

Symfony

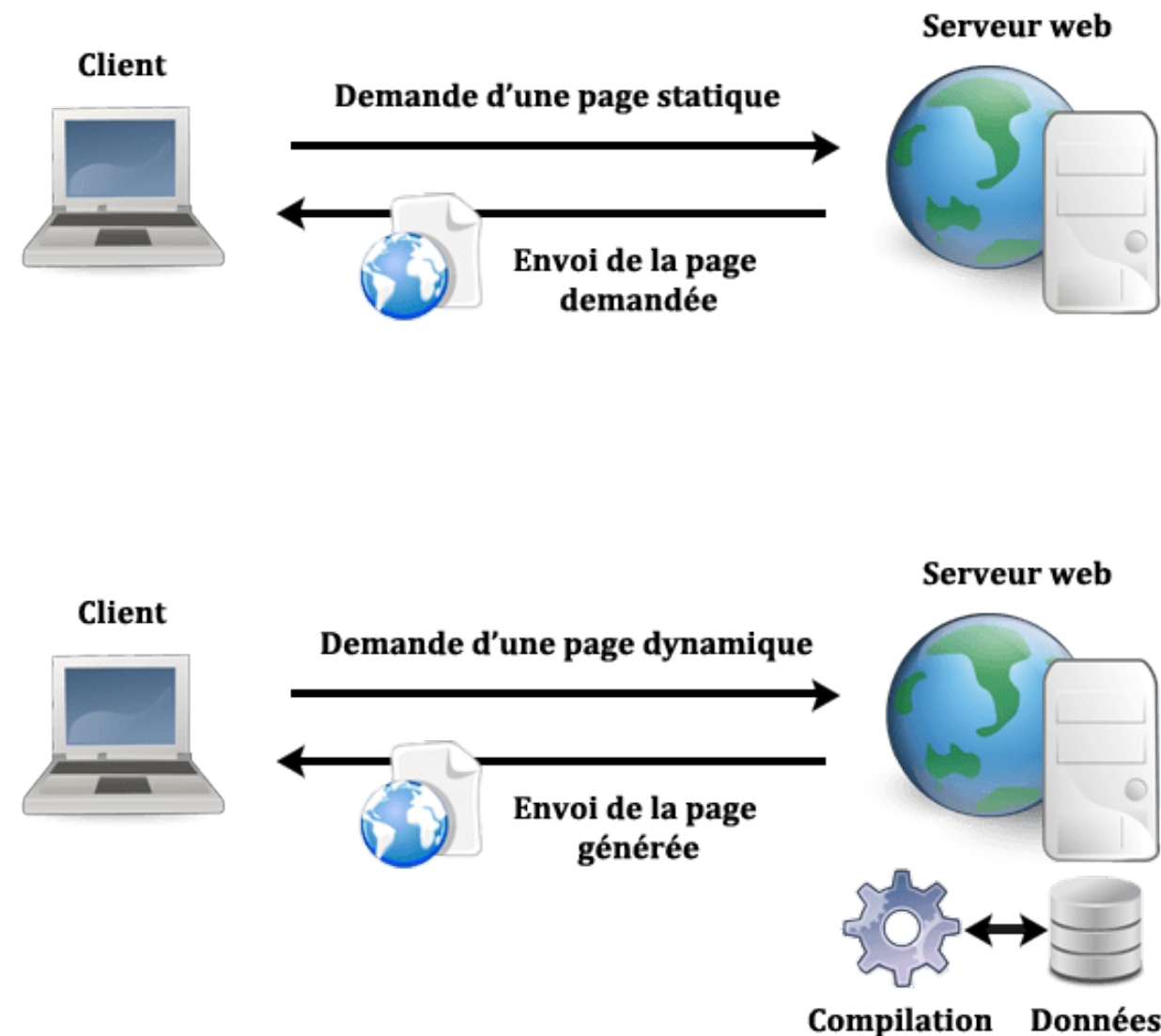
# Sommaire

- **I/ Notions générales**
  - Rappels PHP
  - Rappels POO
  - Framework
  - MVC
  - Symfony
  - formats XML(HTML) / YAML / JSON
- **II/ Installation**
  - Utilisation de la console
  - Installation de PHP + WAMP / LAMP / MAMP
  - Composer
  - Serveur
  - IDE
  - Installation de symfony + Organisation des fichiers
- **III/ Hello, world!**
  - Première page
  - Routage
  - Controllers
  - Templates
- **IV/ Doctrine**
  - BDD
  - ORM
  - Doctrine
  - Entities
  - ...
- **V/ Formulaires**
- **VI/ Utilisateurs**
- **VII/ Autres notions**

# I/ Notions générales

# Rappels PHP

- Web statique vs Web dynamique



# Rappels PHP

- Langage de script interprété (vs langage compilé)

```
[~ » php -a
Interactive shell

[php > $a = "ok"; echo $a;
ok
[php > echo $b; $b = "ok";
PHP Notice:  Undefined variable: b in php shell code on line 1

Notice: Undefined variable: b in php shell code on line 1
```

- Sensible à la casse

# Rappels PHP

- Le code est inséré dans une balise : `<?php //code ?>`
- La commande `echo` permet d'afficher du contenu texte :

```
<html>
<body>
<?php $titre = "My Title" ?>
<h1><?php echo $titre ?> </h1>
<!-- reste de la page -->
</body>
</html>
```

# Rappels PHP : Variables

- Typage faible
- Déclaration non nécessaire
- Noms de variable impérativement précédés de "\$"

```
<?php
// affectation simple
$tonneaux_nb = 10;

// affectation multiple
$tonneaux_nb = $stock = 10 ;
?>
```

# Rappels PHP : Types de variables

Type	Signification	Plage de valeurs	Exemple
Integer	Entier	-2 147 483 648 >> + 2 147 483 647	<code>\$var = 12;</code>
Double	Réel	1.7E-308 >> 1.7E+308	<code>\$var = 3.14159;</code>
String	Chaîne de caractères	caractère alphanumérique	<code>\$var = bienvenue;</code>



# Rappels PHP : Types de variables

- Type défini en fonction de la valeur attribuée
- L'opérateur "+" convertit les types en numérique

```
<?php
$var = "0";
echo var_dump ($var). '<br />'; // string(1) "0"
$var = $var + 1;
echo var_dump ($var). '<br />'; // int(1)
$var += 1.3;
echo var_dump ($var). '<br />'; // float(2.3)
$var = 2 + "8 arbres";
echo var_dump ($var); // int(10) Notice: A non well formed numeric value encountered
?>
```

# Rappels PHP : Types de variables

- Booléen n'existe pas mais TRUE et FALSE existent

```
[~ >> php -a
Interactive shell

[php > echo TRUE;
1
[php > echo FALSE;
[php > echo (TRUE == -1);
1
[php > echo (TRUE == 0);
[php > echo (TRUE == 1);
1
[php > echo (FALSE == -1);
[php > echo (FALSE == 0);
1
[php > echo (FALSE == 1);
```

# Rappels PHP : Chaînes de caractères

```
<?php
$annee_actu = "2020";

echo "Année : $annee_actu<br />";
// Année : 2003

echo 'Année : $annee_actu<br />';
// Année : $annee_actu

echo 'Année : ' . $annee_actu . '<br />';
// Année : 2003

echo 'Année n + 2 = ' . ($annee_actu + 2) . '<br />';
// Année n + 2 = 2005

echo "La variable \$annee_actu vaut $annee_actu";
// La variable $annee_actu vaut 2003
?>
```

# Rappels PHP : Chaînes de caractères - fonctions usuelles

Fonction	Utilisation	Exemple
<code>strlen()</code>	renvoie le nombre de caractères contenus dans la chaîne	<code>\$nbcar = strlen(\$var);</code>
<code>substr()</code>	extraît un nombre de caractères d'une chaîne à partir d'une position	<code>\$extrait = substr(\$var, \$pos, \$nb);</code>
<code>trim()</code>	supprime les espaces avant et après la chaîne	<code>\$var = trim(\$var);</code>
<code>explode()</code>	renvoie dans un tableau les valeurs comprises entre un séparateur	<code>\$tableau = explode(\$delimiteur, \$var);</code>

# Rappels PHP : Tableaux

- Tableaux à indice numérique

```
<?php
$mois[1] = "janvier";
$mois[2] = "février";
$mois[3] = "mars";
$mois[] = "avril"; // l'indice 4 a été automatiquement affecté
?>
```

- Une chaîne de caractères est un tableau

```
<?php
$site = "abcdefgh";
echo "$site[4]<br />"; // e
echo "$site[0]<br />"; // a
?>
```

# Rappels PHP : Tableaux

- La fonction `Array()` permet de déclarer et remplir un tableau :

```
<?php
$mois = array("janvier", "février", "mars", "avril");
echo $mois[1]; // février
?>
```

# Rappels PHP : Tableaux

## - Tableaux à double entrée

```
<?php
$case[1][1] = "vide";   $case[1][2] = "rond";   $case[1][3] = "croix";
$case[2][1] = "rond";   $case[2][2] = "croix";   $case[2][3] = "vide";
$case[3][1] = "croix";  $case[3][2] = "rond";   $case[3][3] = "vide";

echo "la case 3,1 contient : ".$case[3][1]."<br />";
// la case 3,1 contient : croix
?>
```

## - Tableaux associatifs

```
<?php
$tableau['pays'] = "Etats-Unis";
$tableau['CO2'] = "20";
$tableau['Kyoto'] = "non";
?>
```

