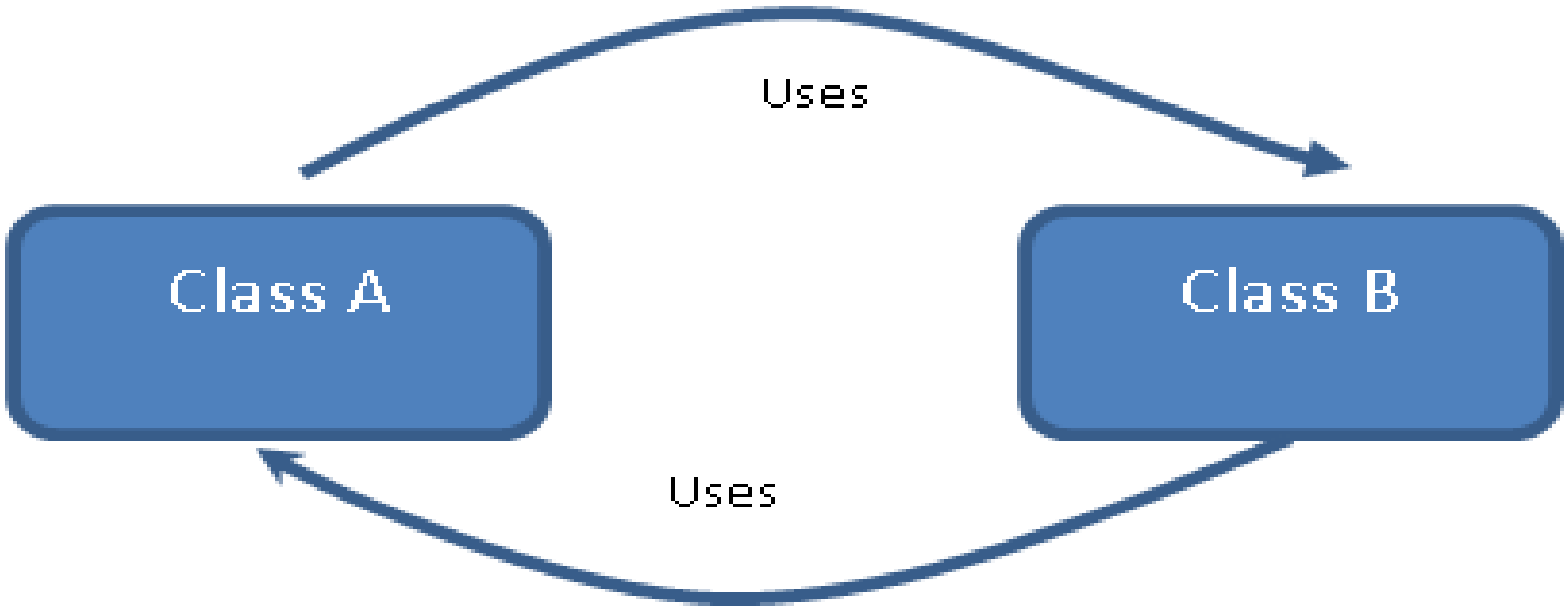
En renvoyant la requête **HTTP** précédente depuis **POSTMAN**

```
A circular reference has been detected when
serializing the object of class
\"App\\Entity\\Personne\"
```

Symfony

En renvoyant la requête **HTTP** précédente depuis **POSTMAN**

```
A circular reference has been detected when
serializing the object of class
\"App\\Entity\\Personne\"
```



Pour résoudre ce problème, une première solution consiste à utiliser l'attribut `#[Ignore]` dans la classe `Adresse`

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\AdresseRepository;
use Doctrine\Common\Collections\Collection;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Attribute\Ignore;

#[ORM\Entity(repositoryClass: AdresseRepository::class)]
class Adresse
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    private $rue;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    private $codePostal;

    #[ORM\Column(type: 'string', length: 30, nullable: true)]
    private $ville;

    #[Ignore]
    #[ORM\ManyToMany(targetEntity: Personne::class, mappedBy: 'adresses')]
    private $personnes;

    // ...

