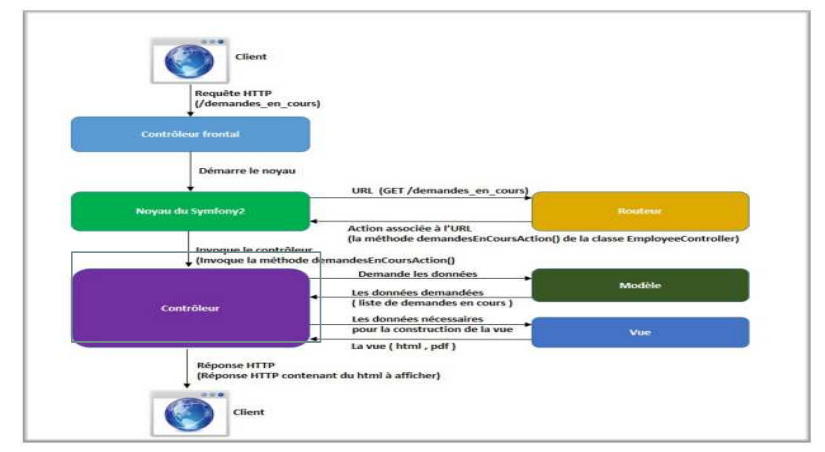
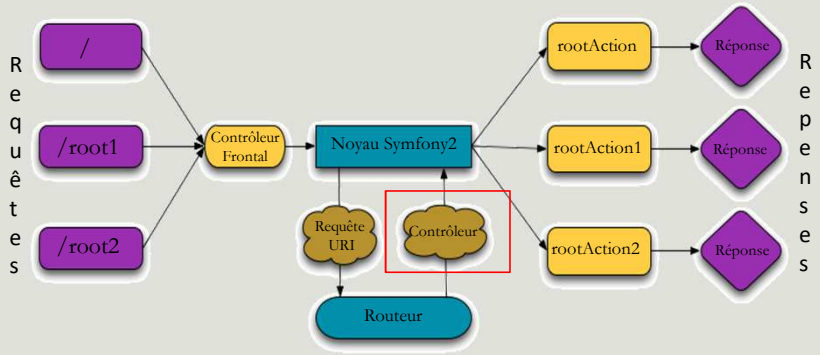
Symfony 4
Les contrôleurs

AYMEN SELLAOUTI

Introduction (1)



Introduction (2)



Introduction (3)

Fonction PHP (action)

Rôle : Répondre aux requêtes des clients.

Page HTML
Document XML
Tableau JSON
Image
Redirection
…

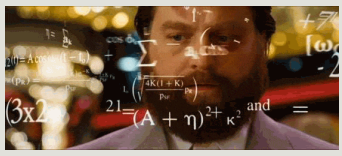
Exemple d'un contrôleur

```php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Response;
// Si on veut que notre contrôleur hérite de la classe afin de pouvoir utiliser ces méthodes helper
class PersonneController extends AbstractController
{
    /**
     * @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
- Faite 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
generateUrl(string $route, array() $parameters)	Génère une URL à partir de la route	String
forward(String Action, array () $parameters)	Forward la requête vers un autre contrôleur	Response
Redirect(string $url, int $statut)	Redirige vers une url	RedirectResponse
RedirectToRoute(string $route, array $parameters)	Redirige vers une route	Response
Render(string $view, array $parameters)	Affiche une vue	Response
Get(string $id)	Retourne un service à travers son id	object
createNotFoundException(String $messag)	Retourne une NotFoundException	NotFoundException

https://symfony.com/doc/4.2/controller.html

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
```

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 YAML et en annotation.

```
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 root dans le contrôleur nous utilisons les noms des paramètres.

Exemple

```
personne:
    path: /personne/{section}
    controller: App\Controller\PersonneController::index
```

```
public function index($section)
{
    $resp = new Response('<html><body>Bonjour'. $section.'!</body></html>');
    return $resp;
}
```

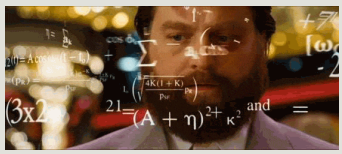
Exemple
Passage de paramétré : root vers contrôleur