# Rappels PHP : Structures de contrôle - if

```
<?php
if ($cond1) {

    // instructions à exécuter si la condition 1 est vraie

} elseif ($cond2) {

    // instructions à exécuter si la condition 2 est vraie

} else {

    // instructions à exécuter si toutes les
    //valeurs des expressions précédentes sont fausses

}
?>
```



# Rappels PHP : Structures de contrôle - switch

```
<?php
switch (date("m")) {
    case 1 :
        $mois = "janvier";
        break;
    case 2 :
        $mois = "février";
        break;
    case 3 :
        $mois = "mars";
        break;

    // ...

    default:
        echo "Il y a un soucis";
        break;
}
echo "Lettre d'information mensuelle $mois ".date("Y");
?>
```

# Rappels PHP : Boucles - while

```
<?php  
while ($cond) {  
    // instructions à exécuter;  
}  
?>
```

# Rappels PHP : Boucles - do ... while

```
<?php  
do {  
    // instructions à exécuter;  
}  
while ($cond) ;  
?>
```

# Rappels PHP : Boucles - foreach

```
<?php  
foreach ($tableau as $cle => $valeur) {  
    // instructions à exécuter;  
}  
?>
```

# Rappels PHP : Fonctions

```
<?php
function calcul_superficie($rayon) {
    $sup = $rayon * $rayon * 3.14159;
    return $sup;
}
?>
```

- Fonction avec argument facultatif

```
<?php
function calcul_superficie($rayon = 12) {
    $sup = $rayon * $rayon * 3.14159;
    return $sup;
}
?>
```

# Rappels PHP : namespace

- Espace de nom
- Permet d'éviter la collision de noms

```
namespace my\name
```

# Rappels PHP : use

## - Importation de classes, fonctions, constantes

*// Before PHP 7*

```
use com\tutorialspoint\ClassA;  
use com\tutorialspoint\ClassB;  
use com\tutorialspoint\ClassC as C;
```

```
use function com\tutorialspoint\fn_a;  
use function com\tutorialspoint\fn_b;  
use function com\tutorialspoint\fn_c;
```

```
use const com\tutorialspoint\ConstA;  
use const com\tutorialspoint\ConstB;  
use const com\tutorialspoint\ConstC;
```

*// PHP 7+ code*

```
use com\tutorialspoint\{ClassA, ClassB, ClassC as C};  
use function com\tutorialspoint\{fn_a, fn_b, fn_c};  
use const com\tutorialspoint\{ConstA, ConstB, ConstC};
```

# Rappels - Programmation orientée objet

- Classe / Objet
  - Attributs
  - Méthodes
- Héritage
- Parent / Enfant
- Polymorphisme
- Surcharge
- Abstraction
- Constructeur / Destructeur



# OOP et PHP - Classe

```
<?php
class phpClass {
    var $var1;
    var $var2 = "constant string";

    function myfunc ($arg1, $arg2) {
        // ...
    }
    // ...
}
?>
```

# OOP et PHP - Classe

```
<?php
class Books {
    /* Member variables */
    var $price;
    var $title;

    /* Member functions */
    function setPrice($par) {
        $this->price = $par;
    }

    function getPrice() {
        echo $this->price . "<br/>";
    }

    function setTitle($par) {
        $this->title = $par;
    }

    function getTitle() {
        echo $this->title . " <br/>";
    }
}
?>
```

# OOP et PHP - Objet

```
<?php  
$physics = new Books;  
$maths = new Books;  
$chemistry = new Books;  
?>
```

# OOP et PHP - Objet

```
<?php
$physics->setTitle( "Physics for High School" );
$chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );

$physics->setPrice( 10 );
$chemistry->setPrice( 15 );
$maths->setPrice( 7 );

$physics->getTitle();
$chemistry->getTitle();
$maths->getTitle();
$physics->getPrice();
$chemistry->getPrice();
$maths->getPrice();
?>
```

# OOP et PHP - Constructeur

```
<?php
function __construct( $par1, $par2 ) {
    $this->title = $par1;
    $this->price = $par2;
}

$physics = new Books( "Physics for High School", 10 );
$maths = new Books ( "Advanced Chemistry", 15 );
$chemistry = new Books ( "Algebra", 7 );

/* Get those set values */
$physics->getTitle();
$chemistry->getTitle();
$maths->getTitle();

$physics->getPrice();
$chemistry->getPrice();
$maths->getPrice();
?>
```

# OOP et PHP - Héritage

```
<?php
class Novel extends Books {
    var $publisher;

    function setPublisher($par) {
        $this->publisher = $par;
    }

    function getPublisher() {
        echo $this->publisher. "<br />";
    }
}
?>
```

# OOP et PHP - Héritage

```
<?php
class Novel extends Books {
    var $publisher;

    function setPublisher($par) {
        $this->publisher = $par;
    }

    function getPublisher() {
        echo $this->publisher. "<br />";
    }
}
?>
```

# OOP et PHP - Membres privés

```
<?php
class MyClass {
    private $car = "skoda";
    $driver = "SRK";

    function __construct($par) {
        // Statements here run every time
        // an instance of the class
        // is created.
    }

    function myPublicFunction() {
        return "I'm visible!";
    }

    private function myPrivateFunction() {
        return "I'm not visible outside!";
    }
}
?>
```



# OOP et PHP - Membres protégés

```
<?php
class MyClass {
    protected $car = "skoda";
    $driver = "SRK";

    function __construct($par) {
        // Statements here run every time
        // an instance of the class
        // is created.
    }

    function myPublicFunction() {
        return "I'm visible!";
    }

    protected function myPrivateFunction() {
        return "I'm visible in child class!";
    }
}
?>
```

# OOP et PHP - Classe abstraite

```
<?php
abstract class MyAbstractClass {
    abstract function myAbstractFunction() {
    }
}
?>
```

# OOP et PHP - Mot clé static

```
<?php
class Foo {
    public static $my_static = 'foo';

    public function staticValue() {
        return self::$my_static;
    }
}

print Foo::$my_static . "\n";
$foo = new Foo();

print $foo->staticValue() . "\n";
?>
```

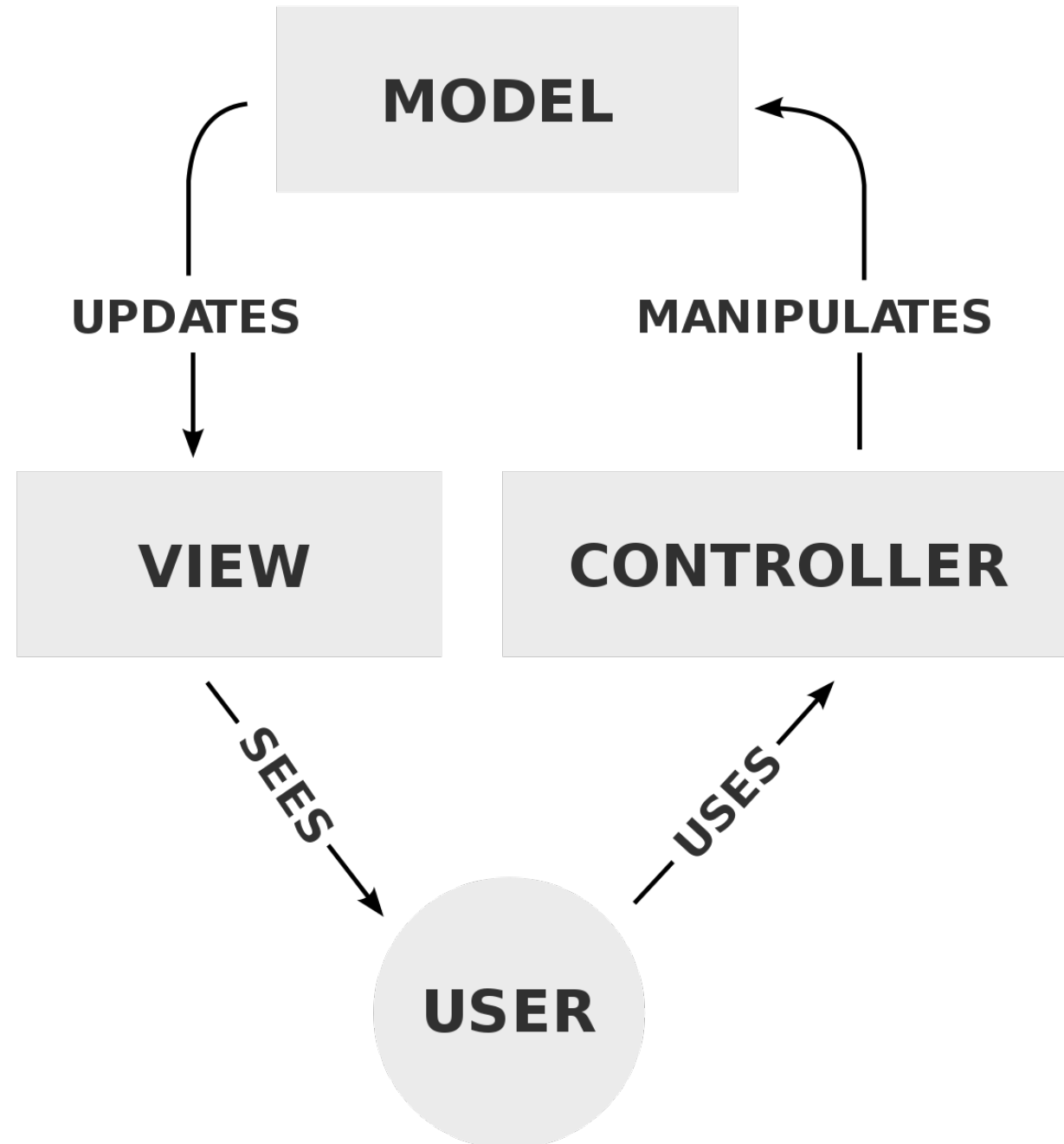
# Framework

- Cadre de travail
- "Squelette" d'application
  - Outils de conception (Fonctions, Modules, ...)
  - Logique de construction
- Idée : ne pas avoir à réinventer des composants basiques

