# Symfony 5 – Contrôleurs

# 1. Introduction

Un contrôleur est une classe PHP qui contient des fonctions dont le rôle est de traiter la requête http du client (objet Request HTTP) et renvoie une réponse HTML, XML, JSON ….(objet Response).

Lors de la création d'un projet Symfony, un répertoire «**Controller**» est créé dans src. Ce répertoire contient le ou les contrôleurs de notre projet.

# 2. Conventions de nommage et structure d'un contrôleur

## 2.1. Conventions de nommage

Le nom des fichiers des contrôleurs doit respecter la convention suivante, à savoir :

* commencer par le nom du contrôleur avec la première lettre en majuscules
* finir par le suffixe **Controller**.
* avoir l'extension « php ».

Les contrôleurs doivent être créés dans le répertoire « Controller ».

La classe créée doit être du même nom que le fichier du contrôleur (sans l'extension bien sûr !) pour que le contrôleur soit chargé automatiquement. Tous les contrôleurs doivent hériter de la classe **Symfony\Bundle\FrameworkBundle\Controller\AbstractController.** Les méthodes, à l’intérieur du contrôleur, peuvent avoir des paramètres issus des URL et des routes définies dans le fichier des routes.

Il est judicieux de regrouper les vues correspondantes à un contrôleur dans un répertoire qui porte le même nom que le contrôleur dans **templates (***Resources/views, versions antérieures***)**.

## 2.2. Structure d'un contrôleur

namespace App\Controller ;

// import de la classe Controller

**use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;**

**class DefaultController extends AbstractController**

{

public function index ()

{

return $this->**render**(…);

}

}

# 3. Fonctionnement du contrôleur

Le rôle du contrôleur est de retourner un objet Response à l'issue d'une requête, donc il faut instancier l'objet **Response** de Symfony et faire un return de cette instance.

## 3.1. L'objet Response

La réponse retournée par le contrôleur peut être :

* un texte à afficher : **response()**
* une vue : $this->**render()** retourne une vue en HTML. La méthode $this->**json**() renvoie un objet JsonResponse si le service de sérialisation est activé, sinon on utilise la fonction **json\_encode()**.
* Un fichier : $this->file(‘chemin du fichier’) ;
* une redirection vers une autre route → **redirect() ou redirectToRoute().** Par défaut, le code http de la redirection est 302. Il peut être modifié via le second paramètre de la méthode redirect. La redirection http nécessite le renvoi d’une réponse de code 30X.
* un forward vers un autre contrôleur → **forward()** qui n'est rien d'autre qu'un raccourci vers forward du service httpKernel. C’est une redirection interne qui consiste à exécuter une seconde action depuis le contrôleur courant. Ce type de redirection est invisible pour le client puisqu’on ne passe par le navigateur.
* Une réponse http 404 (gestion des erreurs) à travers une exception :

**throw $this->createNotFoundException(**'**Message**'**) ;**

* Une erreur http 500 : **throw new \Exception(**'**message**'**) ;**

**Exemples :**

**use Symfony\Component\HttpFoundation\Response ;**

public function index ()

{

return new Response('Bienvenue en BTS SIO') ;;

// ou à travers une Vue

return **$this->render**('[nomSousRepertoireTemplate]/nomVue.html.twig’) ;

// Ou à travers une redirection

return **$this->redirect**($this->generateURL('nom\_route')) ;

// Autre manière de rediriger

return **$this->redirectToRoute**('nom\_route') ;

// ou via un forward

return **$this->forward**('App\Controller\AutreControleur::nomMethode', [arguments]) ;

}

## 3.2. L'objet Request

Pour récupérer les requêtes Http, on utilise l'objet **Symfony\Component\HttpFoundation\Request**.

Cet objet sera utilisé avec les formulaires mais peut également servir à récupérer les paramètres de l'URL qui ne figurent pas dans les routes (avec?param=valeur ou /{parametre} dans l'url).

