

Symphony 6

Introduction

AYMEN SELLAOUTI

C'est quoi symfony ?

« Symfony is a set of PHP Components, a Web Application framework, a Philosophy, and a Community — all working together in harmony. »
[site Officiel Symfony]

- ✓ *Ensemble de composants PHP*
- ✓ *Framework pour les applications web*
 - ✓ *Basée sur des composants*
 - ✓ *Structuration du code*
 - ✓ *Maintenabilité*

C'est quoi symfony ?

- ✓ Une philosophie
 - ✓ Les bonnes pratiques
 - ✓ standardisation
- ✓ *Une très grande communauté*

FrameWork : Cadre De Travail (Boite à outils)

Ensemble de composants servant à créer :

- Fondation
- Architecture

Pourquoi utiliser un Framework

Productivité : ensemble de composants déjà prêt à l'emploi

Propreté du code => maintenabilité et évolutivité

Basée sur une architecture => facilite le travail en équipe

Communauté et documentation

Installation de Symfony (1)

- Afin d'installer Symfony, vous devez disposer de Composer qui est un PHP Package Manager. <https://getcomposer.org/download/>
- Pré requis :
- PHP 8.1 ou plus, ceci dépendra de la version Symfony.
- Lancer la commande : `composer create-project symfony/website-skeleton nomProjet` pour une version d'un projet web qui contient les bibliothèques de bases dédiées à un projet web.
- Lancer la commande : `composer create-project symfony/skeleton nomProjet` pour une version orientée vers les microservices ou les API.

Installation de Symfony (2)

- Vous pouvez aussi utiliser le symfony client : <https://symfony.com/download>
- Installer tout d'abord scoop (pour windows) via ce lien <https://scoop.sh/>
- Lancer ensuite la commande `scoop install symfony-cli`
- Vous n'avez qu'à utiliser donc la commande `symfony new nomProjet` pour avoir votre projet.
- Vous allez avoir deux choix, la version Webapp qui contient l'ensemble des bibliothèques nécessaires pour la création d'une application ajouter l'option `--webapp`.
- La version minimaliste pour créer des api, des micro services ou une application desktop.

Installation de Symfony (3)

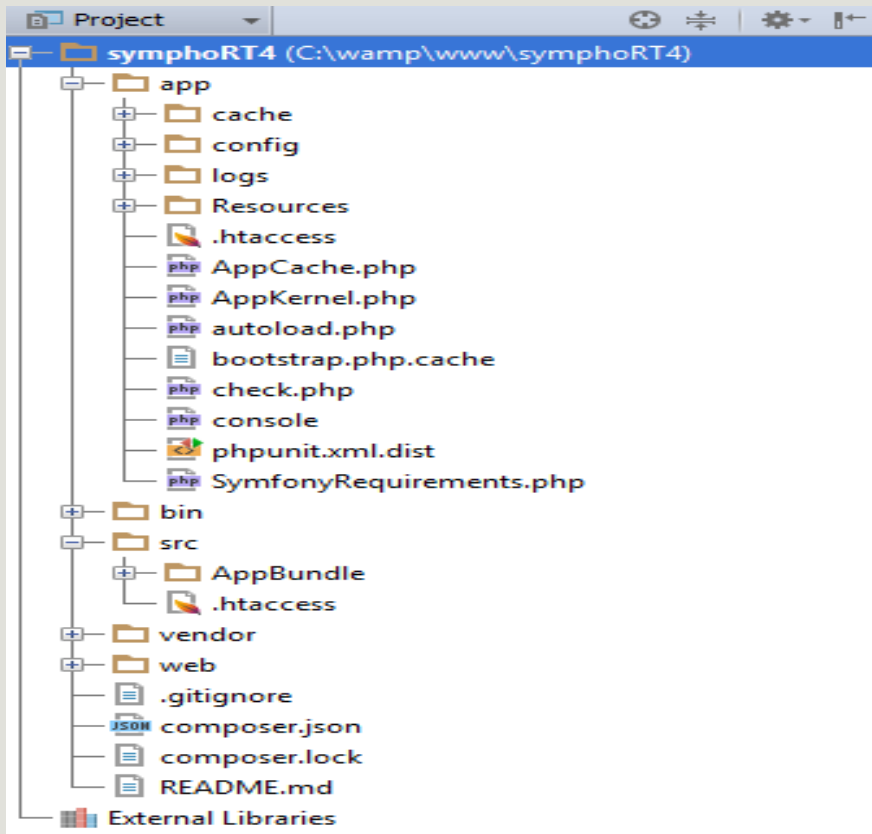
- `symfony new my_project_directory --version="6.1.*" --webapp`
- `symfony new my_project_directory --version="6.1.*"`
- `composer create-project symfony/skeleton:"6.1.*" my_project_directory`
- `composer create-project symfony/skeleton:"6.1.*" my_project_directory`

Symfony Roadmap

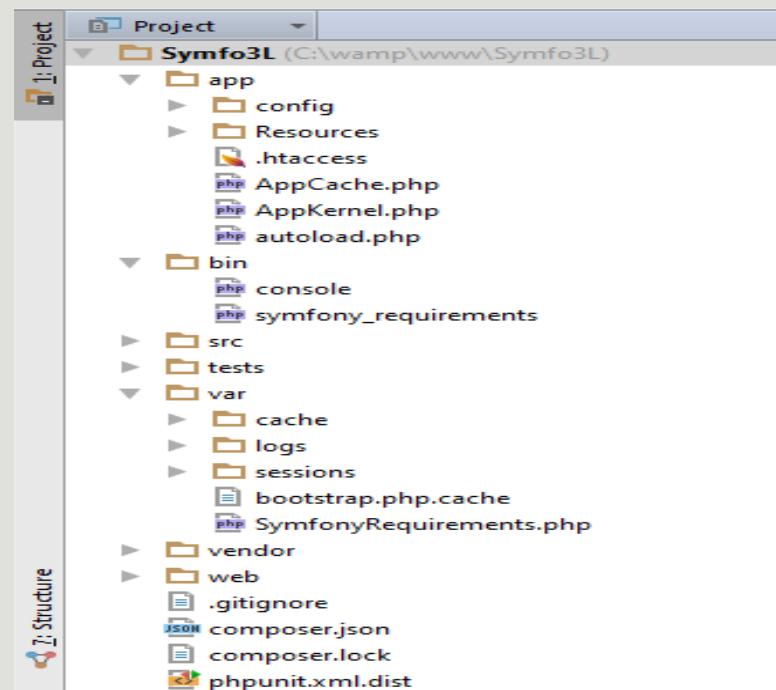
Vous pouvez vérifier les différentes roadmap des versions symfony.
Choisissez les versions stables LTS (Long Term Support)

<https://symfony.com/roadmap>

Architecture d'un projet Symfony 2.8 et 3.*



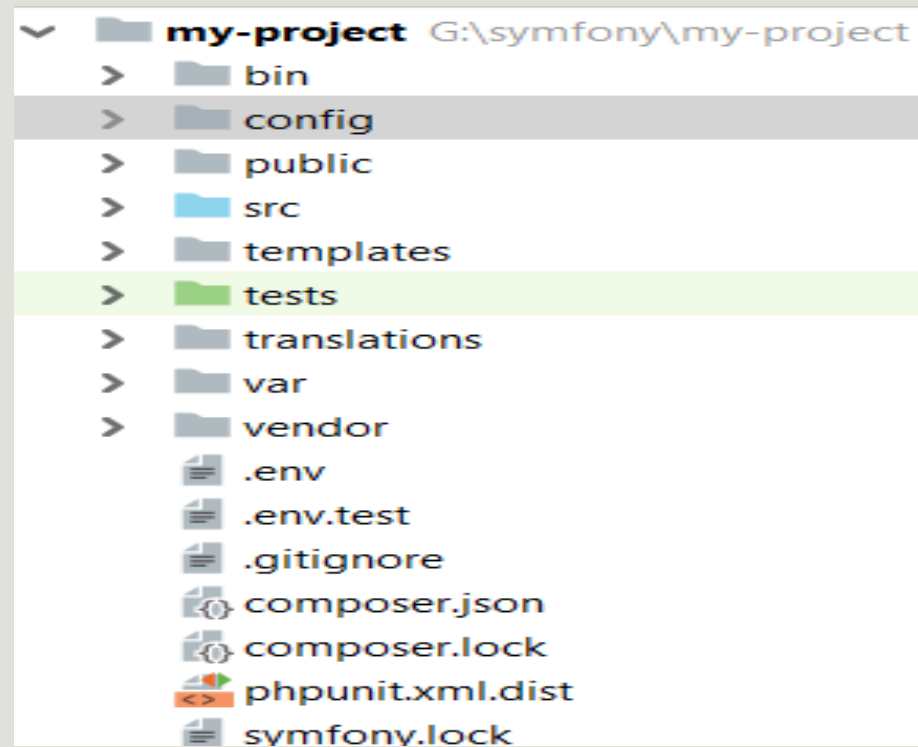
- app/ : La configuration de l'application,
- src/ : Le code PHP du projet,
- vendor/ : Les bibliothèques tierces,
- web/ : Le répertoire Web racine.



Symfony 3

Identifier les
différences entre
symfony2.8 et 3

Architecture d'un projet Symfony 4.*



Le contrôleur frontal (1)

public/[index.php](#)

Il joue le rôle de dispatcheur :

- Intercepte les requêtes
- Appelle le noyau de symfony (AppKernel.php)
- Le noyau prépare la réponse à rendre

https://symfony.com/doc/4.2/create_framework/front_controller.html

Le contrôleur frontal (2)

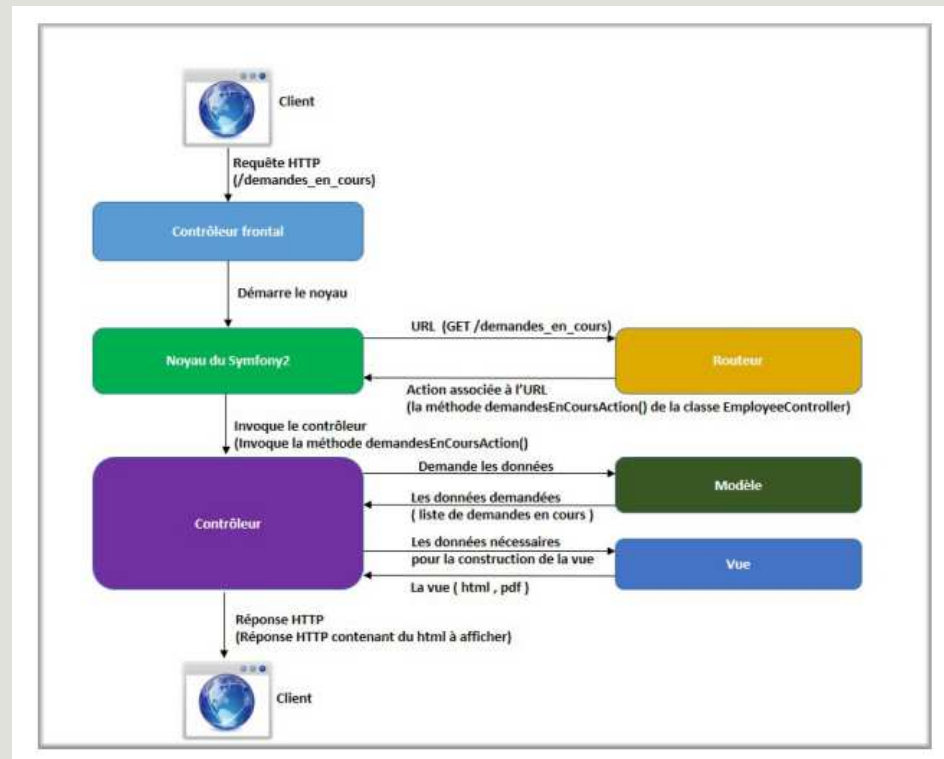
- Le contrôleur principal s'occupe de gérer les requêtes, mais cela signifie plus que simplement déterminer une action à exécuter. En fait, il exécute le code qui est commun à chaque action, soit les étapes suivantes :
- Définir les constantes.
- Déterminer les chemins des bibliothèques Symfony.
- Charger et initialiser les classes du cœur du framework.
- Charger la configuration.

Le contrôleur frontal (3)

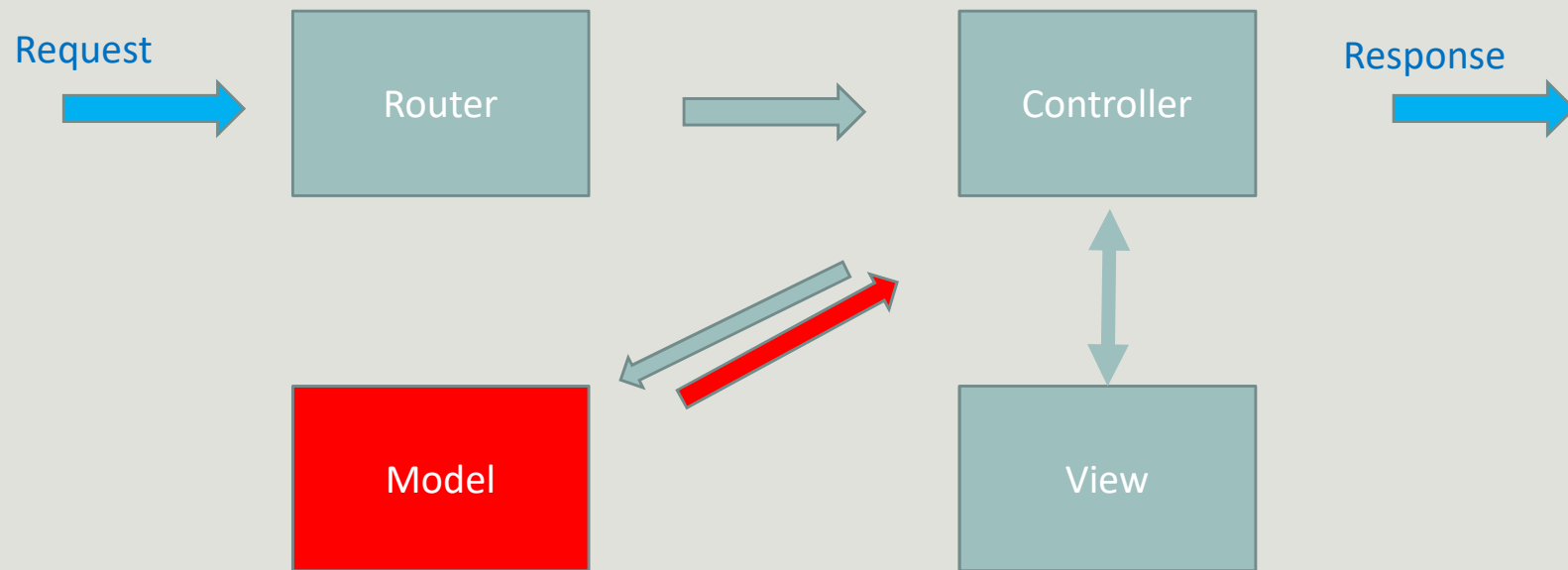
- Décoder la requête URL afin d'identifier les actions à effectuer et les paramètres de la requête.
- Si l'action n'existe pas, faire une redirection sur l'action 404 error.
- Activer les filtres (par exemple, si les requêtes nécessitent une authentification).
- Exécuter les filtres une première fois.
- Exécuter l'action et générer la présentation.
- Exécuter les filtres une seconde fois.
- Renvoyer la réponse.

Architecture MVC

- Symfony n'est pas une architecture MVC (Model View Controller).
- SensioLab définit Symfony comme un HTTP
- Framework basé sur l'échange HTTP Request/Response.
- Cependant on peut l'implémenter.
- La raison pour laquelle Symfony n'est pas défini
- comme telle est que la couche Model n'est pas fournie



Architecture MVC



Cette partie est indépendante de Symfony vous pouvez utiliser ce que vous voulez Doctrine par exemple

Symfony un Framework HTTP (Request/Response)

La classe Request (HttpFoundation)

- Le composant HttpFoundationRequest permet d'encapsuler les propriétés d'une requête http.
- Dans Symfony, la création de cet objet se fait au niveau du contrôleur frontal.
- `$request = Request::createFromGlobals();`
- Ceci revient à appeler le constructeur de la classe et de lui passer les variables globales nécessaires et qui concerne la requête.

```
$request = new Request(  
    $_GET,  
    $_POST,  
    [],  
    $_COOKIE,  
    $_FILES,  
    $_SERVER  
);
```


Symfony un Framework HTTP (Request/Response)

La classe Request (HttpFoundation)

L'objet request contient des informations concernant la requête du client. Ces informations sont récupérables via quelques propriétés public :

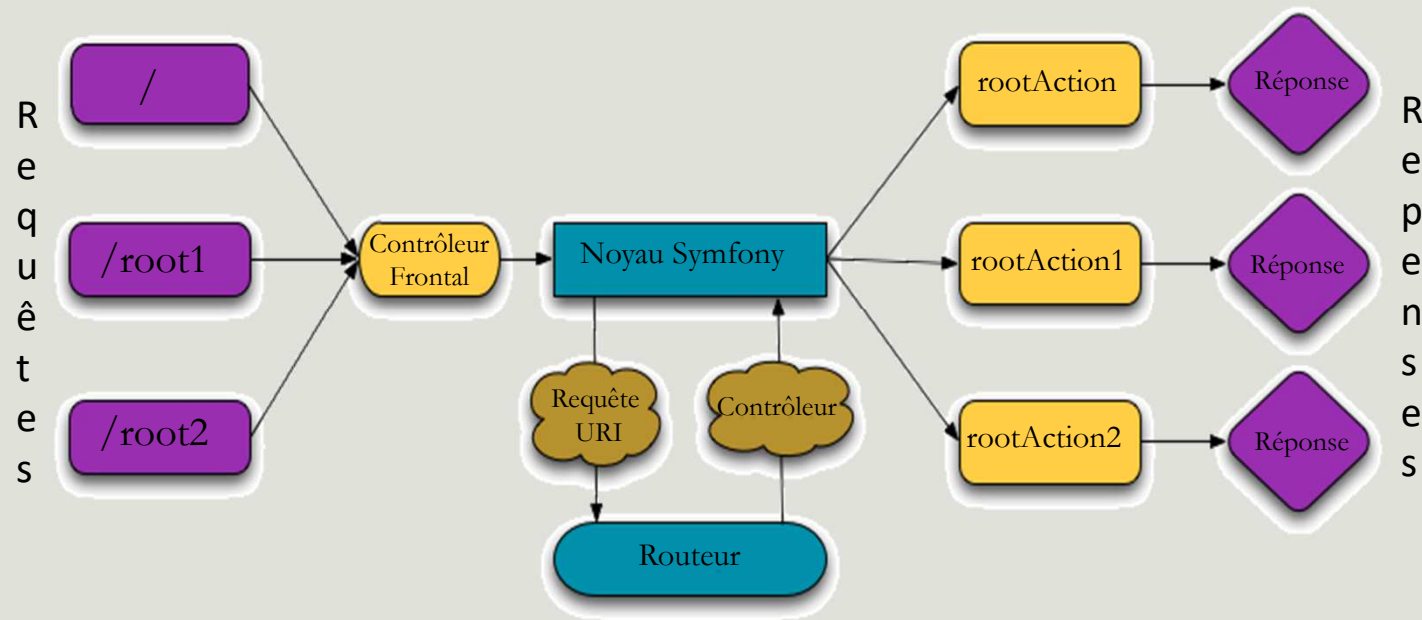
- **request**: équivaut à `$_POST`;
- **query**: équivaut à `$_GET ($request->query->get('name'))`;
- **cookies**: équivaut à `$_COOKIE`;
- **files**: équivaut à `$_FILES`;
- **server**: équivaut à `$_SERVER`;
- **headers**: équivaut à une partie de `$_SERVER ($request->headers->get('User-Agent'))`.

Symfony un Framework HTTP (Request/Response)

La classe Response (HttpFoundation)

- Le composant `HttpFoundationResponse` permet d'encapsuler les propriétés de la réponse à envoyer à l'utilisateur.
- Englobe particulièrement :
 - Le contenu de la réponse
 - Le code de la réponse
 - Un tableau Headers HTTP

Traitement d'une requête au sein de Symfony



Récapitulatif du flux Requête/Réponse

1. Un client envoi une requête HTTP.
2. Chaque requête va exécuter le Front Controller
3. Le front Controller lance Symfony et lui passe les informations de la requête afin qu'elle soit traité.
4. Symfony utilise les routes et les contrôleurs afin de créer une réponse
5. Symfony retourne une réponse à l'utilisateur.

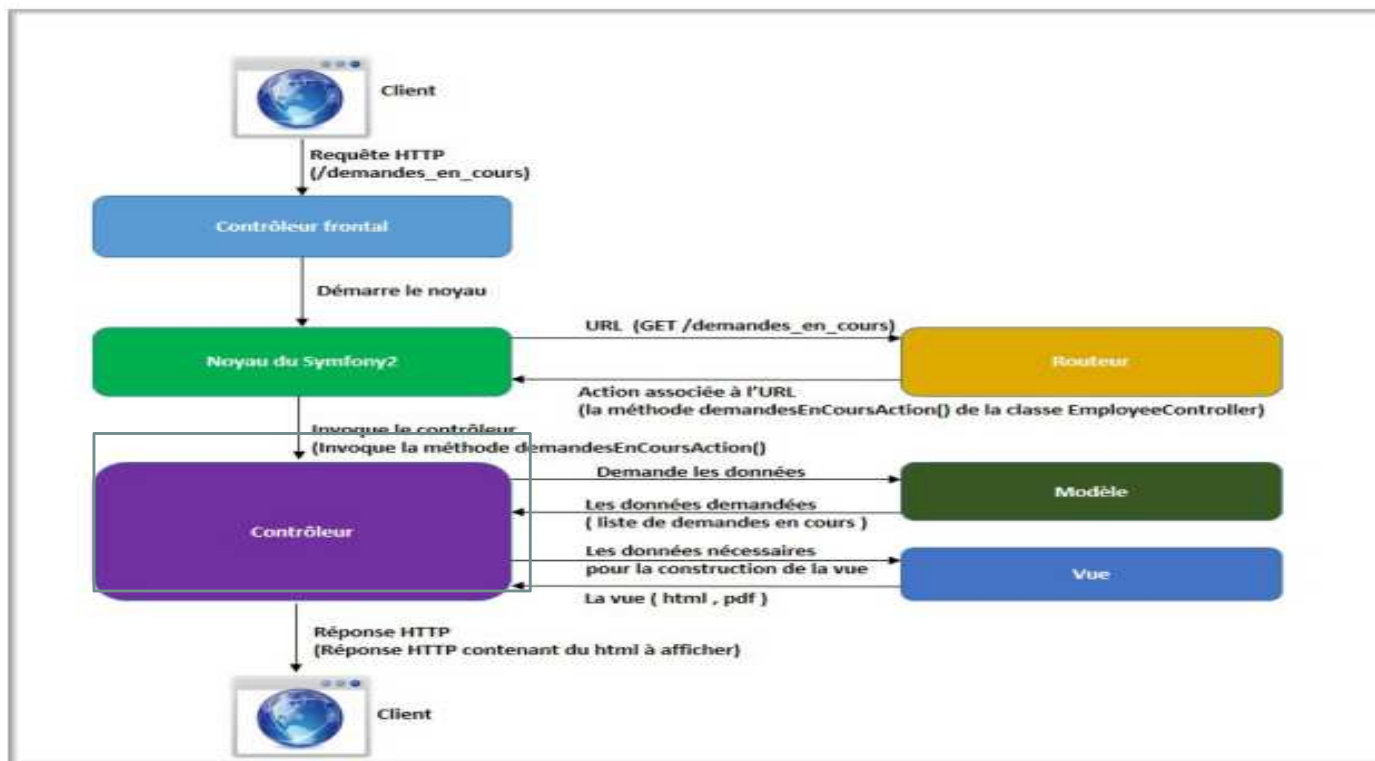
aymen.sellaouti@gmail.com

Symfony 6

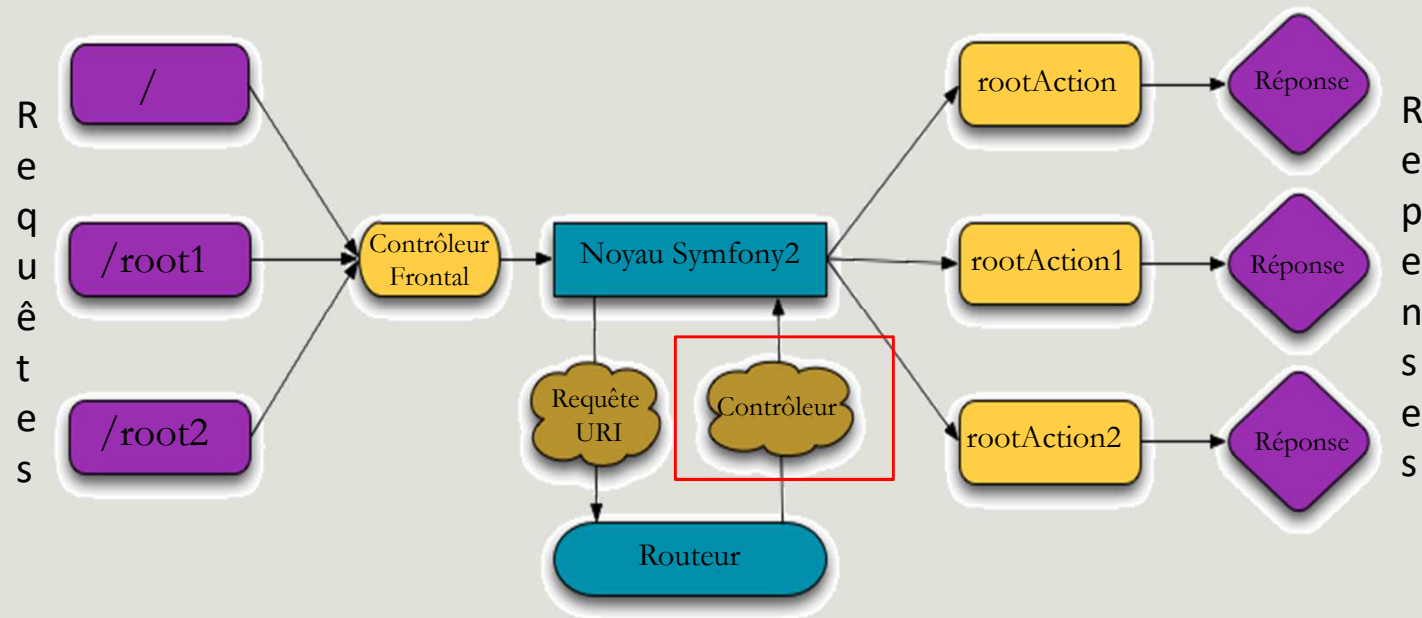
Les contrôleurs

AYMEN SELLAOUTI

Introduction (1)



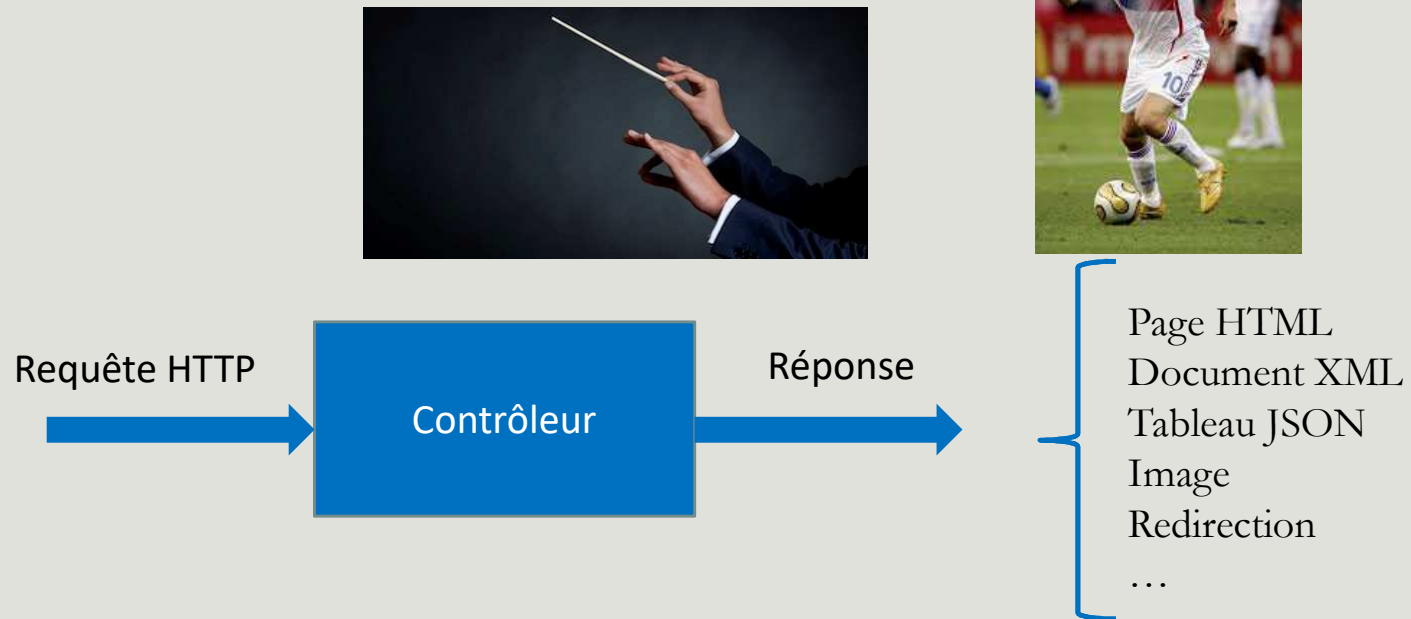
Introduction (2)



Introduction (3)

Fonction PHP (action)

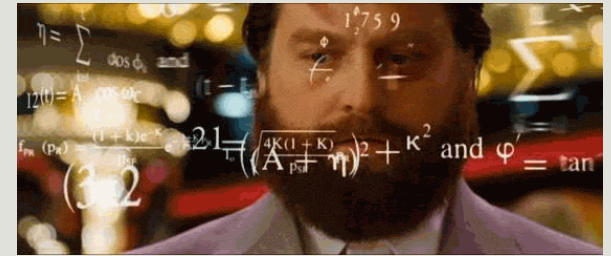
Rôle : Répondre aux requêtes des clients.



