Fonctionnalités de Symfony

Symfony apporte plein de fonctionnalités pratique pour nos contrôleurs, retournons dans notre fichier "**HomeController.php**".

paramètre de route

Une des fonctionnalités pratique des routeurs est de se passer des paramètres get pour obtenir des données directement dans l'url, évidement, Symfony le permet.

Créons une nouvelle route :

```
#[Route("/bonjour/{nom}/{prenom}", name: "bonjour")]
public function bonjour($nom, $prenom, Request $request): Response
{
    dump($request);
    // dd($request);
    return $this->render("home/bonjour.html.twig", [
        "nom" => $nom,
        "prenom"=> $prenom
    ]);
}
```

On ira aussi créer le template correspondant.

Les accolades au niveau de la route me permet de créer des arguments récupérable dans ma méthode par des arguments portant le même nom.

On verra plus tard que l'on peut pousser ces paramètres de route bien plus loins.

Fonctions et classes indispensable

Voyons maintenant ce troisième argument nommé request.

Il faut savoir que lorsque l'on indique une classe pour un argument, Symfony va automatiquement instancier la classe et la ranger dans cet argument. Ici nous avons appelé :

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

dump et dd

Si on utilise la fonction "**dump()**" de Symfony nous allons pouvoir voir ce que cet objet contient.
Mais si on charge notre page, rien ne s'affiche... où est donc notre dump? Il y a bien quelque chose qui a changé, enfin tant qu'on est en environnement de développement.

En effet, vous l'aviez peut être remarqué, mais en environnement de développement, symfony affiche une barre en bas de fenêtre. Celle ci contient de nombreuses informations utiles au développement, pour l'instant arrêtons nous à la cible qui est apparut à la droite de la première liste d'élément. En passant la souri dessus vous verrez votre dump apparaître, coloré et interactif.

Tentez de "**dump**" différentes choses, des strings, nombres, tableaux, vous verrez que ce dump est bien plus pratique que le classique "var_dump" de PHP.

Notons l'existence aussi de la fonction "**dd()**" qui signifie "dump and die" mais dans ce cas la barre ne s'affichera pas et le dump sera directement dans la page.

Twig aussi gère la fonction "**dump()**".

Request

Examinons le contenu de cette Class Request :

- attribute contient des informations lié à notre route dont ses paramètres.
- request contiendra toute les données de $_POST.
- query contiendra toute les données de $_GET.
- server contiendra toute les données de $_SERVER.
- file contiendra toute les données de $_FILES.
- cookies contiendra toute les données de $_COOKIES.
- session contiendra toute les données de $_SESSION.
- Et on trouveras tout un tas d'autres informations comme la méthode, l'URI, la langue et j'en passe.

Vous l'aurez compris Request regroupe toute les superglobals de PHP en plus d'informations spécifique à Symfony.

Spécificité des routes à paramètres

Ordre des routes

ATTENTION : Si maintenant je souhaite créer une route "**/bonjour/anglais/{username}**" Si je la place après notre route bonjour, elle deviendra inaccessible.

```
#[Route("/bonjour/anglais/{username}", name: "hello")]
public function hello($username): RedirectResponse
{
    dd("Hello ".$username);
    return $this->redirectToRoute("bonjour");
}
```

Comme l'indique son nom "**redirectToRoute**" permet de rediriger vers la route donnée en argument.

Symfony lis les routes dans l'ordre où elles arrivent, hors si je tenter d'aller sur "**bonjour/anglais/pierre**" et bien cela matchera bonjour avec "anglais" en nom et "pierre" en prénom.

Au contraire si je la place au dessus de bonjour. Elle deviendra accessible et bonjour ne sera inaccessible que si quelqu'un met "anglais" en nom.

Nous pouvons voir le détail des routes qui ont été vérifié par Symfony si on clique en bas à gauche de la barre de Symfony puis que nous allons sur le menu de gauche dans "**Routing**".

Générer une route à paramètre avec twig

Nous avons vu que nous pouvions utiliser la fonction "**path()**" pour générer une route via son nom, mais comment faire lorsque cette route attend des paramètres ?

Pour cela on peut ajouter des paramètres à cette même fonction :

```
<a href="{{ path("bonjour", {nom: "Jean", prenom: "Fontaine"}) }}">Jean
Fontaine</a>
```

préfixe des routes

Symfony offre la possibilité de préfixer nos routes, si je met l'attribut "Routes" non sur une méthode mais sur la classe elle même, alors toute les méthodes seront préfixé. par exemple :

```
#[Route('/user')]
class UserController
{
  #[Route('/profil', name:"profil")]
  public function profil(): response{}
}
```

Dans cet exemple, si je souhaite atteindre la page profil, la route sera "**/user/profil**" Et toute autre route dans cette classe commencera par "**/user**".

Valeur par défaut des routes

On a vu qu'on pouvait mettre des paramètres à nos routes, et bien comme on peut mettre des valeurs par défaut aux paramètres des fonctions, on peut en faire autant des routes.

```
#[Route("/bonjour/anglais/{username}", name: "hello", defaults:
["username"=>"Charles"])]
```

En ajoutant l'argument default, j'indique que si aucune valeur n'est fourni, alors "Charles" sera prit pour l'username.