}
```

Rien à changer dans le contrôleur

```
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\Normalizer\NormalizerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne')]
    public function index(PersonneRepository $personneRepository,
       NormalizerInterface $normalizer)
    {
        $personnes = $personneRepository->findAll();
        $normalized = $normalizer->normalize($personnes);
        $json = json_encode($normalized);
        return new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
    }
}
```

Extrait du résultat retourné


```
[
    {
        "id": 1,
        "nom": "Morin",
        "prenom": "étienne",
        "adresses": [
            {
                "id": 1,
                "rue": "chemin de Langlois",
                "codePostal": "27588",
                "ville": "Fabre-sur-Marechal"
            },
            {
                "id": 2,
                "rue": "avenue Laurent Pons",
                "codePostal": "47827",
                "ville": "Leleu-la-Forêt"
            }
        ]
    },
    {
        "id": 2,
        "nom": "Boyer",
        "prenom": "Marc",
        "adresses": [
            {
                "id": 3,
                "rue": "place Laurent Buisson",
                "codePostal": "61 153",
                "ville": "Barthelemy"
            }
        ]
    },
]
```

Une deuxième solution consiste à créer des groupes dans l'entité `Personne`

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\Collection;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Attribute\Groups;

#[ORM\Entity(repositoryClass: PersonneRepository::class)]
class Personne
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    #[Groups("personne:read")]
    private $id;

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

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

    #[ORM\ManyToMany(targetEntity: Adresse::class, inversedBy: 'personnes', cascade: ['persist'
       , 'remove'])]
    private $adresses;

    // ...

}
```

Symfony

En utilisant la méthode `normalize`, on indique le·s groupe·s à inclure dans la réponse

```
class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne')]
    public function index(PersonneRepository $personneRepository,
       NormalizerInterface $normalizer)
    {
        $personnes = $personneRepository->findAll();
        $normalized = $normalizer->normalize($personnes, null, [
            'groups' => 'personne:read'
        ]);

        $json = json_encode($normalized);

        return new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
    }
}
```

Symfony

Et si on voulait retourner pour chaque objet `Personne` **sa liste** `adresses`**, on ajoute le groupe** `personne:read` **à** `adresses` **dans** `Personne`

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

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

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

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

    // ...

}
```

Symfony

Et dans `Adresse`, on ajoute `personne:read` pour tous les attributs sauf `personnes`

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

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

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

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

    #[ORM\ManyToMany(targetEntity: Personne::class, mappedBy: 'adresses')]
    private $personnes;

    // ...

}
```

Symfony

On peut aussi utiliser l'interface `SerializerInterface` qui normalise et encode en JSON

```
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\SerializerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne')]
    public function index(PersonneRepository $personneRepository, SerializerInterface
       $serializer)
    {
        $personnes = $personneRepository->findAll();
        $json = $serializer->serialize($personnes, 'json', [
            'groups' => 'personne:read'
        ]);
        return new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
    }
}
```

Symfony

Exercice

- Créez un contrôleur `ApiAdresseController`,
- Modifiez l'action `index` pour qu'elle retourne la liste d'adresses : chacune avec un tableau de ses propriétaires (personnes).

Symfony

Dans `Adresse`, ajoutons le groupe `adresse:read` pour tous les attributs tout en préservant le groupe `personne:read`

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

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

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

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

    #[Groups("adresse:read")]
    #[ORM\ManyToMany(targetEntity: Personne::class, mappedBy: 'adresses')]
    private Collection $personnes;

    // ...

}
```

Symfony

Ajoutons le groupe `adresse:read` **dans** `Personne` **pour tous les attributs sauf** `adresses`

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

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

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

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

    // ...

}
```

Symfony

Exercice

- Créer un contrôleur `ApiAdresse`
- Vérifier que l'action `index` est accessible via la route `/api/adresse`
- Modifier l'action `index` pour qu'elle retourne la liste d'adresses : chaque adresse doit contenir la liste des personnes qui lui sont associées.

Symfony

Créons un contrôleur `ApiAdresseController`

```
php bin/console make:controller ApiAdresse
```

Symfony

Créons un contrôleur `ApiAdresseController`

```
php bin/console make:controller ApiAdresse
```

Le résultat est

```
created: src/Controller/ApiAdresseController.php
```

Symfony

Contenu de `ApiAdresseController`

```php
namespace App\Controller;

use App\Repository\AdresseRepository;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Component\Serializer\SerializerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ApiAdresseController extends AbstractController
{
    #[Route('/api/adresse', name: 'app_api_adresse')]
    public function index(AdresseRepository $rep, SerializerInterface $serializer)
    {
        $adresses = $rep->findAll();

        $json = $serializer->serialize($adresses, 'json', [
            'groups' => 'adresse:read'
        ]);
        $reponse = new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
        return $reponse;
    }
}
```