```php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Response;
// Si on veut que notre contrôleur hérite de la classe afin de pouvoir utiliser ces méthodes helper
class PersonneController extends AbstractController
{
    /**
     * @Route("/personne/{section}", name="personne")
     */
    public function index($section)
    {
        // On crée un objet Response puis on la retourne c'est le rôle du contrôleur
        $resp = new Response(«<html><body>Bonjour $section !</body></html> »);
        return $resp;

    }
}
```


Exercice



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

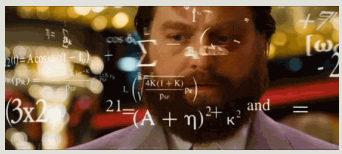
Récupérer les paramètres de la requête (1)

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

```
public function indexAction(Request $req)
{ … }
```


Exercice



- Dans la classe FirstController
- Créer une action second
- Faite en sorte d'y dumper 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 passées dans la requête

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

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')

Exemple

```
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 traitement se fait à travers le service templating qui se charge de créer et d'irriguer un objet de type Response.

Rôle : créer la réponse et la retourner

Méthode :

- Sans passer par les helpers de la classe AbstractController :
  - $this->get('templating')->render('l'url', 'les paramétres à transferer');
- En utilisant les helpers :
  - $this->render ('l'url', 'les paramétres à transferer');

Dans la version 3,4 l'url est présenté selon le format suivant :
@NomBundle/Dossier_dans_ressources/page
Le nom du bundle est sans le mot clé Bundle

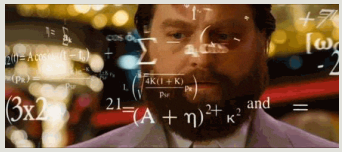
Dans la version 4 le concept de Bundle étant délaissé,
Toutes les vues sont dans le dossier Template
l'url représente l'arborescence à partir de ce dossier

Réponse aux requêtes
Renvoi (2)

Exemple :

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


Exercice



- Créer l'action (contrôlcur) **cv**
- Préparer des variables permettant de créer un mini Portfolio. Le contenu de ces variables devra vous permettre d'afficher vos nom et prénom, votre âge et votre Section.
- Ensuite, utiliser la méthode render afin d'invoquer votre page TWIG en lui passant les variables que vous venez de préparer.
- Sachant que pour afficher une variable dans TWIG il suffit de l'entourer de {{ nomVariable }}, créer une page TWIG « cv.html.twig » dans un dossier « Premier» que vous créerez dans le dossier « templates ».
- Cette page devra afficher les données transmises par votre contrôleur

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)

Méthode :

Redirection vers une **url**.

- $url=$this->get('router')->generate('notre_route');
- 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 fournit avec l'objet **Request.**
- La méthode **getSession()** permet de récupérer la session.
- Dans les pratiques il est préférable d'utiliser le type-hint via L'interface SessionInterface :
  - public function index (SessionInterface $session)
- 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('nomParamétre')
```


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.

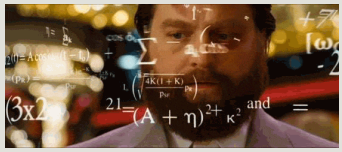
Gestion des sessions : les FlashMessages

- Pour récupérer le Flash message de la TWIG on utilise **app.session.flashbag.get**`('nomParamétre').`
- Vous pouvez aussi utiliser la méthode flashes de la variable globale app qui contient le tableau des flashBags messages.
- La documentation offre beaucoup d'exemple d'affichage des flashBagsMessage

Gestion des sessions : les FlashMessages

- Les variables de sessions qui ne dure 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 utilisez un helper via la méthode **addFlash**.

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.

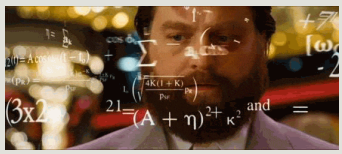
Exemple

```
$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



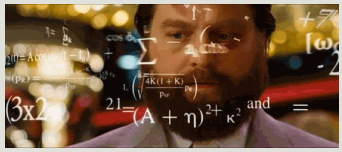
Quelques Astuces

Ajouter bootstrap (c'est encore tôt) pour avoir les classes Alert ou un peu de css pour colorer le background de vos DIV ou paragraphe.

