Fork Php

Building an app with Symfony 3

Part2 Controller - Routing

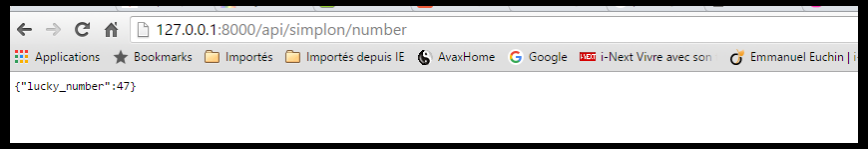
JSON Response

L'objet Response que vous renvoyé dans le contrôleur peut contenir du HTML, JSON ou aussi un fichier binaire comme une image ou un PDF.

Nous allons créer un retour JSON qui renvoi un nombre aléatoire. Ajoutons une seconde méthode à notre contrôleur src/AppBundle/Controller/SimplonController.php

```
// src/AppBundle/Controller/SimplonController.php

// ...
class SimplonController
{
    // ...

    /**
     * @Route("/api/simplon/number")
     */
    public function apiNumberAction()
    {
        $data = array(
            'lucky_number' => rand(0, 100),
        );

        return new Response(
            json_encode($data),
            200,
            array('Content-Type' => 'application/json')
        );
    }
}
```

php bin/console server:run

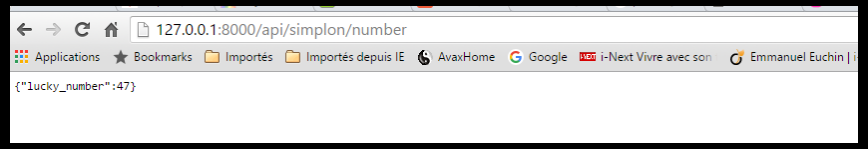


JSON Response

Nous pouvons utiliser JsonResponse:

```
//src/AppBundle/Controller/SimplonController.php

// ...
// --> N'oubliez pas d'ajouter la classe
use Symfony\Component\HttpFoundation\JsonResponse;

class SimplonController
{
   // ...

   /**
    * @Route("/api/simplon/number")
    */
   public function apiNumberAction()
   {
      $data = array(
         'lucky_number' => rand(0, 100),
      );

      // Appelle json_encode et défini l'entête Content-Type
      return new JsonResponse($data);
   }
}
```

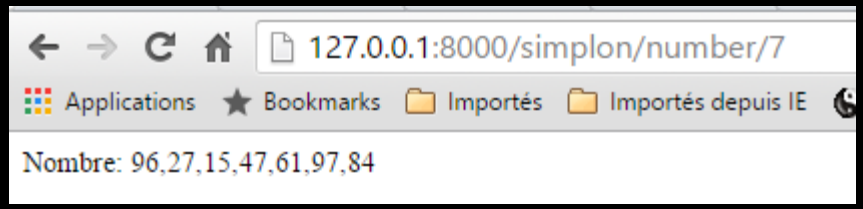
php bin/console server:run



Modèle d'URL dynamique /simplon/number/{count}

Nous voulons générer 5 nombres aléatoire en allant à /simplon/number/5

```
//src/AppBundle/Controller/SimplonController.php

// ...

class SimplonController
{
   // ...

   /**
    * @Route("simplon/number/{count}")
    */
   public function numberAction($count)
   {
                $numbers = array();
                for($i=0;$i<$count;$i++){
                                $numbers[] = rand(0,100);
                }
                $numbersList = implode(',',$numbers);
                return new Response('<html><body>Nombre: '.$numbersList.'</body></html>');
   }
}
```



Arborescence SYMFONY

app/	Contient la configuration et les templates. Tout sauf du PhP.
src/	Votre code PhP va ici. Il n'y a qu'un répertoire src/AppBundle que l'on utilise. Un Bundle est comme un plugin (téléchargeable).
web/	C'est le document root pour le projet qui contient les fichiers publiquement accessible comme le css, les images et le front contrôleur Symfony qui exécute l'app.
tests/	Le test automatique de votre application est ici.
bin/	Les fichiers "binaires" vont ici. La plus importante est la console qui permet d'exécuter des commandes Symfony.
var/	C'est ici que sont créés automatiquement les fichiers comme le cache et les logs.
vendor/	Contient les librairies, packages et bundles téléchargé par composer. Ne pas toucher.