# MVC

- Model / View / Controller
- **Contrôleur** (ou Controller) : génère une réponse à la requête de l'utilisateur indépendamment des données ou de l'affichage
- **Modèle** (ou Model) : abstraction de la couche données
- **Vue** (ou View) : affichage de la page indépendamment des données

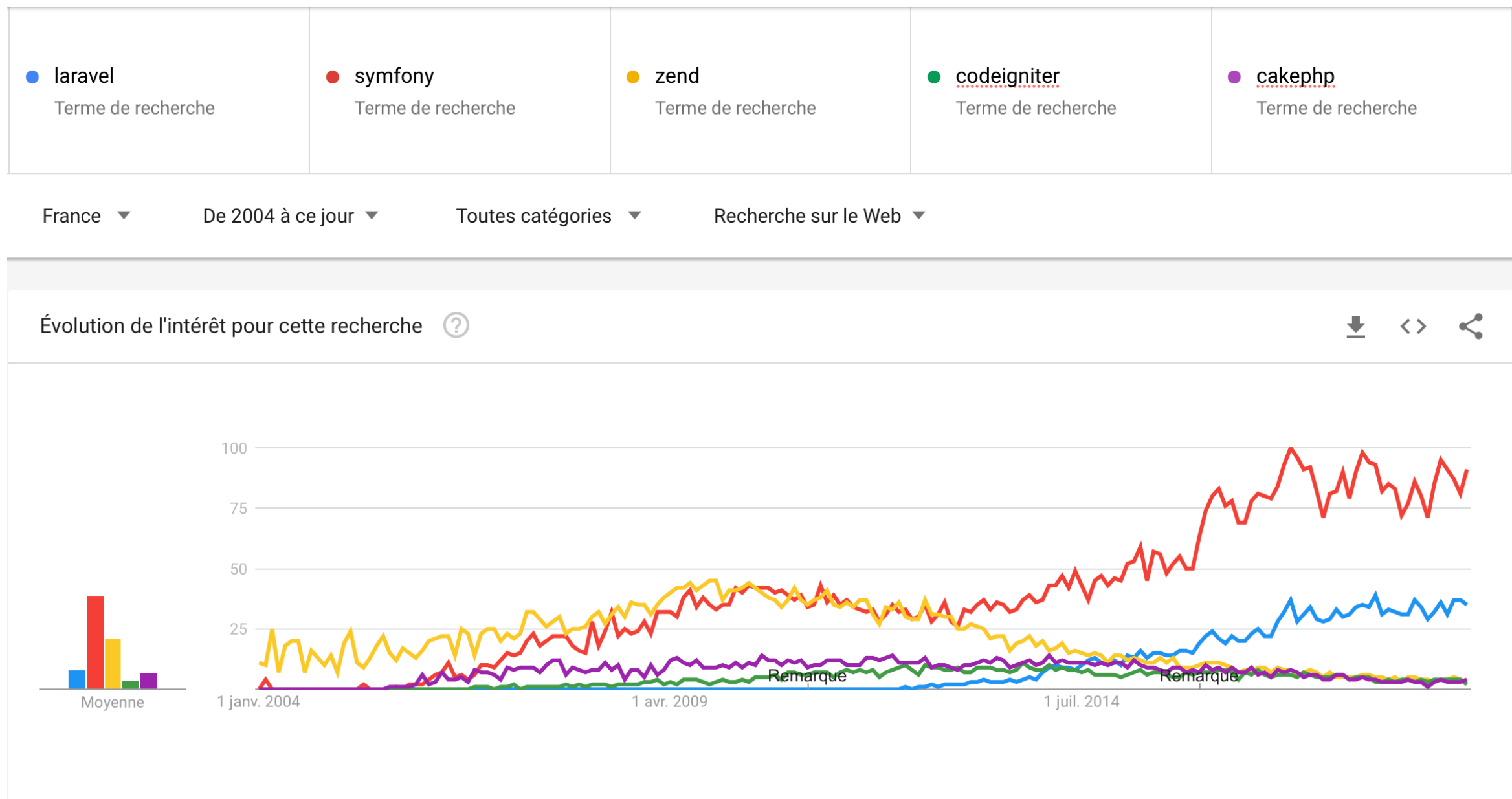
# MVC



# Frameworks PHP

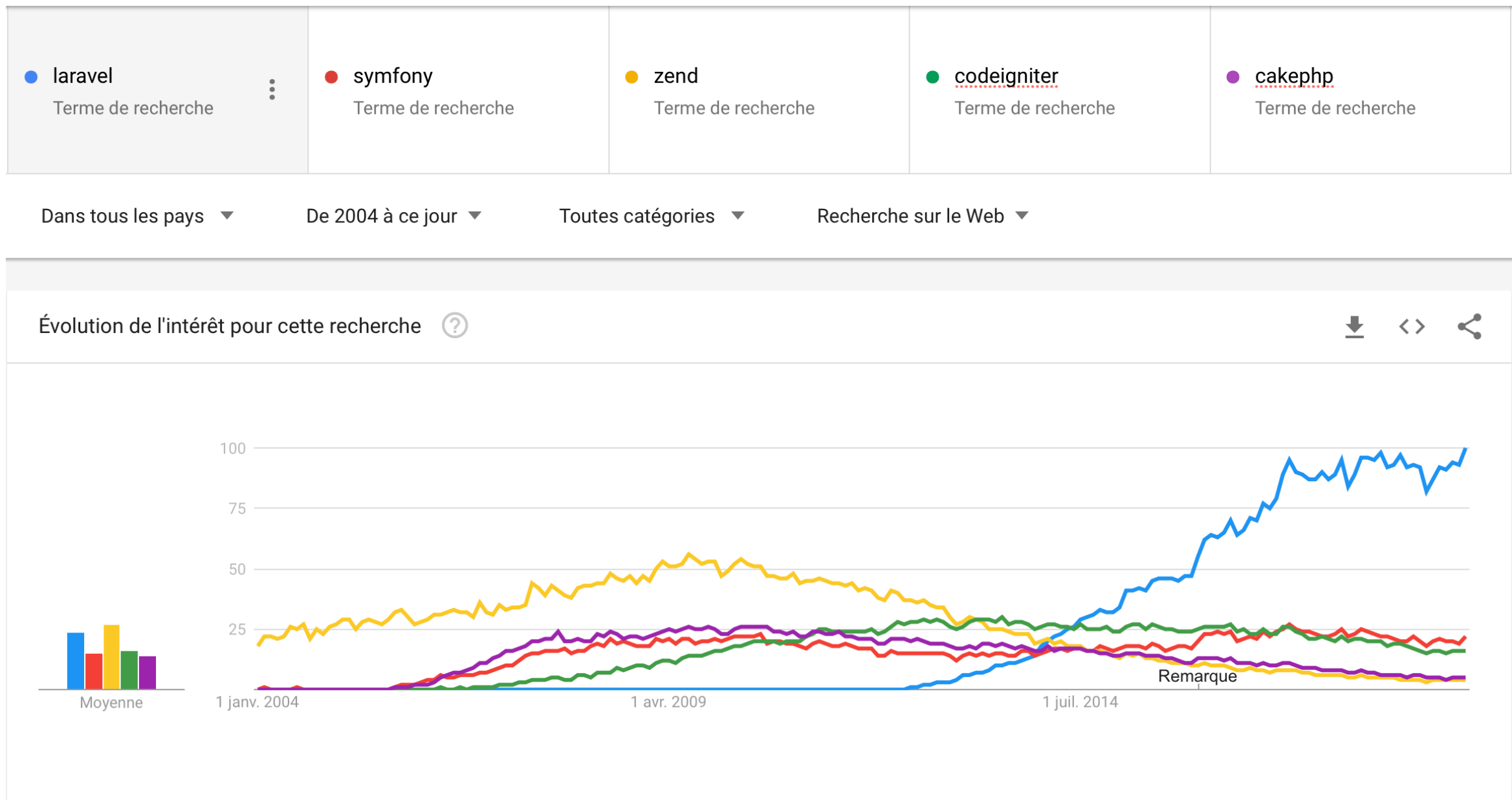


# Frameworks PHP : parts de marché





# Frameworks PHP : parts de marché



# Symfony

- 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

# Entreprises utilisant Symfony

VOGUE

dailymotion



mestic

trivago



# II/ Installation

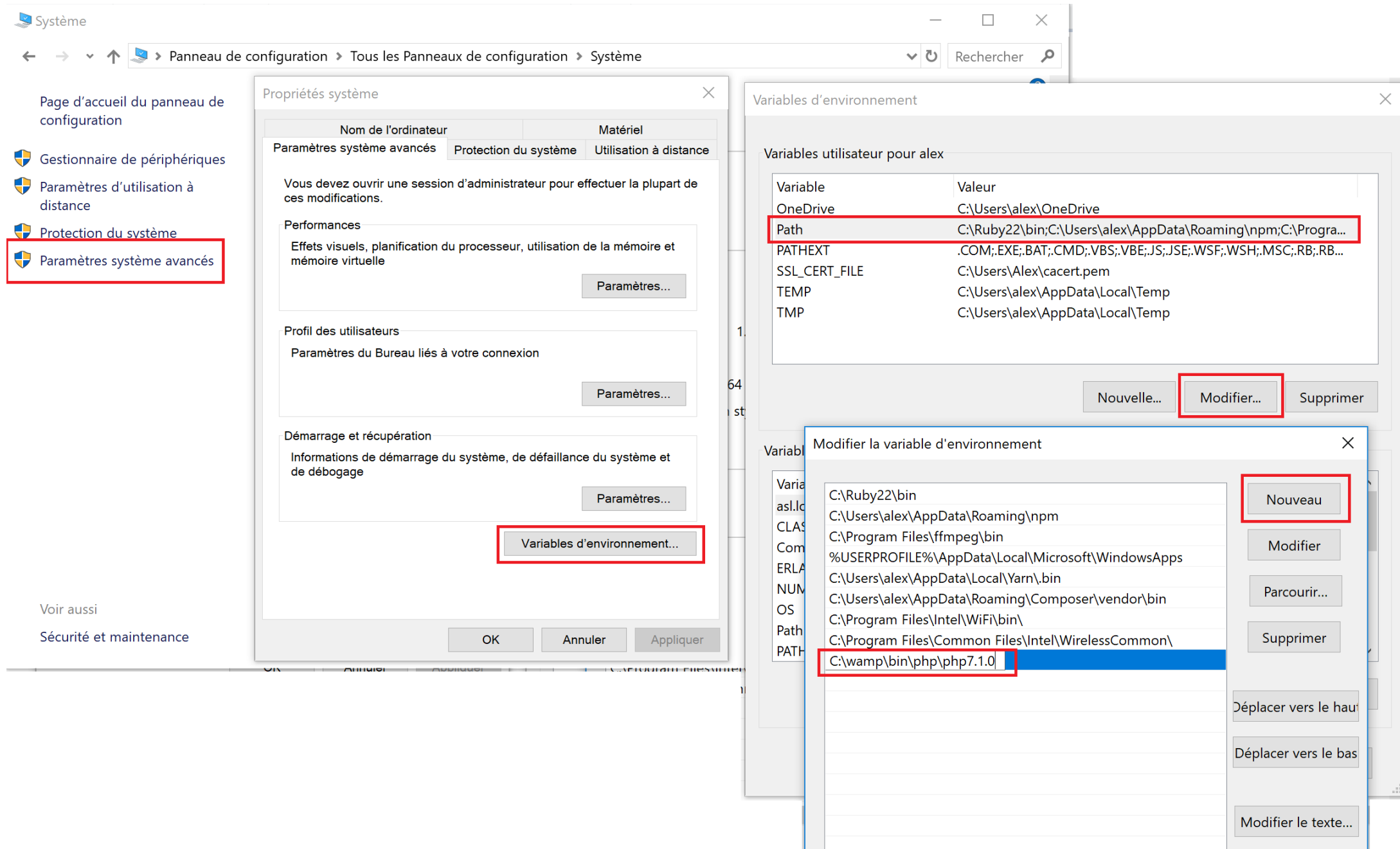
# Console (CLI)

- Shell (UNIX) / CMD (Windows)
- Alternative à l'interface graphique (GUI)
- Interface "bas niveau" avec l'OS permettant :
  - d'installer des paquets / applications
  - d'effectuer des opérations sur le système de fichiers
  - globalement d'être plus efficace en tant que développeur
- PHP : tests rapides possibles

# Installation de PHP

- Dans la console, exécuter `php -v`
  - Si OK : php installé
  - Sinon : il faut installer PHP
- Linux : `$ apt-get install php`
- Mac OSX : `$ brew install php`
- Windows : installer manuellement (via WAMP par ex.) puis ajouter à la variable d'environnement `path`
  - Démarrer > Panneau de configuration > Système et sécurité > Système > Paramètres système avancés

# Installation de PHP : Windows



# Composer

- Gestionnaire de dépendances
- Libre
- Écrit en PHP
- ~ NPM pour Node.js / ~ bundler pour Ruby



# Composer : commandes

- `require` : ajoute la bibliothèque en paramètre au fichier `composer.json` et l'installe.
- `install` : installe toutes les bibliothèques du `composer.json`. Il s'agit de la commande à lancer pour installer les dépendances d'un dépôt PHP.
- `update` : met à jour les bibliothèques du `composer.json`, selon les versions permises qui y sont mentionnées.
- `remove` : désinstalle une bibliothèque et la retire du `composer.json`.

# Composer : installation

- Windows : [getcomposer.org/Composer-Setup.exe](http://getcomposer.org/Composer-Setup.exe)
