Technologies avancées du web
Initiation à Symfony 4

Pierre Pompidor

Table des matières

Symfony 4 est un framework PHP permettant de créer des applications web. Il utilise un l'**ORM Doctrine** pour manipuler les tables/enregistrements/attributs d'une base de données sous la forme de classes/objets/attributs. Un projet Symfony 4 est géré par le gestionnaire de projets **Composer** (qui a également la charge de vérifier les dépendances du projet).

Ce mini-cours vous permettra de créer trois projets :

- le premier permettant d'afficher aléatoirement un entier (directement tiré de la documentation officielle) ;
- le second permettant d'accéder aux enregistrements d'une table mysql *membres* :
  - en listant tous les membres enregistrés ;
  - en listant les informations d'un membre identifié par son id ;
  - en affichant les informations d'un membre identifié par son adresse mail ;
  - en inscrivant de nouveaux membres.

1 Installations préalables (MySQL, PHP 7, Composer) :

1.1 Installation du SGBD MySQL sous Ubuntu :

Les informations suivantes correspondent à l'installation du SGBD MySQL sur votre ordinateur personnel géré par Ubuntu 18.04 (en tout cas >= 16.04).
En salle de TP, un serveur MySQl est déjà accessible.

Verifiez si un serveur MySQL tourne déjà sur votre machine :

```
systemctl status mysql
```

dans la négative, installez le serveur MySQL (et mettez-le en oeuvre en tant que service), ainsi que le client mysql (la commande exécutable sous un terminal).

```
sudo apt update
sudo apt install mysql-server mysql-client
systemctl start mysql
systemctl enable mysql
```

Verifiez si vous pouvez vous connecter sur le serveur en tant que root (et avec un mot de passe vide) mais sans être soi-même super-utilisateur (sans faire de sudo) :

```
mysql -u root -p
```

dans la négative, exécuter les commandes suivantes :

```
sudo mysql -u root -p

> use mysql;
> update user set plugin='mysql_native_password' where user='root';
> flush privileges;
> quit;
```

1.2 Installation de PHP 7 sous Ubuntu :

La dernière version de PHP en tant que serveur et client (à cette date la 7.2), ainsi que les deux modules permettant la gestion de fichiers XML et l'interfaçage de différents systèmes de gestion de bases de données (PDO), doivent être installés.
En salle de TP, PHP et ses modules nécessaires à Symfony sont déjà installés.

Si apache2 (le serveur web) n'est pas déjà installé, installez-le :

```
sudo apt install apache2
```

puis installez PHP et ses modules :

```
sudo apt install php-cli php-xml libapache2-mod-php
```

Remarque : vous pouvez spécifier une installation spécifique en faisant apparaître la version.

```
sudo apt install php7.2-cli php7.2-xml libapache2-mod-php7.2
```

Il se peut que le module PDO soit installé mais que son driver pour MySQL ne le soit pas (vérifiez en exécutant phpinfo.php que la ligne correspondant à ce module ne soit pas à *no-value* mais à *mysql*).
Dans la négative, installez ce driver et relancez Apache :

```
apt-get install php-mysql
systemctl restart apache2
```

1.3 Installation du gestionnaire de projets Composer :

La procédure d'installation de Composer est décrite ici :
`https://www.digitalocean.com/community/tutorials/how-to-install-and-use-composer-on-u`
Ce qui suit n'en reprend que les grandes lignes :

```
sudo apt install curl
cd ~
curl -sS https://getcomposer.org/installer -o composer-setup.php
sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
```

2 Création d'un projet Symfony 4 :

La création d'un projet Symfony 4 est faite par la commande suivante (dans ce qui suit *my-project* doit être remplacé par le nom de votre projet) :

```
composer create-project symfony/skeleton my-project
```

Allez dans le répertoire ainsi créé :

```
cd my-project
```

Voici les dossiers/fichiers importants dans le cadre de notre initiation à Symfony :

- **config/routes.yaml** : spécification des routes
- **src/Controller** : dossier dans lequel une classe va être créée par page HTML renvoyée (dans notre cas)
- **src/Entity** : dossier dans lequel une classe va être créée pour correspondre à une table de la base
- **.env** : spécification du paramétrage de l'accès à une base de données
- **templates** : templates externalisés

composer gère un serveur local de développement (par défaut sur le port 8000).
Installez ce serveur et lancez-le (il tournera en permanence lors de la mise au point de votre application) :

```
composer require symfony/web-server-bundle --dev
php bin/console server:run
```

