

Framework *Symfony*

(Une introduction)

Enrica Duchi, Sylvain Perifel et Cristina Sirangelo

L₃ Info – Université Paris Diderot

Introduction

- ▶ **Symfony** : *framework* côté serveur basé sur **PHP**
- ▶ Intérêts :
 - structuration du code (**MVC**)
 - simplification du développement
 - nombreux modules existants (bibliothèque)
- ▶ Points saillants :
 - routage facile et url propres (via **annotations**)
 - contrôleurs : PHP et objet
 - manipulation des bases de données : **Doctrine**
 - langage de templates : **Twig**
 - gestion de formulaires facilitée
 - etc.

Plan

1. Démarrage et organisation

2. Templates

3. Contrôleurs

4. Base de données

5. Formulaires

Plan

1. Démarrage et organisation

2. Templates

3. Contrôleurs

4. Base de données

5. Formulaires

Installation

- ▶ Télécharger l'installateur (déjà présent sur Lucien)
→ commande `symfony`

- ▶ Nouveau projet :

```
symfony new mon_projet lts
```

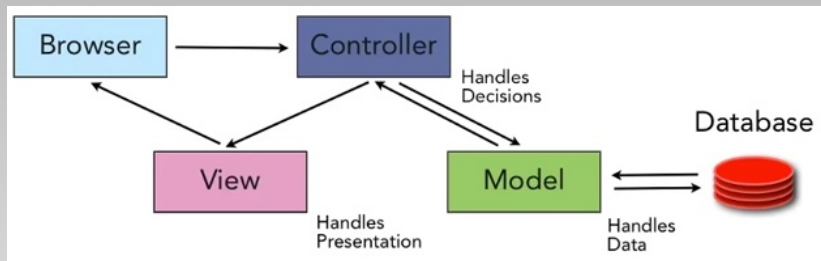
→ télécharge le code nécessaire dans le répertoire `mon_projet`

- ▶ Serveur de développement :

```
php bin/console server:start
```

→ écoute sur `http://localhost:8000`

Modèle-vue-contrôleur



- ▶ Méthode d'organisation du code
- ▶ La partie serveur est coupée en trois morceaux :
 - le **modèle** qui implémente des services (p. ex. gestion de la base de données);
 - le **contrôleur** qui prend les décisions pour générer les pages demandées (\simeq la partie « algorithmique »);
 - la **vue** (*templates*) qui organise la présentation (affichage).

MVC en pratique

- ▶ Contrôleurs en **PHP** (orienté objets) :
 - le *kernel* Symfony fait le chef d'orchestre (*front controller*)
 - ▶ selon l'url reçue, invoque un contrôleur (*controller*) pour générer la page demandée
 - ▶ gestion des url par annotations
 - un contrôleur pour chaque « section » du site (p. ex. login, enregistrement, chat, etc.)
 - ▶ un contrôleur (classe PHP) : plusieurs *actions*, une action pour chaque page à générer
 - une action d'un contrôleur (fonction) : implémente la logique de generation d'une page :
 - ▶ récupération des données / de l'input, traitement et envoi aux vues

MVC en pratique (suite)

- ▶ Vues en Twig :
 - une vue par « type de page »
 - langage de templates simple et puissant
 - décrit la présentation de la page en fonction de plusieurs paramètres passés par le contrôleur
 - separation logique de « calcul » de la page / affichage
 - ▶ mais très flexible : langage riche, possibilité d'inclure de la logique dans la presentation (p. ex. *embedded controllers*)
 - héritage

MVC en pratique (suite)

▶ Modèle :

un ensemble de services (p. ex. gestion de la base de données, des formulaires, de la sécurité (authentification), des messages et mails, etc.)