Symfony

Pour la réponse, on peut simplifier le code en utilisant un objet `JsonResponse`

```
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\Serializer\SerializerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne', methods: ['GET'])]
    public function index(PersonneRepository $personneRepository, SerializerInterface
       $serializer)
    {
        $personnes = $personneRepository->findAll();

        $json = $serializer->serialize($personnes, 'json', [
            'groups' => 'personne:read'
        ]);

        return new JsonResponse($json, 200,  [], true);
    }
}
```

Symfony

On peut aussi simplifier l'écriture en utilisant la méthode `json` de `AbstractController` qui retourne un objet `JsonResponse`

```php
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\SerializerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ApiPersonneController extends AbstractController
{
    #[Route('/api/personne', name: 'app_api_personne', methods: ['GET'])]
    public function index(PersonneRepository $personneRepository)
    {
        $personnes = $personneRepository->findAll();

        return $this->json($personnes, 200,  [], [
            'groups' => 'personne:read'
        ]);
    }
}
```

Symfony

Pour ajouter une nouvelle personne

```
#[Route('/api/personne', name: 'app_api_personne_add', methods: ['POST'])]
public function add(EntityManagerInterface $entityManager, Request $request,
   SerializerInterface $serializer, ValidatorInterface $validator)
{

    $contenu = $request->getContent();
    try {
        $personne = $serializer->deserialize($contenu, Personne::class, 'json');
        $errors = $validator->validate($personne);
        if (count($errors) > 0) {
            return $this->json($errors, 400);
        }

        $entityManager->persist($personne);
        $entityManager->flush();
        return $this->json($personne, 201, [], [
            'groups' => 'personne:read'
        ]);
    } catch (NotEncodableValueException $e) {
        return $this->json([
            'status' => 400,
            'message' => $e->getMessage()
        ]);
    }
}
```

Symfony

Les `use` nécessaires

```
use App\Entity\Personne;
use App\Repository\PersonneRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\SerializerInterface;
use Symfony\Component\Validator\Validator\ValidatorInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Serializer\Exception\NotEncodableValueException;
```

Symfony

Les `use` nécessaires

```
use App\Entity\Personne;
use App\Repository\PersonneRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\SerializerInterface;
use Symfony\Component\Validator\Validator\ValidatorInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Serializer\Exception\NotEncodableValueException;
```


Pensez à installer le `Validator`

```
composer require symfony/validator
```

Symfony

Pour tester avec **Postman**

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

```
{
    "nom": "Morena",
    "prenom": "Andreas"
}
```

- Cliquer sur `Send`

Symfony

En cas de problème avec `Property-access`, installez le

```
composer require symfony/property-access
```

Symfony

En essayant d'ajouter une nouvelle personne avec deux adresses, les adresses ne seront pas insérées (ni associées à la personne)

```
{
    "nom": "Cage",
    "prenom": "Nicolas",
    "adresses": [
        {
            "rue": "chemin de Gibbes",
            "codePostal": "13014",
            "ville": "Marseille"
        },
        {
            "rue": "rue des plantes",
            "codePostal": "75014",
            "ville": "Paris"
        }
    ]
}
```

Symfony

Pour résoudre le problème précédent

Ajouter un setter à l'attribut `adresses` dans la classe `Personne`

Symfony

Pour résoudre le problème précédent

Ajouter un setter à l'attribut `adresses` dans la classe `Personne`

Exemple de `setter` pour `adresses` à ajouter dans `Personne`

```
public function setAdresses(array $adresses): self
{
    $this->adresses = $adresses;
    return $this;
}
```

Symfony

En renvoyant la requête `POST` précédente avec Postman

```
{
    "nom": "Cage",
    "prenom": "Nicolas",
    "adresses": [
        {
            "rue": "chemin de Gibbes",
            "codePostal": "13014",
            "ville": "Marseille"
        },
        {
            "rue": "rue des plantes",
            "codePostal": "75014",
            "ville": "Paris"
        }
    ]
}
```

Le résultat est

```
{
    "id": 11,
    "nom": "Cage",
    "prenom": "Nicolas",
    "adresses": [
        {
            "id": 12,
            "rue": "chemin de Gibbes",
            "codePostal": "13014",
            "ville": "Marseille"
        },
        {
            "id": 13,
            "rue": "rue des plantes",
            "codePostal": "75014",
            "ville": "Paris"
        }
    ]
}
```