L'exemple suivant est tiré de la documentation officielle :
`https://symfony.com/doc/current/page_creation.html`

Dans un autre onglet de votre terminal, modifiez le fichier **config/routes.yaml** pour créer la route `/lucky/number` :

```
app_lucky_number:
    path: /lucky/number
    controller: App\Controller\LuckyController::number
```

Créez la classe qui instanciée va répondre à cette route pour envoyer un flux HTML :

```
<?php
namespace App\Controller;
use Symfony\Component\HttpFoundation\Response;

class LuckyController
{
    public function number()
    {
        $number = random_int(0, 100);
        return new Response(
            '<html><body>Lucky number: '.$number.'</body></html>'
        );
    }
}
?>
```

Testez votre application en l'invocant dans l'URL de votre navigateur :

```
http://localhost:8000/lucky/number
```

3 Création d'un projet Symfony 4 interfaçant une base de données :

L'interfaçage d'une base de données passe par la création d'**entities** qui sont les classes gérées par l'ORM Doctrine pour "mapper" les tables de la base de données.
Ce qui suit est inspiré des documentations sises à ces URL :
`https://symfony.com/doc/current/doctrine.html`
`https://symfony.com/doc/current/doctrine/reverse_engineering.html`

Nous nous intéresserons ici qu'à l'exploitation d'une base de données déjà créée.

3.1 Installations préalables :

Les configurations suivantes pour *composer* sont nécessaires :

```
composer require symfony/orm-pack
composer require symfony/maker-bundle --dev
```

3.2 Base de données MySQL :

Nous considérons avoir à disposition la base *covoit* contenant la table *membres* :

```
CREATE DATABASE IF NOT EXISTS covoit;
USE covoit;
CREATE TABLE IF NOT EXISTS membres ( id smallint unsigned not null auto_increment,
                                     mail varchar(30) not null, nom varchar(30) not
                                     prenom varchar(20) not null, primary key (id));
INSERT INTO membres ( id, mail, nom, prenom )
                    VALUES (null, 'pompidor@lirmm.fr', 'Pompidor', 'Pierre');
INSERT INTO membres ( id, mail, nom, prenom )
                    VALUES (null, 'meynard@lirmm.fr', 'Meynard', 'Michel');
```

Modifiez la ligne *DATABASE_URL* dans le fichier *.env* comme suit :

sur votre ordinateur personnel

```
DATABASE_URL=mysql://root:@127.0.0.1:3306/covoit
```

et à la fac :

```
DATABASE_URL=mysql://<votreLogin>:<votrePassword>@prodpeda-venus:3306/<votreLogin>
```

Enfin faites produire l'**entity** correspondante à votre table :

```
php bin/console doctrine:mapping:import 'App\Entity' annotation --path=src/Entity
```

le fichier *src/Entity/Membres.php* est alors créé.

3.3 Liste de tous les membres de la table Membres :

Modifiez le fichier *src/Entity/Membres.php* pour créer un accesseur à l'attribut *nom* :

```
public function getNom() {
   return $this->nom;
}
```

puis créez le controller *MembresController* :

```
<?php
namespace App\Controller;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use App\Entity\Membres;

class MembresController extends AbstractController
{
    public function listeMembres() {
        $membres = $this->getDoctrine()
                   ->getRepository(Membres::class)
                   ->findAll();
        if (!$membres) {
            throw $this->createNotFoundException(
                        'Aucun membre trouvé !' );
        }
        $html = '<html><body> Liste des membres : <br/> <ul>';
        foreach ($membres as $membre) {
                $html .= '<li>'.$membre->getNom().'</li>';
        }
        $html .= '</ul></body></html>';
        return new Response($html);
    }
}
?>
```

Créez la route */membres* dans le fichier *config/routes.yaml* :

```
app_membres:
    path: /membres
    controller: App\Controller\MembresController::listeMembres
```

et enfin invoquez le controller sur l'URL suivante :

```
http://localhost:8000/membres
```