- UNIX : entrer la commande suivante dans la console

```
$ php -r "eval('?'>.file_get_contents('http://getcomposer.org/installer'));"
```

- Pour l'installer globalement :
  - Windows : ajouter un lien dans le path
  - UNIX :

```
$ sudo mv composer.phar /usr/local/bin/composer
```

# Git

- Gestionnaire de version décentralisé
- Libre
- Créé par Linus Torvalds
- En CLI mais possibilité d'utiliser une GUI
- Alternatives : SVN, CVS (mais Git le plus utilisé)
- Nécessaire à composer pour installer les dépendances

# Git : commandes

- `git init` : crée un nouveau dépôt
- `git clone` : clone un dépôt distant
- `git add` : ajoute les nouveaux fichiers
- `git commit` : "sauvegarde" l'état actuel des fichiers
- `git push` : publie les modifications à distance
- `git pull` : récupère les dernières modifications distantes du projet et les fusionne avec le projet actuel

# Git : installation

- Windows : <https://gitforwindows.org>
- Linux : `$ sudo apt-get install git-core`
- Mac OSX : `$ sudo brew install git-core`

# Symfony : installation

- Se placer dans le répertoire de travail
- Lancer la commande :

```
$ composer create-project symfony/skeleton mon-projet
```

# Symfony : lancer le serveur

- Soit on installe Symfony dans un répertoire déjà pointé par un serveur existant
  - ex: pour WAMP C : \wamp\www>
- Soit on utilise le serveur de développement intégré à Symfony (ne pas utiliser en production)

```
$ composer require symfony/web-server-bundle
```

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

# Symfony : lancer le serveur

## Welcome to Symfony 4.3.2



Your application is now ready. You can start working on it at:

`/Users/pierreportejoie/Desktop/IPI/Dev/mon-projet/`

### What's next?



Read the documentation to learn

[How to create your first page in Symfony](#)

You're seeing this page because debug mode is enabled and you haven't configured any homepage URL.



# IDE : PhpStorm

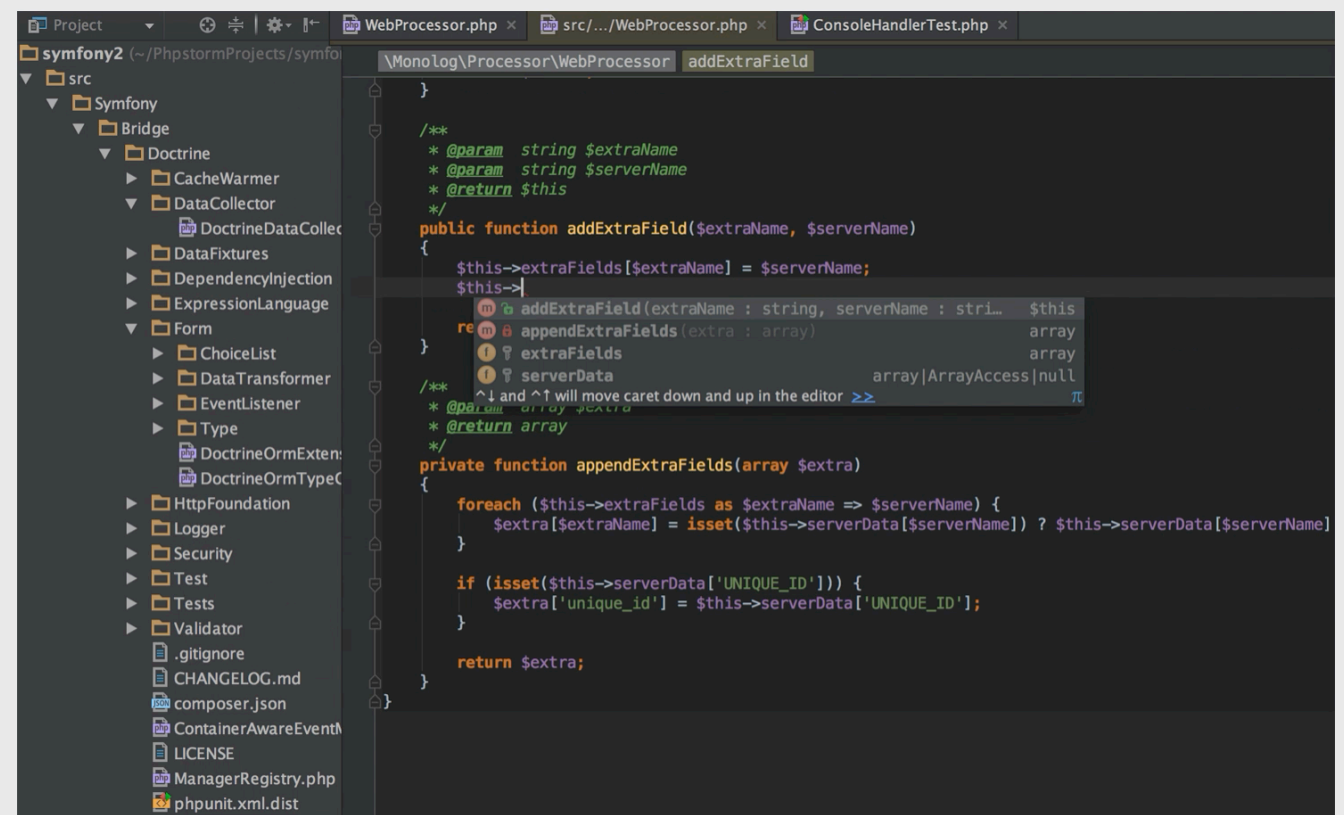
PhpStorm is perfect for working with Symfony, Laravel, Drupal, WordPress, Zend Framework, Magento, Joomla!, CakePHP, Yii, and other frameworks.