Symfony

Exercice

Créez puis testez, avec **Postman**, les méthodes **HTTP**

- `PUT` (Route : `/personne/{id}`) qui permettra de modifier une personne selon son identifiant
- `DELETE` (Route : `/personne/{id}`) qui permettra de supprimer une personne selon son identifiant
- `GET` (Route : `/personne/{id}`) qui permettra de retourner une personne selon son identifiant
- `GET` (Route : `/personne/{id}/adresse`) qui permettra de retourner les adresses d'une personne selon son identifiant
- `GET` (Route : `/personne/{idPersonne}/adresse/{idAdresse}`) qui permettra de retourner l'adresse d'une personne selon leurs identifiants

Symfony

Correction de DELETE

```php
#[Route('/api/personne/{id}', name: 'app_api_personne_delete', methods: ['DELETE'])]
public function delete(int $id, EntityManagerInterface $em)
{
    $personne = $em->getRepository(Personne::class)->find($id);
    if ($personne) {
        $em->remove($personne);
        $em->flush();
        return $this->json([
            'status' => 204,
            'message' => "Personne avec l'identifiant $id supprimée avec succès"
        ]);
    }
    return $this->json([
        'status' => 404,
        'message' => "Aucune correspondance avec l'identifiant $id"
    ]);
}
```

Correction de PUT

```php
#[Route('/api/personne/{id}', name: 'app_api_personne_put', methods: ['PUT'])]
public function edit(int $id, EntityManagerInterface $em, SerializerInterface
   $serializer, Request $request)
{
    $data = $request->getContent();
    try {
        $personne = $serializer->deserialize($data, Personne::class, 'json');
        if ($id != $personne->getId()) {
            return $this->json([
                'status' => 400,
                'message' => "Incohérence : l'identifiant $id ne correspond pas à l'
                  identifiant reçu dans le body"
            ]);
        }
        $p = $em->getRepository(Personne::class)->find($id);
        if (!$p) {
            return $this->json([
                'status' => 404,
                'message' => "Aucune correspondance avec l'identifiant $id"
            ]);
        }
        $p = $personne;
        $em->flush();
        return $this->json($p, 202, [], ['groups' => 'personne:read']);
    } catch (NotEncodableValueException $e) {
        return $this->json([
            'status' => 400,
            'message' => $e->getMessage()
        ]);
    }
}
```

Symfony

Correction de GET (item)

```
#[Route('/api/personne/{id}', name: 'app_api_personne_show', methods: ['GET'])]
public function show(int $id, EntityManagerInterface $em)
{
    $personne = $em->getRepository(Personne::class)->find($id);
    if ($personne) {
        return $this->json($personne, 200, [], ['groups' => 'personne:read']);
    }
    return $this->json([
        'status' => 404,
        'message' => "Aucune correspondance avec l'identifiant $id"
    ]);
}
```

Symfony

Correction de `GET` (adresses d'une personne)

```php
#[Route('/api/personne/{id}/adresse', name: 'app_api_personne_show_adresse', methods: [
   'GET'])]
public function showAdresses(int $id, EntityManagerInterface $em)
{
    $personne = $em->getRepository(Personne::class)->find($id);
    if ($personne) {
        return $this->json($personne->getAdresses(), 200, [], ['groups' => 'personne:
           read']);
    }
    return $this->json([
        'status' => 404,
        'message' => "Aucune correspondance avec l'identifiant $id"
    ]);
}
```

Correction de `GET` (adresse d'une personne)

```
#[Route('/api/personne/{idPersonne}/adresse/{idAdresse}', name: '
   app_api_personne_show_adresse_id', methods: ['GET'])]
public function showAdresseById(int $idPersonne, int $idAdresse, EntityManagerInterface
    $em)
{
    $personne = $em->getRepository(Personne::class)->find($idPersonne);
    if (!$personne) {
        return $this->json([
            'status' => 404,
            'message' => "Aucune personne avec l'identifiant $idPersonne"
        ]);
    }
    $adresse = $em->getRepository(Adresse::class)->find($idAdresse);
    if (!$adresse) {
        return $this->json([
            'status' => 404,
            'message' => "Aucune adresse avec l'identifiant $idAdresse"
        ]);
    }
    $ids = array_map(fn ($adr) => $adr->getId(), $personne->getAdresses()->toArray());
    if (in_array($idAdresse, $ids)) {
        return $this->json($adresse, 200, [], ['groups' => 'adresse:read']);
    }
    return $this->json([
        'status' => 404,
        'message' => "L'adresse avec l'identifiant $idAdresse n'appartient pas à la
          personne avec l'identifiant $idPersonne"
    ]);
}
```