3.4 Liste des infos pour un membre désigné par son id / email :

Cet exemple illustre le passage d'un paramètre et l'utilisation de deux autres méthodes de recherche en plus de *findAll()* :

— **find()** : recherche par un identifiant
— **findBy()** : recherche par n'importe quelle combinaison d'attributs

Créez le controller *InfosMembreController* :

```
<?php
namespace App\Controller;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use App\Entity\Membres;

class InfosMembreController extends AbstractController {
   public function infosMembreParId($id) {
       $membre = $this->getDoctrine()
                      ->getRepository(Membres::class)
                      ->find($id);
        $html = '<html><body> Infos sur un membre : <br/> <ul>';
        $html .= '<li>'.$membre->getNom().'</li>';
        $html .= '</ul></body></html>';
        return new Response($html);
   }
    public function infosMembresParMail($mail) {
        $membres = $this->getDoctrine()
                   ->getRepository(Membres::class)
                   ->findBy(array('mail' => $mail));
        $html = '<html><body> Liste des membres : <br/> <ul>';
        foreach ($membres as $membre) {
                 $html .= '<li>'.$membre->getNom().'</li>'; }
        $html .= '</ul></body></html>';
        return new Response($html);
    }
}
?>
```

Créez les routes suivantes dans le fichier *config/routes.yaml* :

```
app_membre_id:
    path: /membre/{id}
    controller: App\Controller\InfosMembreController::infosMembreParId

app_membre_mail:
    path: /membre/mail/{mail}
    controller: App\Controller\InfosMembreController::infosMembresParMail
```

et enfin invoquez le controller sur les URLs suivantes :

```
http://localhost:8000/membre/1
http://localhost:8000/membre/mail/pompidor@lirmm.fr
```

3.5 Formulaire d'inscription et liste des membres mis à jour :

Les informations ci-dessous sont inspirées de :
`https://symfony.com/doc/current/forms.html`

3.5.1 Gestion d'un formulaire d'inscription :

Dans le projet précédent, installez les modules permettant de créer des formulaires et d'utiliser **twig** (le moteur de template) :

```
composer require symfony/form
composer require symfony/twig-bundle
```

Créez dans `src/Controller` le contrôleur *NouveauMembreController.php* générant le formulaire et la réception des données émises par celui-ci :

```php
<?php
namespace App\Controller;

use App\Entity\Membres;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

class NouveauMembreController extends AbstractController
{
    public function nouveauMembre(Request $request)
    {
        $membre = new Membres();

        $form = $this->createFormBuilder()
            ->add('mail', TextType::class, array('label' => 'Adresse email '))
            ->add('nom', TextType::class, array('label' => 'Nom '))
            ->add('prenom', TextType::class, array('label' => 'Prénom '))
            ->add('save', SubmitType::class, array('label' => 'Nouveau membre'))
            ->getForm();
//          ->add('DateNaissance', DateType::class, array('label' => 'Date de naissa
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
           $membre = $form->getData();

           $html = '<html><body> Nouveau membre : <br/> <ul>';
           foreach ($membre as $attribut) {
              $html .= '<li>'.$attribut.'</li>';
           }
           $html .= '</ul></body></html>';
           return new Response($html);
        }

        return $this->render('nouveauMembre.html.twig', array(
            'form' => $form->createView(),
        ));
    }
}
?>
```

Créez dans le dossier `templates` le **template** *nouveauMembre.html.twig* :

```
{{ form_start(form) }}
{{ form_widget(form) }}
{{ form_end(form) }}
```

Rajoutez la route suivante dans le fichier *config/routes.yaml* :

```
app_membre:
    path: /membre
    controller: App\Controller\NouveauMembreController::nouveauMembre
```

et enfin invoquez le controller sur :

```
http://localhost:8000/membre
```

3.6 Insertion dans la base de données :

Dans l'entité *Membres.php* spécifiez les attributs de la classe en *public*.
(Nous pourrions bien sûr mettre en place des *setters*.)