## All the PHP tools

The editor actually 'gets' your code and deeply understands its structure, supporting all the PHP language features for modern and legacy projects. It provides the best code completion, refactorings, on-the-fly error prevention, and more.

## Front-end technologies included

Make the most of the cutting edge front-end technologies, such as HTML 5, CSS, Sass, Less, Stylus, CoffeeScript, TypeScript, Emmet, and JavaScript, with refactorings, debugging, and unit testing available. See the changes instantly in the



III/ Première page

# Afficher une première page

- Route : l'URL (par ex. `/about`) de la page pointant vers le contrôleur
- Contrôleur : Fonction PHP permettant de construire la page

# Afficher une première page - contrôleur

```
<?php
// src/Controller/LuckyController.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>'
        );
    }
}
```

# Afficher une première page

- Route

```
# config/routes.yml
```

```
app_lucky_number:
```

```
  path: /lucky/number
```

```
  controller: App\Controller\LuckyController::number
```

- La page est maintenant disponible

<http://localhost:8000/lucky/number>

# Afficher une première page

- Il est possible de définir les routes directement dans le contrôleur, via des annotations

composer require annotations

```
// src/Controller/LuckyController.php
```

```
// ...
```

```
use Symfony\Component\Routing\Annotation\Route;
```

```
class LuckyController
```

```
{
```

```
    /**
```

```
     * @Route("/lucky/number")
```

```
     */
```

```
    public function number()
```

```
    {
```

```
        // this looks exactly the same
```

```
    }
```

```
}
```

# Afficher une première page

- Plutôt que de renvoyer du HTML en dur dans la réponse, on peut demander le rendu d'un template

```
// src/Controller/LuckyController.php

// ...
class LuckyController extends AbstractController
{
    /**
     * @Route("/lucky/number")
     */
    public function number()
    {
        $number = random_int(0, 100);

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

# Afficher une première page

- Les templates sont présents dans le répertoire  
templates/
- La syntaxe `{{ number }}` permet d'afficher des variables

```
{# templates/lucky/number.html.twig #}  
<h1>Your lucky number is {{ number }}</h1>
```



# Afficher une première page

- Affichage de la page :

<http://localhost:8000/lucky/number>

- La barre d'outils a disparu car il n'y a pas de balise `<body>`

# Route - wildcards (\*)

```
// src/Controller/BlogController.php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class BlogController extends AbstractController
{
    /**
     * Matches /blog exactly
     *
     * @Route("/blog", name="blog_list")
     */
    public function list()
    { // ... }

    /**
     * Matches /blog/*
     * but not /blog/slug/extra-part
     *
     * @Route("/blog/{slug}", name="blog_show")
     */
    public function show($slug)
    { // ... }
}
```

# Routage - wildcards (\*)

- Une requête sur `/blog` lancera l'exécution de la fonction `list()`
- Une requête sur `/blog/exemple` lancera l'exécution de la fonction `show('exemple')`
- Une requête sur `/blog/foo/bar` ne fonctionnera pas
  - le slash `'/'` est exclu par défaut de la chaîne de caractères envoyée en argument
  - Il faudrait créer une nouvelle route du type `/blog/{slug}/{option}`

# Routage - wildcards (\*)

- Problème : un blog qui renvoie
  - une liste d'articles paginée sur des url du type `/blog/{page}` (ex: `/blog/2` pour la 2e page de la liste)
  - les articles sur des url du type `/blog/{slug}` (ex: `/blog/mon-article` pour voir l'article "Mon article")
- Les 2 types de routes correspondent à `/blog/*`

# Routage - requirements

- Solution : ajouter des *requirements* via des expressions régulières

```
class BlogController extends AbstractController
{
    /**
     * @Route("/blog/{page}", name="blog_list", requirements={"page"="\d+"})
     */
    public function list($page)
    {    // ... }

    /**
     * @Route("/blog/{slug}", name="blog_show")
     */
    public function show($slug)
    {    // ... }
}
```

- Pour tester des expressions régulières : <https://regexpr.com>

# Routage - requirements

## - Syntaxe alternative

```
class BlogController extends AbstractController
{
    /**
     * @Route("/blog/{page}", name="blog_list", requirements={"page"="\d+"})
     */
    public function list($page)
    {
        // ...
    }
}
```

# Routage - valeurs par défaut

- Valeur par défaut (ex: on veut que `/blog/1` renvoie la même page que `/blog`)

```
class BlogController extends AbstractController
{
    /**
     * @Route("/blog/{page}", name="blog_list", requirements={"page"="\d+"})
     */
    public function list($page = 1)
    {
        // ...
    }
}
```

- Pour forcer l'url à `/blog/1` : `/blog/{!page}`

# Contrôleur

- Fonction lisant les informations d'un objet `request` et renvoyant un objet `response`
- Ici le contrôleur est la fonction `number()`

```
// src/Controller/LuckyController.php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class LuckyController
{
    /**
     * @Route("/lucky/number/{max}", name="app_lucky_number")
     */
    public function number($max)
    {
        $number = random_int(0, $max);

        return new Response(
            '<html><body>Lucky number: '.$number.'</body></html>'
        );
    }
}
```



# Contrôleur : redirection

```
use Symfony\Component\HttpFoundation\RedirectResponse;

// ...
public function index()
{
    // redirects to the "homepage" route
    return $this->redirectToRoute('homepage');

    // does a permanent - 301 redirect
    return $this->redirectToRoute('homepage', [], 301);

    // redirect to a route with parameters
    return $this->redirectToRoute('app_lucky_number', ['max' => 10]);

    // redirects to a route and maintains the original query string parameters
    return $this->redirectToRoute('blog_show', $request->query->all());

    // redirects externally
    return $this->redirect('http://symfony.com/doc');
}
```

# Contrôleur - utiliser un template

- La méthode `render()` permet d'afficher une page HTML dynamique

```
// renders templates/lucky/number.html.twig  
return $this->render('lucky/number.html.twig', ['number' => $number]);
```

# Contrôleur - erreurs HTTP

- Codes HTTP courants :
  - 200 : succès de la requête
  - 301 : redirection permanente
  - 302 : redirection temporaire
  - 401 : utilisateur non authentifié
  - 403 : accès refusé
  - 404 : page non trouvée
  - 5XX : erreurs serveur

# Contrôleur - erreurs HTTP

- Le contrôleur peut renvoyer ces erreurs grâce aux fonctions `create{exception}()` (ex : `createNotFoundHttpException()`)
- Il est aussi possible de renvoyer des exceptions "basiques" auquel cas le contrôleur enverra une réponse avec le statut HTTP 500

- [AccessDeniedHttpException.php](#)
- [BadRequestHttpException.php](#)
- [ConflictHttpException.php](#)
- [ControllerDoesNotReturnRespons...](#)
- [GoneHttpException.php](#)
- [HttpException.php](#)
- [HttpExceptionInterface.php](#)
- [LengthRequiredHttpException.php](#)
- [MethodNotAllowedHttpException....](#)
- [NotAcceptableHttpException.php](#)
- [NotFoundHttpException.php](#)
- [PreconditionFailedHttpException....](#)
- [PreconditionRequiredHttpExcepi...](#)
- [ServiceUnavailableHttpException....](#)
- [TooManyRequestsHttpException....](#)
- [UnauthorizedHttpException.php](#)
- [UnprocessableEntityHttpExceptio...](#)
- [UnsupportedMediaTypeHttpExce...](#)

# Contrôleur - erreurs HTTP

## - Envoi d'une réponse avec statut HTTP 404

```
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;

// ...
public function index()
{
    // retrieve the object from database
    $product = ...;
    if (!$product) {
        throw $this->createNotFoundException('The product does not exist');

        // the above is just a shortcut for:
        // throw new NotFoundHttpException('The product does not exist');
    }

    return $this->render(...);
}
```

# Contrôleur - erreurs HTTP

- Envoi d'une réponse "basique"

```
// this exception ultimately generates a 500 status error  
throw new \Exception('Something went wrong!');
```

- Dans tous les cas, côté utilisateur, on aura :
  - Une page d'erreur en mode production
  - Une page d'erreur avec informations de débog en développement
- Il est possible de customiser ses propres pages d'erreur

# Contrôleur - récupération de paramètres

- Paramètres HTTP :

`site.com/request?param1=arg1&param2=arg2`

- Il est possible de récupérer les paramètres via l'objet Request

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