Controller

Le contrôleur est un PhP appelable que vous créez qui prend les informations de la requête http et crée et renvoi une réponse HTTP qui peut être du HTML, XML, un tableau Json sérialisé, une image, une redirection, un 404 ou tout ce que vous souhaitez.

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

public function helloAction()
   {
            return new Response('Hello World!');
}
```

Quelques exemples:

- Controller A prépare un objet Response représentant le contenu de la page d'accueil.
- Controller B lit le paramètre 'jeton' d'une requête pour charger une entrée du blog dans la bdd et crée un objet Response affichant ce blog. Si le jeton n'est pas trouvé, on renvoi un code status 404.
- Controller C traite la soumission d'un formulaire de contact. Il lit les informations du formulaire depuis la requête, enregistre le contact dans la bdd et envoi un mail au contact pour l'informer. Enfin, il crée un objet Response qui redirige le navigateur du client vers la page 'merci'.

Requête, Contrôleur et Réponse

Chaque requête traitée par un projet symfony passe parle même cycle de de fonctionnement. Le framework s'occupe du travail répétitif, vous n'avez qu'à écrire votre propre code dans une méthode du contrôleur:

1. Chaque requête est traité par un simple Front Controller (app.php ou app_dev.php) qui "bootstrap" l'application.
2. Le routeur lit les informations de la requête (l'URI), trouve une route qui correspond aux infos et lit le paramètre _controller de la route.
3. Le contrôleur de la route est exécuté avec le code à l'intérieur créé et retourne un objet Response.
4. L'entête http et le contenu de l'objet Response est envoyé au client.

Créer une page est facile en créant un contrôleur (#3) et en créant une route qui dirige vers ce contrôleur (#2).

Un simple contrôleur

Un contrôleur est habituellement une méthode d'une classe controller.

Les contrôleurs sont aussi appelés actions.

```
// src/AppBundle/Controller/HelloController.php
namespace AppBundle\Controller;

use Symfony\Component\HttpFoundation\Response;

class HelloController
{
    public function indexAction($name)
    {
        return new Response('<html><body>Hello '.$name.'!</body></html>');
    }
}
```

Le contrôleur est la méthode indexAction, à l'intérieur de la classe controller (HelloController).

Une classe contrôleur contient en général plusieurs contrôleurs/actions (ex: updateAction, deleteAction).

Use permet d'utiliser le namespace définit au préalable pour importer la classe Response retournée par le contrôleur.

Mappage d'URL dans un contrôleur.

Paramètres de route comme arguments du contrôleur

Nous pouvons passer des données par URL pour le contrôleur

```
// src/AppBundle/Controller/HelloController.php
// ...
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

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

Pour en passer plusieurs

```
// src/AppBundle/Controller/HelloController.php
// ...

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class HelloController
{
    /**
     * @Route("/hello/{firstName}/{lastName}", name="hello")
     */
    public function indexAction($firstName, $lastName)
    {
        // ...
    }
}
```

L'ordre des arguments du contrôleur n'ont pas d'importance.
Chaque argument du contrôleur requis doit correspondre avec un argument du routeur. Pour éviter ce risque, nous pouvons rendre l'argument optionnel en lui affectant une valeur par défaut:
public function indexAction($firstName, $lastName, $foo = 'bar')

La requête comme argument du contrôleur

Quand vous aurez besoin de lire les paramètres de requête, lire une entête ou accéder à un fichier uploadé.
Toutes ces informations sont stockés dans l'objet Request de Symfony:

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

public function indexAction(Request $request, $page)
{
    $title = $request->attributes->get('title');

    // ...
}
```

La classe de base Controller

Symfony propose une classe optionnelle de base Controller à hériter. Elle vous donne accès à un nombre de méthode d'aide et à tous les services via un container.
Ajouter avec use la classe Controller et héritons (extends) là sur notre contrôleur HelloController

```
// src/AppBundle/Controller/HelloController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class HelloController extends Controller
{
    // ...
}
```

Redirection:

Pour rediriger l'utilisateur sur une autre page il faut utiliser la méthode redirectToRoute():