Exemple d'un contrôleur

```
#[Route('/personne', name: 'personne')]
public function index(){
    // On crée un objet Response puis on la retourne c'est le rôle du contrôleur
    $resp = new Response('<html><body>Bonjour le monde !</body></html>');
    return $resp;
}
```

Exercice



- Créer une classe FirstController
- Créer une action first
- Faire en sorte que lors de l'appel de la route /first cette action soit exécuté et qu'elle affiche une page contenant 'Hello forma'

Fonctions de base de la classe AbstractController

Méthode	fonctionnalité	Valeur de retour
<code>generateUrl(string \$route, array() \$parameters)</code>	Génère une URL à partir de la route	String
<code>forward(String Action, array () \$parameters)</code>	Forward la requête vers un autre contrôleur	Response
<code>Redirect(string \$url, int \$statut)</code>	Redirige vers une url	RedirectResponse
<code>RedirectToRoute(string \$route, array \$parameters)</code>	Redirige vers une route	Response
<code>Render(string \$view, array \$parameters)</code>	Affiche une vue	Response
<code>Get(string \$id)</code>	Retourne un service à travers son id	object
<code>createNotFoundException(String \$messag)</code>	Retourne une NotFoundException	NotFoundException

Génération automatique d'un contrôleur

- Afin d'automatiquement générer un contrôleur via la console, vous pouvez utiliser le maker de Symfony disponible depuis sa version 4.

```
php bin/console make:controller NomController  
symfony console make:controller NomController
```

Lien entre la route et le contrôleur

Création de la route

- Voici un exemple de correspondance entre une route et le contrôleur qui lui est associé :
- Nous prenons l'exemple d'une route écrite en attributs, YAML et en annotation.

```
#[Route('/personne', name: 'personne')]
```

personne:

path: /personne

controller: App\Controller\PersonneController::index

```
/**  
 * @Route("/personne", name="personne")  
 */
```

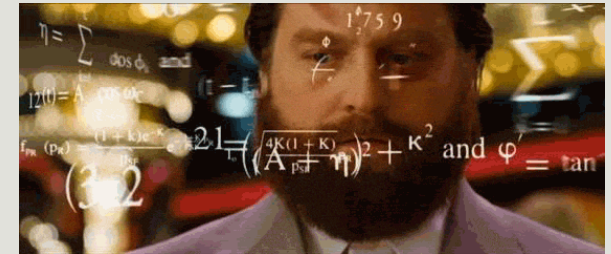
Lien entre la route et le contrôleur

Passage de paramétré : route vers contrôleur

- Afin de récupérer les paramètres de la route dans le contrôleur nous utilisons les noms des paramètres.
- Exemple

```
#[Route('/first/{section}', name: 'app_first')]
public function index($section)
{
    $resp = new Response('<html><body>Bonjour'. $section.'!</body></html>');
    return $resp;
}
```

Exercice



- Dans la classe FirstController
- Créer une **méthode** param qui prend en paramètre une variable nom à travers la route.
- Faite en sorte d'afficher Bonjour suivi du nom passé en paramètre.

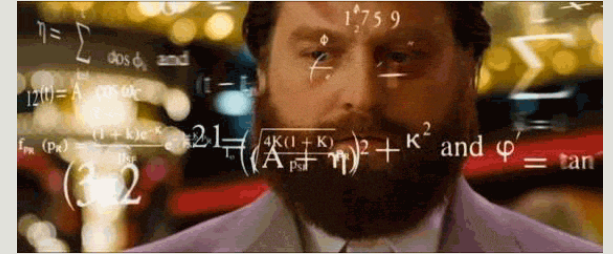
Récupérer les paramètres de la requête (1)

Afin de récupérer l'objet **Request** dans le contrôleur, il suffit d'utiliser le **type-hint** et le déclarer dans l'entête du contrôleur en question En spécifiant qu'il s'agit d'un objet de type Request.

Exemple

```
use Symfony\Component\HttpFoundation\Request;
public function indexAction(Request $req)
{
    ...
}
```

Exercise



- Dans la classe FirstController
- Créer une action second
- Faire en sorte d'y dumper (via la fonction dump) l'objet Request et l'objet Response. Vérifier les informations encapsulées par ses deux objets.

Récupérer les paramètres de la requête (2)

- L'objet **Request** permet de récupérer l'ensemble des attributs passés dans la requête

Type de paramètres	Méthode Symfony2	Méthode traditionnelle	Exemple
Variables d'URL	<code>\$request->query</code>	<code>\$_GET</code>	<code>\$request->query->get('var')</code>
Variables de formulaire	<code>\$request->request</code>	<code>\$_POST</code>	<code>\$request->request->get('var')</code>
Variables de cookie	<code>\$request->cookies</code>	<code>\$_COOKIE</code>	<code>\$request->cookies->get('var')</code>

Récupérer les paramètres de la requête (3)

Exemple : pour l'url suivante

<http://127.0.0.1/symfoRT4/web/index.php/test/bonjour/forma?groupe=1>

Pour récupérer le groupe passé dans l'url (donc du Get) on devra récupérer le request puis utiliser `$request->query->get('tag')`

```
public function index($section, Request $req)
{
    $groupe = $request->query->get('groupe');
    return new Response(« Bonjour ». $section. « groupe ». $groupe);
}
```

Récupérer les paramètres de la requête (4)

- La classe Request offre plusieurs informations concernant la requête HTTP à travers un ensemble de méthodes (https://symfony.com/doc/current/introduction/http_fundamentals.html#symfony-request-object)
- `getMethod()` : retourne la méthode de ma requête
- `isMethod()` : vérifie la méthode
- `getLocale` : retourne la locale de la requête (langue)
- `isXmlHttpRequest` : retourne vrai si la requête est de type `XmlHttpRequest`
- ...

Réponse aux requêtes

Renvoi (1)

- Le rôle principale du contrôleur est de répondre à la requête du client en envoyant une réponse.
- Vous pouvez créer un objet Response et y injecter votre contenu et le retourner.
- Sinon, le traitement se fait à travers un Helper qui utilise le [service templating](#) qui se charge de créer et d'irriguer un objet de type [Response](#).

Rôle : créer la réponse et la retourner

Méthode :

- **En utilisant les helpers :**
 - `$this->render ('l'url', 'les paramètres à transferer');`

Réponse aux requêtes

Renvoi (2)

Exemple :

```
public function index ($section, Request $req)
{
    $groupe = $request->query->get('groupe');
    return $this->render('default/index.html.twig', array(
        'section'=>$section,
        'groupe'=>$groupe)
    );
}
```

$\eta = \sum \cos \phi_i$ and \sum
 $12(t) = A \cos \omega t$
 $f_{\text{pre}}(\text{Pr}) = \frac{(1+\kappa)e^{-\kappa}}{(1+\text{Pr})}$
 $2.1 = \left(\frac{\sqrt{4K(1+K)}}{(A_{\text{pre}} + \eta)} \right)^2 + \kappa^2$ and $\varphi' = \tan$
 1759
 $(1-t)$
 (3.2)

- 
- The screenshot shows a web browser window. The address bar displays the URL `127.0.0.1/forminsat/web/app_dev.php/cv`. The page content is in French and reads:
- Bonjour
Je m'app  le aymen sellauti
J'ai 33 ans
Et je suis de la section GL

Réponse aux requêtes

Redirection

- **Rôle** : Redirection vers une deuxième page (en cas d'erreur ou de paramètres erronées ou de user non identifié par exemple)
- Redirection vers une **url**.
 - return this->**redirect**(\$url);
- Redirection vers une **route**
 - return this->**redirectToRoute**('nomDeMaRoute');

Réponse aux requêtes

Forwarding

- **Rôle** : Forwarder vers une action
- **Méthode** :
 - `$response = $this->forward('App\Controller\NomController::NomAction', array('name' => $name));`

Gestion des sessions

- Une des fonctionnalités de base d'un contrôleur est la manipulation des **sessions**.
- Un objet **session** est fourni avec l'objet **Request**.
- La méthode **getSession()** permet de récupérer la session.
- Il est préférable d'utiliser le type-hint via L'interface **SessionInterface** :
 - `public function index (SessionInterface $session)`

Gestion des sessions

- L'objet Session fournit deux méthodes : **get()** pour récupérer une variable de session et **set()** pour la modifier ou l'ajouter.
- **get** prend en paramètre la variable de session.
- **set** prend en entrée deux paramètres la **clef** et la **valeur**.
- Dans la TWIG on récupère les paramètres de la session avec la méthode

```
app.session.get ( 'nomParamètre' )
```

Gestion des sessions : les méthodes

- **all()** Retourne tous les attributs de la session dans un tableau de la forme clef valeur
- **has()** Permet de vérifier si un attribut existe dans la session. Retourne Vrai s'il existe Faux sinon
- **replace()** Définit plusieurs attributs à la fois: prend un tableau et définit chaque paire clé => valeur
- **remove()** Efface un attribut d'une clé donnée.
- **clear()** Efface tous les attributs.

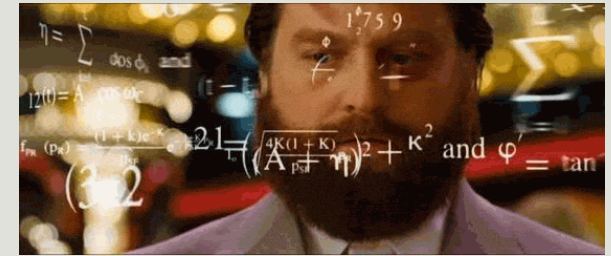
Gestion des sessions : les FlashMessages

- Les variables de sessions qui ne durent que le temps d'une seule page sont appelées **message Flash**.
- Utilisées généralement pour afficher un message après un traitement particulier (Ajout d'un enregistrement, connexion, ...).
- La méthode **getFlashBag()** permet de récupérer l'objet FlashBag à partir de l'objet session.
- La méthode **add** de cet objet permet d'ajouter une variable à cet objet.
- Vous pouvez utiliser un helper via la méthode **addFlash**.

Gestion des sessions : les FlashMessages

- Pour récupérer le Flash message de la TWIG on utilise `app.session.flashbag.get ('nomParamètre')`.
- Vous pouvez aussi utiliser la méthode `flashes` de la variable globale `app` qui contient le tableau des flashBags messages.

Exercice

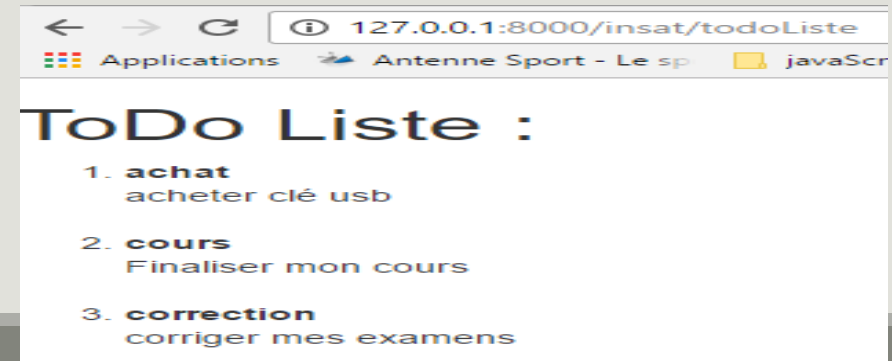


- Créer un contrôleur appelé `ToDoController`
- Créer une première action (`indexAction`) qui permet d'initialiser un tableau associatif de `todos` et qui le met dans la session puis appelle la page `'listeToDo.html.twig'`. Lors de l'appel de ce contrôleur, il faudra vérifier si la liste des `todo` existe déjà dans la session. Si la liste existe il ne faut pas la réinitialiser.

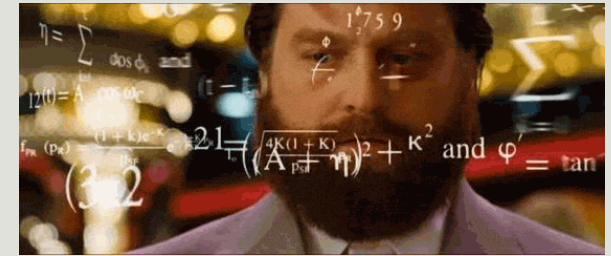
```
Exemple      $todos = array(  
    'achat'=>'acheter clé usb',  
    'cours'=>'Finaliser mon cours',  
    'correction'=>'corriger mes examens'  
);
```

Astuce : Afin d'afficher les éléments et les clés d'un tableau associatif dans la twig on peut utiliser la syntaxe suivante qui sera explicité dans le chapitre consacré aux TWIG.

```
{% for cle,element in tableau %}  
    {{ cle }} {{ element }}  
{% endfor %}
```

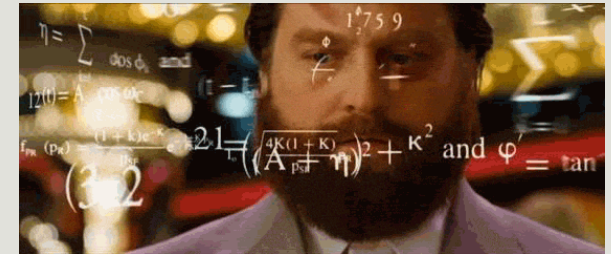


Exercice



- Quelques Astuces
- Ajouter bootstrap pour avoir les classes Alert ou un peu de css pour colorer le background de vos DIV ou paragraphe.
- Utiliser la fonction [unset](#) de php qui permet de supprimer un élément du tableau partir de sa clé.
- Pour ajouter un élément dans un tableau associatif il suffit d'utiliser la section suivante : **`$monTab['identifiant']=$var ;`**

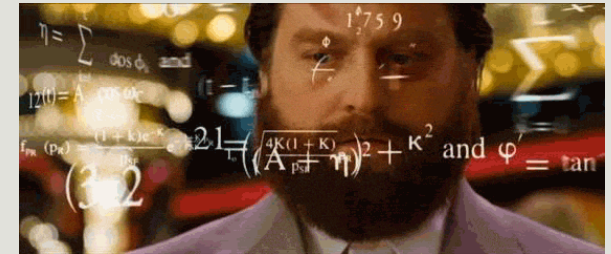
Exercice



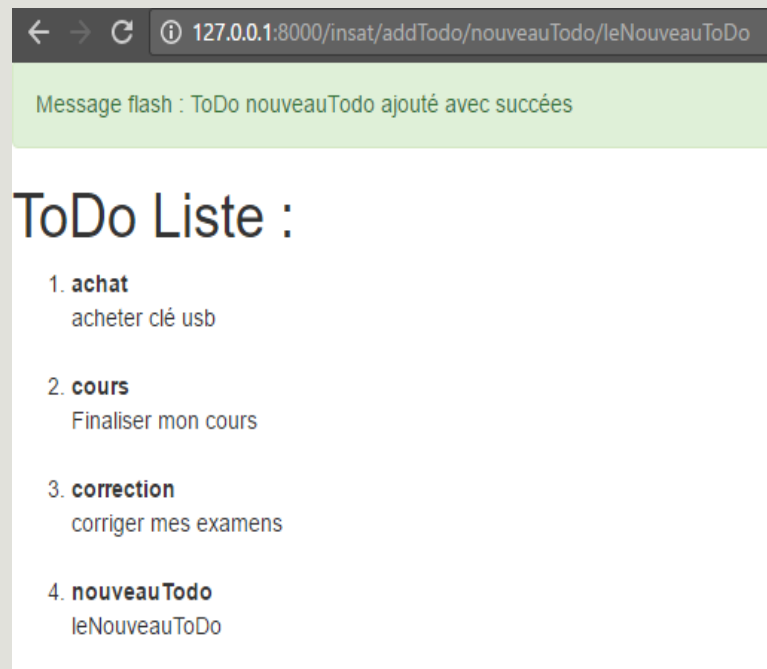
- Créer une action `addToDoAction` qui permet d'ajouter un `ToDo` ou de le mettre à jour. Cette action devra afficher la liste mise à jour. Si la liste de `ToDo` n'est pas encore initialisée, un message d'erreur sera affiché.



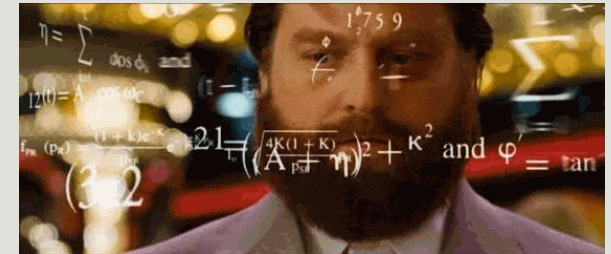
Exercice



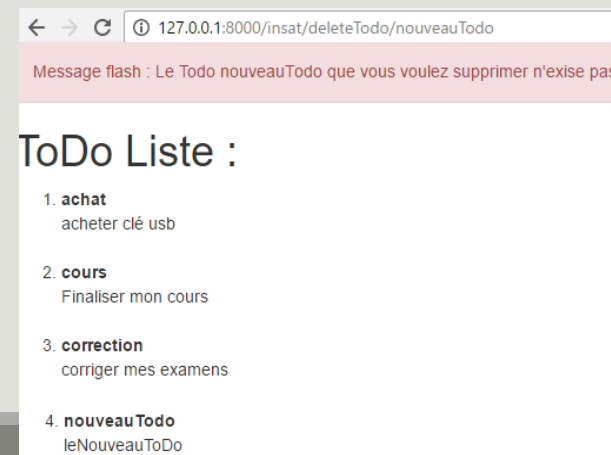
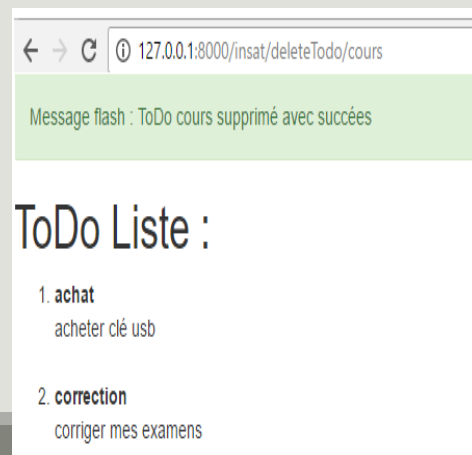
- Si le ToDo est ajouté avec succès, un message de succès sera affiché. Si le ToDo est mis à jour il faut le mentionner.



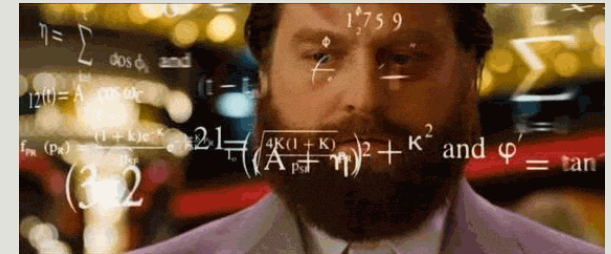
Exercice



- Créer une action deleteToDo qui permet de supprimer un ToDo à partir de son indice dans le tableau. Cette action devra afficher la liste mise à jour. Si le todo à supprimer n'existe pas, un message d'erreur est affiché.
- Si la suppression est effectuée avec succès un message est affiché.



Exercice



- Créer une action resetToDo qui permet de vider la session et de la remettre à son état initial. Prenez en considération le cas où la liste n'est pas encore initialisée



```
{# templates/base.html.twig #}
{# read and display just one flash message type #}
{% for message in app.flashes('notice') %}
    <div class="flash-notice">
        {{ message }}
    </div>
{% endfor %}

{# read and display several types of flash messages #}
{% for label, messages in app.flashes(['success', 'warning']) %}
    {% for message in messages %}
        <div class="flash-{{ label }}">
            {{ message }}
        </div>
    {% endfor %}
{% endfor %}

{# read and display all flash messages #}
{% for label, messages in app.flashes %}
    {% for message in messages %}
        <div class="flash-{{ label }}">
            {{ message }}
        </div>
    {% endfor %}
{% endfor %}
```

<https://symfony.com/doc/current/controller.html#flash-messages>

aymen.sellaouti@gmail.com

Symfony 6 Routing

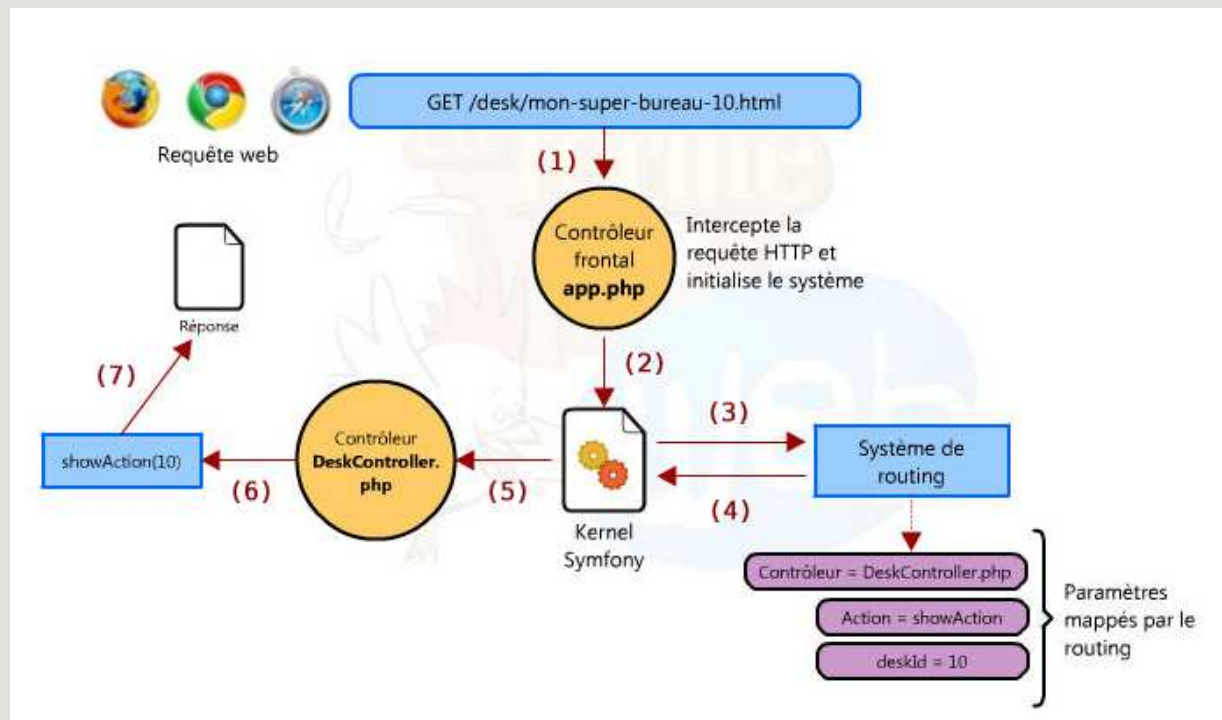
AYMEN SELLAOUTI

Introduction

Système permettant de

- gérer les liens internes du projet
- avoir des URLs plus explicites
- associer une URL à une action

Introduction



Cette architecture illustre le fonctionnement de Symfony.

1- Requête de l'utilisateur

2- La requête est envoyée vers le noyau de Symfony.

3- Le Noyau consulte le routeur afin de connaître quel Action exécuter.

4- Le routeur envoie les informations concernant l'URI

5- Le noyau invoque l'action exécuter.

6- Exécution de l'action

7- L'action retourne la réponse.

Routing (<http://www.lafermeduweb.net/>)

Format de gestion du routing

Les fichiers de routing peuvent être de quatre formats différents :

- YAML
- XML
- PHP
- Annotations
- **Attributs**

Skeleton d'une route

- Jusqu'à la version 3, Sensio recommande l'utilisation du format Yaml au sein des applications. Les bundles développés sont partagés entre deux formats : le XML et le Yaml.
- Sensio a décidé de recommander Yaml parce qu'il est « *user-friendly* ».
- A partir de la version 3.4, la documentation s'est focalisée essentiellement sur les annotations et la recommandation. Nous allons donc voir ces deux formats.
- Dans la version 8 de PHP, une nouvelle syntaxe plus lisible et plus fonctionnelle a été introduite : les **attributs**.

Squelette d'une route en utilisant les Annotations et les attributs

```
/**
 *
 * @Route("/blog", name="blog_list")
 */
public function list()
{
    // ...
}
```

```
#[Route('/blog', name: 'blog_list')]
public function list(): Response
{
    // ...
}
```

Squelette d'une route en utilisant YAML

```
# config/routes.yaml
blog_list:
  # Matches /blog exactly
  path:      /blog
  controller: App\Controller\BlogController::list
```

<https://symfony.com/doc/current/routing.html>

Squelette d'une route en utilisant XML

```
<!-- config/routes.xml -->
<?xml version="1.0" encoding="UTF-8" ?>
<routes xmlns="http://symfony.com/schema/routing"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://symfony.com/schema/routing
        https://symfony.com/schema/routing/routing-1.0.xsd">

    <!-- Matches /blog exactly -->
    <route id="blog_list" path="/blog"
controller="App\Controller\BlogController::list">
        <!-- settings -->
    </route>
</routes>
```

<https://symfony.com/doc/current/routing.html>

Squelette d'une route en utilisant PHP

```
<?php
// config/routes.php
use App\Controller\BlogController;
use
Symfony\Component\Routing\Loader\Configurator\RoutingConfigurator;

return function (RoutingConfigurator $routes) {
    // Matches /blog exactly
    $routes->add('blog_list', '/blog')->controller(
        [BlogController::class, 'list']
    );
};
```

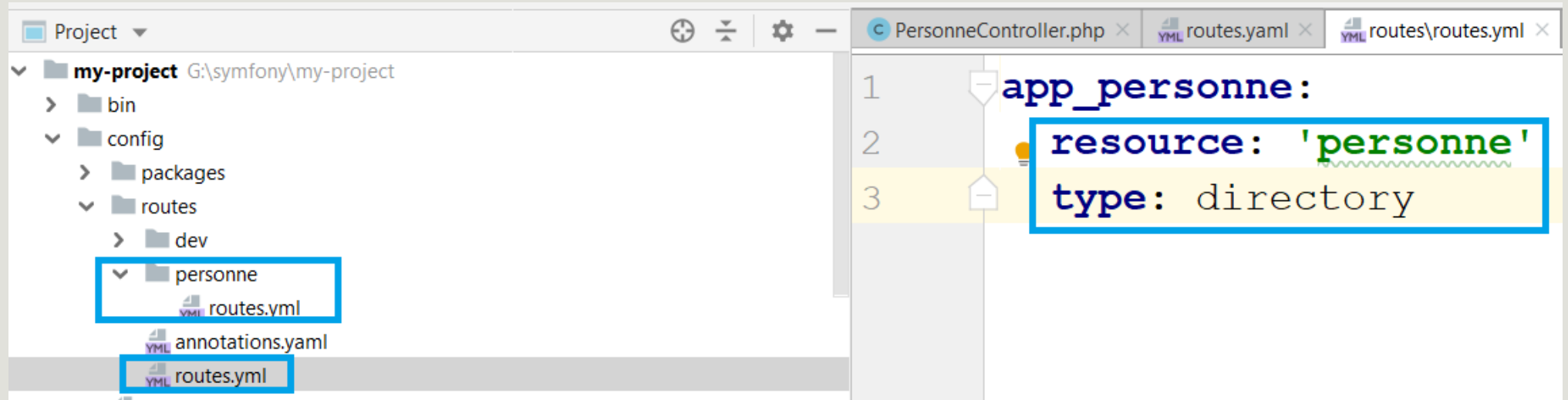
<https://symfony.com/doc/current/routing.html>

Organisation des routes YAML

- Lorsque vous utilisez YAML, il est préférable de ne pas centraliser l'ensemble de vos routes dans un même fichier. Ceci peut nuire à la lisibilité de vos routes.
- Décomposer vos routes en des fichiers logiques. Exemple si vous avez une partie Back et une partie front, ayez deux fichiers `back.yaml` et `front.yaml`.
- Garder le fichier principal de vos routes pour appeler ces fichiers la.