```
public function index(Request $request, $firstName, $lastName)
{
    $page = $request->query->get('param1');

    // ...
}
```

# Contrôleur - objet request

- L'objet `request` permet de récupérer toute disponibles dans la requête HTTP

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

public function index(Request $request)
{
    $request->isXmlHttpRequest(); // is it an Ajax request?

    $request->getPreferredLanguage(['en', 'fr']);

    // retrieves GET and POST variables respectively
    $request->query->get('page');
    $request->request->get('page');

    // retrieves SERVER variables
    $request->server->get('HTTP_HOST');

    // retrieves an instance of UploadedFile identified by foo
    $request->files->get('foo');

    // retrieves a COOKIE value
    $request->cookies->get('PHPSESSID');

    // retrieves an HTTP request header, with normalized, lowercase keys
    $request->headers->get('host');
    $request->headers->get('content-type');
}
```



# Contrôleur

- Un contrôleur accepte une requête et renvoie un objet `response`
- Symfony fournit un contrôleur *abstrait* `AbstractController` comprenant des fonctions usuelles telles que :
  - `render()`
  - `redirectToRoute()`
  - `createNotFoundException()`

# Templates

- Un template est un fichier texte permettant de générer du contenu dynamique
- En PHP classique, on utilise de simples fichiers .php
- Avec Symfony, on peut utiliser un langage de templating : Twig (fichiers .html.twig)

# Templates - Exemple PHP

```
<html>
  <head>
    <title>Welcome to Symfony!</title>
  </head>
  <body>
    <h1><?= $page_title ?></h1>

    <ul id="navigation">
      <?php foreach ($navigation as $item): ?>
        <li>
          <a href="<?= $item->getHref() ?>">
            <?= $item->getCaption() ?>
          </a>
        </li>
      <?php endforeach ?>
    </ul>
  </body>
</html>
```

# Templates - Example Twig

```
<html>
  <head>
    <title>Welcome to Symfony!</title>
  </head>
  <body>
    <h1>{{ page_title }}</h1>

    <ul id="navigation">
      {% for item in navigation %}
        <li>
          <a href="{{ item.href }}">
            {{ item.caption }}
          </a>
        </li>
      {% endfor %}
    </ul>
  </body>
</html>
```

# Templates - Twig

- Twig permet de séparer le code de l'affichage
- Le code est géré par le contrôleur qui passe des arguments
- Le fichier .html.twig gère uniquement l'affichage

# Templates - Twig

- Twig utilise 3 types de syntaxe :
  - `{ { ... } }` : "écrit" quelque chose (`print` une variable)
  - `{ % ... % }` : "fait" quelque chose (structures de contrôle, boucles, ...)
  - `{ # ... # }` : "commente" quelque chose (équivalent de PHP `/* comment */`)

# Templates - Twig

- Twig comprend des *filtres*, permettant de modifier du contenu avant de l'afficher

```
{{ title }}           // affiche : My title  
{{ title|upper }}     // affiche : MY TITLE
```

- Twig comprend aussi un grand nombre de functions

```
{% for i in 1..10 %}  
    <div class="{ cycle(['even', 'odd'], i) }">  
        <!-- some HTML here -->  
    </div>  
{% endfor %}
```

# Templates - Twig

- Exemple de boucle avec condition :

```
<ul>
    {% for user in users if user.active %}
        <li>{{ user.username }}</li>
    {% else %}
        <li>No users found</li>
    {% endfor %}
</ul>
```



# Twig - Héritage

- Tous les templates d'un projet partagent souvent des éléments communs (header, footer, sidebar, ...)
- Symfony permet de construire un template "de base" comprenant tous ces éléments qui seront ensuite redéfinis par ses enfants
- Un enfant étend le template parent et redéfinis (*override*) les blocs qui le concernent

# Twig - Héritage

## - Template parent

```
{# templates/base.html.twig #}

<html>
  <head>
    <meta charset="UTF-8">
    <title>{% block title %}Test Application{% endblock %}</title>
  </head>
  <body>
    <div id="sidebar">
      {% block sidebar %}
        <ul>
          <li><a href="/">Home</a></li>
          <li><a href="/blog">Blog</a></li>
        </ul>
      {% endblock %}
    </div>

    <div id="content">
      {% block body %}{% endblock %}
    </div>
  </body>
</html>
```

# Twig - Héritage

## - Template enfant

```
{# templates/blog/index.html.twig #}
{% extends 'base.html.twig' %}

{% block title %}My cool blog posts{% endblock %}

{% block body %}
    {% for entry in blog_entries %}
        <h2>{{ entry.title }}</h2>
        <p>{{ entry.body }}</p>
    {% endfor %}
{% endblock %}
```

# Twig - Héritage

- Résultat possible de l'appel du template enfant

```
<html>
  <head>
    <meta charset="UTF-8">
    <title>My cool blog posts</title>
  </head>
  <body>
    <div id="sidebar">
      <ul>
        <li><a href="/">Home</a></li>
        <li><a href="/blog">Blog</a></li>
      </ul>
    </div>

    <div id="content">
      <h2>My first post</h2>
      <p>The body of the first post.</p>

      <h2>Another post</h2>
      <p>The body of the second post.</p>
    </div>
  </body>
</html>
```

- Le contenu d'un bloc n'ayant pas été redéfini reste celui du parent

# Twig - Héritage : Règles

- `{% extends %}` doit être la première balise du template
- Il est souvent souhaitable d'avoir de nombreux `{% block %}` dans le parent
- En cas de duplication de contenu entre les enfants, cela peut signifier qu'on pourrait utiliser un `{% block %}` supplémentaire dans le parent
- Pour ré-utiliser le contenu du parent, il est possible d'utiliser la fonction `{{ parent() }}`

# Twig - Balises et helpers

## - Inclusion d'autres templates

```
{# templates/article/article_details.html.twig #}
```

```
<h2>{{ article.title }}</h2>
```

```
<h3 class="byline">by {{ article.authorName }}</h3>
```

```
<p>
```

```
    {{ article.body }}
```

```
</p>
```

```
{# templates/article/list.html.twig #}
```

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

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

```
    <h1>Recent Articles</h1>
```

```
    {% for article in articles %}
```

```
        {{ include('article/article_details.html.twig', { 'article': article }) }}
```

```
    {% endfor %}
```

```
{% endblock %}
```

# Twig - Balises et helpers

- Lien vers une page
- Contrôleur :

```
class WelcomeController extends AbstractController
{
    /**
     * @Route("/", name="welcome", methods={"GET"})
     */
    public function index()
    {
        // ...
    }
}
```

- Template :

```
<a href="{{ path('welcome') }}">Home</a>
```

# Twig - Balises et helpers

- Lien vers une page, cas plus complexe
- Contrôleur :

```
class ArticleController extends AbstractController
{
    /**
     * @Route("/article/{slug}", name="article_show", methods={"GET"})
     */
    public function show($slug)
    {
        // ...
    }
}
```

- Template :

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



# Twig - Balises et helpers

- Lien vers les ressources : images
- Installation du package asset

```
composer require symfony/asset
```

- Utilisation :

```

```

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

# Twig - Balises et helpers

- Lien vers les ressources : CSS & JS, On peut utiliser l'héritage
- Template parent :

```
{# templates/base.html.twig #}
```

```
<html>
```

```
  <head>
```

```
    {# ... #}
```

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

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

```
    {% endblock %}
```

```
  </head>
```

```
  <body>
```

```
    {# ... #}
```

```
    {% block javascripts %}
```

```
      <script src="{{ asset('js/main.js') }}"></script>
```

```
    {% endblock %}
```

```
  </body>
```

```
</html>
```

# Twig - Balises et helpers

- Lien vers les ressources : CSS & JS
- Template enfant :

```
{# templates/contact/contact.html.twig #}
{% extends 'base.html.twig' %}

{% block stylesheets %}
    {{ parent() }}

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

{# ... #}
```

# Twig - aide-mémoire

Description	PHP	Twig
Afficher une variable	<code>&lt;?php echo \$pseudo; ?&gt;</code>	<code>{{ pseudo }}</code>
Afficher l'index d'un tableau	<code>&lt;?php echo \$user['id']; ?&gt;</code>	<code>{{ user['id'] }}</code> ou <code>{{ user.id }}</code>
Afficher l'attribut d'un objet, dont le getter respecte la convention \$objet->getAttribut()	<code>&lt;?php echo \$user-&gt;getId(); ?&gt;</code>	<code>{{ user.id }}</code>
Afficher une variable en lui appliquant un filtre. Ici, passer le texte en majuscules	<code>&lt;?php echo strtoupper(\$pseudo); ?&gt;</code>	<code>{{ pseudo upper }}</code>
Afficher une variable en lui appliquant un filtre. Ici, supprimer les balises HTML puis passage des premières lettres en majuscules.	<code>&lt;?php echo ucwords(strip_tags(\$news-&gt;getTexte())); ?&gt;</code>	<code>{{ news.texte striptags title }}</code>
Utiliser un filtre avec des arguments. (date est ici de type datetime)	<code>&lt;?php echo \$date-&gt;format('d/m/Y'); ?&gt;</code>	<code>{{ date date('d/m/Y') }}</code>
Concaténer	<code>&lt;?php echo \$nom.' '.\$prenom; ?&gt;</code>	<code>{{ nom ~ " " ~ prenom }}</code>

# Twig - aide-mémoire

Variable	Description
<code>{{ loop.index }}</code>	Le numéro de l'itération courante (en commençant par 1).
<code>{{ loop.index0 }}</code>	Le numéro de l'itération courante (en commençant par 0).
<code>{{ loop.revindex }}</code>	Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 1).
<code>{{ loop.revindex0 }}</code>	Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 0).
<code>{{ loop.first }}</code>	true si c'est la première itération, false sinon.
<code>{{ loop.last }}</code>	true si c'est la dernière itération, false sinon.
<code>{{ loop.length }}</code>	Le nombre total d'itérations dans la boucle.