- la plupart des services courants fournis par Symfony
- service de gestion de la bd (mysql) : **Doctrine**
 - ▶ correspondance : données dans la bd \Leftrightarrow objets PHP
- possibilité de créer ses propres services (accessibles par tous les contrôleurs)

Organisation des fichiers

- ▶ Deux répertoires pour développer : `src` (code PHP) et `app` (le reste : configuration, templates, etc.)
 - `src/AppBundle/Controller` : les contrôleurs
 - `src/AppBundle/Entity` : les classes pour la base de données
 - `app/Resources/views` : les vues (templates)
 - `app/config` : les fichiers de configuration
- ▶ Autres répertoires :
 - `web` : contient les fichiers publiquement accessibles (images, CSS, etc.)
 - `bin` : contient les exécutables (notamment `console`)
 - `var` : cache, logs, etc.
 - `tests` : pour les tests automatiques
 - `vendor` : les modules additionnels

Plan

1. Démarrage et organisation

2. Templates

3. Contrôleurs

4. Base de données

5. Formulaires

base.html.twig

Le template de base, avec les blocs à remplir :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>{% block title %}Titre{% endblock %}</title>
    {% block stylesheets %}
      <link href="{ asset('css/monstyle.css') }}" rel="stylesheet" />
    {% endblock %}
  </head>
  <body>
    {% block body %}{% endblock %}
    {% block javascripts %}{% endblock %}
  </body>
</html>
```

Exemple

Template appelé avec l'objet `articles` en argument :

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

{% block title %}Exemple{% endblock %}

{% block stylesheets %}
{{ parent() }}
<link href="{ asset('css/exemple.css') }}" rel="stylesheet" />
{% endblock %}

{% block body %}
    <h1>Bienvenue</h1>

    {% for article in articles %}
    <a href="{ path('voir', {'id':article.id}) }}">{{article.title}}</a>
    {% endfor %}

{% endblock %}
```

Exemple

Template appelé avec l'objet `articles` en argument :

```
{% extends 'base.html.twig' %} (héritage)
```

```
{% block title %}Exemple{% endblock %} (bloc écrasé)
```

```
{% block stylesheets %}
```

```
{% parent() %} (inclure le contenu du bloc parent)
```

```
<link href="{% asset('css/exemple.css') %}" rel="stylesheet" />
```

```
{% endblock %}
```

```
{% block body %}
```

```
<h1>Bienvenue</h1>
```

(passé en paramètre)

```
{% for article in articles %} (boucle)
```

```
<a href="{% path('voir', {'id':article.id}) %}">{{article.title}}</a>
```

```
{% endfor %}
```

(lien avec passage de paramètre)

```

```

(lien vers un doc public)

```
{% endblock %}
```

Plan

1. Démarrage et organisation

2. Templates

3. Contrôleurs

4. Base de données

5. Formulaires

Base

Routage et code simples (DefaultController):

```
<?php

namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller {
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction () {
        return $this->render('home.html.twig');
    }
}
```


Base

Routage et code simples (DefaultController):


```
<?php

namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller {
    /**
     * @Route("/", name="homepage") (action à exécuter quand l'url est /)
     */
    public function indexAction () {
        return $this->render('home.html.twig');
    }
}
```

(appel du template `home.html.twig`)



Routage avec paramètre

```
// ...
class LuckyController extends Controller
{
    /**
     * @Route("/lucky/number/{count}")
     */
    public function numberAction($count)
    {
        $numbers = array();
        for ($i = 0; $i < $count; $i++) {
            $numbers[$i] = rand(0, 100);
        }

        $numbersList = implode(', ', $numbers);

        return $this->render(
            'lucky/number.html.twig',
            array('luckyNumberList' => $numbersList) );
    }
}
```

Routage avec paramètre

```
// ...
class LuckyController extends Controller
{
    /**
     * @Route("/lucky/number/{count}")
     */
    public function numberAction($count)
    {
        $numbers = array();
        for ($i = 0; $i < $count; $i++) {
            $numbers[$i] = rand(0, 100);
        }

        $numbersList = implode(', ', $numbers);

        return $this->render(
            'lucky/number.html.twig',
            array('luckyNumberList' => $numbersList) );
    }
}
```