```
public function indexAction()
{
    return $this->redirectToRoute('homepage');

    // redirectToRoute is equivalent to using redirect() and generateUrl() together:
    // return $this->redirect($this->generateUrl('homepage'));
}
```

Pour un lien extérieur:

```
public function indexAction()
{
    return $this->redirectToRoute('homepage', array(), 301);
}
```

Gestion des sessions

Symfony fournit un objet Session qui permet de stocker des informations sur l'utilisateur par un cookie.

- Pour récupérer la session, nous appellerons la méthode getSession() de l'objet Request. Elle renvoie une SessionInterface composée de méthodes faciles pour stocker et lire les infos de la session.

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

public function indexAction(Request $request)
{
    $session = $request->getSession();

    // store an attribute for reuse during a later user request
    $session->set('foo', 'bar');

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

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

L'objet Response

Elle a pour rôle de renvoyer au client l'entête et le contenu d'un message text.

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

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

// create a JSON-response with a 200 status code
$response = new Response(json_encode(array('name' => $name)));
$response->headers->set('Content-Type', 'application/json');
```

La propriété de headers est un objet HeaderBag qui posséde des méthodes pour lire ou écrire les entêtes. Les noms d'entête sont normalisés en utilisant Content-Type.

- Pour Json, il y a JsonResponse
- Pour les fichiers, il y a BinaryFileResponse
- Pour les reponses streamé, il y a StreamedResponse

Nous y reviendrons plus tard.

L'objet Request

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

public function indexAction(Request $request)
{
    $request->isXmlHttpRequest(); // is it an Ajax request?

    $request->getPreferredLanguage(array('en', 'fr'));

    // retrieve GET and POST variables respectively
    $request->query->get('page');
    $request->request->get('page');

    // retrieve SERVER variables
    $request->server->get('HTTP_HOST');

    // retrieves an instance of UploadedFile identified by foo
    $request->files->get('foo');

    // retrieve a COOKIE value
    $request->cookies->get('PHPSESSID');

    // retrieve an HTTP request header, with normalized, lowercase keys
    $request->headers->get('host');
    $request->headers->get('content_type');
}
```

Comme pour l'objet Response, les entêtes de requête sont stockés dans un objet HeaderBag.

Routing

Nous préférons avoir de beaux URLs du type /read/intro-to-symfony à la place de index.php?article_id=57.

Nous allons voir comment créer:

- Des routes complexes pour diriger vers les contrôleurs
- Générer des URLs dans des templates et des contrôleurs.
- Charger des ressources routes depuis des Bundles (ou autres)
- Débugger vos routes.

Création d'une route

Symfony charge toutes les routes de votre application dans un simple fichier de routage. Ce fichier est habituellement app/config/routing.yml, mais peut être configuré pour être un autre (xml ou php) par le fichier de configuration de l'application.

```
# app/config/config.yml
framework:
    # ...
    router: { resource: '%kernel.root_dir%/config/routing.yml' }
```

Route basique:

```
// src/AppBundle/Controller/MainController.php


// ...
class MainController extends Controller
{
    /**
     * @Route("/")
     */
    public function homepageAction()
    {
        // ...
    }
}
```

Cette route correspond à la page d'accueil (/) et dirige vers le contrôleur AppBundle:Main:homepage.

Création d'une route

Symfony charge toutes les routes de votre application dans un simple fichier de routage. Ce fichier est habituellement app/config/routing.yml, mais peut être configuré pour être un autre (xml ou php) par le fichier de configuration de l'application.

```
# app/config/config.yml
framework:
    # ...
    router: { resource: '%kernel.root_dir%/config/routing.yml' }
```

Route basique:

```
// src/AppBundle/Controller/MainController.php
// ...
class MainController extends Controller
{
    /**
     * @Route("/")
     */
    public function homepageAction()
    {
        // ...
    }
}
```

Cette route correspond à la page d'accueil (/) et dirige vers le contrôleur AppBundle:Main:homepage.

Route avec paramètres:

```
// src/AppBundle/Controller/BlogController.php

// ...
class BlogController extends Controller
{
    /**
     * @Route("/blog/{slug}")
     */
    public function showAction($slug)
    {
        // ...
    }
}
```

Les paramètres sont obligatoires.

Paramètres requis et optionnels:

Imaginons que nous voulons gérer la pagination avec une adresse /blog/2 pour afficher la page 2