# Twig - aide-mémoire

Twig	PHP	
Capitalize	ucfirst	
Date	Date	
Number_format	Number_format	
Lower	Strtolower	
Upper	Strtoupper	
Sort	Sort	
Split	Explode	
Raw	??	
Slice	Substr	
Replace	strreplace	

III/ BDD

# BDD

- On a vu comment appeler des vues (*Controller*) et les afficher avec des Templates (*View*)
- Reste la partie "M" de MVC : *Model*
- Les données sont persistées en BDD et accédées via la partie *Model*
- Symfony utilise la librairie Doctrine (ORM) pour communiquer avec la BDD



# ORM

- ORM : Object-Relational mapping
- Couche d'abstraction entre la base de données et un système orienté objet
- Pas besoin de concevoir manuellement la BDD
- Pas besoin de requêter directement la BDD
- On ne travaille qu'avec des objets, l'ORM se charge de créer, persister, modifier, supprimer, etc. les entrées de la BDD

# Doctrine : installation

## - Installation des paquets

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

## - Configuration

```
# .env  
  
# customize this line!  
DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name"
```

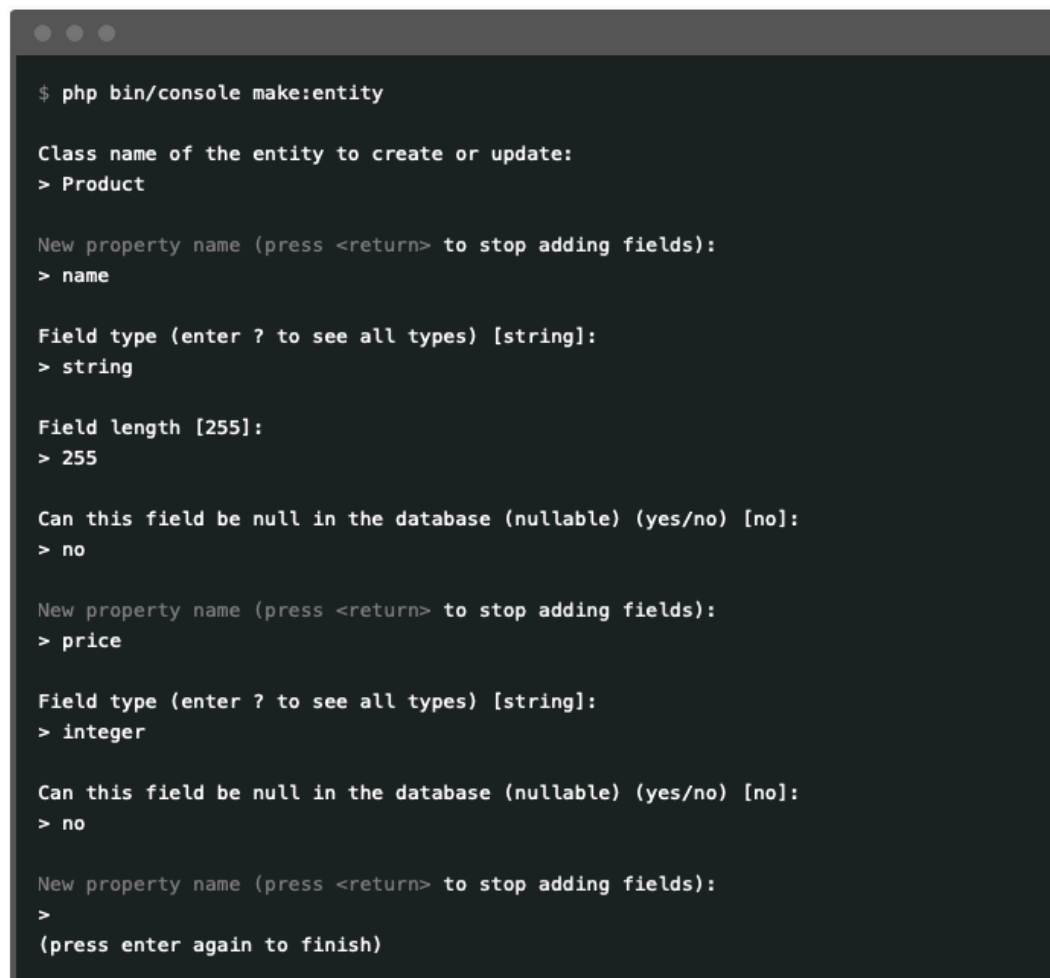
## - Lancement de la création

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

# Doctrine : Entités

- Entité : classe correspondant à la table que l'on veut créer en BDD

```
$ php bin/console make:entity
```



```
$ php bin/console make:entity

Class name of the entity to create or update:
> Product

New property name (press <return> to stop adding fields):
> name

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 255

Can this field be null in the database (nullable) (yes/no) [no]:
> no

New property name (press <return> to stop adding fields):
> price

Field type (enter ? to see all types) [string]:
> integer

Can this field be null in the database (nullable) (yes/no) [no]:
> no

New property name (press <return> to stop adding fields):
>
(press enter again to finish)
```

# Doctrine : Entités

- Un nouveau fichier correspondant à l'entité a été créé : `src/Entity/Product.php`

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\ProductRepository")
 */
class Product
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $name;

    /**
     * @ORM\Column(type="integer")
     */
    private $price;

    public function getId()
    {
        return $this->id;
    }

    // ... getter and setter methods
}
```

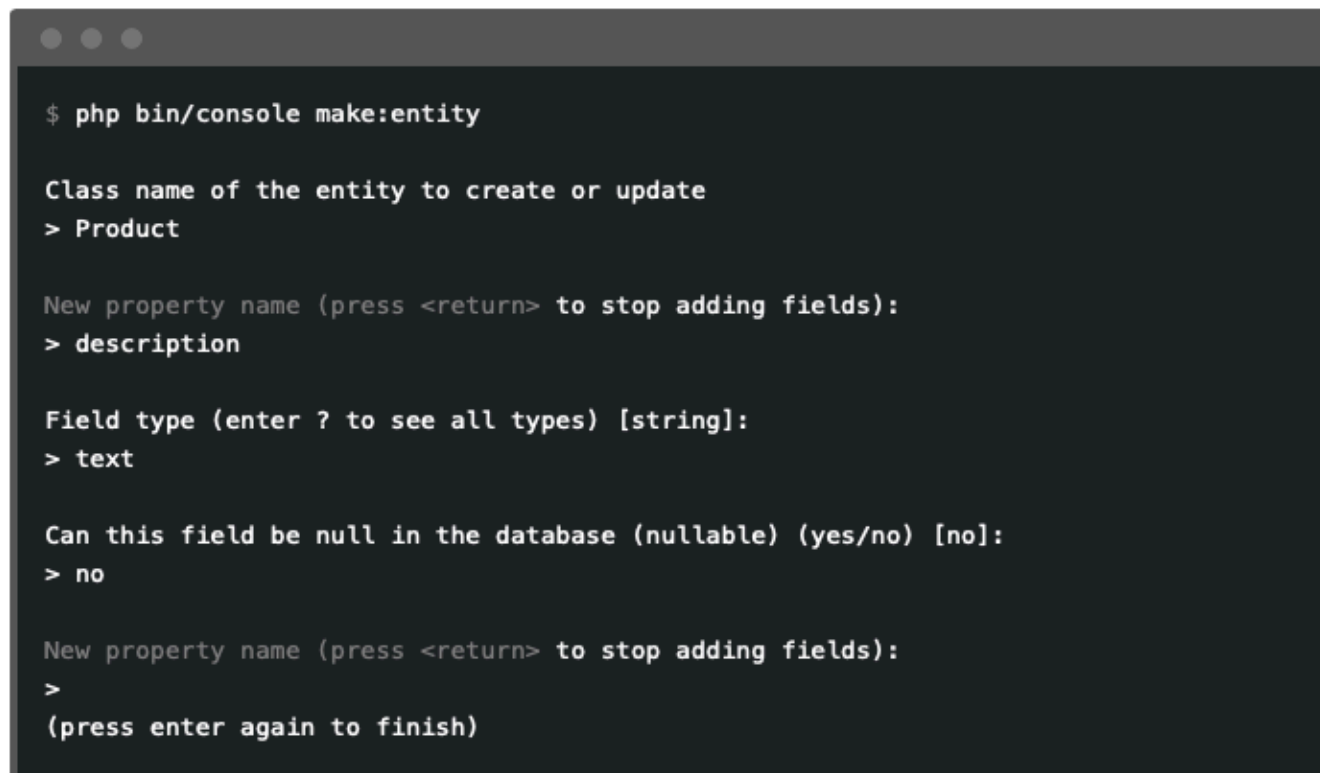
# Doctrine : Entités

- La commande `make:entity` crée le fichier mais il s'agit simplement d'une aide
- On peut ajouter / supprimer des champs et méthodes en fonction des besoins
- La BDD ne comprend pas encore la table créée, pour cela on va générer un fichier `.php`, dit de *migrations*, puis l'exécuter

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

# Doctrine : Entités

- Pour ajouter de nouveaux champs à l'Entité on peut :
  - Modifier le fichier à la main
  - Utiliser à nouveau `make:entity`