(appel du template `number.html.twig` avec paramètre)

Plan

1. Démarrage et organisation

2. Templates

3. Contrôleurs

4. Base de données

5. Formulaires

Doctrine

- ▶ Communication facilitée avec la base de données
- ▶ Entités représentées par des objets
- ▶ Configuration dans `app/config/parameters.yml`

- ▶ Définir une table :

```
php bin/console doctrine:generate:entity  
(script interactif)
```

- ▶ Crée la classe `src/AppBundle/Entity/Nom_table.php` comme interface abstraite pour la base, et le fichier `src/AppBundle/Repository/Nom_tableRepository.php` (initialement vide) où les requêtes principales seront codées.
- ▶ Créer la table dans la base :

```
php bin/console doctrine:schema:update --force
```

Insertion

```
use AppBundle\Entity\Product;
use Symfony\Component\HttpFoundation\Response;

public function createAction() {
    $product = new Product();
    $product->setName('A Foo Bar');
    $product->setPrice('19.99');
    $product->setDescription('Lorem ipsum dolor');
    $em = $this->getDoctrine()->getManager();
    $em->persist($product);
    $em->flush();
    ...
}
```

Insertion

```
use AppBundle\Entity\Product;
use Symfony\Component\HttpFoundation\Response;

public function createAction() {
    $product = new Product();
    $product->setName('A Foo Bar');
    $product->setPrice('19.99');
    $product->setDescription('Lorem ipsum dolor');
    $em = $this->getDoctrine()->getManager();
    $em->persist($product); (≃ « add » et « commit »)
    $em->flush(); (≃ « push »)
    ...
}
```

Requêtes préprogrammées

```
$repository = $this->getDoctrine()  
    ->getRepository('AppBundle:Product');  
  
// query by the primary key (usually "id")  
$product = $repository->find($id);  
  
// dynamic method names to find based on a column value  
$product = $repository->findOneById($id);  
$product = $repository->findOneByName('foo');  
  
// find *all* products  
$products = $repository->findAll();  
// find a group of products based on an arbitrary column value  
$products = $repository->findByPrice(19.99);  
  
// query for one product matching by name and price  
$product = $repository->findOneBy(  
    array('name' => 'foo', 'price' => 19.99) );  
// query for all products matching the name, ordered by price  
$products = $repository->findBy(  
    array('name' => 'foo'),  
    array('price' => 'ASC') );
```


Requêtes en SQL

```
$em = $this->getDoctrine()->getManager();  
$query = $em->createQuery(  
    'SELECT p  
    FROM AppBundle:Product p  
    WHERE p.price > :price  
    ORDER BY p.price ASC'  
)->setParameter('price', '19.99');  
$products = $query->getResult();
```

Plan

1. Démarrage et organisation

2. Templates

3. Contrôleurs

4. Base de données

5. Formulaires

Formulaire

- ▶ À partir d'un objet contenant des champs (p. ex. venant d'une base de données), création simplifiée de formulaires contenant ces champs
- ▶ Types de champs devinés automatiquement
- ▶ Facilitation du traitement

Exemple

```
$task = new Task();

$form = $this->createFormBuilder($task)
    ->add('task', TextType::class)
    ->add('dueDate', DateType::class)
    ->add('save', SubmitType::class, array('label' => 'Create Task'))
    ->getForm();

$form->handleRequest($request);
if ($form->isSubmitted() && $form->isValid()) {
    $em = $this->getDoctrine()->getManager();
    $em->persist($task);
    $em->flush();
    return $this->redirectToRoute('task_success');
}

return $this->render('default/new.html.twig',
    array( 'form' => $form->createView() ));
```

Dans le template

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

(personnalisation possible avec `{{ form_row(form.task) }}`
par exemple)

Documentation

`https://symfony.com/doc/3.4/index.html`

- ▶ guides et tutoriels
- ▶ référence Symfony

Éviter OpenClassrooms ici! (doc issue de la version 2)