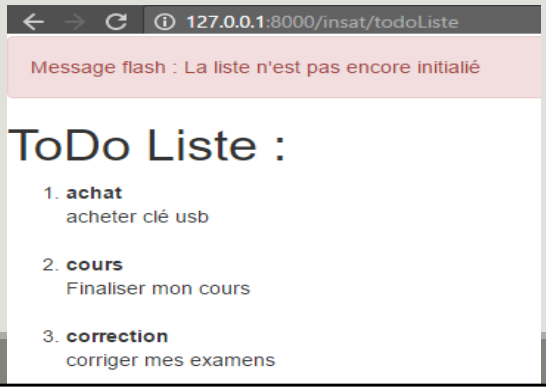
Utiliser la fonction **unset** de php qui permet de supprimer un élément du tableau partir de sa clé.

Pour ajouter un élément dans un tableau associatif il suffit d'utiliser la section suivante :
$monTab['identifiant']=$var ;

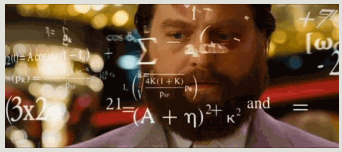
Exercice



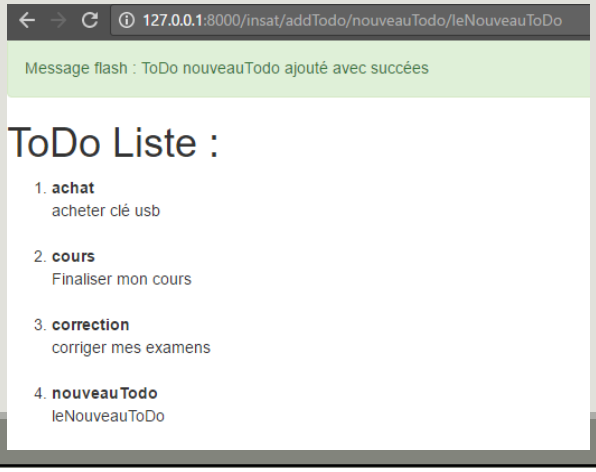
- Créer une action addToDoAction qui permet d'ajouter un ToDo ou de le mettre à jour. Cette action devra afficher la liste mise à jour. Si la liste de ToDo n'est pas encore initialisée, un message d'erreur sera affiché.



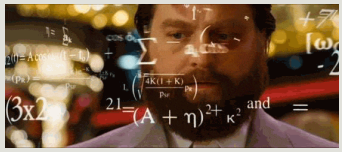
Exercice



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

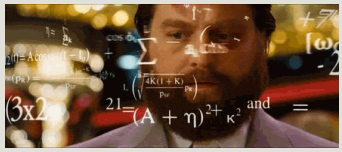


Exercice



- 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

```twig
{# 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

Bonne pratiques

- Symfony suit la philosophie de "thin controllers and fat models".
- Cela signifie que les contrôleurs doivent ne conservez que la fine couche de code nécessaire pour coordonner les différentes parties de l'application.
- Les méthodes de votre contrôleur doivent simplement appeler d'autres services, déclencher des événements si nécessaire, puis renvoyer une réponse, mais ils ne doivent contenir aucune logique métier réelle.
- Si c'est le cas, refactorisez votre code en dehors du contrôleur et dans un service.
- Faites en sorte que votre contrôleur étende la AbstractController fourni par Symfony et utiliser des annotations pour configurer le routage, la mise en cache et la sécurité autant que possible.
- N'ajouter pas le suffix Action aux noms de vos actions.

aymen.sellaouti@gmail.com