```
$ php bin/console make:entity

Class name of the entity to create or update
> Product

New property name (press <return> to stop adding fields):
> description

Field type (enter ? to see all types) [string]:
> text

Can this field be null in the database (nullable) (yes/no) [no]:
> no

New property name (press <return> to stop adding fields):
>
(press enter again to finish)
```

- Pour ajouter la nouvelle propriété à la table :

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

# Doctrine : Entités

```
// src/Controller/ProductController.php
namespace App\Controller;

// ...
use App\Entity\Product;

class ProductController extends Controller
{
    /**
     * @Route("/product", name="product")
     */
    public function index()
    {
        // you can fetch the EntityManager via $this->getDoctrine()
        // or you can add an argument to your action: index(EntityManagerInterface $entityManager)
        $entityManager = $this->getDoctrine()->getManager();

        $product = new Product();
        $product->setName('Keyboard');
        $product->setPrice(1999);
        $product->setDescription('Ergonomic and stylish!');

        // tell Doctrine you want to (eventually) save the Product (no queries yet)
        $entityManager->persist($product);

        // actually executes the queries (i.e. the INSERT query)
        $entityManager->flush();

        return new Response('Saved new product with id '.$product->getId());
    }
}
```

# Doctrine : Entités

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

- Chargement de l'*entity manager*, objet permettant d'enregistrer et récupérer les objets depuis la BDD

```
$product = new Product();  
$product->setName('Keyboard');  
$product->setPrice(1999);  
$product->setDescription('Ergonomic and stylish!');
```

- Instantiation et modification de l'objet `$product`

```
$entityManager->persist($product);
```

- Préparation à l'enregistrement de l'objet en BDD

```
$entityManager->flush();
```

- Envoi de la commande en BDD



# Doctrine : Entités

- Pour tester l'insertion des données :

<http://localhost:8000/product>

- Pour vérifier l'insertion en BDD :

```
$ php bin/console doctrine:query:sql 'SELECT * FROM product'
```

# Doctrine : Entités

## - Récupération d'objets (fetch)

```
// src/Controller/ProductController.php
// ...

/**
 * @Route("/product/{id}", name="product_show")
 */
public function show($id)
{
    $product = $this->getDoctrine()
        ->getRepository(Product::class)
        ->find($id);

    if (!$product) {
        throw $this->createNotFoundException(
            'No product found for id '.$id
        );
    }

    return new Response('Check out this great product: '.$product->getName());

    // or render a template
    // in the template, print things with {{ product.name }}
    // return $this->render('product/show.html.twig', ['product' => $product]);
}
```

# Doctrine : Entités

- Repository : classe permettant de récupérer les entités d'une certaine classe
- Donne accès à plusieurs méthodes

```
$repository = $this->getDoctrine()->getRepository(Product::class);

// look for a single Product by its primary key (usually "id")
$product = $repository->find($id);

// look for a single Product by name
$product = $repository->findOneBy(['name' => 'Keyboard']);
// or find by name and price
$product = $repository->findOneBy([
    'name' => 'Keyboard',
    'price' => 1999,
]);

// look for multiple Product objects matching the name, ordered by price
$products = $repository->findBy(
    ['name' => 'Keyboard'],
    ['price' => 'ASC']
);

// look for *all* Product objects
$products = $repository->findAll();
```

# Doctrine : Entités

- Queries automatiques avec SensioFrameworkExtraBundle

```
$ composer require sensio/framework-extra-bundle
```

- Utilisation :

```
// src/Controller/ProductController.php

use App\Entity\Product;
// ...

/**
 * @Route("/product/{id}", name="product_show")
 */
public function show(Product $product)
{
    // use the Product!
    // ...
}
```

# Doctrine : Entités - mise à jour d'un objet

```
/**
 * @Route("/product/edit/{id}")
 */
public function update($id)
{
    $entityManager = $this->getDoctrine()->getManager();
    $product = $entityManager->getRepository(Product::class)->find($id);

    if (!$product) {
        throw $this->createNotFoundException(
            'No product found for id '.$id
        );
    }

    $product->setName('New product name!');
    $entityManager->flush();

    return $this->redirectToRoute('product_show', [
        'id' => $product->getId()
    ]);
}
```

# Doctrine : Entités - suppression d'un objet

```
$entityManager->remove($product);  
$entityManager->flush();
```

- La méthode `remove()` ne supprime pas directement l'objet en BDD
- La méthode `flush()` exécute la suppression

# Doctrine : Entités - utilisation de SQL

```
// src/Repository/ProductRepository.php  
// ...  
  
public function findAllGreaterThanPrice($price): array  
{  
    $conn = $this->getEntityManager()->getConnection();  
  
    $sql = '  
        SELECT * FROM product p  
        WHERE p.price > :price  
        ORDER BY p.price ASC  
    ';  
    $stmt = $conn->prepare($sql);  
    $stmt->execute(['price' => $price]);  
  
    // returns an array of arrays (i.e. a raw data set)  
    return $stmt->fetchAll();  
}
```

- Attention : uniquement les données sont renvoyées (pas d'objet instancié)

# IV/ Formulaire



# Formulaires

- La construction d'un formulaire est souvent une étape compliquée dans la création d'une application web
- Grande force de Symfony : simplification de ce processus
- Installation :

```
$ composer require symfony/form
```

# Formulaires : création d'un formulaire simple

- On veut construire un formulaire permettant de créer une application de type "TODO List"
- Avant toute chose on va créer une entité "tâche"
- On reliera ensuite cette entité à un formulaire
- L'entrée et la validation du formulaire insèrera un nouvel objet en BDD

# Formulaires : création d'un formulaire simple

## - Création de l'entité

```
// src/Entity/Task.php
namespace App\Entity;

class Task
{
    protected $task;
    protected $dueDate;

    public function getTask()
    {
        return $this->task;
    }

    public function setTask($task)
    {
        $this->task = $task;
    }

    public function getDueDate()
    {
        return $this->dueDate;
    }

    public function setDueDate(\DateTime $dueDate = null)
    {
        $this->dueDate = $dueDate;
    }
}
```

# Formulaires : création d'un formulaire simple

## - Création du formulaire 1/2

```
// src/Controller/TaskController.php  
namespace App\Controller;  
  
use App\Entity\Task;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\Form\Extension\Core\Type\DateType;  
use Symfony\Component\Form\Extension\Core\Type\SubmitType;  
use Symfony\Component\Form\Extension\Core\Type\TextType;  
use Symfony\Component\HttpFoundation\Request;
```

# Formulaires : création d'un formulaire simple

## - Création du formulaire 2/2

```
class TaskController extends AbstractController
{
    public function new(Request $request)
    {
        // creates a task and gives it some dummy data for this example
        $task = new Task();
        $task->setTask('Write a blog post');
        $task->setDueDate(new \DateTime('tomorrow'));

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

        return $this->render('task/new.html.twig', [
            'form' => $form->createView(),
        ]);
    }
}
```

# Formulaires : création d'un formulaire simple

- Affichage du formulaire dans le template

```
{# templates/task/new.html.twig #}  
{{ form(form) }}
```

- Résultat côté client :

Task

Due date

# Formulaires : création d'un formulaire simple

- Envoi du formulaire
- Par défaut, le formulaire est envoyé via une requête POST sur le contrôleur qui l'affiche
- Il faut donc gérer dans le même contrôleur la création du formulaire et la réception des données envoyées par celui-ci

# Formulaires : création d'un formulaire simple

```
public function new(Request $request)
{
    // just setup a fresh $task object (remove the dummy data)
    $task = new Task();

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

    $form->handleRequest($request);

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

        // do some actions with the data, save it, etc.

        return $this->redirectToRoute('task_success');
    }

    return $this->render('task/new.html.twig', [
        'form' => $form->createView(),
    ]);
}
```



# Formulaires : création d'un formulaire simple

- 3 possibilités pour l'exemple précédent :
- Absence de requête : le contrôleur va simplement afficher le formulaire
- Présence d'une requête non-valide : le contrôleur va afficher le formulaire et les erreurs de validation spécifiques
- Présence d'une requête valide : le contrôleur va effectuer une redirection vers la route choisie

# Formulaires : validation

- Installation

```
$ composer require symfony/validator
```

- La validation se fait au côté serveur, elle est définie au niveau de l'entité

```
use Symfony\Component\Validator\Constraints as Assert;
```

```
class Task
{
    /**
     * @Assert\NotBlank
     */
    public $task;

    /**
     * @Assert\NotBlank
     * @Assert\Type("\DateTime")
     */
    protected $dueDate;
}
```

# Formulaires : champs

- **Champs texte :**
  - TextType
  - TextareaType
  - EmailType
  - IntegerType
  - PasswordType
  - UrlType
  - ...
- **Champs dateTime :**
  - DateType
  - DateTimeType
  - TimeType
  - ...
- **Autres champs :**
  - CheckboxType
  - RadioType
  - HiddenType
  - ...