SYMFONY 5

Les fondamentaux

Qu'est ce que Symfony ?

- **Un ensemble de composants PHP**,
- Un modèle MVC prêt à l'emplois,
- Une grande communauté,
- Une maintenance régulière.

Les composants phares

- **Asset** : générateur d'URL,
- **Composer** : gestionnaire de dépendances,
- **Doctrine** : gestionnaire de BDD,
- **Form** : gestionnaire de formulaires,
- **Routing** : uniformise les requêtes HTTP,
- **Security** : gère les autorisations,
- **Translation** : gère les traductions,
- **Twig** : moteur de templates.

Démarrer un projet

1/ Installer Composer :

https://getcomposer.org/download/

2/ Installer Symfony :

https://symfony.com/download

3/ Créer un nouveau projet Symfony :

```
symfony new my_project_name --full
```

4/ Lancer le serveur Symfony

```
cd my_project_name/
symfony server:start
```

L'organisation des fichiers

- **bin/** : executables pour Symfony,
- **config/** : configurations
- **public/** : CSS, JS, images, etc
- **src/** : orm, controllers, formulaires,
- **templates/** : vues
- **tests/** : tests unitaires
- **translations/** : traductions
- **var/** : cache, logs, sessions,
- **vendor/** : librairies

Vocabulaire

- **Bundle** : Brique, contient tout le code qui gère une fonctionnalité,
- **Entité** : ORM,
- **Route** : URLs,
- **Service** : Utilitaires, class accessible depuis n'importe quel fichier,
- **NameSpace** : Compartiments

Erreur

```php
<?php
    class maClass{
        // ...
    }
?>

<?php
    class maClass{
        // ...
    }
?>
```

VS

Succès

```php
<?php
    namespace namespace1;

    class maClass{
        // ...
    }
?>

<?php
    namespace namespace2;

    class maClass{
        // ...
    }
?>
```

Créer une page

https://symfony.com/doc/master/page_creation.html

Créer une page

1. Choisir / créer un controller,
2. Créer une méthode,
3. Créer une route,
4. Créer une vue.

Un peu de pratique

Créer le controller CategorieController

Twig et les templates

Extension :

`*.html.twig`

Afficher la valeur d'une variable :

`{{ ... }}`

Définir quelque chose :

`{% ... %}`

Twig et les templates

Afficher d'une variable :

```
{{ maVar }}
```

Etendre un template :

```
{% extends 'base.html.twig' %}
```

Les blocs :

```
{% block [nom_du_bloc] %}
   ...
{% endblock %}
```

Paramétrer son application

config/
- **packages/**
  - Configuration des paquets installés
- **routes/**
  - Configuration des routes
- **bundles.php**
  - Import des bundles
- **services.yaml**
  - Paramètres et services

.env

La base de données

Doctrine

Créer une base de données

```
php  bin/console  doctrine:database:create
```

unix_socket: /Applications/MAMP/tmp/mysql/mysql.sock

Les entités - création

Créer une entité :

```
php bin/console make:entity [Name]
```

Ma BDD est-elle à jour ?

```
php bin/console doctrine:schema:validate
```

Un peu de pratique

Créer une entité Catégorie

Les entités - migration

Préparer une migration

```
php bin/console make:migration
```

Mettre à jour la BDD avec la migration

```
php bin/console doctrine:migrations:migrate
```

Attention aux pertes de données

Les entités - récupération 1/2

Entity manager :

```
$this->getDoctrine()->getManager();
```

Récupérer une entité :

```
$em->getRepository([Entite]::class)->findAll();
```

Afficher toute une variable :

```
{{ dump([Variable]) }}
```

Twig - conditions et boucles

SI :

```
{% if categories is not empty %}
   ...
{% else %}
   ...
{% endif %}
```

FOR … IN :

```
{% for category in categories %}
   ...
{% endfor %}
```

Les formulaires - préparation

Charger les types de champs souhaités :

```
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
```

Préparer le formulaire :

```
$category = new Category();
$form = $this->createFormBuilder($category)
  ->add( 'name', TextType::class )
  ....
  ->add( 'save', SubmitType::class, array( 'label' => 'Maj' ) )
  ->getForm();
....
Return $this->render(... [ 'form' => $form->createView() ] );
```

Les formulaires - affichage

Envoyer le formulaire à la vue :

```
'form' => $form->createView()
```

Afficher le formulaire :

```
{{ form_start(form) }}
  {{ form_errors(form) }}
  {{ form_row(form.field1) }}
  {{ form_label(form.field2) }}
  {{ form_widget(form.field2) }}
{{ form_end(form) }}
```

Les formulaires - soumission

Détecter les requêtes HTTP :

```
use Symfony\Component\HttpFoundation\Request;
```

Détecter l'envois du formulaire :

```
public function index( Request $request ) {
...
$form->handleRequest($request);
...
if ( $form->isSubmitted() && $form->isValid() ) {
  // Le formulaire à été soumis et est valide
}
```

Les formulaires - sauvegarde

Récupérer les données envoyées :

```
$category => $form->getData()
```

Récupérer une donnée :

```
$category->getName();
```

Sauvegarde en BDD :

```
$em->persist($category);
$em->flush();
```

Formulaires - validation

Charger le validateur et définir des règles de validation :

```
use Symfony\Component\Validator\Constraints as Assert;
...
/**
 * ...
 * @Assert\NotBlank()
 */
private $name;
```