```
// src/AppBundle/Controller/BlogController.php
// ...
/**
 * @Route("/blog/{page}")
 */
public function indexAction($page)
{
    // ...
}
```

Pour afficher la première page, nous devrons avoir l'adresse /blog/1 ce qui n'est pas pratique car nous aurons préféré arriver cette page lorsqu'aucun paramètre n'est envoyée /blog

```
// src/AppBundle/Controller/BlogController.php
// ...
/**
 * @Route("/blog/{page}", defaults={"page" = 1})
 */
public function indexAction($page)
{
    // ...
}
```

URL	Route	Paramètres
/blog	blog	{page}=1
/blog/1	blog	{page}=1
/blog/2	blog	{page}=2

Donc $page aura lorsque ce n'est pas envoyé par l'URL la valeur 1 .

Ajout de paramètres:

```
// src/AppBundle/Controller/BlogController.php
// ...
class BlogController extends Controller
{
    /**
     * @Route("/blog/{page}", defaults={"page" = 1})
     */
    public function indexAction($page)
    {
        // ...
    }

    /**
     * @Route("/blog/{slug}")
     */
    public function showAction($slug)
    {
        // ...
    }
}
```

URL	Route	Paramètres
/blog/2	blog	{page}=2
/blog/my_blog_post	blog	{page}="my_blog_post"

Les deux routes pointent vers le même modèle d'URL. Le routeur de symfony prend la première route trouvée donc ici celle avec $page. Ce qui fait que pour un texte envoyé en paramètre, il sera pris comme une page donc problème.

Pour palier ce problème, nous pouvons utiliser les expressions régulières comme dans l'exemple suivant:

```
// src/AppBundle/Controller/BlogController.php

// ...

/**
 * @Route("/blog/{page}", defaults={"page": 1}, requirements={
 *     "page": "\d+"
 * })
 */
public function indexAction($page)
{
    // ...
}
```

URL	Route	Paramètres
/blog/2	blog	{page}=2
/blog/my_blog_post	blog_show	{slug}="my_blog_post"
/blog/2-my-blog-post	blog_show	{slug}="2-my-blog-post"

Le \d+ est une expression régulière qui demande à ce que le paramètre soit digital (numérique)

Pouvons mixer ces méthodes pour obtenir:

```
// src/AppBundle/Controller/MainController.php

// ...
class MainController extends Controller
{
    /**
     * @Route("/{_locale}", defaults={"_locale": "en"}, requirements={
     *     "_locale": "en|fr"
     * })
     */
    public function homepageAction($_locale)
    {
    }
}
```

URL	Paramètres
/	{_locale}="en"
/en	{_locale}="en"
/fr	{_locale}="fr"
/es	ne marche pas

Ajout de méthodes HTTP:

Nous pouvons aussi regarder la méthode des requêtes entrantes (GET, HEAD, POST, PUT,DELETE).
Exemple, vous voulez créer un API pour votre blog qui possède 2 routes. Une pour afficher un post (GET ou HEAD) et une autre pour updater un post (PUT)

```
// src/AppBundle/Controller/MainController.php
namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
// ...

class BlogApiController extends Controller
{
    /**
     * @Route("/api/posts/{id}")
     * @Method({"GET","HEAD"})
     */
    public function showAction($id)
    {
        // ... return a JSON response with the post
    }

    /**
     * @Route("/api/posts/{id}")
     * @Method("PUT")
     */
    public function editAction($id)
    {
        // ... edit a post
    }
}
```

Exemple de route complexe:

```
// src/AppBundle/Controller/ArticleController.php

// ...
class ArticleController extends Controller
{
    /**
     * @Route(
     *     "/articles/{_locale}/{year}/{title}.{_format}",
     *     defaults={"_format": "html"},
     *     requirements={
     *         "_locale": "en|fr",
     *         "_format": "html|rss",
     *         "year": "\d+"
     *     }
     * )
     */
    public function showAction($_locale, $year, $title)
    {
    }
}
```

- /articles/en/2010/my-post
- /articles/fr/2010/my-post.rss
- /articles/en/2013/my-latest-post.html