Organisation des routes YAML

- Afin d'identifier un fichier de ressources « route » utiliser la clé **resource** afin d'informer le chemin de la ressource et la clé **type** afin de spécifier le type de votre ressource (dans notre cas c'est directory).



Les préfixes

- Dans certains cas vous avez besoins d'avoir des **endpoints particuliers pour un ensemble de fonctionnalités**. Par exemple lorsque vous aller réaliser une partie administration vous aurez généralement des routes qui commencent par /admin.
- Afin de gérer ca, le routeur de Symfony offre la possibilité d'ajouter des **préfixes** à vos routes.

Les préfixes YAML

- Afin de **préfixer** une route YAML aller dans le fichier où vous avez appelé la ressource et ajouter la clé **prefix** :

```
app_personne:  
  resource: 'personne'  
  type: directory  
  prefix: /personne
```

https://symfony.com/doc/4.2/routing/external_resources.html

Préfixe annotation

- Une annotation `Route` sur une classe Contrôleur définit un préfixe sur toutes les routes des actions de ce contrôleur

```
/**
 * @Route("/personne")
 */
class PersonneController extends AbstractController
{
}
```

```
#[Route('/first')]
class FirstController extends AbstractController
```

Paramétrage de la route : Yaml

- Nous pouvons ajouter autant de paramètre dans la route
- Le séparateur est ‘/’

➤ Exemple

front_article:

path: /article/{year}/{langue}/{slug}/{format}

controller: App\Controller\BlogController::add

- Ici l'url doit contenir l'année de l'article {year}, la langue de l'article {langue} les mots clefs {slug} ainsi que le format {fomrat}

Paramétrage de la route : Annotation

```
/**
 * @Route("/hello/{name}", name="front_homepage")
 */
public function test ($name) {}

/**
 * @Route("/article/{year}/{locale}/{slug}/{format}",
name="front_article")
 */
public function showArticle($year,$locale,$slug,$format) {}
```

Paramétrage de la route : Attributs

```
#[Route('/second/{name}', name: 'app_second')]
public function index($name): Response
{
    return $this->render('second/index.html.twig', [
        'myName' => $name,
    ]);
}
```


Paramétrage de la route : valeurs par défaut Annotations

- En utilisant les annotations, nous ajoutons un champ `defaults` qui contiendra les valeurs par défaut.

```
/**
 * @Route(
 *     "/article/{year}/{locale}/{slug}/{format}",
 *     name="front_article",
 *     defaults={"format":"html", "slug":"Symfony"}
 * )
 */
```

```
public function showArticleAction($year, $_locale, $slug, $_format) {
    dump($year, $_locale, $slug, $_format);
    die();
}
```

Important : Seul les paramètres optionnels terminant la route pourront être absents de l'URL

- Maintenant l'URL suivante devient correcte : <http://127.0.0.1:8000/article/2005/fr> et les variables slug et format prendront leur valeur par défaut

Paramétrage de la route : valeurs par défaut Annotations Raccourci

- Vous pouvez utiliser le raccourci { attribut **?defaultValue** }.
- Evitez ce type de raccourci si vous avez des routes complexes afin d'avoir une bonne lisibilité de vos routes.

```
/**
 * @Route(
 *     "/article/{year}/{locale}/{slug?symfony}/{format?html}",
 *     name="front_article",
 * )
 */
public function showArticleAction($year, $locale, $slug, $format) {
    dump($year, $_locale, $slug, $_format);
    die();
}
```

Important : Seul les paramètres optionnels terminant la route pourront être absents de l'URL

- Maintenant l'URL suivante devient correcte : <http://127.0.0.1:8000/article/2005/fr> et les variables slug et _format prendront leur valeur par défaut

Paramétrage de la route : valeurs par défaut Attributes

```
#[Route(
    '/second/{name}',
    name: 'app_second',
    defaults: ['name' => 'aymen']
)]
public function index($name): Response
{
    return $this->render('second/index.html.twig', [
        'myName' => $name,
    ]);
}
```

Paramétrage de la route : valeurs par défaut Attributes

```
#[Route(
    '/second/{name?skander}',
    name: 'app_second',
)]
public function index($name): Response
{
    return $this->render('second/index.html.twig', [
        'myName' => $name,
    ]);
}
```

Paramétrage de la route : valeurs par défaut Yaml

- Afin d'avoir des valeurs par défauts nous utilisons la syntaxe suivante :

Syntaxe

front_article:

path: /article/ {year} / {locale} / {slug} / {format}

controller: App\Controller\BlogController::add

defaults:

attribut: defaultValue

Paramétrage de la route : Requirements Annotation

- En utilisant les annotations, nous ajoutons un champ **requirements** qui contiendra les différentes contraintes.

```
/**
 * @Route (
 *     "/article/{year}/{locale}/{slug}/{format}",
 *     name="front_article",
 *     defaults={"_format":"html", "slug":"Symfony"},
 *     requirements={
 *         "locale" : "fr|en",
 *         "year" : "\d{4}"
 *     }
 * )
 */
public function showArticleAction($year, $_locale, $slug, $_format) {}
```

Paramétrage de la route : Requirements Annotation, le raccourci

- Vous pouvez utiliser le raccourci { attribut `<requirement>` }.
- Evitez ce type de raccourci si vous avez des routes complexes afin d'avoir une bonne lisibilité de vos routes.

```
/**
 * @Route(
 *     "/article/{year<\d+>}/{locale}/{slug}.{format}",
 *     name="front_article")
 */
public function showArticleAction($year, $_locale, $slug, $_format) {
}
```

Paramétrage de la route : Requirements Attributs

- En utilisant les annotations, nous ajoutons un champ `requirements` qui contiendra les différentes contraintes.

```
#[Route(
    '/second/{name}/{age}',
    name: 'app_second',
    requirements: ['age'=> '\d+']
)]
public function index($name, $age): Response
{
    return $this->render('second/index.html.twig', [
        'myName' => $name,
        'myAge'
    ]);
}
```


Paramétrage de la route : Requirements Attributs

- En utilisant les annotations, nous ajoutons un champ `requirements` qui contiendra les différentes contraintes.

```
#[Route(
    '/second/{name}/{age<\d+>}',
    name: 'app_second',
)]
public function index($name, $age): Response
{
    return $this->render('second/index.html.twig', [
        'myName' => $name,
        'myAge' => $age
    ]);
}
```

Requirements autres exemples

$\backslash d$ équivalente à $\backslash d\{1\}$

$\backslash d+$ ensemble d'entiers

Ordre de traitement des routes (1)

Le traitement des routes se fait de la première route vers la dernière.

Attention à l'ordre d'écriture de vos routes.

front: Comment accéder au path front_pages ? Quel est le problème avec ces 2 routes

path: /front/{page}

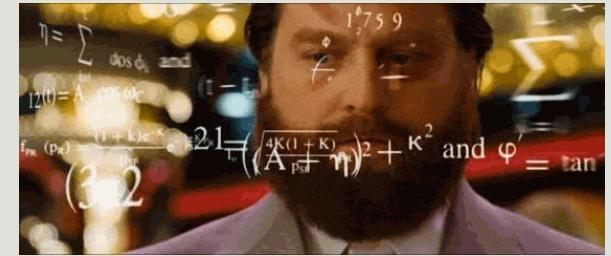
controller: App\Controller\BlogController::front

front_pages:

path: /front/{Keys}

controller: App\Controller\BlogController::show

Exercice



- Reprenez les deux routes précédentes.
- Testez l'accessibilité à la deuxième route.
- Ajouter ce qu'il faut pour remédier au problème

Ordre de traitement des routes (2)

front:

path: /front/{page}

controller: App\Controller\BlogController::front

}

front_pages:

path: /front/{Keys}

controller: App\Controller\BlogController::show

Les deux routes sont de la forme /front/* donc n'importe quel route de cette forme sera automatiquement transféré au Default controller pour exécuter l'index action.

Proposer une solution

Ordre de traitement des routes (3)

front:

path: /front/{page}

controller: App\Controller\BlogController::front

requirements:

page: \d+

front_pages:

path: /front/{Keys}

controller: App\Controller\BlogController::show

Tester le fonctionnement des routes suivantes : /front/1234

/front/test-ordre-de-routeing

Ordre de traitement des routes (4)

Que se passe t-il si on inverse l'ordre des deux routes ? Est-ce que la solution persiste?

front_pages:

path: /front/{Keys}

controller: App\Controller\BlogController::show

front:

path: /front/{page}

controller: App\Controller\BlogController::front

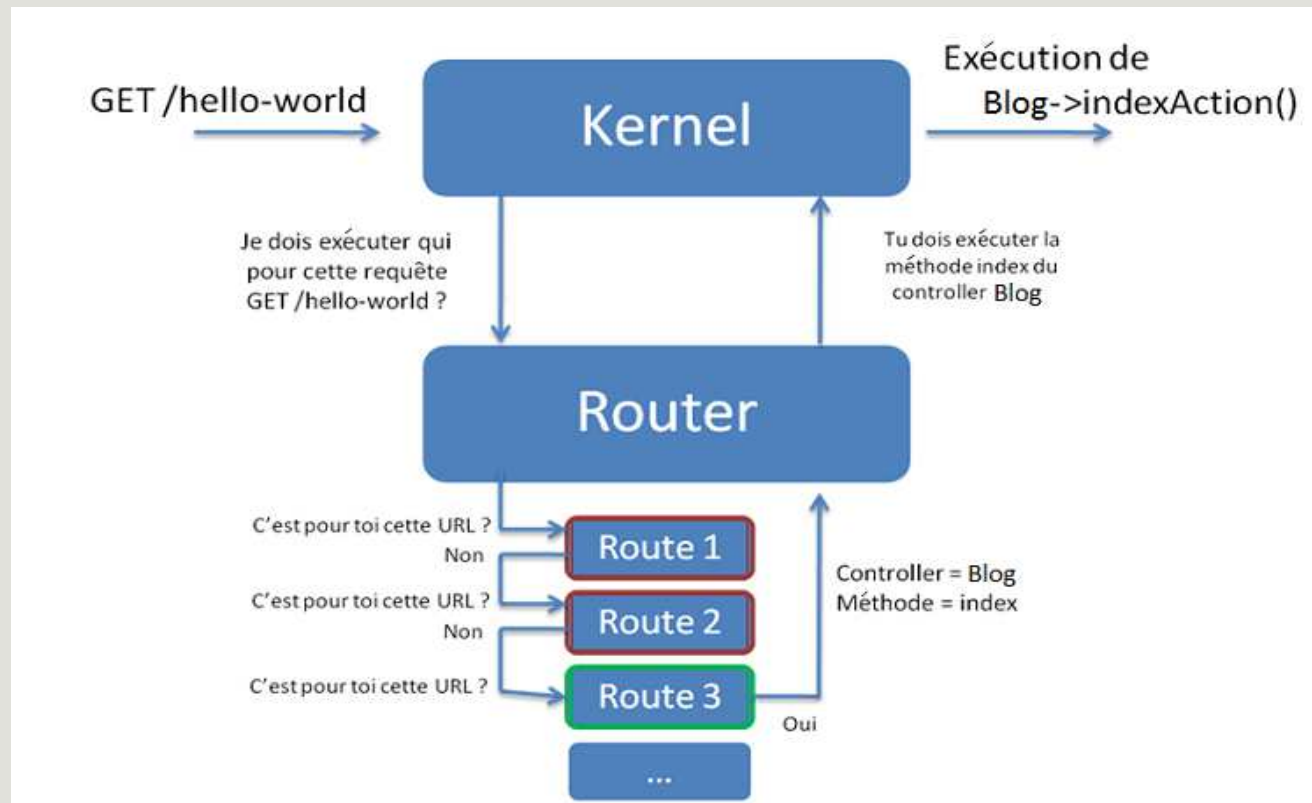
requirements:

page: \d+

Ordre de traitement des routes (5)

- Les Routes précédentes Gagnent toujours
- L'ordre des routes est très important.
- En utilisant un ordre clair et intelligent, vous pouvez accomplir tout ce que vous voulez.
- <http://symfony.com/fr/doc/current/book/routing.html>

Ordre de traitement des routes (6)



Débogage des routes

➤ Afin de visualiser l'ensemble des routes utiliser la debug toolbar

➤ Vous pouvez aussi utiliser la commande :

`symfony console debug:router`

`symfony console debug:router`

➤ On peut aussi vérifier quelle route correspond à une URL spécifique

`symfony console router:match URI`

`php bin/console router:match URI`

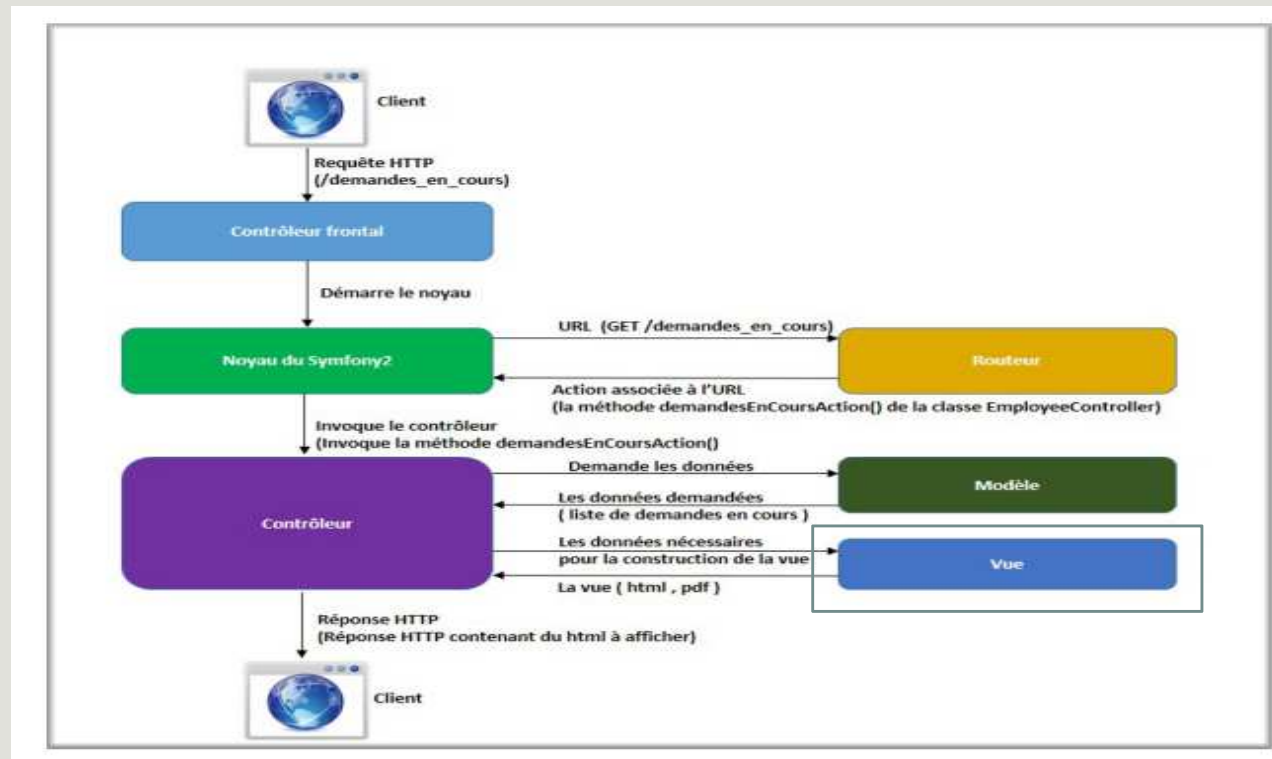
Route name	Pattern	Log
_url	_url{token}	Path "_url{token}" does not match
_profile_home	_profile	Path "_profile" does not match
_profile_search	_profilesearch	Path "_profilesearch" does not match
_profile_search_bar	_profilesearch_bar	Path "_profilesearch_bar" does not match
_profile_purge	_profilepurge	Path "_profilepurge" does not match
_profile_info	_profileinfo{token}	Path "_profileinfo{token}" does not match
_profile_graphs	_profilegraphs	Path "_profilegraphs" does not match
_profile_search_results	_profilesearch_results	Path "_profilesearch_results" does not match
_profile	_profile{token}	Path "_profile{token}" does not match
_profile_router	_profile{token}/router	Path "_profile{token}/router" does not match
_profile_exception	_profile{token}/exception	Path "_profile{token}/exception" does not match
_profile_exception_css	_profile{token}/exception.css	Path "_profile{token}/exception.css" does not match
_configuration_home	_configuration	Path "_configuration" does not match
_configuration_step	_configurationstep{token}	Path "_configurationstep{token}" does not match
_configuration_final	_configurationfinal	Path "_configurationfinal" does not match

Symphony 6

TWIG

AYMEN SELLAOUTI

Introduction (1)



Introduction (2)

- Moteur de Template PHP.
- Développé par l'équipe de Sensio Labs,
- Directement intégré dans Symfony (pas besoin de l'installer).

Introduction (3)

- L'objectif principal de Twig est de permettre aux développeurs de séparer la couche de présentation (Vue) dans des Templates dédiés, afin de favoriser la maintenabilité du code.
- Idéal pour les graphistes qui ne connaissent pas forcément le langage PHP et qui s'accommoderont parfaitement des instructions natives du moteur, relativement simples à maîtriser.
- Il y a quelques fonctionnalités en plus, comme l'héritage de templates.
- Il sécurise vos variables automatiquement (`htmlentities()`, `addslashes()`)

Syntaxe de bases de Twig

- **{{ maVar }}** : Les doubles accolades (équivalent de l'écho dans php ou du `<%= %>` pour les jsp) permettent d'imprimer une valeur, le résultat d'une fonction...
- **{% code %}**: Les accolades pourcentage permettent d'exécuter une fonction, définir un bloc...
- **{# Les commentaires #}**: Les commentaires.

Comment afficher une variable dans TWIG (1)

Fonctionnalité	Exemple Twig	Équivalent PHP
Afficher une variable	Variable : {{ MaVariable }}	Pseudo : <?php echo \$MaVariable; ?>
Afficher le contenu d'une case d'un tableau	Identifiant : {{ tab['id'] }}	Identifiant : <?php echo \$tab['id']; ?>
Afficher l'attribut d'un objet, dont le getter respecte la convention \$objet->getAttribut()	Identifiant : {{ user.id }}	Identifiant : <?php echo \$user->getId(); ?>
Afficher une variable en lui appliquant un filtre. Ici, « upper » met tout en majuscules :	MaVariable majus : {{ MaVariable upper }}	MaVariable majus: <?php echo strtoupper(\$MaVariable); ?>

Comment afficher une variable dans TWIG (2)

Fonctionnalité	Exemple Twig	Équivalent PHP
Afficher une variable en combinant les filtres. « striptags » supprime les balises HTML. « title » met la première lettre de chaque mot en majuscule.	Message : {{ news.texte striptags title }}	Message : <?php echo ucwords(strip_tags(\$news->getTexte())); ?>
Utiliser un filtre avec des arguments. Attention, il faut que date soit un objet de type Datetime ici.	Date : {{ date date('d/m/Y') }}	Date : <?php echo \$date->format('d/m/Y'); ?>
Concaténer	Identité : {{ nom ~ " " ~ prenom }}	Identité : <?php echo \$nom.' '. \$prenom; ?>

Comment afficher une variable dans les TWIGS :

Exemple :

```
{# set permet de déclarer des variables #}  
{% set MaVariable = 'test' %}  
Bonjour {{ MaVariable }}!  
<br>  
{# On affiche l'indexe du tableau tab envoyé par le controleur#}  
Identifiant : {{ tab['1'] }}  
<br>  
{# Application de quelques filtres #}  
MaVariable majus : {{ MaVariable|upper }}  
<br>  
{% set texte = '<i>test</i>' %}  
Message sans les filtres : {{ texte }}  
<br>  
Message avec les filtres : {{ texte|striptags|title }}  
<br>  
{# now nous donne la date système #}  
{{ "now"|date("m/d/Y") }}  
<br>  
{# Concaténer #}  
concat : {{ MaVariable ~"~"~ texte }}
```

← → ↻ 127.0.0.1/ecommerceN/web/app_dev.php/hello

Bonjour test!
Identifiant : varTab2
MaVariable majus : TEST
Message sans les filtres : <i>test</i>
Message avec les filtres : Test
04/02/2015
concat : test<i>test</i>

Twig et l'affichage d'un attribut

Lorsqu'on veut accéder à un attribut d'un objet avec twig on a le fonctionnement suivant pour l'exemple `{{personne.name}}` :

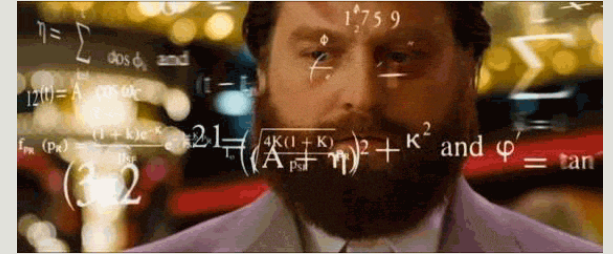
- ✓ Vérification si `personne` est un tableau, si oui vérifier que nous avons un index valide dans ce cas elle affiche le contenu de `personne['name']`
- ✓ Sinon, si `personne` est un objet, elle vérifie si `name` est un attribut valide public. Si c'est le cas, elle affiche `personne->name`.
- ✓ Sinon, si `personne` est un objet, elle vérifie si `name()` est une méthode valide publique. Si c'est le cas, elle affiche `personne->name()`.
- ✓ Sinon, si `personne` est un objet, elle vérifie si `getName()` est une méthode valide. Si c'est le cas, elle affiche `personne->getName()`.
- ✓ Sinon, et si `personne` est un objet, elle vérifie si `isName()` est une méthode valide. Si c'est le cas, elle affiche `personne->isName()`.
- ✓ Sinon, elle n'affiche rien et retourne null.

Les filtres

- Un **filtre** permet de changer l'affichage d'une variable sans changer son contenu. Il peut avoir un ou plusieurs paramètres.

Filtre	Description	Exemple Twig
Upper	Met toutes les lettres en majuscules.	<code>{{ var upper }}</code>
Striptags	Supprime toutes les balises XML.	<code>{{ var striptags }}</code>
Date	Formate la date selon le format donné en argument. La variable en entrée doit être une instance de Datetime.	<code>{{ date date('d/m/Y') }}</code> Date d'aujourd'hui : <code>{{ "now" date('d/m/Y') }}</code>
Format	Insère des variables dans un texte, équivalent à printf .	<code>{{ "Il y a %s pommes et %s poires" format(153, nb_poires) }}</code>
Length	Retourne le nombre d'éléments du tableau, ou le nombre de caractères d'une chaîne.	Longueur de la variable : <code>{{ texte length }}</code> Nombre d'éléments du tableau : <code>{{ tableau length }}</code>

Exercice



- Reprenez l'exemple du cv et appliquer les filtres suivants sur l'affichage :
- Les nom et prénoms doivent être en majuscule
- L'âge doit être affiché en entier positif
- Donner des valeurs par défaut aux variables

Les fonctions

Une **fonction** peut changer la valeur d'une variable et peut avoir un ou plusieurs paramètres.

Quelques fonctions offertes par twig :

```
{% if date(user.created_at) < date('-2days') %}  
    {# do something #}  
{% endif %}
```

➤ **date** : convertie un argument en une date afin de permettre une comparaison entre dates.

➤ **max, min** : retourne le min et le max d'un ensemble

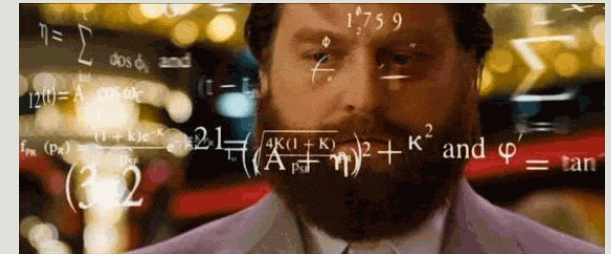
➤ **parent** : utiliser dans l'héritage et retourne le contenu du bloc parent.

➤ **random** : retourne un élément aléatoire

selon les paramètres passées à la fonction

```
{{ random(['apple', 'orange', 'citrus']) }} {# example output: orange #}  
{{ random('ABC') }} {# example output: C #}  
{{ random() }} {# example output: 15386094  
(works as the native PHP mt_rand function) #}  
{{ random(5) }} {# example output: 3 #}  
{{ random(50, 100) }} {# example output: 63 #}
```

Exercice



- Créer une page TWIG qui **prend en paramètre un tableau de notes**, qui l'affiche, affiche le nombre de ces éléments, le min et le max de ce tableau en utilisant les fonctions et les pipes twig.
- Le nombre d'éléments du tableau doit être paramétrable à travers la route et doit être de 5 par défaut.
- Le contenu du tableau doit être aléatoire.

902
527
65
215
815
802
80

Nombre de valeurs : 7

Min : 65

Max : 902

Créer vos propres fonctions ou filtres

- Afin de personnaliser des fonctions ou des filtres TWIG vous devez créer une [extensionTwig](#).
- Une extension est une classe qui implémente l'interface [TWIG_ExtensionInterface](#) ou étend la classe abstraite [AbstractExtension](#).
- Pour créer un filtre, il faut implémenter la méthode [getFilters](#). Elle retourne un tableau d'instance de la classe [TwigFilter](#). Chaque instance définit un filtre.
- Le constructeur de la classe [TwigFilter](#) prend en paramètre le nom du filtre suivi d'un tableau. Le tableau prend deux paramètres, la classe qui contient la méthode à exécuter et la méthode à exécuter.
- De même pour une TwigFunction


```

namespace App\Twig;

use Twig\Extension\AbstractExtension;
use Twig\TwigFilter;

class AppExtension extends AbstractExtension
{
    public function getFilters()
    {
        return [
            new TwigFilter('price', [$this, 'formatPrice']),
        ];
    }

    public function formatPrice($number, $decimals = 0,
$decPoint = '.', $thousandsSep = ',')
    {
        $price = number_format($number, $decimals, $decPoint,
$thousandsSep);
        $price = '$'.$price;

        return $price;
    }
}

```

```

// src/Twig/AppExtension.php
namespace App\Twig;

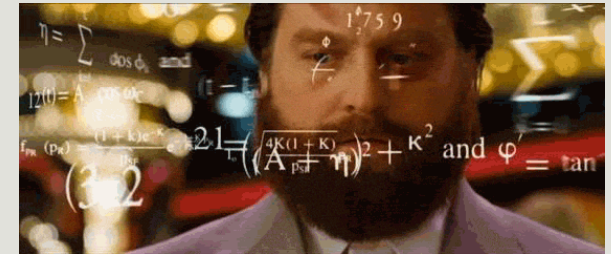
use Twig\Extension\AbstractExtension;
use Twig\TwigFunction;

class AppExtension extends AbstractExtension
{
    public function getFunctions()
    {
        return [
            new TwigFunction('area', [$this, 'calculateArea']),
        ];
    }

    public function calculateArea(int $width, int $length)
    {
        return $width * $length;
    }
}

```

Exercice



- Créer un filtre DefaultImage qui permet d'afficher une image par défaut si l'image passée en paramètre est Null ou si elle n'existe pas.