Il existe aussi une syntaxe simplifié qui prendra la forme suivante :

```
#[Route("/bonjour/{nom}/{prenom?Jean}", name: "bonjour")]
```

Ici si on ne fourni pas le prénom, il sera égale à "Jean".

Attention, quand j'ai plusieurs paramètres, je ne peux donner une valeur par défaut que si tous les paramètres à sa droite ont des valeurs par défaut.
Ici je ne pourrais pas donner de valeur par défaut à nom si je n'en fourni pas à prenom.

Des regex et des routes

Actuellement je peux placer absolument n'importe quoi pour mes valeurs "username", "nom" et "prénom". Si je dis que je m'appelle "55 22" ça sera valide. Heureusement il est possible de bloquer cela grâce à nos amies les regex.

```
#[Route("/bonjour/anglais/{username}",
    name: "hello",
    defaults: ["username"=>"Charles"],
    requirements: ["username"=>"^[a-zA-Z]+$"])]
```

Maintenant si je tente de rentrer des chiffres pour ma route "hello" on remarquera que c'est la route "Bonjour" qui est chargé.
Pareillement une version raccourcie existe :

```
#[Route("/bonjour/{nom<^[a-zA-Z]+$>}/{prenom<^[a-zA-Z]+$>?Jean}", name:
"bonjour")]
```

Maintenant nos routes n'acceptes que des lettres.

La barre de développement

Nous avons déjà vu que cette barre nous permet de voir notre routing et nos dump mais ce n'est pas tout ce dont elle est capable, en partant de gauche :

- le statut de la requête et le nom de la route entre autre.
- le temps de chargement de la page.
- la mémoire utilisée.
- les erreurs, warning et dépréciations.
- les détails du cache.
- l'utilisateur connecté. (nous y reviendrons plus tard)
- les détails sur le rendu du twig.
- les dumps si il y en a.

et sur la droite de la barre :

- les détails sur le serveur.
- les détails sur Symfony.
- Un bouton permettant de replier la barre.

Sur la plupart de ces éléments vous pouvez cliquer pour avoir encore plus de détail.

La barre disparaîtra si vous passez en environnement de production. N'oubliez pas de le faire quand vous déploierez votre projet.

Gestion des sessions

Symfony a sa propre façon de gérer les sessions, ajoutons à notre page "Bonjour" un compteur de visite :

```
$sess = $request->getSession();

if($sess->has('nbVisite')) $nb = $sess->get("nbVisite")+1;
else $nb = 1;

$sess->set("nbVisite", $nb);
```

"**getSession()**" représente le "session_start()" et le reste des méthodes sont assez clair :

- has permet de vérifier l'existence d'un élément en session,
- get de récupérer une valeur en session.
- set de paramétrer une valeur en session.
- remove de supprimer une valeur en session.

du côté twig pour l'affiche on utilisera :

```
<h2>Déjà {{ app.session.get("nbVisite") }} visite(s) !</h2>
```

On verra par la suite que dans cette variable "app" se cache plus que les sessions.

les messages flash

Symfony à son propre système de message flash, très simple d'utilisation il permet d'en stocker plusieurs et de les afficher au moment voulu selon sa catégorie.

Je vais ajouter à mon controller "**bonjour**" :

```
$this->addFlash("bonjour", "Bonjour $prenom $nom !");
```

Et à mon controller "**hello**" :

```
$this->addFlash("redirect", "Vous avez été redirigé!");
$this->addFlash("Hello", "Hello $username");
```

Maintenant je peux me rendre sur mon template et ajouter :

```
{% for message in app.flashes('bonjour') %}
    <div>{{ message }}</div>
{% endfor %}
```

"**app.flashes()**" me permet de récupérer tous les messages d'une catégorie donnée.
Mais je peux vouloir récupérer plusieurs catégories de message, dans ce cas :

```
{% for categorie, messages in app.flashes(['Hello', "redirect"]) %}
  {% for message in messages %}
    <div>
      <strong>{{ categorie }} :</strong>
      <br>
      {{ message }}
    </div>
  {% endfor %}
{% endfor %}
```

Ici je me permet de récupérer tous les messages des catégories données en paramètre. il me faudra alors une second boucle pour les afficher un à un.

J'ai aussi la possibilités de ne pas donner de paramètre pour récupérer absolument tous les messages flash.

```
{% for categorie, messages in app.flashes %}
```

Maintenant à vous de jouer, *faites l'exercice 1 !*

Une fois terminé, simplifions nous la vie pour la suite et ajoutons dans notre base.html.twig l'affichage des messages flash et bootstrap.

```
<div class="container">
  {% for label, messages in app.flashes %}
    {% for message in messages %}
      <div class="alert alert-{{ label }}">
        {{ message }}
      </div>
    {% endfor %}
  {% endfor %}
  {% block body %}{% endblock %}
</div>
```