Passez l'objet *membre* en paramètre à la méthode *createFormBuilder()* :

```
  $form = $this->createFormBuilder($membre)
               -> ...
```

et modifiez le test sur la soumission du formulaire :

```
        if ($form->isSubmitted() && $form->isValid()) {
           $membre = $form->getData();

           $entityManager = $this->getDoctrine()->getManager();
           $entityManager->persist($membre);
           $entityManager->flush();

           $html = '<html><body> Nouveau membre : <br/> <ul>';
           foreach ($membre as $attribut) {
              $html .= '<li>'.$attribut.'</li>';
           }
           $html .= '</ul></body></html>';
           return new Response($html);
        }
```

Nous pouvons également chaîner ce controller vers celui qui liste les membres :

```
        if ($form->isSubmitted() && $form->isValid()) {
           $membre = $form->getData();

           $entityManager = $this->getDoctrine()->getManager();
           $entityManager->persist($membre);
           $entityManager->flush();

           return $this->redirectToRoute('app_membres');
        }
```

Dans les deux cas, l'objet est maintenant inséré dans la base de données. Mais ce que nous aimerions maintenant faire est d'intégrer le formulaire d'inscription à côté de la liste des membres remise à jour à chaque inscription.

3.7 Intégration dans la même interface du formulaire et de la liste :

Nous aimerions faire apparaître le formulaire d'inscription à côté de la liste des membres qui sera remise à jour à chaque nouvelle inscription.

3.7.1 Avec un seul controller et un seul template :

Voici la classe PHP de ce controller :

```
class NouveauMembreEtAffichageMembresController extends AbstractController {
    public function main(Request $request) {
        $membre = new Membres();
        $membres = array();

        $form = $this->createFormBuilder($membre)
            ->add('mail', TextType::class, array('label' => 'Adresse email '))
            ->add('nom', TextType::class, array('label' => 'Nom '))
            ->add('prenom', TextType::class, array('label' => 'Prénom '))
            ->add('save', SubmitType::class, array('label' => 'Nouveau membre'))
            ->getForm();
```

```
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
           $membre = $form->getData();
           $entityManager = $this->getDoctrine()->getManager();
           $entityManager->persist($membre);
           $entityManager->flush();

           $membres = $this->getDoctrine()
                           ->getRepository(Membres::class)
                           ->findAll();
        }

        return $this->render('nouveauMembreEtAffichage.html.twig', array(
            'form' => $form->createView(), 'membres' => $membres
        ));
    }
}
```

et le template associé :

```
<html>
  <body> <b> Gestion des membres : </b> <br/><br/>
    <div id="inscription" style="width: 500px; height:100%; float:left">
      <h3> Inscription </h3>
         {{ form_start(form) }}
         {{ form_widget(form) }}
         {{ form_end(form) }}
    </div>
    <div id="listeMembres">
      <h3>Membres</h1>
      <ul>
         {% for membre in membres %}
            <li> test {{ membre.nom }} </li>
         {% endfor %}
      </ul>
    </div>
  </body>
</html>
```

Nous remarquons l'usage de la boucle sous twig :

```
{% for membre in membres %}
   ...
{% endfor %}
```

3.7.2 Avec deux controllers invoqués dans le template d'un controller englobant :

Cette intégration, naturelle avec Angular, ne semble pas du tout l'être sous Symfony.

(Ce qui suit est inspiré de la documentation sise à cette URL :
`https://symfony.com/doc/current/templating/embedding_controllers.html`)

En effet, j'aurais aimé créer le template *default.html.twig* :

```
<html>
  <head> <style> </style> </head>
  <body> <b> Gestion de membres : </b> <br/><br/>
    <div style="width: 500px; float:left">
      {{ render(controller(
           'App\\Controller\\NouveauMembreInsertionIntegreeController
                                                        ::nouveauMembre'))
      }}
    </div>
    <div>
      {{ render(controller(
          'App\\Controller\\MembresController::listeMembres'))
      }}
    </div>
  </body>
</html>
```

Il est à remarquer que l'utilisation de *render(url()* pour invoquer un controller via une route ne fonctionnerait pas mieux :
`{{ render(url('<route>')) }}`

et le controlleur *DefaultController.php* :

```
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class DefaultController extends AbstractController {
    public function main() {
       return $this->render('default.html.twig');
    }
}
?>
```

invoqué sur la route qui permet d'appeler l'application intégrée :

```
app_index:
    path: /
    controller: App\Controller\DefaultController::main
```

permet l'affichage des deux templates des deux controllers, mais la validation du formulaire ne permet pas l'insertion dans la base de données d'un nouvel enregistrement.
Un mystère à creuser...