Structures de contrôle

- Les structures que ce soit séquentielles ou itératives sont souvent très proches du langage naturel.
- Elles sont introduites entre `{% %}`
- Généralement elle se termine par une expression de fin de block

Affectation de variables

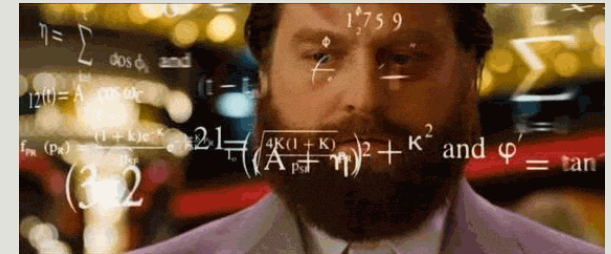
- Afin d'affecter une valeur à une variable dans TWIG, utiliser la syntaxe suivante :

```
{% set maVar = « valeur » %}
```

Exemple

```
{% set sum = 0 %}
```

Exercice



- Reprenez cet exercice :
- Créer une page TWIG qui prend en paramètre un tableau de notes, qui l'affiche, affiche le nombre de ces éléments, le min et le max de ce tableau en utilisant les fonctions et les pipes twig.
- Le nombre d'éléments du tableau doit être paramétrable à travers la route et doit être de 5 par défaut.
- Le contenu du tableau doit être aléatoire.
- Ajouter la somme et la moyenne des éléments

839
893
946

Nombre de valeurs : 3

Min : 839

Max : 946

Somme : 2678

Moyenne : 892.666666666667

Structure conditionnelle

if

Syntaxe :

```
{% if cnd %}
```

Block de traitement

```
{%endif%}
```

Exemple

```
{% if employee.salaire < 250 %}
```

ce salaire est inférieur au smig

```
{%endif%}
```

Structure conditionnelle

IF else elseif

Syntaxe :

```
{% if cnd %}
```

block traitement

```
{% elseif cnd2 %}
```

block traitement

```
{% else %}
```

block traitement

```
{% endif %}
```

Exemple

```
{% if maison.temperature <0 %}
```

Très Froid

```
{% elseif maison.temperature <10 %}
```

Froid

```
{% else %}
```

Bonne température

```
{% endif %}
```

Tests

is defined l'équivalent de **isset** en php

Rôle vérifie l'existence d'une variable

Exemple: `{% if MaVariable is defined %} J'existe {% endif%}`

even

Rôle vérifie si la variable est pair

Exemple : `{% if MaVariable is even %} Pair{% endif%}`

odd

Rôle vérifie si la variable est impair

Exemple : `{% if MaVariable is odd%} Impair{% endif%}`

Structure itérative

```
{% for valeur in valeurs %}
```

block à répéter

```
{% else %}
```

Traitements à faire si il n'y

a aucune valeur

```
{% endfor %}
```

Exemple

La formation de l'équipe A est :


```
{% for joueur in joueurs %}
```

 { {joueur.nom} }

```
{% else %}
```

La liste n'a pas encore été renseignée

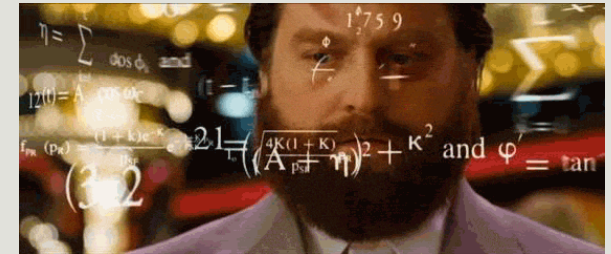
```
{% endfor %}
```


Structure itérative

➤ La boucle for définit une variable loop ayant les attributs suivants :

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

Exercice



- Reproduire ce tableau permettant d'afficher un tableau de user.

name	firstname	age
sellaouti	aymen	36
Ben Slimen	Ahmed	45
Abdennebi	Mohamed	28

Accès aux Templates

Les Templates doivent se trouver dans l'un des emplacements suivants :

- templates/
- /Resources/views d'un bundle

Afin d'accéder aux Template, une convention de nommage est définie :

Si on est dans vos vues : Dossier/pageTwig

Exemples

index.html.twig // ce fichier se trouve directement dans templates

Si vous voulez accéder à la vue d'un bundle :

@NomBundle/Dossier/pageTwig

Le nom du bundle ne doit pas contenir le mot Bundle

Nommage des pages TWIG

- Par convention le nommage des vues dans symfony se fait selon la convention suivante :

NomPage.FormatFinal.MoteurDeTemplate

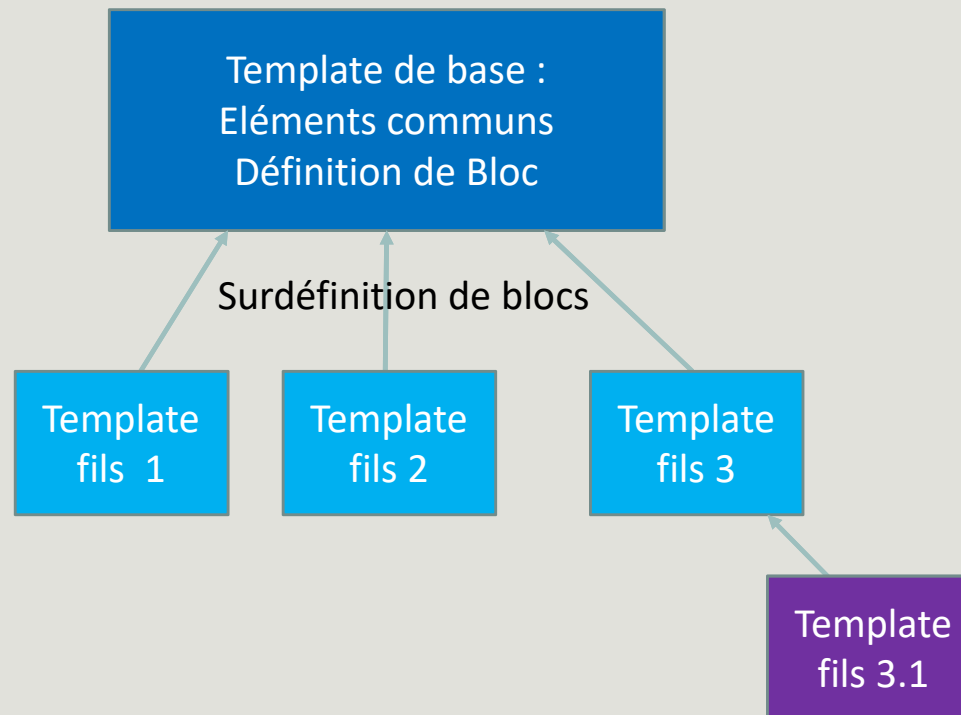
Exemple

index.html.twig

- Le nom du fichier est index, le format final sera du html et le moteur de template suivi est le TWIG

Héritage 1- Définition

- Vision proche de l'héritage dans l'orienté objet



Héritage 2 – Exemple de Template père

➤ Exemple de Template de base

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>{% block title %}Bonjour je suis le bloc principal!{% endblock %}</title>
    {% block stylesheets %}{#ici je vais définir mes fichiers css#}{% endblock %}
    <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}" /> {#ici c'est le favicon de mon appli#}
  </head>
  <body>
    {% block body %}
    {#ici c'est le contenu du Body#}
    Bienvenue dans le body du bloc principal
    {% endblock %}
    {% block javascripts %}{#ici je vais définir mes fichiers js#}{% endblock %}
  </body>
</html>
```



127.0.0.1/symfoRT4/web/app_dev.php/as/base

Bienvenu dans le body du bloc principal

Héritage 3- Syntaxe

- Afin d'hériter d'un Template père il faut étendre ce dernier avec la syntaxe suivante :

```
{% extends 'TemplateDeBase' %}
```

- Exemple :

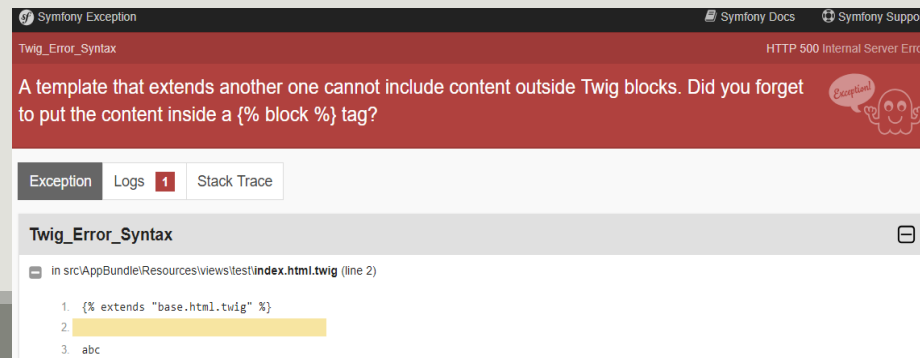
```
{% extends 'base.html.twig' %}
```

- Si le fils ne surcharge aucun des blocs et n'ajoute rien on aura le même affichage que pour la base



Héritage 4- Le Bloc

- L'élément de base de l'héritage est le **bloc**
- Un bloc est défini comme suit : `{% block nomBloc%}` contenu du `bloc{%endblock%}`
- Pour changer le contenu de la page il faut sur-définir le bloc cible
- En héritant d'une page et si vous écrivez du code à l'extérieur des blocs vous aurez le message suivant qui est très explicite.



Héritage 5- Récupérer le contenu d'un bloc parent

- Pour récupérer le contenu d'un bloc père il suffit d'utiliser la méthode `parent()`

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

{% block body %}

    {{ parent() }}<br>
    Bonjour J'ai pris mon indépendance et j'ai défini mon propre body

{% endblock %}
```

← → ↺ 127.0.0.1/symfoRT4/web/app_dev.php/as/fils

Bienvenu dans le body du bloc principal
Bonjour J'ai pris mon indépendance et j'ai défini mon propre body

- Ceci peut être aussi appliqué pour toute la hiérarchie

```
{% extends 'Rt4AsBundle:Default:fils.html.twig' %}

{% block body %}

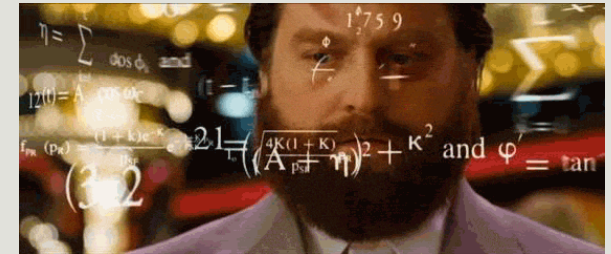
    {{ parent() }}<br>
    Bonjour J'ai pris moi aussi mon indépendance <b> je suis le petit fils </b> et j'ai défini mon propre body

{% endblock %}
```

← → ↺ 127.0.0.1/symfoRT4/web/app_dev.php/as/ptifils

Bienvenu dans le body du bloc principal
Bonjour J'ai pris mon indépendance et j'ai défini mon propre body
Bonjour J'ai pris moi aussi mon indépendance je suis le petit fils et j'ai défini mon propre body

Exercice



- Aller dans la page base.html.twig
- Ajouter y via Bootstrap une partie header avec un menu.
- Ajouter une partie footer.
- Gérer vos blocs de sorte que vous ayez le maximum de flexibilité.
- Exemple :

[Navbar](#) [Home](#) [Features](#) [Pricing](#) [Disabled](#)

799
392
19
380
214

Nombre de valeurs : 5
Min : 19
Max : 799
Somme : 1804
Moyenne : 360.8

Copyright © Your Website

Inclusion de Template

- Afin d'inclure un Template ou un fragment de code dans un autre template on utilise la syntaxe suivante :

```
{% include 'Dossier/nomTemplate' with {'labelParam1':  
    param1, 'labelParam2': param2,... } %}
```

- Le chemin commence par le dossier Template
- Exemple :

```
{{ include('Default/section.html.twig', {'Section': section})
```

Inclusion de Contrôleur

Que faire si on veut inclure un template dynamique ? Un template des meilleurs ventes un template des articles les plus vus, les derniers cours postés ...

La solution consiste à afficher la valeur de retour de la fonction `render` qui prend en paramètres le contrôleur à appeler et les attributs qu'il attend de recevoir.

Inclure un contrôleur sans ou avec des paramètres.

Syntaxe :

```
{{ render(controller('StringSyntaxControllerName::function', {'labelParam1':  
param1,'labelParam2': param2,... }))) }}
```

Exemple

```
{{ render(controller('App\\Controller\\PersonneController::List', { 'page': 10 }))) }}
```

Génération de liens avec TWIG

- Twig permet de générer des liens à partir des noms de root en utilisant la fonction `path`.

```
rt4_as_homepage:
  path:      /hello/{name}
  defaults: { _controller: Rt4AsBundle:Default:index }

rt4_as_base:
  path:      /base
  defaults: { _controller: Rt4AsBundle:Default:base }

rt4_as_fils:
  path:      /fils
  defaults: { _controller: Rt4AsBundle:Default:fils }

rt4_as_ptifils:
  path:      /ptifils
  defaults: { _controller: Rt4AsBundle:Default:petitFils }
```

```
{% extends 'Rt4AsBundle:Default:fils.html.twig' %}

{% block body %}

    {{ parent() }}<br>
    Bonjour J'ai pris moi aussi mon indépendance <b> je suis le petit fils </b> et j'ai défini mon propre body <br>

    <a href="{{ path('rt4_as_base') }}">mon grand père est ici</a><br>
    <a href="{{ path('rt4_as_fils') }}">mon père est ici </a><br>

{% endblock %}
```

Génération de liens paramétrable avec TWIG

Si le lien contient des paramètres, on garde la même syntaxe de la fonction `path` et on y ajoute comme deuxième paramètre un tableau contenant les paramètres passer à la route.

Syntaxe : `{{ path('root', { 'param1': param1, 'param2': param2,... }) }}`

```
rt4_as_homepage:
  path: /hello/{name}
  defaults: { _controller: Rt4AsBundle:Default:index }

rt4_as_base:
  path: /base
  defaults: { _controller: Rt4AsBundle:Default:base }

rt4_as_fils:
  path: /fils
  defaults: { _controller: Rt4AsBundle:Default:fils }

rt4_as_ptifils:
  path: /ptifils
  defaults: { _controller: Rt4AsBundle:Default:petitFils }
```

```
{% extends 'Rt4AsBundle:Default:fils.html.twig' %}

{% block body %}

    {{ parent() }}<br>
    Bonjour J'ai pris moi aussi mon indépendance <b> je suis le petit fils </b> et j'ai défini mon propre body <br>

    <a href="{{ path('rt4_as_base') }}">mon grand père est ici</a><br>
    <a href="{{ path('rt4_as_fils') }}">mon père est ici </a><br>
    <a href="{{ path('rt4_as_homepage', {name: 'aymen'}) }}">Bonjour </a><br>

{% endblock %}
```

Génération de liens absolus

- Pour générer l'url absolue nous utilisons la fonction `url`. Elle prend en paramètre le nom de la root souhaitée. Cette fonctionnalité peut être utile par exemple si vous envoyez un lien par mail. Ce lien ne peut pas être relatif sinon il sera traité relativement à la ou il est exécuté et non relativement à votre serveur.

```
{% extends 'Rt4AsBundle:Default:files.html.twig' %}

{% block body %}

    {{ parent() }}<br>
    Bonjour J'ai pris moi aussi mon indépendance <b> je suis le petit fils </b> et j'ai défini mon propre body <br>

    <a href="{{ path('rt4_as_base') }}">mon grand père est ici</a><br>
    <a href="{{ path('rt4_as_fils') }}">mon père est ici </a><br>
    <a href="{{ path('rt4_as_homepage', {name: 'aymen'}) }}">Bonjour </a><br>

    url relative : {{ path('rt4_as_fils') }} <br>
    url absolue : {{ url('rt4_as_fils') }}

{% endblock %}
```

Bienvenu dans le body du bloc principal

Bonjour J'ai pris mon indépendance et j'ai défini mon propre body

mon père est ici

Bonjour J'ai pris moi aussi mon indépendance **je suis le petit fils** et j'ai défini mon propre body

mon grand père est ici

mon père est ici

Bonjour

url relative : /symfoRT4/web/app_dev.php/as/fils

url absolue : http://127.0.0.1/symfoRT4/web/app_dev.php/as/fils

Asset

- Pour générer le path d'un fichier (img, css, js,...) nous utilisons la fonction `asset`.
- Cette fonction permet la portabilité de l'application en permettant une génération automatique du path du fichier indépendamment de l'emplacement de l'application.
- Exemple :
- Si l'application est hébergé directement dans la racine de l'hôte alors le path des images est `/images/nomImg`.

Asset

➤ Si l'application est hébergé dans un sous répertoire de l'hôte alors le path des images est /**nomApp**/images/nomImg.

Syntaxe :

asset('ressource')

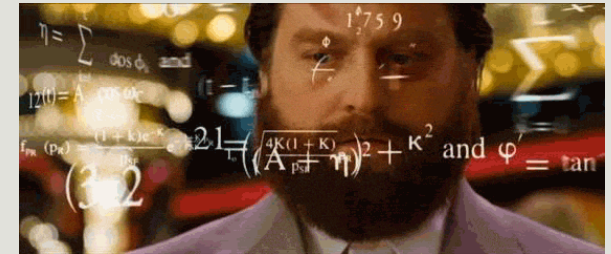
Exemples

```

```

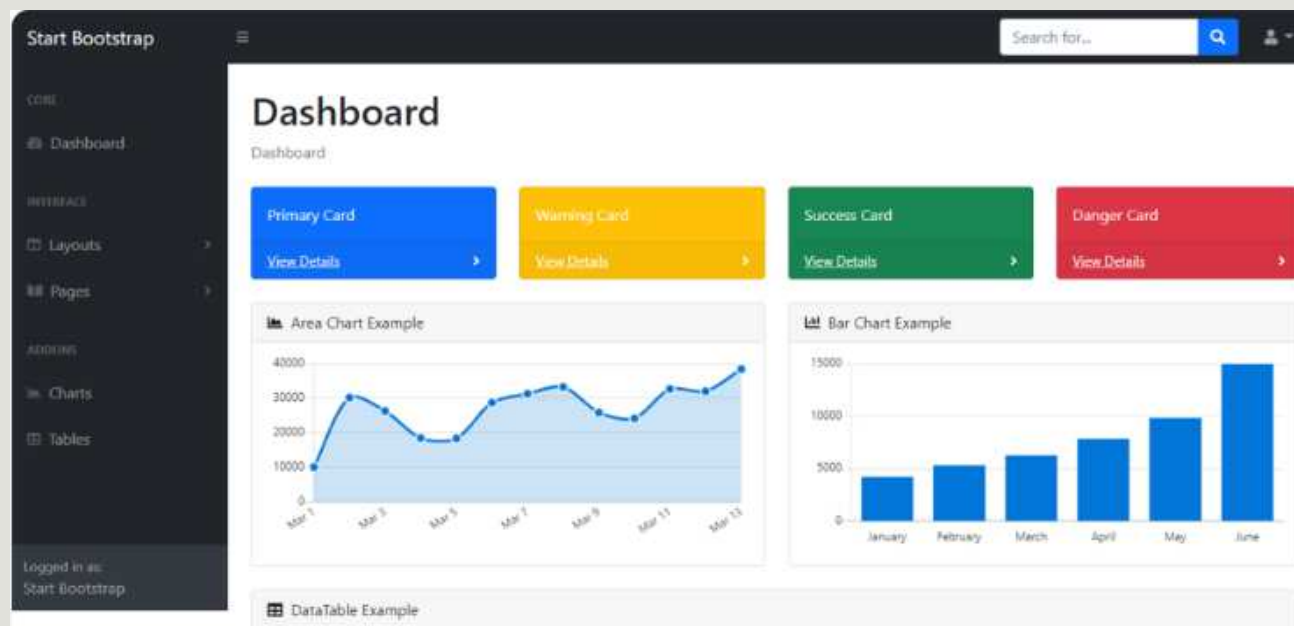
```
<link href="{{ asset('css/blog.css') }}" rel="stylesheet" type="text/css" />
```

Exercice

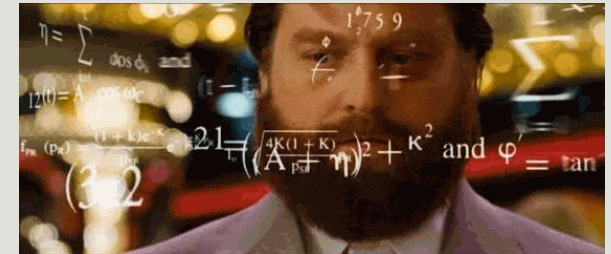


- Intégrer le template du lien suivant

<https://themewagon.com/themes/free-bootstrap-5-admin-template-sb-admin/>



Exercice



- Ajouter deux block. Un block Header et un block footer contenant le header et le footer de votre template.
- Créer deux twig header.html.twig et footer.html.twig.
- Mettez y le code du header et du footer.
- Incluez le header à travers un controller
- Incluez le footer à travers le twig directement

Accéder aux variables globales

A chaque requête, Symfony vous fournit une variable globale `app` dans votre Template. Cette variable vous permet d'accéder à plusieurs informations très utiles.

1. `app.session` que nous avons vu dans le skill Controller nous permet de récupérer la `session courante`. Elle a comme valeur `null` en cas d'absence de session.
2. `app.user` permet de récupérer `l'utilisateur courant` connecté et `null` si aucun utilisateur n'est connecté.
3. `app.environment` : permet de récupérer l'environnement courant, e.g. dev, prod, test.
4. `app.debug` retourne true si on est dans le debug mode false sinon.

Débugger avec dump

- Afin de déboguer les variables dans votre page TWIG vous pouvez utiliser la fonction dump.
- Exemple :

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

{% block body %}

    {{ dump(app) }}

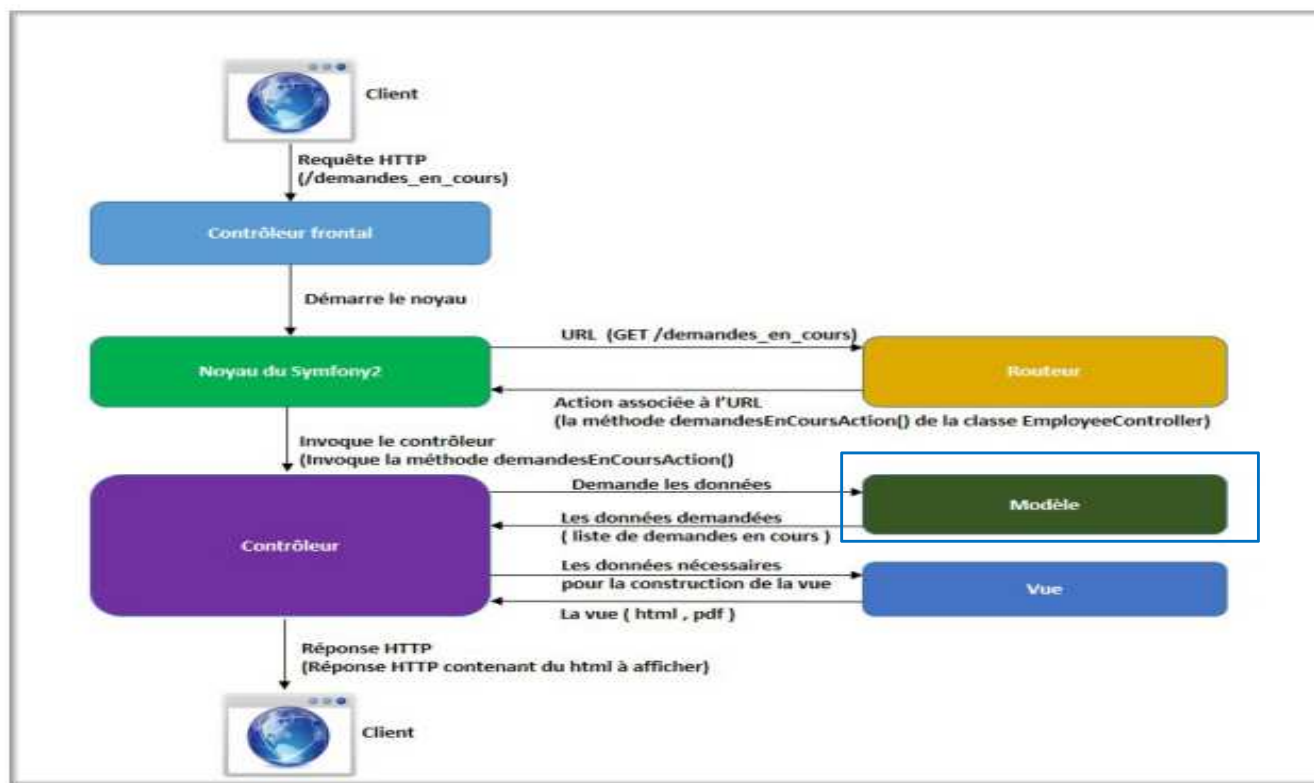
{% endblock %}
```

Symfony 6

L'ORM DOCTRINE

AYMEN SELLAOUTI

Introduction (1)

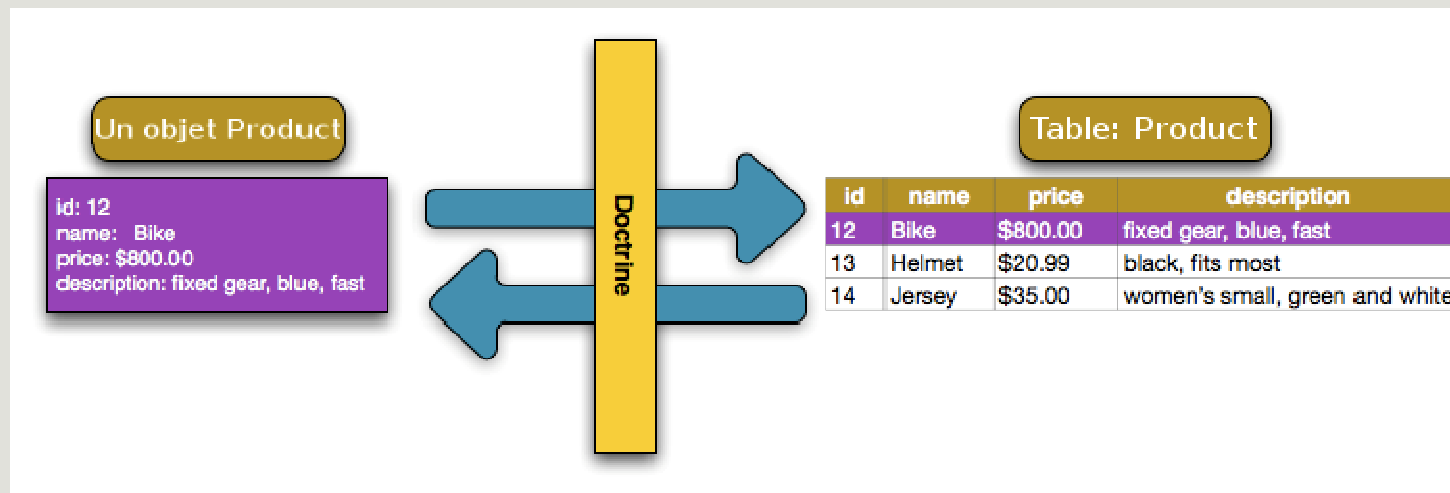


Introduction (2)

- ORM : Object Relation Mapper
- Couche d'abstraction
- Gérer la persistance des données
- Mapper les tables de la base de données relationnelle avec des objets
- Crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.
- Propose des méthodes prédéfinies



Introduction (3)



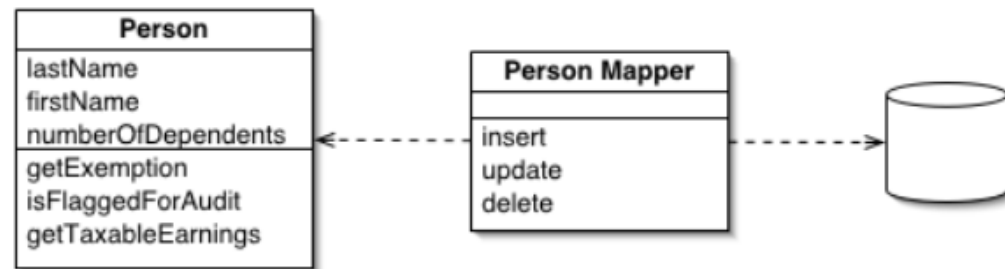
- Doctrine : ORM le plus utilisé avec Symfony2
- Associe des classes PHP avec les tables de votre BD (mapping)

Fonctionnement de Doctrine

Doctrine utilise deux design pattern objet :

- Data Mapper
- Unit of Work

Data Mapper



- C'est la **couche entre les objets (entités) et les tables de la base de données**.
- Dans le cas de PHP elle **synchronise** les données stockées en base de données avec vos objets PHP.
- Elle se charge **d'insérer** et de **mettre à jour** les données de la base en se basant sur le contenu des propriété de votre objet.
- Elle peut aussi **supprimer** des enregistrement de la base de données.
- Elle peut aussi **hydrater** vos objets en utilisant les informations contenues dans la base de données.
- Ceci permet d'avoir une **abstraction complète** de la base de données vu que les objets sont indépendants du système de stockage. C'est le Data Mapper qui se charge de ça.
- Doctrine implémente ce design pattern via l'objet **EntityManager**.

Patterns of Enterprise Application Architecture

Unit of Work

Unit of Work
registerNew(object)
registerDirty(object)
registerClean(object)
registerDeleted(object)
commit
rollback

- Afin d'éviter une multitude de petite requêtes envoyé à votre base de données, et de garder un historique des requêtes effectuées au niveau de votre base de données, Doctrine utilise le design pattern **Unit of work**.
- Pour une raison de performance et d'intégrité, La synchronisation effectuée par l'Entity Manager ne se fait pas pour chaque changement avec la base de données.
- Le design pattern **Unit of Work** permet de gérer l'état des différents objets hydratés par l'Entity Manager.
- Une transaction est ouverte regroupant l'ensemble des opérations. Le commit de cette transaction déclenchera l'exécution de l'ensemble de ses requêtes.
- En cas d'échec l'ensemble des requêtes et annulées.

Les entités (1)

- Objet PHP
- Les entités représente les objets PHP équivalentes à une table de la base de données.
- Une entité est généralement composée par les attributs de la tables ainsi que leurs getters et setters
- Manipulable par l'ORM

Les entités (2)

Exemple

