esprit Se former autrement	Année Universitaire : 2020-2021 **Workshops Framework Symfony4** **UP-Web**

Workshop : JSON

I- Introduction

L'objectif de ce workshop est de manipuler JSON côté serveur dans une application symfony4

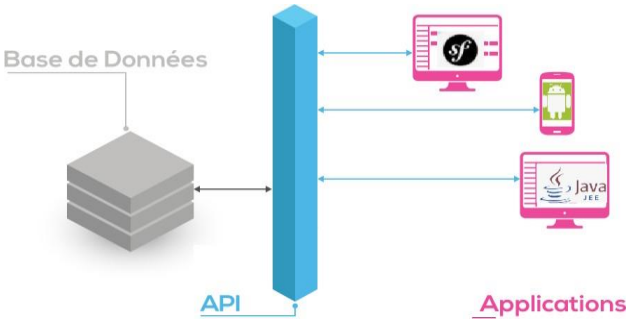

Figure 1: Consommation d'une API par les trois parties Java, Symfony et mobile

Le principe de sérialisation

- JSON est un format de données léger, facile à lire et à écrire et compatible avec pas mal de langages de développement.

- Le framework PHP Symfony nous offre un composant **Serializer** pour sérialiser les objets en différents formats (json, xml ...).

- On peut utiliser la sérialisation dans une API, des microservices, services et pour la récupération d'ob jets depuis la base de données.

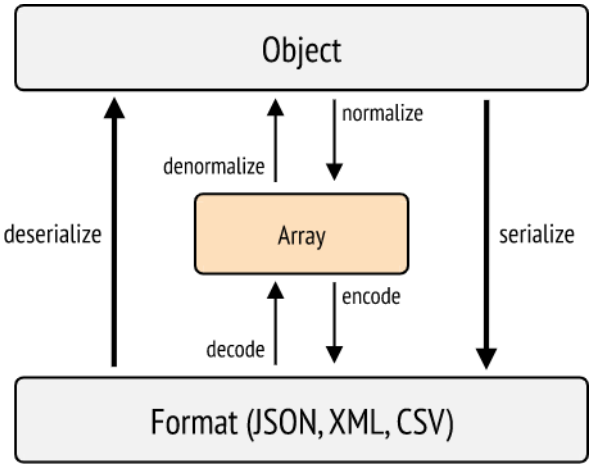

Figure 2: La sérialisation des objets

- D'après la figure ci-dessus, le « **serializer** » permet de :
  - ✔ Transformer des objets en un array à travers « **normalize** ».
  - ✔ Transformer ces tableaux en format JSON ou XML à travers « **encode** ».

- Le « **deserializer** » fait l'inverse, il décode le format en un tableau avant de letransformer en un objet.

II- Etude de cas :

Soit un contrôleur « **ProductController** » et une entité appelé « Product » comme suit :

Product
id : number, clé primaire
title : string
quantity : number
price : number

II.1- Initialisation et Rappel

II.1.1- Création de projet

Pour créer un projet symfony4 appelé **mySMFProject, deux façons sont possibles.**

a- Website skeleton

composer create-project symfony/website-skeleton mySMFProject "4.4.*"

⇨ cette commande vous permet de créer un projet contenant le nécessaire pour créer un website avec symfony. Il contient les bundles utiles.

b- Skeleton

composer create-project symfony/skeleton mySMFProject "4.4.*"

⇨ Cette commande permet de créer la squelette d'un projet symfony dédié à la partie backend contenant uniquement le bundle « FrameworkBundle »
Dans ce cas, il faut installer les bundles utiles dans votre projet à chaque fois où vous avez besoin d'un bundle bien déterminé

composer require symfony/web-server-bundle
composer require makerBundle
composer require symfony/orm-pack
composer require symfony/serializer
composer require symfony/property-access
composer require symfony/validator
composer require symfony/form

A chaque fois où vous avez besoin d'une dépendance, il faut l'installer

II.1.2- Rappel

Si vous voulez utilisez le serveur web Apache alors :

- Créez votre projet sous www (avec wamp) ou bien htdocs (avec xampp)
- Démarrez par la suite votre serveur Apache
- Lancez votre application en tapant l'url : localhost/mySMFProject/public/index.php.

Si vous allez utiliser le serveur de symfony alors :

- Créez votre projet n'importe où vous voulez.
- Installez le serveur de symfony : pointez sur votre projet créé et lancez la commande : **composer require symfony/web-server-bundle**
- Lancez votre serveur symfony avec la commande
  ⇨ **php bin/console server:run**
- Lancez votre application en tapant **localhost :8000**
- Créez un contrôleur « **ProductController** » en tapant la commande :
  ⇨ **php bin/console make:controller Product**
- Configurer une base de données « esprit » dans le fichier .env :

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/esprit"
```

- Générer la base de données « esprit » avec la commande :
  ⇨ **php bin/console doctrine:database:create**
- Créez l'entité « Product »
  ⇨ **php bin/console make:entity**
- Créez le fichier de migrations
  ⇨ **php bin/console make:migration**
- Générer la table « product »
  ⇨ **php bin/console doctrine:migrations:migrate**

II.2- CRUD d'une entité avec le format JSON

II.2.1. Read : Récupérer la liste des produits