***Récupération des paramètres en dehors de routing.yaml***

Par exemple dans l’URL [http://localhost/sf?login=dandre](http://localhost/sf4?login=dandre) , l’objet Request va nous permettre de récupérer la valeur de la variable login.

public function accueil(**Request $request**) {

$login = **$request->query->get**('login');

return new Response($login);

}

L'objet Request contient des informations concernant les requêtes du client. Ces informations sont accessibles à travers des attributs.

Dans le fichier routing.yaml, on ne passe pas de paramètres à la route : path : /sf

|  |  |
| --- | --- |
| **Type de paramètres (PHP natif)** | **Symfony** |
| Variable d'URL (après?var=valeur) ou de formulaire ($\_GET['champ']) | $request->query->get('var') (équivalent de $\_GET['var'])  $request->query->get('champ') ; |
| Variable de formulaire ($\_ POST['champ']) | $request->request->get('champ') ; |
| Variable de cookie : $\_COOKIE['nom'] | $request->cookies->get('nom') ; |
| Variable de serveur : $\_SERVER['SERVER\_NAME'] | $request->server->get('SERVER\_NAME') ; |
| Variable d'en-tête : $\_SERVER['HTTP\_\*']  Ex : $\_SERVER['HTTP\_HOST']  $\_SERVER['HTTP\_USER\_AGENT'] | $request->headers->get('HTTP\_HOST') ;  $request->headers->get('User-Agent') ; |

La méthode **get('variable')** permet de récupérer la valeur de la variable. La méthode **set('variable','valeur')** permet d'initialiser la variable. Si le paramètre n'est pas défini dans l'URL, la méthode retourne une chaîne vide.

# **4. Les services**

Dans Symfony5, les contrôleurs sont déjà enregistrés en tant que services avec la configuration par défaut du fichier services.yaml.

Un contrôleur peut également avoir besoin d’autres services.

Un service est un objet PHP (donc une instance d'une classe) qui remplit une seule et unique fonction avec une certaine configuration. Par exemple, l’envoi d’un mail ou vérifier qu’un texte n’est pas un spam… L'intérêt d'un service est d'être utilisable dans tout le code de l'application. Tout objet qui doit être utilisé à plusieurs endroits de l'application devra être un service afin d'alléger les contrôleurs. Un service est accessible grâce au Service Container. Ce dernier est une classe dont chacune des méthodes retourne un service.

Un objet peut avoir besoin d’autres objets pour accomplir sa fonction. Se pose alors la question de l’instanciation de ces objets et dans quel ordre. Le conteneur de services permet d'organiser et d'instancier ces objets. Pour récupérer un service, il suffit de le demander au conteneur en l'appelant par son nom. Il est chargé de s’occuper de l’injection de dépendances, c’est-à-dire que la classe n’est plus chargée d’aller chercher elle-même les dépendances dont elle a besoin mais c’est le conteneur qui s’en occupe lorsque le service est appelé.

Les services favorisent la réutilisabilité du code et allègent les contrôleurs.

Un service est **persistant**, c'est à dire que la classe du service est instanciée une seule fois lors de la première demande de récupération du service. En cas d'un nouvel appel du même service, c'est la première instance qui est retournée et utilisée (fonctionnement comparable au patron de conception Singleton avec une meilleure flexibilité).

## 4.1. Créer un service

Pour réutiliser un code, il faut le mettre dans un service.

1. Créer un répertoire Service dans le répertoire src
2. Créer une classe PHP dans ce répertoire. Le namespace devient : App\Service
3. Définir la méthode à réutiliser dans la classe avec la visibilité public ainsi que le constructeur qui valorise les attributs et qui nous permettra de créer, par la suite, un objet qui accède à la méthode.

## 4.2. Récupérer un service

Pour appeler un service, on utilise la méthode : $service = **$this->get('nomService')**.

Pour avoir la liste des services disponibles, il faut taper la commande suivante :

Sf5 : php bin/console debug:autowiring [--all]

Nous avons déjà utilisé des services tels que Request et Templating.

* **Session : => use Symfony\Component\HttpFoundation\Session\SessionInterface**

SessionInterface $session ;

**Initialiser une variable de session** : $session->set('user',$user)**;**

**Lire une variable des session** : $session->get('user') ;

*Voir détails sur les sessions : symfony.com/doc/current/session.html*