```
<?php

namespace Rt4\AsBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Etudiant
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="Rt4\AsBundle\Entity\EtudiantRepository")
 */
class Etudiant
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
```

```
/**
 * @var integer
 *
 * @ORM\Column(name="numEtudiant", type="integer")
 */
private $numEtudiant;

/**
 * @var integer
 *
 * @ORM\Column(name="cin", type="integer")
 */
private $cin;

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

/**
 * @var string
 *
 * @ORM\Column(name="prenom", type="string", length=255)
 */
```

```
private $prenom;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="dateNaissance", type="date")
 */
private $dateNaissance;

/**
 * Get id
 *
 * @return integer
 */
public function getId()
{
    return $this->id;
}

/**
 * Set numEtudiant
 *
 * @param integer $numEtudiant
 * @return Etudiant
 */
```

```
/**
 * Set numEtudiant
 *
 * @param integer $numEtudiant
 * @return Etudiant
 */
public function setNumEtudiant($numEtudiant)
{
    $this->numEtudiant = $numEtudiant;

    return $this;
}

/**
 * Get numEtudiant
 *
 * @return integer
 */
public function getNumEtudiant()
{
    return $this->numEtudiant;
}
```

Configuration des entités

- Configuration Externe : YAML, XML, PHP
- Configuration Interne : annotations, **attributs (php8)**
- Choix de la configuration ?
- Deux Visions :
 - Pro-Externe
 - Séparation complète des fonctionnalités spécialement lorsque l'entité est conséquente
 - Pro-Interne
 - Plus facile et agréable de chercher dans un seul fichier l'ensemble des information, plus de visibilité

Mapping : Annotation et attributs des entités (1)

- **Rôle** : Faire le lien entre les entités et les tables de la base de données
- Lien à travers les **métadonnées**
- **Remarque** : Un seul format par bundle (impossibilité de mélanger)
- **Syntaxe** :
 - **/****
 - *** les différentes annotations**
 - ***/**
- **Remarque** : Afin d'utiliser les annotations il faut ajouter :

```
use Doctrine\ORM\Mapping as ORM;
```

Mapping : Annotation et attributs des entités (2)

Entity

Permet de définir un **objet** comme une **entité**

Applicable sur une **classe**

Placée avant la définition de la classe en PHP

Syntaxe : @ORM\Entity

```
#[ORM\Entity]
```

Paramètres :

repositoryClass (facultatif). Permet de préciser le namespace complet du repository qui gère cette entité.

Exemple :

```
@ORM\Entity(repositoryClass="Rt4\AsBundle\Entity\animalRepository")
```

```
#[ORM\Entity(repositoryClass: PersonneRepository::class)]
```

Mapping : Annotation et attributs des entités (3)

Table

Permet de spécifier le nom de la table dans la base de données à associer à l'entité

Applicable sur une classe et placée avant la définition de la classe en PHP

Facultative sans cette annotation le nom de la table sera automatiquement le nom de l'entité

Généralement utilisable pour ajouter des préfixes ou pour forcer la première lettre de la table en minuscule

Syntaxe : @ORM\Table()

Exemple :

```
/**  
 * @ORM\Table('animal')  
 */
```

```
#[ORM\Table('table')]
```

```
@ORM\Entity(repositoryClass="Rt4\\AsBundle\\Entity\\animalRe  
pository")  
*/
```

Mapping : Annotation et attributs des entités

(4)

Column

Permet de définir les caractéristiques de la colonne concernée

Applicable sur un attribut de classe juste avant la définition PHP de l'attribut concerné.

Syntaxe : @ORM\Column()

Exemple :

```
/**  
 * #[ORM\Column(type: Types::STRING, length: 255)]  
 * @ORM\Column(param1="valParam1", param2="valParam2")  
 */
```

Mapping : Annotation et attributs des entités (5)

Les paramètres de Column

Paramètre	Valeur par défaut	Utilisation
type	string	Définit le type de colonne comme nous venons de le voir.
name	Nom de l'attribut	Définit le nom de la colonne dans la table. Par défaut, le nom de la colonne est le nom de l'attribut de l'objet
length	255	Définit la longueur de la colonne (pour les strings).
unique	false	Définit la colonne comme unique (Exemple : email).
nullable	false	Permet à la colonne de contenir des NULL.
precision	0	Définit la précision d'un nombre à virgule(decimal)
scale	0	le nombre de chiffres après la virgule (decimal)

Mapping : Annotation et attributs des entités (6)

Les types de Column

Type Doctrine	Type SQL	Type PHP	Utilisation
string	VARCHAR	string	Toutes les chaînes de caractères jusqu'à 255 caractères.
integer	INT	integer	Tous les nombres jusqu'à 2 147 483 647.
smallint	SMALLINT	integer	Tous les nombres jusqu'à 32 767.
bigint	BIGINT	string	Tous les nombres jusqu'à 9 223 372 036 854 775 807.
boolean	BOOLEAN	boolean	Les valeurs booléennes true et false.
decimal	DECIMAL	double	Les nombres à virgule.

Mapping : Annotation et attributs des entités (7)

Les types de Column

Type Doctrine	Type SQL	Type PHP	Utilisation
date ou datetime	DATETIME	objet DateTime	Toutes les dates et heures.
time	TIME	objet DateTime-	Toutes les heures.
text	CLOB	string	Les chaînes de caractères de plus de 255 caractères.
object	CLOB	Type de l'objet stocké	Stocke un objet PHP en utilisant serialize/unserialize.
array	CLOB	array	Stocke un tableau PHP en utilisant serialize/unserialize.
float	FLOAT	double	Tous les nombres à virgule. Attention, fonctionne uniquement sur les serveurs dont la locale utilise un point comme séparateur.

Mapping : Annotation et attributs des entités (8)

Conventions de Nommage

Même s'il reste facultatif, le champs « name » doit être modifié afin de respecter les conventions de nommage qui diffèrent entre ceux de la base de données et ceux de la programmation OO.

- Les noms de classes sont écrites en « **Pascal Case** » TheEntity.
- Les attributs de classes sont écrites en « **camel Case** » oneAttribute
- Les noms des tables et des colonnes en SQL sont écrites en minuscules, les mots sont séparés par « _ » one_table, one_column.

Gestion de la base de données (1)

Configuration de l'application

- Afin de configurer la base de données de l'application il faut renseigner les champs dans le fichier .env qui est renseigné dans le fichier [config/packages/doctrine.yml](#)

```
doctrine:
  dbal:
    url: '%env(resolve:DATABASE_URL)%'
    # IMPORTANT: You MUST configure your server version,
    # either here or in the DATABASE_URL env var (see .env file)
    #server_version: '13'
  orm:
    auto_generate_proxy_classes: true
    naming_strategy:
doctrine.orm.naming_strategy.underscore_number_aware
    auto_mapping: true
    mappings:
      App:
        is_bundle: false
        dir: '%kernel.project_dir%/src/Entity'
        prefix: 'App\Entity'
        alias: App
```

Gestion de la base de données (1)

Configuration de l'application

```
# DATABASE_URL="sqlite:///kernel.project_dir%/var/data.db"  
DATABASE_URL="mysql://root:@127.0.0.1:3306/sf1test?serverVersion=10.4.24-  
MariaDB&charset=utf8mb4"  
#DATABASE_URL="postgresql://username:pwd@127.0.0.1:5432/app?serverVersion=10.4.24-  
MariaDB&charset=utf8"
```

Gestion de la base de données (2)

Création de base de données

Afin de créer la base de données du projet 2 méthodes sont utilisées :

- Manuelle en utilisant le SGBD (le nom de la BD doit être le même que celui mentionné dans le fichier doctrine.yml)
- En utilisant la ligne de command avec la command suivante :
 - `php bin/console doctrine:database:create`
 - `symfony console doctrine:database:create`
 - Une base de données avec les propriétés mentionnées dans .env sera automatiquement générée

Gestion de la base de données (3)

Génération des entités

Deux méthodes pour générer les entités :

- Méthode manuelle (non recommandée)
 - Créer la classe
 - Ajouter le mapping
 - Ajouter les getters et les setters (manuellement ou en utilisant la commande suivante :

`php bin/console make:entity --regenerate App`
(elle crée les getters et les setters de toutes les entités)

Gestion de la base de données (3)

Génération des entités

Deux méthodes pour générer les entités :

- Méthode en utilisant les commandes
 - Il suffit de lancer la commande suivante :
`php bin/console make:entity`
`symfony console make:entity`
 - Ajouter les attributs ainsi que les paramètres qui vont avec
 - Une fois terminé, Doctrine génère l'entité avec toutes les métadonnées de mapping

Gestion de la base de données

Les migrations

- Les **migrations** sont une nouvelle façon utilisée par Symfony 4 afin de gérer les mises à jours et évolutions de votre base de données.
- Ayant une images des différentes évolutions de votre base de donnée, vous pouvez annuler des changements ou passer d'une version à une autre.

Gestion de la base de données

Les migrations : les commandes

- Pour connaître l'état de vos fichiers de migrations, vous pouvez utiliser la commande
- `php bin/console doctrine:migration:status`

```
>> Name: Application Migrations
>> Database Driver: pdo_mysql
>> Database Host: 127.0.0.1
>> Database Name: sf4Forma
>> Configuration Source: manually configured
>> Version Table Name: migration_versions
>> Version Column Name: version
>> Migrations Namespace: DoctrineMigrations
>> Migrations Directory: G:\symfony\my-project/src/Migrations
>> Previous Version: 2019-06-03 14:23:41 (20190603142341)
>> Current Version: 2019-06-04 15:20:25 (20190604152025)
>> Next Version: Already at latest version
>> Latest Version: 2019-06-04 15:20:25 (20190604152025)
>> Executed Migrations: 2
>> Executed Unavailable Migrations: 0
>> Available Migrations: 2
>> New Migrations: 0
```

Gestion de la base de données

Les migrations : les commandes

Créer un fichier de migration

- Pour créer un fichier de migration, utilisez la commande :
`symfony console doctrine:migration:generate.`
- Ceci vous créera un fichier de migration vide de tout traitement.
- Si vous avez besoin d'écrire votre propre code de migration vous pouvez le faire.

Les migrations

Créer un fichier de migration pour adapter votre base de données à vos entités

- Pour créer un fichier de migration, utilisez la commande :

`symfony console doctrine:migration:diff`

- Vous pouvez aussi utiliser

`php bin/console make:migration`

- Cette commande fait appel à la commande précédente.

```
<?php
//...

final class Version20220920155201 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE personne (id INT AUTO_INCREMENT NOT NULL, name VARCHAR(255) NOT NULL,
age SMALLINT NOT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('DROP TABLE personne');
        $this->addSql('DROP TABLE test');
        $this->addSql('DROP TABLE messenger_messages');
    }
}
```

Migration

Exécuter une migration particulière

Afin d'exécuter une migration particulière que ce soit la méthode up ou la méthode down utiliser la méthode suivante :

```
php bin/console doctrine:migration:execute 'DoctrineMigrations\<numVersion>' --fct
```

Exemple

```
symfony console doctrine:migrations:execute --up 'DoctrineMigrations\Version20220920155201'
```

Résumé

:**diff** [diff] Génère une migration en comparant la base de données avec les informations de mapping.

:**execute** [execute] Exécute une migration manuellement.

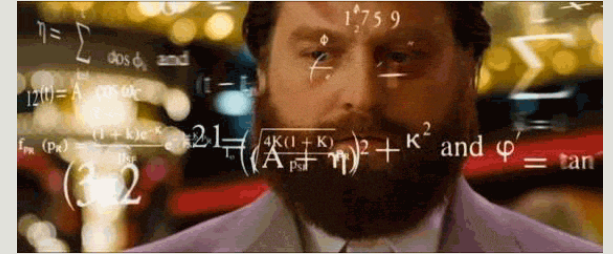
:**generate** [generate] Crée une classe de Migration.

:**migrate** [migrate] Effectue une migration vers le fichier de migration le plus récent ou celui spécifié.

:**status** [status] Affiche le status des migrations.

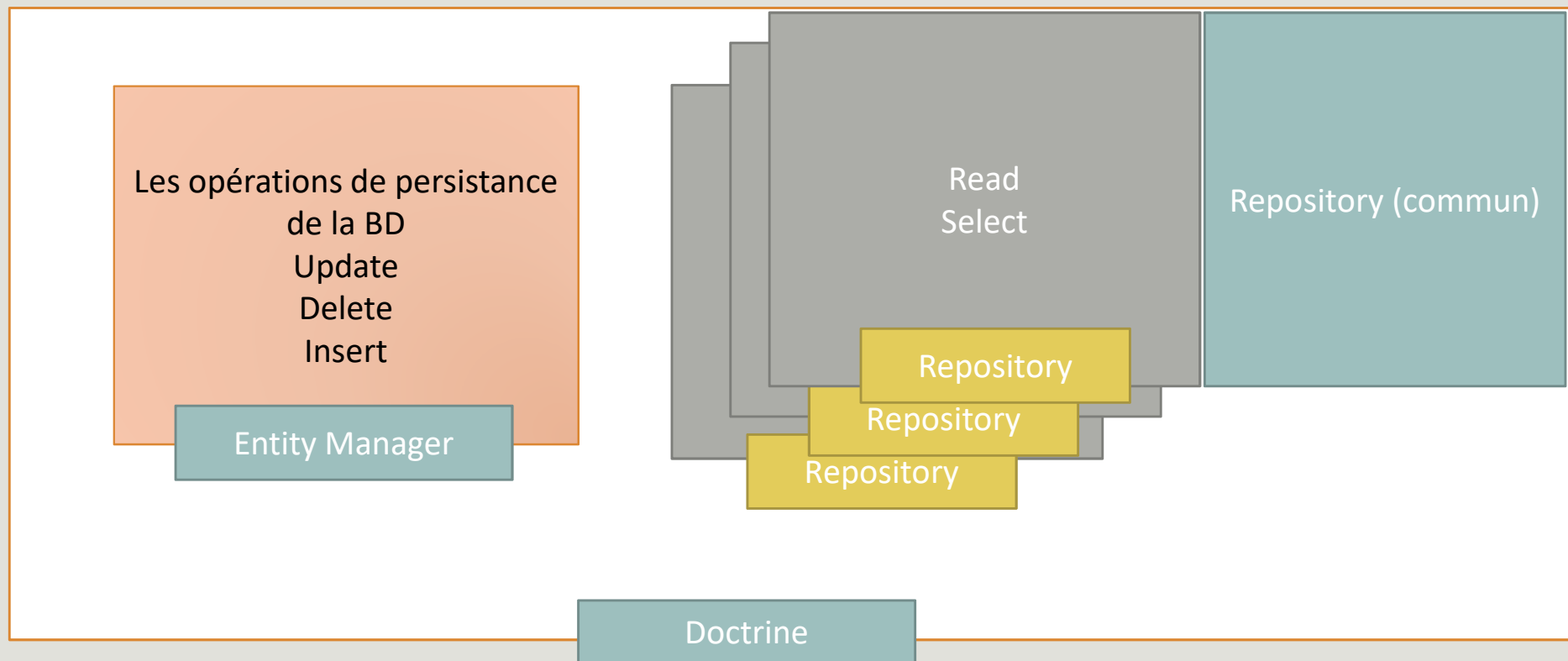
:**version** [version] Ajoute et supprime manuellement des versions à partir de la version en base.

Exercice



- Créer votre base de données à travers la ligne de commande.
- Générer une Entité Personne.
- Cette entité contient un attribut id, un attribut name, firstname, age.

Doctrine



Le service Doctrine

- **Rôle** : permet la gestion des données dans la BD : **persistance** des données et **consultation** des données.

Avant Symfony 6

Méthode :

- `$this->get('doctrine');`
- `$this->getDoctrine();` //helper (raccourcie de la classe Controller)

Le service Doctrine offre deux services pour la gestion d'une base de données :

- Le Repository qui se charge des requêtes Select
- L'EntityManager qui se charge de persister la base de données donc de gérer les requêtes INSERT, UPDATE et DELETE.

A partir de Symfony 6

- Doctrine **n'est plus accessible via les helpers**. Vous devez l'injecter via la classe **ManagerRegistry** en le spécifiant au niveau **méthode ou au niveau constructeur**.

```
public function index(ManagerRegistry $doctrine): Response {  
}
```

```
public function __construct(private ManagerRegistry $doctrine)  
{}
```

Le service EntityManager

Rôle : L'interface ORM proposée par doctrine offrant des méthodes prédéfini pour persister dans la base ou pour mettre à jour ou supprimer une entité.

Méthode :

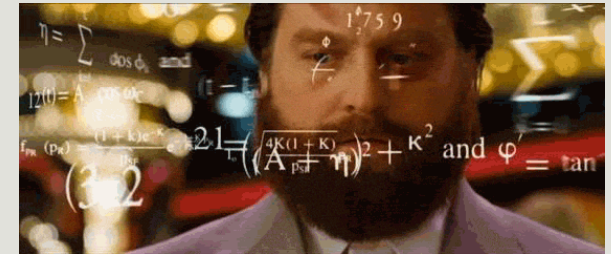
- `$EntityManager = $this->get('doctrineorm.entity_manager')`
- `$doctrine->getManager();`
- L'injecter en tant que service (à voir dans la partie service)

Le service EntityManager

Insertion des données

- Enregistrement des données
- Etant un ORM, Doctrine traite les objets PHP
- Pour enregistrer des données dans la BD il faut préparer les objets contenant ces données la
- La méthode `persist()` de l'entityManager permet d'associer les objets à persister avec Doctrine
- Afin d'exécuter les requêtes sur la BD (enregistrer les données dans la base) il faut utiliser la méthode `flush()`
- L'utilisation de flush permet de profiter de la force de Doctrine qui utilise les Transactions
- La persistance agit de la même façon avec l'ajout (insert) ou la mise à jour (update)

Exercice



- Créer un contrôleur `PersonneController`
- Créer une action qui permet l'ajout d'une `Personne` au niveau de la base de données.
- Les données seront introduites via la route.
- Une fois la personne ajoutée, une page contenant ses informations sera affichée.
- Faites de même pour la mise à jour, faite en sorte que cette action prenne l'id de la personne à modifier et les nouvelles valeurs.

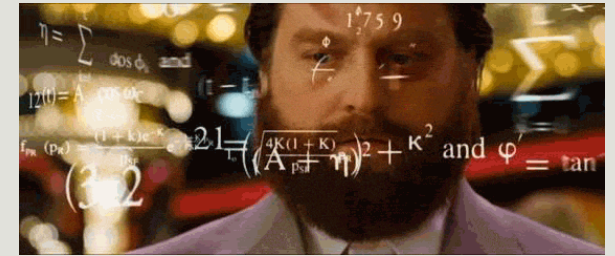
Le service EntityManager

Suppression d'une entité

Suppression d'une entité

La méthode `remove()` permet de supprimer une entité

Exercice



- Créer une action qui permet la suppression d'une personne en utilisant son id.
- Si la personne n'existe pas un message d'erreur est affiché

Une petite parenthèse : Fixtures

- Les fixtures sont utilisées pour charger des données « fake » au sein de votre base de données pour tester les fonctionnalités que vous avez développé.
- Installer le bundle responsable des Fixtures.
- `composer require --dev orm-fixtures`
- Créer une classe `VotreEntitéFixture` à l'aide de la commande :
- `symfony console make:fixtures`

Une petite parenthèse : Fixtures

- Implémenter la méthode load avec le fonctionnement que vous voulez pour charger vos données.
- Lancer vos fixtures : `php bin/console doctrine:fixtures:load`, cette commande effacera le contenu de votre Base de données.
- Si vous voulez garder la base ajouter l'option `--append`
`php bin/console doctrine:fixtures:load --append`

Fixture exemple

```
<?php

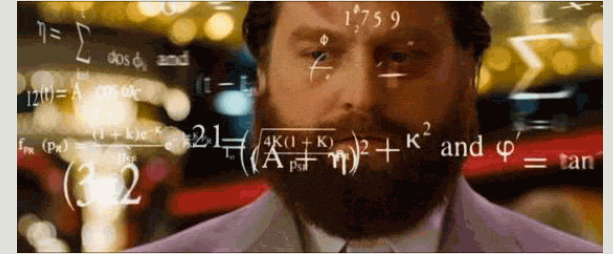
namespace App\DataFixtures;

use App\Entity\Personne;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Common\Persistence\ObjectManager;

class PersonneFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        for($i=0; $i<10; $i++) {
            $personne = new Personne();
            $personne->setAge(mt_rand(20, 80));
            $personne->setName("personne $i");
            $personne->setJob("Job $i");
            $manager->persist($personne);
        }

        $manager->flush();
    }
}
```

Exercice



- Créer un fixture permettant d'ajouter quelques personnes dans la base de données.
- Vous pouvez utiliser le composant faker

<https://fakerphp.github.io/>

- `composer require fakerphp/faker`

Le Repository

➤ Les repositories (dépôts)

➤ Des classes PHP dont le rôle est de permettre à l'utilisateur de récupérer des entités d'une classe donnée.

➤ Syntaxe :

➤ Pour accéder au repository de la classe Maclasse on utilise Doctrine ou on l'injecte (voir dans la partie Service)

➤ `$repo = $doctrine->getRepository(ClassName::class);`

Le Repository

- Quelques méthodes offertes par le repository :
- `$repository->findAll();` // récupère tous les entités (enregistrements) relatifs à l'entité associé au repository
- `$repository->find($id);` // requête sur la clé primaire
- `$repository->findBy();` // retourne un ensemble d'entités avec un filtrage sur plusieurs critères (nbre donné)
- `$repository->findOneBy();` // même principe que `findBy` mais une seule entité
- `$repository->findByNomPropriété();`
- `$repository->findOneByNomPropriété();`

Le Repository

findAll

➤ findAll()

➤ **Rôle** : retourne l'ensemble des entités qui correspondent à l'entité associée au repository. Le format du retour est un Array

➤ Exemple :

➤ //On récupère le repository de l'entity manager correspondant à l'entité Etudiant

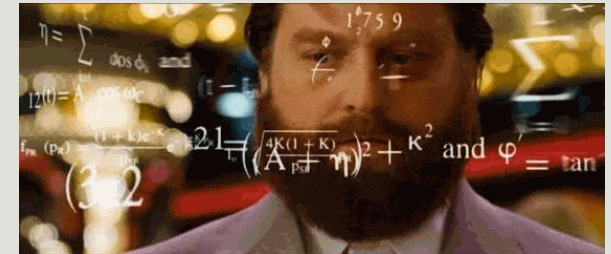
➤ \$repository = \$doctrine->getRepository(**ClassName::class**) ;

➤ //On récupère la liste des étudiants




























➤ \$listAdverts = \$repository->findAll();

➤ Généralement le tableau obtenu est passé à la vue (TWIG) et est affiché en utilisant un foreach

Exercice



- Créer une page list.html.twig
- Faire en sorte que cette page affiche la liste des personnes de votre base de données.

Template		
Sellaouti Aymen Sellaouti Age : 39.   	Jelassi Nidhal Jelassi Age : 38.   	Sellaouti Skander Sellaouti Age : 3.   
Manon Le Michaud Susanne Manon Le Michaud Age : 35.   	Denise Chartier Martin Denise Chartier Age : 54.   	Bernard Bousquet André Bernard Bousquet Age : 47.   
Nicolas Costa Astrid Nicolas Costa Age : 28.   	Camille Marty Thérèse Camille Marty Age : 47.   	Sylvie Merle Agnès Sylvie Merle Age : 41.   

Le Repository

find

➤ `find($id)`

➤ **Rôle** : retourne l'entité qui correspond à la clé primaire passé en argument. Généralement cette clé est l'id.

➤ Exemple :

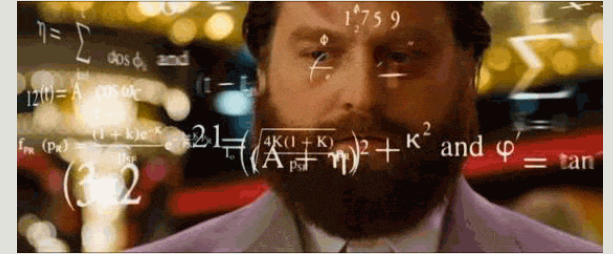
➤ `//On récupère le repository de l'entity manager correspondant à l'entité Etudiant`

➤ `$repo = $doctrine->getRepository(Etudiant::class);`

➤ `//on lance la requête sur l'étudiant d'id 2`

➤ `$etudiant = $repository->find(2);`

Exercise



- Créer une page `profil.html.twig`
- Faire en sorte que cette page affiche le profil d'une personne selon son id.

Le Repository

findBy

➤ findBy()

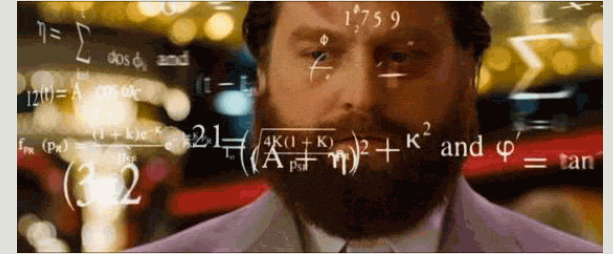
➤ **Rôle** : retourne l'ensemble des entités qui correspondent à l'entité associée au repository comme findAll sauf qu'elle permet d'effectuer un filtrage sur un ensemble de critère passés dans un Array. Elle offre la possibilité de trier les entités sélectionnées et facilite la pagination en offrant un nombre de valeur de retour.

➤ **Syntaxe**: `$repository->findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null);`

➤ **Exemple** : `$repository = $doctrine->getRepository(Etudiant::class);`

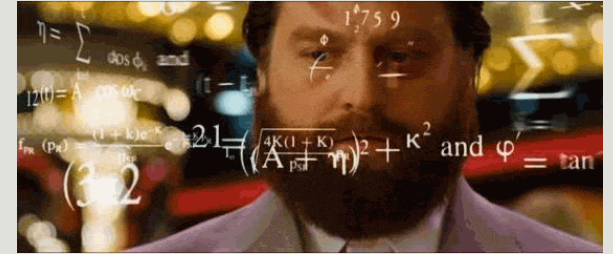
➤ `$listeEtudiants = $repository->findBy(array('section' => 'RT4','nom' => 'Mohamed'), array('date' => 'desc'),10, 0);`

Exercice



- Créer une méthode qui permet d'afficher la liste des personnes d'un nom donné ordonnée par prénom.
- Le nombre maximum de résultat à afficher est de 5.

Exercice



- Créer une méthode qui permet d'afficher la liste des personnes d'un nom donné ordonnée par prénom.
- Le nombre maximum de résultat à afficher est de 5.

Le Repository

findOneBy

➤ findOneBy()

➤ **Rôle** : Même principe que FindBy mais en retournant une seule entité ce qui élimine automatiquement les paramètres d'ordre de limite et d'offset

➤ Exemple :

➤ `$repository = $doctrine->getRepository(Etudiant::class);`

➤ `$Etud = $repository->findOneBy(
 array('section' => 'RT4','nom' => 'Mohamed')
);`

Le Repository

findByPropriété

➤ findByPropriété()

- Rôle : En remplaçant le mot **Propriété** par le **nom d'une des propriété de l'entité**, la fonction va faire le même rôle que **findBy** mais avec un seul critère qui est le nom de la propriété et sans les options.

Exemple :

- `$repository = $doctrine->getRepository(Etudiant::class);`
- `$listeEtudiants = $repository->findByNom('Aymen');`

Le Repository

findOneByPropriété

- `findOneByPropriété()`
- **Rôle** : En remplaçant le mot **Propriété** par le **nom d'une des propriété de l'entité**, la fonction va faire le même rôle que `findOneBy` mais avec un seul critère.
- **Exemple** :
- `$repository = $doctrine->getRepository(Etudiant::class);`
- `$listeEtudiants = $repository->findOneByNom('Aymen');`

Le Repository

Création de requêtes

- Les requêtes de doctrine sont écrites en utilisant le langage de doctrine le Doctrine Query Language **DQL** ou en utilisant un Objet créateur de requêtes le **CreateQueryBuilder**
- **createQuery** : Méthode de l'Entity Manager
- **CreateQueryBuilder** : Méthode du repository

DQL	≈	SQL
Classes + Propriétés		Tables + colonnes

Le Repository

CreateQuery et DQL

- Le DQL peut être défini comme une adaptation du SQL adapté à l'orienté objet et donc à DOCTRINE
- La requête est défini sous forme d'une chaine de caractère
- Afin de créer une requête DQL il faut utiliser la méthode `createQuery()` de l'EntityManager
- La méthode `setParameter('label','valeur')` permet de définir un paramètre de la requête

Le Repository

CreateQuery et DQL

- Pour définir **plusieurs paramètres** ou bien utiliser **setParameter** **plusieurs fois** ou bien la méthode `setParameters(array('label1','valeur1', 'label2','valeur2'),... 'labelN','valeurN'))`
- Une fois la requête créée la méthode **getResult()** permet de récupérer un tableau de résultat
- Le langage DQL est explicité dans le lien suivant :

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-query-language.html>

Le Repository : Création de requêtes createQuery

