Les Controlleurs

Lotfi KHEDIRI

Un contrôleur simple

- Un contrôleur est généralement **une méthode** à l'intérieur d'une classe de contrôleur

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

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

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

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

Un contrôleur simple

- Le mot-clé **use** importe la classe Response, que le contrôleur doit renvoyer.
- La classe peut techniquement être appelée n'importe quoi, mais elle est suffixée avec '**Controller**' par convention.
- La méthode d'action est autorisée à avoir un ou +ieurs paramètres : $max grâce à {max}
- Le contrôleur **crée** et **renvoie** un objet '**Response**'.
- Pour *afficher* le résultat de ce contrôleur, on doit lui mapper une URL via une route : @Route("/lucky/number/{max}")
- Pour voir la page, accédez à cette URL dans le navigateur: *http: // localhost: 8000 /lucky/number/100*

Les services du contrôleur de base

- Pour faciliter le développement, Symfony est livré avec une classe de contrôleur de base **facultative** appelée AbstractController.
- Cette classe **peut** être étendue pour accéder à ses méthodes.
- Importer cette classe de contrôleur, puis modifiez votre classe pour qu'elle hérite de AbstractController

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

 class LuckyController extends AbstractController
{
    // ...
}
```

- Vous avez maintenant accès à des méthodes comme $ this-> render () et bien d'autres comme

```
$url = $this->generateUrl('app_lucky_number', ['max' => 10]);
```

Redirection

- Si vous souhaitez **rediriger** l'utilisateur vers une autre page, utilisez les méthodes **redirectToRoute()** et **redirect()**:

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

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

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

Rendering Templates

- Si vous renvoyer du HTML, vous souhaiterez rendre un **template** (La vue).
- La méthode **render()** rend un modèle **et** place ce contenu dans un objet de type **Response** pour vous.

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

- Les templtes et **Twig** sont expliqués plus en détail dans le chapitre Création et utilisation de templates
- Lien : https://symfony.com/doc/current/templates.html

Récupération des services

- Symfony fournit beaucoup d'objets utiles, appelés *services* .
- Ceux-ci sont utilisés pour le rendu des modèles, l'envoi d'e-mails, l'interrogation de la base de données et tout autre "travail" auquel on peut penser
- Si vous avez besoin d'un service dans un contrôleur **typez (type-hinting)** un argument avec son nom de classe (ou interface).
- Symfony vous transmettra automatiquement le service dont vous avez besoin

Récupération des services : Exemple

```
use Psr\Log\LoggerInterface;
// ...
/**
* @Route("/lucky/number/{max}")
*/
public function number($max, LoggerInterface $logger)
{
    $logger->info('We are logging!');
    // ...
}
```

Quels autres services peut-on utiliser

- Pour les voir, utilisez la commande console **bin/console debug:autowiring**

```
Autowirable Types
=================

 The following classes & interfaces can be used as type-hints when autow

 Interface for annotation readers.
 Doctrine\Common\Annotations\Reader (annotations.cached_reader)

 Contract covering object managers for a Doctrine persistence layer Manag
 Doctrine\Persistence\ManagerRegistry (doctrine)

 A wrapper around a Doctrine\DBAL\Driver\Connection that adds features l
n levels, configuration, emulated transaction nesting, lazy connecting a
 Doctrine\DBAL\Connection (doctrine.dbal.default_connection)

 Connection interface. Driver connections must implement this interface.
```

Générer des contrôleurs

- Pour gagner du temps, installer Symfony Maker et demander à Symfony de générer une nouvelle classe de contrôleur

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

```
$ php bin/console make:controller BrandNewController

created: src/Controller/BrandNewController.php
created: templates/brandnew/index.html.twig
```

Gestion des erreurs et 404 pages

- Lorsque rien n'est trouvé, on doit renvoyer une réponse 404. Pour ce faire, lancez un type d'exception spécial

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

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

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

- une page d'erreur est montrée à l'utilisateur final (mode prod) et une page d'erreur de débogage complète est montrée au développeur(mode dev)
- Pour personnaliser la page d'erreur affichée à l'utilisateur, consultez l'article *Comment personnaliser les pages d'erreur*

L'objet **Request** en tant qu'argument de contrôleur

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

public function index( Request $request , $firstName, $lastName)
{
    $page = $request->query->get('page', 1);
    // ...
}
```

- Les informations de la demande Http sont stockées dans l'objet **Request** de Symfony donc on peut
  - Lire les paramètres de la requête,
  - récupérer un en-tête de demande ou
  - accéder à un fichier téléchargé

Gérer la session

- Symfony fournit un **service de session**, utiliser pour stocker des informations sur l'utilisateur entre les demandes (les requests).

- La session est activée par défaut, mais ne sera démarrée que si elle est utilisée (lire ou écrire dans la session)

- Pour obtenir la session, ajoutez un argument de type SessionInterface dans les paramètres de la fonction contrôleur de votre page

Gérer la session : Exemple

```php
use Symfony\Component\HttpFoundation\Session\SessionInterface;

public function index(SessionInterface $session)
{
    // stores an attribute for reuse during a later user request
    $session->set('foo', 'bar');

    // gets the attribute set by another controller in another request
    $foobar = $session->get('foobar');

    // uses a default value if the attribute doesn't exist
    $filters = $session->get('filters', []);
}
```

Messages Flash

- On peut stocker des messages spéciaux, appelés **messages "flash"** dans la session de l'utilisateur.
- Les messages flash sont destinés à être utilisés une **seule fois**: ils disparaissent automatiquement de la session dès que vous les récupérez.
- Cette fonctionnalité rend les messages "flash" particulièrement utiles pour stocker les notifications des utilisateurs.

Messages Flash : Exemple (dans le controleur)

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

public function update(Request $request)
{       // ...
    if ($form->isSubmitted() && $form->isValid()) {
        // do some sort of processing

        $this->addFlash('notice','Your changes were saved!');
        // $this->addFlash() is equivalent to $request
           // ->getSession()->getFlashBag()->add()

        return $this->redirectToRoute(...);
    }

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

- La clé du message (notice dans cet exemple) peut être n'importe quoi: cette clé sera utilisée pour récupérer le message.

Messages Flash : Exemple (dans la vue)

- Dans le template de la page suivante (ou mieux encore, dans le template de mise en page de base), lire tous les messages flash de la session en utilisant la méthode **flashes()** fournie par la variable d'application globale Twig app

```
{# 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 %}
```

L'objet Response

- Dans Symfony, un contrôleur est requis pour renvoyer un objet **Response**

```
use Symfony\Component\HttpFoundation\Response;

// creates a simple Response with a 200 status code (the
   default)
$response = new Response('Hello '.$name, Response::HTTP_OK);

// creates a CSS-response with a 200 status code
$response = new Response('<style> ... </style>');
$response->headers->set('Content-Type', 'text/css');
```

Accéder aux valeurs de configuration

- Pour obtenir la valeur d'un paramètre de configuration à partir d'un contrôleur, utilisez la méthode **getParameter()**

```
public function index()
{
    $projectDir = $this->getParameter('kernel.project_dir')
    // ...
}
```

Retour d'une réponse JSON

- utilisez la méthode **json()** d'assistance qui renvoie un objet de type **JsonResponse** qui code automatiquement les données

```
// ...
public function index()
{
    // returns '{"username":"jane.doe"}' and sets the proper
     //Content-Type header
    return $this->json(['username' => 'jane.doe']);


    // the shortcut defines three optional arguments
    // return $this->json($data, $status = 200, $headers = [],
  $context = []);
}
```