Supprimer la vérification HTML5 d'un champ :

```
$form = $this->createFormBuilder($category)
...
 ->add('description', TextareaType::class, array('required' => false) )
```

Twig - route name et path

Nommer ses routes - controller

```
@Route( "/categories", name="categories" )
```

Liens - vue

```
{{ path( 'categories' ) }}
```

Routes dynamiques (avec paramètres) - controller

```
@Route( "/category/{id}", name="category" )
```

Liens avec paramètres - vue

```
{{ path( 'category', { 'id': category.id } ) }}
```

Les entités - récupération 2/2

Récupérer une occurence particulière :

```
use App\Entity\Category;

/**
 * @Route( "/category/{id}", name="category" )
 */
public function category( Category $category ) {
  // ça y est, vous posséder la bonne catégorie
}
```

Formulaires - class

FormType

Externalise et automatise la création des formulaires

Un peu de pratique

Sur la page d'une catégorie, mettre en place le formulaire d'édition de la catégorie.

Formulaires - design

Méthodes	Avantages	Inconvénients
Dans la page	Rapide et simple	Pas réutilisable sur les autres pages
Dans un template à part	Réutilisable sur plusieurs pages	Long à mettre en place

Formulaires - design

Par défaut :

```
/vendor/symfony/twig-bridge/Resources/views/Form/
form_div_layout.html.twig
```

Créer son propre fichier de rendus :

```
/templates/form/fields.html.twig
```

Indiquer à Symphony d'utiliser notre fichier de rendus :

/config/packages/twig.yml

```
twig:
  form_themes:
    - 'form/fields.html.twig'
```

Les messages Flash

Ajouter un message Flash :

```
$this->addFlash(
    '[type]',
    '[message]'
);
```

Afficher les messages Flash :

```
{% for type, messages in app.flashes %}
    {% for message in messages %}
        <div class="{{ type }}">
            {{ message }}
        </div>
    {% endfor %}
{% endfor %}
```

Les jointures

Les différents types de jointures :
http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/association-mapping.html

Un **article** ne peut avoir qu'une **seule catégorie** mais une **catégorie** peut avoir **plusieurs articles** :

- article.php : *ManyToOne*
- category.php : *OneToMany*

Jointure et formulaire

Adapter le formulaire d'ajout d'article :

```
->add([var], EntityType::class, array(
  'class' => [entity]::class,
  'choice_label' => [field]
))
```

Supprimer une instance

Supprimer un objet :

```
$em->remove($article);
$em->flush;
```

Suppression et enfants

Supprimer les entités enfants :

```
@ORM\OneToMany(... , cascade={"persist", "remove"})
```

- **Persist :** met automatiquement à jour l'enfant en cas de création ou MAJ du parent,
- **Remove :** supprime automatiquement l'enfant en cas de suppression du parent

Conserver les entités enfants :

```
@ORM\JoinColumn(... , onDelete="SET NULL")
```

Un peu de pratique

Exercice 1

Upload de fichier

Côté entité

```
use Symfony\Component\Validator\Constraints as Assert;
...
/**
 * @ORM\Column(type="text", nullable=true)
 * @Assert\File(mimeTypes={ "image/png", "image/jpeg" })
 */
  private $image;
```

Côté formulaire

```
use Symfony\Component\Form\Extension\Core\Type\FileType;
...
  ->add('image', FileType::class, array('label'=>'Image (JPEG, PNG'))
```

Upload de fichier

Côté controller

```
$image = $produit->getImage();

$imageName = md5(uniqid()).'.'.$image->guessExtension();

$image->move(
 $this->getParameter('upload_directory').'/produits',
 $imageName
);

$produit->setImage($imageName);
```

Upload de fichier

String -> File

```
use Symfony\Component\HttpFoundation\File\File;
...
$produit->setImage(
 new File($this->getParameter('upload_directory').'/produits/'.
$produit->getImage())
);
```

Un peu de pratique

Exercice 2

Suppression automatique de fichier

Coté entité

```
@ORM\HasLifecycleCallbacks()
...
 /**
  * @ORM\PostRemove
  */
public function deleteImage(){
 if(file_exists(__DIR__.'/../../public/uploads/produits/'.$this->image)){
  unlink(__DIR__.'/../../public/uploads/produits/'.$this->image);
 }
 return true;
}
```

Traductions simples

Configuration :

```
config/packages/translation.yaml
```

Les traductions :

```
translations/messages.[locale].yaml
```

Afficher une traduction (vue) :

```
{{ 'categories.title'|trans }}
```

Afficher une traduction (controller) :

```
public function test(TranslatorInterface $translator) {
$translator->trans('categories.added');
```

Traductions (un peu moins) simples

```
Il y a %nb% catégories
```

Vue :

```
{{ 'categories.there_is'|trans( { '%nb%' : 2 } ) }}
```

Afficher une traduction (controller) :

```
$translator->trans('categories.there_is', [ '%nb%' => 2 ] );
```

Un peu de pratique

Exercice 3