```
public function findPersonneByIntervalAge2 ($min, $max)
{
    $query = $this->_em->createQuery(
        'SELECT p
        FROM App\Entity\Personne p
        WHERE p.age >= :ageMin And p.age <= :ageMax
        ORDER BY p.age
        ASC'
    )
    ->setParameter('ageMin', $min)
    ->setParameter('ageMax', $max)
    ;
    return $query->execute();
}
```


Le Repository QueryBuilder

- **Constructeur** de requête DOCTRINE Alternative au DQL
- Accessible via le **Repository**
- Le résultat fourni par la méthode **getQuery** du **QueryBuilder** permet de **générer la requête en DQL**
- De même que le **createQuery**, une fois la requête créée la méthode **getResult()** permet de récupérer un tableau de résultat.

Le Repository QueryBuilder

- Afin de récupérer le QueryBuilder dans notre Repository, on utilise la méthode **createQueryBuilder**.
- Cette méthode récupère en paramètre l'alias de l'entité cible et offre la requête « select from » de l'entité en question.
- Généralement l'alias est la première lettre en minuscule du nom de l'entité
- Si aucun paramètre n'est passé a createQueryBuilder alors on aura une requête vide et il faudra tout cons

```
$qb=$this->_em->createQueryBuilder()  
    ->select ('t')  
    ->from ($this->_entityName, 'alias');
```

```
$qb=$this->createQueryBuilder('t')
```

Le Repository

QueryBuilder : Méthodes

➤ **from('entityName','entityAlias')**

➤ from(\$this->_entityName,'t')

➤ **where('condition')** permet d'ajouter le where dans la requête

➤ where('t.destination= :dest')

➤ **setParameter('nomParam',param)** permet d'ajouter la définition d'un des paramètres définis dans le where

➤ setParameter('dest',\$dest)

Le Repository

QueryBuilder : Méthodes

- **andWhere('condition')** permet d'ajouter d'autres conditions
 - `andWhere('t.statut = :status')`
- **orderBy('nomChamp','ordre')** permet d'ajouter un orderBy et prend en paramètre le champ à ordonner et l'ordre DESC ou ASC.
 - `orderBy('t.dateTransfert','DESC')`
- **setParameters(array(1=>'param1',2=>'param2'))**

Le Repository QueryBuilder : Méthodes

- **orWhere**
- **groupBy**
- **having**
- **andHaving**
- **orHaving**
- **leftJoin**
- **rightJoin**
- **Join**
- **innerJoin**
- **...**

Le Repository

QueryBuilder : Méthodes

- **getQuery()** : retourne la requête dql
- **getResult()** : retourne un tableau d'objets contenant le résultat
- **getOneOrNullResult()** : retourne le premier résultat ou Null
- **getSingleScalarResult()** : retourne un résultat sous format scalaire. Imaginer le use case ou vous voulez récupérer le COUNT ou la SUM d'un de vos objets.

Requêter des attributs spécifiques

- Afin de sélectionner des **propriétés particulières** utiliser la méthode select du **QueryBuilder**

Syntaxe :

```
->select ( 'SUM(u.age) as sumAge, AVG(u.age) as  
avgUserAge ' )
```

Réutilisation de la logique de vos requêtes

- Imaginer les uses cases suivants :
- Vous voulez sélectionner les formations d'un topic donné.
- Vous voulez sélectionner les formations d'un topic donné dont le nombre d'inscrits est inférieur à un nombre données.
- Vous voulez sélectionner les formations d'un topic donné dont le nombre d'inscrits est supérieur à un nombre données.
- Vous voulez sélectionner les formations d'un topic donné entre deux dates ...
- On remarque ici une redondance dans ces différentes requêtes.
- L'idée est donc d'isoler ce traitements redondant dans une méthode et de la réutiliser.


```

/**
 * @param $min
 * @param $max
 * @return mixed
 */
public function getPersonneByAge($min, $max) {

    $qb = $this->createQueryBuilder('p');
    $qb = $this->findByAge($qb, $min, $max);
    return $qb->getQuery()->getResult();
}

/**
 * @param QueryBuilder $qb
 * @param $min
 * @param $max
 * @return QueryBuilder
 */
private function findByAge(QueryBuilder $qb, $min, $max) {
    if($min) {
        $qb->andWhere('p.age > :minAge')
            ->setParameter('minAge', $min);
    }
    if($max) {
        $qb->andWhere('p.age < :maxAge')
            ->setParameter('maxAge', $max);
    }
    return $qb;
}

```

```
public function getCommunesByZipCode($zipCode = null, $like = null,
$type = null) {
    $qb = $this->createQueryBuilder('l')
        ->select('l.id, l.name, l.zipCode')
        ->where('l=1');

    if ($zipCode) {
        $qb = $qb->andWhere('l.zipCode = :zipCode ')
            ->setParameter('zipCode', $zipCode);
    }

    if ($like) {
        $qb = $qb->andWhere('l.name like :like ')
            ->setParameter('like', "%$like%");
    }

    return $qb->getQuery()->getArrayResult();
}
```

Gestion des relations entre les entités (1)

Les types de relation

Les entités de la BD présentent des relations d'association :

- A **OneToOne** B : à une entité A on associe une entité de B et inversement
- A **ManyToOne** B : à une entité B on associe plusieurs entité de A et à une entité de A on associe une entité de B
- A **ManyToMany** B : à une entité de A on associe plusieurs entité de B et inversement

Gestion des relations entre les entités (2)

Relation unidirectionnelle et bidirectionnelle

- La notion de navigabilité de UML est la source de la notion de relation unidirectionnelle ou bidirectionnelle
- Une relation est dite navigable dans les deux sens si les deux entité doivent avoir une trace de la relation.
- Exemple : Supposons que nous avons les deux classes CandidatPresidentielle et Electeur.
- L'électeur doit savoir à qui il a voté donc il doit sauvegarder cette information par contre le candidat pour cause d'anonymat de vote ne doit pas connaître les personnes qui ont voté pour lui.
- On aura donc un attribut Candidat dans la table Electeur mais pas de collection ou tableau nommé électeur dans la table CandidatPresidentielle. Ici on a une relation **unidirectionnelle**.

Création de la relation

- Vous pouvez créer votre relation via la console.
- Créer un attribut et mettez y comme type « **relation** ».
- Ceci va générer un attribut annoté avec la relation et les informations minimalistes nécessaires pour sa création.

```
/**  
 * @ORM\ManyToMany(targetEntity="App\Entity\Hobbies", inversedBy="personnes")  
 */  
private $hobbies;
```

Gestion des relations entre les entités (3)

OneToOne unidirectionnelle

- Relation **unidirectionnelle**
puisque Media ne référence
pas Etudiant

```
/**Entity **//  
Class Etudiant  
{  
// ...  
/**  
 * @ORM\OneToOne(targetEntity=  
targetEntity="App\Entity\Media"  
)  
 */  
    private $media;  
// ...  
}  
/**Entity **//  
Class Etudiant  
{  
// ...  
}
```

Gestion des relations entre les entités (4)

OneToOne Bidirectionnelle

- Si nous voulons qu'à partir du media on peut directement savoir à quel étudiant il appartient nous devons faire une relation bidirectionnelle
- Media aussi doit référencer Etudiant

```
/**Entity **//  
Class Etudiant  
{  
// ...  
/**  
 * @ORM\OneToOne(targetEntity= "App\Entity\Media")  
 */  
    private $media;  
// ...  
}  
/**Entity **//  
Class Media  
{  
/**  
 * @ORM\OneToOne(targetEntity=  
 "App\Entity\Etudiant",mappedBy="media")  
 */  
    private $etudiant;  
// ...  
}
```

Gestion des relations entre les entités (5)

ManyToOne Unidirectionnelle

- Relation **unidirectionnelle**
puisque Section ne référence
pas Etudiant

```
/**Entity **//  
Class Etudiant  
{  
    // ...  
    /**  
     * @ORM\ManyToOne(targetEntity="App\Entity\Section")  
     */  
    private $section;  
    // ...  
}  
/**Entity **//  
Class Section  
{  
    // ...  
}
```


Gestion des relations entre les entités (6)

OneToMany Bidirectionnelle

- Si nous voulons connaître dans l'objet section l'ensemble des étudiants qui lui sont affectés alors on doit avoir une relation bidirectionnelle
- Section aussi doit référencer Etudiant
- On aura une relation **OneToMany** côté Section puisqu'à « One » Section on a « Many » Etudiants.
- On doit ajouter l'attribut **mappedBy** côté **OneToMany** et **inversedBy** côté **ManyToOne**
- On doit spécifier dans le **constructeur** du **OneToMany** que l'attribut **mappé** est de type **ArrayCollection** en l'instanciant

```
/**Entity **//  
Class Section  
{  
  // ...  
  /**  
   * @ORM\OneToMany(targetEntity="App\Entity\Etudiant",  
   *mappedBy="section")  
   */  
  private $etudiants;  
  // ...  
  public function __construct() {  
    $this->etudiants = new ArrayCollection ();  
  }  
}/**Entity **//  
Class Etudiant  
{ //...  
  /**  
   * @ORM\ManyToOne(targetEntity="App\Entity\Section",  
   inversedBy="etudiants")  
   */  
  private $section;  
  // ...  
}
```

Gestion des relations entre les entités (7)

ManyToMany

- Relation **unidirectionnelle** puisque Cours ne référence pas Prof
- Ici on peut savoir quels sont les cours de chaque étudiant mais pas l'inverse (on peut l'extraire via une requête)

```
/**Entity **//
Class Matiere
{
// ...
}
/**Prof**//
Class Etudiant
{ //...
/**
 * @ORM\ManyToMany(targetEntity= "App\Entity\Matiere ")
 */
private $cours ;

/ ...
/**
 * Constructor
 */
public function __construct ()
{
    $this->matieres = new
\Doctrine\Common\Collections\ArrayCollection();
}
}
```

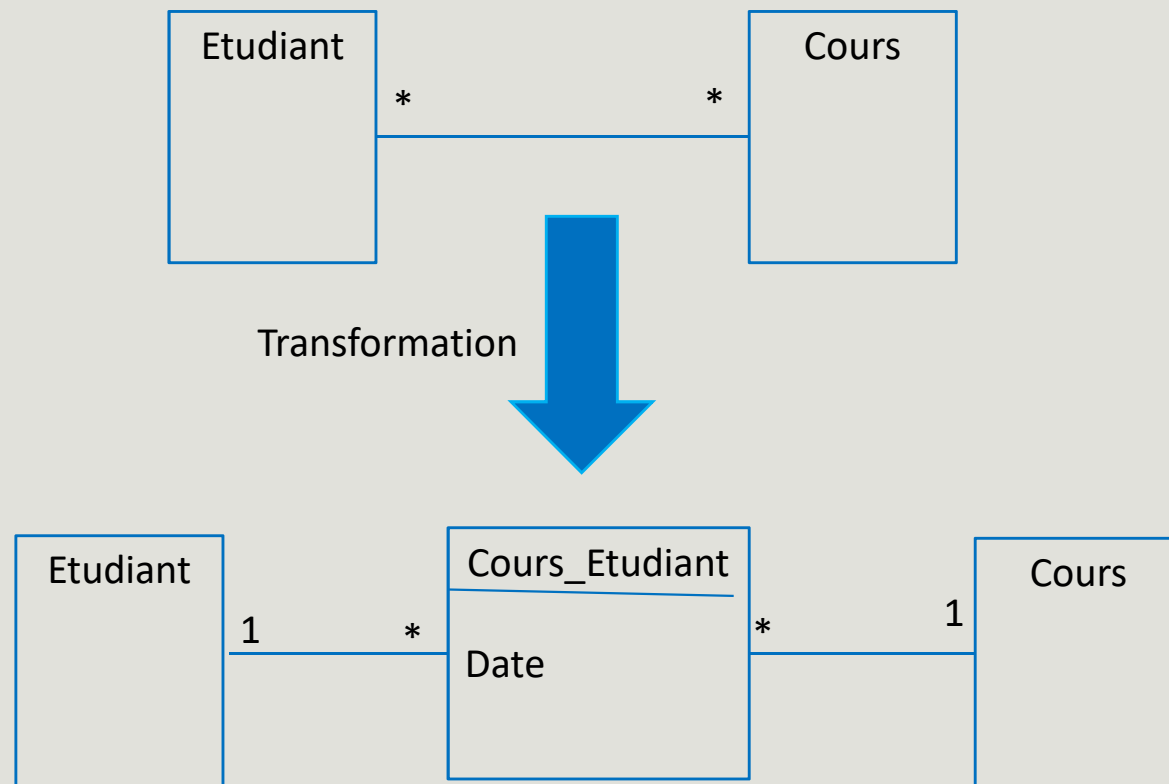
Gestion des relations entre les entités (8)

ManyToMany Bidirectionnelle

- On doit spécifier dans les deux constructeurs que l'attribut mappé est de type ArrayCollection en l'instanciant
- Même chose que la bidirectionnelle OneToMany en ajoutant les mappedBy et inversedBy
- Cette relation peut être traduite à deux relation OneToMany/ManyToOne entre les 3 classes participantes.

Gestion des relations entre les entités (9)

Cas particulier ManyToMany

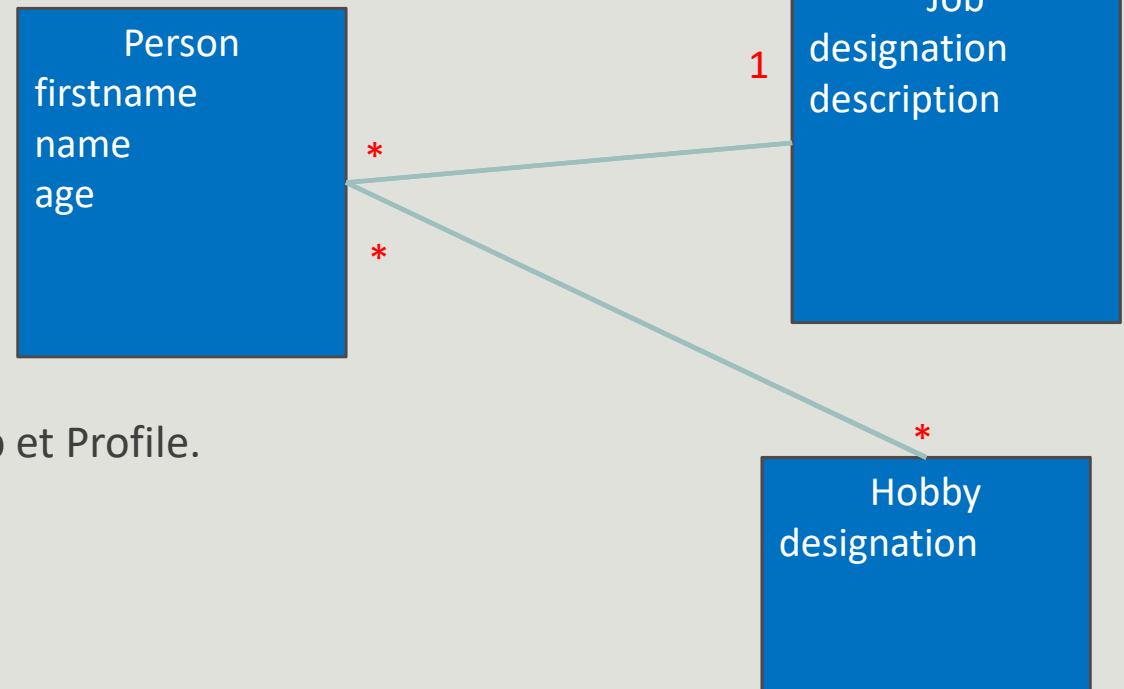


$\eta = \sum \cos \phi_i$ and $I_2(t) = A \cos \phi_i$
 $f_{pe} (Pe) = \frac{(1+\kappa)e^{-\kappa}}{1+e^{-\kappa}}$
 $2.1 \sqrt{(\frac{\sqrt{K(1+K)}}{A_{pe} + \eta})^2 + \kappa^2}$ and $\varphi' = \tan^{-1}$

Personne

Job

Hobby



Créer les fixtures pour les entités Hobby, Job et Profile.

Détour : les Traits

- Servent à externaliser du code redondant dans plusieurs classes différentes.
- Pourquoi les traits et non l'héritage ? Parce que PHP ne supporte pas l'héritage multiple.
- Non instanciable
- Un trait peut contenir des méthodes et des attributs
- Syntaxe :

```
Trait nomTrait{  
    public $x;  
    Function fct1(){  
    }  
    Function fct2(){  
    }  
}
```

Les événements de Doctrine (1)

➤ Appelé aussi les callbacks du cycle de vie, ce sont des méthodes à exécuter par doctrine et dépendant d'événement précis :

- PrePersist
- PostPersist (s'exécute après le \$em->flush() non après \$em->persist())
- PreUpdate
- PostUpdate
- PreRemove
- PostRemove

Afin d'informer doctrine qu'une entité contient des callbacks nous devons utiliser l'annotation

`@ORM\HasLifecycleCallbacks()`

Ceci ne s'applique que lors de l'utilisation des annotations

```
/**
 * Transfer
 *
 * @ORM\Table(name="transfer")
 *
 * @ORM\Entity(repositoryClass="AppBundle\
Repository\TransferRepository")
 * @ORM\HasLifecycleCallbacks()
 */
class Transfer
```

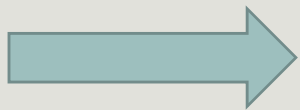
Les événements de Doctrine (2)

- Pour informer Doctrine de l'existence d'un événement on utilise maintenant l'annotation sur l'action à réaliser.

```
/**
 * @PrePersist
 */
public function onPersist() {
    $this->createdAt = new \DateTime('NOW');
}
```

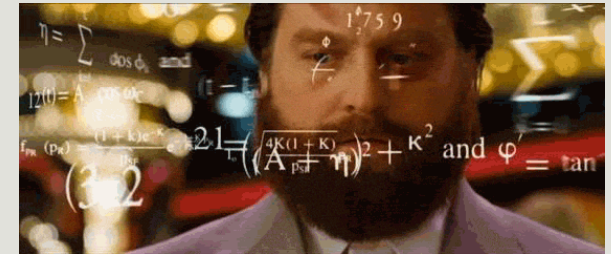

Les traits et Les événements de Doctrine

- Imaginons maintenant que nous voulons « sécuriser plusieurs de nos entités » et d'avoir un peu d'historique. L'idée est d'avoir deux attributs qui sont `createdAt` et `modifiedAt` pour avoir toujours une idée sur la création de notre entité et de sa dernière modification.
- L'idée est de créer pour chacune des entités à suivre des lifecycle callback qui vont mettre à jour ces deux attributs lors de la création (`prePersist`) et la modification (`preUpdate`).
- Est-ce normal de le refaire pour toutes les entités ?
- Si la réponse est non, que faire alors ?



Les traits

Exercice

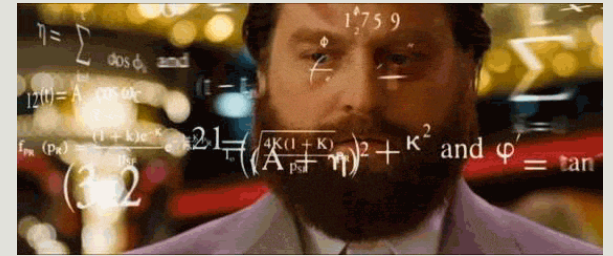


Créer le Trait qui permet la gestion de la date d'ajout et de modification d'une entité.

Associer le avec l'entité formation

Tester le

Exercise



Mettez à jour vos fixtures afin qu'elle permettent de remplir les trois tables (person, hobby et job)

<https://symfony.com/bundles/DoctrineFixturesBundle/current/index.html#loading-the-fixture-files-in-order>

Symfony

Les formulaires

AYMEN SELLAOUTI

Introduction

- Rôle très important dans le web
- Vitrine, interface entre les visiteurs du site web et le contenu du site
- Généralement traité en utilisant du html `<form> ... </form>`
- Symfony et les formulaires : [le composant Form](#)
- Bibliothèque dédiée aux formulaires

Qu'est ce qu'un formulaire Symfony

La philosophie de Symfony pour les formulaires est la suivante :

Vision 1

- Un formulaire est l'image d'un objet existant
- Le formulaire sert à alimenter cet objet.

```
Classe Exemple  
{  
    private $id;  
    private $nom;  
    private $age;  
}
```



Nom
Age

Vision 2

- Un formulaire sert à récupérer des informations indépendantes de n'importe quel objet.

Comment créer un formulaire

Méthodes de création de formulaire

La création du formulaire se fait de 2 façons différentes :

- 1) Dans le contrôleur qui va utiliser le formulaire
- 2) En externalisant la définition dans un fichier

Comment créer un formulaire FormBuilder

- La création d'un formulaire se fait à travers le Constructeur de formulaire FormBuilder
- Exemple :
- `$monPremierFormulaire= $this->createFormBuilder($objetImage)`
- Pour indiquer les champs à ajouter au formulaire on utilise la méthode `add` du `FormBuilder`

Comment créer un formulaire FormBuilder

La méthode `add` contient 3 paramètres :

- 1) le nom du champ dans le formulaire
- 2) le type du champ
- 3) un array qui contient des options spécifiques au type du champ

Exemple :

```
$monPremierFormulaire= $this->createFormBuilder($exemple)  
->add('nom', TextType::class)  
->add('age', IntegerType::class)
```

Comment créer un formulaire

Récupérer le formulaire avec `getForm()`

➤ Pour récupérer le formulaire créé, il faut utiliser la méthode `getForm()`

Exemple :

```
$monPremierFormulaire= $this->createFormBuilder($exemple)  
->add('nom',TextType::class)  
->add('age', IntegerType::class)  
->getForm();
```

Externalisation de la définition des formulaires

AbstractType

Afin de rendre les formulaires réutilisables, Symfony permet l'externalisation des formulaires en des objets.

- **Convention de nommage** : L'objet du formulaire doit être nommé comme suit **NomObjetType**
- Cet objet doit obligatoirement étendre la classe **AbstractType**
- Deux méthodes doivent obligatoirement être implémentées :
 - **buildForm(FormBuilderInterface \$builder, array \$options)** qui est la méthode qui va permettre la création et la définition du formulaire
 - Il y a aussi la méthode **configureOptions** qui permet de définir l'objet associé au formulaire. Cette fonction est obligatoire si vous voulez associer votre form à une classe.

Externalisation de la définition des formulaires

Commande de génération de formulaire

- Maker permet aussi d'automatiquement générer la classe du formulaire

`php bin/console make:form FormNameType`

`symfony console make:form FormNameType`

- Exemple :

`symfony console make:form PersonneType`

Externalisation de la définition des formulaires

Commande de génération de formulaire

```
<?php
namespace Rt4\AsBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

class TacheType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('matache')
            ->add('date')
        ;
    }
}
```

Externalisation de la définition des formulaires

Commande de génération de formulaire

- Maker vous demandera si votre formulaire est associé à une entité ou non. Répondez selon votre besoin.
- La récupération du formulaire au niveau des contrôleurs devient beaucoup plus facile :
- `$form = $this->createForm(TacheType::class, $entity);`
- Le second parameter n'est pas obligatoire

Affichage du formulaire dans TWIG

CreateView

- Afin d'afficher le formulaire crée, il faut transmettre la vue de ce formulaire à la page Twig qui doit l'accueillir.
- La méthode `createView` de l'objet `Form` permet de créer cette vue
- Il ne reste plus qu'à l'envoyer à la page twig en question

➤ Exemple :

```
$form= $this->createForm (ExempleType::class,$exemple) );  
return $this->render('Rt4AsBundle:Default:myform.html.twig',  
                    array('form'=>$form->createView()));
```

Affichage du formulaire dans TWIG form

Deux méthodes permettent d'afficher le formulaire dans Twig :

1) Afficher directement la totalité du formulaire avec la méthode `form`

```
{{ form(nomDuFormulaire) }}
```

2) Afficher les composants du formulaire `séparément un à un` (généralement lorsqu'on veut personnaliser les différents champs)

Customiser vos Form avec Bootstrap

Afin d'intégrer directement **bootstrap** sur vos formulair, il suffit de :

- Spécifier à symfony dans le fichier **twig.yml** que vous voulez du Bootstrap pour vos formes.
- Informer la Twig qui contient vos formulaire qu'elle doit utiliser ce thème la

```
twig:  
  default_path: '%kernel.project_dir%/templates'  
  form_themes: ['bootstrap_5_layout.html.twig']
```

Récupérer les données envoyées

- La gestion de la soumission des formulaires se fait à l'aide de la méthode `handleRequest($request)`
- `HandleRequest` vérifie si la requête est de type POST. Si c'est le cas, elle va mapper les données du formulaire avec l'objet affecté au formulaire en utilisant les setters de cet objet. Si aucun objet n'est mappé, vous pouvez récupérer directement ces données.
- Cette fonction prend en paramètre la requête HTTP de l'utilisateur qui est encapsulé dans Symfony au sein d'un objet de la classe `Request` de **HttpFoundation**.

Récupérer les données envoyées

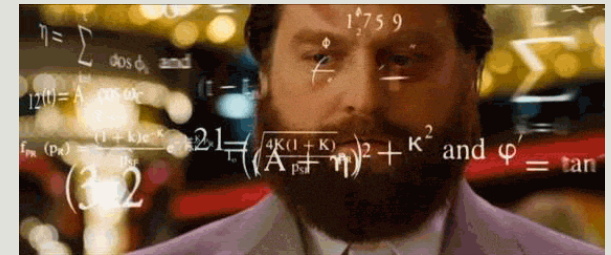
- Vous pouvez récupérer les **données envoyées** via votre formulaire en accédant au champ via la méthode **getData()** de l'objet form.

Exemple : **\$form->getData()** retourne un tableau associatif avec les données envoyées par le formulaire.

- Chaque élément aura comme clé le contenu de l'attribut **name** dans le formulaire.