```
/**
 * @Route("/api/listproducts", name="listproducts")
 */
public function getAllProducts(SerializerInterface $serializer): Response
{
    $listp=$this->getDoctrine()->getRepository(Product::class)->findAll();
    $jsonContent = $serializer->serialize($listp,"json");
    return new Response($jsonContent);

}
```

- N'oubliez pas d'ajouter les use des classes utilisées

```
use Symfony\Component\Serializer\SerializerInterface;
```

⇨ http://localhost:8000/api/listproducts, pour afficher la liste des étudiants dans une page HTML

II.2.2. Récupération d'un produit selon l'id :

Le code ci-dessous permet de récupérer un seul produit selon son id :

- Avec path param

```
//http://localhost:8000/product/1
/**
 * @Route("/api/product/{id}", name="product")
 */
public function getProduct($id,SerializerInterface $serializer): Response
{
    $product=$this->getDoctrine()->getRepository(Product::class)->find($id);
    $jsonContent = $serializer->serialize($product,"json");
    return new Response($jsonContent);
}
```

- Avec query param :

```
//http://localhost:8000/product?id=1
/**
 * @Route("/api/product2", name="product2")
 */
public function getProduct2(Request $request,SerializerInterface
$serializer): Response
{
    $product=$this->getDoctrine()->getRepository(Product::class)-
>find($request->get('id'));
    $jsonContent = $serializer->serialize($product,"json");
    return new Response($jsonContent);
}
```

II.2.3. Ajout d'un produit :

Le code ci-dessous permet d'ajouter un produit

```php
/**
 * @Route("/api/addProduct", name="addProduct")
 */
public function addProduct(Request $request,SerializerInterface $serializer) :
Response {
    //récupérer le contenu de la requête envoyé
    $data=$request->getContent();
    $product = $serializer->deserialize($data, Product::class, 'json');
    $em=$this->getDoctrine()->getManager();
    $em->persist($product);
    $em->flush();
    $jsonContent = $serializer->serialize($product,"json");
    return new Response($jsonContent);


}
```

- Pour le test, appelez l'url suivant : http://localhost:8000/api/addProduct

Pour faire les contrôles de saisies côté serveur en utilisant les asserts vous pouvez utiliser symfony/form

```php
/**
 * @Route("/api/add2Product", name="add2Product")
 */
public function add2Product(Request $request,SerializerInterface $serializer) :
Response {
    $product = new Product();
    $form=$this->createForm(ProductType::class,$product);
    //pour utiliser json_decode il faut ajouter "ext-json": "*" dans composer.json
    $data=json_decode($request->getContent(),true);
    $form->submit($data);
    if($form->isSubmitted() && $form->isValid()) {
        $em = $this->getDoctrine()->getManager();
        $em->persist($product);
        $em->flush();
        $jsonContent = $serializer->serialize($product, "json");
      //  return new Response(Response::HTTP_CREATED);
        return new Response($jsonContent);
    }
    //return new Response($form->getErrors());
    return new Response($serializer->serialize($form->getErrors(),"json"));
}
```

- Pour le test, appelez l'url suivant : http://localhost:8000/api/add2Product

ProductType :

```php
<?php

namespace App\Form;

use App\Entity\Product;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
```

```php
class ProductType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title')
            ->add('quantity')
            ->add('price')
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Product::class,
      //csrf_ptection est important pour pouvoir valider le formulaire sinon form is invalid sinon il faut envoyer ce token depuis frontend
            'csrf_protection'=>false         ]);
    }
}
```

Annexes : Consommation d'api developpé avec symfony depuis Angular

Lors de la consommation d'api avec Angular, le problème rencontré est celui de CORS : Cross-Origin Resource Sharing.
CORS est un protocole permettant aux scripts s'exécutant coté client d'interagir avec des ressources d'origine différent.

Pour remédier à ce problème, plusieurs solutions existantes.
Les deux solutions les plus pertinentes :

- Configurer le serveur de la partie back-end de façon que le problème de CORS soit résolu. (server-side solution)
- Configurer Angular CLI proxy (client-side solution)

1- **Configurer le serveur de la partie back-end de façon que le problème de CORS soit résolu. (server-side solution)**

   Si le backend est symfony alors pour résoudre ce problème vous pouvez installer le bundle externe NelmioCorsBundle (version 1.5 pour symfony 3.4, la version 2 est dédié à symfony >= 4)

   https://github.com/nelmio/NelmioCorsBundle

   https://www.youtube.com/watch?v=agMVgKrQ3Hk

NB : D'après la configuration proposé que ce soit dans le lien git ou d'après la vidéo, il faut que le path de chaque api soit précédé par un préfix /api.

2- **Configurer Angular CLI proxy (client-side solution)**

https://www.techiediaries.com/fix-cors-with-angular-cli-proxy-configuration/