```
public function showFormAction(Request $request) {  
    $form->handleRequest($request);  
    if ($form->isSubmitted() ) {  
        $data = $form->getData();  
        // ToDo  
    }  
}
```

Exercice



Récupérer les données envoyées à travers le formulaire et afficher le résultat.

Affichage du formulaire dans TWIG

Les composants du formulaire

- `form_start()` affiche la balise d'ouverture du formulaire HTML, soit `<form>`. Il faut passer la variable du formulaire en premier argument, et les paramètres en deuxième argument. L'index `attr` des paramètres, et cela s'appliquera à toutes les fonctions suivantes, représente les attributs à ajouter à la balise générée, ici le `<form>`. Il nous permet d'appliquer une classe CSS au formulaire, ici `form-horizontal`.
- **Exemple** : `{{ form_start(form, {'attr': {'class': 'form-horizontal'}}) }}`
- `form_errors()` affiche les erreurs attachées au champ donné en argument.
- `form_label()` affiche le label HTML du champ donné en argument. Le deuxième argument est le contenu du label.

Affichage du formulaire dans TWIG (3)

Les composants du formulaire (2)

`form_widget()` affiche le champ HTML

Exemple : `{{ form_widget(form.title, {'attr': {'class': 'form-control'}}) }}`

`form_row()` affiche le label, les erreurs et le champ.

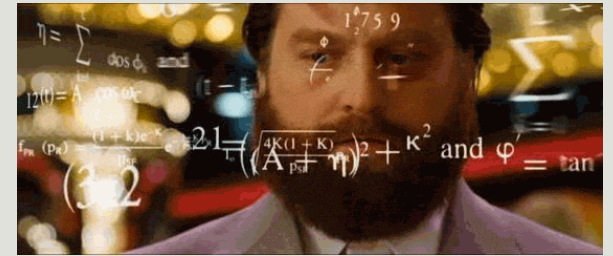
`form_rest()` affiche tous les **champs manquants** du formulaire.

`form_end()` affiche la balise de fermeture du formulaire HTML

Remarque : Certains types de champ ont des options d'affichage supplémentaires qui peuvent être passées au widget. Ces options sont documentées avec chaque type, mais l'option **attr** est commune à tous les types et vous permet de modifier les attributs d'un élément de formulaire.

Exercice

Reprenez le formulaire que vous avez créé et changez-le en décortiquant chaque champs.



Passer une URL à l'objet du formulaire

Ne pouvons pas accéder dans la classe `AbstractType` à la méthode `generateUrl` afin de modifier l'action du formulaire, il faut donc procéder ainsi :

- Utiliser le **troisième paramètre** de la méthode **`createForm`**. C'est un tableau associatif contenant un ensemble d'option. On peut y ajouter deux clé :
- **`action`** : pour ajouter l'url de l'action
- **`method`** : si vous voulez modifier l'attribut `method` qui est par défaut à `post`.

```
$form = $this->createForm(FakeFormType::class, null ,array(  
    'action' => $this->generateUrl('personne.add'),  
    'method' => 'GET'  
));
```


Les propriétés d'un champ dans le formulaire

Le troisième paramètre de la méthode **add** est un tableau d'options pour les attributs du formulaire

Parmi les options communes à la majorité des champs nous citons :

- **label** : pour le label du champ si cette option n'est pas mentionné alors le label sera le nom du champ
- **required** : Permet de dire si le champ est obligatoire ou non (Par défaut l'option required est défini à true)

Les principaux types dans le formulaire

Liste des types

- Les formulaires sont composés d'un ensemble de champs
- Chaque champ possède un nom, un type et des options
- Symfony propose une grande panoplie de types de champ

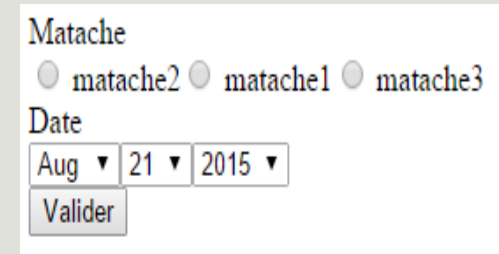
Texte	Choix	Date et temps	Divers	Multiple	Caché
TextType TextareaType EmailType IntegerType MoneyType NumberType PasswordType PercentType SearchType RangeType...	ChoiceType EntityType CountryType LanguageType LocaleType TimezoneType CurrencyType	DateType DatetimeType TimeType BirthdayType	CheckboxType FileType RadioType	CollectionType RepeatedType	HiddenType CsrfType
http://symfony.com/doc/current/forms.html					

Les principaux types dans le formulaire

Le type choice

- Type spécifique aux champs optionnels (select, boutons radio, checkboxes)
- Pour spécifier le **type d'options** qu'on veut avoir il faut utiliser le paramètre **expanded**. S'il est à **false** (valeur par défaut) alors nous aurons **une liste déroulante**. S'il est à **true** alors nous aurons des **boutons radio** ou des **checkbox** qui dépendra du paramètre **multiple**
- Exemple :

<http://symfony.com/doc/current/reference/forms/types/choice.html>



A form snippet showing a choice type with radio buttons. The label 'Matache' is above three radio buttons labeled 'matache2', 'matache1', and 'matache3'. Below this is a date field with three dropdowns showing 'Aug', '21', and '2015', and a 'Valider' button.

Expanded=true



A form snippet showing a choice type with a dropdown menu. The label 'Matache' is above a dropdown menu currently showing 'matache2'. The dropdown is open, showing a list with 'matache2' (highlighted), 'matache1', and 'matache3'. Below this is a date field with three dropdowns showing 'Aug', and a 'Valider' button.

Expanded=false

Les principaux types dans le formulaire

Le type Entity

Champ choice spécial

Les choices (les options) seront chargés à partir des éléments d'une entité Doctrine

```
->add('emploi',EntityType::class, array(  
    'class' => 'Tekup\BdBundle\Entity\Emploi',  
    'choice_label'=>'designation',  
    'expanded'=>false,  
    'multiple'=>true)  
)
```

Balise HTML	expanded	multiple
Liste déroulante	false	false
Liste déroulante (avec attribut <code>multiple</code>)	false	true
Boutons radio	true	false
Cases à cocher	true	true

<http://symfony.com/doc/current/reference/forms/types/entity.html>

Personnaliser le choice label

Afin de personnaliser ce que vous voulez afficher dans vos choix, vous avez deux solutions :

1. Définir la méthode magique **to_string** de votre entité, c'est la méthode appelé par défaut en cas d'absence d'une information sur ce qu'il faut afficher.
2. Affecter à la propriété **choice_label** une **callback function** qui retournera la chaîne à afficher pour chaque enregistrement. Elle prendra en paramètre l'instance de l'entité à traiter.

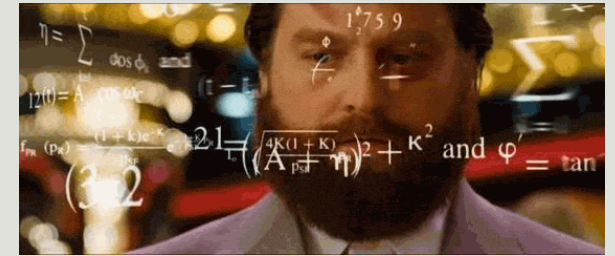
```
->add('formateur', EntityType::class, array(  
    'choice_label' => function(Formateur $formateur) {  
        return (  
            sprintf( '%s-%s', $formateur->getName(),  
                    $formateur->getField()  
            );  
        }  
    )  
);
```

EntityType query_builder

- Afin de customiser la liste de choix de l'utilisateur vous pouvez utilisé la propriété **query_builder**

```
$builder->add('users', EntityType::class, [  
    'class' => User::class,  
    'query_builder' => function (EntityManager $er) {  
        return $er->createQueryBuilder('u')  
            ->orderBy('u.username', 'ASC');  
    },  
    'choice_label' => 'username',  
]);
```

Exercice



- Créer une méthode permettant d'ajouter une personne à travers le formulaire.

Les principaux types dans le formulaire (4)

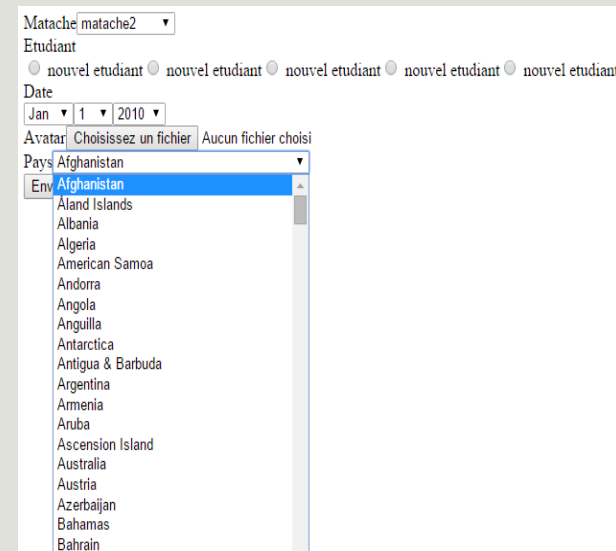
Le type country

Affiche la liste des pays du monde

La langue d’affichage est celle de la locale de votre application (config.yml)

Exemple

```
->add ( 'pays' , CountryType::class)
```



The screenshot shows a web form with the following elements:

- A dropdown menu labeled "Matache" with the value "matache2" selected.
- A section titled "Etudiant" containing five radio buttons, all labeled "nouvel etudiant".
- A "Date" field with a calendar icon, showing "Jan 1 2010".
- An "Avatar" field with a label "Choisissez un fichier" and a button "Aucun fichier choisi".
- A "Pays" field with a dropdown menu showing "Afghanistan" selected. Below the dropdown is a scrollable list of countries starting with "A", including: Aland Islands, Albania, Algeria, American Samoa, Andorra, Angola, Anguilla, Antarctica, Antigua & Barbuda, Argentina, Armenia, Aruba, Ascension Island, Australia, Austria, Azerbaijan, Bahamas, and Bahrain.

<http://symfony.com/doc/current/reference/forms/types/country.html>

Ne pas afficher un champs du formulaire

- Dans certains cas, vous n'avez pas envie d'afficher un champs de votre entité. Prenons l'exemple de l'état. Par défaut et lorsque vous créer une formation, vous voulez qu'elle soit active. Ce n'est pas un choix dépendant du créateur.
- Votre objet **form** contient une méthode **remove** (l'opposé de add) qui permet de supprimer un champs.
- Pensez à ajouter une valeur au champs supprimé ou bien ajouter une **valeur initiale** au niveau de l'entité à cet attribut.

Les principaux types dans le formulaire (4)

Le type file

- Le type **file** permet l'upload de n'importe quel type de fichier.
- Créer un champ de ce type dans votre form et mettez l'option mapped à false.
- Le champ permet de récupérer un **objet** de type **uploadedFile** contenant le **path de l'objet à uploader**
- Afin de récupérer ce champs utiliser votre objet **form** et accéder au **paramètre** ayant le même nom que votre propriété. Ensuite via la méthode **getData** récupérer votre objet. Exemple pour une propriété image : **\$monImage = \$form['image']->getData();**
- Pour pouvoir gérer cet objet il **faut** le copier dans le **répertoire web de votre projet** et de préférence dans un dossier spécifique pour vos images ou vos upload.

Les principaux types dans le formulaire (4)

Le type file

- Attribuer un nom unique à votre fichier pour ne pas avoir de problème lors de l'ajout de fichier ayant le même nom (vous pouvez utiliser la méthode suivante `md5(uniqueid())`;
- Pour récupérer le nom de votre file utiliser `getClientOriginalName()`
- Pour récupérer l'extension vous pouvez utiliser la méthode `guessExtension` de votre objet `file`.
- Pour déplacer votre fichier utiliser la méthode `move($pathsrc,$pathdest)` **de votre objet file.**
- `__DIR__` vous donne le `path` de l'endroit où vous l'utilisez.
- Vous pouvez créer un paramètre dans `services.yml` afin d'y stocker le `path` de votre dossier et le récupérer dans le Controller avec la méthode `getParameter('nom du paramètre')`;
- Remarque : `%kernel.root_dir%` vous permet de récupérer le `path` du dossier `app`

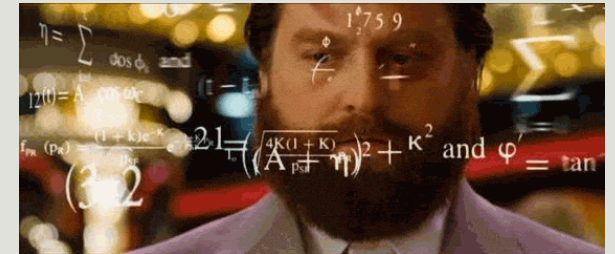
Le type file

```
/** @var UploadedFile $file */
$file = $form->get('file')->getData();
if ($file) {
    $originalFilename = pathinfo($file->getClientOriginalName(), PATHINFO_FILENAME);
    // this is needed to safely include the file name as part of the URL
    $safeFilename = $slugger->slug($originalFilename);
    $newFilename = $safeFilename . '-' . uniqid() . '.' . $file->guessExtension();
    // Move the file to the directory where brochures are stored
    try {
        $file->move(
            $this->getParameter('upload_directory'),
            $newFilename
        );
    } catch (FileNotFoundException $e) {
        // ... handle exception if something happens during file upload
    }
}
```

parameters:

upload_directory: '%kernel.project_dir%/public/uploads'

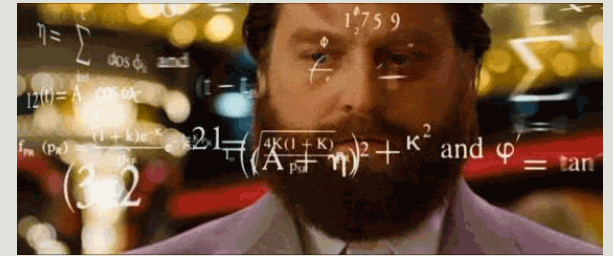
Exercice



- Ajoutez un champs image pour l'entité Personne et mettez en place le mécanisme d'upload de l'image.

Exercice

- Ajouter la fonctionnalité de mise à jour d'une personne.



Les validateurs

Définition

Le validateur est conçu pour valider les objets selon des *contraintes*.

Le validateur de symfony est utilisé pour attribuer des *contraintes* sur les formulaires.

La validation peut être faite de plusieurs façons :

- YAML (dans le fichier `validation.yml` dans le dossier `/Resources/config` du Bundle en question)
- Annotations **sur l'entité de base du formulaire**
- XML
- PHP

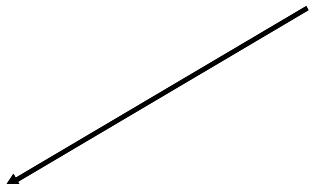
La méthode `isValid()` du FORM déclenche le processus de validation

<http://symfony.com/doc/current/reference/constraints.html>

Exemple Valideur

```
<?php
namespace AppBundle\Entity;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;
/**
 * @ORM\Table(name="personne")
 */
class Personne
{
    /**
     * @var int
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
    /**
     * @Assert\File(mimeTypes = {"application/pdf"})
     * @ORM\Column(name="path", type="string")
     */
    // ...
}
```

Ici nous indiquons à Symfony qu'il ne faut accepter que les fichiers dont le type est pdf



Les validateurs : Les annotations

Afin de pouvoir utiliser les annotations de validation il faut importer la class `Constraints`

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

Syntaxe :

```
@Assert\MaContrainte(option1="valeur1", option2="valeur2", ...)
```

Exemples :

```
@Assert\NotBlank( message = " Ce champ ne doit pas être vide ")
```

```
@Assert\Length(min=4, message="Le login doit contenir au moins {{ limit }} caractères.")
```

```
@Assert\Url()
```

Enlever la validation HTML

Afin d'enlever la validation html ajouter le mot clé novalidate à votre form

```
{{ form_start(form, {'attr': {'novalidate': 'novalidate'}}) }}  
{{ form_widget(form) }}  
{{ form_end(form) }}
```

Les annotations : Les contraintes de base

Contrainte	Rôle	Options
NotBlank Blank	La contrainte NotBlank vérifie que la valeur soumise n'est ni une chaîne de caractères vide, ni NULL. La contrainte Blank fait l'inverse.	-
True False	La contrainte True vérifie que la valeur vaut true, 1 ou "1". La contrainte False vérifie que la valeur vaut false, 0 ou "0".	-
NotNull Null	La contrainte NotNull vérifie que la valeur est strictement différente de null.	-
Type	La contrainte Type vérifie que la valeur est bien du type donné en argument.	type (option par défaut) : le type duquel doit être la valeur, parmi array, bool,int, object

Les annotations : Nombre, date

Contrainte	Rôle	Options
Range	La contrainte Range vérifie que la valeur ne dépasse pas X, ou qu'elle dépasse Y.	min : nbre de car minimum max : nbre de car maximum minMessage : msg erreur nbre de car min maxMessage : msg erreur nbre de car max invalidMessage : msg erreur si non nombre
Date	vérifie que la valeur est un objet de type Datetime, ou une chaîne de type YYYY-MM-DD.	-
Time	vérifie qque c'est un objet de type Datetime, ou une chaîne type HH:MM:SS.	-
DateTime	vérifie que c'est un objet de typeDatetime, ou une chaîne de caractères du type YYYY-MM-DD HH:MM:SS.	

Les annotations : File

Contrainte	Rôle	Options
File	La contrainte File vérifie que la valeur est un fichier valide, c'est-à-dire soit une chaîne de caractères qui pointe vers un fichier existant, soit une instance de la classe File (ce qui inclut UploadedFile).	maxSize : la taille maximale du fichier. Exemple : 1M ou 1k. mimeTypes : mimeType(s) que le fichier doit avoir.
Image	La contrainte Image vérifie que la valeur est valide selon la contrainte précédente File (dont elle hérite les options), sauf que les mimeTypes acceptés sont automatiquement définis comme ceux de fichiers images. Il est également possible de mettre des contraintes sur la hauteur max ou la largeur max de l'image.	maxSize : la taille maximale du fichier. Exemple : 1M ou 1k. minWidth /maxWidth : la largeur minimale et maximale que doit respecter l'image. minHeight /maxHeight : la hauteur minimale et maximale que doit respecter l'image.

Validation Exemple

```
/**
 * @var string
 * @Assert\Length(min="3",max="10",maxMessage="Trop c'est trop")
 * @ORM\Column(name="nom", type="string", length=50)
 */
private $nom;

/**
 * @var string
 * @Assert\File(mimeTypes = {"application/pdf"},mimeTypesMessage="Le
fichier doit être du format PDF")
 * @ORM\Column(name="path", type="string")
 */
private $path;
```

Nom	<div>ERROR Trop c'est trop</div> <div>plus que 10 lettres</div>
Age	<div>21</div>
Path	<div>ERROR Le fichier doit être du format PDF</div> <div>Choisir un fichier 007.jpg</div> <div>Enregistrer</div>

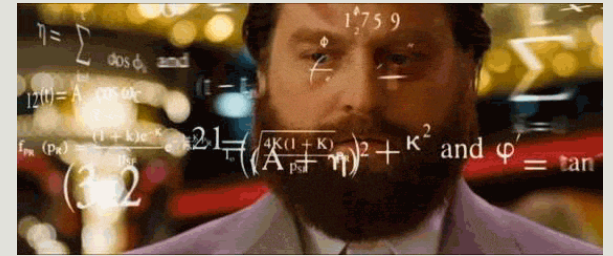
Valider des champs non mappés

Lorsque le champs que vous voulez valider est non mappé et que vous souhaitez le valider, il faut ajouter un [paramètre constraints](#) dans le [tableau d'option de votre méthode add](#).

```
->add('imageFile', FileType::class, array(  
    'mapped'=> false,  
    'constraints'=> array(  
        new Image(),  
    ),  
));
```

Exercice

- Ajouter les validateurs nécessaires à votre formulaire.



Symfony Les Services

AYMEN SELLAOUTI

Définition d'un Service

- Lorsque vous travaillez avec Symfony, vous aurez besoin d'une même fonctionnalité à plusieurs endroits différents. Vous aurez par exemple besoin d'accéder à votre base de données dans plusieurs contrôleurs.
- Un Service est **une classe simple** qui fournit un « Service ». Vous avez déjà utilisé plusieurs services tel que Doctrine ou TWIG. Elle doit être accessible partout dans votre code.
- Dans Symfony tous les services sont gérés par le **conteneur de service « Symfony container »**.

Service = Classe PHP accessible partout + une configuration

https://symfony.com/doc/current/service_container.html

Responsabilité unique (Single Responsibility)

- L'un des principes de base sur lequel se base la notion de Service est le principe de Responsabilité unique. Ce principe fait parti des principes **SOLID** de l'orienté objet. Pour le premier principe de « **Single Responsibility** » il implique que votre code au sein d'une classe **ne doit avoir qu'une seule responsabilité**.
- Prenons l'exemple du service mailer qui se charge de l'envoi de mail, donc un seul type de tâche à effectuer.
- Dans le cas ou votre service se charge de 2 tâches différentes, pensez à diviser votre classe en deux services différents.
- L'utilisation des services et du principe de **Single Responsibility** permet d'avoir un **code réutilisable** et facilement **maintenable**.

Le conteneur de services

- Un service est une classe associée à une configuration. Si on utilise plusieurs services et qu'on doit tout gérer, l'utilisation des services deviendrait un peu pénible. C'est pour cela que Symfony nous fournit un conteneur de service.
- Un **conteneur de service** est un **objet** qui a pour rôle de **gérer l'ensemble des services** de votre application. C'est lui qui vous permettra d'accéder à vos services. **Si vous avez besoin d'utiliser un service vous devez passer par le conteneur.**

Le conteneur de services

- Cependant le rôle du conteneur n'est pas simplement de fournir les services mais aussi de les préparer. En effet, c'est lui qui va préparer l'objet en l'instanciant et en instanciant toutes les classes dont dépend votre service.
- Tous les services (objets) ne sont pas instanciés à chaque requête. En effet, le conteneur est paresseux il utilise le lazy loading afin d'assurer une plus grande vitesse d'exécution : il n'instancie pas un service avant que vous le demandiez. Par exemple, si vous n'utilisez jamais le service de mailing lors d'une requête, le conteneur ne l'instanciera jamais.

```
Service Container  
new Service1(p1, p2);
```

```
Class c1 {  
  service1= new Service1(p1, p2);  
}
```

```
Class c2 {  
  service1 = new Service1(p1, p2);  
  service3 = new Service3(p1,p2,p3);  
}
```

```
Class c3 {  
  service2= new Service2(p1);  
  service3 = new Service3(p1,p2,p3);  
}
```

Workflow d'un conteneur de service

Lors de la demande d'un service S1, le conteneur suit le workflow suivant :

Vérifie si la classe S1 est déjà instancié.

1. Si oui la fournir
2. Sinon
 1. Vérifier si le service dépend d'autres classes.
 1. Si oui, les instancier ou les récupérer si elles sont instanciées
 2. Instancier le service

Injection de dépendances

- L'injection des services dans les différentes classes de votre projet se fait à travers **l'injection de dépendances**. En effet, vous êtes entrain de travailler avec un principe très important et qui permet de minimiser les dépendances entre les classes. Ici l'instanciation n'est plus votre soucis. C'est le conteneur de service qui s'occupe de tout.
- L'injection de dépendance est un patron de conception qui a vu le jour afin de palier à un problème très récurrent : Avoir dans son code des classes qui **dépendent les unes des autres**. L'injection de dépendance palie à ce problème en découplant les classes.
- Avec le conteneur de service de Symfony on peut injecter des services ou des paramètres.

Accéder à un service

Plusieurs services sont déjà accessibles dès l'installation de Symfony. Comme nous l'avons déjà mentionné, vous avez déjà utilisé des services comme TWIG et Doctrine.

Deux façon permettent d'accéder à un service :

1- En passant par le **conteneur de service** et sa méthode **get**. Cette méthode prend en paramètre l'identifiant unique du service. **Dans le guide des bonnes pratiques de Symfony, cette méthode est à éviter au maximum.**

Exemple : pour récupérer doctrine on utilise la syntaxe suivante :

```
$doctrine = $this->container->get ( 'doctrine' ) ;
```

Accéder à un service

Lorsque nous avons utilisé doctrine, nous avons utilisé un helper et la méthode `getDoctrine`, cette méthode n'est rien qu'un raccourci qui appelle le service de doctrine.

2- En utilisant la nouvelle façon utilisable à partir de la version 3.3.

Exemple :

```
use Psr\Log\LoggerInterface;

public function listAction(LoggerInterface $logger)
{
    $logger->info('J utilise le service de logging');
}
```

Lister les services disponibles

Afin de lister les services disponibles dans votre projet, utiliser la commande suivante

`php bin/console debug:container`

Les services les plus connues sont :

Service ID	Class name
doctrine	Doctrine\Bundle\DoctrineBundle\Registry
logger	Symfony\Bridge\Monolog\Logger
router	Symfony\Bundle\FrameworkBundle\Routing\Router
session	Symfony\Component\HttpFoundation\Session\Session
twig	Twig_Environment
validator	Symfony\Component\Validator\Validator\ValidatorInterface

Création d'un premier service

Créer un dossier service.

Créer une classe PremierService

Introduisez y une méthode getRandomString(\$nb) qui prend en paramètre le nombre de caractère et retourne une chaine de taille ce nombre la. Ensuite, appelez ce service dans une de vos fonctions et afficher la valeur de retour.

```
public function
getRandomString($nb) {
    $char =
    'abcdefghijklmnopqrstuvwxyz01234
    56789';
    $chaine = str_shuffle($char);
    return substr($chaine, 0, $nb);
}
```

```
use AppBundle\Service\PremierService;

public function
testAction(PremierService $ps) {
    dump($ps->getRandomString($nb));
    //...
}
```

Configuration des services

Le premier service que vous venez de créer a fonctionné automatiquement sans aucune intervention de votre part et sans aucune configuration. Ceci est le cas à partir de la version 3.3 de Symfony qui a permis d'avoir le principe [d'autowiring](#).

En accédant au fichier [service.yaml](#) vous allez trouvé cette configuration générique avec une explication de chaque ligne de code. Le fait d'avoir un [autowire à true](#) a permis à Symfony d'utiliser le [Type-Hint](#) que vous avez déjà vu et qui a chargé automatiquement votre service.

```
services:
    # default configuration for services in *this* file
    _defaults:
        autowire: true           # Automatically injects dependencies in your services.
        autoconfigure: true    # Automatically registers your services as commands,
# event subscribers, etc.
```

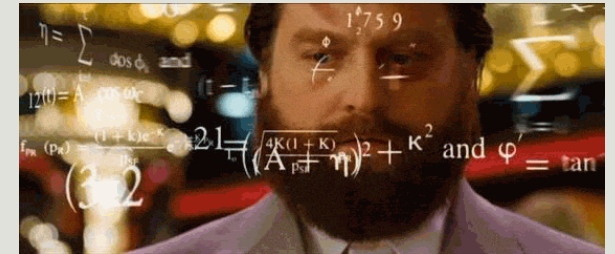
[php bin/console debug:autowiring](#) : permet d'afficher les services autowiré

Configurer un groupe de service

La seconde partie de la configuration, comme l'indiquent les commentaires, permet de définir un **ensemble de services** en une seule ligne. Ici nous indiquons à Symfony que **toutes les classes du dossier src** peuvent être utilisées comme un **service**. L'attribut **exclude** permet de spécifier les classes à exclure de cette liste. Ici c'est les classes des dossiers Entity, Repository et Tests qui le sont.

```
# makes classes in src/ available to be used as services  
# this creates a service per class whose id is the fully-qualified class name  
App\:  
    resource: ' ../src/*'  
    exclude: ' ../src/{DependencyInjection,Entity,Migrations,Tests,Kernel.php}'
```

Exercice



- Le logger est un service très important. Il permet de logger des informations que vous pouvez récupérer en mode dev mais surtout en mode prod ou votre débbugger n'est pas fonctionnel.
- Reprenez l'action de modification et d'ajout d'une personne et logger les informations sur ces actions.

Injection manuelle des arguments

Lorsque vous voulez ajouter des variables à vos services, l'autowiring ne marche plus. Dans ce cas, vous devez injecter ces paramètres manuellement.

Au niveau de votre fichier service.yml, vous devez configurer votre service et définir vos arguments.

Si vous avez besoin d'injecter la variable uniquement au niveau d'un ou plusieurs services particuliers, vous devez le faire de la manière suivante :

```
App\Controller\PersonneController:  
  arguments:  
    - $defaultImagePath: '%kernel.project_dir%/public/images/default.png'
```

Si vous avez besoin d'injecter la variable partout, vous devez le faire de la manière suivante :

```
_defaults:  
  autowire: true  
  autoconfigure: true  
  bind:  
    $defaultImagePath: '%kernel.project_dir%/public/images/default.png'
```


Ajouter des paramètres

```
parameters:
  locale: 'en'
  imagePath: 'images/default.png'

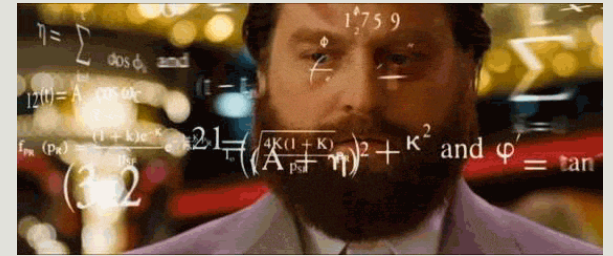
services:
  _defaults:
    autowire: true
    autoconfigure: true
    bind:
      $defaultImagePath: '%imagePath%'
```

Récupérer les paramètres

- Pour récupérer les paramètres il faut faire la même chose que pour les services et les injecter au niveau du constructeur

```
private $personneDirectory;  
public function __construct($personneDirectory)  
{  
    $this->personneDirectory = $personneDirectory;  
}
```

Exercise



- L'upload d'image est une fonctionnalité qui se répète souvent. D'un autre côté ceci est un traitement métier qui ne doit pas figurer dans vos contrôleurs. N'oubliez pas «thin controller».
- Créer un service qui permet de gérer l'upload et utiliser le.

Injecter un service ou des paramètres de configuration dans un autre service

Afin d'injecter des services dans un autre service nous devons passer par le constructeur du service cible et du type Hinting. Le fait d'avoir l'option d'autowiring qui cible le service en question permet de gérer ça.

```
class MaClasse
{
    private $logger;
    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }
    public function LoggerMessage()
    {
        $this->logger->info('J'utilise un service injecté grâce à l'autowiring et le
Type Hinting appelé au niveau de mon constructeur!');
    }
}
```

Pour les paramètres, le principe reste le même et les paramètres sont injectés manuellement.

Envoi d'email

- Symfony utilise le bundle SwiftMailer pour l'envoi d'email.
- Si vous n'avez pas le bundle installer le : `composer require symfony/swiftmailer-bundle`
- La configuration se fait au niveau de votre fichier `.env` dans la variable `MAILER_URL`. Sachant que vous allez utiliser des informations critiques telles le mot de passe du compte email à utiliser, utiliser le fichier `.env.local`.
- Si vous utiliser Gmail voici la configuration requise :
- `MAILER_URL=google://username:password@localhost`

Envoi d'email

Symfony mailer

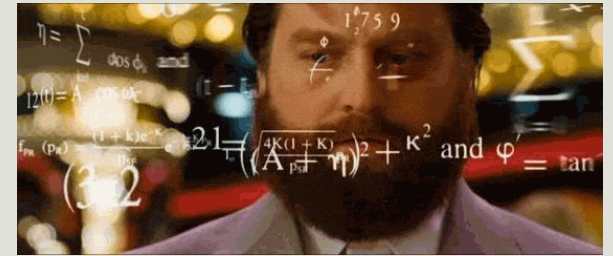
- SwiftMailer sera dépricaded à partir de la verion 6 de Symfony, l'alternative est symfony mailer
<https://symfony.com/doc/current/mailer.html>
- Si vous n'avez pas le bundle installer le : `composer require symfony/mailer`
- Installer ensuite le mailer que vous voulez utiliser : `composer require symfony/google-mailer`
- La configuration se fait au niveau de votre fichier `.env` dans la variable `MAILER_DSN`. Sachant que vous allez utiliser des informations critiques telques le mot de passe du compte email à utiliser, utiliser le fichier `.env.local`.
- Si vous utiliser Gmail voici la configuration requise :
- `MAILER_DSN=gmail://email:password@localhost?verify_peer=0`
- Il faut activer le two factor authentication et générer un mot de passe pour application

Envoi d'un email

```
$message = (new \Swift_Message('Hello Email'))
    ->setFrom('aymn.noreply@gmail.com')
    ->setTo($to)
    ->setBody(
        $template,
        $contentType
    );
if ($attachement) {
    $attachementObject = new \Swift_Attachment($attachement,
        'attachement.pdf',
        'application/pdf' );
    $message->attach($attachementObject);
}

$this->mailer->send($message);
```

Exercice



- Créer un service permettant la manipulation des emails.
- Essayer de rendre votre service générique.

Symfony Sécurité

AYMEN SELLAOUTI

Introduction

- Un site est généralement décomposé en deux parties :
 - Partie public : accessible à tous le monde
 - Partie privée : accessible à des utilisateurs particuliers.
 - Au sein même de la partie privée, certaines ressources sont spécifiques à des rôles ou des utilisateurs particuliers.

Nous identifions donc deux niveaux de sécurité :

Authentification

Autorisation

Authentification

- Processus permettant d'authentifier un utilisateur.
- Deux réponses possibles
 - Non authentifié : Anonyme.
 - Authentifié : membre

Security Bundle

- Le Bundle qui gère la sécurité dans Symfony s'appelle SecurityBundle.
- Si vous ne l'avez pas dans votre application, installer le via la commande

composer require security

Fichier de configuration security.yml

```
security:
  # https://symfony.com/doc/current/security/authenticator_manager.html
  enable_authenticator_manager: true
  # https://symfony.com/doc/current/security.html#c-hashing-passwords
  password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
  # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
  providers:
    users_in_memory: { memory: null }
  firewalls:
    dev:
      pattern: ^/(_(profiler|wdt)|css|images|js)/
      security: false
    main:
      lazy: true
      provider: users_in_memory
      # switch_user: true
  # Easy way to control access for large sections of your site
  # Note: Only the *first* access control that matches will be used
  access_control:
    # - { path: ^/admin, roles: ROLE_ADMIN }
```

Le Firewall qui gère la configuration de l'authentification de vos utilisateurs

Cette partie assure que le débogueur de Symfony ne soit pas bloqué

La classe user

- L'ensemble du système de sécurité est basé sur la classe **User** qui représente l'utilisateur de votre application.
- Afin de créer la classe **User**, utiliser la commande :
symfony console make:user
- Si vous n'avez pas le MakerBundle, installer le.
- Cette outils vous posera un ensemble de questions, selon votre besoin répondez y et il fera tout le reste.

UserProvider

Le User Provider est un ensemble de classe associé au bundle Security de Symfony et qui ont deux rôles

- **Récupérer le user de la session.** En effet, pour chaque requête, Symfony charge l'objet user de la session. Il vérifie aussi que le user n'a pas changé au niveau de la BD.
- Charge l'utilisateur pour réaliser certaines fonctionnalités comme la fonctionnalité ***se souvenir de moi.***

UserProvider

- Afin de définir le **userProvider** que vous voulez utiliser passer par le fichier de configuration **security.yaml**

```
providers:
  users:
    entity:
      # the class of the entity that represents users
      class: 'App\Entity\User'
      # the property to query by - e.g. username, email, etc
      property: email
```


$\eta = \sum \cos \phi_i$ and $I_2(t) = A \cos \phi_i$
 $f_{\text{pre}}(p_{\text{re}}) = \frac{(1+\kappa)e^{-\kappa}}{1 + \frac{1}{A^2} \left(\left(\frac{4K(1+\kappa)}{p_{\text{re}} + \eta} \right)^2 + \kappa^2 \right)}$ and $\varphi' = \tan^{-1} \frac{1}{\sqrt{5}}$

Ajouter la classe user.

```
graph LR; P[Personne] -- "1" --> J[Job]; P -- "*" --> H[Hobby];
```

The diagram illustrates a database schema with three tables: **Personne**, **Job**, and **Hobby**.

- Personne** table attributes: `firstname`, `name`, `age`.
- Job** table attributes: `designation`, `description`.
- Hobby** table attribute: `designation`.

Relationships are shown as lines connecting the tables:

- A line connects **Personne** to **Job** with a **1** at the **Personne** end and a ***** at the **Job** end, indicating a one-to-many relationship.
- A line connects **Personne** to **Hobby** with a ***** at the **Personne** end and a ***** at the **Hobby** end, indicating a many-to-many relationship.

Encoder le mot de passe

- Vous n'avez pas toujours besoin de mot de passe
- En cas de besoin, vous pouvez configurer la manière avec lequel votre mot de passe doit être géré dans le fichier security.yml.

```
# config/packages/security.yml
security:
  # ...
  password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    # use your user class name here
    App\Entity\User:
      # Use native password hasher, which auto-selects the best
      # possible hashing algorithm (starting from Symfony 5.3 this is
      # "bcrypt")
      algorithm: auto
```

```
security:
  encoders:
    App\Entity\User:
      algorithm: auto
```

Avant Symfony 5,3

A partir de Symfony 5,3

Encoder le mot de passe

- Le service responsable de l'encodage des mots de passe est le service `UserProviderEncoder` (avant `Symfony 5.3`) ou `UserPasswordHasher` à partir de `Symfony 5.3`.
- Afin de l'utiliser, et comme tous les services de `Symfony`, il suffit de l'injecter.

```
private $userPasswordHasher;  
public function __construct( UserPasswordHasherInterface $userPasswordHasher)  
{  
    $this->userPasswordHasher = $userPasswordHasher;  
}
```

```
public function __construct( private UserPasswordHasherInterface $passwordEncoder)  
{  
}
```

$\eta = \sum \cos \phi_i$ and $I_2(t) = \frac{1}{A} \cos \phi_i$
 $f_{\text{res}}(\text{Pr}) = \frac{(1+\kappa)e^{-\kappa}}{1+2\kappa}$
 $2\frac{1}{\Gamma}((\frac{\sqrt{\kappa(1+\kappa)}}{A_{\text{Pr}} + \eta})^2 + \kappa^2)$ and $\varphi' = \tan^{-1} \frac{1.759}{\phi}$

- <https://symfony.com/doc/master/bundles/DoctrineFixturesBundle/index.html>

Authentication et Firewall

- Le système d'authentification de Symfony est configuré au niveau de la partie **firewalls** de votre fichier security.yaml.
- Cette section va définir comment vos utilisateurs seront authentifié, e.g. API Token, Formulaire d'authentification.

```
firewalls:  
  dev:  
    pattern: ^/(_(profiler|wdt)|css|images|js)/  
    security: false  
  main:  
    anonymous: true
```

Authentification et Firewall

- Comme décrite dans la documentation, **l'authentification dans Symfony** ressemble à de « **la magie** ».
- En effet, au lieu d'aller construire une route et un contrôleur afin d'effectuer le traitement, vous devez simplement activer un « **authentication provider** ».
- « **L'authentication provider** » est du code qui s'exécute **automatiquement avant chaque appel d'un contrôleur**.
- Symfony possède un ensemble d' « authentication provider » prêt à l'emploi. Vous trouverez leur description dans la documentation : https://symfony.com/doc/current/security/auth_providers.html
- Dans la documentation, il est conseillé de passer par les « Guard Authenticator » qui permettent un contrôle total sur toutes les parties de l'authentification.

Les Guard Authenticator

- Un « Guard authenticator » est une classe qui vous permet un control complet sur le processus d'authentification.
- Cette classe devra ou implémenter l'interface [AuthenticatorInterface](#) ou étendre la classe abstraite associée au besoin, e.g. [AbstractFormLoginAuthenticator](#) en cas de formulaire d'authentification ou [AbstractGuardAuthenticator](#) en cas d'api
- La commande [make:auth](#) permet de générer automatiquement cette classe.
- Une fois lancée, cette commande vous demande si vous voulez créer un « [empty authenticator](#) » ou un « [login form authenticator](#) ».

Guard Authenticator

(Symfony <5.3)

Chaque guard devra implémenter les méthodes suivantes :

➤ **supports(Request \$request)**

Cette méthode est la première à être appelée. Elle sera appelée à chaque requête et votre travail consiste à décider si l'authentificateur doit être utilisé pour cette requête (return true) ou s'il doit être ignoré (return false).

➤ **getCredentials(Request \$request)**

Elle sera appelée à chaque requête et votre travail consiste à lire le Token (ou quelle que soit votre information "d'authentification") à partir de la demande et de les renvoyer. Ces informations d'identification sont ensuite transmises en tant que premier argument de getUser().

➤ **getUser(\$credentials, UserProviderInterface \$userProvider)**

\$credentials représente la valeur retournée par getCredentials(). Votre rôle consiste donc à retourner un objet qui implémente la UserInterface qui sera donc un objet de votre classe User. Une fois retourné, la méthode checkCredentials() sera appelée. Si vous retourner null (ou throw an [AuthenticationException](#)) l'authentification sera avorté.

Guard Authenticator

(Symfony <5.3)

➤ **checkCredentials(\$credentials, UserInterface \$user)**

Si `getUser()` retourne un objet `User`, cette méthode est appelée. Votre travail est de vérifier si les credentials sont correct. Pour un login form, C'est l'endroit où vous allez vérifier si le mot de passe est correct. Pour passer l'authentification vous devez retourner `true`. Si vous retourner autre chose (ou throw an [AuthenticationException](#)), l'authentification va échouer.

➤ **onAuthenticationSuccess(Request \$request, TokenInterface \$token, \$providerKey)**

Appelé après une authentification réussie. Votre travail est de retourner un objet [Response](#) qui sera envoyé au client ou `null` pour continuer la requête.

➤ **onAuthenticationFailure(Request \$request, AuthenticationException \$exception)**

Appelé si une authentification échoue. Votre travail est de retourner un objet [Response](#) object qui sera envoyé au client. L'exception vous informera sur la cause de l'échec de l'authentification.

Guard Authenticator

Symfony 5.3

- A partir de la version 5.3, vous devez implémenter uniquement les méthodes **authenticate** et la méthode **onAuthenticationSuccess**.
- Il y a aussi des méthodes optionnelles que vous pouvez surcharger :
 - **supports**
 - **onAuthenticationFailure**
 - **start**
- Symfony utilise à partir de la version 5,3 **un nouveau Authenticator based Security**
- **Pour la gestion des utilisateurs elle utilise Passport**

Guard Authenticator

Symfony 5.3

```
public function authenticate(Request $request): PassportInterface
{
    $username = $request->request->get('username', '');
    $request->getSession()->set(Security::LAST_USERNAME, $username);
    return new Passport(
        new UserBadge($username),
        new PasswordCredentials($request->request->get('password', '')),
        [
            new CsrfTokenBadge('authenticate', $request->get('_csrf_token')),
        ]
    );
}
```

```
public function onAuthenticationSuccess(Request $request, TokenInterface
$token, string $firewallName): ?Response
{
    if ($targetPath = $this->getTargetPath($request->getSession(),
$firewallName)) {
        return new RedirectResponse($targetPath);
    }
    throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
}
```

Activer le guard (Symfony 5.3)

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    appLogin:
        pattern: ^/login
        custom_authenticator: App\Security\LoginFormAuthenticator
    logout:
        path: app_logout
        # where to redirect after logout
        # target: app_any_route
```

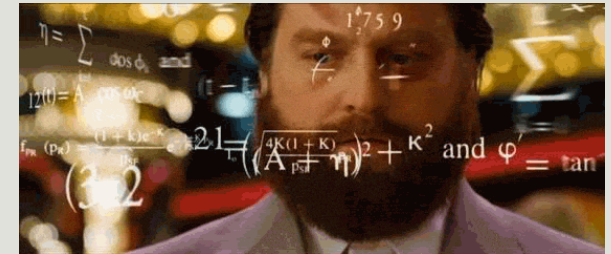
Se déconnecter

- Afin de se **déconnecter**, il suffit d'ajouter la clé **logout** dans votre **firewalls configuration dans security.yaml**.
- Ajouter ensuite une **méthode vide logout dans votre securityController avec la route associé à votre méthode logout**.
- Vous pouvez débiter les autres options de logout avec la commande
symfony console debug:config security

```
/**  
 * @Route("/logout", name="logout")  
 */  
public function logout() {  
  
}
```

```
firewalls:  
    main:  
        logout:  
            path: logout
```

Exercice



- Créer un LoginForm en utilisant la commande
`symfony console make:auth`.
- Terminer les étapes définies par la commande à la fin de son exécution.

Récupérer le user dans le contrôleur

Afin de récupérer le user dans un contrôleur, il suffit d'utiliser la méthode helper **getUser**.

Utiliser ensuite sa méthode

```
public function list()  
{  
    $user = $this->getUser();  
}
```

Récupérer le user dans le service

Afin de récupérer le user dans un service, il suffit d'injecter le Security Service.

Utiliser ensuite sa méthode `getUser`

```
use Symfony\Component\Security\Core\Security;
class HelperService
{
    private $security;
    public function __construct(Security $security)
    {
        $this->security = $security;
    }
    public function sendMoney() {
        $user = $this->security->getUser()
    }
}
```


Remember Me

- Afin d'activer la fonctionnalité 'se souvenir de moi' ajouter la configuration suivante dans le fichier `security.yml` :

```
main:
    anonymous: true
    remember_me:
        secret: '%kernel.secret%'
```

- Décommenter le code affichant **le bouton remember** me dans la Twig `login.html.twig`

```
<div class="checkbox mb-3">
    <label>
        <input type="checkbox" name="_remember_me"> Remember me
    </label>
</div>
```

- Ajouter le **RememberMeBadge** dans le tableau qui est le troisième paramètre passé à **Passport**

```
return new Passport(
    new UserBadge($email),
    new PasswordCredentials($password),
    [
        new RememberMeBadge(),
    ]
);
```

Remember Me

Activation automatique

- Afin d'activer automatiquement cette option, ajouter la propriété `always_remember_me` à `true` sous la clé `remember_me` à `true` dans [security.yml](#) :

```
main:
    anonymous: true
    remember_me:
        always_remember_me: true
```

- Vous pouvez aussi le faire via le badge lui-même

```
return new Passport(
    new UserBadge($email),
    new PasswordCredentials($password),
    [
        (new RememberMeBadge())->enable(),
    ]
);
```

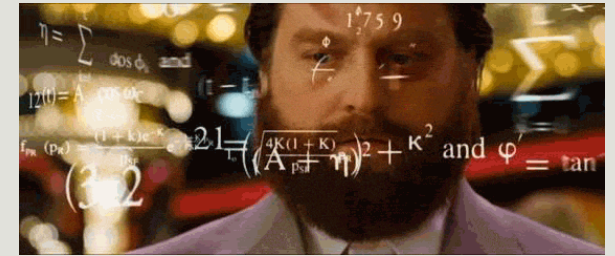
Remember Me

Sécuriser RememberMe

- Si une personne récupère le rememberMe token, il sera connecté jusqu'à la péremption du token
- Afin d'éviter ça, utiliser l'option `signature_properties` qui fera en sorte de signer le token avec le password de l'utilisateur. Dans ce cas, si le user change son mot de passe, le token ne sera plus valide.

```
main:
  anonymous: true
  remember_me:
    always_remember_me: true
    signature_properties: [password]
```

Exercise



- Fait en sorte que le lien login de votre template vous envoi vers la page de login.
- Ajouter un lien pour le logout.

Register

- Afin de permettre l'ajout ou l'enregistrement de vos utilisateurs, vous pouvez utiliser la commande :

`symfony console make:registration-form`

- Cette fonctionnalité n'a rien de particulier, elle permet tout simplement d'ajouter un utilisateur dans votre base de données.
- Vous pouvez personnaliser le contrôleur généré comme vous le voulez.

Authentification manuelle d'un utilisateur

- Une fois l'utilisateur inscrit, vous pourrez l'authentifier d'une façon manuelle en injectant le service **UserAuthenticatorInterface**.
- Utiliser sa méthode authenticateUser qui prend en paramètre le user, votre authenticator et l'objet request

```
public function register(Request $request, UserPasswordHasherInterface
$userPasswordHasherInterface, LoginFormAuthenticator $authenticator,
UserAuthenticatorInterface $userAuthenticator): Response
{
    //...
    if ($form->isSubmitted() && $form->isValid()) {
        //...
        // Authenticate user
        // retourne un Objet Response, celui généré par la méthode onAuthenticationSuccess
        return $userAuthenticator->authenticateUser($user, $authenticator, $request);
    }
}
```

Autorisation

- Processus permettant d'autoriser un utilisateur à accéder à une ressource selon son rôle.

Le processus d'authentification suit deux étapes.

- 1- Lors de l'authentification l'utilisateur est associé à un ensemble de rôles.
- 2- Lors de l'accès à une ressource, on vérifie si l'utilisateur a le rôle nécessaire pour y accéder.

Les Rôles

- Chaque utilisateur connecté a au moins un rôle : le `ROLE_USER`
- Tous les rôles commencent par `ROLE_`

```
/**
 * @see UserInterface
 */
public function getRoles(): array
{
    $roles = $this->roles;
    // guarantee every user at least has ROLE_USER
    $roles[] = 'ROLE_USER';

    return array_unique($roles);
}
```


Définir les droits d'accès

Les droits d'accès sont définis de deux façons :

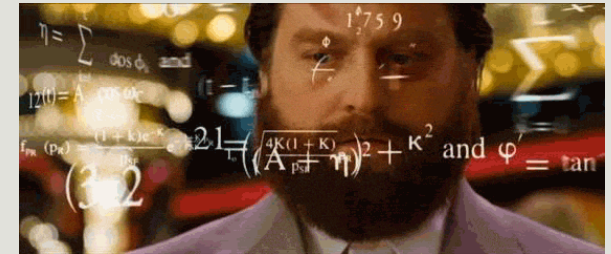
- 1- Dans le fichier security.yaml
- 2- Directement dans la ressource

Sécuriser les patrons d'url

- La méthode la plus basique pour sécuriser une partie de votre application est de sécuriser un patron d'url complet dans votre fichier security.yaml.
- Ceci se fait sous la clé access_control. Chaque entrée est un objet avec comme clé :
 - -**path** : le pattern à protéger
 - -**roles** : les rôles qui peuvent accéder à ce pattern.
- Lorsque vous essayer d'accéder à une ressource, Symfony cherche dans cette rubrique s'il y a un matching avec la route recherché de haut vers le bas. Le premier qu'il trouve lui permet de vérifier si vous avez le bon rôle pour accéder à la ressource demandé ou non. **L'ordre donc est très important.**

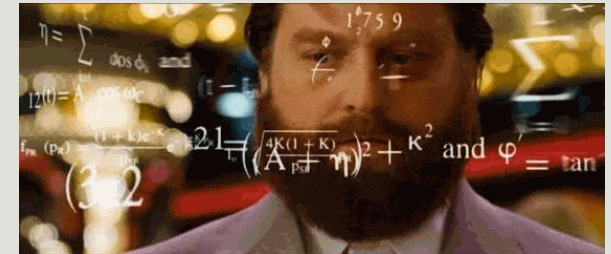
```
access_control:
  - { path: ^/admin, roles: ROLE_ADMIN }
  - { path: ^/profile, roles: ROLE_USER }
```

Exercice



- Créer une action avec la route `'/admin'`. Décommenter les **access_control** et essayer deux scénarios.
 1. Connecter vous en tant que USER et essayer d'accéder à cette route. Que se passe t-il ?
 2. Déconnecter vous et essayer d'y accéder. Que se passe t-il ?
- Modifier votre classe UserFixture et faite en sorte d'ajouter un user avec le ROLE_ADMIN. N'effacer rien de votre base.
- Connecter vous en tant qu'admin. Essayer d'accéder à la route `/admin`. Que se passe t-il ?

Exercice



- Créer une action permettant d'inscrire des utilisateurs.
- Créer une action pour l'admin lui permettant d'ajouter des utilisateurs avec le ROLE qu'il veut.

Définir la route à activer en cas d'erreur 401

- Par défaut lorsque un utilisateur non authentifié essaye d'accéder à une ressource protégé, un page d'erreur apparaît.
- Cependant ce n'est pas le comportement standard. Ce qu'on veut généralement c'est de le rediriger vers la page de login.
- Pour ce faire, vous devez implémenter la méthode **start**.
- A l'intérieur de cette méthode implémenter le comportement que vous voulez, **c'est cette méthode qui sera appelé en cas de 401.**

Sécuriser Les contrôleurs

Vous pouvez directement sécuriser vos contrôleurs en utilisant :

1- Le helper **denyAccessUnlessGranted**('ROLE_*');

2- En utilisant l'annotation **@IsGranted**('ROLE_*')

```
/**
 * Matches /blog exactly
 *
 * @IsGranted("ROLE_ADMIN")
 * @Route("/blog", name="blog_list")
 */
public function list()
{
    // ...
}
```

```
/**
 * Matches /blog exactly
 *
 * @Route("/blog", name="blog_list")
 */
public function list()
{
    $this->denyAccessUnlessGranted("ROLE_USER");
    // ...
}
```

Sécuriser un service

Afin de sécuriser un service, il suffit d'injecter le Security Service.

Utiliser ensuite sa méthode **isGranted**

```
use Symfony\Component\Security\Core\Security;
class HelperService
{
    private $security;
    public function __construct(Security $security)
    {
        $this->security = $security;
    }
    public function sendMoney() {
        if ($this->security->isGranted("ROLE_ADMIN")) {
            // Todo Send Money
        }
    }
}
```

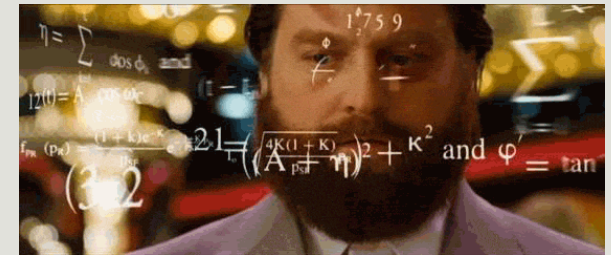
https://symfony.com/doc/current/security/securing_services.html

Sécuriser vos pages TWIG

Si vous voulez vérifier le rôle de l'utilisateur avant d'afficher une ressource ou des informations dans vos pages TWIG, utiliser la méthode `is_granted('ROLE_*)`

```
{% if is_granted('ROLE_ADMIN') %}  
    <a href=« /admin">Administration</a>  
{% endif %}
```


Exercice



Faite en sorte que le menu s'adapte au rôle de l'utilisateur connecté.

Hiérarchie de rôles

- Vous pouvez définir une hiérarchie de rôles.
- Dans le fichier **security.yaml** et sous la clé **role_hierarchy**, définissez le **rôle principale** suivie de **l'ensemble des rôles dont il hérite**.
- Un use case très récurant est quand l'admin possède tout les droits, donc l'admin devra hériter de tous les rôles.

```
role_hierarchy:  
  ROLE_COMMERCIAL:      ROLE_AGENT  
  ROLE_SECRETARY:       ROLE_COMMERCIAL  
  ROLE_ADMIN:           [ROLE_PARTNER, ROLE_SECRETARY]
```

- Ici un commercial a les accès de l'Agent.
- L'admin peut avoir les accès du Partner et de la secrétaire.

Spécial Rôles

- **PUBLIC_ACCESS** : Un utilisateur non authentifié
- **IS_AUTHENTICATED_REMEMBRED** : Vérifie qu'un utilisateur est authentifié indépendamment de son ROLE.
- **IS_AUTHENTICATED_FULLY** : Vérifie qu'un utilisateur est authentifié indépendamment de son ROLE. Cependant si le user est authentifié à cause de la fonctionnalité 'remember_me' alors il n'est pas authenticated fully.